Adaptive Delivery of Real-Time Streaming Video

by

Nicholas G. Feamster

S.B., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2000)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

© Massachusetts Institute of Technology 2001. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
May 23, 2001

Certified by ....
Hari Balakrishnan
Assistant Professor
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

JUL 1 1 2001

LIBRARIES

BARKER

Adaptive Delivery of Real-Time Streaming Video

by

Nicholas G. Feamster

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

While there is an increasing demand for streaming video applications on the Internet, various network characteristics make the deployment of these applications more challenging than traditional Internet applications like email and the Web. The applications that transmit data over the Internet must cope with the time-varying bandwidth and delay characteristics of the Internet and must be resilient to packet loss. This thesis examines these challenges and presents a system design and implementation that ameliorates some of the important problems with video streaming over the Internet.

Video sequences are typically compressed in a format such as MPEG-4 to achieve bandwidth efficiency. Video compression exploits redundancy between frames to achieve higher compression. However, packet loss can be detrimental to compressed video with interdependent frames because errors potentially propagate across many frames. While the need for low latency prevents the retransmission of all lost data, we leverage the characteristics of MPEG-4 to *selectively* retransmit only the most important data in order to limit the propagation of errors. We quantify the effects of packet loss on the quality of MPEG-4 video, develop an analytical model to explain these effects, and present an RTP-compatible protocol—which we call *SR-RTP*—to *adaptively* deliver higher quality video in the face of packet loss.

The Internet's variable bandwidth and delay make it difficult to achieve high utilization, TCP-friendliness, and a high-quality constant playout rate; a video streaming system should adapt to these changing conditions and tailor the quality of the transmitted bitstream to available bandwidth. Traditional congestion avoidance schemes such as TCP's additive-increase/multiplicativedecrease (AIMD) cause large oscillations in transmission rates that degrade the perceptual quality of the video stream. To combat bandwidth variation, we design a scheme for performing quality adaptation of layered video for a general family of congestion control algorithms called *binomial congestion control* and show that a combination of smooth congestion control and clever receiver-buffered quality adaptation can reduce oscillations, increase interactivity, and deliver higher quality video for a given amount of buffering.

We have integrated this selective reliability and quality adaptation into a publicly available software library. Using this system as a testbed, we show that the use of selective reliability can greatly increase the quality of received video, and that the use of binomial congestion control and receiver quality adaptation allow for increased user interactivity and better video quality.

Thesis Supervisor: Hari Balakrishnan
Title: Assistant Professor

At the moment it's just a Notion, but with a bit of backing I think I could turn it into Concept, and then an Idea.

- Woody Allen

I not only use all the brains that I have, but all that I can borrow.

- Woodrow Wilson

Acknowledgments

I want to express my most heartfelt gratitude and appreciation to my advisor, Professor Hari Balakrishnan, who has been a constant source of inspiration for me, professionally and personally. Hari is the best advisor I could have ever hoped for. His technical savvy, excellent taste for important research problems, and ability to recognize that being successful entails far more than just being "smart" have helped me become a much more well-rounded person. Besides guiding my research, Hari has allowed me to recognize and address some of my weaknesses, and develop into a more effective researcher.

I am very grateful also to my colleague, Deepak Bansal, who helped me get acquainted with the research group at LCS and was an excellent person to work with. Deepak was also always patient with any questions I would ask him, and it has truly been a pleasure working with him. Deepak also wrote parts of Chapter 4, which appeared earlier in a workshop paper we co-authored.

The lab would not be the same place without Greg Harfst, who has been a constant source of support and advice (not to mention help with communication skills). He makes the lab a very fun and exciting place to be.

Thanks also to Dave Andersen and Godfrey Tan who helped read sections of this thesis in its conference paper form. Dave was great for answering some of my sillier programming questions or helping me find the answer. Dorothy Curtis helped me get acquainted with the Congestion Manager software and helped me with the all too common idiosyncrasies of equipment and software. She also has reminded me about the merits of `gzip` and `df -k` when dealing with large sequences of images!

My officemates, Sidney Chang and Peter Yang, have been wonderful, friendly people. I was fortunate to be able to share the office with such friendly and caring people.

I thank Susie Wee and John Apostolopoulos for giving me the spark to work in the area of video, and providing me excellent advice throughout my undergraduate and Master's career. While I worked at HP Labs, they opened me up to the exciting world of video processing. As a boss, Susie took great risks to allow me to get a jump start on research, and is a major reason for my being where I am today.

I thank DARPA and NTT for funding this research.

Finally, I would like to dedicate this thesis to my parents, who have made many sacrifices to allow me to be where I am today, and without whom my education would not be possible. My parents have taught me to value integrity, determination, and a concern for other people that I have found tremendously valuable, and I am very grateful for everything that they have empowered me to do.

Contents

1 Introduction **13**

1.1 Motivation . . . 13

1.2 Packet Loss . . . 14

1.2.1 The Problem . . . 15

1.2.2 The Solution . . . 16

1.3 Bandwidth Variation . . . 16

1.3.1 The Problem . . . 17

1.3.2 The Solution . . . 17

1.4 Delay Variation . . . 18

1.5 System Architecture . . . 19

1.5.1 Overview . . . 19

1.5.2 Loss-resilience . . . 20

1.5.3 Bandwidth Adaptation . . . 20

1.6 Contributions . . . 21

2 Related Work **23**

2.1 MPEG-4 Background . . . 23

2.2 Media Transport . . . 25

2.3 Error Resilience . . . 25

2.3.1 Coding Techniques . . . 26

2.3.2 Channel Techniques . . . 26

2.3.3 Retransmission-based Techniques . . . 27

2.3.4 Error Concealment Postprocessing . . . 27

2.4 Congestion Control . . . 28

2.4.1 Congestion Manager . . . 29

2.5 Quality Adaptation . . . 29

3 Selective Reliability **31**

3.1 Problem: Error Propagation . . . 33

3.2 Packet Loss Model . . . 35

  3.2.1 Experimental Results . . . 36

  3.2.2 Analytic Model . . . 36

3.3 Selective Reliability is Beneficial . . . 38

3.4 SR-RTP Protocol . . . 40

  3.4.1 Header Formats . . . 40

  3.4.2 Loss Detection and Recovery Decisions . . . 41

  3.4.3 Results . . . 42

3.5 Receiver Postprocessing . . . 44

  3.5.1 Recovery Techniques . . . 45

  3.5.2 Results . . . 48

  3.5.3 Summary . . . 50

3.6 Summary . . . 51

4 Bandwidth Adaptation **53**

4.1 Congestion Control . . . 54

4.2 Binomial congestion controls . . . 54

4.3 Quality Adaptation . . . 56

  4.3.1 Simulcast . . . 58

  4.3.2 Hierarchical Encoding . . . 59

4.4 Results . . . 61

  4.4.1 Instantaneous adaptation . . . 63

  4.4.2 Hierarchical encoding . . . 65

4.5 Summary . . . 67

5 Implementation and Evaluation **69**

5.1 Implementation . . . 69

  5.1.1 Application Development . . . 69

  5.1.2 Implementation Details . . . 73

5.2 Sample Application . . . 75

6 Conclusion **77**

A SR-RTP API Documentation **79**

A.1 Initialization Functions . . . 79

A.2 Data Path Functions . . . 80

B Quality adaptation **83**

B.1 Buffering requirements 83

B.2 Inter-layer buffer allocation 83

  B.2.1 Scenario 1 84

  B.2.2 Scenario 2 85

List of Figures

1-1 Variable Delay Requires Receiver Buffering. 18
1-2 System architecture. 19

2-1 MPEG encoder block diagram. 24
2-2 Frame dependencies in an MPEG bitstream. 24

3-1 I-frame #48 from the "coastguard" stream with packet loss. 32
3-2 B-frame #65 from "coastguard" stream showing propagation of errors. 33
3-3 Y PSNR values over time with varying degrees of packet loss. 34
3-4 Average PSNR over 100 seconds of 30 frames per second video as a function of the packet loss rate. 35
3-5 The effects of packet loss on frame rate. 37
3-6 Frame dependencies in an MPEG bitstream. 38
3-7 The effects of recovering reference frame data on frame rate. 39
3-8 SR-RTP header for selective reliability. 40
3-9 SR-RTCP Receiver Report for selective reliability. 40
3-10 The benefits of selective reliability. 42
3-11 SR-RTP can provide benefits for channels of various bandwidths. 43
3-12 Conventional approach for I-frame error concealment. 45
3-13 I-frame Concealment Exploiting Temporal Dependencies. 45
3-14 Conventional approach for I-frame error concealment. 46
3-15 I-frame Concealment Exploiting Temporal Dependencies. 46
3-16 Example I-frame with Packet Loss. 47
3-17 I-frame Packet Loss Recovery using Simple Replacement. 48
3-18 I-frame Packet Loss Recovery making use of motion information. 49
3-19 Benefits of Postprocessing. 50

4-1 Window evolution vs. time for SQRT and AIMD congestion controls. 55
4-2 Optimal inter-layer buffer distribution as derived in [57, 59]. 56

4-3 Figure showing the window evolution vs. time for binomial congestion control algorithm 60
4-4 Optimal inter-layer buffer allocation for binomial congestion control. 60
4-5 Experimental testbed for bandwidth adaptation experiments. 62
4-6 AIMD congestion control with immediate adaptation. 62
4-7 SQRT congestion control with immediate adaptation. 62
4-8 AIMD congestion control with simulcast quality adaptation. 64
4-9 SQRT congestion control with simulcast quality adaptation. 64
4-10 Frequency of layer drops of various magnitudes using AIMD congestion control and instantaneous rate adaptation. 65
4-11 AIMD congestion control for hierarchical encoding with receiver-buffered quality adaptation. 66
4-12 SQRT congestion control for hierarchical encoding with receiver-buffered quality adaptation. 66
4-13 Buffering requirements in KBytes for adding layers for AIMD and SQRT algorithms. 67

5-1 Protocol stack for implementation of selective reliable RTP with congestion management. 70
5-2 Summary of SR-RTP API. 70
5-3 Reassembly of ADUs at the SR-RTP enabled receiver. 71
5-4 Congestion Manager Architecture 74
5-5 Sample Application 76

Any intelligent fool can make things bigger, more complex, and more violent. It takes a touch of genius—and a lot of courage—to move in the opposite direction.

- E. F. Schumacher

Chapter 1

Introduction

This thesis presents a video streaming system to deliver high-quality video across the Internet to clients. The Internet is a "best-effort" network, characterized by packet losses, time-varying available bandwidth, and variable end-to-end latencies. We incorporate and evaluate two techniques—selective reliability and bandwidth adaptation—to tackle these problems. We provide software libraries to enable this functionality and describe an MPEG-4 video server that uses our framework to deliver video to Internet clients.

1.1 Motivation

The range of applications being deployed on the Internet today is considerably broader than it was a few years ago. In addition to traditional applications such as interactive terminals (e.g., telnet), bulk file transfer (e.g., email, FTP), and the World Wide Web, the Internet is becoming an attractive medium for a broader spectrum of applications. Applications that rely on the real-time delivery of data, such as video-conferencing tools, Internet telephony, and streaming audio and video players are gaining prominence in the Internet application space. In particular, streaming media applications have considerable potential to change the way people watch video. While most people today think of sitting in front of a television to watch a movie, the ability to deliver high-quality streaming video would allow the Internet to compete with traditional modes of video content distribution.

Using the Internet as a medium for transmission of real-time interactive video is a challenging problem. Typically, video has been broadcast in a continuous fashion over the airwaves or over cable networks, media that allow the synchronous delivery of information. However, because the Internet is a best-effort packet-switched network, data will not arrive at the receiver synchronously. As the data is sent from the sender to the receiver, it can be lost or reordered. While traditional Internet applications *adapt* to these characteristics, these anomalies are potentially detrimental to the delivery of real-time media, which should be highly interactive, reasonably reliable, and at a constant rate.

Although recent efforts have made some progress in streaming media delivery, today's solutions are proprietary, inflexible, and do not provide the user with a pleasant viewing experience [42, 56]. In general, current streaming video applications deliver low quality pictures, require large amounts of buffering and thus do not allow for high user interactivity, do not respond well to the changing conditions on the Internet, and do not cooperate with other applications with which they are sharing bandwidth, such as Web transfers.

We assert that the lack of an open framework hampers innovation, particularly in the area of adaptive video delivery in the face of changing network conditions. While today's streaming applications are closed and proprietary, the emerging MPEG-4 standard is gaining increasing acceptance and appears to be a promising open standard for Internet video [12, 16, 22, 31, 36, 44]. We believe that MPEG-4 has the potential to make significant inroads as the preferred streaming media format over the next few years because of its superior compression, ability to code individual objects in a video stream, and increasing interest from industry.

However, before MPEG-4-based Internet video distribution can be widely deployed, several challenges and problems need to be solved. These include:

- *Handling packet loss.* Packet losses on the Internet can severely hamper the quality of a compressed MPEG-4 bitstream.

- *Handling bandwidth variation.* Available bandwidth varies with time, and the streaming system should adjust its sending rate and the quality of the transmitted bitstream in accordance with these changes.

- *Handling delay variation.* Delays on the Internet are variable, which causes problems for an application that wants to play out received data at a constant rate.

A successful streaming media system should thus adapt to changing network conditions and degrade gracefully in the face of packet loss. We address these problems and present a complete system that enables the adaptive transmission of streaming MPEG-4 video.

1.2 Packet Loss

Packets can be lost on the Internet; this is primarily caused by congestion, which causes packets to queue up and eventually be dropped at intermediate routers. We discuss the effect this can have on video transmission, as well as our proposed solution.

1.2.1 The Problem

While packet loss generally degrades the performance of any Internet data transfer, the effect is much more detrimental on compressed data. Data compression reduces the number of bits required to represent a stream of data by removing redundancy inherent in data. For example, inter-frame video compression algorithms such as MPEG-4 exploit temporal correlation between frames to achieve high levels of compression by independently coding reference frames, and representing the majority of the frames as the difference from each frame and one or more reference frames. Although this reduces the number of bits that must be sent to represent a video sequence, losing bits in a compressed bitstream can be catastrophic because these losses cannot be recovered via redundancy in the data. One approach to making a bitstream more resilient to packet loss is to add redundancy back into the stream (e.g., via error correcting codes); this, of course, offsets some of the gains from compression by increasing the amount of data that must be transmitted.

Thus, compression makes the bitstream less resilient to packet loss, because errors due to packet loss in a reference frame propagate to all of the dependent difference frames; this phenomenon is called *propagation of errors*. There is a fundamental tradeoff between bandwidth efficiency (obtained by compression) and error resilience (obtained by coding or retransmission). Inter-frame compression schemes (such as MPEG-4) achieve significant compression of bits in comparison to other schemes that do not exploit temporal correlation (such as motion JPEG [74]), but they are also less resilient to packet loss because of the dependencies that exist between data from different frames. While many methods have been proposed to add redundancy to the bitstream to allow for more effective error correction [8, 9, 68, 73], they also reduce much of the gains garnered from compression.

Traditional TCP-based Internet applications such as the Web and email emphasize complete reliability over reduced latency. However, real-time applications that stream data at a relatively constant rate, especially those that aim to offer a high degree of user interactivity, require low latency for packet delivery, as well as small latency variance. As such, transport protocols such as TCP [50, 67] that guarantee reliable, in-order delivery of packets do not address the requirements of real-time multimedia applications, for which timely delivery of packets is more important than complete reliability. In an inter-frame video compression scheme, not all bits are of equal importance. Errors in reference frames are more detrimental than those in derived frames due to the propagation of the errors contained in the reference frame and should therefore be given a higher level of protection than other data in the bitstream. One approach is to add redundancy to more important portions of the bitstream, or to code more important portions of the stream at a relatively higher bitrate [1, 28, 45, 65]; however, this approach reduces compression gains and in many cases does not adequately handle the bursty packet loss patterns observed along many Internet paths.

Additionally, streaming video must reconcile the conflicting constraints of delay and error resilience. In order to maintain a high level of user interactivity, delay must remain relatively small

(200 ms is the estimated perceptual tolerance of the user [10]). However, short delay means that lost packets often cannot be retransmitted, thus resulting in a displayed picture of significantly lower quality than in the loss-free case.

1.2.2 The Solution

Our approach to solving this problem while preserving the benefits of aggressive compression is to use *selective reliability*. Some researchers in the past have argued that retransmission-based error resilience is infeasible for Internet streaming because retransmission of lost data takes at least one additional round-trip time, which may be too much latency to allow for adequate interactivity [64, 73]. However, in a streaming system that transports video bitstreams with inter-dependent frames, *careful* retransmission of certain lost packets can provide significant benefits by alleviating the propagation of errors.

To adapt to packet losses, our system uses the concept of application-level framing (ALF) [14]. Because dealing with data loss is application-dependent, the application, rather than the transport layer, is best capable of handling these losses appropriately. The ALF principle articulates that data must be presented to the application in units that are both meaningful to that application and independently processible. These units, called application data units (ADUs), are also the unit of error recovery. We have used this philosophy in our design of a backwards-compatible selective retransmission extension to RTP [62], which provides semantics for requesting the retransmission of independently-processible portions of the bitstream and a means for reassembling fragmented portions of independently processible units. This idea is useful in mitigating the propagation of errors.

1.3 Bandwidth Variation

The bandwidth between any two points on the Internet usually varies over time, because the connecting links are shared with many competing flows, the majority of which are TCP-based applications that are bursty in nature. We discuss the effects bandwidth variation can have on streaming video and present a solution that allows the receiver to:

- Vary the *transmission rate* smoothly in a manner that shares bandwidth well with TCP flows, and

- Adapt the *video quality* to correspond to the average available bandwidth.

1.3.1 The Problem

In addition to providing a means for recovering appropriately from packet loss, a video streaming system for the Internet should adapt its sending rate and the quality of the video stream it sends in accordance with available bandwidth. It is widely believed that the stability of the modern Internet is in large part due to the cooperative behavior of the end hosts implementing the window increase/decrease algorithms described in [2, 33]. A video streaming system that adapts to varying network conditions is preferable to one that is not, because it can deliver video at the highest possible quality at the available bandwidth and share bandwidth fairly with other Internet flows. A streaming system should tailor the quality of transmitted video according to available bandwidth and adjust its rate in a fashion that minimizes the changes in quality seen by the end user while simultaneously delivering a high-quality stream.

Because bandwidth and delay on the Internet vary with time, an application should tailor its rate of delivery to present conditions using *congestion control*, and adjust the quality of the transmitted video according to available bandwidth (a process called *quality adaptation*). To accomplish this, our video server uses information in RTCP receiver reports to discover lost packets and round-trip time variations and adapt its sending rate according to a certain congestion control algorithm using the Congestion Manager (CM) [3] framework. The sender can use the CM to adjust its transmission rate according to a variety of congestion control algorithms, including the additive-increase/multiplicative-decrease (AIMD) algorithm used by TCP [50, 67]. However, rapid oscillations in the instantaneous sending rate, such as those caused by AIMD, can degrade the quality of the received video by increasing the required buffering and inducing layer oscillations.

1.3.2 The Solution

Rate oscillations degrade the quality of received video because they require more buffering to sustain a smooth playout and often result in variable quality, which is visually unappealing to the receiver. Overcoming the oscillatory nature of AIMD congestion control to smoothen bitstream can be done in two ways:

- Alternatives to AIMD congestion control [6, 20, 61].
- Use a combination of quality adaptation and receiver buffering [58].

One way of reducing the magnitude of rate oscillations is to use congestion control algorithms that result in smaller oscillations than AIMD, such as equation-based congestion control [20], TEAR (TCP Emulation at Receivers) [61], or binomial congestion control [6]. In addition, a streaming system can achieve long-term smoothing of video quality in the face of transient changes in bandwidth by using the rules of *quality adaptation* (QA) and receiver buffering developed in [57].

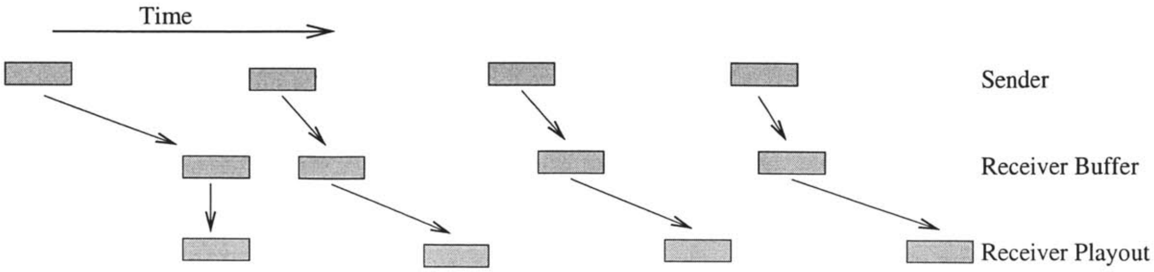


Figure 1-1: Variable Delay Requires Receiver Buffering. The sender streams packets at a constant rate, but the network introduces *jitter*, or delay variation. To combat this, the receiver uses a buffer to smooth the playout of packets.

Receiver buffering not only reduces jitter, but if enough data are buffered, also enables the receiver to sustain momentary drops in the sending rate by playing out of its buffer at a higher rate than the server is currently sending. This buffer accumulates when the sender is transmitting faster than the receiver is playing out, by not aggressively adding layers whenever any spare bandwidth becomes available. Rejaie et al. observe that receiver buffering in conjunction with quality adaptation can reduce the effect of oscillations that result from AIMD. We expand upon the proposed QA scheme so that it can be used in conjunction with binomial congestion controls [6, 18], a family of TCP-friendly congestion control algorithms that can reduce rate oscillation and thus required buffering at the receiver. Using a combination of binomial congestion controls and quality adaptation, our system achieves a smoother playout rate and the ability to playout at a higher overall rate with less buffering at the receiver.

1.4 Delay Variation

Delays on the Internet are variable; that is, the time it takes for a packet to travel from one point to another can vary tremendously and be unpredictable over short periods of time. This phenomenon, called *network jitter* is primarily due to queueing of packets at intermediate routers, but can also happen if packets are taking multiple paths to the destination.

Such an effect can be detrimental to a streaming video system, which would like packets to arrive at a relatively constant rate. To combat such delay variation, playout buffering can be used at the receiver. When packets arrive faster than expected, they can be placed in the buffer for playout at a slightly later time.

Smooth quality of a received video signal depends on appropriate buffering. In particular, receiver buffering must be large enough to:

1. account for network jitter,
2. allow time for retransmission of lost packets, and

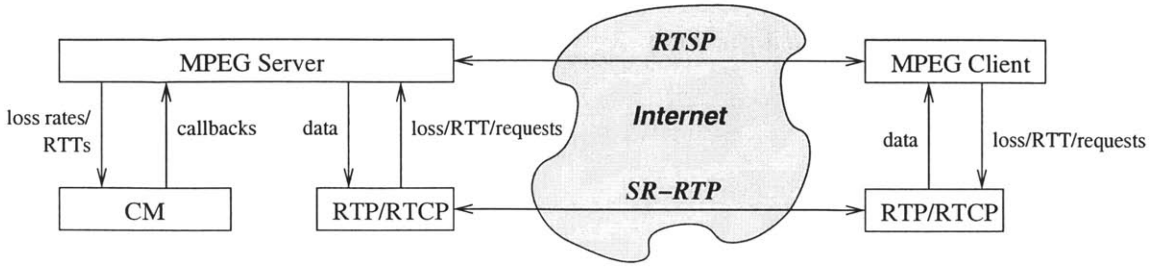


Figure 1-2: System architecture. Feedback is sent to the streaming application via RTCP, which is used to appropriately adjust the transmission rate. Selective reliability is enabled using SR-RTP in combination with the RTCP feedback channel. Using SR-RTP, our backwards compatible extensions to RTP [62], the client can optionally request that certain lost packets be retransmitted.

3. enable quality adaptation.

We will argue that, in many realistic situations, the buffering required at the receiver to combat network jitter is small in comparison to the buffering that is needed to sustain changes in bandwidth variation for quality adaptation.

1.5 System Architecture

We now describe our architecture for unicast streaming of MPEG-4 video that implements techniques for selective reliability and bandwidth adaptation. This architecture has been developed as the basis for next-generation streaming systems such as the DivX Networks platform [16].

1.5.1 Overview

Figure 1-2 shows the components of our system. The server listens for requests on an RTSP [63] port, establishes session parameters via SDP [25], and streams requested data to the client via RTP (over UDP) [62] that has been extended to support application-level framing (ALF) and selective reliability. Feedback is provided to the server via RTCP receiver reports and is used to adjust the congestion window size at the server using the Congestion Manager (CM) [4]. The CM implements a TCP-friendly congestion control algorithm for the MPEG-4 streams and provides an API by which the server adapts to prevailing network conditions.

Our system supports backwards-compatible extensions to RTP/RTCP (which we call *SR-RTP*) that allow for the application-level framing of the data with Application Data Units (ADUs) [14]. ADUs enable fragmentation and reassembly of independently processible units of data and also make selective recovery of application specific data units possible at the receiver. For MPEG-4, one frame of the compressed video bit-stream (separated by VOP start codes) corresponds to one ADU. These frames are packetized by the sender and then, when they are received by the receiver, are reassembled and passed to the application layer for decoding once the complete frame has been received. An ADU may involve multiple packets (or ADU fragments): each is independently named in order to efficiently request and perform selective retransmissions.

1.5.2 Loss-resilience

We have extended RTP to provide selective reliability. Each video frame is an ADU; we specify information such as the sequence number of that ADU in the header. Additionally, should an ADU not fit within one packet (for high bitrate streams), we provide a mechanism for specifying the byte offset within an ADU. The length of the ADU is also contained within the extension header.

The server packetizes the video (in the case of MPEG-4, the bitstream is packetized on resynchronization marker boundaries), labels the packets with ADU sequence numbers and offsets, and sends the SR-RTP packets over UDP. These semantics allow the client to reassemble the packet and to determine if any data is missing. On receiving a packet, the client sends back an ACK to the receiver, acknowledging the successful receipt of an ADU (or portion thereof). Alternatively, the client can send a retransmission request requesting retransmission of a specific portion of the bitstream.

1.5.3 Bandwidth Adaptation

Internet conditions change with time, and a video streaming system should adapt to these changes accordingly. Unlike traditional Internet applications, which seek to transmit at the highest attainable rate, video applications prefer to send data at a relatively constant bitrate. Various researchers have argued that congestion control algorithms such as TCP's additive increase multiplicative decrease (AIMD) algorithm are not amenable to video due to their large rate oscillations.

To combat this, we first use the Congestion Manager (CM) [3] architecture to give the application, rather than the transport layer, control of congestion avoidance. Using this framework, an application is free to use congestion control algorithms such as binomial congestion control [6], equation-based congestion control such as TFRC [20], or TEAR [61], which allow for a much smoother rate of transmission. We have implemented and experimented with binomial congestion control.

In addition to performing congestion control that interacts well with other flows on the Internet, our streaming media server adapts the quality of its transmission based on prevailing network conditions. We define a mechanism is known as *quality adaptation*, which can be performed in a number of ways. One option, called *simulcast*, encodes the bitstream at various target bitrates and switches between the previously encoded layers as the available bandwidth changes. An alternative quality adaptation scheme uses *hierarchical encoding* [37, 40, 72], where the video stream is encoded at a base layer and one or more enhancement layers, which when combined render the stream at a higher quality. As the available bandwidth varies, the number of enhancement layers is adjusted by the server. Our system provides mechanisms for suitable congestion control algorithms and quality adaptation strategies.

1.6 Contributions

This thesis demonstrates the importance of selective reliability for coping with packet loss, and presents an implementation to enable such a framework. Specifically, we make the following contributions:

- An analysis of the effects of packet loss on the overall quality of an MPEG-4 bitstream and the effects of propagation of errors due to packet loss, and an analytical model that explains these effects and quantifies the importance of selective reliability.
- Mechanisms for smoother playout of layered video using a combination of smooth congestion control algorithms and receiver buffering.
- A system to enable the transmission of MPEG-4 video in the face of packet loss, bandwidth variation, and delay variation.

We have designed and implemented a framework that adapts for variations in bandwidth and delay and uses selective reliability to recover limit the propagation of error that can result from transmitting compressed video over a lossy channel. Furthermore, we have developed real-world applications to use this framework and allowed other application developers to make use of these functionalities. The alpha release of the software library supporting selective reliability in RTP ("SR-RTP") is available from `http://nms.lcs.mit.edu/projects/videocm/`; as of early April, the libraries had been downloaded by nearly 5,000 developers. The software is also featured on the ProjectMayo Web site at `http://www.projectmayo.com/`.

This work thus not only presents new ways to deal with the traditional problems of the Internet for streaming video, but it also presents a viable set of techniques that can potentially make the Internet a distribution channel for high-quality streaming video.

What saves a man is to take a step. Then another step. It is always the same step, but you have to take it.

- Antoine de Saint-Exupéry

Chapter 2

Related Work

This chapter overviews prior work in streaming video delivery and discusses how it relates to our system. We first give an overview of the MPEG-4 video encoding standard. We discuss media transport protocols, related error control and concealment schemes, congestion control for streaming media, and approaches to video quality adaptation.

2.1 MPEG-4 Background

The Motion Picture Experts Group (MPEG) started the MPEG-4 project in 1993 with the goal of producing low-bitrate streams, but the standard has since been expanded to include a broader range of bitrates and applications. The most notable addition to MPEG-4 since MPEG-2 (the standard used to compress video for DVD and HDTV) is the capability to represent video sequences as a composition of independent audiovisual objects [31]. However, both standards use similar techniques to exploit spatial and temporal redundancy.

MPEG-4 is the latest standard for inter-frame compression and storage of digital video from the Moving Picture Experts Group [12, 22, 31, 36, 44]. The standard is expected to be the predominant encoding for Internet video, and offers significant advantages over current formats due to its ability to code video sequences on an object-by-object basis and its capability to code at a wide range of bitrates. Companies such as DivX Networks [16] have recently spawned efforts to make the MPEG-4 compliant DivX codec the default encoding format for Internet video by open-sourcing their codec and sponsoring applications built on top of this underlying format [51]. Furthermore, MPEG-4 has some error resilience capabilities, such as video packetization markers, which are useful for the transport of a video stream across a loss-prone network.

Figure 2-1 shows a system block diagram of an MPEG encoder. While *spatial redundancy* can be exploited simply by coding each frame separately (just as it is exploited in still images), many video sequences exhibit *temporal redundancy*, where two consecutive frames are often very similar.

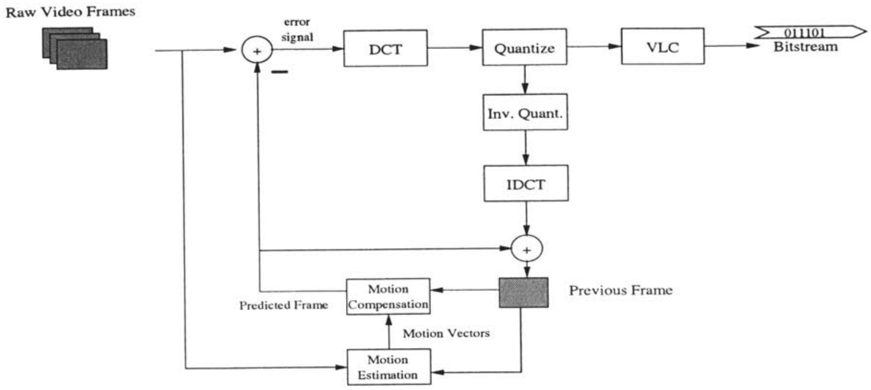


Figure 2-1: MPEG encoder block diagram.

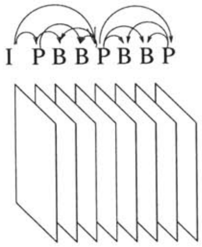


Figure 2-2: Frame dependencies in an MPEG bitstream.

An MPEG bitstream takes advantage of this by using three types of frames.[1]

Figure 2-2 shows the dependencies that exist within an MPEG bitstream. "I-VOPs" or "I-frames" are *intra-coded* images, coded independently of other frames in a manner similar to a JPEG image. These are thus *reference frames* and do not exploit temporal redundancy. MPEG uses two types of dependent frames: predictively coded frames ("P-VOPs" or "P-frames"), and bi-directionally coded frames ("B-VOPs" or "B-frames"). P-frames are coded predictively from the closest previous reference frame (either an I-frame or a preceding P-frame), and B-VOPs are coded bi-directionally from the preceding and succeeding reference frames.

Dependent frames are coded by performing *motion estimation*, which includes a process called *block matching*, and *motion compensation*. Block matching is the process by which 16 × 16 pixel blocks of the dependent image, called *macroblocks*, are matched against macroblocks in the reference frame to find the closest possible match for that block. The amount by which each blocks have moved in relation to the reference frame is encoded in a *motion vector*. The combination of the macroblocks from the reference frame and the motion vectors can produce a rough estimate of the temporally dependent frame, which is how motion compensation is done; the difference between this estimate and the actual frame is the error signal which is coded into the bitstream, along with the motion vectors.

---

[1] In fact, MPEG-4 codes each independent object within a frame as a "VOP", or "video object plane", but for simplicity and without loss of generality, we will use the terms *frame* and *VOP* interchangeably.

2.2 Media Transport

The Real-time Transport Protocol (RTP) [62] is a protocol that provides end-to-end network functions to enable the transport of real-time data, such as video, audio, and simulation data. The Real-Time Streaming Protocol (RTSP) [63] is an application-level protocol that is typically used to control the delivery of data with real-time properties. Our system implementation uses RTSP and RTP and its companion feedback control protocol, RTCP, to transport video data. We provide backwards-compatible extensions RTP to support selective retransmission of packets, or *selective reliability*, and call this protocol extension *SR-RTP*.

RFC 3016 defines an RTP payload format for MPEG-4 for the purpose of directly mapping MPEG-4 Audio and Visual bitstreams onto packets without using the MPEG-4 Systems standard [35]. This specification works in concert with SR-RTP, as it simply defines the manner in which an MPEG-4 bitstream can be mapped into packets. Previous RFCs define payload formats for MPEG-1/2 video and bundled MPEG-2 video and audio layers [13, 30].

Concurrent work proposes one mechanism for performing multiple selective retransmissions of generic media data [11, 43]. This work has a different emphasis from ours as it focuses on determining whether multiple selective retransmissions can be performed in the event of loss of a retransmitted packet and is less flexible with regard to prioritizing retransmissions (this scheme uses only a binary indicator for retransmission). As we show, prioritization of retransmissions can be especially valuable to reduce the adverse effects of error propagation in schemes like MPEG-4. Other concurrent work describes a mechanism for making retransmission requests of lost packets using RTCP [77]. While this provides similar functionality to our SR-RTCP receiver reports, this design does not provide integrated congestion management information, nor allow the receiving application to specify lost regions of data and prioritize sender retransmissions.

Raman et al. propose a mechanism for improving the rendering of images transmitted over the Internet [55]. This work argues that the in-order semantics of TCP cause images to be rendered in bursts rather than smoothly, and thus propose a new transport protocol, the Image Transport Protocol (ITP), which allows images, including compressed JPEG and JPEG2000 images, to be rendered in a smoother fashion by dividing the compressed image into independently-processible units that can be processed out of order as they arrive at the application. Our protocol for selective retransmission, SR-RTP, is based on certain aspects of the ITP work, such as the use of application-level framing (ALF) [14] and receiver-driven retransmission requests.

2.3 Error Resilience

Some prior work has been done with respect to developing error recovery and concealment schemes for real-time video. Many of these efforts rely on a means of specifying priorities for different pieces of the bitstream over others, while others give higher priority to lower layers of the bitstream.

Wah et al. provide a survey of error-concealment schemes for real-time audio and video transmission [73]. They claim that any scheme for error concealment that relies on retransmission is not viable due to the delay imposed by retransmission requests. However, in the case of MPEG-4, where I-frames can occur as infrequently as once per half-second, we argue and show that the benefits of retransmission are clear—if the loss of an I-frame implies the loss of half a second worth of video, then retransmission can help ameliorate jitter that would otherwise result from packet loss. We also note that other researchers show that retransmission can be a feasible option for error recovery in some cases [60].

2.3.1 Coding Techniques

Gringeri et al. analyzed MPEG-4's built-in error resilience capabilities such as video packetization and error correcting codes and examined propagation of errors on an inter-frame video bitstream when bit errors occur [24]. Packetization of the MPEG-4 bitstream significantly improves the decoded video quality significantly by localizing errors. In contrast, we examine propagation of errors due to *packet loss* and develop a model to describe the effects of these errors.

In Quality Assurance Layering (QAL), high priority data is protected with FEC techniques, and the loss of low priority packets do not propagate because each frame is temporally dependent only on the high priority data from the reference frame [45, 65]. However, the utility of such a scheme is limited because the size of the redundant information due to FEC (required for reliable receipt of the high priority layer) counteracts the gains from compression efficiency.

Another approach to error concealment is multiple description coding (MDC) [68], a joint sender-receiver approach for designing transforms. This scheme allows for the division of the bitstream into multiple equally important "descriptions", such that each additional description is useful in enhancing the quality of the received video.

One approach is to use the priorities associated with a bitstream to provide error protection at the time of encoding [27]. This source-coding approach allocates more bits to more important information, such as header and motion information, while allocating fewer bits to texture information, within a particular video packet. While this is a potentially appropriate scheme for wireless networks, it is not applicable to an environment where losses are on a per-packet basis.

2.3.2 Channel Techniques

Bolot and Turletti propose forward error correction (FEC) in several projects as a means for providing error recovery and packet reconstruction [8, 9]. However, this scheme relies on the fact that the FEC information contained in one packet itself is not lost. FEC-based schemes simply add redundant information, which can potentially increase network traffic and potentially aggravate existing packet loss if congestion already exists. Furthermore, these types of schemes reduce compression efficiency and provide little benefit in the event of bursty packet loss. They also provide a mechanism for performing rate control using data provided in RTCP receiver reports [9]. However, this system relies on the application itself to implement its own congestion control mechanisms, whereas we use an application-independent congestion manager for congestion control.

Priority Encoding Transmission (PET) [1] is an application-independent approach for sending messages over a lossy network based on a specified prioritization scheme. In this scheme, the source assigns different priorities to various segments of the bitstream; information is then distributed over all of the packets sent per message, such that each packet contains relatively more information about high priority data. [38] expands on this work to provide a mechanism for using PET to create a hierarchical encoding of MPEG-1 video bitstreams.

An RTP payload format for packet-level forward error correction has been defined to achieve uneven error protection of RTP-encapsulated data [39]. However, this scheme also relies on providing redundant information for error recovery, thus creating unnecessary network traffic.

2.3.3 Retransmission-based Techniques

Papadopoulos and Parulkar describe a prior retransmission-based error control system at the kernel level, using playout buffering, conditional retransmission requests, and other techniques to alleviate the effects of packet loss [49]. In particular, this system employs conditional retransmission requests to avoid triggering late, unnecessary transmissions by abandoning retransmission if the display time is less than the estimated round trip time.

Rhee proposes a retransmission-based error control technique which does not incur additional latency in playout times by rearranging the temporal dependency of frames so that a reference frame is referenced by its dependent frames much later than its display time, thereby masking the delay in recovering lost packets [60]. Specifically, the scheme uses late-arrival packets to reconstruct the reference frames in order to limit propagation of errors in dependent frames.

2.3.4 Error Concealment Postprocessing

This thesis primarily focuses on recovering for errors using mechanisms that do not rely on alteration of the encoded bitstream or processing of the decoded image at the receiver. Alternatively, error concealment can be performed at the decoder using post-processing techniques such as temporal prediction, interpolation, or energy minimization techniques. A survey paper by Wang [75] also gives an in-depth overview of many of these techniques.

A simple approach exploits temporal correlation between frames by replacing a damaged macroblock with a spatially corresponding macroblock from a previous frame. If the missing macroblocks are instead replaced with the motion compensated block (that which is specified by the motion vector of the damaged block), significant recovery can be achieved [23]. One advantage of performing post-processing in this fashion is its simplicity. However, this method requires that the motion information for the missing macroblock is received undamaged; alternatively, the damaged motion information can be estimated from the motion information for the surrounding macroblocks. This approach is similar to the postprocessing techniques discussed in Section 3.5. However, in our method, we are specifically interested in recovering I-frame data to limit error propagation. I-frames never have motion information associated with them, so we must make use of other techniques to estimate the motion-compensated replacement blocks.

Because video signals are typically smooth, both temporally and spatially, missing data can often be interpolated from available data. Prior work proposes interpolating lost coefficients in the frequency domain from the four surrounding neighbor blocks [29]. However, this method cannot be used for inter-coded frames because the DCT coefficients for error signals are not highly correlated in adjacent blocks. As such, this technique is more useful for recovering errors in still images.

Yet other techniques apply that permit the recovery of motion vector information, in the event that motion vector information is lost. Various research has proposed several different means of dealing with the loss of motion vectors; common techniques include setting motion vectors to zeros, using the motion of the corresponding blocks in the preceding frame, or using the mean or median of the motion from spatially adjacent blocks [26]. We are primarily concerned with *generating* motion vector information for the I-frame we wish to repair and use similar techniques to achieve these goals.

Kieu and Ngan have done some prior work in error concealment techniques for layered video, and have discovered that by sending low frequency coefficients in the base layer and higher frequency coefficients in the enhancement layer, it is beneficial to perform error concealment on the coefficients in the enhancement layer rather than simply discarding damaged enhancement layer data [34].

2.4 Congestion Control

Much attention has focused on developing congestion control algorithms for streaming media applications in recent years. Early proposals for multimedia congestion control [32, 48, 58, 66, 69] were essentially variants of TCP without the in-order, reliable delivery of data semantics associated with TCP. More recently, proposals such as TFRC [20], TEAR [61], and binomial controls [6] have focused on the problem of large oscillations associated with TCP's congestion control. In TFRC, the sender explicitly adjusts its sending rate as a function of the measured rate of loss events based on the TCP-friendly equation developed in [47]. In the TEAR protocol, the receiver emulates the congestion window evolution of a TCP sender. The receiver maintains an exponentially weighted moving average of the congestion window, and divides this by the estimated round trip time to obtain a TCP-friendly sending rate. Binomial controls proposed in [6] generalize TCP's increase/decrease rules to derive a family of TCP-friendly congestion control algorithms with varying degree of oscillating behavior.

2.4.1 Congestion Manager

The Congestion Manager (CM) [3, 4] is a framework that allows an application to perform end-to-end congestion control independent of the transport mechanism, *without having to implement its own congestion control.* For example, an application may wish to perform TCP-style congestion control, even though complete reliability and in-order packet delivery guaranteed by TCP are not required. The Congestion Manager separates congestion control from the transport semantics and provides the application to perform congestion control in a fashion well-suited to the needs of that particular application.

Our video streaming application uses CM to perform congestion control while using SR-RTP as the transport layer. Using CM enables our application to be *TCP-friendly* when streaming video to a receiver. In Chapter 4, we will explore appropriate congestion control mechanisms to use for video streaming.

2.5 Quality Adaptation

Rejaie et al. propose a quality adaptation scheme using receiver buffering for AIMD-controlled transmission and playback for hierarchically-encoded video [57, 59]. Long-term coarse-grained adaptation is performed by adding and dropping layers of the video stream, while using AIMD to react to congestion. Receiver buffering alleviates the short-term variations in the sending rate caused by the oscillatory nature of AIMD. A new layer will be added only if, at any point, the total amount of buffering at the receiver is sufficient to survive an immediate backoff and continue playing all of the existing layers plus the new layer, and the instantaneous available bandwidth is greater than the consumption rate of the existing layers, plus the new layer. When the total amount of buffering falls below the amount required for a drop from a particular rate, then the highest layer is dropped. Additionally, buffer space is allocated between layers so as to place a greater importance on lower layers, thereby protecting these layers if a reduction in the transmission rate were to occur. The conditions for the addition and deletion of layers and inter-layer buffer allocation for AIMD and hierarchical encoding are described in [59].

Great services are not canceled by one act or one single error.

- Benjamin Disraeli

Chapter 3

Selective Reliability

Packet loss can have detrimental effects on the quality of received video because limited redundancy in compressed video streams reduces resilience to data loss. While traditional Internet applications can recover from packet loss using the complete, in-order delivery semantics of transport protocols such as TCP [50, 67], real-time streaming applications that require low latency cannot afford to wait for the complete retransmission of all lost packets. The Real-Time Transport Protocol (RTP) [62] is commonly layered on top of UDP and provides some features such as timestamping and sequencing that aid in the real-time transport of data, but it does not offer any means for recovering lost data. Traditionally, retransmission has not been used as a mechanism for recovering from packet loss for real-time video.

The effects of packet loss can be alleviated using one or more of the three following mechanisms:

- Selective retransmission.
- Postprocessing error concealment.
- Error correcting codes.

This thesis focuses on using a combination of selective retransmission (enabled by our backwards-compatible extensions to RTP, called *SR-RTP*) and postprocessing error concealment at the receiver to recover from packet losses that occur in more important portions of the compressed bitstream.

Some arguments have been made against using retransmission for error recovery [9, 68, 73], primarily because of the latency required to do so. However, because of the nature of inter-frame compression, certain types of packet loss can be excessively detrimental to the quality of the received bitstream. We show that such losses can be corrected via retransmission *without* significantly increasing delay, using only a few frames' worth of extra buffering.

In this chapter, we develop the case for *selective reliability*, whereby certain portions of an MPEG-4 bitstream can be transmitted reliably via retransmissions. In any inter-frame video compression

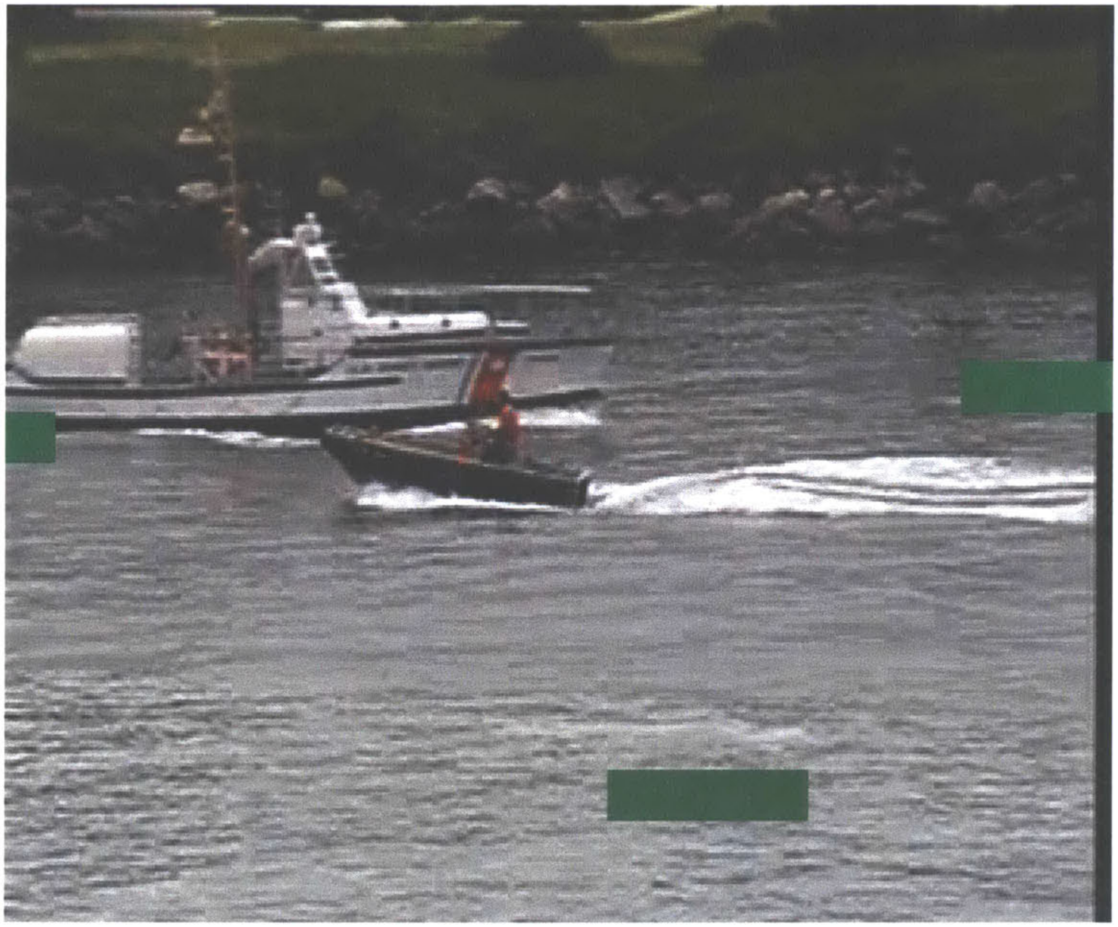


Figure 3-1: I-frame #48 from the "coastguard" stream with packet loss. Y PSNR = 21.995697

standard (e.g., MPEG-4), an inherent tradeoff exists between achieving a high compression ratio and limiting the propagation of errors between frames. Schemes such as forward error correction (FEC) increase error resilience but limit the gains that are achieved with compression. Using more reference frames ("I-frames") can limit the propagation of error, but this also increases the required bitrate since reference frames are considerably larger than temporally predicted frames.

While we primarily focus on the use of selective retransmission for the purposes of error recovery, we also show how selective retransmission can be used in conjunction with a variety of other error control and concealment techniques. For example, when delay is particularly high, retransmission of any lost packets may not be feasible. As such, while the selective retransmission enabled by SR-RTP is useful in certain circumstances, in other circumstances other types of error concealment may be more useful. Section 3.5 explores how SR-RTP can be used in conjunction with alternate packet loss recovery schemes.

The rest of this chapter describes the problem in detail, presents an analysis of video frame rate in the presence of packet loss, and makes a quantitative case for selective reliability. We analyze the quality degradation caused by packet loss, focusing on whole packet erasures to model congestion-

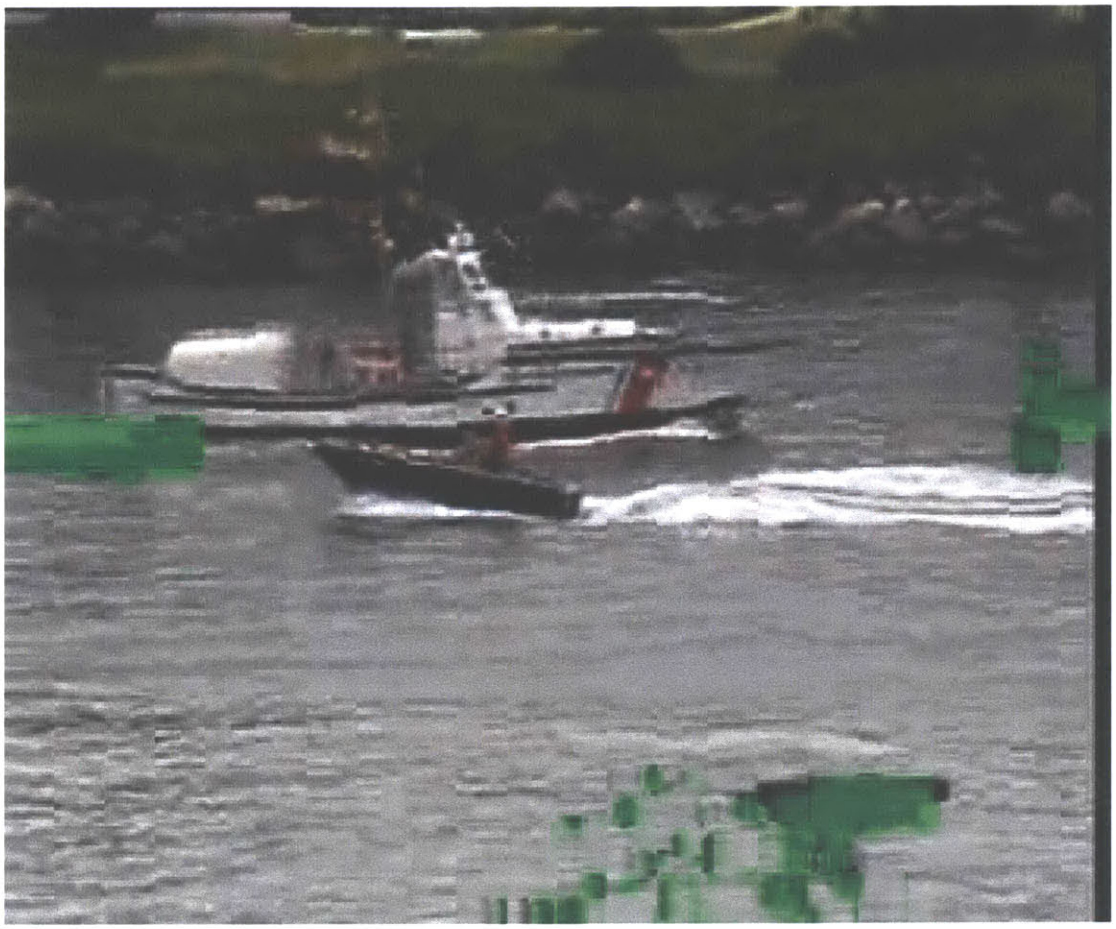


Figure 3-2: B-frame #65 from "coastguard" stream showing propagation of errors. Y PSNR = 17.538345

related packet loss.

3.1 Problem: Error Propagation

The ability to successfully decode a compressed bitstream with inter-frame dependencies thus depends heavily on the receipt of reference frames (i.e., I-frames, and, to a smaller degree P-frames). While the loss of one or more packets in a frame can degrade its quality, the more problematic situation is the propagation of errors to dependent frames. This can often aggravate the effects of a packet loss, particularly in a high-motion sequence where the motion vectors for a missing portion of the reference frame are large in magnitude. An example of error propagation for a particular video sequence ("coastguard") is shown in Figures 3-1 and 3-2; the rectangular patch near the bottom of Figure 3-1 is the result of a single loss in an I-frame (no local error concealment is done in this example). This error spreads to neighboring frames as well, as shown in Figure 3-2 which depends on several preceding differentially coded frames.

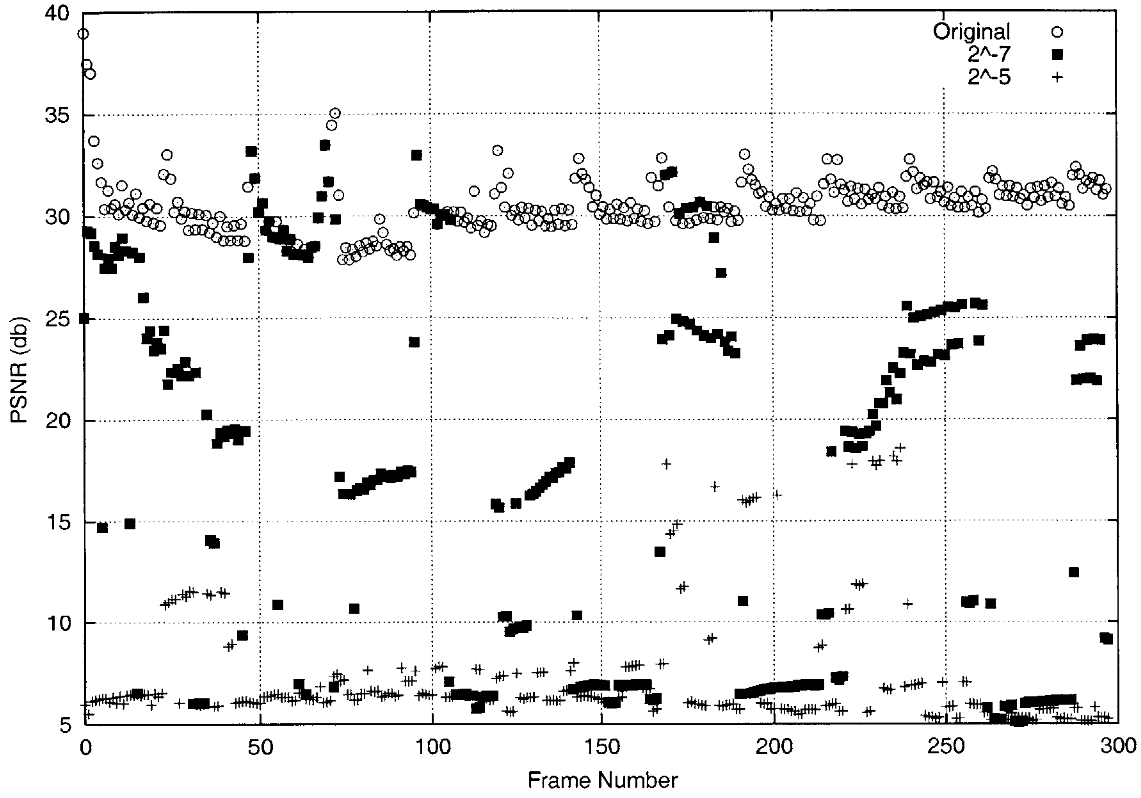


Figure 3-3: Y PSNR values measured against the original frames with varying degrees of packet loss. As the packet loss rate increases, the frame-by-frame PSNR drops dramatically, thus lowering the fraction of frames received which are suitable for display.

Figure 3-3 shows the evolution of frame-by-frame PSNR for the luminance (i.e., "Y") component as a function of the original raw frame number for various packet loss rates. Peak Signal to Noise Ratio (PSNR) is an indicator of picture quality that is derived from the root mean squared error (RMSE). The PSNR for a degraded $N_1 \times N_2$ 8-bit image f' from the original image f is computed by the formula

$$PSNR = 20\log_{10} \frac{255}{\left(\frac{1}{N_1 N_2}\sum_{x=0}^{N_1-1}\sum_{y=0}^{N_2-1}\left[f(x,y) - f'(x,y)\right]^2\right)^{1/2}}$$

The evolution for a decoded bitstream with no packet loss is also included as a baseline. As the packet loss rate increases, the quality (in PSNR) of an increasing number of the decoded frames becomes too poor for viewing. We recognize that PSNR is not an accurate metric of perceptual quality, but we use it because it is simple and is an approximate indicator. We generalize this in Figure 3-4 by averaging the observed frame rate over time for a given video sequence. If we assume that the viewer is capable of tolerating only frames that are at least a certain PSNR quality, and that frames below such a quality are not pleasing to view, we can show how packet losses degrade

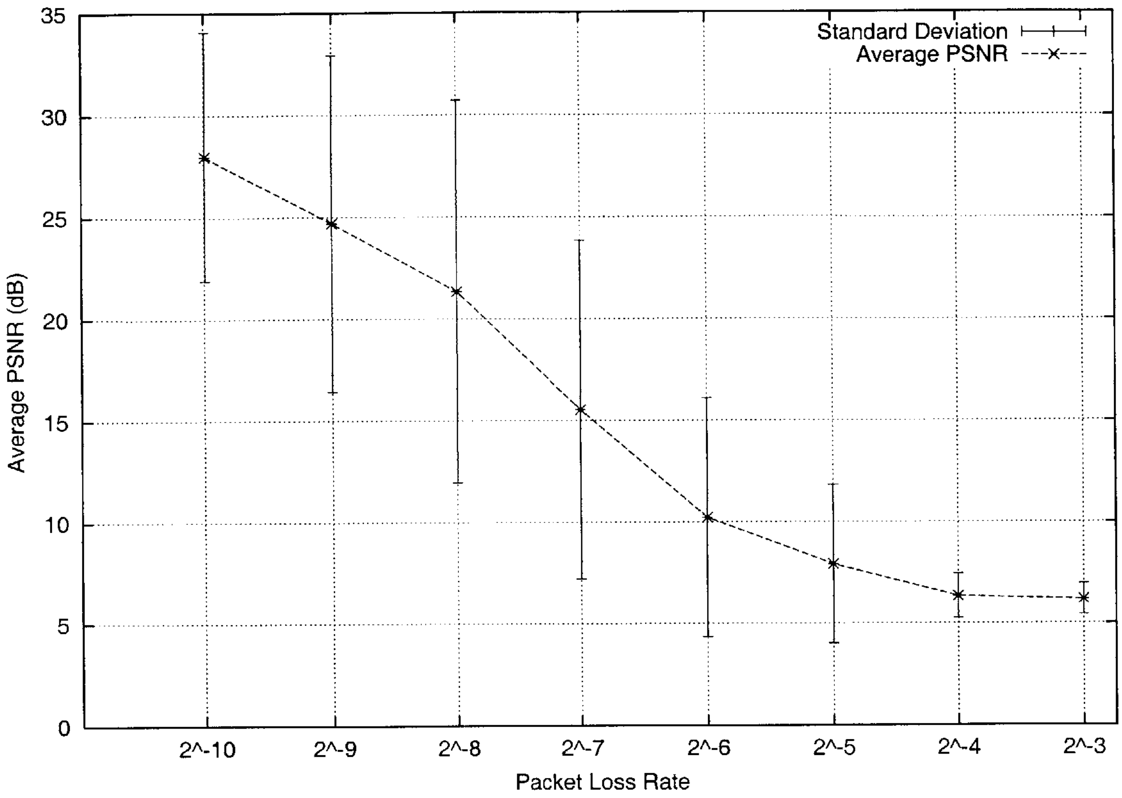


Figure 3-4: Average PSNR over 100 seconds of 30 fps video as a function of the packet loss rate. As the packet loss rate increases, the overall quality of the bitstream severely degrades.

frame rate. We model this in Section 3.2.

3.2 Packet Loss Model

We now analyze the effects of packet loss on the observed frame rate at the receiver. Using these results, we argue that, under certain circumstances, selective reliability can improve the quality of the received bitstream.

Our model is based on two important premises. The first is that packet loss will result in degradation of quality of the video stream at the receiver; i.e., that there is some signal loss caused by the loss of a packet. This is true in general, unless packet-level FEC or erasure codes are extensively used (but notice that such mechanisms do reduce the effective bitrate of the transmission). The second premise is that, below a certain PSNR level, frames are not viewable by users. While it is true that PSNR does not necessarily accurately model perceptual quality, it has been extensively used in the literature. Because the viewability threshold varies from sequence to sequence, we perform our analysis for several PSNR thresholds. We note that the general method can be used for any quality metric, not just PSNR.

3.2.1 Experimental Results

To better understand how packet loss affects the quality of video received by a client, we will first look generally at how packet loss affects the average PSNR of a video sequence. In addition to PSNR, the quality of delivered video depends on the *frame rate*, which is the rate at which frames whose PSNR is above some threshold is played at the receiver.

Figure 3-4 shows how increasing packet loss rates greatly degrade the overall quality of the received pictures. In this experiment run on the "coastguard" stream, packets were dropped according to a Bernoulli process at each loss rate, ranging from 2^{-10} to 2^{-3} in powers of two. The vertical error bars plot the standard deviation of received PSNR across 100 seconds of a 30 frames per seconds video sequence. Frames with PSNR values smaller than 20 dB are generally unviewable, which means that even individual frames are not particularly useful without a correction mechanism at packet loss rates larger than 2^{-8}.

Figure 3-5 shows the measured results of the resulting frame rates as a function of the packet loss rate, with one curve per PSNR threshold. For the "coastguard" stream, we measure the number of frames per second, on average, that are above a set PSNR threshold and plot it as a function of the Bernoulli packet loss rate. As the picture quality threshold increases for a given packet loss rate, the number of acceptable frames in the sequence (the frame rate) decreases. For a given picture quality threshold, an increase in the packet loss rate results in a considerable reduction in the frame rate. This graph shows that as the packet loss rate p increases, the frame rate degrades roughly as $f(p) = \alpha(1-p)^c$ for some constants α and c.

To understand better how packet loss affects frame rate, we now develop a simple analytic model to explain these results. We find that the analytic model matches the experimental results rather well.

3.2.2 Analytic Model

Our goal is to derive a relationship between the packet loss rate p and the observed frame rate f. When calculating the frame rate, f, we assume that if the quality of a frame (i.e., PSNR) falls beneath a certain threshold, then the frame is "dropped." We express the observed frame rate f as $f_o(1-\phi)$, where ϕ is the "frame drop rate", the fraction of frames dropped, and f_o is the frame rate of the original bitstream in frames per second (e.g., 30 fps).

The frame drop rate ϕ is a sum of conditional probabilities:

$$\phi = \sum_i P(f_i) \cdot P(\overline{F}|f_i) \tag{3.1}$$

where i runs over the three possible frame types (I, P, and B), and $\overline{F}$ represents the event that a frame is "useless" because it falls below a certain quality threshold. f_i is the event that the

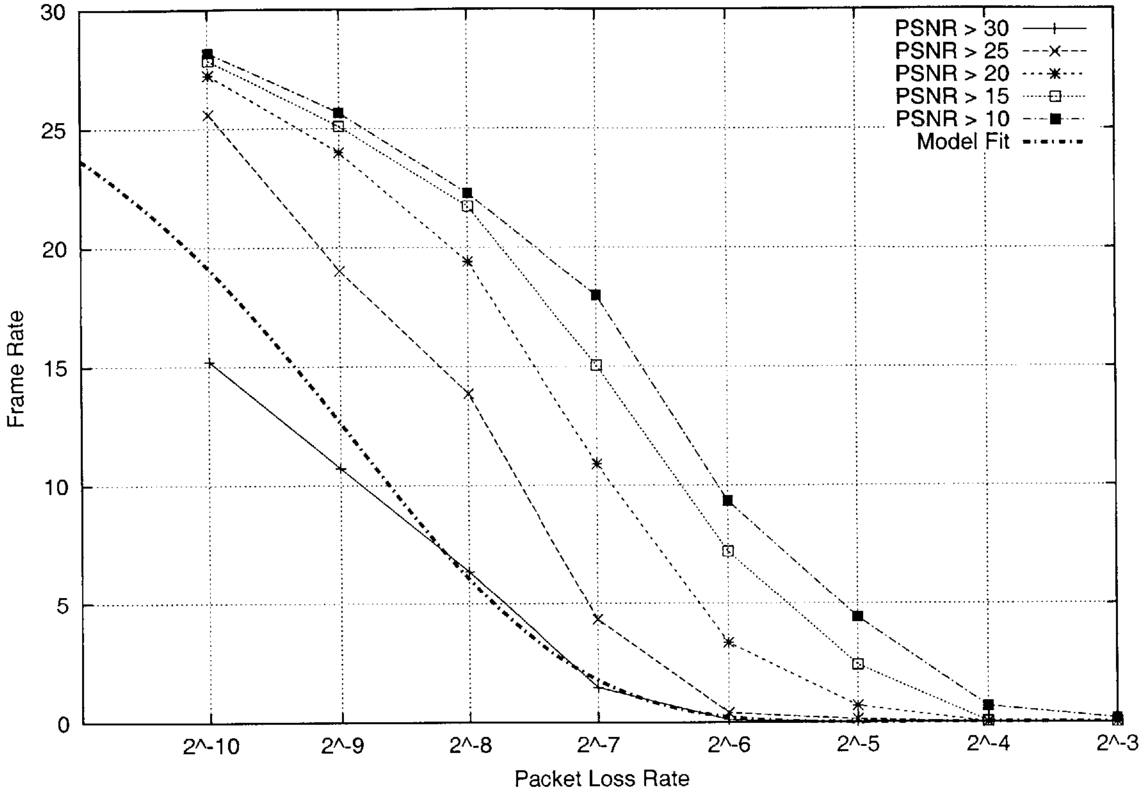


Figure 3-5: The effects of packet loss on frame rate.

corresponding frame is of type i. The *a priori* probabilities $P(f_i)$ can be determined directly from the fractions of bitstream data of each frame type.

Next, we express the conditional probabilities $P(\overline{F}|f_i)$ for each frame type f_i. We do this under the simplifying assumption that if even one packet within a frame is lost (or the effects of one packet loss from a reference frame are seen), that the frame is rendered useless (relaxing this assumption makes the analysis more complicated, although the general form of the result does not change). In this case, determining $P(\overline{F}|I)$ is simply a Bernoulli random variable, expressible as one minus the probability that no packets are lost within the I-frame. Thus,

$$P(\overline{F}|I) \quad = \quad 1 - (1-p)^{S_I} \tag{3.2}$$

where S_I is the number of packets on average in an I-frame, and p is the packet loss rate.

The conditional probabilities for P and B frames are somewhat more involved, and require an understanding of the inter-frame dependencies in MPEG video. These dependencies are shown in Figure 3-6. Every P-frame depends on the preceding I or P frame in the "group of video object planes" (GOV), and every B-frame depends on the surrounding two reference frames (the closest two I or P frames that surround it). Thus, the successful decoding of a P-frame depends on *all* I

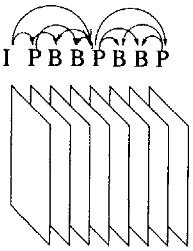


Figure 3-6: Frame dependencies in an MPEG bitstream.

and P frames that precede it in the GOV, and the successful decoding of a B-frame depends on the successful decoding of the surrounding reference frames, which implies the successful decoding of all preceding I and P frames *and* the succeeding I or P frame. These dependencies can be expressed in the following relationships:

$$
\begin{aligned}
P(\overline{F}|P) &= \frac{1}{N_P}\sum_{k=1}^{N_P}\left(1-(1-p)^{S_I+kS_P}\right) && (3.3)\\
P(\overline{F}|B) &\leq \frac{1}{N_P}\sum_{k=1}^{N_P}\left(1-(1-p)^{S_I+(k+1)S_P+S_B}\right) && (3.4)
\end{aligned}
$$

Here, S_P is the average number of packets in a P-frame, N_P is the number of P-frames in a GOV, and S_B the number of packets in a B-frame. These simplify to the following closed form expressions:

$$
\begin{aligned}
P(\overline{F}|P) &= 1-\frac{(1-p)^{S_I}}{N_P\left(1-(1-p)^{S_P}\right)}\left(1-(1-p)^{S_P N_P}\right) && (3.5)\\
P(\overline{F}|B) &\leq 1-\frac{(1-p)^{S_I+S_P+S_B}}{N_P\left(1-(1-p)^{S_P}\right)}\left(1-(1-p)^{S_P N_P}\right) && (3.6)
\end{aligned}
$$

We can then obtain an expression for ϕ using equations 3.1, 3.2, 3.5, and 3.6. Given this expression for ϕ, we can determine $f = f_o(1-\phi)$, given values of N_P, S_I, S_P, S_B, and f_o. We have graphed this curve in Figure 3-5 using the parameters of the "coastguard" bitstream we used in our experiments; the model matches the experimental results rather closely.

This result can by extended to derive analytical results for lower PSNR thresholds, assuming that there is a relationship between the number of packets lost in a particular frame and PSNR degradation. Instead of performing the calculations so that one packet loss results in a "useless" packet, we can generalize to allow for n losses, with a larger value of n corresponding to a lower threshold PSNR.

3.3 Selective Reliability is Beneficial

We have established that packet loss substantially affects the frame rate of a received video sequence and would like to somehow recover some of these packet losses. While it would be preferable to be

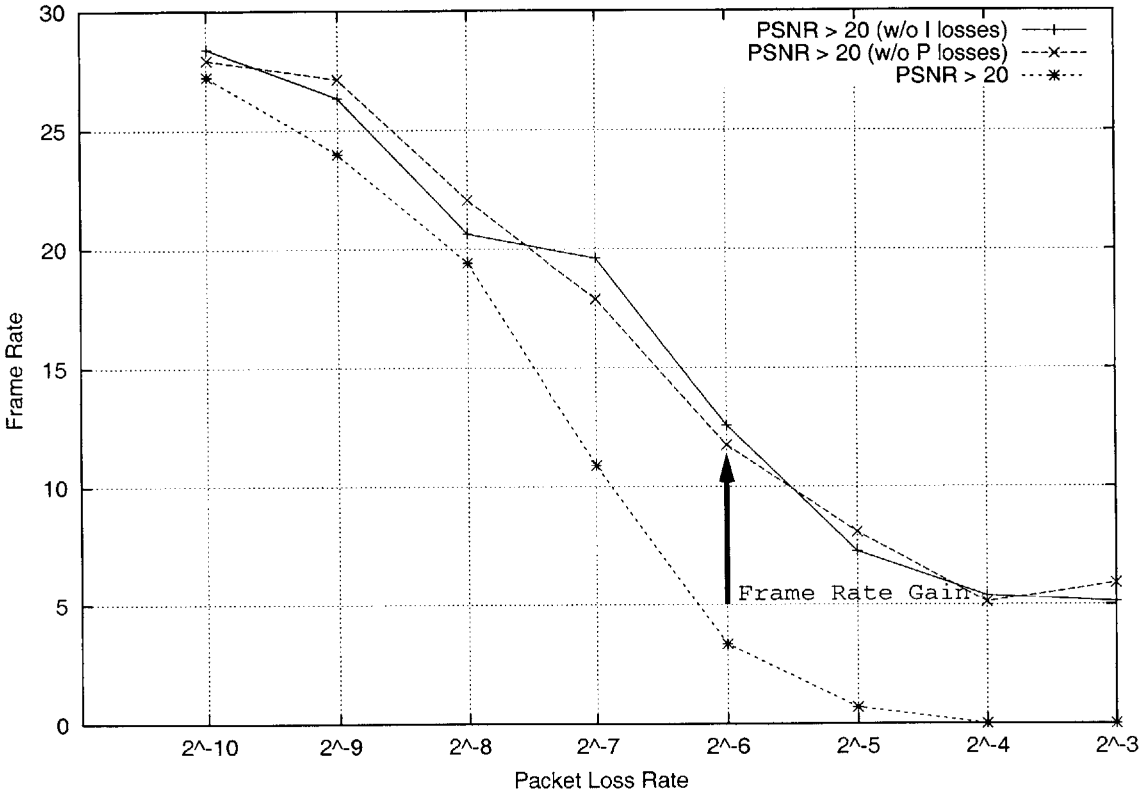


Figure 3-7: The effects of recovering reference frame data on frame rate. By recovering packet losses in I-frames, the frame rate for a given acceptable quality can be increased up to 3 times. This graph also shows that recovering all P-frames is roughly as effective as recovering only I-frame data.

able to recover all packets, the latency of the network, as well as bandwidth constraints, limit this possibility. Fortunately, the structure of an MPEG-4 bitstream allows us to capitalize on the notion that some data are more important that others. By judiciously recovering the more important data in the bitstream, we can substantially increase the observed frame rate.

Figure 3-7 shows the effects of recovering lost I-frame packets via retransmissions on the effective frame rate for a given PSNR threshold of 20 dB. Recovering I-frame data can increase the effective frame rate significantly, in some cases by up to three times the frame rate without recovery. One upper curve shows the effects of recovering only I-frame data, whereas the other curve shows the effects of recovering only P-frame data. In both cases, the frame rate is significantly increased; this shows that recovering only the I-frame packets in a group of pictures results in comparable gains to recovering all P-frame data across a group of pictures. Therefore, by recovering either the I-frame data or the P-frame data via selective reliability, it is possible to significantly improve the quality of received video—there is no real need to recover all missing packets. Furthermore, recovering missing B-frame packets is not particularly useful.

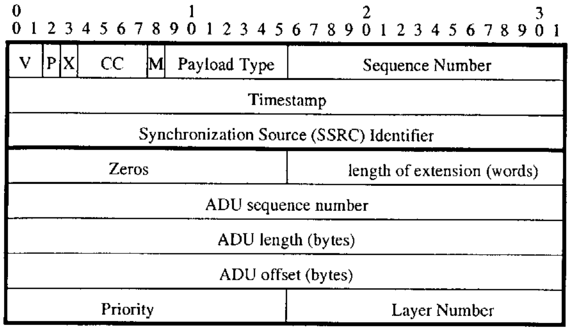

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
V
P
X
CC
M
Payload Type
Sequence Number
Timestamp
Synchronization Source (SSRC) Identifier
Zeros
length of extension (words)
ADU sequence number
ADU length (bytes)
ADU offset (bytes)
Priority
Layer Number

Figure 3-8: SR-RTP header for selective reliability.

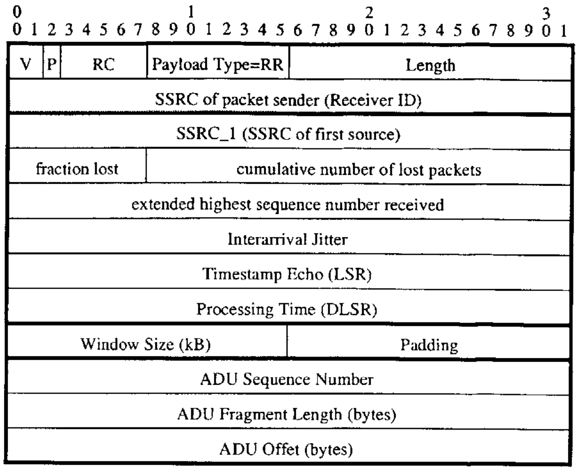

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
V
P
RC
Payload Type=RR
Length
SSRC of packet sender (Receiver ID)
SSRC_1 (SSRC of first source)
fraction lost
cumulative number of lost packets
extended highest sequence number received
Interarrival Jitter
Timestamp Echo (LSR)
Processing Time (DLSR)
Window Size (kB)
Padding
ADU Sequence Number
ADU Fragment Length (bytes)
ADU Offet (bytes)

Figure 3-9: SR-RTCP Receiver Report for selective reliability.

3.4 SR-RTP Protocol

This section provides an overview of the SR-RTP protocol, a backwards compatible extension to RTP that allows for the selective retransmission of lost packets. Using SR-RTP, it is possible to recover from the losses

3.4.1 Header Formats

Figure 3-8 shows the extended RTP header with extension to enable selective retransmission of lost packets. The first three octets of the packet are identical to the RTP header format specification [62]. Additionally, we have provided a generic selectively reliable RTP (SR-RTP) extension that provides for application-level framing (ALF) [14] as well as selective reliability. The *Zeros* field is a field that is required by the standard to allow multiple interoperating implementations to operate independently with different header extensions; we set this to all zeros.

The *ADU sequence number* field uniquely identifies the ADU; in the case of MPEG-4 video, one frame corresponds to one ADU, so this is equivalent to a frame number. The *ADU length* field indicates the number of bytes contained within that particular ADU; this allows the transport layer to detect missing packets at the end of an ADU. The *ADU offset* uniquely identifies one packet within an ADU and allows for reassembly of a packet when reordering occurs. The final octet provides a *Priority* field that allows the transport layer to specify the relative importance of packets. In particular, for the purposes of our experiments, we mark MPEG-4 I-frames with a high priority so that a retransmission request is sent on an I-frame loss but not a P-frame or B-frame loss. The *Layer* field is used when transmitting layered video to specify the layer of video to which the packet corresponds; this feature can be used to calculate playout times, in decoding, or for caching purposes.

Figure 3-9 shows an SR-RTCP receiver report, largely the same as an RTCP receiver report, but with profile-specific extensions added to the end of the header. The *Length* field indicates how many requests to expect at the end of the header. The first four octets of the extension serve as an ACK to the sender acknowledging the receipt of a particular ADU fragment and report the current window size of the receiver for flow control purposes. Optionally, the report can include one or more *ADU requests*, of 3 octets each; these requests uniquely identify the ADU fragment that is to be retransmitted.

3.4.2 Loss Detection and Recovery Decisions

SR-RTP detects packet loss by finding gaps in packet arrivals, which can be determined given information about the length of each ADU and the offset of each packet within an ADU. We assume that I-frames consist solely of intra-coded macroblocks, and predicted frames consist primarily of predicted macroblocks. In such a situation, the priority for retransmission of missing blocks is generally determined from surrounding blocks. Specifically, there are four cases of packet loss that must be detected:

- *Mid-frame loss.* A mid-frame loss is detected simply by detecting a gap in the reconstructed ADU. The priorities of the missing packets are equal to the priority of the surrounding packets. In the event that surrounding packets in the same ADU have differing priorities, the highest priority is assumed for the missing portion.

- *Start-of-frame loss.* A start of frame loss is detected in a similar fashion to a mid-frame loss. If the first packet received for a particular ADU has a nonzero offset, a loss will be detected at the start of the frame with priority for those packets equal to those that *follow* the gap.

- *End-of-frame loss.* If the number of bytes in a particular ADU is less than the reported length for that ADU, and no gaps exist in the received data, a loss will be detected at the end of the frame with priority for those packets equal to those that *precede* the gap.

- *Complete frame loss.* A complete frame loss can be detected by a gap in the ADU sequence number space. In this case, our system foregoes any retransmissions of that frame's data. The likelihood that a lost frame is an I-frame is very low since I-frames are large; because of the size of an I-frame, complete retransmission is also expensive and should be avoided in general.

Using this logic, SR-RTP detects packet loss and optionally requests retransmission of ADU gaps based on the determined priority of the lost region. As a simple scenario, priorities could be assigned such that missing I-frame packets are retransmitted, while other gaps in data are ignored. A more complex retransmission policy could assign different priorities to different frames. For example, missing packets from P-frames might be retransmitted with varying priorities, since P-frames that

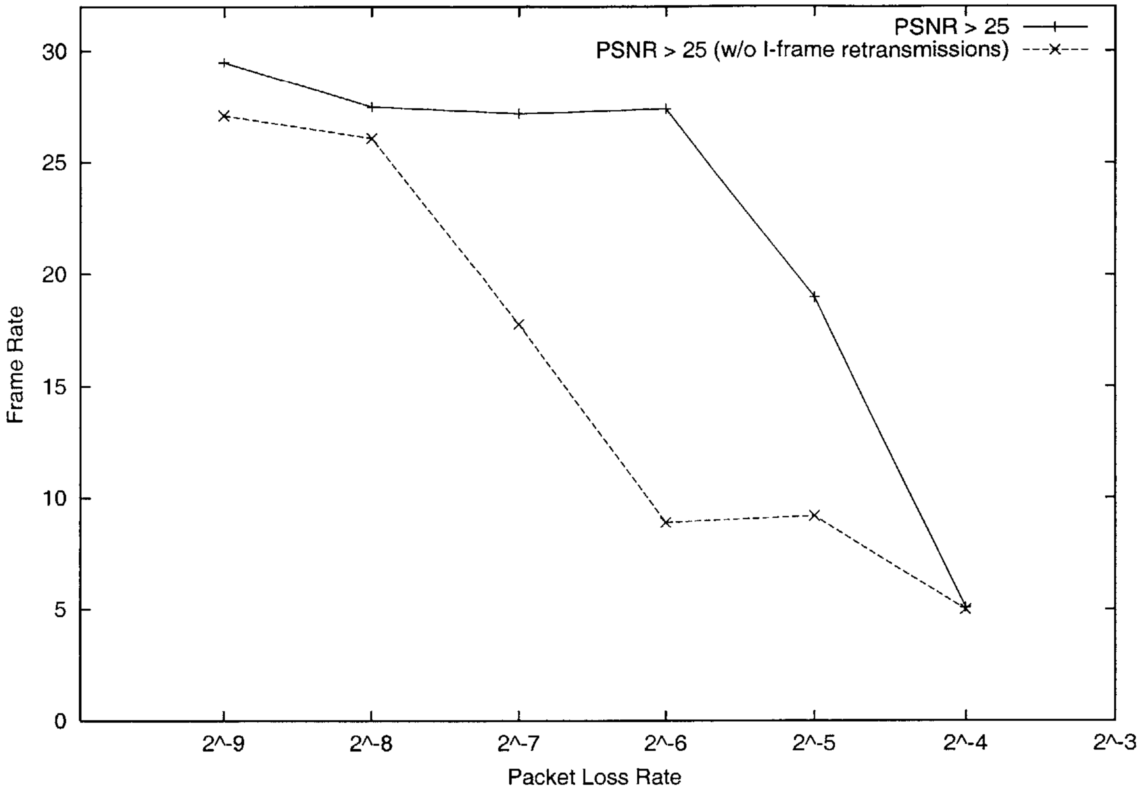


Figure 3-10: The benefits of selective reliability. Substantial quality improvement can be achieved by performing selective retransmission of I-frames.

are closer to the preceding I-frame are more valuable for preserving picture quality than later P-frames in the GOV.

3.4.3 Results

We conducted experiments to show that selective reliability is both a feasible and beneficial means of improving the quality of received video. The video server (running on a Pentium 4 1.5 GHz Linux 2.2.18 box, with Congestion Manager) streamed data to the receiver (a Pentium II 233 MHz Linux 2.2.9 box) across a 1.5 Mbps link, configured using Dummynet [17]. To examine the gains of selective reliability for various packet loss rates, we streamed 300 frames of a 20 Kbps 30 fps sequence across a 1.5 Mbps 50 ms link, varying packet loss rates from 2^{-10} to 2^{-3}. We also studied the performance of SR-RTP for various bandwidths by transmitting 300 frames of 20Kbps 30 fps bitstream across a 50 ms link with a 2^{-5} packet loss rate.

Our experiments show that selective retransmission of I-frame data can result in significant performance gains. We present our findings from experiments on an emulated network that show considerable performance improvement for only a small amount of buffering, and discuss the tradeoff between reliability, interactivity, and general buffering requirements.

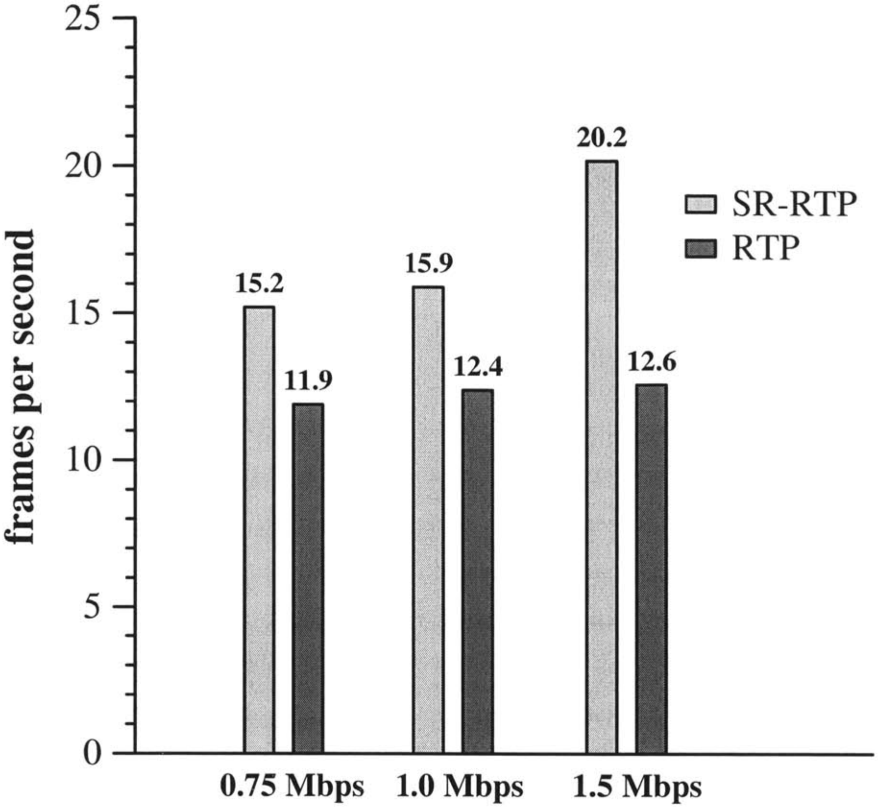


Figure 3-11: SR-RTP can provide benefits for channels of various bandwidths. This graph shows results for various bandwidths using a fixed PSNR threshold of 25 dB and packet loss rate of 2^{-5}.

Benefits of Selective Reliability

Using the 200 ms of initial buffering and the buffering required to combat round-trip time jitter (actually, in the case of these experiments, buffering for retransmission was dwarfed by the amount of buffering required to combat round-trip time jitter), we were able to achieve significant gains in resulting video quality by performing selective retransmissions of I-frame data.

Performing selective recovery on important data within an MPEG-4 bitstream results in significant improvements in perceptual quality. As mentioned in the previous section, different amounts of buffering will allow for a variable amount of selective retransmission. Figure 3-10 shows two curves: the bottom curve is the resulting picture quality for various packet loss rates without performing selective retransmission, and the upper curve shows the corresponding picture quality that can be achieved by using selective retransmission. This graph shows the potential for quality gain that exists under certain conditions. For other quality thresholds, bitrates, etc., the benefits will vary; nevertheless, it is clear that selective reliability can be a boon in certain circumstances. These results generally correspond to our expected results in Figure 3-7.

In a second experiment, we fixed the acceptable picture quality at 25 dB and the packet loss rate at 2^{-5} and examined the benefits of selective reliability for various bandwidths. The results in Figure 3-11 show that selective reliability can provide significant frame rate improvements at different bandwidths.

Buffer Requirements

There is a fundamental tradeoff between the amount of reliability obtained via retransmission and the degree of interactivity possible. For instance, one extreme is simply to transmit the bitstream over TCP; while this provides complete reliability, the degree of interactivity is small because of the delays incurred while achieving complete reliability [41]. Smooth quality of a received video signal depends on appropriate buffering. In particular, receiver buffering must be large enough to (1) account for network jitter, (2) allow time for retransmission of lost packets, and (3) enable quality adaptation. The buffering required to counteract network jitter is a function of the variance in network delay, where the instantaneous jitter j_i can be expressed as $|(A_i - A_{i-1}) - (S_i - S_{i-1})|$ [41, 62]. Using this, the required buffering to counteract network jitter is $\beta\delta_i$, where δ_i is smoothed jitter; smaller values of β reduce overall delay, and larger values decrease the likelihood of late (hence, effectively lost) packets. Buffering for retransmission of lost packets also depends on the absolute network round-trip time. Buffering for quality adaptation depends on the absolute transmission rate. A larger rate results in a larger backoff in the event of a packet loss, and thus requires more buffering to sustain playout at the current layer. We have shown that required QA buffering is $O(R)$ for SQRT congestion control and $O(R^2)$ for AIMD [18].

The dominant factor depends on the relation of the absolute round-trip time to the RTT variance, as well as the absolute transmission rate. As the absolute RTT becomes large with respect to RTT variance, buffering due to retransmission requests will dominate buffering required to counteract jitter, and vice versa. As the absolute bitrate grows large, the amount of buffering required for QA will increase; using more aggressive congestion control algorithms such as AIMD also result in more buffering required for QA.

3.5 Receiver Postprocessing

This section focuses on the benefits that the SR-RTP protocol extensions provides for performing error concealment via decoder post-processing and argues that

- SR-RTP is a complementary scheme that can be used in combination with other techniques, and

- SR-RTP helps provide information about packet loss to the application that can be used in performing other error concealment techniques.

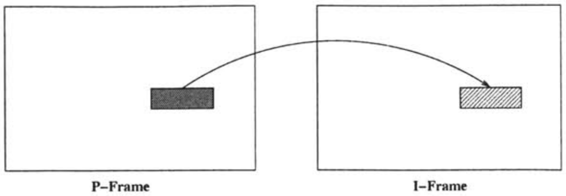


Figure 3-12: Conventional Approach. Replace missing I-frame macroblocks with the corresponding macroblocks from the preceding P-frame. This can work appropriately in scenes were the background is relatively uniform or there is low motion in the region with missing macroblocks. Replacement requires knowledge of the error locations, as well as a preceding frame that has appropriate texture.

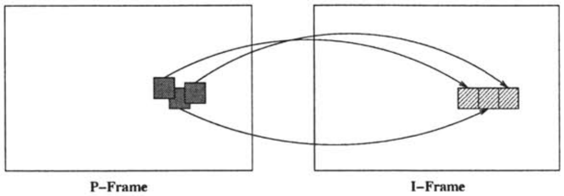


Figure 3-13: Improved Approach. In scenes with high motion, simple replacement will not work as well, because the appearance of the missing region no longer corresponds well with the same region in previous frames. However, if texture and motion information are preserved from previous frames, we can essentially perform motion compensation on the I-frame to yield acceptable results.

We propose two postprocessing routines for performing temporal error concealment of I-frames that can be used at the decoder to recover errors in I-frames and provide similar benefits to those discussed earlier in this chapter.

3.5.1 Recovery Techniques

In this section, we examine different ways of recovering I-frame data in the face of packet loss. First, we describe a conventional macroblock replacement approach, where missing macroblocks are replaced. However, this does not work well in high-motion scenes.

Because motion in video sequences tends to be highly correlated, however, we can make use of motion information that may be present in previous portions of the bitstream to reconstruct the lost information in the current I-frame.

We can assume that the location of a packet loss within a given picture can be detected. This is reasonable, since transport layer protocols such as SR-RTP [71] that support application layer framing (ALF) [14] functionality are now available. Mechanisms of this flavor allow the transport layer to detect when a certain segment of data has been lost and inform the application of these losses.

We first examine a simple algorithm that performs macroblock replacement that can be useful for reconstructing motion in low-motion scenes. For scenes with higher motion, we present an algorithm for recovering I-frames that exploits both temporal correlation and the motion vector information in the bitstream.

Macroblock Replacement

A simple approach replaces the missing macroblocks with the same blocks from a previous frame. This works well in low-motion sequences, or when the loss occurs in a uniform background region.

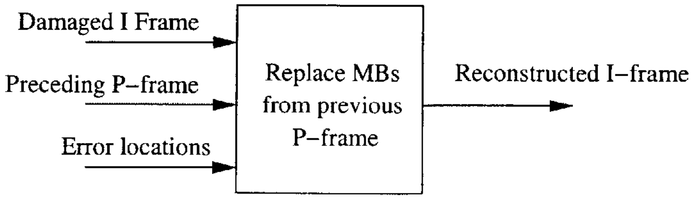

Damaged I Frame
Preceding P–frame
Error locations
Replace MBs from previous P–frame
Reconstructed I–frame

Figure 3-14: Conventional Approach. Replace missing I-frame macroblocks with the corresponding macroblocks from the preceding P-frame. This can work appropriately in scenes were the background is relatively uniform or there is low motion in the region with missing macroblocks. Replacement requires knowledge of the error locations, as well as a preceding frame that has appropriate texture.

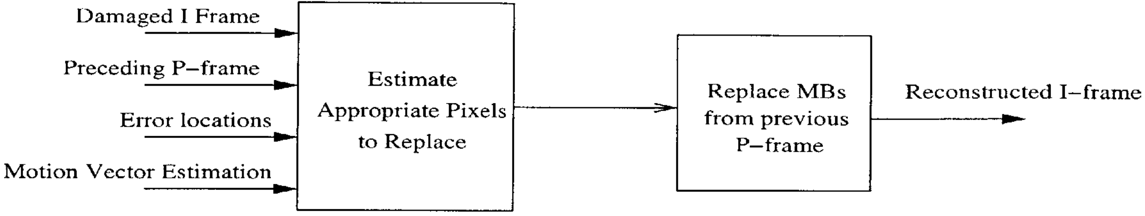

Damaged I Frame
Preceding P–frame
Error locations
Motion Vector Estimation
Estimate Appropriate Pixels to Replace
Replace MBs from previous P–frame
Reconstructed I–frame

Figure 3-15: Improved Approach. In scenes with high motion, simple replacement will not work as well, because the appearance of the missing region no longer corresponds well with the same region in previous frames. However, if texture and motion information are preserved from previous frames, we can essentially perform motion compensation on the I-frame to yield acceptable results.

However, when this is not the case, this approach does not work very well, as the corresponding blocks from a previous frame will not be as similar to the blocks that are missing. Figure 3-14 shows a block diagram of this method, and Figure 3-12 shows a conceptual illustration of the replacement method.

We have included a couple of pictures that show an example of using simple macroblock replacement to reconstruct I-frame images. In an area where no motion is predicted, the reconstruction will either work perfectly if in fact no motion has occurred, or there will be slight distortion in the image if in fact some motion has occurred in that area.

Motion-Compensated Reconstruction

In the event of high motion, simple macroblock replacement will not be acceptable, because the missing macroblock data will not correspond well with the same blocks in a previous frame. In this case, we must search for appropriate corresponding pixel values for the missing macroblocks. Fortunately, with high probability there will be motion vectors in the bitstream from which we can estimate the motion that has occurred in the region where packet loss has occurred. For example, MPEG-4 video compression uses motion vectors to estimate the B-frame(s) immediately prior to the given I-frame; motion vectors also exist from the preceding P-frame for these B-frames.

In this case, we can estimate the motion that has occurred between this P-frame (which we will use to obtain the pixel data) and the given I-frame using the available motion vector information. For our experiments, we used the motion vector from the P-frame to the B-frame for the corresponding

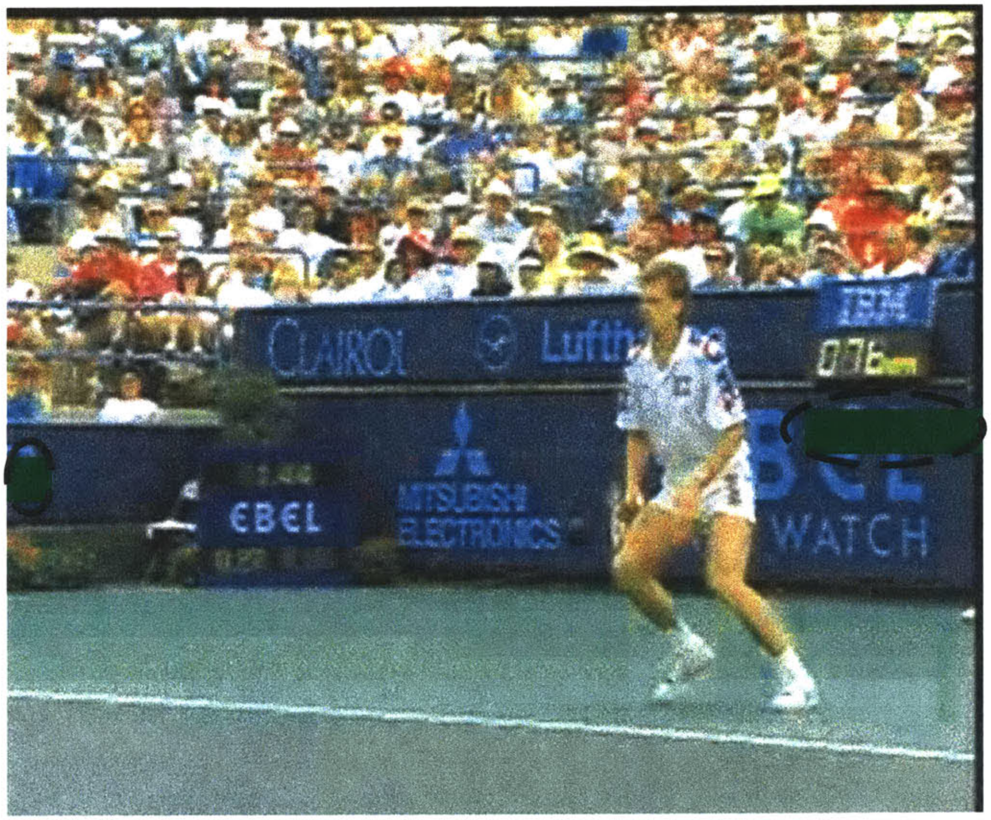


Figure 3-16: I-frame with Packet Loss. Packet loss is visible in the area above the “watch” advertisement. This error will propagate to subsequent frames if not corrected.

macroblock and simply double its value (since now we are predicting twice as far in the future). Essentially, we can think of this as essentially *performing motion compensation on the I-frame.* The conceptual illustration of this method is shown in Figure 3-13, and the corresponding block diagram is shown in Figure 3-15.

In addition to knowledge about the location of the error and texture data from the preceding P-frame, the decoder must also have access to relevant motion information; if this data is somehow lost, it can be estimated using spatial techniques, such as an averaging of motion vectors for surrounding macroblocks. Figure 3-15 assumes that motion information is not lost and we can perform temporal error concealment, using the motion vectors from P_{n-2} to B_{n-1} to estimate the motion vectors MV' for P_{n-2} to I_n. The motion vectors MV' are then used to locate the relevant texture data in P_{n-2} to use for replacement in I_n. If too much information is lost, it may be better to recover from errors via retransmission of the missing data, using the selective retransmission features of SR-RTP.

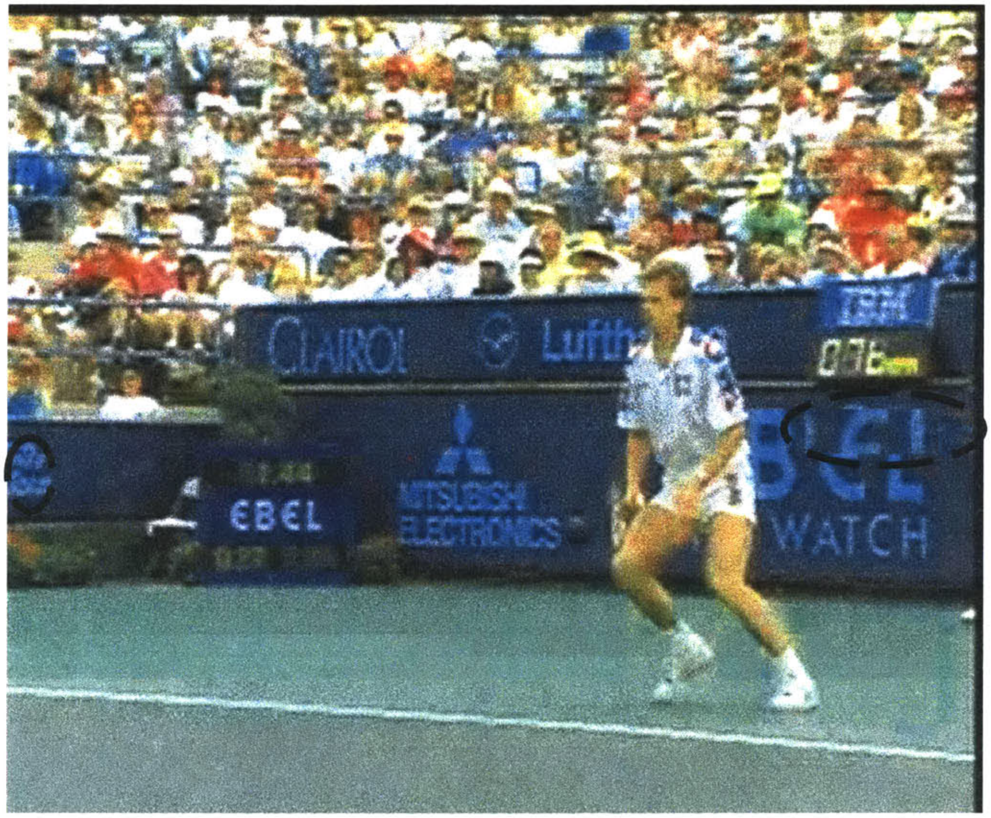


Figure 3-17: I-frame Packet Loss Recovery using Simple Replacement. While this works well for scenes with low motion, distortion is quite noticeable in the area containing the "EL" characters.

3.5.2 Results

In MPEG-4, a single packet is delimited by surrounding *resynchronization markers*; thus, each packet is independently processible and contains complete macroblocks. Since we are considering the problem of recovering from I-frame loss, the problem boils down to macroblock replacement. By subjecting the *bitstream* to random errors rather than simply removing sections of an I-frame picture, we are more likely to be simulating the types of losses a decoder would actually see over the Internet.

After subjecting various bitstreams to random uniform packet loss, we decoded these bitstreams using a modified version of the MoMuSys decoder. This gave us the video sequences as concatenated YUV data and provided us with the appropriate motion vector information (which also has losses) that we were able to convert into RGB matrices and input into Matlab to examine our postprocessing designs.

Figure 3-16 shows the original I-frame that is subjected to packet loss, resulting in several missing macroblocks in the picture. Such losses result in errors that propagate through the bitstream in the

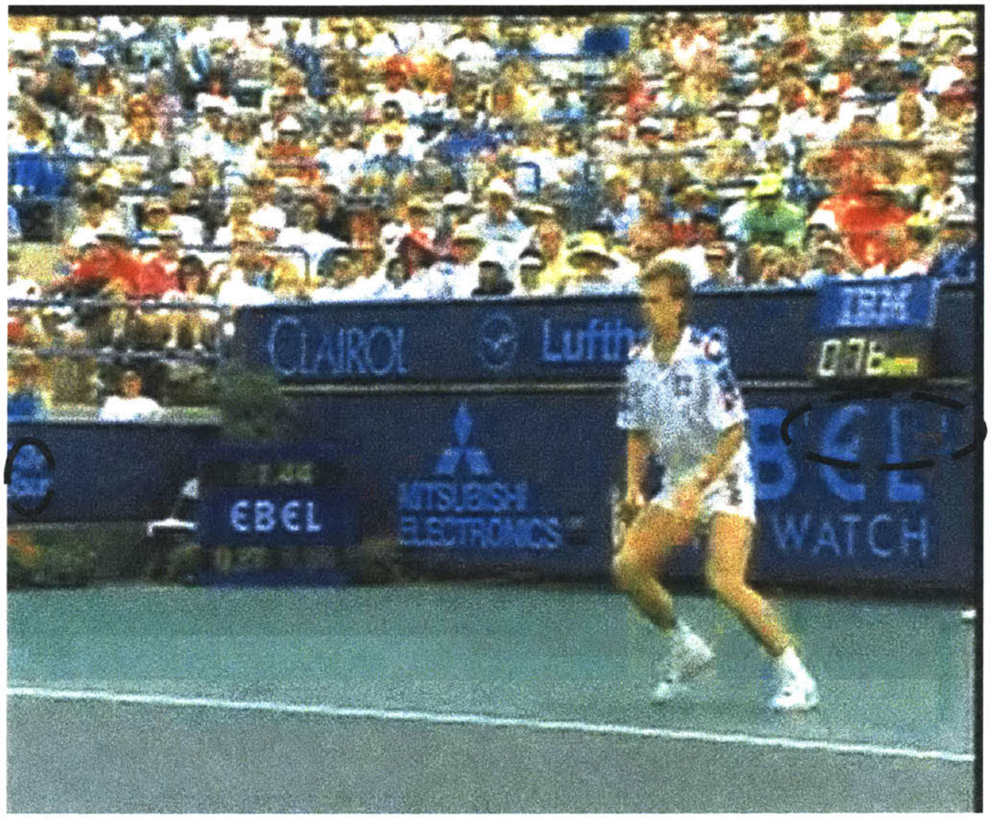


Figure 3-18: I-frame Packet Loss Recovery making use of motion information. Reconstruction of the damaged region is much more accurate.

same manner as previously described. As a result, recovering the errors in this I-frame will prevent further propagation of errors through the bitstream.

If we attempt to correct the I-frame using simple macroblock replacement using texture data from a previous P-frame, we obtain the corrected I-frame that is shown in Figure 3-17. While this image is certainly an improvement, it is a fairly inaccurate reconstruction of the image, as it does not account for motion that has occurred in the video. Inaccuracies are especially evident around the lettered area of the background behind the tennis player.

Fortunately, we can make better use of the information that is available to us in the preceding portion bitstream—by utilizing some the motion information in the preceding GOV, we can estimate the motion that has occurred between the preceding P-frame and the given I-frame. Given this motion, we can often make a more intelligent decision regarding the appropriate macroblocks to replace in the image. That is, because we can estimate the motion that has occurred between the P-frame and the I-frame, we know roughly which pixels in the P-frame correspond to the missing pixels in the I-frame and can replace the missing pixels more accurately. The result of this approach is shown in Figure 3-18; clearly, the I-frame has been reconstructed more accurately.

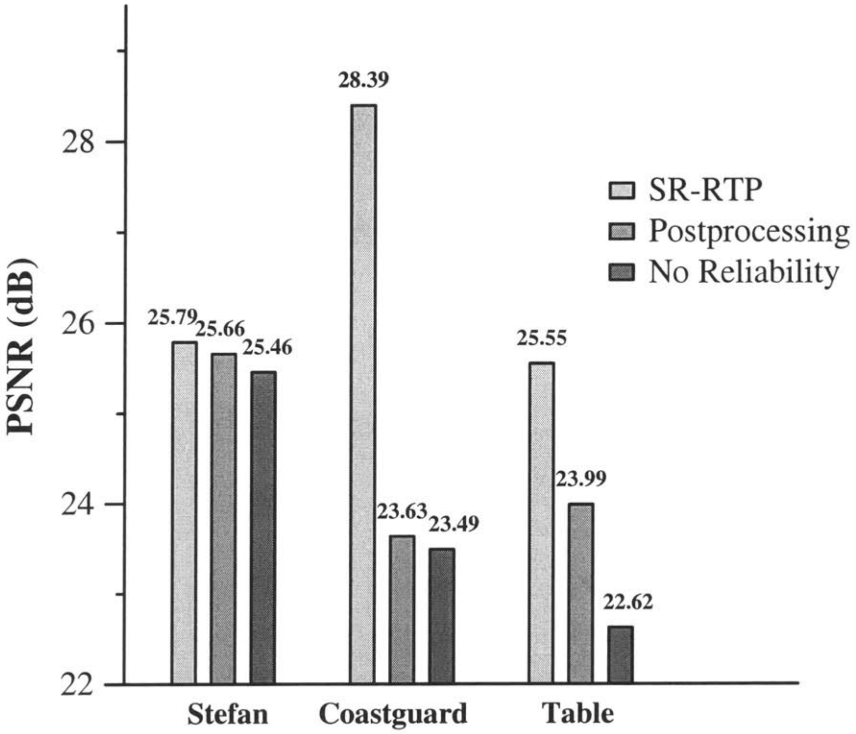


Figure 3-19: The benefits of receiver postprocessing. Given knowledge about the location of packet losses in an I-frame, which SR-RTP can provide, the receiver can perform temporal concealment on I-frames to recover from packet loss and thus alleviate propagation of errors, even if selective retransmission is not possible. This graph shows the gains that receiver postprocessing can provide for I-frames from 3 different sequences: *coastguard*, *stefan*, and *table*.

Figure 3-19 summarizes how using temporal postprocessing at the receiver to recover I-frames can result in improved image quality. We examine three distinct instances of packet loss in I-frames in three independent video sequences: *coastguard*, *stefan*, and *table* (a table tennis scene). In certain cases recovery of I-frame data can improve the PSNR of that reference frame by more than 2 dB. While perfect recovery via a scheme like selective retransmission vis SR-RTP allows for the highest possible quality at the decoder, in cases of high end-to-end latency, a scheme such as receiver postprocessing allows for reasonable I-frame recovery to take place. Even if selective retransmission is not possible, SR-RTP can provide information regarding losses to the decoder and thus aid in receiver postprocessing. Thus, receiver postprocessing can be used in combination with selective retransmission to improve the quality of important data in compressed video, thereby limiting the effects of error propagation.

3.5.3 Summary

In this section, we have proposed an alternate method for error concealment that can be used in conjunction with SR-RTP. Error concealment schemes such as the one we have presented here rely on the receiver accurate knowledge of the location of missing information, as well as the fact that the frame itself is a key frame and thus should be corrected. SR-RTP makes use of the ALF principle and thus makes it possible for the application to determine the location of loss information, as well as the fact that the missing information is of high priority and should thus be replaced.

This error concealment scheme is enabled by SR-RTP, and can be used in conjunction with selective retransmission. If the channel is experiencing particularly high loss rates, the receiver is likely better off performing error concealment than asking for retransmission, as the retransmission is likely to be lost. On the other hand, if the picture contains particularly erratic motion (non-transformational, etc.), previous texture data is missing or corrupt, motion vectors for performing reconstruction cannot be accurately estimated, or computational power on the receiver is limited, selective retransmission is preferable over postprocessing.

3.6 Summary

This chapter discussed various approaches to coping with packet loss problems presented by the Internet for streaming video. We examined the effects of propagation of errors and developed an analytical model to describe how the quality of received video is affected by packet loss. We also conducted experiments to verify the accuracy of our model and quantified the benefits of performing error recovery via selective retransmission for packet losses in MPEG-4 I-frames. To implement selective retransmission, we have designed and implemented a system to use $SR-RTP$, a backwards-compatible extension to RTP that allows for application-level framing and client-driven selective retransmission based on the priority of lost packets. Finally, we have conducted experiments to show that SR-RTP works well over emulated network conditions and shown how its semantics enable complementary error concealment schemes, such as temporal error concealment.

Turbulence is life force. It is opportunity. Let's love turbulence and use it for change.

- Ramsay Clark

Chapter 4

Bandwidth Adaptation

Internet conditions change with time; as a result, real-time media applications should tailor their transmission rate in a manner that achieves high utilization while sharing bandwidth appropriately with competing traffic (e.g., Web traffic, email, streaming media, etc.). Additionally, the quality of the received video should correspond to the available bandwidth, so that users receive the highest possible quality video for their available bandwidth.

In order to adapt to changing conditions appropriately, real-time streaming applications must:

- Adapt to transient changes in available bandwidth (*congestion control*), and
- Tailor the quality of video delivered to the available rate (*quality adaptation*).

If the receiver could afford an unlimited amount of buffering, solving the problem of bandwidth would be easy—the receiver could simply buffer up a large enough amount of data initially to sustain any subsequent bandwidth variation (several currently used applications have taken this approach [42, 56]). However, this limits the amount of interactivity that the user can have because it introduces a large amount of playback delay.

In this chapter, we discuss the qualities of a congestion control algorithm that is amenable to streaming media delivery and introduce, *binomial congestion control* algorithms, a family of TCP-friendly congestion control algorithms. We then introduce *quality adaptation*, a receiver buffering technique for layered video that allows the receiver to smooth out transient changes in bandwidth. Finally, we apply receiver-buffered quality adaptation to binomial congestion control and show that using a member of the binomial congestion control family, *SQRT*, we can reduce

- the magnitude of rate oscillations, and
- the buffering required by the receiver to play out a given number of layers of video.

Thus, we can achieve a TCP-friendly bandwidth adaptive video application that is still interactive for users.

4.1 Congestion Control

Unlike bulk file transfers, streaming video servers seek to achieve smooth playback quality, rather than simply transmit at the highest attainable bandwidth. This therefore calls for suitable mechanisms to smooth the playback rate, which would otherwise oscillate when a stream is sent using a traditional additive-increase/multiplicative-decrease (AIMD) rate control algorithm as in TCP [6, 20, 58, 61]. The process of probing for bandwidth and reacting to observed congestion induces oscillations in the achievable transmission rate, and are an integral part of the nature of many end-to-end congestion management algorithms.

Adaptation to bandwidth changes are achieved by performing congestion control – when packet loss is detected, the sender slows its transmission rate accordingly. TCP applications use additive-increase/multiplicative-decrease (AIMD) to tailor the transmission rate to available bandwidth [33]. When packet loss is detected, the sender reduces its sending rate by a factor of two. Failure to detect packet loss results increasing the transmission rate in an additive fashion.

This multiplicative decrease behavior of AIMD is detrimental to a real-time application such as a streaming video server, which aims to transmit at a constant rate. As such, alternatives that offer a smoother transmission rate have recently been proposed; these include TFRC [20], TEAR [61], and binomial controls [6].

4.2 Binomial congestion controls

Binomial congestion controls generalize TCP's increase/decrease rules in the following way:

$$\begin{aligned} &\text{I: } w_{t+R} \leftarrow w_t + \alpha / w_t^k ; \alpha > 0 \\ &\text{D: } w_{t+\delta t} \leftarrow w_t - \beta w_t^l ; 0 < \beta < 1 \end{aligned} \tag{4.1}$$

k and l are the parameters of binomial controls and w_t is the instantaneous window value, which governs the transmission rate. For $k = 0$, $l = 1$, we get AIMD used by TCP; for $k = -1, l = 1$, we get MIMD (multiplicative increase/multiplicative decrease used by *slow start* in TCP [33]); for $k = -1, l = 0$, we get MIAD; and for $k = 0, l = 0$ we get AIAD, thereby covering the class of all linear algorithms. [6] shows that if $k + l = 1$, then the flow is *TCP-friendly*. In our experiments, we use $k = 1/2$, $l = 1/2$, resulting in a *SQRT algorithm*, which achieves a much smoother transmission rate while remaining friendly to competing TCP flows.

Previous work [6] showed that a binomial congestion control satisfying the (k, l) rule , i.e. $k + l = 1$, is TCP-friendly. Further, one member of this family, SQRT ($k = l = 0.5$) appeared attractive for streaming media delivery due to its smaller magnitude of oscillations since the reduction in transmission rate is proportional only to $\sqrt{w}$, compared to AIMD where the reduction is proportional

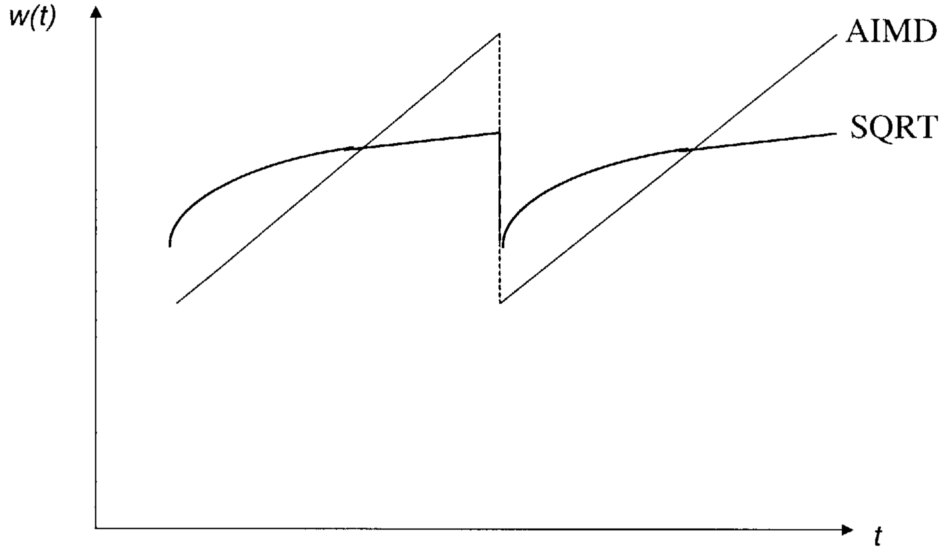


Figure 4-1: Window evolution vs. time for SQRT and AIMD congestion controls. $w(t)$ is the window value at time t. AIMD increases its window by a constant amount and backs off by a factor of 2, resulting in large oscillations. SQRT's increase and decrease rules are a function of the current window size, thus resulting in less oscillatory bandwidth probing.

to w. However, the potential benefits of such an algorithm and its impact on quality adaptation algorithms have not been studied, nor has its interaction with layered media delivery. Figure 4-1 shows the nonlinear evolution of the congestion window for the SQRT control algorithm.

The video server reports the loss and round-trip time information received from the client via RTCP receiver reports to the CM, which adjusts the congestion window and tells the application the appropriate rate at which to send data. Figure 3-9 shows the SR-RTCP header that is used to support bandwidth adaptation. Fortunately, the original RTCP receiver report already provides the necessary information to perform bandwidth adaptation. The second octet of the report block, which contains loss information, can be used to detect packet losses. The *LSR* field is used as a timestamp echo, and the *DLSR* field indicates the amount of processing time at the receiver. Thus, $RTT = \tau - LSR - DLSR$, where τ is the time at which the SR-RTCP report was received at the sender. Using this loss and round-trip time information, the server can dynamically determine the instantaneous bandwidth of the path and adapt its transmission window size.

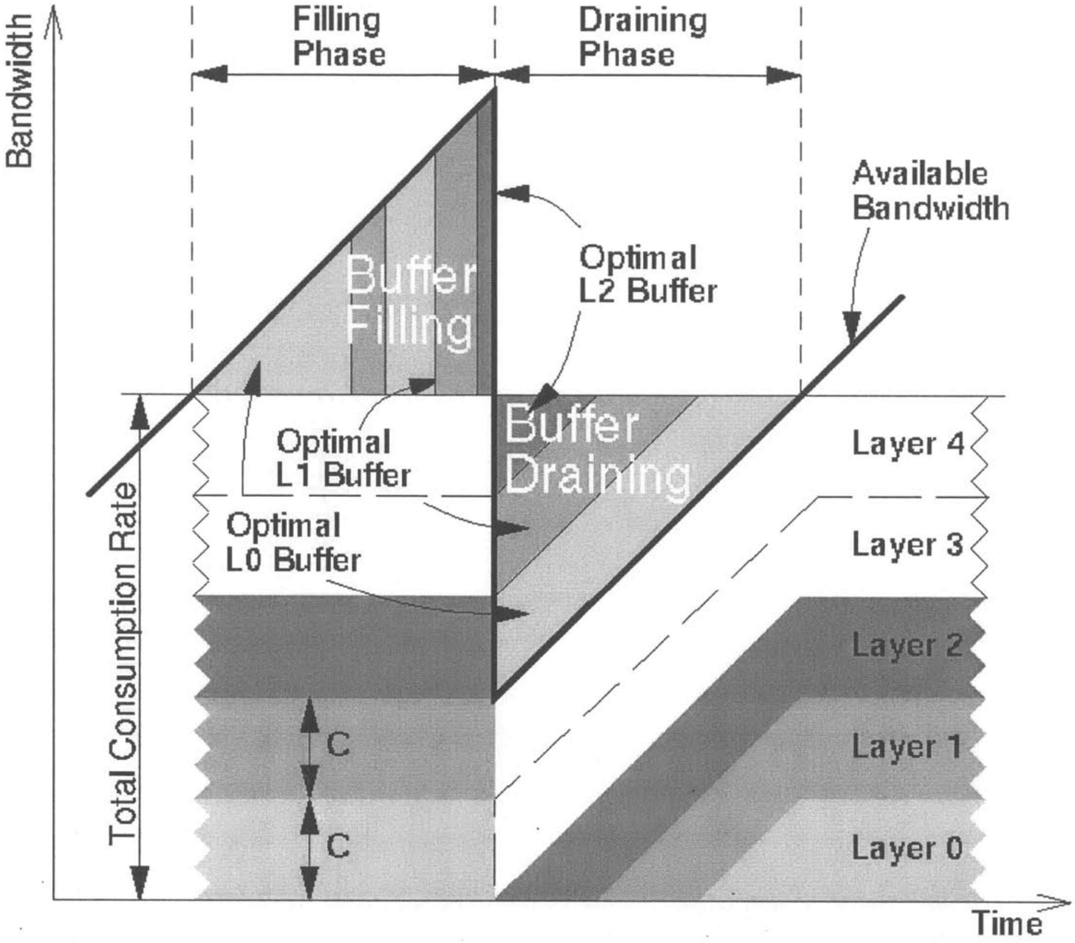

Bandwidth
Filling Phase
Draining Phase
Available Bandwidth
Optimal L2 Buffer
Buffer Filling
Optimal L1 Buffer
Optimal L0 Buffer
Buffer Draining
Total Consumption Rate
C
C
Layer 4
Layer 3
Layer 2
Layer 1
Layer 0
Time

Figure 4-2: Optimal inter-layer buffer distribution as derived in [57, 59]. Figure from [57], reprinted with permission. At any given time, no more than C bits can be played out per layer, which is the case at any point in time during the draining phase. Lower layers are favored in the actual receiver buffer (shown by the "Buffer Draining" triangle), and an decreasing portion of each layer is played from the buffer as the transmission rate increases.

4.3 Quality Adaptation

A video server should adjust the quality of the video according to the available bandwidth in an appropriate fashion. That is, assuming that a video stream can be sent at various qualities depending on the attainable bitrate, the server must make appropriate decisions regarding the quality of video to send at any given time. Rapid variations in the quality of the received video negatively affect the user's viewing experience; however, the server should also stream the highest quality video for the available bandwidth. Therefore, the server must make decisions regarding when to increase or decrease the quality of the transmitted video as bandwidth varies with time. This mechanism is called *quality adaptation*.

To appropriately determine the quality of video which should be sent, rules must be defined in order to determine when a layer should be added or removed. These rules will vary depending on the methods of quality adaptation and congestion control which are employed.

In the case of instantaneous adaptation, the layer switching rules are simple: simply send video at the highest possible layer which can be sent at any given time. But, the magnitude and frequency of oscillations in the congestion control algorithm will govern the magnitude and frequency of layer switching and hence, the user perceived quality.

To ensure low layer switching, some hysteresis may be provided, either by delaying switching to a higher quality video (unless there is some confidence that the higher quality video can be sustained) or by buffering data at the receiver for future playout. These quality adaptation algorithm however, depends on whether the data is being simulcast or available as hierarchically encoded.

Rejaie et al. describe a quality adaptation scheme for hierarchically encoded data and AIMD congestion control, where a new layer is added only when the two conditions hold [59]:

$$\begin{aligned} R &> (n_a + 1)C \\ \sum_{i=0}^{n_a - 1} buf_i &\geq \frac{((n_a + 1)C - \frac{R}{2})^2 T}{2\alpha} \end{aligned} \tag{4.2}$$

Here, R is the current transmission rate, n_a the number of currently active layers, C the bandwidth/layer, buf_i the amount of data buffered for layer i, α the rate of linear increase in bandwidth, and T the round trip time. These rules ensure that the server adds a new layer only when:

1. The instantaneous available bandwidth is greater than the consumption rate of the existing layers plus the new layer, and,
2. There is sufficient total buffering at the receiver to survive an immediate backoff and continue playing all the existing layers plus the new layer.

The layers are dropped when any of the above rules is not satisfied. Additionally, Rejaie et al. derives the optimal allocation of buffers at the receiver among various layer based on the observation that the distribution of buffering must provide maximal protection against dropping layers for any likely pattern of short-term reduction in available bandwidth[59]. Since for hierarchically encoded data, the presence of lower layer data is essential for playing out data from higher layer, this implies that:

1. Allocating more buffering for the lower layers not only improves their protection but also increases efficiency of buffering, and
2. Buffered data for each layer cannot provide more than its consumption rate (i.e., C). Thus, there is a minimum number of buffered layers needed to cope with short-term reductions

in available bandwidth for successful recovery. This minimum is directly determined by the reduction in bandwidth (or the number of backoffs, called *smoothing factor* in [59]) that we intend to absorb by buffering.

These observations enable them to derive conditions for optimal allocation of buffering for AIMD, as shown in Figure 4-2. As the figure shows, as the rate increases above the average transmission rate, we begin filling the receiver buffer with the base layer. As the rate continues to increase, the receiver begins to buffer higher layers. The intuition for when to add an additional layer can be seen from the distribution of the buffer areas in the draining phase.

At any given time, each layer can only contribute a maximum of C bits/layer to the overall bitrate (since this is the amount each layer contributes to the overall bitrate). Accordingly, the maximum amount of buffering that a receiver should buffer of any one layer is $C \cdot t$ bits, where t is the amount of time the receiver is in the draining phase. However, because the sender is also transmitting some data, the receiver does not need to completely buffer every layer because it can expect to receive some of the data in transmission. However, the buffering strategy is arranged so that, while at no point is any more than C bits/second buffered for any given layer, the buffer allocation is weighted more heavily towards lower layers.

It is easiest to understand this diagram by looking at the draining phase. The data that is buffered at the receiver from the filling phase is shown in the triangle bounded below by the transmission rate and above by the average consumption rate. By looking at the diagram during draining, one can see that, at any given time, 5 layers are being played out. Initially, the three lower layers are played out from the buffer; the buffering for layer 2 runs out first, but this is compensated by the fact that the transmission rate has increased enough for this layer to be sent from the server, so that, at any given time, we always have C bits per layer for all five layers.

4.3.1 Simulcast

In simulcast, the sender encodes video at various increasing transmission rates; in our analysis, we assume a constant rate spacing between successive layers. To ensure low variation in the quality of video stream, some hysteresis is required so that transmission is not switched to a higher quality video unless there is some confidence that the higher quality video can be sustained. This implies that a video sender should not send highest quality video based on the instantaneous transmission rate.

We assume that a packet drop due to congestion may happen at any time. Under this assumption, the transmission should be switched to a higher layer only if the higher layer can be sustained after an immediate backoff. Note that unlike hierarchical encoding, buffered data for one layer becomes useless once the layer is switched. Thus, if buffering is done to provide the hysteresis margin on backoffs, each time the layering is switched, the buffer must be built from the new layer. This results in choppiness in video playout. Thus, for an adaptive simulcast video, we assume that there are no buffers available to alleviate backoff and drop in sending rates.

Under the above assumption, a simple method to determine the quality of video data to transmit is to use the instantaneous transmission rate and send the highest layer possible assuming that a backoff happens immediately. Then, for a TCP-friendly binomial control, the video quality that should be sent should be the highest encoding available such that

$$C < R - \beta R^l$$

where C is the bit rate at which the video stream is encoded and R is the instantaneous transmission rate. For TCP-style AIMD, $\beta = 1/2$ and $l = 1$, so

$$C < R/2$$

and for SQRT:

$$C < R - \beta\sqrt{R}$$

Thus, using this simple algorithm, SQRT congestion control allows transmission of a higher layer compared to AIMD. The above derivation assumes that no buffering exists for a given quality of video at the receiver. If, on the other hand, one is willing to tolerate a certain amount of delay, one may buffer data at the receiver in order to sustain an immediate backoff, playing out video at a higher rate than the instantaneous rate.

4.3.2 Hierarchical Encoding

In the case of hierarchical encoding, more complex rules govern the quality adaptation. We derive the buffering requirements and inter-layer buffer allocation strategy for binomial congestion control algorithms.

Similar to [59], a layer should be added when the following conditions hold (assuming one backoff, i.e., *smoothing factor*[1] = 1):

$$R > (n_a + 1)C$$
$$\sum_{i=0}^{n_a - 1} buf_i \geq A$$

---

[1] The *smoothing factor* is defined in [59] as the number of immediate backoffs that can be sustained at any point in time.

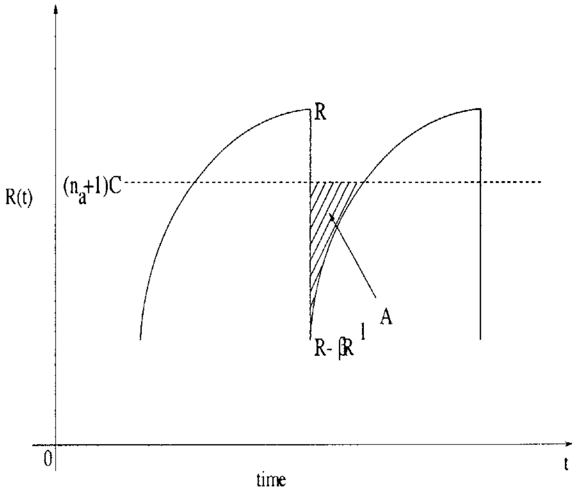


Figure 4-3: Figure showing the window evolution vs. time for binomial congestion control algorithm

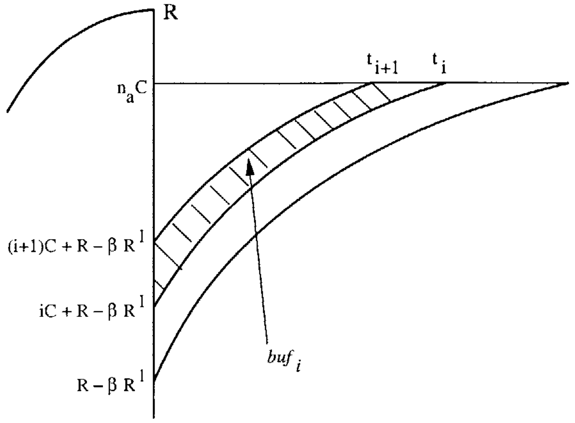


Figure 4-4: Optimal inter-layer buffer allocation for binomial congestion control.

where A is the area of the shaded portion in Figure 4-3. This area, derived in the appendix, is given by:

$$
\begin{aligned}
A = {} & \frac{(n_a+1)C[(n_a+1)^{k+1}C^{k+1} - (R-\beta R^l)^{k+1}]T}{(k+1)\alpha} - \\
& \frac{[(n_a+1)^{k+2}C^{k+2} - (R-\beta R^l)^{k+2}]T}{(k+2)\alpha}
\end{aligned}
\tag{4.3}
$$

Setting $k = 0$ for AIMD yields Equation 4.2 as derived in [59]. Since $k = 1/2$ for SQRT algorithms, we get the following conditions for adding a layer:

$$
\begin{aligned}
R &> (n_a+1)C \\
\sum_{i=0}^{n_a-1} buf_i &\geq \frac{(n_a+1)C[(n_a+1)^{3/2}C^{3/2} - (R-\beta\sqrt{R})^{3/2}]T}{(3/2)\alpha} \\
& - \frac{[(n_a+1)^{5/2}C^{5/2} - (R-\beta\sqrt{R})^{5/2}]T}{(5/2)\alpha}
\end{aligned}
\tag{4.4}
$$

Assuming that we have layers available at all encodings (i.e., a continuity assumption), it follows that $n_a + 1$ corresponds to the encoding of the video data at the mean rate of transmission. Thus,

For AIMD:

$$(n_a+1)C = \frac{3R}{4}$$

and for SQRT:

$$(n_a + 1)C < R - \frac{\beta}{2}\sqrt{R} \tag{4.5}$$

The inequality in 4.5 follows from the concavity of the window vs. time curve 4-3. Putting these values into Equations 4.5 and 4.4, we get

For AIMD:

$$\sum_{i=0}^{n_a-1} buf_i \geq \frac{(3R/4 - R/2)^2 T}{2\alpha} = O(R^2) \tag{4.6}$$

while for SQRT:

$$\sum_{i=0}^{n_a-1} buf_i \geq \frac{(R^{3/2}\beta^2 + o(R))T}{8\alpha} = O(R^{3/2}) \tag{4.7}$$

Thus, the buffer required for adding a layer with SQRT is significantly lower than the buffer requirements when AIMD is used.

Figure 4-4 shows the optimal amount of buffering required for layer i with a binomial algorithm. We can express this buffering requirement buf_i, in terms of the layer i, the current rate R, the current layer's rate n_aC, and the parameters for binomial congestion control, α, β, k, and l. This buffer allocation for a scheme which uses binomial congestion control is similar to that of the AIMD scheme [59] and is derived in Appendix B.

4.4 Results

To evaluate our bandwidth adaptation mechanisms, we conducted experiments using the testbed shown in Figure 4-5. The video server (running on a Pentium 2 Linux 2.2.9 box) streamed data to the receiver (also a Pentium 2 Linux 2.2.9) across a 1.5 Mbps link with 200 ms latency, configured using Dummynet [17]. We introduced background Web cross-traffic using the SURGE toolkit [7] to emulate the varying network conditions that an actual streaming server might see in the face of competing Web traffic on the Internet. Our test MPEG-4 video streams were encoded as independent simulcast bitstreams discrete rates between 100 Kbps and 700 Kbps, in constantly-spaced increments of 100 Kbps (i.e., 7 simulcast layers). These bitstreams were encoded with the MoMuSys MPEG-4 verification model encoder that is available to members of the MPEG group.

It is possible for MPEG-4 to be hierarchically encoded using fine-grained scalability (FGS); one method for doing this is proposed in [52]. For simplicity, however, we have used simulcast layering, where each layer is independent. Figures 4-6 and 4-7 summarize the benefits of using a binomial congestion control scheme. In particular, for simulcast video, using a binomial congestion control

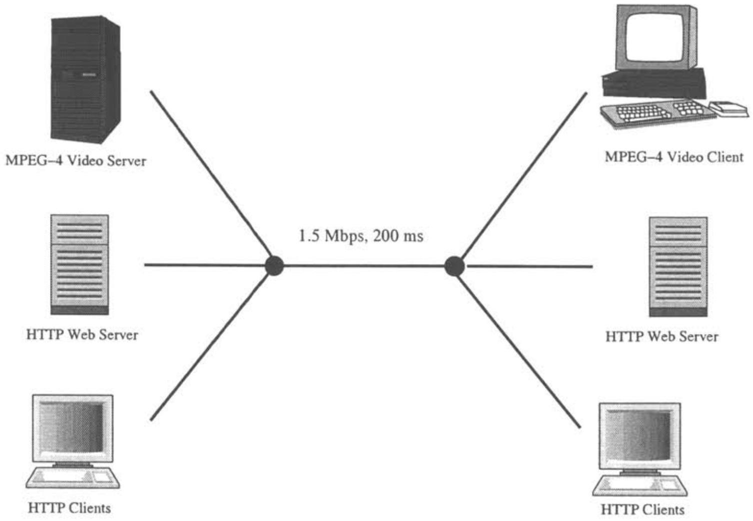

MPEG–4 Video Server
HTTP Web Server
HTTP Clients
1.5 Mbps, 200 ms
MPEG–4 Video Client
HTTP Web Server
HTTP Clients

Figure 4-5: Experimental testbed for bandwidth adaptation experiments. The video server streamed data to a client using our bandwidth adaptation techniques. To emulate actual network conditions, we generated background Web cross-traffic using the SURGE toolkit [7].

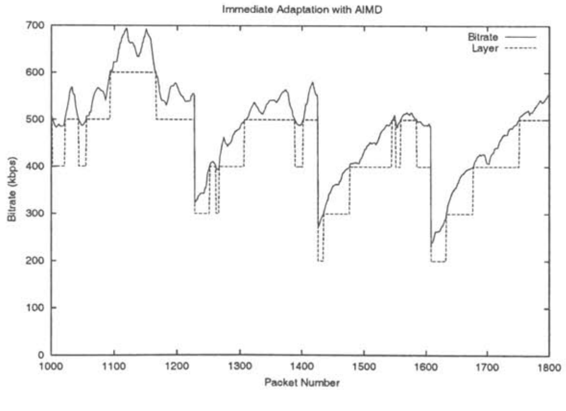


Figure 4-6: AIMD congestion control with immediate adaptation.

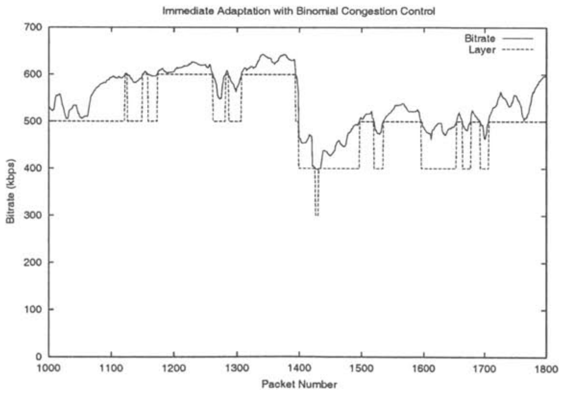


Figure 4-7: SQRT congestion control with immediate adaptation.

scheme results in smaller rate oscillations and thus less layer switching, resulting in higher perceptual quality.

We have implemented the quality adaptation rules derived for AIMD [59] as well as our own quality adaptation rules for binomial congestion control [18]. As such, our streaming architecture supports quality adaptation for video for various types of congestion control algorithms, thus providing a framework for transmitting high quality video in the face of changing network conditions.

4.4.1 Instantaneous adaptation

Figures 4-6 and 4-7 show excerpts of traces of an MPEG-4 stream transfer from the video server to the client. In this case, the sender adapts the number of layers based on the *instantaneous* available bandwidth. While this approach has the advantage of adding layers aggressively and quickly taking advantage of bandwidth as it becomes available, it results in rapid oscillations as a result of response to the sawtooth-like behavior of the congestion control algorithms.

An important point to note in these graphs is that SQRT congestion control reduces the magnitude of layer switching in response to changes in available bandwidth. The multiplicative decrease of AIMD algorithms results in drastic reduction in the quality of video sent by the sender immediately following a packet drop. SQRT congestion control on the other hand, exhibits a significantly smaller reduction in the video quality as the sending rate is much smoother.

The average number of layers dropped when a backoff occurs is noticeably smaller with SQRT, because the magnitude of the rate change upon a drop is smaller than in AIMD. Furthermore, SQRT results in a more constant rate of transmission, which reduces the amount of buffering at the receiver to reduce jitter.

Figure 4-10 shows the frequency of various drop magnitudes for two different bottleneck bandwidths, 1.5 Mbps and 2.0 Mbps. In each case, SQRT did not cause any layer drops of more than one layer. However, AIMD often resulted in many layer drops per backoff. Because small oscillations in layer adaptation (i.e., drops of one layer upon backoff) are more tolerable to the user than large variations in quality, this suggests that SQRT provides much higher perceptual quality to the user by reducing the number of abrupt layer drops. Furthermore, as available bandwidth increases (compare the 1.5 Mbps and 2.0 Mbps histograms), the rate at which the server can transmit data increases, increasing the magnitude of backoff on packet loss. This is because the backoff is proportional to R for AIMD and $\sqrt{R}$ for SQRT, where R is the transmission rate before the drop. Thus, as the available bandwidth increases, the benefit of SQRT congestion control with respect to layer dropping becomes more pronounced.

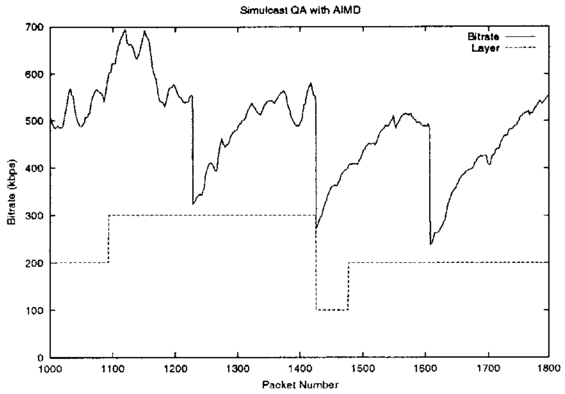


Figure 4-8: AIMD congestion control with simulcast quality adaptation.

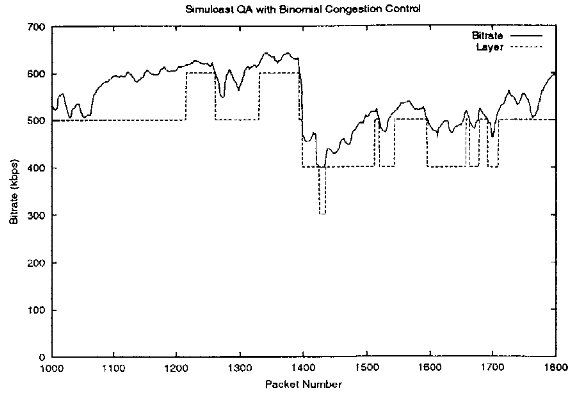


Figure 4-9: SQRT congestion control with simulcast quality adaptation.

Simulcast

Figures 4-8 and 4-9 show excerpts from a similar MPEG-4 transfer, but with the layering decisions made by the simple simulcast quality adaptation rules. That is, the video server will not add an additional layer unless it can support that layer following one immediate backoff at any given time.

Here, the benefits of SQRT congestion control are significant. SQRT results in a higher average layer sent for a given average sending rate. This is because the sender can send a higher layer and still sustain backoffs, given that the amount by which the server backs off with SQRT is considerably smaller.

Our simulcast rules drastically reduce the frequency of layer changes using AIMD, because a layer is not added until the sender is certain that it can continue to play that layer should a backoff occur at any given time. However, because the amount by which the rate changes via one backoff in AIMD is large, the sender is not able to add layers as quickly. As a result, the target bitrate of the video transmitted by the sender is much smaller than the average available rate, as shown in Figure 4-8.

Since the amount by which the sender reduces its rate upon packet loss with SQRT is smaller than with AIMD, the sender can add layers more aggressively as the rate increases, since a backoff that may ensue at any given time will not result in as drastic of a rate reduction as with AIMD. As a result, the sender is capable at sending much higher quality video under simulcast using SQRT than with AIMD. This can be seen by comparing Figures 4-8 and 4-9.

Thus, for any given rate, simulcast quality adaptation rules in conjunction with SQRT congestion control result in a higher target bitrate video being transmitted for a given transmission rate. This has important applications for streaming video. For one, when starting from slow start, the video application can send the highest quality video more quickly as the rate increases (i.e., faster convergence). Second, given a rate which the sender is capable of sending at any given time, the

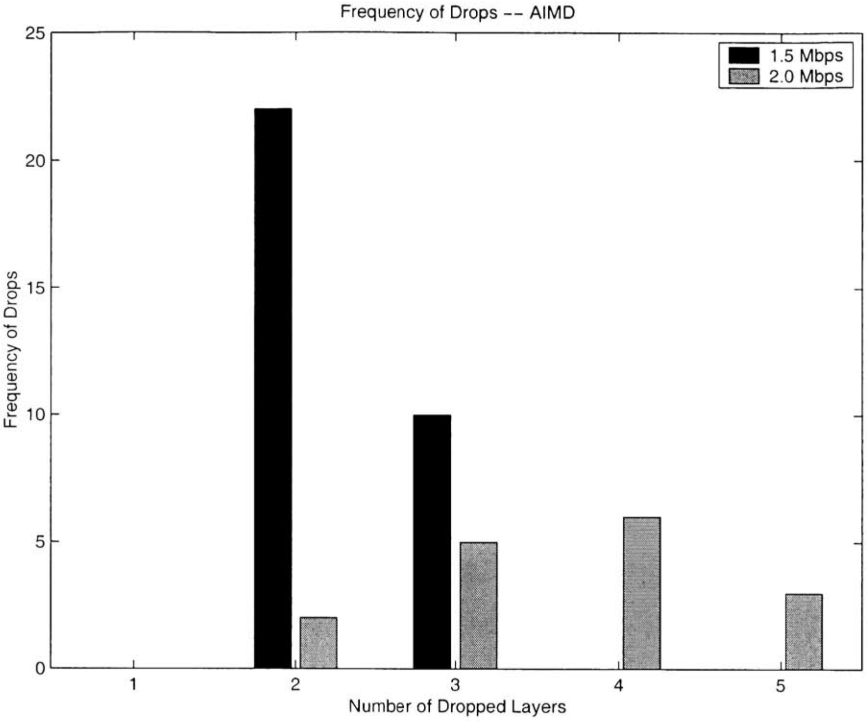


Figure 4-10: Frequency of layer drops of various magnitudes using AIMD congestion control and instantaneous rate adaptation. For SQRT congestion control, not shown in the figure, all backoffs resulted in dropping exactly one layer. The benefits of SQRT become more significant as bottleneck bandwidth increases and AIMD backoffs become larger.

server can send higher layers under SQRT congestion control, because the magnitude of backoffs is smaller, thus resulting in a higher quality of video received by the client.

Another interesting point to note by comparing Figures 4-7 and 4-9 is that the introduction of the more conservative simulcast rule did not have a considerably great effect on reducing the oscillations in layer switching. This appears to be because SQRT backoffs are often rather small, and the instantaneous channel bandwidth seen by the sender undergo transient oscillations due to RTT variations, which can change the perceived sending rate temporarily by more than one SQRT backoff. This effect is not seen in AIMD, because these rate variations are smaller than one backoff. One area of future work is in handling these transient variations appropriately in the context of simulcast quality adaptation and binomial congestion control.

4.4.2 Hierarchical encoding

With hierarchical encoding and receiver buffered quality adaptation, the oscillations resulting from sending layers using instantaneous or simulcast rules can be further reduced, as buffering built up at the receiver can be used to play out video at higher layers, even if the rate momentarily drops below the total bit-rate being supported by the layers being sent.

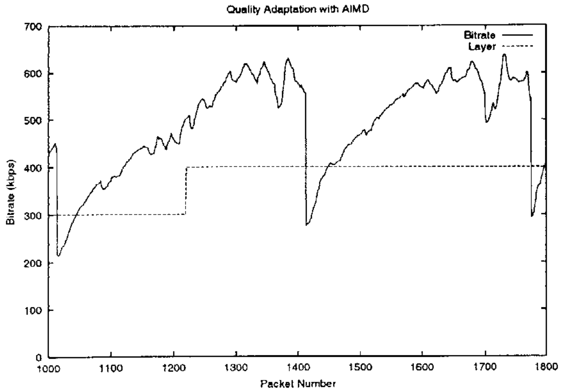


Figure 4-11: AIMD congestion control for hierarchical encoding with receiver-buffered quality adaptation.

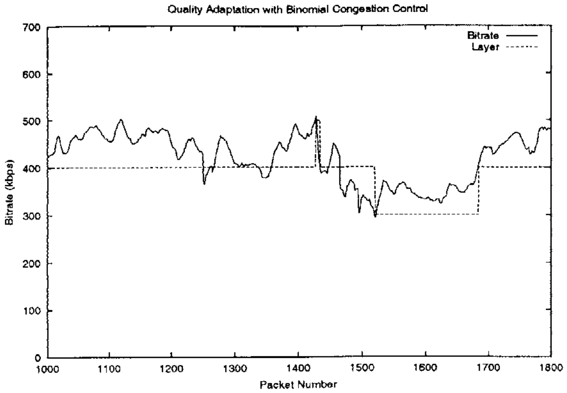


Figure 4-12: SQRT congestion control for hierarchical encoding with receiver-buffered quality adaptation.

In Section 4, we showed that, for a given rate, SQRT requires much less buffering at the receiver, thus resulting in a higher degree of interactivity and faster convergence. Figure 4-13 verifies our analytical findings. This trial shows that considerably less receiver buffering is required to add a layer, thus resulting in faster convergence to the appropriate rate and a higher degree of interactivity. As with simulcast quality adaptation, the number (but not magnitude) of oscillations seen by a system employing hierarchical encoding and SQRT is mildly higher compared to AIMD, but these oscillations are offset by higher overall quality of received video at the receiver.

Using hierarchical encoding with receiver buffering, both AIMD and SQRT congestion control algorithms result in convergence to the same transmitted layer, but AIMD is much slower in converging because a lot more buffering must be built up to sustain a backoff at any particular time. Figures 4-11 and 4-12 show layered quality adaptation for AIMD and SQRT, configured to sustain up to two immediate backoffs in transmission rate due to (unforeseen) congestion.

Thus, not only does AIMD require more buffering at the receiver than SQRT, but it also takes a longer time to play out the layers associated with the average rate. This result indicates that significantly higher interactivity is possible using SQRT congestion control. If a user wants to perform random access on a video stream (forward, rewind, etc.) to a point where no data is buffered for the video stream at any layer, using SQRT congestion control allows the video server to converge and start playing a higher quality of video much more quickly than AIMD does. Using SQRT congestion control also reduces the initial perceived latency at the beginning of a stream before a video clip can start being rendered.

Smooth quality of a received video signal depends on appropriate buffering. In particular, receiver buffering must be large enough to (1) account for network jitter (delay variation), (2) allow time for retransmission of lost packets, and (3) enable quality adaptation. The buffering required to counteract network jitter is a function of the variance in network delay, where the instantaneous

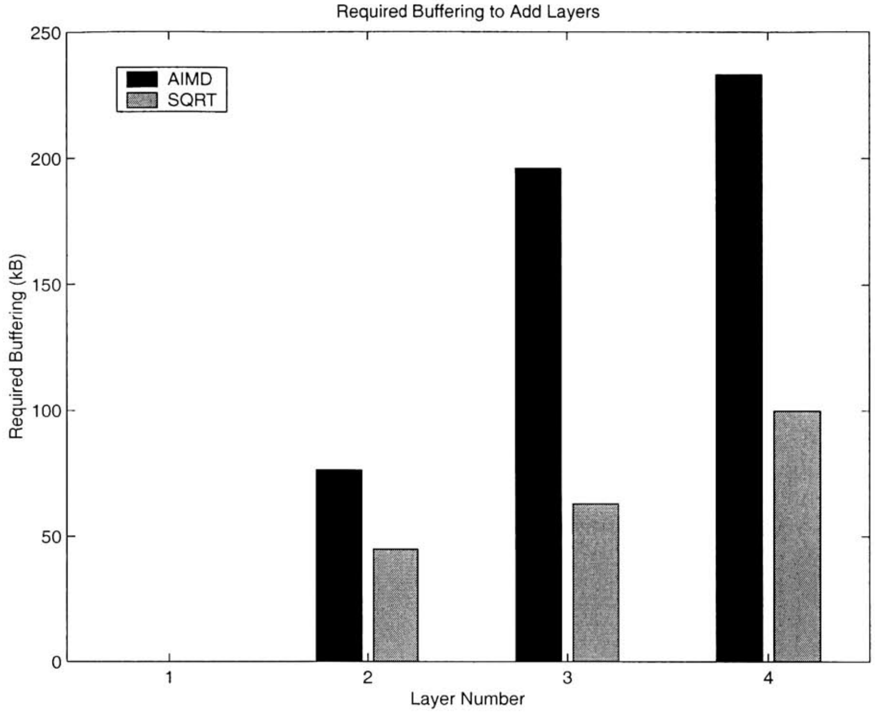


Figure 4-13: Buffering requirements in KBytes for adding layers for AIMD and SQRT algorithms. SQRT congestion control algorithms permit a client application to play video at a higher rate for a given amount of buffering.

jitter j_i can be expressed as $|(A_i - A_{i-1}) - (S_i - S_{i-1})|$ [41, 62]. Using this, the required buffering to counteract network jitter is $\beta\delta_i$, where δ_i is smoothed jitter; smaller values of β reduce overall delay, and larger values decrease the likelihood of late (hence, effectively lost) packets. Buffering for retransmission of lost packets also depends on the absolute network round-trip time. Buffering for quality adaptation depends on the absolute transmission rate. We have shown that required QA buffering is $O(R^{3/2})$ for SQRT congestion control and $O(R^2)$ for AIMD. The dominant factor for the required buffering thus depends on the relation of the absolute round-trip time to the RTT variance and the absolute transmission rate. As the absolute RTT becomes large with respect to RTT variance, buffering due to retransmission requests will dominate buffering required to counteract jitter, and vice versa.

4.5 Summary

In this chapter, we have addressed the issue of how binomial congestion control algorithms, which reduce oscillations in the transmission rate, can be used by layered streaming video applications to improve video quality and interactivity. We studied the interaction between the quality adaptation mechanisms for variable rate and hierarchically encoded data and the underlying congestion control algorithm, building on the work by Rejaie et al [59].

From our analysis and experiments, we present two main results:

- Using binomial congestion control algorithms, such as SQRT, reduces rate oscillations and thus results in less buffering and smoother playout of frames at the receiver.
- SQRT can reduce the amount of required receiver buffering to play high-quality video, thus increasing interactivity.

Besides binomial controls, several other algorithms such as TFRC and TEAR have been proposed to reduce the oscillations in the sending rate. Evaluating the benefits of these other schemes and comparing them with the binomial algorithms for streaming media delivery is a topic for future work. While congestion control for multimedia is an active topic of research, more research is needed on the streaming application algorithms to adapt to the vagaries of the network and the underlying layers to ensure better user experience for real applications. We believe that our MPEG-4 server, integrated with the Congestion Manager [3], provides an attractive platform for such work.

Human kind cannot bear very much reality.

- T.S. Eliot

Chapter 5

Implementation and Evaluation

This section describes experiences implementing selectively reliable RTP (SR-RTP), integrating this framework with the Congestion Manager (CM) [3, 4, 5], and building applications with this library.

5.1 Implementation

We have implemented the SR-RTP library, which enables selective reliability and bandwidth adaptation using CM, as well as an accompanying RTSP [63] client/server library which allows an application developer to easily layer RTSP on top of SR-RTP. Software is available from the OpenDivX project Web site [51], as well as from our project Web site [71]. As of May 3, over 5,000 people had downloaded the software libraries since its release on March 23, 2001.

The adaptive video streaming server uses the protocol stack shown in Figure 5-1. Selective reliability extensions to RTP and the corresponding protocol are implemented above the UDP layer and use CM kernel extensions to adapt the transmission rate using reports of loss rates and observed round-trip times.

5.1.1 Application Development

In this section, we describe how an application developer might extend an existing application to support SR-RTP functionality. The software release consists of the following two libraries:

- The main **SR-RTP library**, which can be used independently as a means of enabling bandwidth adaptation and selective reliability when transporting data.

- An auxiliary **RTSP library**, which uses the SR-RTP library and includes RTSP client and receiver that use the underlying SR-RTP functionality.

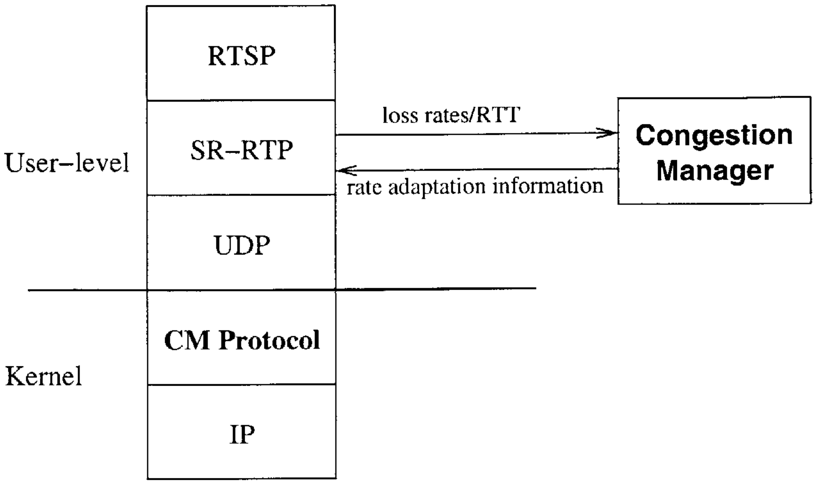

RTSP
SR–RTP
UDP
CM Protocol
IP
User–level
Kernel
loss rates/RTT
rate adaptation information
Congestion Manager

Figure 5-1: Protocol stack for implementation of selective reliable RTP with congestion management.

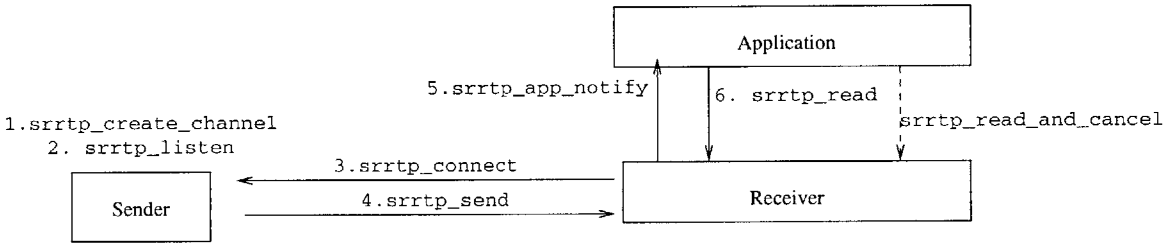

Application
5.srrtp_app_notify
6. srrtp_read
srrtp_read_and_cancel
1.srrtp_create_channel
2. srrtp_listen
3.srrtp_connect
4.srrtp_send
Sender
Receiver

Figure 5-2: Summary of SR-RTP API. The receiver's API is callback-based, such that when a complete ADU arrives, the application is notified by a call to `srrtp_app_notify()`. Upon receiving this callback, the application can read the ADU into the application memory with a call to `srrtp_read()` and being processing it. If the application needs to process a time-sensitive ADU at a given point, it can call `srrtp_read_and_cancel()`, which forces a read of the incomplete ADU and asks the transport layer to act as if that ADU had been completely received.

While the SR-RTP library can be used independently of the RTSP library, streaming applications commonly use RTSP at the application layer. Our auxiliary library thus makes it easier to incorporate RTSP/SR-RTP functionality into a video playback application that does not support streaming.

SR-RTP Library

The SR-RTP library is designed as a backwards-compatible extension to RTP [62], but has included additional functionality for permitting receiver-driven retransmission requests from the server, appropriately packetizing data by frames, and setting the appropriate priorities. The library is configurable to work with or without the Congestion Manager (CM) extensions to the Linux kernel.

If the library is configured to interoperate with CM, the library will use feedback from SR-RTP receiver reports to appropriately adjust the bandwidth and layer video quality appropriately. Otherwise, the library will support only the selective retransmission functionality, which does not require

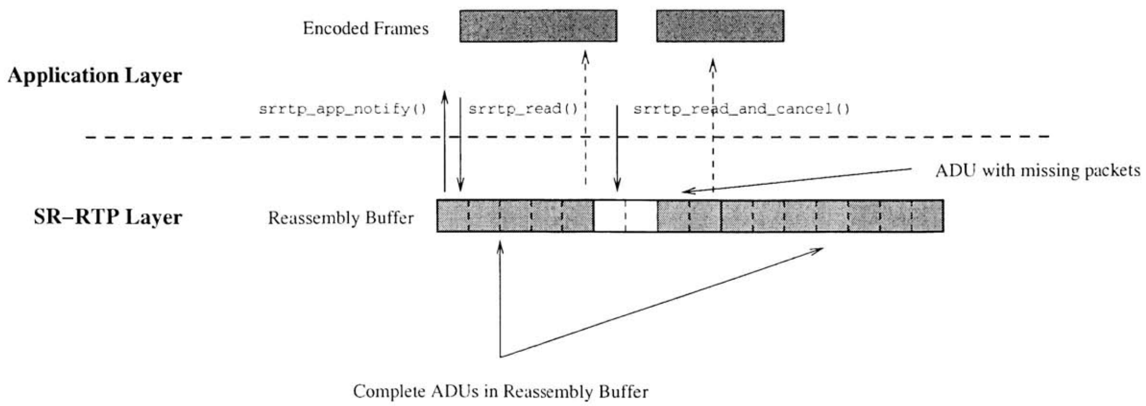


Figure 5-3: Reassembly of ADUs at the SR-RTP enabled receiver. When a complete ADU arrives, SR-RTP makes an `srrtp_app_notify()` callback to the application, which subsequently calls `srrtp_read()` to read the complete ADU into the application's memory. If playout time for an ADU occurs before a complete ADU arrives, the application makes an `srrtp_read_and_cancel() call to SR-RTP`, which cancels further retransmit requests for that ADU.

CM to operate. All functions discussed below have a corresponding function call which performs the equivalent functionality plus additional functions required to support bandwidth adaptation with CM. Figure 5-2 summarizes the calls that are described in further detail below.

The sender and receiver initialize the channel by invoking `srrtp_create_channel()`. An SR-RTP listener can bind to a listening socket by calling `srrtp_listen()`, and a client connects to this listening socket using `srrtp_connect()`. These functions establish connections on two ports, one for data, and one for the receiver feedback channel, over which SR-RTP receiver reports are sent from the receiver back to the sender to provide loss and round-trip time information.

When the sender wishes to send data to the receiver, it calls the `srrtp_send()` function, which expects a buffer of data that is independently processible by the application (i.e., and ADU). In the case of MPEG video transmission, for example, this function is called once per MPEG frame. This function subsequently fragments the frame into packets and labels each packet in a fashion that the application can understand. That is, the transport layer attaches header information such as the ADU sequence number, as well as the offset of that particular packet within an ADU, thus enabling reassembly of fragmented ADUs at the receiver, as well as the detection of lost packets. The sender can also optionally give a particular packet priority value. Typically, all packets within one frame will receive the same priority so that the priority of a missing packet can be determined from surrounding packets.

After the sender has completely sent an ADU, it keeps recently sent packets in a *retransmission buffer*. In the event that one or more packets must be retransmitted, the sender need only find the packet in its retransmission buffer and send the packet again.

As packets arrive at the receiver, they are reassembled in a *reassembly buffer*. At this point, one of two things can happen:

- the entire ADU will arrive, or
- sections of the ADU will be missing.

Figure 5-3 shows how the receiver reassembles packets into ADUs for processing by the application layer. As soon as the entire ADU arrives at the receiver, the `srrtp_app_notify()` callback is made to the application layer. The application then handles each ADU in an independent fashion. In the case of MPEG video, an `srrtp_app_notify()` callback implies the arrival of a complete MPEG video frame. At this point the application will first call the `srrtp_read()` function to read the given ADU from the reassembly buffer into the application memory, and will subsequently decode this frame and place it into a playout buffer for future display. Generally, when any application receives the `srrtp_app_notify()` callback, it will call `srrtp_read()` and subsequently perform any application-specific processing.

If sections of the ADU are missing, the receiver will send a data-driven request to the sender asking for the retransmission of lost packets. This operation is *completely transparent to the application.* The application can ask SR-RTP to cancel any further retransmission requests by calling the `srrtp_read_and_cancel()` function. This function reads the specified ADU from the retransmission buffer and informs the SR-RTP layer at the receiver that it should no longer make any retransmission requests for that particular ADU. This can be useful for controlling the retransmission of timely data.

More details of the SR-RTP library API are included in Appendix A.

Bandwidth adaptation is performed using the Congestion Manager (CM) extensions to the Linux kernel [3, 5]. A Linux 2.2.18 kernel supporting CM functionality is publicly available. Bandwidth adaptation is performed by the sender if it supports CM functionality – note that *the receiver does not need to support CM to enable bandwidth adaptation for streaming video* – all complexity is contained in the server.

The receiver periodically reports any losses that have occurred in the transmission. The receiver also echoes a timestamp for the data packet sent by the sender in the acknowledgement, so that the sender can determine the approximate round-trip time of the channel. Note that the receiver sends these receiver reports as receipt acknowledgements (ACKs), and, thus, they will be sent more than once every 5 seconds.

RTSP Library

The auxiliary RTSP library is based on the RTSP reference implementation freely available from Real Networks. We have extended its functionality to allow for integration with SR-RTP and the transmission of layered MPEG-4 video. This library is software in development that provides a streaming infrastructure to be used in conjunction with SR-RTP/CM. In particular, we are using this library to integrate a SR-RTP enabled client into XMPS [76].

The library consists of two primary C++ classes, `RTSP_Server` and `RTSP_Client`. Instantiation of either of these classes performs the appropriate initialization of the SR-RTP channel as discussed above; both the client and server are RTSP-compliant [63]. In additionally, the application layer of the server listens on the RTSP port for RTSP requests from the client. The server then processes the request accordingly, opens the appropriate MPEG-bitstream files, and begins sending the appropriate bitstream frame-by-frame to the client using the `srrtp_send()` function call. The server also has a layering callback, which is made by the SR-RTP library. This callback is transparent to the application and performs the appropriate layer switching between MPEG video layers (i.e., performs hierarchical or simulcast layering) according to the bandwidth available and the quality adaptation rules that govern layer switching.

The `RTSP_Client` class performs typical RTSP requests, but also interfaces to the SR-RTP library to support selective retransmission functionality. In particular, there is a small, fixed amount of frames that the client initially buffers before starting a playout timer; the frame rate is known by the exchange of session parameters (e.g., frame rate, spatial resolution, number of layers, etc.) using the Session Description Protocol (SDP) [25]. Once a few frames have been buffered at the receiver, playout of the video begins, and a timer event is registered to read frames out of the playout buffer at a rate corresponding to the reported frame rate. If a particular frame is not present in the playout buffer at the time it must be played out, this is likely because it has not been completely received at the SR-RTP layer. At this point, the client makes a `srrtp_read_and_cancel()` call, which forces the application processing of the incomplete frame and cancels any further retransmissions and retransmission requests for this particular ADU.

5.1.2 Implementation Details

Integrating selective retransmission with the application is a tricky implementation task, but our system exports a simple API to make this integration relatively seamless. While decisions about whether a retransmission request should be made are based on the *priority* of surrounding fragments, the client application must also communicate with SR-RTP to inform the transport protocol about when various retransmissions are no longer necessary. For example, if a frame has already been played, no further retransmission requests should be made for packets belonging to that frame; this process is called *cancellation*.

SR-RTP

To facilitate application level framing and cancellation, our system makes use of a callback-based API. When an ADU (i.e., frame) has been received in its entirety, SR-RTP makes a callback to the application, using `srrtp_app_notify(adu_seqno, length)`. The application then reads the frame

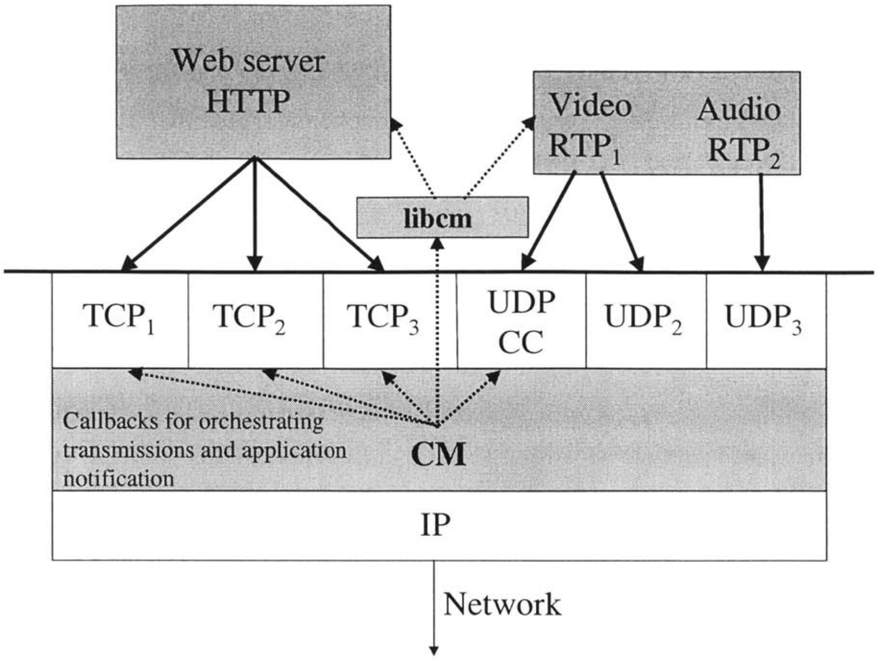


Figure 5-4: Congestion Manager (CM) Architecture. The CM is implemented as an extension to the Linux kernel and allows for application-independent congestion control. Applications interface to the CM using the functions exported by `libcm`. UDP-CC is a congestion-controlled UDP socket implemented using CM. Figure from [3], reprinted with permission.

into the playout buffer. When 200 ms worth of frames (the initial amount of data that can be buffered without affecting interactivity) has been buffered, the application begins reading frames from the playout buffer for decoding. As each frame is read from the playout buffer, the application calls `srrtp_update(adu_seqno)`, which tells the transport layer to stop requesting retransmission of packets that belong to frames that precede this frame (and also to ignore late-arriving packets from these frames).

Congestion Manager

An overview of the Congestion Manager (CM) architecture is shown in Figure 5-4. CM sits in between the transport layer and the network layer and allows applications to utilize congestion control, independent of the application itself or the transport layer that the application uses. CM enforces congestion control policies using callback mechanisms, depicted in Figure 5-4 with dotted lines. CM uses a mechanism called a *congestion controller* that performs window-based congestion avoidance functionality in a manner similar to AIMD's window-based approach. User-space clients (including SR-RTP, which sits in user space) use `libcm` as an interface to the CM kernel functionality. The Congestion Manager software is available from the project's Web page [15].

CM provides application-independent congestion control functionality to applications by providing a few additional system calls. All of the CM functionality is enabled in an application-transparent manner by the SR-RTP library. When the application wishes to initialize a connection, the SR-RTP layer calls `cm_open()` to specify the connection to which congestion management should be applied. CM can be used in a number of ways, but the most effective way seems to be to the the "Application Level Framing" mode. When an application uses this mode of operation, it calls `cm_register()` to register the appropriate application level callback function; when it wants to send data, it makes a call to `cm_request()`. CM then makes the appropriate application callback when it is permitted to send data. This allows CM to control the rate at which the application sends data. When the application receives feedback data from the receiver regarding loss and round-trip time information, it makes a call to `cm_update()`, which informs CM about channel characteristics; using this information, CM can appropriately adjust the rate at which it gives the application permission to send data. The SR-RTP layer makes all calls to CM and controls bandwidth adaptation.

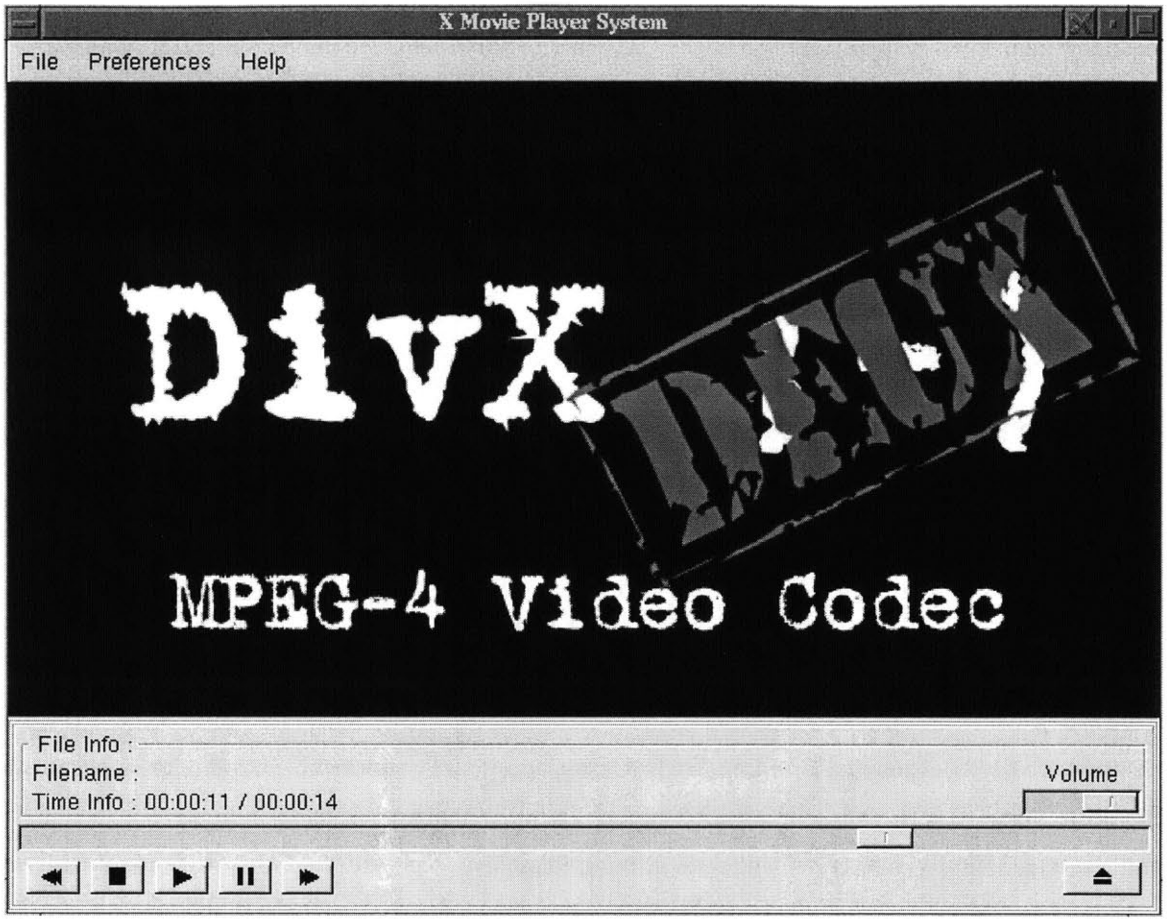
In the future, routers will likely enable congestion detection with mechanisms other than packet loss. The DECbit scheme allows routers to compute the average queue size and set an "Explicit Congestion Notification" (ECN) bit in the packet header when the queue length exceeds a certain threshold [54]. Additionally, Random Early Detection (RED) gateways have a similar mechanism for using queue length characteristics to probabilistically mark packets to indicate congestion to endpoints [21]. Our video server can adapt to congestion using these alternate congestion notification signals. The CM interface supports performing congestion control in response to explicit congestion notification (ECN) [19, 53]. When the server receives a packet with the ECN bit set, the SR-RTP layer reports this to the CM using `cm_update()`, at which point the CM adjusts the server's rate accordingly.

Quality Adaptation

The implementation of receiver-buffered quality adaptation is derived from the *ns* (network simulator) [46] implementation of QA for AIMD. The quality adaptation module receives information regarding the round-trip time of the channel, and keeps track of the number of layers it is sending, as well as the total number of bytes it has sent. Using this information, as well as the instantaneous available rate, the sender estimates the total amount of data that the receiver has buffered and makes appropriate decisions, based on the congestion control algorithm being used (e.g., AIMD, SQRT, etc.) as to whether enough data is buffered at the receiver for another layer to be added.

5.2 Sample Application

In order to demonstrate the usefulness of selective reliability and bandwidth adaptation in a video streaming system, as well as the ease of integrating this functionality into existing video playback



File Info :
Filename :
Time Info : 00:00:11 / 00:00:14

Figure 5-5: Sample Application. XMPS can be extended to support the RTSP media type and SR-RTP for transport using the SR-RTP library.

applications.

Several MPEG-4 applications are in development for Linux and Windows, and are available as part of the OpenDivX project [51]. The X Movie Player System [76], available for Linux, can be extended with a DivX plug-in to support the playout of MPEG-4 video. However, currently neither the Linux nor the Windows implementation supports the playout of streaming video. We are currently extending both the Linux and the Windows versions to support selective reliability and bandwidth adaptation as described in this thesis. We expect to release the streaming-enabled Windows and Linux players within three months.

The most absurd and reckless aspirations have sometimes led to extraordinary success.

- Vauvenargues

Chapter 6

Conclusion

In order for video streaming to succeed on the Internet, streaming systems must account for the anomalies of packet loss and changes in bandwidth and delay that make the delivery of real-time video on the Internet challenging. We analyzed the effects of packet loss on the quality of MPEG-4 video and proposed a model to explain these effects. We showed that, by recovery of only the most important data in the bitstream, significant performance gains can be achieved without much additional penalty in terms of latency. Finally, we have used these findings to design a system that:

- Employs backwards compatible extensions to RTP to enable selective retransmission of important data, incorporates postprocessing techniques, and
- Integrates the server with the Congestion Manager to enable the application to perform TCP-friendly congestion control that is more amenable to the transmission of video.

Through the combination of these techniques, our streaming system adapts to changing network conditions and delivers high-quality, interactive video to Internet clients.

Nevertheless, many open questions remain before streaming video can truly become as pervasive as television. One of these questions involves how to schedule packets appropriately for retransmission. Given that packets have been lost, likely due to congestion, certain packets, such as I-frame packets, may need to be retransmitted to control the propagation of errors. However, one of the tradeoffs associated with the retransmission of lost packets is that this consumes bandwidth that might otherwise be used to send future packets in the video sequence. Given bursty packet loss, a server might decide to schedule retransmissions of lost packets in a number of different ways.

It is likely also that the layered video techniques proposed in this thesis could also be implemented over a differentiated services network. Transmission of video over such a network has been examined recently [70], how to perform bandwidth adaptation and schedule packet retransmissions over such a network remains an open question.

The Internet holds great promise as a medium for streaming video. Video streaming will transform the way we use the Internet by expanding the range of applications that people use the Internet for. However, using the Internet for purposes that it was not originally designed, such as streaming multimedia applications, presents serious challenges. In this thesis, we addressed the challenges that bandwidth and delay variation and packet loss present to streaming video applications and modeled their effects, design and implemented a system to counteract these effects, and integrated our system with existing video playback applications to demonstrate the practicality of our solution. While many questions and challenges still remain, we have addressed some difficult barriers to making high-quality video streaming over the Internet a reality.

Appendix A

SR-RTP API Documentation

SR-RTP is made usable by applications via an an application programming interface (henceforth called the "SR-RTP API", or simply the "API"). To use the SR-RTP API, applications need to include `rtp-api.h`. Below we describe the syntax and semantics of the SR-RTP API commands in its alpha release.

The important API calls are documented below. More thorough documentation is being written, but this should provide all of the necessary information to construct a working SR-RTP application. Many of these function calls have a complementary function that should be called when the Congestion Manager is enabled.

A.1 Initialization Functions

- `rtp_channel_t srrtp_create_channel() rtp_channel_t srrtp_create_cm_channel(int cm_socktype, int cm_congctlalg)`

  srrtp_create_channel creates an SR-RTP communications channel. This is the API call that initializes the channel. For use with the Congestion Manager (CM), srrtp_create_cm_channel provides a means of opening an SR-RTP channel where cm_socktype specifies the type of CM socket to use (i.e., buffered, rate-controlled, or ALF).

- `int srrtp_open(rtp_channel_t ch, u_int32_t saddr, u_short sport, u_short scport, int *rtpsock, int *rtcpsock) int srrtp_cm_open(rtp_channel_t ch, u_int32_t saddr, u_short sport, u_short scport, int *rtpsock, int *rtcpsock)`

  srrtp_open initializes a connection to a listening SR-RTP socket.

  Parameters:

  - ch: instance of the SR-RTP channel

  - saddr: INET address of the machine to connect to
  - sport: port on the machine that is listening. The control (RTCP) port will be automatically set to one higher than this number.
  - rtpsock: returns the file descriptor of the data socket
  - rtcpsock: returns the file descriptor of the control socket

- `int srrtp_listen(rtp_channel_t ch, u_short rport, u_short rcport, int *rtpsock, int *rtcpsock) int srrtp_cm_listen(rtp_channel_t ch, u_short rport, u_short rcport, int *rtpsock, int *rtcpsock)`

  srrtp_open initializes an SR-RTP socket to listen for incoming connections.

  Parameters:

  - ch: instance of the SR-RTP channel
  - rport: port on which to listen for data
  - rcport: port on which to listen for control data
  - rtpsock: returns the file descriptor of the data socket
  - rtcpsock: returns the file descriptor of the control socket

A.2 Data Path Functions

- `void srrtp_read(rtp_channel_t ch, int seqno, char *data, int *len, u_int32_t *fromaddr, u_int16_t *fromport) void srrtp_read_and_cancel(rtp_channel_t ch, int seqno, char *data, int *len, u_int32_t *fromaddr, u_int16_t *fromport)`

  This function is called by the application in order to read an ADU from the reassembly buffers into

  Parameters:

  - ch: instance of the SR-RTP channel
  - seqno: ADU sequence number (monotonically increasing)
  - data: buffer to hold received data
  - len: reported length of received data
  - fromaddr: reported origin address
  - fromport: reported origin port

- `void srrtp_send(rtp_channel_t ch, char *adu, int len, u_int8_t more, u_int16_t priority)`

  This function sends an ADU on the SR-RTP data channel. All packet fragmentation, etc., is performed invisibly to the application.

  Parameters:

  - ch: instance of the SR-RTP channel
  - adu: buffer holding ADU to be sent
  - len: length of buffer
  - more: are there more ADUs to send?
  - priority: priority field (set MSB if retransmit is necessary)

- `void srrtp_app_notify(rtp_channel_t ch, int seqno, int len, int layer)`

  An SR-RTP application must implement this callback. The SR-RTP layer makes this callback when a complete ADU arrives. The application is notified when this event occurs, and responds appropriately, usually by calling srrtp_read.

- `void srrtp_app_send(rtp_channel_t ch)`

  This is used with the CM if a send callback is made from the CM. It is a way for the application to send ADUs via callback mechanisms.

Appendix B

Quality adaptation

B.1 Buffering requirements

For binomial controls, the increase in sending rate (R) per round trip (T) is governed by the following equation ([6]):

$$\begin{aligned}
dR/dt &= \frac{\alpha}{R^kT} \\
A_1 &= \int_{t_1}^{t_2} Rdt \\
&= \int_{R-\beta R^l}^{(n_a+1)C} \frac{R^{k+2}T}{(k+1)\alpha}dt \\
&= \frac{[(n_a+1)^{k+2}C^{k+2}]T}{\alpha(k+2)} \\
A &= (n_a+1)C(t_2-t_1) - A_1 && \text{(B.1)} \\
t_2 - t_1 &= \frac{[(n_a+1)^{k+1}C^{k+1} - R^{k+1}]}{(k+1)\alpha} \\
\Rightarrow A &= \frac{(n_a+1)C[(n_a+1)^{k+1}C^{k+1} - R^{k+1}]}{(k+1)\alpha} - \frac{(n_a+1)^{k+2}C^{k+2} - R^{k+2}]}{(k+2)\alpha} && \text{(B.2)}
\end{aligned}$$

B.2 Inter-layer buffer allocation

For binomial controls, the optimal inter-layer buffer allocation is determined by the following equation:

$$buf_i = [(n_aCt_i - N_i) - (n_aCt_{i+1} - N_{i+1})] \qquad \text{(B.3)}$$

where t_i is the amount of time taken for the rate to increase from $(iC + (R - \beta R^l))$ to $n_a C$ along the curve $R_i(t)$. N_i is the area under the curve $R_i(t)$ from the time period 0 to t_i. The buffer allocation for layer i, buf_i is shown by the shaded area in Figure 4-4. Thus from the equation relating the evolution of rate over time,

$$\begin{aligned} \frac{dR}{dt} &= \frac{\alpha}{R^k} \\ R(t) &= \left[(k+1)\alpha t + R_0^{k+1}\right]^{\frac{1}{k+1}} \end{aligned}$$

it is possible to determine an expression for t_i, the time taken for the instantaneous rate R to increase to $n_a C$ for a generic binomial increase by α / R^k.

$$t_i = \frac{1}{\alpha(k+1)} \left[(n_a C)^{k+1} - (iC + (R - \beta R^l))^{k+1}\right] \tag{B.4}$$

The curve $R_i(t)$ is defined by the curve which originates at the rate corresponding to i layers above the backoff, i.e., $(iC + (R - \beta R^l))$ and increases in a binomial fashion. The area under the rate curve $R_i(t)$, N_i, can be expressed as:

$$\begin{aligned} N_i &= \int_0^{t_i} R_i(t) dt \\ &= \int_0^{t_i} \left[(k+1)\alpha t + (iC + (R - \beta R^l))^{k+1}\right]^{\frac{1}{k+1}} \\ &= \frac{1}{\alpha(k+2)} \left[(n_a C)^{k+2} - (iC + (R - \beta R^l))^{k+2}\right] \end{aligned} \tag{B.5}$$

There are two scenarios considered in [58] corresponding to γ immediate backoffs (smoothing factor) and the backoffs uniformly separated. These are the worst cases possible and thus, give an upper bound on the buffer requirements for each layer.

B.2.1 Scenario 1

We can now substitute these results into equation B.3 and generalize for γ successive backoffs for Scenario 1 to obtain

$$\begin{aligned} buf_i = & \frac{n_a C}{\alpha(k+1)}[((i+1)C + (R - \gamma\beta R^l))^{k+1} - (iC + (R - \gamma\beta R^l))^{k+1}] \\ & + \frac{1}{\alpha(k+2)}[(iC + (R - \gamma\beta R^l))^{k+2} - ((i+1)C + (R - \gamma\beta R^l))^{k+2}] \end{aligned} \tag{B.6}$$

There are two important notes with respect to this derivation. The first is that analytically we have shown an upper bound for the buffering required for γ immediate successive backoffs, in reality, the quantity of each successive backoff would be smaller because the backoff is reducing from the already smaller rate. In our implementation, we have calculated exact buffering requirements, but this result cannot be shown in a clean analytically closed form.

Second, it should be noted that the implementation calculates the buffer requirements for transmitting N layers by simply summing the optimal inter-layer buffer allocation requirements for each layer. This strategy was adopted because this was the approach taken in the *ns* simulation code for receiver buffered quality adaptation in Rejaie et al.'s work [58].

B.2.2 Scenario 2

In Scenario 2, the γ backoffs are divided into γ_i initial immediate backoffs (as in Scenario 1), followed by $\gamma - \gamma_i$ successive backoffs to $n_aC - \beta(n_aC)^l$.

The total amount of buffering required for Scenario 2 is the amount of buffering required in a Scenario 1 situation with γ_i backoffs (Equation B.5) (denoted by buf_{i1}), plus the amount of buffering required to sustain $(\gamma - \gamma_i)$ successive backoffs from n_aC to $n_aC - \beta(n_aC)^l$ (denoted by buf_{i2}). Thus,

$$\begin{aligned}
buf_i &= buf_{i1} + (\gamma - \gamma_i)buf_{i2} \\
buf_{i1} &= \frac{n_aC}{\alpha(k+1)}[((i+1)C + (R - \gamma_i\beta R^l))^{k+1} - (iC + (R - \gamma_i\beta R^l))^{k+1}] \\
&\quad + \frac{1}{\alpha(k+2)}[(iC + (R - \gamma_i\beta R^l))^{k+1} - ((i+1)C + (R - \gamma_i\beta R^l))^{k+1}] \qquad \text{(B.7)} \\
buf_{i2} &= \frac{n_aC}{\alpha(k+1)}[(n_aC)^{k+1} - (n_aC - \beta(n_aC)^l))^{k+1}] \\
&\quad + \frac{1}{\alpha(k+2)}[(n_aC)^{k+2} - (R - \beta(n_aC)^l))^{k+2}] \qquad \text{(B.8)}
\end{aligned}$$

Bibliography

[1] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. In *Proc. 35th Ann. IEEE Symp. on Foundations of Computer Science*, pages 604–612, November 1994.

[2] M. Allman and V. Paxson. *TCP Congestion Control.* Internet Engineering Task Force, April 1999. RFC 2581.

[3] D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in Internet applications. In *Proc. Symposium on Operating Systems Design and Implementation*, October 2000.

[4] H. Balakrishnan, H. S. Rahul, and S. Seshan. An Integrated Congestion Management Architecture for Internet Hosts. In *Proc. ACM SIGCOMM*, pages 175–187, Cambridge, MA, September 1999.

[5] H. Balakrishnan and S. Seshan. *The Congestion Manager.* Internet Engineering Task Force, Nov 2000. Internet Draft draft-balakrishnan-cm-03.txt (`http://www.ietf.org/internet-drafts/draft-balakrishnan-cm-03.txt`). Work in progress, expires May 2001.

[6] D. Bansal and H. Balakrishnan. Binomial Congestion Control Algorithms. In *Proceedings INFOCOM 2001*, April 2001. Also available as MIT-LCS-TR-806 from `http://nms.lcs.mit.edu/papers/cm-binomial.html`.

[7] P. Barford. SURGE – Scalable URL Reference Generator. `http://www.cs.bu.edu/students/grads/barford/Home.html`, 1998.

[8] J.C Bolot and T. Turletti. Adaptive error control for packet video in the internet. In *Proceedings of International Conference on Internet Protocols*, September 1996.

[9] J.C Bolot and T. Turletti. Experience with control mechanisms for packet video in the internet. *SIGCOMM Computer Communication Review*, 28(1), January 1998.

[10] P. Brady. Effects of transmission delay on conversational behavior on echo-free telephone circuits. Technical report, Bell Laboratories, January 1971.

[11] C. Burmeister, R. Hakenberg, J. Rey, A. Miyazaki, and K. Hata. Multiple selective retransmissions in RTP. In *11th International Packet Video Workshop*, Kyongju, Korea, May 2001.

[12] L. Chiariglione. MPEG-4 FAQs. Technical report, ISO/IEQ JTC1/SC29/WG11, July 1997.

[13] M. Civanlar, G. Cash, and B. Haskell. *RTP Payload Format for Bundled MPEG.* Internet Engineering Task Force, May 1998. RFC 2343.

[14] D. Clark and D. Tennenhouse. Architectural Consideration for a New Generation of Protocols. In *Proc. ACM SIGCOMM*, pages 200–208, Philadelphia, PA, September 1990.

[15] Congestion Manager Project Web Page. `http://nms.lcs.mit.edu/projects/cm/`, 2001.

[16] Divx networks. `http://www.divxnetworks.com/`, 2001.

[17] Dummynet. `http://www.iet.unipi.it/~luigi/ip_dummynet`, September 1998.

[18] N. Feamster, D. Bansal, and H. Balakrishnan. On the interactions between congestion control and layered quality adaptation for streaming video. In *11th International Packet Video Workshop*, Kyongju, Korea, May 2001.

[19] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Comm. Review*, 24(5), October 1994.

[20] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-Based Congestion Control for Unicast Applications. In *Proc. ACM SIGCOMM '00*, pages 43–54, Stockholm, Sweden, September 2000.

[21] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4), August 1993.

[22] International Organization for Standardization. *Overview of the MPEG-4 Standard*, December 1999.

[23] M. Ghanbari. Cell-loss concealment in atm video codecs. *IEEE Trans. CAS for Video Tech.*, 3(3):238–247, June 1993.

[24] S. Gringeri, R. Egorov, K. Shuaib, A. Lewis, and B. Basch. Robust compression and transmission of mpeg-4 video. In *Proc. ACM Multimedia*, 1999.

[25] M. Handley and V. Jacobson. *Session Description Protocol.* Internet Engineering Task Force, April 1998. RFC 2327.

[26] P. Haskell and D. Messerschmitt. Resynchronization of motion compensated video affected by ATM cell loss. In *Proc. ICASSP '92*, San Francisco, CA, March 1992.

[27] W. Heinzelman. *Application-Specific Protocol Architectures for Wireless Networks.* PhD thesis, Massachusetts Institute of Technology, June 2000.

[28] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-Efficient Communication Protocols for Wireless Microsensor Networks. In *Proc. Hawaaian Int'l Conf. on Systems Science*, January 2000.

[29] S.S. Hemami and T. H.-Y. Meng. Transform coded image reconstruction exploiting interblock correlation. *IEEE Transactions on Image Processing*, 4(7):1023–1027, July 1995.

[30] D. Hoffman, G. Fernando, and V. Goyal. *RTP Payload Format for MPEG1/MPEG2 Video.* Internet Engineering Task Force, January 1998. RFC 2250.

[31] International Organization for Standardisation, Atlantic City. *Information Technology - Generic Coding of Audio-Visual Objects, Part 2: Visual*, October 1998. ISO/IEC JTC 1/SC 29/WG 11 N 2502 FDIS 14496-2.

[32] S. Jacobs and A. Eleftheriadis. Providing Video Services over Networks without Quality of Service Guarantees. In *WWW Cons. Workshop on Real-Time Multimedia and the Web*, 1996.

[33] V. Jacobson. Congestion Avoidance and Control. In *Proc. ACM SIGCOMM*, pages 314–329, August 1988.

[34] L. H. Kieu and K. N. Ngan. Cell-loss concealment techniques for layered video codecs in an atm network. *IEEE Transactions on Image Processing*, 3(5):666–676, September 1994.

[35] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata. *RTP Payload Format for MPEG-4 Audio/Visual Streams.* Internet Engineering Task Force, November 2000. RFC 3016.

[36] Rob Koenen. Overview of the MPEG-4 standard. Technical report, ISO/IEC JTC1/SC29/WG11, December 1999. `http://www.cselt.it/mpeg/standards/mpeg-4/mpeg-4.htm`.

[37] J.-Y. Lee, T.-H. Kim, and S.-J. Ko. Motion prediction based on temporal layering for layered video coding. In *Proc. ITC-CSCC*, volume 1, pages 245–248, July 1998.

[38] C. Leicher. Hierarchical encoding of MPEG sequences using priority encoding transmission (PET). Technical Report TR-94-058, ICSI, Berkeley, CA, November 1994.

[39] A. Li, F. Liu, and J. Villasenor. *An RTP Payload Format for Generic FEC with Uneven Level Protection.* Internet Engineering Task Force, November 2000. Internet Draft draft-ietf-avt-ulp-00.txt (`http://www.ietf.org/internet-drafts/draft-ietf-avt-ulp-00.txt`). Work in progress, expires May 2001.

[40] S. McCanne. *Scalable Compression and Transmission of Internet Multicast Video.* PhD thesis, Univ. of California at Berkeley, 1996.

[41] S. McCanne. Scalable multimedia communication with internet multicast, light-weight sessions, and the MBone. Technical Report CSD-98-1002, August 1998.

[42] Microsoft Windows Media Player. `http://www.microsoft.com/windows/mediaplayer/`.

[43] A. Miyazaki et al. *RTP Payload Format to Enable Multiple Selective Retransmissions.* Internet Engineering Task force, November 2000. `http://www.ietf.org/internet-drafts/draft-ietf-avt-rtp-selret-00.txt`.

[44] MPEG home page. `http://www.cselt.it/mpeg/`.

[45] M. Normura, T. Fujii, and N. Ohta. Layered packet loss protection for variable rate coding using DCT. In *Proceedings of International Workshop on Packet Video*, September 1988.

[46] ns-2 Network Simulator. http://www.isi.edu/nsnam/ns/, 2000.

[47] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP throughput: A Simple Model and its Empirical Validation. In *Proc. ACM SIGCOMM*, Vancouver, Canada, September 1998.

[48] J. Padhye, J. Kurose, D. Towsley, and R. Koodli. A Model Based TCP-friendly Rate Control Protocol. In *Proc. NOSSDAV*, July 1999.

[49] C. Papadopoulos and G. Parulkar. Retransmission-based error control for continuous media applications. In *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 5–12, 1996.

[50] J. B. Postel. *Transmission Control Protocol.* Internet Engineering Task Force, September 1981. RFC 793.

[51] Project mayo. `http://www.projectmayo.com/`, 2001.

[52] H. Radha and Y. Chen. Fine-granular-scalable video for packet networks. In *9th International Packet Video Workshop*, New York, NY, April 1999.

[53] K. Ramakrishnan and S. Floyd. *A Proposal to Add Explicit Congestion Notification (ECN) to IP.* Internet Engineering Task Force, Jan 1999. RFC 2481.

[54] K. K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems*, 8(2):158–181, May 1990.

[55] S. Raman, H. Balakrishnan, and M. Srinivasan. An Image Transport Protocol for the Internet. In *Proc. International Conference on Network Protocols*, Osaka, Japan, 2000.

[56] Real Networks. `http://www.real.com/`.

[57] R. Rejaie, M. Handley, and D. Estrin. Quality Adaptation for Unicast Audio and Video. In *Proc. ACM SIGCOMM*, Cambridge, MA, September 1999.

[58] R. Rejaie, M. Handley, and D. Estrin. RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In *Proc. IEEE INFOCOM*, volume 3, pages 1337–1345, New York, NY, March 1999.

[59] R. Rejaie, M. Handley, and D. Estrin. Layered Quality Adaptation for Internet Video Streaming. *IEEE Journal on Selected Areas of Communications (JSAC)*, Winter 2000. Special issue on Internet QOS. Available from `http://www.research.att.com/~reza/Papers/jsac00.ps`.

[60] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proc. ACM SIGCOMM*, September 1998.

[61] Rhee, I., Ozdemir, V. and Yi, Y. TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming. NCSU Technical Report, April 2000.

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

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

[64] N. Seshadri and V. Vaishampayan. Application of Multiple Description Codes to Image and Video Transmission Over Lossy Networks.

[65] K. Shimamura, Y. Hayashi, and F. Kishino. Variable bitrate coding capable of compensating for packet loss. In *Proceedings of Visiual Communications and Image Processing*, pages 991–998, November 1988.

[66] D. Sisalem and H. Schulzrinne. The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme. In *Proc. NOSSDAV*, July 1998.

[67] W. R. Stevens. *TCP/IP Illustrated, Volume 1.* Addison-Wesley, Reading, MA, November 1994.

[68] X. Su and B. W. Wah. Multi-description video streaming with optimized reconstruction-based DCT and neural-network compensations. *IEEE Transactions on Multimedia*, 3(3), May 2001.

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

[70] W. Tan and A. Zakhor. Packet classification schemes for streaming MPEG video over delay and loss differentiated networks. In *11th International Packet Video Workshop*, Kyongju, Korea, May 2001.

[71] Adaptive Video Streaming Home Page. `http://nms.lcs.mit.edu/projects/videocm/`, 2001.

[72] M. Vishwanath and P. Chou. An efficient algorithm for hierarchical compression of video. In *Proc. IEEE International Conference on Image Processing*, November 1994.

[73] B. W. Wah, X. Su, and D. Lin. A survey of error-concealment schemes for real-time audio and video transmissions over the internet. In *Proc. Int'l Symposium on Multimedia Software Engineering*, pages 17–24, Taipei, Taiwan, December 2000. IEEE.

[74] Wallace, G. The JPEG Still Picture Compression Standard. *Communications of the ACM*, April 1991.

[75] A. Wang and M. Schwartz. Achieving bounded fairness for multicast and TCP traffic in the Internet. In *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, September 1998.

[76] XMPS Home Page. `http://xmps.sourceforge.net/`, 2001.

[77] K. Yano, M. Podolsky, and S. McCanne. *RTP Profile for RTCP-based Retransmission Request for Unicast session.* Internet Engineering Task Force, July 2000. `http://www.ietf.org/internet-drafts/draft-ietf-avt-rtprx-00.txt`.