ABSTRACT

Srinath R. Joshi. Packet Loss Recovery for Unicast Interactive Video Transmission over the Internet. (Under the direction of Professor Injong Rhee.)

The current Internet is not reliable; packet loss rates are frequently high, and varying over time. Transmitting high-quality interactive video over the Internet is challenging because the quality of compressed video is very susceptible to packet losses. Loss of packets belonging to a video frame manifests itself not only in the reduced quality of that frame but also in the propagation of that distortion to successive frames. This error propagation problem is inherent in many motion-based video codecs due to the interdependence of encoded video frames. Recently a new approach to improving error resilience of compressed video, called *Recovery from Error Spread using Continuous Updates*(RESCU), has been proposed. In RESCU, picture coding patterns can be adjusted to enable the use of transport level recovery schemes in recovering lost packets without having to introduce any playout delay at the receiving end. In this thesis, we propose dynamic loss recovery schemes which when combined with RESCU, effectively alleviate error propagation in the transmission of interactive video. In these schemes, picture coding patterns are *dynamically* adapted to current network conditions in order to maximize the effectiveness of hybrid transport level recovery (employing both forward error correction and retransmission) in reducing error propagation. Through an experimental study based on actual Internet transmission traces representing various network conditions, we study the effectiveness and limitations of our proposed techniques and compare their performance with that of existing video error recovery techniques such as Intra-H.261 and H.263+(NEWPRED). Our study indicates that the proposed techniques exhibit better error resilience and incur much less bit overhead than existing video error recovery techniques. This document also describes the implementation of a prototype interactive video transmission system employing a proposed loss recovery scheme.

Packet Loss Recovery for Unicast Interactive Video Transmission over the Internet

by

Srinath R. Joshi

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, NC

February 2000

APPROVED BY:

Douglas S. Reeves

George N. Rouskas

Injong Rhee, Chair of Advisory Committee

To

Aai, Dada, Anna, and Tai.

BIOGRAPHY

Srinath R. Joshi was born in Bombay, India in 1973. He received his B.Tech degree in Computer Science and Engineering from the University of Calicut in 1995. He worked as a Software Engineer with Wipro Infotech and later Aditi Technologies in Bangalore, India before joining the graduate program in the Computer Science department at North Carolina State University in 1998. He is currently working towards completion of a M.S. in Computer Science.

ACKNOWLEDGEMENTS

I would like to thank Dr. Injong Rhee for supporting me as advisor during the course of my thesis research and for his invaluable encouragement and advice during my stay at NCSU. Working with him has been a highly rewarding experience and I really enjoyed the many hours of brain storming, which led to the fruition of this thesis.

I would also like to thank Ranjith Jayaram for his invaluable help in implementing the prototype system.

This thesis was supported in part by NSF CAREER ANI-9875651.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Packet losses are fairly common in the Internet. During busy time, about 5% to 10% packet losses over the connections between the east and west coasts of US, or over the trans-atlantic or trans-pacific connections are not unusual. Packet losses occur mainly due to congestion (or lack of bandwidth capacity) on the path from the source to the destination. A substantial number of packet losses can also be observed even in high speed networks such as vBNS when network interfaces, operating systems, or applications are too overloaded to keep up with the rate at which packets arrive or are transmitted [11]. Since packet losses in the current best-effort Internet cannot be avoided, applications like Internet-based video telephony must make provision against packet loss to minimize its impact on their performance.

The quality of compressed video is very susceptible to packet loss or corruption because of the way video signals are compressed. Most of the bits in the compressed video depend on previously transmitted bits so that they cannot be decoded if the previously transmitted bits are lost. For instance, typical video encoders (MPEG-I, MPEG-II, H.261, H.263) use variable length entropy coding (VLC), such as Huffman coding. Imagine that a number of variable length codewords are concatenated together to form a bit stream, and part of the stream is missing or corrupted. The remaining part after the corrupted bits cannot be decoded because the decoder does not know where each codeword starts – it lost synchronization with the encoded bit stream [51].

Furthermore, motion estimation and compensation in these codecs pose an even more severe problem, namely *error propagation* or *error spread*. Motion estimation removes temporal redundancy in successive video frames (inter frame) by encoding only pixel value differences between the current image and its motion-predicted version reconstructed using a previously encoded image (reference frame or R-frame). Image distortion in a reference frame due to packet losses can propagate to its succeeding frames.

Many researchers [41, 36, 30, 55, 13] have proposed using retransmission of lost packets by delaying frame playout times to allow arrival of retransmitted packets *before* the display times of their video frames. Any packets received after their display times will be discarded. In these schemes, the display time of a frame is delayed by at least three one-way trip times after its initial transmission (two for frame transmission and one for a retransmission request). This latency can significantly impair interactive communication under the current Internet environment.

Forward error correction is also commonly proposed for error recovery of continuous media transmission [29, 1, 26, 6, 3, 5]. However, conventional FEC schemes do not work well for interactive video. This is because unless the playout time of a frame is delayed, both the original packets and their parity packets must be transmitted within the same frame interval, rendering the schemes very susceptible to burst losses. Moreover, since FEC is applied to a block of packets, before FEC packets are computed and transmitted, a large delay must transpire.

Recently an entirely complementary scheme called *Recovery from Error Spread using Continuous Updates* (RESCU) is proposed [42, 43, 44]. Unlike other existing conventional approaches, RESCU focuses on eliminating error propagation using transport level recovery rather than preventing errors from happening in the first place. In RESCU, packets do not have to arrive in time for them to be "useful" for the display of that video frame. Clearly, if packets can arrive before the display time of their frame, that is optimal. However in today's Internet, where packet losses and long network latency are common, recovering lost packets before the display times of their frames is not always possible. Therefore, some repair packets may arrive after the display times of their frames. While the conventional techniques discard these "late" repair packets, RESCU can use them to stop error propagation. This is facilitated by buffering displayed frames, and later restoring them whenever repair packets arrive.

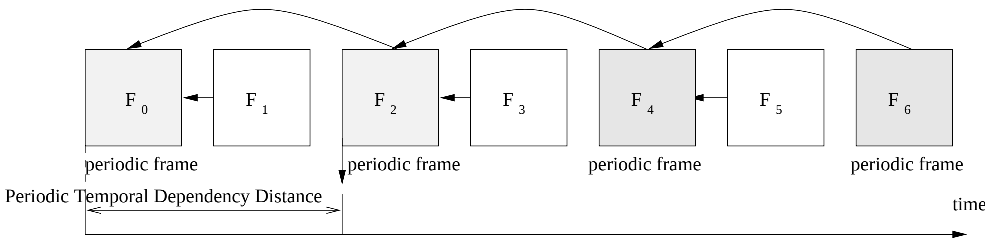

Figure 1.1: RESCU with PTDD 2

1.1 Overview of RESCU

We base our discussion on H.261, an International Telecommunication Union (ITU) video standard (although RESCU can be applicable to any video codec that employs motion compensation such as H.263). In H.261, a video sequence consists of two types of video frames: *intra-frame* (I-frame) and *inter-frame* (P-frame). I-frame removes only spatial redundancy present in the frame. P-frame is encoded through motion estimation using another P-frame or I-frame as a reference frame (R-frame). For each image block in a P-frame, motion estimation finds a closely matching block within its R-frame, and generates the displacement between the two matching blocks as a motion vector. The pixel value differences between the original P-frame and a motion-predicted image of the P-frame obtained by simply cut-and-pasting the matching image blocks from its R-frame are encoded along with the motion vectors. If any packet(s) belonging to a video frame are lost, not only is that frame shown with distortion but the error also propagates to the succeeding frames until the next synchronization point (an I-frame). However, I-frames cannot be sent often since they have a large number of packets, which would increase bandwidth consumption.

RESCU Using RESCU, we can improve the error resilience of H.261. In RESCU, packets arriving after their display times are not discarded but instead used to reduce error propagation. We define the *deadline of a packet* to be the time by which the packet must arrive at the receiver to be useful. RESCU allows this deadline to be arbitrarily adjusted through the *temporal dependency distance* (TDD) of a frame, which is the number of frame intervals between that frame and the frame it temporally depends on. By extending TDD,

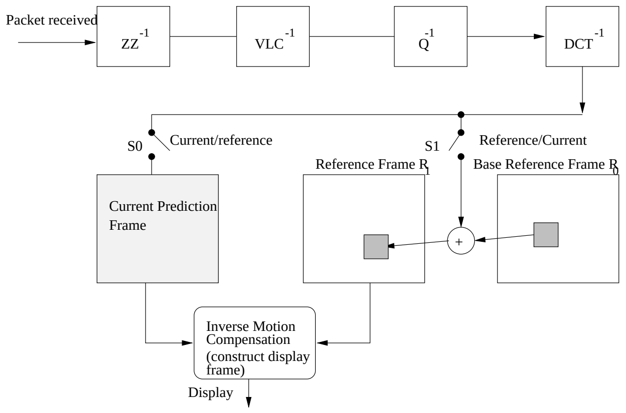

Figure 1.2: H.261 decoder modified to handle the recovery of reference frames.

we can arrange a frame to be referenced much later than its display time[1]. This adjustment essentially masks out the delay in repairing lost packets. For instance, we can make every p-th frame (which we call *periodic frame*) reference another periodic frame p frame intervals away. The TDD of the periodic frame is called *periodic TDD* (PTDD) [2]. Every non-periodic frame (frames between two consecutive periodic frames) depends only on its immediately preceding periodic frame. Thus the TDD of the non-periodic frames is between 1 and PTDD-1. Figure 1.1 shows an example of RESCU picture coding patterns with PTDD 2. Although a periodic frame may be displayed with errors because of some loss of its packets, when these losses can be recovered within a PTDD period, the errors will stop propagating beyond the next periodic frame. Also, errors in non-periodic frames do not propagate at all because all non-periodic frames temporally depend only on periodic frames. Note that extending TDD does not affect frame playout times because all frames are still displayed at their scheduled display times.

Figure 1.2 shows a H.261 decoder modified to handle the recovery of reference frames using recovered packets. The only difference from the original H.261 decoder is one additional frame buffer, base reference frame R_0, added to handle the recovery. In the figure, the current frame CP contains only the prediction error and motion vectors of the

[1]H.261 video standard always uses the immediately previous frame for motion compensation. That is, in H.261, TDD is always 1. Thus, the H.261 encoder and decoder, modified to support extension of TDD, will not be standard-compatible. In H.263+ video, extension of TDD can be supported through the use of a coding option supported by the standard.

[2]That is, if one frame interval is equal to δ_f time units then PTDD p is equal to a time period of $p \times \delta_f$ units.

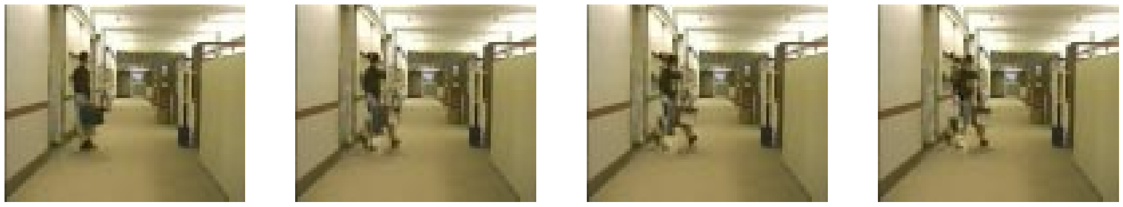

Figure 1.3: Error propagation

current frame while the reference frame buffer R_1 contains the fully motion compensated image of the reference frame (i.e., periodic frame) of CP, and the base reference frame buffer R_0 contains the reference frame of R_1. When a packet is received and decoded into an image block, the decoder determines whether the block belongs to the current frame being decoded or its reference frame. If it is for the current frame, then the block is stored into frame buffer CP along with its motion vector. If it is for the reference frame, the block is added with its temporally dependent block in frame buffer R_0 and stored into R_1. At each display time, the current frame is constructed using the information in CP and R_1. If the current frame is a periodic frame, after displaying that frame, R_1 is copied to R_0 and the displayed image is copied to R_1. In this scheme, as long as the packets belonging to R_1 arrive before the construction of the current frame, the packet can be used to prevent errors in the reference frame from propagating to the current frame.

Figures 1.3 and 1.4 show video clips from a proof-of-concept experiment. The distortion in the second picture of Figure 1.3 is due to packet losses, which propagates even though the rest of frames are correctly received in time. However, in Figure 1.4, when RESCU is used, the quality of the third picture immediately bounces back when the packets for the second frame are recovered before the decoding of the third frame.

Supporting RESCU in H.263+ does not require any change in the current standard of H.263+. The International Telecommunication Union (ITU) adopted a technique called *Reference Picture Selection*(RPS) [22] which allows the encoder to select any previously decoded frames as a reference frame for prediction. Since RPS allows the reference frame address of a frame to be encoded with that frame, PTDD can be adjusted by simply modifying this address.

Replenishment It is also possible that RESCU can fail. When buffers are not available, or PTDD is too short for incurred repair delays, periodic frames cannot be recovered before

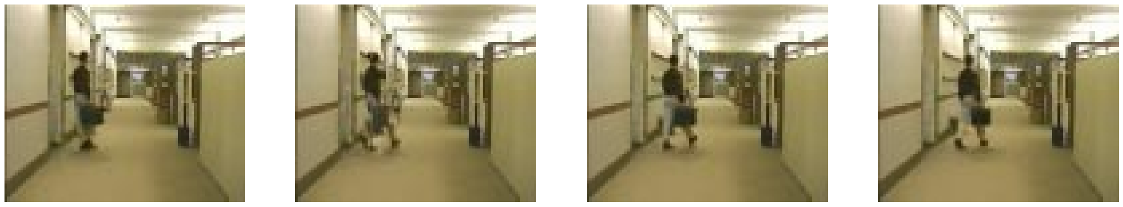

Figure 1.4: RESCU stops error propagation

the decoding of their dependent frame. This also leads to error propagation. To prevent this type of error propagation, a commonly adopted technique called *replenishment* can be used. The receiver can detect losses in periodic frames not recovered even after a PTDD period and notify the sender about those irrecoverable losses. On reception of such a notification, the sender can code the next frame as an intra-frame. The intra-frame stops error propagation due to the earlier (unrecovered) losses because the intra-frame does not have temporal signal dependency with any frames transmitted earlier. However, since it significantly increases bandwidth consumption a recovery scheme has to strive to minimize the number of replenishments.

1.2 Packet loss recovery and RESCU

1.2.1 RESCU + Retransmission

Retransmission is the most commonly used loss recovery technique for reliable data transport. The sender (or another receiver in a multicast environment) simply retransmits the packets reported missing by a receiver[3]. Since repair packets are retransmitted only when some indications exist that the packets are lost, retransmission incurs very low bit overhead. In addition, retransmission is less susceptible to burst losses. This is because the time distance between the time when the initial losses occur and the time when the corresponding retransmission effects is large enough for the initial losses to not affect the loss of retransmitted packets.

However, for interactive video transmission, conventional retransmission techniques do not work well. Conventional techniques require retransmitted packets to arrive within a single frame interval after the time that they are first lost, but the associated delays

[3]Most real-time video applications use UDP as a transport protocol. Since UDP does not perform any retransmission itself, it is indeed the application which retransmits lost packets.

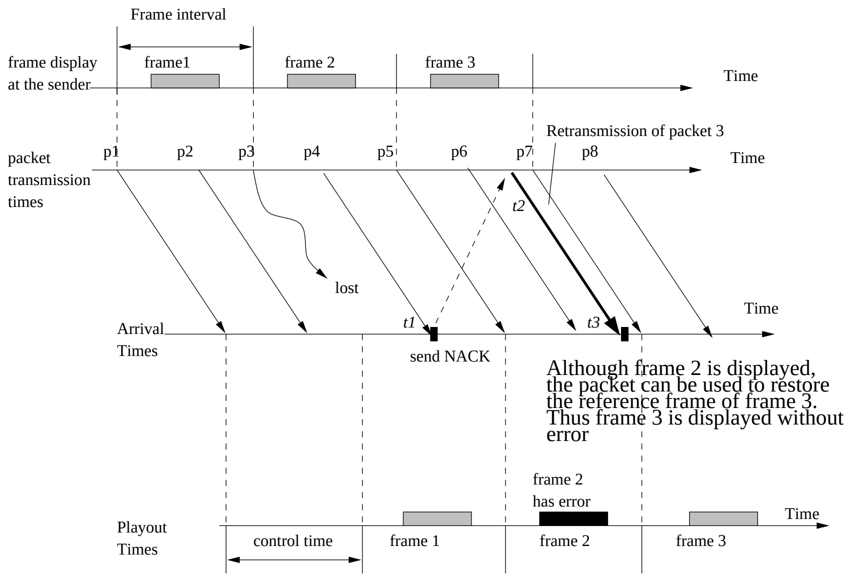
Frame interval
frame display at the sender
frame1
frame 2
frame 3
Time
Retransmission of packet 3
packet transmission times
p1
p2
p3
p4
p5
p6
p7
p8
Time
t2
lost
Time
Arrival Times
t1
send NACK
t3
Although frame 2 is displayed, the packet can be used to restore the reference frame of frame 3. Thus frame 3 is displayed without error
frame 2 has error
Playout Times
control time
frame 1
frame 2
frame 3
Time

Figure 1.5: The recovery of frames using retransmission.

in detecting and retransmitting the lost packets are often larger than one frame interval.

In contrast, RESCU can effectively mask out repair delays since retransmitted packets need to be received only within a PTDD period. Figure 1.5 illustrates error recovery using RESCU and retransmission in a video stream containing two packets per frame and PTDD 2. In this figure Packet 3 is lost, and the receiver receives packet 4 at time t_1. Recognizing that packet 3 is not received, the receiver sends a retransmission request (NACK) to the sender. The sender gets the NACK at time t_2 and retransmits packet 3. The retransmitted packet arrives at time t_3 which is before frame 3 is displayed. Packet 3 is now used to restore the reference frame of frame 3 (frame 1), so frame 3 can be decoded and displayed without an error. This retransmission technique is fundamentally different from other retransmission techniques [41, 36, 30, 55] in that it does not introduce any delay in frame playout times.

1.2.2 RESCU + Forward Error Correction (FEC)

One main disadvantage of retransmission-based loss recovery is that its performance is too sensitive to transmission delays. Although RESCU can accommodate larger transmission delays than conventional retransmission schemes, a larger transmission delay requires larger PTDD. As PTDD increases, compression efficiency gets lower because two

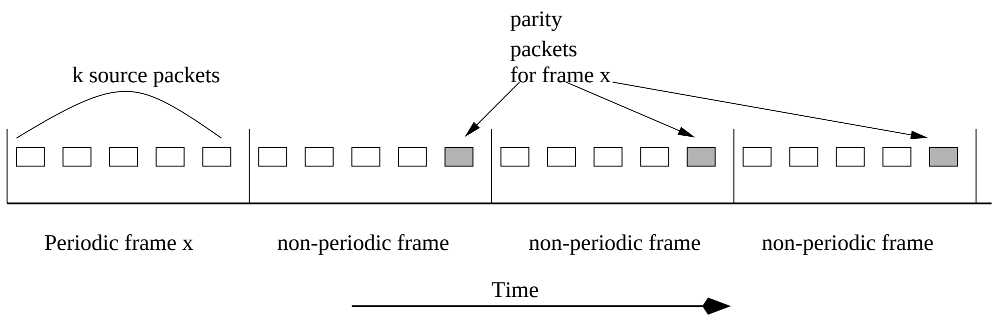

Figure 1.6: Interleaving FEC packets with packets of non-periodic frames.

consecutive periodic frames may not have much temporal redundancy, and the TDD of non-periodic frames also increases. In addition, packet losses in periodic frames can be restored only after one round trip time. Thus, during the time, non-periodic frames can have errors propagation.

Furthermore, in some networks, sending feedback to the sender can be costly. Over direct broadcast satellite links or cable modems, feedback channels are highly bandwidth limited and contention-based. Some mobile wireless hosts simply do not have extra capacity to send feedback frequently to the sender. Furthermore, since FEC is an open-loop recovery scheme, its associated recovery delays can be much shorter than retransmission. Thus FEC is a very compelling alternative for these environments.

Linear Block Coding (LBC) is a commonly used FEC scheme in which k source packets, $d_1, d_2, \ldots, d_k$, are encoded into n packets(i.e., $n-k$ FEC packets). These n packets constitute a *block*. The LBC decoder at the receiver side can reconstruct the k original data packets using any k packets from the n packet block. Efficient (n, k) LBC encoding and decoding algorithms have been developed and implemented to achieve real-time performance [2, 24, 45, 35]. For instance, the software coder by Rizzo [45] can achieve throughput of 11 MB/s on a 133 MHz Pentium.

On the other hand, FEC is not effective when the losses of original data packets and that of FEC repair packets are correlated. RESCU greatly alleviates this problem by allowing the FEC repair packets of a periodic frame to be dispersed over the PTDD period. FEC packets can be spaced farther apart (Δ time units) from each other so that their losses are less correlated, thus diffusing the effect of bursty losses. Also, the FEC repair packets of a block can be sent Δ time units after the data packets in that block. Figure 1.6 shows a packet sequence in RESCU where FEC packets are sent at each frame interval within the PTDD period. That is, Δ is set to one frame interval, δ_f.

Conventional FEC schemes can be categorized into two kinds. One type is to transmit both data and their parity packets within the same frame interval. The other scheme is to transmit the parity packets in later frame intervals than the interval in which data packets are sent. The former scheme is susceptible to burst packet losses and since FEC is applied to a block of packets, before FEC packets are computed and transmitted, large delay must transpire. The latter scheme has to introduce additional delays in frame playout times to allow enough time for the receiver to receive parity packets and restore the currently displayed images. Although these schemes can be effective for a one-way, near-real-time video transmission, it seriously impairs interactive video communication.

In general, the bit overhead of FEC schemes is sensitive to packet size. This is because each packet contains a set of protocol headers which incur almost a constant bit overhead no matter how small a packet is. Over the Internet, the header size could be at least 40 octets (IPv4) when RTP/UDP/IP is used for video transmission. Thus, it is more advantageous to use larger packets to reduce the bit overhead due to protocol headers. However, larger packets increase the granularity of forward error correction, thus, potentially also increasing bit overhead. Recognizing this header overhead, researchers are proposing new Internet standards for header compression [9, 12] which reduce the header size up to 2 octets. Thus, it is expected that the sensitivity of a FEC scheme to packet size will not be critical in the future.

1.3 Problem statement

So far we have seen how transport level recovery schemes and RESCU can work together in curtailing error propagation in interactive video transmission over lossy networks, like the Internet. However, both retransmission-based recovery and FEC-based recovery have their own drawbacks. The main drawback of retransmission-based recovery is that long round-trip delay (RTT) can prolong error propagation. Since lost packets can be recovered only after one RTT, errors can propagate to at least next RTT/δ_f frames. The main drawback of FEC is that it incurs bit overhead regardless of packet losses since FEC packets are transmitted proactively. Therefore, it wastes bandwidth during error-free transmission; retransmission has less of this problem. A hybrid technique employing both retransmission and FEC can be very effective if it can take advantage of the tradeoffs of FEC and retransmission.

Furthermore, network conditions vary over time; congestion, transmission latency, loss rates, and available bandwidth frequently change. As network conditions change, the effectiveness of transport level recovery (retransmission and FEC), and thus the associated recovery delays change. Thus, PTDD must be adapted to varying network conditions to allow enough time for packets to be delivered before the recovery of succeeding periodic frames. Note that PTDD cannot be set arbitrarily large because it reduces compression efficiency. Therefore, we must choose a minimum PTDD under given network conditions that maximizes the periodic frame recovery probability. Packet loss rate, loss burst length, and transmission delays play an intricate role in determining PTDD. For instance, if the loss rate increases, then either more FEC packets or retransmission attempts are required to maintain high loss recovery probabilities. Thus PTDD has to be extended to accommodate these increased number of repair attempts. Furthermore, loss burst characteristics significantly also affect PTDD. As network traffic undergoes more bursty losses, retransmission becomes more effective than FEC since repair packets are transmitted only when losses occur. Hence, the loss burst characteristics influence the decision to use FEC or retransmission or both for recovery. When retransmission is used, PTDD must be at least as long as one round trip time (RTT). When FEC is used, PTDD should be at least as long as the product of the time interval between two consecutive FEC packets and the number of FEC packets required for protecting a periodic frame. When a hybrid technique combining FEC and retransmission is used, finding minimum PTDD becomes very complex.

In this thesis, we first present a dynamic hybrid loss recovery scheme, at the core of which is an algorithm to find minimum PTDD that is sufficiently large to achieve the desired recovery probability of periodic frames based on predicted network conditions. The network conditions during the next PTDD period are predicted based on the knowledge of the past network behaviors. This algorithm relies on exhaustive search around its parametric space to find minimum PTDD (we call it *exhaustive hybrid RESCU*). For experimental evaluation we also create two variants of this dynamic hybrid scheme, each using either FEC or retransmission (but not both) as a transport level recovery mechanism. The advantage of predictive modeling of future events as used in this hybrid scheme is that when network conditions are stable and predictable, it can achieve very high error resilience with controlled bit overhead. Moreover, this scheme requires only one additional frame buffer in the decoder. However, when network conditions are highly bursty and unpredictable, PTDD estimation based on predictive modeling of future events tends to waste bandwidth because bit overhead is incurred even if (predicted) packet losses do not occur. In addition, since this algorithm requires the calculation of recovery probability, it may be computationally expensive. Thus, exhaustive hybrid RESCU is more suitable for a QoS controlled environment such as DiffServ, where network characteristics are more predictable and do not vary frequently.

To overcome some of the disadvantages of the exhaustive algorithm, we develop another dynamic algorithm for hybrid loss recovery, called *lazy hybrid RESCU*. This algorithm adjusts PTDD for retransmission when the actual packet losses cannot be recovered through a proactive recovery technique such as FEC alone, so as to minimize the chances of ensuing frames being affected by the losses in the periodic frame. The main advantage of this strategy is that even if a proactive technique fails, it can still recover from losses. This allows lazy hybrid RESCU to budget only a small amount of bit overhead for proactive recovery. Since further bit overhead due to retransmission is incurred only when unanticipated bursty losses occur, this algorithm can outperform the exhaustive hybrid algorithm when the packet loss characteristics exhibit a high degree of variability, such as in today's Internet. This scheme is very computationally efficient and thus, suitable as an on-line protocol. However, this scheme needs more frame buffers at the encoder and the decoder than the exhaustive recovery scheme which needs just one additional frame buffer at the decoder.

The remainder of the thesis is organized as follows. In Chapter 2, we discuss related work. In Chapter 3, we describe in detail the proposed dynamic loss recovery schemes. In Chapter 4, we present the experimental result. In Chapter 5, we describe implementation of a prototype video transmission system employing lazy hybrid RESCU. In Chapter 6, we present concluding remarks and briefly comment on possible future work.

Chapter 2

Related Work

Error concealment is one of widely proposed error control techniques (e.g.,[19, 25, 56, 33]). Although error concealment techniques can be combined with error recovery techniques, they are not strictly error recovery techniques [8], so we do not discuss them. We focus only on recovery techniques for video transmission.

2.1 Feedback-based recovery

Recently H.263+ incorporated two feedback-based techniques: error tracking and reference picture selection. Error tracking (ET) utilizes the intra-coding of blocks to stop error propagation, but limits its use to severely impaired image regions only [49]. ET requires the encoder to know the location and extent of erroneous image regions in displayed images. This can be achieved by feedback from the receiver. The receiver sends information about missing packets, and the encoder estimates the region of error propagation in the displayed images, and intra-codes those blocks contained the region. The reference picture selection (RPS) mode allows the encoder to select one of several previously decoded frames as a reference picture for motion estimation. It is designed to support a coding technique called NEWPRED [28]. In NEWPRED, using feedback from the receiver, the encoder uses for prediction only those pictures that are reported to be received (in the *ACK mode*), or not reported missing (in the *NACK mode*). Since motion prediction is always based on the frames that are received by the receiver, error propagation is effectively eliminated. Wada [54] proposed a scheme similar to ET. As in ET, the encoder computes the extent of error propagation caused by lost packets. However, unlike ET where affected macroblocks are intra-coded, Wada's scheme simply excludes them from next motion prediction. Thus, error propagation can be stopped when new frames are coded using only those regions that are not affected by lost packets.

Retransmission has recently attracted much attention for packet loss recovery in video transmission. Ramamurthy and Raychaudhuri [41] applied a similar technique to video transmission over ATM. They analyzed the performance of video transmission over an ATM network when both retransmission and error concealment are used to repair errors occurring from cell loss. Padopoulos and Parulkar [36] proposed an implementation of an ARQ scheme for continuous media transmission. Various techniques including selective repeat, retransmission expiration, and conditional retransmission are implemented inside a kernel. Their experiment over an ATM connection showed the effectiveness of their scheme.

Retransmission is also used for video multicast. Li *et al.* [30] proposed an elegant scheme for multicasting MPEG-coded video. By transmitting different frame types (I, P and B frames) of MPEG to different multicast groups, they implemented a simple layering mechanism in which a receiver can adjust frame play-out times during congestion by joining or leaving a multicast group. Retransmission is used for high priority streams. The scheme is shown to be effective for non-interactive real-time video applications. In a video conference involving a large number of participants, different participants may have different service requirements. While some participants may require real-time interactions with other participants, others may simply want to watch or record the conference. Xu *et al.* [55] contended that retransmission can be effectively used for the transmission of high quality video to the receivers that do not need a real-time transfer of video data. They designed a new protocol called *structure-oriented resilient multicast* (STORM) in which senders and receivers collaborate to recover lost packets using a dynamic hierarchical tree structure.

Remarks A drawback of feedback-based techniques is that when the networks do not support feedback channels, they are useless. If feedback channels are supported, but limited in bandwidth, then those techniques requiring frequent feedback may not be effective.

ET and NEWPRED adopted in H.263+ have a serious limitation. First, they modify their picture coding patterns based on specific information about lost packets (such as which frame lost packets belong to or which macro blocks lost packets contain). Thus, continuous feedback from a receiver is essential in their performance. In some networks such as wireless cable modems and direct satellites, feedback channels are highly limited and contention-based, or even unavailable. Often mobile hosts are too low powered to transmit frequent feedback. Furthermore, since these schemes are entirely closed-loop recovery techniques, their effectiveness is very sensitive to delays in receiving feedback (round trip delays).

In addition, all of existing retransmission techniques (except RESCU) use frame playout delays to compensate for retransmission delays.

2.2 Proactive recovery

Error propagation can be alleviated by intra-coding more image blocks, but at the expense of compression efficiency. Several popular video conferencing tools, such as `nv`[18], `vic`[34] and `CU-SeeMe`[15], adopt this approach. Using a technique called *conditional replenishment*, these tools filter out the blocks that have not changed much from the previous frame and intra-code the remaining blocks. Since all the coded blocks are temporally independent, packet loss affects only those frames that are contained in lost packets.

FEC has been successfully applied to audio transmission [4, 3, 38, 39]. There are only a few studies on applying FEC to video transmission. *Priority encoding transmission* (PET) [1, 29, 50], encodes different segments of video data with different priority. Each packet contains relatively more redundant information about the higher priority segments of the data, so the information with a higher priority can have a higher chance of correct reception. The PET scheme is also incorporated into `vic` [34] and is reported a good performance [52]. This good performance is partially due to `vic`'s intracoding which limits error propagation caused by loss of lower priority segments.

Bolot and Turletti [6] proposed an interesting FEC technique for packet video where a packet contains the redundant information of some of previous packets. The redundant information is created by encoding the image blocks contained in the previous packets with a large quantization step. They claimed that if the video source is not bursty, long burst losses are rare, and the proposed scheme would work well for video.

H.263+ also includes a similar technique called *independent segment decoding* (ISD) [27]. In the ISD mode, each video slice is encoded as an individual picture (or subvideo) independent of other slices. In particular, each slice boundary is treated just like picture boundary. ISD does not eliminate error propagation, but limits the extent of error propagation to a slice.

MPEG-4 adopted several error resilient techniques for wireless video transmission [51]. These include resynchronizations strategies, data partitioning, reversible VLCs, and header extension codes. Most of these techniques focus on preventing lost data from affecting the decoding of received data. For instance, loss of some header information affects all the data that tagged on the header although they are received correctly. The techniques minimize this effect.

Remarks A drawback of proactive techniques is that it wastes bandwidth under error-free transmission. Furthermore, in wireless mobile networks, the assumption made in [6] does not hold as long burst losses can be common due to fading and channel interference. Thus, the FEC schemes mentioned above are susceptible to burst errors because both data and FEC-encoded packets have to be transmitted at the same frame interval. Also if FEC-encoded packets are transmitted over a longer period, additional playout delays may be incurred. MPEG-4's techniques can be used along with RESCU to further reduce the effect of data loss.

2.3 Hybrid recovery

Hybrid techniques combine ARQ (retransmission) and FEC for better error resilience. There are two types of hybrid techniques: *type-I hybrid ARQ* [14], and *type-II hybrid ARQ* [31]. Type-I hybrid ARQ transmits both error detection and correction data at the initial transmission of data. If the receiver cannot recover lost packets using the transmitted parity data, it requests retransmission of the same data from the sender. Type-II hybrid ARQ does not send any redundant data with the first transmission, but sends only parity data when retransmission is requested.

Liu and El Zarki [32] applied a hybrid ARQ technique for video transmission. They proposed a hybrid ARQ technique that combines the benefit of type-I and type-II techniques, and showed its efficacy for video transmission over wireless networks. They showed that using rate-compatible punctured convolutional (RCPC) codes [20, 23], one or two retransmission attempts achieve a low packet error rate under a perfectly interleaved Rayleigh fading channel.

Hybrid ARQ techniques were also studied in reliable multicast [47, 35]. While FEC helps reduce occurrences of independent losses, retransmission repairs correlated packet losses. They showed that hybrid ARQ reduces the bandwidth overhead of repairing packet losses in reliable multicast involving many receivers.

Remarks Although hybrid ARQ schemes work better than FEC or retransmission alone, they do not overcome the limitations of traditional recovery techniques. Since retransmission always incurs delays and FEC is still susceptible to burst losses, hybrid ARQ scheme still fall short of handling the retransmission delays and burst losses without introducing delays in frame playout times. Delaying frame playout reduces interactiveness and introduces annoying jitters.

Chapter 3

Dynamic Packet Loss Recovery

In this chapter we first describe the end-to-end system architecture and loss model used in the development of our proposed dynamic algorithms for packet loss recovery, followed by a detailed description of those algorithms.

3.1 System architecture

The sender consists of three components: *video encoder*, *transmitter*, and *adapter*. The video encoder captures a live video frame and compresses it at every frame interval, δ_f time units. The encoder passes the compressed bit stream to the transmitter which packetizes the stream and sends the packets to the receiver. Each packet header (including retransmitted packets)contains a unique packet sequence number, and current timestamp. The transport protocol is based on RTP/RTCP [48]. If the loss recovery scheme uses retransmission then the transmitter also buffers all the packets of a periodic frame until they are no longer needed for retransmission.

The receiver sends a negative acknowledgment (NACK) for each lost packet (including repair packets) of the periodic frames only. Recall that since non-periodic frames are not used as reference frames, loss of their packets does not cause any error propagation. Hence, we do not recover non-periodic frames after their display and therefore, explicit feedback (in the form of NACKs) regarding their lost packets is not necessary. This has an added advantage of reducing the use of the feedback channel.

While receiving packets sent by the sender, the receiver collects statistics on the network parameters such as the number of packets lost, and the mean loss burst length.

Packet losses are detected by gaps in the sequence number of received packets. The mean loss burst length is estimated by adding all instances of loss bursts (including a single loss) observed in an interval of 500 ms, and dividing the total by the number of burst loss instances (including single loss) in that same interval. The fraction of packets lost since the last report (as per the RTCP terminology) is included in the receiver report of RTCP, and the mean loss burst length is added in the application-specific extension field of an RTCP report packet. A receiver report (RR) packet is sent to the sender every 500 ms.

The adapter receives the RR and computes the mean loss rate by taking the weighted average over its old value, and the current sample of the loss rate. It also computes the mean loss burst length in a similar fashion. The round trip time (RTT) is estimated from the timestamp carried in the RTCP report [48].

Before transmitting each periodic frame, the adapter chooses a suitable PTDD period using one of the PTDD algorithms presented in this chapter. The adapter performs this task using the values of the network parameters obtained from receiver reports. If the loss recovery scheme makes use of FEC, then the adapter must also compute the number of FEC repair packets to be transmitted during that PTDD period, and pass that information to the transmitter which encodes FEC repair packets for the periodic frame, and interleaves them with other data packets (packets of non-periodic frames) being transmitted over that PTDD period.

3.2 Loss model

Packet loss in the Internet is often characterized as bursty. That is, when a packet is lost, it is more probable to lose the following packet(s). This loss process in the Internet is commonly modeled using a two state continuous Markov chain $\{X(t)\}$, where $X(t) \in \{0, 1\}$. In this model, a packet sent at time t is lost if $X(t) = 1$ and not lost if $X(t) = 0$.

The infinitesimal generator of this Markov chain is:

$$Q = \begin{pmatrix} -\mu_0 & \mu_0 \\ \mu_1 & -\mu_1 \end{pmatrix}$$

The stationary distribution associated with this Markov chain is: $\pi = (\pi_0, \pi_1)$ where, π_0 and π_1 are the probabilities of the system initially residing in states 0 and 1 respectively. π_0 and π_1 can be expressed as follows: $\pi_0 = \mu_1/(\mu_0+\mu_1)$ and $\pi_1 = \mu_0/(\mu_0+\mu_1)$.

μ_0 and μ_1 can be computed based on following measurable parameters: the mean packet loss rate (p), the mean loss burst length (b), and the mean packet transmission rate (λ) [35]. These parameters are measured at the end points of the connection, and characterize the network conditions on the end-to-end path from the sender to the receiver. Using the network parameters mentioned above, μ_0 and μ_1 can be expressed as follows: $\mu_0 = -\pi_1 \lambda \log(1 - 1/b)$ and $\mu_1 = \mu_0(1-p)/p$. In [35], the authors derive expressions for finding the probability that the two-state model will be in state j at time $t+\tau$ if it is in state i at time τ. If $p_{i,j}(t)$ denotes that probability, then $p_{i,j}(t), i = 0, 1, j = 0, 1$, can be expressed as follows.

$$\begin{aligned}
p_{1,0}(t) &= \mu_1(1 - exp(-(\mu_0 + \mu_1)t))/(\mu_0 + \mu_1), \\
p_{0,1}(t) &= \mu_0(1 - exp(-(\mu_0 + \mu_1)t))/(\mu_0 + \mu_1), \\
p_{1,1}(t) &= (\mu_0 + \mu_1 exp(-(\mu_0 + \mu_1)t))/(\mu_0 + \mu_1), \\
p_{0,0}(t) &= (\mu_1 + \mu_0 exp(-(\mu_0 + \mu_1)t))/(\mu_0 + \mu_1)
\end{aligned}$$

for all $t > 0$.

Let $\mathcal{D}(m,n)$ be the probability of receiving exactly n packets out of m packets under the two state model (correlated loss), where the packets are transmitted at intervals of $\delta = \frac{1}{\lambda}$ units. We can obtain its value by solving following recursive equations obtained from [47].

Let $\mathcal{D}(m,n,1)$ be the probability of sending m packets, receiving exactly n of those packets and finishing up in state 1, and $\mathcal{D}(m,n,0)$ be the similar probability except that the final state is 0. It follows that:

$$\begin{aligned}
\mathcal{D}(m,n,1) &= \mathcal{D}(m-1,n,1) \times p_{1,1}(\delta) + \mathcal{D}(m-1,n,0) \times p_{0,1}(\delta) \\
\mathcal{D}(m,n,0) &= \mathcal{D}(m-1,n-1,1) \times p_{1,0}(\delta) + \mathcal{D}(m-1,n-1,0) \times p_{0,0}(\delta) \\
\mathcal{D}(m,n) &= \mathcal{D}(m,n,1) + \mathcal{D}(m,n,0)
\end{aligned}$$

The base conditions used by the recursive equations given above are:

$$\mathcal{D}(n,n,1) = 0$$

$$\begin{aligned}
\mathcal{D}(n,n,0) &= \pi_0 \times (p_{0,0}(\delta))^n + \pi_1 \times p_{1,0}(\delta) \times (p_{0,0}(\delta))^{n-1} \\
\mathcal{D}(n,0,1) &= \pi_0 \times p_{0,1}(\delta) \times (p_{1,1}(\delta))^{n-1} + \pi_1 \times (p_{1,1}(\delta))^n \\
\mathcal{D}(n,0,0) &= 0
\end{aligned}$$

3.3 Exhaustive hybrid RESCU

Exhaustive hybrid RESCU is a dynamic hybrid loss recovery scheme, at the core of which is an algorithm which performs an exhaustive search to find PTDD that gives the least bit overhead while maintaining the recovery probability (P_{hybrid}) of a periodic frame above a given threshold, θ. The algorithm uses recently observed network conditions to predict network conditions like loss events, transmission latencies etc. in the near future. Following assumptions are made in the computation of PTDD.

- The reduction in compression efficiency due to PTDD i can be modeled using a monotonically non-decreasing function, $g(i)$.
- RTT does not change over a PTDD period [1].
- Feedback from the receiver is always reliably received [2].
- No packets are received out of order [3].

3.3.1 Protocol

The hybrid protocol runs using the system architecture described in Section 3.1. For each periodic frame F_i, the *adapter* determines an appropriate PTDD period, *ptdd*, the number of FEC packets, f, and FEC distance, Δ, according to the latest network conditions.

Suppose that frame F_i is packetized into k data packets. The *transmitter* first sends these k data packets back-to-back during one frame interval. It then transmits the f FEC packets encoded to protect frame F_i, one FEC packet at each successive interval of Δ time, interleaving with the packets of non-periodic frames. We call this the *initial transmission* of data and FEC packets.

[1]The main purpose of this assumption is to simplify the analysis. In our actual transmission tests, we allow RTT to change during the PTDD period

[2]In our experimental setup, we allow feedback packets to be lost.

[3]It has been observed that generally the scale of the reordering is very short [37] so that a 8-20 ms wait can distinguish most reordering from packet losses. Therefore reordering does not have a significant impact.

For each lost packet (data and repair packets) of the periodic frame F_i, the receiver sends a NACK to the sender. Because of our assumption of in-order delivery, we can say that NACKs for lost data packets of F_i arrive at the sender before those for lost FEC packets of F_i. The transmitter maintains a counter c_i for F_i which keeps the number of NACKs received for F_i. After c_i becomes equal to f, each received NACK triggers retransmission of one data packet reported lost among the k data packets of F_i. If a retransmitted packet is reported lost, then it is retransmitted again with a new sequence number. When the sender receives any NACKs for the periodic frame F_i after its PTDD period is expired and $c_i \geq f$, it clearly indicates that the periodic frame is still unrecovered. Hence, the next frame to be transmitted is intra-coded (I-frame).

3.3.2 Estimating Δ

Δ is the time separation between data packets and FEC packets, and also the time separation between two consecutive FEC packets. In order to avoid correlated losses of data packets and FEC packets, Δ must be larger than the correlation timescale of a loss burst. That is, when a packet loss occurs at time t, the probability of a packet loss at time $t + \Delta$ should be close to the random loss probability π_1 (i.e., p). Recall that, $p_{1,1}(t)$ represents the probability that the loss model presented in Section 3.2 is in state 1 at time $t + \tau$, given that it was in state 1 at time τ. We can find Δ such that $(p_{1,1}(\Delta) - p)/p \leq \epsilon$, where ϵ is a small constant (in our experiment, we set it to 0.01). The above equation leads to $\Delta = \frac{-1}{\mu_0+\mu_1} ln \left(\frac{\mu_0}{\mu_1}\right) \epsilon$.

3.3.3 Computing recovery probability, P_{hybrid}

We now describe how to compute the probability that a periodic frame, F_i, comprised of k data packets and f FEC packets can be recovered within the PTDD period, *ptdd*, through the hybrid recovery protocol described above. This probability is denoted as $P_{hybrid}(k, f, ptdd)$. There are exactly three distinct cases that should be considered for computing $P_{hybrid}(k, f, ptdd)$.

1. At least k packets are received out of the initial transmission of $k + f$ (data and FEC) packets, and therefore, the periodic frame is recovered without any retransmission. Let P_1 denote the probability of this event.

2. More than f data packets are reported lost in the initial transmission of k data packets. In this case, retransmission is necessary. Let P_2 denote the probability of this event.

3. f or fewer data packets in the initial transmission of k data packets are reported lost, but loss of FEC packets causes retransmission. Let P_3 denote the probability of this event.

It can be easily observed that the three cases mentioned above are mutually exclusive and they cover all the recovery scenarios that can arise using the hybrid recovery protocol. Thus, $P_{hybrid}(k, f, ptdd) = P_1 + P_2 + P_3$.

Computing P_1 Given the estimated value of Δ, we can now assume that FEC repair packets undergo uncorrelated losses, and the loss probability is p. Clearly, the periodic frame can be recovered without retransmission if and only if i packets out of k data packets are received and at least $k-i$ packets out of f FEC packets are received (making the total number of packets, data and FEC, received at least equal to k). Thus, we can compute P_1 as follows.

$$P_1 = \sum_{i=max(k-f,0)}^{k} \left\{ \mathcal{D}(k,i) \times \sum_{j=k-i}^{f} P(f,j) \right\}$$

As FEC packets spaced Δ time apart are assumed to undergo uncorrelated losses, $P(f,j)$ which denotes the probability of receiving exactly j packets out of f FEC packets under uncorrelated losses, can be computed using a (1-state) Bernoulli model as: $P(f,j) = C_j^f(1-p)^j(p)^{f-j}$. Note that the transmission of k data packets (sent back-to-back) will undergo correlated losses. Therefore, $\mathcal{D}(k,i)$ is used in P_1 to denote the probability of receiving i packets out of k data packets.

Computing P_2 According to the protocol, for each NACK received after c_i becomes equal to f, the sender retransmits a lost data packet. When more than f data packets out of the initial transmission of k data packets are lost, as in this case, c_i eventually becomes larger than f, triggering retransmission. At most r, $r = \left\lfloor \frac{ptdd}{rtt} \right\rfloor$, retransmission attempts are possible, before the PTDD expires, to recover lost packet(s). rtt is the Round Trip Time.

The recovery process under this case can be considered as a combination of the following two events:

1. Loss of more than f data packets causes a series of retransmissions to ensure that $k-f$ packets are received (its probability denoted as $P_{frtx}(f,k,r)$),
2. The "remaining" f repair packets are received when either
   (a) all the f FEC packets are received without loss or
   (b) some (say, x packets) of the f FEC packets are lost, triggering a series of retransmissions to recover additional x data packets(its probability denoted as $P_r(f)$).

The probability of the first event, $P_{frtx}(f,k,r)$, can be expressed as follows.

$$P_{frtx}(f,k,r) = \sum_{i=f+1}^{k} \{\mathcal{D}(k,k-i) \times R(i-f,r)\}$$

The summation over i from $f+1$ to k indicates that at least $f+1$ data packets are lost during the initial transmission of k data packets.

$R(n,r)$ denotes the probability that n lost packets are recovered within r retransmission attempts. Since consecutive retransmissions of the same packet are separated by rtt which is longer than Δ, losses experienced by these retransmission attempts (of the same packet) are uncorrelated. However, retransmitted packets sent together (triggered by correlated loss of data packets) undergo correlated losses. If i ($i>f$) packets are lost when k data packets are initially transmitted, only $i-f$ packets out of those i lost data packets have to be retransmitted. Some of these retransmitted packets could be lost again and thus those lost packets must be retransmitted again. This continues until PTDD expires (i.e, r retransmission attempts have been exhausted) or all the $i-f$ lost packets have been recovered. Thus $R(n,r)$ can be computed recursively as follows.

$$R(n,r) = \sum_{j=0}^{n} \{\mathcal{D}(n,j) \times R(n-j,r-1)\}$$

The base conditions for this equation are: $R(n,1) = \mathcal{D}(n,n), R(0,r) = 1, R(n,0) = 0$.

The probability of receiving all the f FEC repair packets (case 2.a) is $(1-p)^f$. Computation of $P_r(f)$, probability of the event described in (2.b), is explained in detail later in this chapter. We can now express the probability of the second event as: $(1-p)^f + P_r(f)$.

Now, P_2 can be expressed as follows.

$$P_2 = P_{frtx}(f,k,r) \times ((1-p)^f + P_r(f))$$

Computing P_3 In this case, since only f or fewer data packets are lost during the initial transmission of k data packets, retransmission is not triggered. However, as FEC packets are lost, eventually causing c_i to become larger than f, retransmission effects. We can express P_3 as follows.

$$P_3 = \sum_{i=1}^{f} \{\mathcal{D}(k, k-i) \times P_r(i)\}$$

The summation over i from 1 to f indicates that at least 1 data packet is lost during the initial transmission but no more than f data packets are lost. $P_r(i)$ denotes the probability that at least one of the i repair packets required to recover the periodic frame is received through retransmission because insufficient number of FEC packets are received. The computation of $P_r(i)$ is described below.

Computing $P_r(n), n \leq f$ Recall that according to the hybrid recovery protocol, retransmission effects for each NACK received only after c_i becomes equal to f. In other words, when n repair packets are needed, retransmission will not effect unless $f-n-1$ or more FEC packets are lost. Let the f FEC packets be denoted as $FEC_{f-1}, FEC_{f-2}, \cdots, FEC_0$, where FEC_0 is the FEC packet transmitted last in the PTDD [4].

We now introduce the notion of a "critical" FEC packet and define the term. When n repair packets are needed for recovery, we say that the FEC packet FEC_x, $(0 \leq x \leq f-1)$, is *critical* if and only if $n-x-1$ packets have been received out of the $f-x-1$ FEC packets $(FEC_{f-1} \cdots FEC_{x+1})$ transmitted prior to FEC_x. Clearly, loss of the critical FEC packet, FEC_x, will trigger retransmission. We can express the probability that retransmission is triggered due to loss of FEC_x, denoted $P_{xr}(f, n, x)$, as follows.

$$P_{xr}(f, n, x) = C_{n-x-1}^{f-x-1} \times (1-p)^{n-x-1} \times (p)^{f-n} \times p$$

Since FEC packets are sent at intervals of Δ time units, if retransmission effects due to loss of FEC_x, then for successful recovery, that retransmission has to be received in the remaining $ptdd - (f-x)\Delta$ time units. Therefore, r', the number of retransmission attempts possible within that time, is equal to $\lfloor \frac{ptdd-(f-x)\Delta}{rtt} \rfloor$. We can then express $R_x(f, x, 0)$, the probability that this retransmission be successful, as follows.

$$R_x(f, x, 0) = R_p(r) = 1 - p^{r'}$$

[4]It can be observed that as per this numbering scheme, there are x FEC packets in the PTDD period after FEC_x.

$R_p(r^{'})$ denotes the probability of receiving the retransmitted packet within $r^{'}$ possible retransmissions. Consecutive retransmissions (of the same packet) which are separated by rtt, undergo uncorrelated losses (because rtt is longer than Δ), and they can be modeled using a (1-state) Bernoulli model.

Once retransmission is triggered due to loss of a critical FEC packet, FEC_x, all the following x FEC packets (FEC_{x-1} through FEC_0) also become critical and the loss of each of these FEC packets will also trigger retransmission. For example, if FEC_{x-j}, ($1 \leq j \leq x$), is reported lost triggering retransmission, $R(f,x,j)$, the probability of receiving that retransmitted packet can be expressed as follows.

$$R_x(f,x,j) = R_p(\left\lfloor \frac{ptdd - (f - x + j)\Delta}{rtt} \right\rfloor)$$

Thus, the probability of receiving all the remaining x repair packets (with or without retransmission), denoted $P_x(f,x)$, can be expressed as follows.

$$P_x(f,x) = \prod_{j=1}^{x} \{p \times R_x(f,x,j) + (1-p)\}$$

Note that the term $R_x(f,x,j)$ is multiplied by p because retransmission is not triggered unless the critical FEC repair packet is lost.

Summing up the analysis, we can express $P_r(n)$ as follows.

$$P_r(n) = \sum_{x=0}^{n-1} \{P_{xr}(f,n,x) \times R(f,x,0) \times P_x(f,x)\}$$

It can be observed that the three main components in the computation of $P_{hybrid}(k,f,ptdd)$ viz. P_1, P_2, and P_3 use all the $D(k,0)\cdots D(k,k)$ terms. Additionally, computation of $R(n,r), n \leq k$, a component of P_2, may require any combination of the $D(0,0)\cdots D(k,k)$ terms. Therefore, before starting the computation of $P_{hybrid}(k,f,ptdd)$ we can compute all the $D(0,0)\cdots D(k,k)$ terms only once and store them to avoid duplicate computation, thereby speeding up the computation of P_{hybrid}. Based on the recursive expressions for $D(m,n)$ given in [47], we can use dynamic programming to compute all of $D(0,0)\cdots D(k,k)$ in $O(k^2)$. It can be further observed that we can use dynamic programming to compute all of $j!, 1 \leq j \leq k$, once in $O(k)$ time and store them. This would virtually allow us to compute $P(f,j)$ and C_{n-x-1}^{f-x-1} terms in $O(1)$.

Also, $R(n,r)$ can be computed using dynamic programming. All of $R(i,1), 1 \leq i \leq n$, are already available because by definition $R(i,1) = D(i,i)$. Having computed $R(i,1)$s, any $R(i, 2)$ can be computed in $i + 2$ steps. Continuing in this manner, $R(n, r)$ can be computed in $O((n + 1)n(r - 2))$ time when $r \geq 2$.

Computing PTDD and the number of FEC packets So far we have described how to compute the probability, P_{hybrid}, of recovering a periodic frame within a known PTDD period using a specified number of FEC packets and retransmissions under given network conditions. Our main goal is to choose the minimum PTDD and the minimum number of FEC packets such that P_{hybrid} meets the desired recovery threshold, θ, (in our experiments θ is set to 0.95), and at the same time results in minimal bit overhead. Increasing PTDD or adding more FEC packets increases recovery probability as well as bit overhead. Increasing PTDD beyond a point where there is little temporal redundancy between periodic frames is clearly not advantageous. Therefore a certain $ptdd_{max}$ should be used as an upper bound on PTDD. We set $ptdd_{max}$ to 1 second, and the maximum number of allowed FEC packets, f_{max}, is chosen not to exceed the number of data packets of the periodic frame.

Note that adding f FEC packets of b bits results in a bit overhead equal to $f \times b$ bits and PTDD i results in overhead equal to $g(i)$ bits ($g(i)$ is defined in Section 3.3). Based on this, we have a function $overhead(k, f, ptdd)$ that estimates the total bit overhead associated with the periodic frame comprised of k data packets, f FEC packets and having PTDD equal to $ptdd$. Note that the total bit overhead also includes the overhead due to retransmissions. If ER denotes the expected number of retransmissions that may effect during the PTDD, and b denotes the packet size in bits, then the bit overhead due to retransmissions will be equal to $ER \times b$. Now, the problem of finding the minimum PTDD and the minimum number of FEC packets can be formulated as an optimization problem which finds $ptdd$, and f, such that $overhead(k, f, ptdd)$ is the smallest possible with constraints: $P_{hybrid}(k, f, ptdd) \geq \theta$, $\delta_f \leq ptdd \leq ptdd_{max}$ and $0 \leq f \leq f_{max}$. Clearly, we can solve this problem by examining all possible combinations of $ptdd$ and f that satisfy the constraints. Note that since $ptdd$ is an integral multiple of δ_f, the frame interval, there are only a finite number of possible combinations satisfying the given constraints.

In the exhaustive hybrid recovery algorithm presented above, the sender estimates PTDD that can satisfy the given threshold of the recovery probability of a periodic frame and at the same time results in minimal bit overhead. Such estimation can be effective in network environments where recent history can be used to accurately predict events in the near future or when traffic characteristics are known a priori as in a diffserv environment.

3.4 Variants of exhaustive hybrid RESCU

In order to measure the performance improvement achieved by a hybrid recovery scheme, we create two variants of exhaustive hybrid RESCU viz. *FEC-only RESCU* and *retransmission-only RESCU*. FEC-only RESCU uses only FEC packets to recover periodic frames while retransmission-only RESCU uses only retransmission to recover periodic frames.

Let $P_{fec}(k, f)$ denote the probability that a periodic frame comprised of k data packets and f FEC packets can be recovered using FEC alone. The component P_1 in P_{hybrid}, described in Section 3.3 denotes this probability. Therefore, $P_{fec}(k, f) = P_1$. Thus, the problem of finding the minimum f such that $P_{fec}(k, f) \geq \theta$ can be solved by examining all possible $f \leq f_{max}$. Knowing f, we can set *ptdd* to be equal to $\left\lceil \frac{f\Delta + \delta_f}{\delta_f} \right\rceil$.

A retransmission-only variant of exhaustive hybrid RESCU can be easily created by setting the number of FEC packets f to 0. Let $P_{retx}(k, ptdd)$ denote the probability that a periodic frame can be recovered under such a retransmission-only scheme. Then we can say that $P_{retx}(k, ptdd) = P_{hybrid}(k, 0, ptdd)$. Since $f = 0$, it can be further observed that the component P_3 in $P_{hybrid}(k, 0, ptdd)$ is 0, $P_1 = D(k, k)$, and $P_2 = \sum_{i=1}^{k} \left\{ \mathcal{D}(k, k-i) \times R(i, \left\lceil \frac{ptdd}{rtt} \right\rceil) \right\}$. We can find *ptdd* for retransmission-only RESCU in the same way as in the exhaustive hybrid algorithm.

3.5 Lazy hybrid RESCU

The main difficulty in determining PTDD in the exhaustive hybrid scheme lies in predicting the "optimal" degree of contribution that FEC and retransmission, each can make to the future recovery of periodic frames and its incurred bit overhead. Under highly varying network conditions, such prediction can be difficult.

Retransmission is necessary only when the actual packet losses in a periodic frame give a clear indication that it will not be possible to recover that periodic frame through FEC packets alone. Thus, its operation is reactive in nature. However, in the exhaustive scheme, the PTDD also depends on the estimation of how many times the sender will have to retransmit a lost packet (because of its possible repeated losses). Thus, like FEC, retransmission also has a "proactive" contribution to the bit overhead in the exhaustive scheme because PTDD is set large enough to accommodate the retransmission delays caused by the predicted number of retransmissions. As mentioned earlier, such estimation may be effective only under accurately predictable network conditions. Large PTDD reduces compression efficiency and this overhead is incurred even if retransmission does not occur.

Retransmission is most effective when applied in a truly "reactive" fashion where retransmission overhead is incurred only at the time of retransmission. In this section, we develop a hybrid protocol where PTDD is adjusted for retransmission only when retransmission happens. During normal transmission, minimal proactive recovery is performed. This technique is best suited under a network environment where packet loss characteristics may show a high degree of variability. Additionally, this protocol greatly reduces computational complexity for finding PTDD because it does not require the computation of the recovery probability.

3.5.1 Protocol

The protocol runs on the system architecture described in Section 3.1. Before transmitting a periodic frame F_i, the adapter determines the number of FEC packets, f_i, and Δ required to recover that periodic frame, and sets the next PTDD period to $\left\lceil \frac{f\Delta+\delta_f}{\delta_f} \right\rceil$. Should retransmission occur for a periodic frame F_i, the next frame F_j encoded after that retransmission uses F_i as its reference frame (instead of its immediately preceding periodic frame F_k) and F_j becomes a new periodic frame. The rationale behind this adjustment is that F_k must be damaged because of the error propagation from unrecovered F_i, and referencing F_k will continue error propagation. Since the lost packets of F_i are retransmitted before the transmission of F_j, we can optimistically assume that F_i will be recovered before the decoding of F_j. The next PTDD period after the transmission of F_j is recomputed according to f_j (the number of FEC packets for F_j). Figure 3.1 illustrates how PTDD is adjusted based on retransmission. If the time difference between F_i and F_j becomes too large, then the temporal redundancy between the two frames is minimal. Thus, we set a limit on the time difference for using motion-compensated coding, $ptdd_{max}$, to be 1 second. If the difference is larger than this limit, then F_j is intra-coded.

The transmitter maintains a counter c_i for a periodic frame F_i to record the number of NACKs received for F_i. Retransmission of lost data packets of F_i effects only when all of the following conditions are met: (1) c_i is larger than f_i, (2) F_i was transmitted less than 1 second earlier, and (3) if F_{intra} is the intra-frame sent most recently, F_i was transmitted

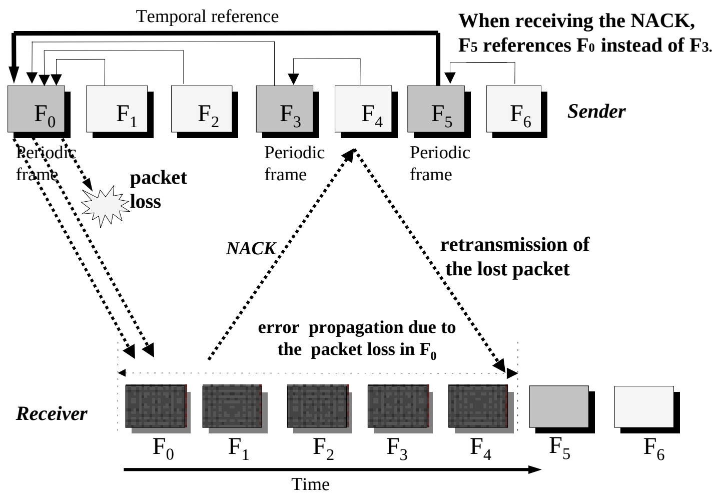
Temporal reference
When receiving the NACK, F5 references F0 instead of F3.
F0
F1
F2
F3
F4
F5
F6
Sender
Periodic frame
packet loss
Periodic frame
Periodic frame
NACK
retransmission of the lost packet
error propagation due to the packet loss in F0
Receiver
F0
F1
F2
F3
F4
F5
F6
Time

Figure 3.1: An example of PTDD adjustment in the lazy hybrid RESCU

after F_{intra} and no packets of periodic frames sent between F_{intra} and F_i are retransmitted.

It can be observed that this protocol requires more frame buffers at the codecs than the exhaustive protocol. Since the periodic frames to be referenced are determined by feedback from the receiver, the encoder must store all the periodic frames transmitted within the maximum PTDD period allowed by the system ($ptdd_{max}$). Since feedback can be lost, the decoder store any damaged periodic frames received within the $ptdd_{max}$ period.

3.5.2 Computing f_i and Δ

f_i and Δ are computed based on the latest "short-burst" loss characteristics. We define the *short-burst* loss characteristics to be the mean loss rate and mean burst length of packet losses that appear in a loss burst involving only a small number of consecutive packet losses (in our experiments we consider a loss burst involving four or less packets as a short burst). The reason behind using only short-burst characteristics for computing f_i and Δ is that FEC is more effective when packet losses are uncorrelated (near-independent). When packet losses occur in long bursts, retransmission can be a more effective recovery method. The mean loss rate and burst length are computed using a weighted moving average of the sampled data. Since it is hard to predict when long burst losses can occur, the use of FEC to protect against such long burst losses is ineffective as it incurs unnecessarily high bit overhead during a quiescent period.

Using the short-burst loss characteristics, we can compute Δ as described in Section 3.3. f_i is computed as follows. Suppose that the periodic frame F_i consists of k data packets. Then the sender decides to add f_i FEC packets such that the number of packets expected to be received at the receiver, $EX(k, f_i)$, is at least equal to k (so that recovery is possible through FEC alone). When f FEC packets are added to protect k packets of the periodic frame, the expected number of packets received is computed as follows.

$$EX(k, f_i) = \sum_{i=0}^{k} \{i \times D(k,i)\} + \sum_{j=0}^{f_i} \{j \times P(f_i, j)\}$$

$P(f_i, j)$ denotes the probability of receiving exactly j packets out of f_i FEC packets, whose losses are assumed to be uncorrelated. $P(f_i, j)$ can be computed using a (1-state) Bernoulli model as $P(f_i, j) = C_j^{f_i}(1-p)^j(p)^{f_i-j}$. By adding just enough FEC to protect against expected losses minimizes the unnecessary FEC overhead and most of the time the losses are likely to be repaired by FEC alone. Additionally, when the estimate of FEC packets is not sufficient to recover lost packets, retransmission can be used to augment FEC in the recovery process.

Chapter 4

Experimental Result

Loss recovery schemes presented in this thesis aim to improve video quality under lossy Internet environment by focusing on removing error propagation associated with motion-compensated video coding. To study the effectiveness of these schemes, we measure their performance under varying network conditions produced by actual Internet traces. We compare their performance with that of existing solutions like NEWPRED [28] and Intra-H.261 [34] which address the error propagation problem. We use three H.263 anchor video sequences produced by the Telenor H.263 codec, one for each of MPEG-4 class A, B and E tests. The three video sequences are described in Table 4.1.

4.1 Compression efficiency of RESCU

Tables 4.2, 4.3, and 4.4 show the percentage increase in average bits per frame for each video sequence as PTDD increases when RESCU is combined with H.261 (RESCU+H.261). The percentage increases are shown w.r.t the PTDD 1 (i.e, when each frame is encoded using

Video Sequence	Class	Description
container	A	low spatial detail and low amount of motion
news	B	medium spatial detail and medium amount of motion
children	E	hybrid natural and synthetic movements

Table 4.1: Test Video Sequences

PTDD	RESCU+H.261	Periodic frame	Intra frame
1 (bits/f)	18880	18880	88360
2 (%)	6	12	N/A
3 (%)	10	28	N/A
4 (%)	14	32	N/A
5 (%)	18	49	N/A
6 (%)	20	50	N/A
7 (%)	23	64	N/A
8 (%)	26	67	N/A
9 (%)	29	79	N/A
10 (%)	32	81	N/A

Table 4.2: Bit rates per frame of RESCU as PTDD increases using *container*

its immediately preceding frame). As expected, when PTDD is set to one, the average bits per frame of RESCU+H.261 are the same as the average bits per frame of H.261. The tables also show the percentage increase in average bits per periodic frames in RESCU+H.261 (denoted *Periodic frame* in the tables), and the average bits per frame when we fully intra-code every frame (denoted *Intra frame* in the tables). The results indicate that for each increment of PTDD, the compression efficiency of RESCU drops about 3% to 5% in *container*, and about 2% to 12% in *news* and *children*. From the tables, we can see that when more motion is present, the bit overhead of RESCU increases. We will see from the results in Section 4.2.2 that RESCU achieves the best quality and bit rate tradeoffs compared to all the techniques we tested. In addition, since the bit overhead of periodic frames is much less than I-frames, there exists much advantage in exploiting temporal redundancy between two periodic frames than coding them as I-frames.

4.2 Simulation Setup

4.2.1 Internet transmission

To emulate the loss behavior encountered in the Internet, we collected traces of 12 minute long video transmission over a trans-pacific connection every hour for a two week period. The frame rate is set to 10 frames per second. Full-search motion estimation and the image size of CIF (352×288 color) is used for all experiments. The default quantization step of 8 is used. A video frame is first compressed using a RESCU codec which is built using

PTDD	RESCU+H.261	Periodic frame	Intra frame
1 (bits/f)	19344	19344	90760
2 (%)	10	26	N/A
3 (%)	19	47	N/A
4 (%)	26	61	N/A
5 (%)	33	76	N/A
6 (%)	36	85	N/A
7 (%)	42	96	N/A
8 (%)	46	103	N/A
9 (%)	52	110	N/A
10 (%)	52	119	N/A

Table 4.3: Bit rates per frame of RESCU as PTDD increases using *news*

PTDD	RESCU+H.261	Periodic frame	Intra frame
1 (bits/f)	34856	34856	111600
2 (%)	12	25	N/A
3 (%)	20	41	N/A
4 (%)	26	51	N/A
5 (%)	30	58	N/A
6 (%)	38	70	N/A
7 (%)	37	75	N/A
8 (%)	42	81	N/A
9 (%)	50	90	N/A
10 (%)	52	93	N/A

Table 4.4: Bit rates per frame of RESCU as PTDD increases using *children*

No.	Avg. Packet Loss Rate (%)	Avg. RTT (ms)
1	0.5	245
2	5.03	268
3	8.20	255
4	8.41	261
5	8.91	355
6	12.23	313
7	13.84	276
8	14.04	800
9	15.35	572
10	17.45	976

Table 4.5: Average statistics for 10 different transmission traces

an implementation of H.261, and then is packetized into approximately 256-byte packets such that the individual packets contain an integral number of macroblocks.

A wide range of round trip times (RTT) (from about 250 ms to over 1000 ms), and loss rates (from 0.5% to 18%) are observed. The mean loss burst length is less than 3. Most of loss bursts are short. Occasionally, long loss bursts involving more than 100 packets are also seen. Out of about 200 traces obtained, we select 10 representative traces covering a spectrum of (mean) loss rate and round trip time. Table 4.5 summarizes the average traffic characteristics observed in the selected transmission traces.

4.2.2 Trace-driven simulation

We extract the profile information of each trace which consists of statistics on the loss rate, round trip delays, and the instances of loss bursts of lengths from 1 to over 200, observed for every non-overlapping 10 second segment. Each trace yields 72 profile informations to form one error model for a simulation transmission experiment. Each error model is applied to construct a UCB/VINT network simulator *ns* setup. In the simulator, an error model obtained from a trace controls transmission latency and the number of packets being dropped for a simulated 10 second period. This arrangement allows the simulator to follow the profile information of the corresponding 10 second period in that trace.

Video codecs (NEWPRED, RESCU and Intra-H.261) built by modifying an implementation of H.261, and the error models of the selected traces are integrated with UCB/VINT network simulator *ns*. Figure 4.1 illustrates the simulation setup which imple-

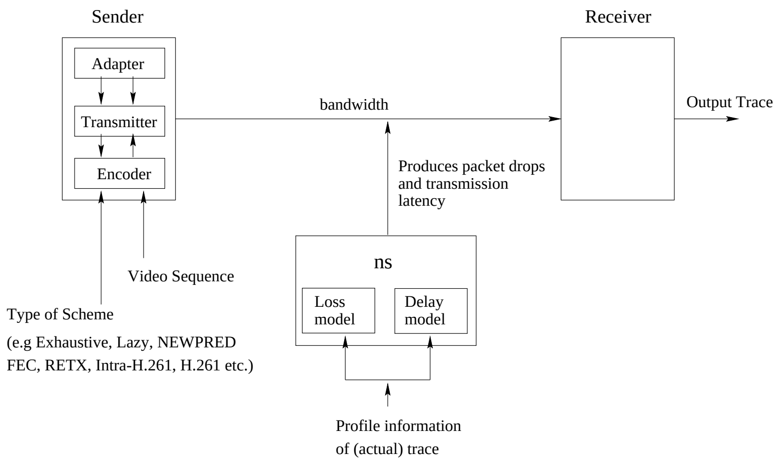
Sender
Adapter
Transmitter
Encoder
bandwidth
Receiver
Output Trace
Produces packet drops and transmission latency
ns
Loss model
Delay model
Video Sequence
Type of Scheme
(e.g Exhaustive, Lazy, NEWPRED FEC, RETX, Intra-H.261, H.261 etc.)
Profile information of (actual) trace

Figure 4.1: Simulation Setup

ments the system architecture described in Section 3.1. The transmitter packetizes compressed video frames and passes the packetized sequence to `ns` to produce packet drops and transmission latency. At the receiving end, a trace is generated which records all the received packets and their received time. This output trace is analyzed to measure the end video quality using an off-line decoder. We conduct 10 *ns* simulation runs, each with a different error model.

4.3 Results of performance comparison

In this section, we report the result of performance comparison of hybrid RESCU, FEC, RETX, NEWPRED, H.261, and Intra-H.261.[1] Results of the comparison are presented in the following order for each video sequence.

- Comparison of the performance of lazy hybrid RESCU (`lazy`) with that of FEC-only RESCU (`FEC`) and, retransmission-only RESCU (`RETX`).
- Comparison of the performance of `lazy` with that of exhaustive hybrid RESCU (`exhaustive`).
- Comparison of the performance of `lazy` with that of Intra-H.261, NEWPRED, and H.261.

[1]Since the performance of Intra-H.261 depends on the available bandwidth, we test the performance of Intra-H.261 at a bandwidth matching approximately the maximum bandwidth used by any of the dynamic hybrid RESCU schemes.

Next, we present the results of our tests for the three video sequences mentioned and discuss our observations in detail.

4.3.1 Container

Lazy, RETX, FEC Figures 4.2 and 4.3 show the average PSNR and bit rate under `lazy`, `FEC` and `RETX` schemes. Up to about 13% loss rate, the average PSNRs of `lazy` and `FEC` are very similar while that of `RETX` is up to 1 dB lower. The lower quality under `RETX` can be attributed to the longer repair latency associated with retransmission. This is more pronounced at higher loss rates since more packets lost result in more retransmissions which can be lost as well, making `RETX` very sensitive to network latency. The sensitivity of video quality to network latency in `RETX` is clearly seen when the average PSNR of `RETX` around 15% loss rate is about 2dB higher than that around 14% loss rate in spite of the higher loss rate. The RTT in the trace with 15% loss rate is about 200 ms shorter than that in the trace with 14% loss rate. `FEC` and `lazy` both have similar average PSNR at almost all loss rates. The slightly lower average PSNR of `lazy` at high loss rates is also because of more use of retransmission to augment FEC in the recovery process and the sensitivity of retransmission to RTT.

The bit rate of `lazy`, shown in Figure 4.3, is the least among the three recovery schemes, consistently (up to 20kbps) lower than that of `FEC`. This can be attributed to: (1) at any loss rate `FEC` adds more FEC packets than `lazy`, and (2) when recovery is not possible through FEC alone, `lazy` can retransmit repair packets whereas `FEC` has to use an I-frame which consumes much more bandwidth than an inter-coded P-frame.

At low loss rates, few packet losses are expected, and therefore, only a few FEC packets need be added. Hence `lazy` matches the bit rate of `RETX`, as repair overhead is incurred only when retransmission happens. However, in `RETX` PTDD has to be long enough to accommodate long RTTs. Additionally, as loss rate increases PTDD has to be adjusted to accommodate more retransmission attempts. Although retransmission happens only when losses actually occur, the reduced compression efficiency due to increased PTDD shows up in increased bit rate of `RETX` at high loss rates.

Lazy, exhaustive Figures 4.4 and 4.5 show the average PSNR and bit rate at different loss rates under `exhaustive` and `lazy` schemes. The average PSNRs of both the schemes are very similar at all loss rates. Recall that although `exhaustive` tries to minimize bit overhead (due to FEC and PTDD) needed to protect the periodic frame, it has to satisfy the desired recovery probability. On the other hand, `lazy` adds FEC only to protect against expected losses and uses retransmission to augment FEC in case of unanticipated losses. Therefore, even though `exhaustive` shows slightly better quality at high loss rates, it has a somewhat higher bit rate.

Lazy, Intra-H.261, H.261, NEWPRED Figures 4.6 and 4.7 show the average PSNR and bit rate at different loss rates under `H.261`, `NEWPRED`, `Intra-H.261` and `lazy`. The average PSNR of `H.261` falls rapidly (to as much as 12dB lower) as loss rate increases because of error propagation. Even at the lowest loss rate [2], its average PSNR is lower than that under `NEWPRED` and `lazy`. This shows that even a few packet losses can affect H.261. `Intra-H.261` shows a linear degradation in quality as loss rate increases and significantly higher bit rate than those of other schemes. `NEWPRED` on the other hand can quickly recover from packet losses when RTT is small enough. But at higher loss rates, only a small number of frames are received correctly and only they are used as reference frames. However, even in low motion video sequences like *container*, motion-prediction is effective only up to a certain time after that frame. This is the reason for high bit rate of `NEWPRED` at high loss rates and small RTT. When the RTT is long, `NEWPRED` shows poorer error resilience resulting in its average PSNR up to 4 dB lower than that of `lazy`. `lazy` consistently gives better average PSNR than the other three schemes and lower bit rates than `NEWPRED` and `Intra-H.261`.

4.3.2 News

Lazy, RETX, FEC Figures 4.8 and 4.9 show average PSNR and bit rate at different loss rates under `FEC`, `RETX` and `lazy` schemes. The relative video qualities of the three schemes are similar to that seen for the *container* video sequence and similar arguments apply. However the drop in the average PSNR, and the increase in bit rates as loss rate increases are steeper than those seen for *container*. Since there is lesser temporal redundancy among frames (due to more motion), each individual frame coded using motion compensation contains more information, and results in higher bit rates. Also, the loss of packets further reduces the quality of their individual frames.

[2]The lower quality of Intra-H.261 for this loss rate is because for a target bit rate there is a limit on the maximum video quality it can sustain.

It is interesting to observe the bit rates of the three schemes: `lazy` gives up to 20 kbps less bit rates than `FEC` and up to 50 kbps less bit rates than `RETX`. Note that in `RETX`, for retransmission to be made feasible, PTDD has to be at least as long as one RTT delay. For this video sequence long PTDD is costlier than the continuous overhead due to FEC packets in `FEC` and `lazy`, which is the main reason for the clear difference in the bit rate of `RETX` and the other two schemes. The lower bit rate of `lazy` compared to `FEC` is again due to the effective use of retransmission.

Lazy, exhaustive Figures 4.10 and 4.11 show the performance of `exhaustive` and `lazy`. Recall that `exhaustive` tries to minimize the bit overhead incurred in protecting a periodic frame while satisfying desired threshold of recovery probability. As a long PTDD (to accommodate retransmission repair latency) for this video sequence is costlier than the continuous overhead due to FEC packets, `exhaustive` relies almost exclusively on FEC, and therefore, it has a performance similar to that of `FEC` shown in Figures 4.8 and 4.9. However, the effective use of retransmission in `lazy` results in lower bit overhead.

Lazy, Intra-H.261, H.261, NEWPRED Figures 4.12 and 4.13 show the performance of `H.261`, `NEWPRED`, `Intra-H.261` and `lazy`. `lazy` shows better average PSNR than all the other schemes. The average PSNR of `H.261` drops very quickly as the loss rate increases. `lazy` shows up to 2 dB higher PSNR than that under `NEWPRED` but the bit rate is much lower in all but the instances with high RTT. `Intra-H.261` shows 2-3 dB lower PSNR in spite of 35%-40% higher bit rate. The bit rate of `lazy` and `NEWPRED` in *news* has increased quite a bit (about 10%) from that in *container*. This is because these techniques allow the time distance between a frame and its temporally dependent frame to be larger than one frame interval, and thus more motion present in the input video sequence significantly reduces compression efficiency. However, for video sequences with a medium degree of motion such as *news*, `lazy` still maintains the highest video quality with only a small amount of bit overhead even under high loss rates.

4.3.3 Children

Lazy, RETX, FEC Figures 4.14 and 4.15 show the performance of `FEC`, `RETX` and `lazy`. This video sequence contains a high degree of both natural and synthetic movements. In this video sequence, neighboring frames have a low degree of temporal redundancy. The average

PSNRs of all the schemes drop quickly as the loss rate increases. We can say that for the loss rates up to 8%, their average PSNRs are similar. However. as loss rate increases beyond 8%, `FEC` outperforms `lazy` both in terms of the average PSNR and bit rate. This is because of `lazy`'s reliance on retransmission. When little temporal redundancy is present among neighboring video frames, RESCU incurs high bit overhead for retransmission because use of retransmission requires PTDD larger than retransmission delays. In `FEC`, PTDD is generally much shorter than that in `RETX` and `lazy`.

Lazy, exhaustive Figures 4.16 and 4.17 show the performance of `exhaustive` and `lazy`. As a long PTDD (to accommodate retransmission repair latency) for this video sequence is costlier than the continuous overhead of FEC, `exhaustive` relies almost exclusively on FEC, and therefore, it has a performance similar to that of `FEC` discussed above, both in terms of average PSNR and bit rate.

Lazy, H.261, Intra-H.261, NEWPRED Figures 4.18 and 4.19 show the performance of `H.261`, `NEWPRED`, `Intra-H.261`, and `lazy`. As expected, the average PSNR drops quickly in `H.261` as the loss rate increases. Despite much steeper drop in video quality than in *news* and *container*, `lazy` still shows a better performance both in terms of the average PSNR and bit rate when compared to `NEWPRED` and `Intra-H.261`.

4.4 Adaptiveness of hybrid RESCU

To examine the adaptiveness of lazy hybrid RESCU, we run the scheme over a trace where network traffic shows high variations. In Figure 4.20, we plot the result of the experiment performed using *container*. The first graph shows the average PSNR over every 5 frame period (i.e., every half second period), the second graph shows the loss percentage of each frame, and the third graph shows the average bit rate over every 5 frame period.

The adaptiveness of lazy hybrid is clearly visible when it adapts the amount of repair overhead to maintain high quality under varying network conditions. From the first graph, we observe that the video quality of the RESCU scheme drops when packet loss occurs. However, it immediately bounces back and generally sustains good quality with PSNRs between 35dB and 40dB. Around frames 200-450, frames 700-1350, frames 2200-2400, and frames 2600-2800, the trace experiences high packet losses. During these times, we observe that the bit rate increases beyond 225 kbits/sec which is the results of more repair traffic. During the other times, the bit rate drops to around 180 kbits/sec which is approximately the same bit rate as H.261. This is because during heavy loss periods, FEC packets are not enough to recover from losses, and retransmission is used more often. Since during quiescent periods, retransmission does not effect and only a small number of FEC packets are added, the bit overhead drops to minimum. These results combined with the results presented in the previous sections indicate that lazy RESCU is able to adapt to varying network conditions to minimize bit overhead while sustaining good video quality.

Increased bit rates during lossy congested period will only aggravate congestion. The intent of our work is to show the effect of recovery. For a more practical solution, any loss recovery scheme has to be combined with a congestion control mechanism. When combined with a congestion control mechanism, RESCU has to increase the ratio of repair traffic over data traffic. Since RESCU can achieve a very good tradeoff of video quality over bit overhead, RESCU can be a recovery mechanism to be used with congestion control. The performance of RESCU under a congestion control mechanism needs further study however.

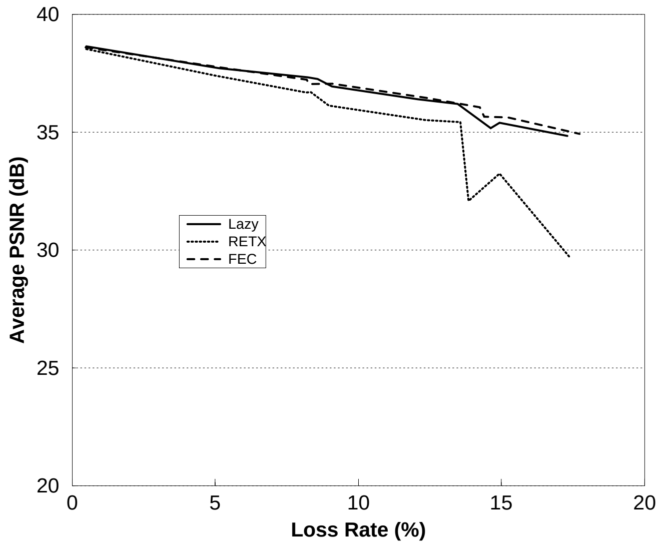

Figure 4.2: The average PSNR under `lazy` , `FEC`, and `RETX` for video sequence *container*

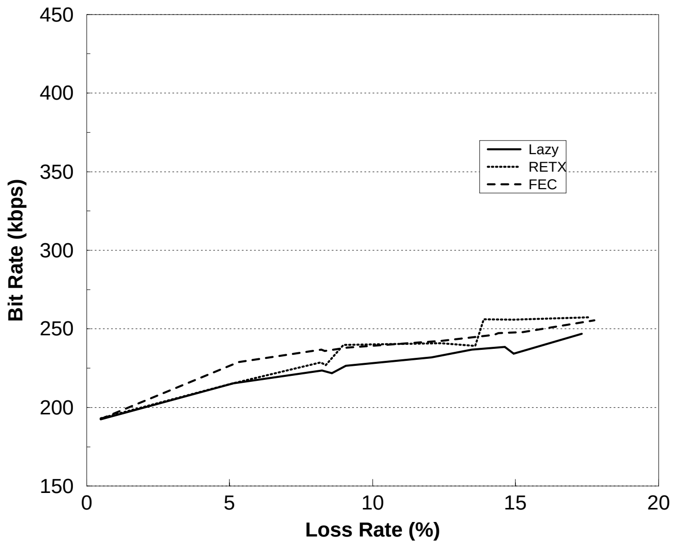

Figure 4.3: The average bit rate under `lazy`, `FEC`, and `RETX` for video sequence *container*

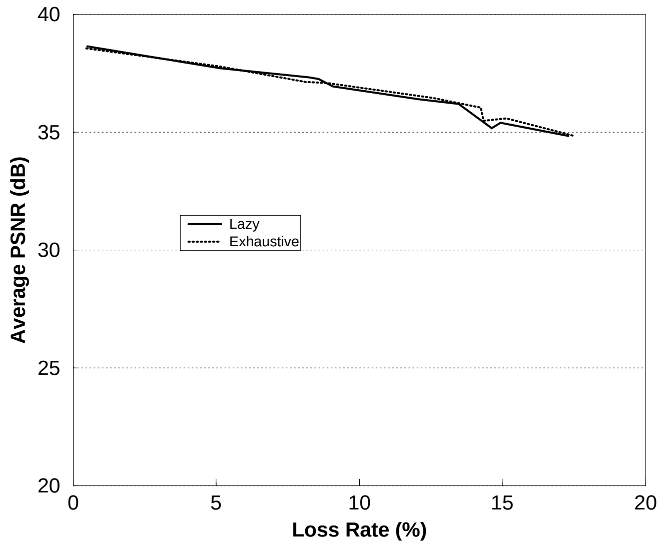

Figure 4.4: The average PSNR under `lazy`, and `exhaustive` for video sequence *container*

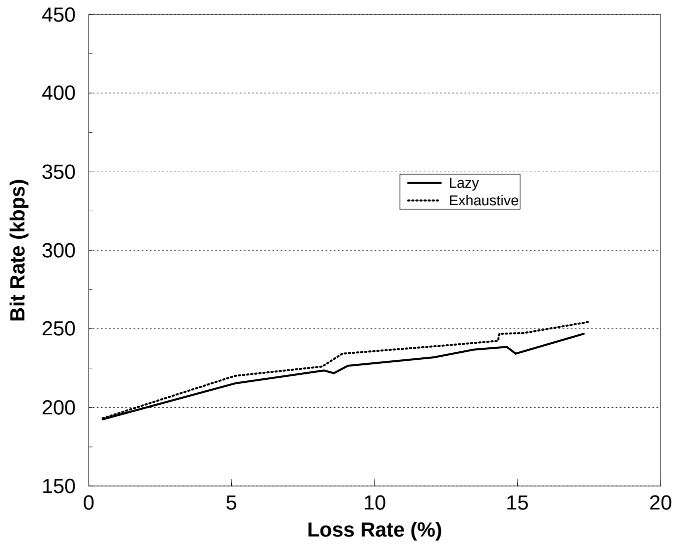

Figure 4.5: The average bit rate under `lazy`, and `exhaustive` for video sequence *container*

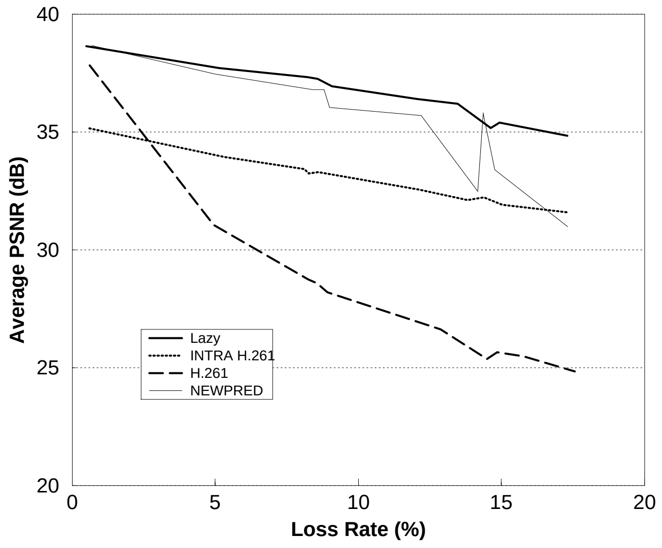

Figure 4.6: The average PSNR under `lazy`, `H.261`, `Intra-H.261`, and `NEWPRED` for video sequence *container*

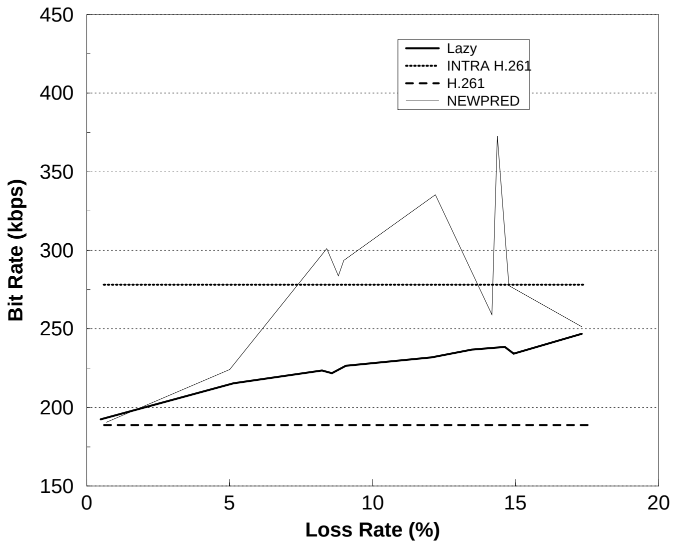

Figure 4.7: The average bit rate under `lazy`, `H.261`, `Intra-H.261`, and `NEWPRED` for video sequence *container*

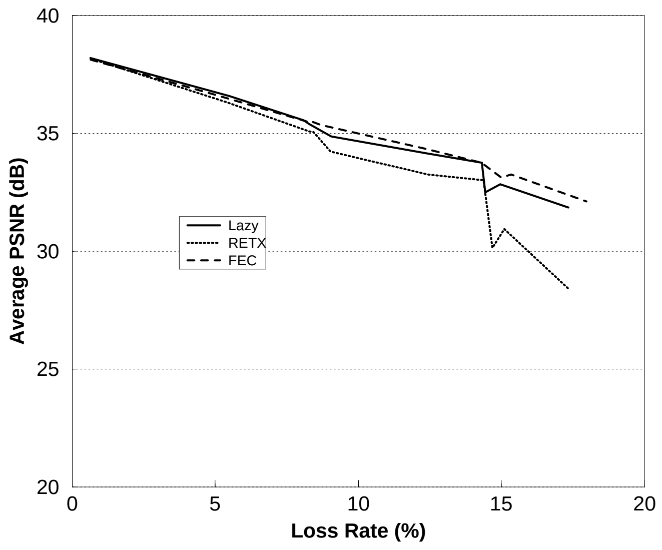

Figure 4.8: The average PSNR under `lazy`, `FEC`, and `RETX` for video sequence *news*

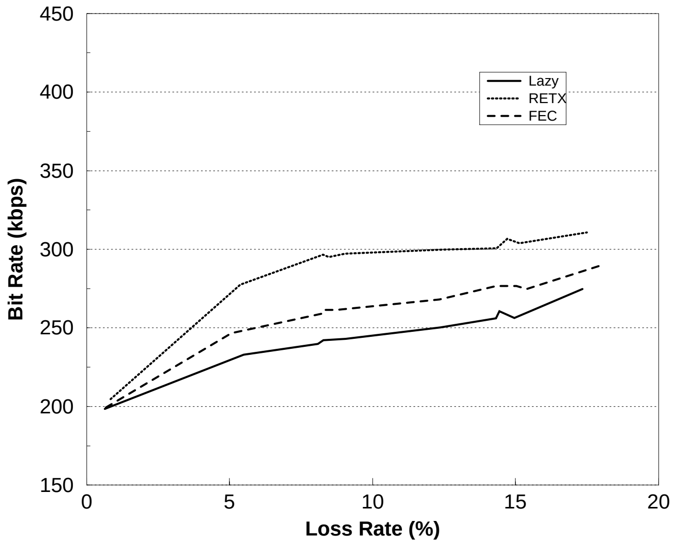

Figure 4.9: The average bit rate under `lazy`, `FEC`, and `RETX` for video sequence *news*

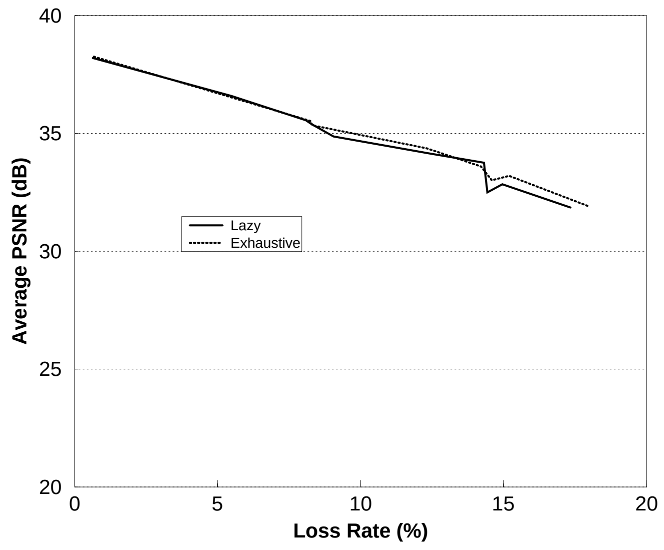

Figure 4.10: The average PSNR under `lazy`, and `exhaustive` for video sequence *news*

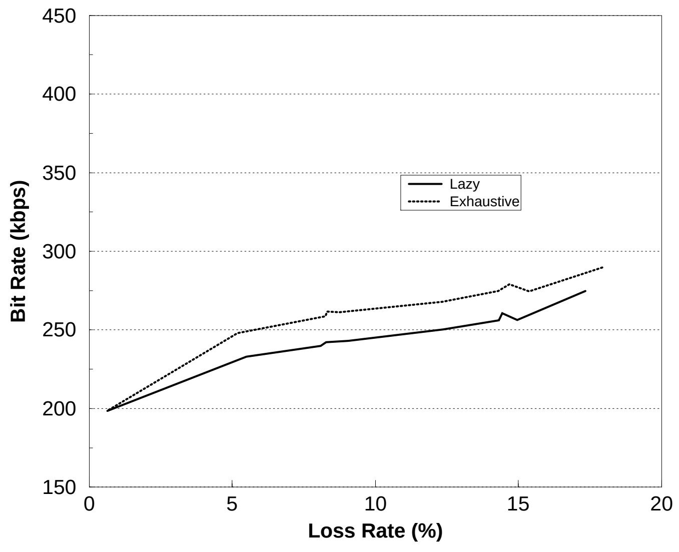

Figure 4.11: The average bit rate under `lazy`, and `exhaustive` for video sequence *news*

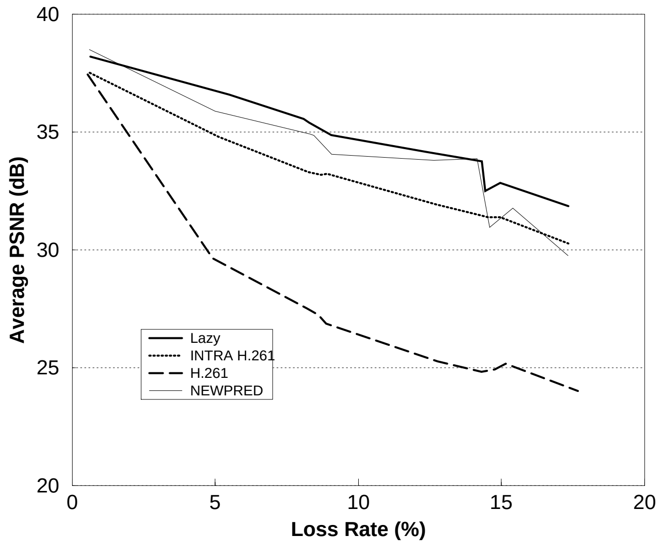

Figure 4.12: The average PSNR under `lazy`, `H.261`, `Intra-H.261`, and `NEWPRED` for video sequence *news*

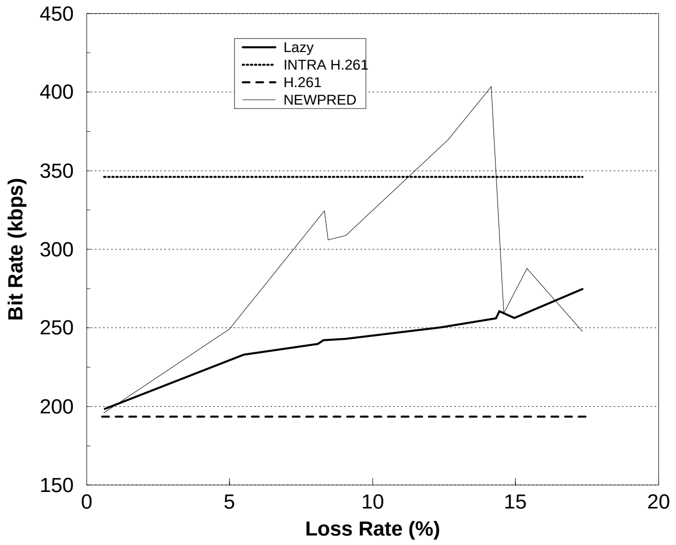

Figure 4.13: The average bit rate under `lazy`, `H.261`, `Intra-H.261`, and `NEWPRED` for video sequence *news*

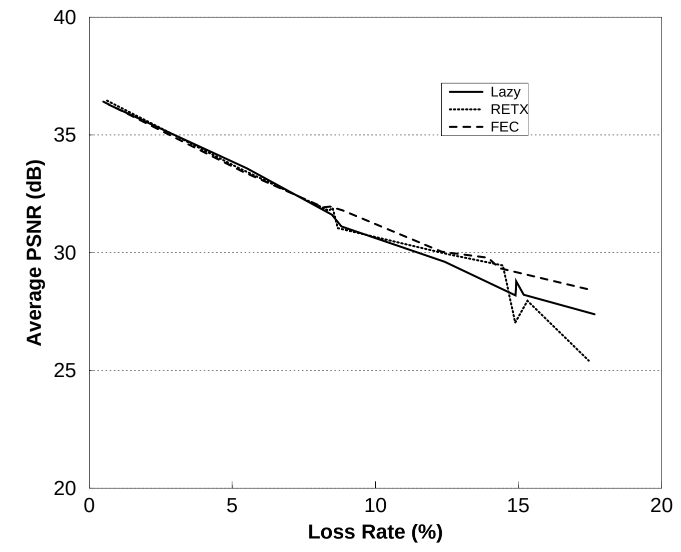

Figure 4.14: The average PSNR under `lazy`, `FEC`, and `RETX` for video sequence *children*

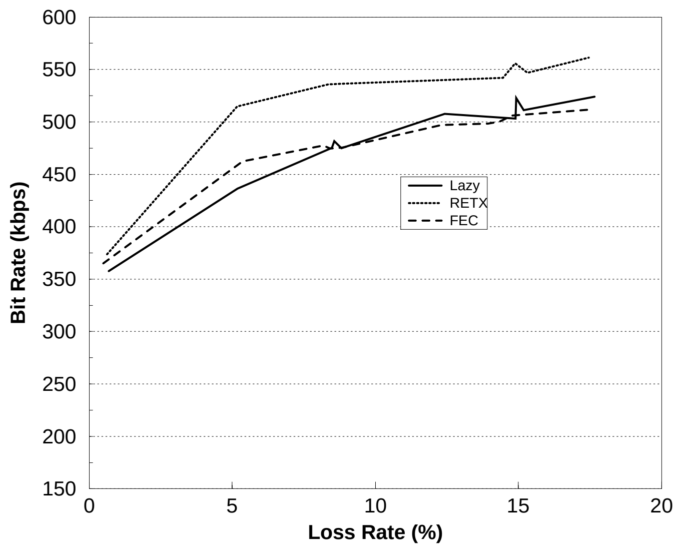

Figure 4.15: The average bit rate under `lazy`, `FEC` and `RETX` for video sequence *children*

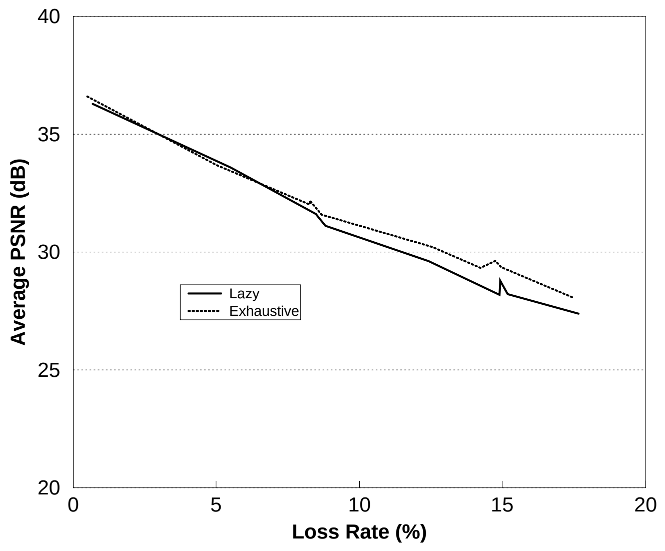

Figure 4.16: The average PSNR under `lazy`, and `exhaustive` for video sequence *children*

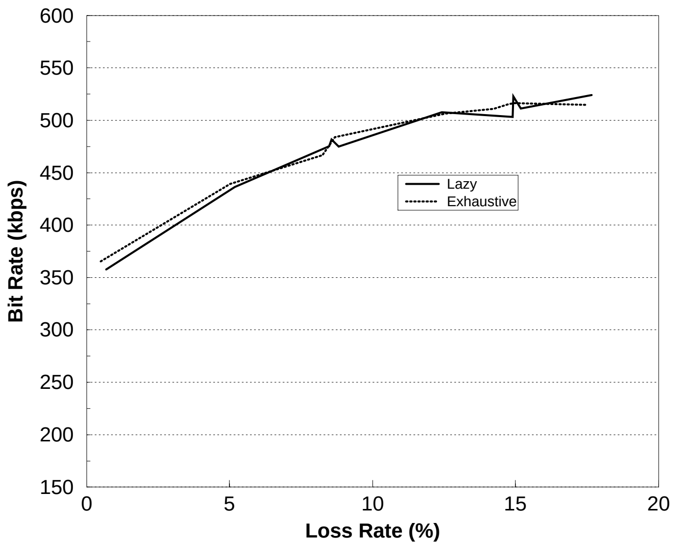

Figure 4.17: The average bit rate under `lazy`, and `exhaustive` for video sequence *children*

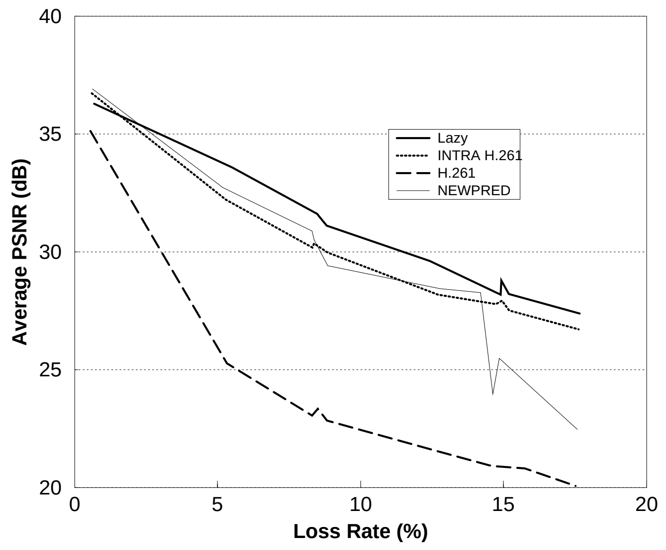

Figure 4.18: The average PSNR under `lazy`, `H.261`, `Intra-H.261`, and `NEWPRED` for video sequence *children*

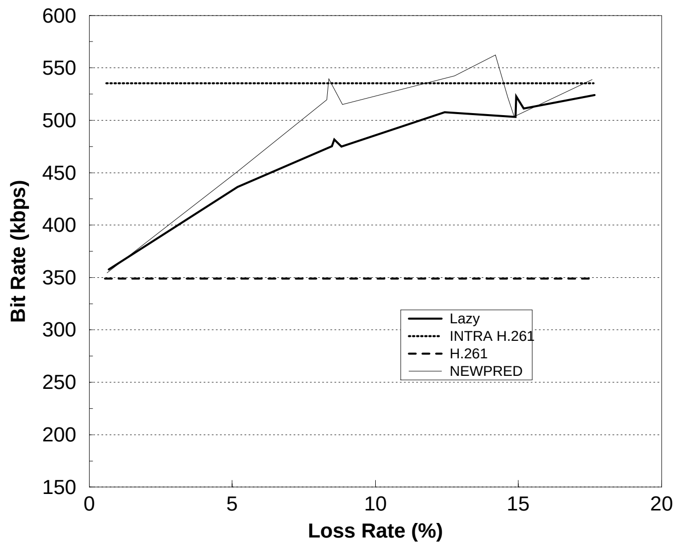

Figure 4.19: The average bit rate under `lazy`, `H.261`, `Intra-H.261`, and `NEWPRED` for video sequence *children*

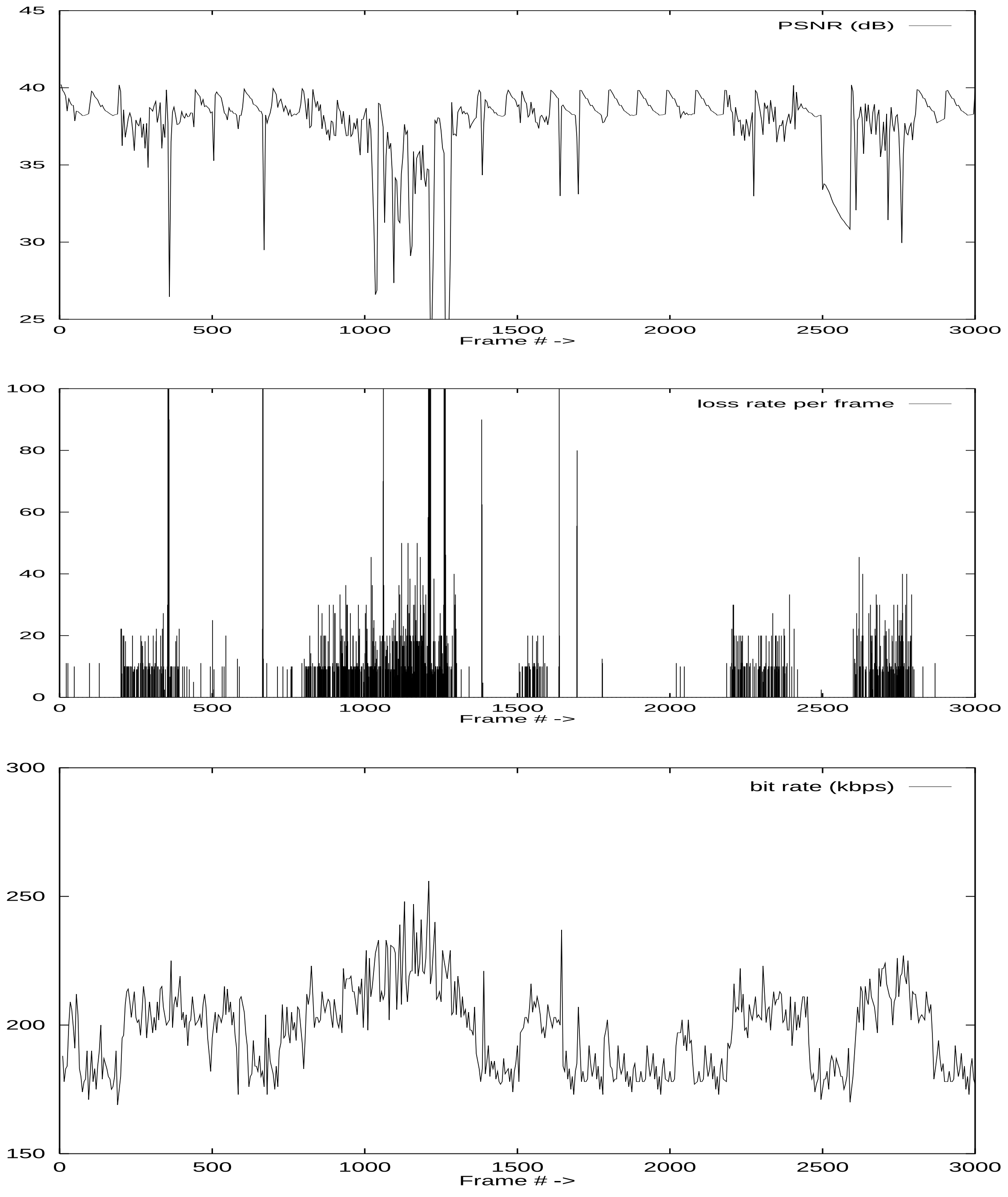

Figure 4.20: The performance of Lazy hybrid RESCU under varying network conditions

Chapter 5

Prototype

In this chapter we describe the implementation of a prototype video transmission system comprised of one sender and one receiver. The sender transmits live, H.263+ [22] compressed video to the receiver using RTP/UDP/IP protocols. Lazy hybrid RESCU algorithm, described in Section 3.5 is employed for loss recovery. This prototype system has been developed by extending the MASH toolkit [17]. This prototype has been successfully implemented and can be used for real video transmission over a network. We first present a brief overview of the H.263+ video standard, RTP/RTCP protocols and the MASH toolkit, followed by a description of the system architecture, and the detailed system operation.

5.1 Introduction

5.1.1 H.263+ video standard

In chapter 1.1, we presented an overview of the H.261 video standard and described RESCU with reference to it. H.261 is one of the earliest international video standards, developed for transmitting video over ISDN lines. Using H.261, video can be transmitted at bit rates of 64kbps or higher. H.263 [21] is a recent video standard aimed at transmitting video at much lower bit rates than H.261. Its basic video coding algorithm is based on H.261, that is, its source coding algorithm is a hybrid of inter-picture prediction to utilize temporal redundancy and transform (DCT) coding of the remaining signal to reduce spatial redundancy. However, unlike H.261 which uses full pixel precision for motion compensation, H.263 uses half pixel precision. This allows it to achieve better compression efficiency (and thus lower bit rates) than H.261.

In 1998, H.263 video standard was revised and sixteen negotiable coding options were included for improved compression performance. This revised version is commonly called the H.263+ video standard. Among the new options included in H.263+, the ones with the biggest impact on the the error resilience of the video content are the slice structured mode, reference picture selection mode, etc.

In general, once a motion vector has been determined for a macroblock in a video frame, it is differentially coded with respect to the motion vectors of adjacent macroblocks belonging to that video frame. A common method of packetizing compressed video frame data is to add an integral number of macroblocks to a packet. However, such fragmentation at macroblock boundaries is not suitable for an underlying packet transport infrastructure because if packets belonging to a video frame are lost, any dependencies across packet boundaries will affect the decoding of other received packets of that video frame.

The slice structured mode supports fragmentation at macroblock boundaries. However, in this mode, once a motion vector has been determined for a macroblock in a slice, it is differentially coded with respect to the motion vectors of adjacent macroblocks belonging to that slice only. This difference between slice boundaries and simple macroblock boundaries allows slice header locations within the encoded bitstream to act as resynchronization points for packet loss recovery and also allows out-of-order slice decoding within a picture.

The reference picture selection mode (RPS) allows the use of an older reference picture rather than the one immediately preceding the current picture. Usually, the last transmitted frame is implicitly used as the reference picture for inter-frame prediction. If the reference picture selection mode is used, the data stream carries information on what reference frame should be used, indicated by the TRP field in the picture header which is the temporal reference number of the reference frame. As noted earlier in Section 1.1, RESCU can be easily supported in H.263+ without requiring any change in the current standard of H.263+. Since RPS allows the reference frame address of a frame to be encoded with that frame, PTDD can be adjusted by simply modifying this address.

Besides the commonly used I (Intra) and P (Inter) picture coding, the H.263+ standard also supports Bi-directional encoding of pictures, called B picture coding. As opposed to no reference frame in case of an I-frame and one previously transmitted (I or P) frame as a reference frame in case of a P-frame, a B-frame has two reference frames - a previously transmitted (I or P) frame and the *next* P frame in the temporal order. B-frames generally result in improved compression efficiency as compared to that of P-frames. Also, B-frames are not used as reference frames for prediction of any other pictures. Hence, they can be discarded if necessary without adversely affecting any subsequent pictures.

5.1.2 RTP/RTCP

Real-time Transport Protocol (RTP) provides end-to-end delivery services for data with real-time characteristics, such as interactive audio and video [48]. These services include payload type identification, sequence numbering, timestamping and delivery monitoring. Applications typically run RTP on top of UDP to make use of its multiplexing and checksum services; both protocols contribute parts of the transport protocol functionality. However, RTP may be used with other suitable underlying network or transport protocols. RTP supports data transfer to multiple destinations using multicast distribution if provided by the underlying network.

RTP itself does not provide any mechanism to ensure timely delivery or provide other quality-of-service guarantees. It does not guarantee delivery or prevent out-of-order delivery, nor does it assume that the underlying network is reliable and delivers packets in sequence. The sequence numbers included in RTP allow the receiver to reconstruct the sender's packet sequence, but sequence numbers might also be used to determine the proper location of a packet, for example in video decoding, without necessarily decoding packets in sequence.

While RTP is primarily designed to satisfy the needs of multi-participant multimedia conferences, it is not limited to that particular application. Unicast interactive (or streaming) video, storage of continuous data, interactive distributed simulation also find RTP applicable.

RTP consists of two closely-linked parts:

- The real-time transport protocol (RTP), to carry data that has real-time properties.
- The RTP control protocol (RTCP), to monitor the quality of service and to convey information about the participant(s) in an on-going session. For example, RTCP reception reports (RR) sent periodically from receiver to the sender, carry feedback on quality of reception such as loss rate, jitter estimate etc.

RTP is based on the principles of application level framing [10] and is intentionally designed to be malleable to provide the information required by a particular application. RTP is often integrated into the application processing rather than implemented as a separate layer. The RTP protocol framework is deliberately not complete. Unlike conventional protocols in which additional functions might be accommodated by making the protocol more general or by adding an option mechanism that would require parsing, RTP is intended to be tailored through modifications and/or additions to the headers as needed. In addition to the RTP protocol specifications, the application should also follow a payload format specification, which defines how a particular payload, such as an audio or video encoding, is to be carried in RTP. Our prototype uses the RTP payload header format applicable to the transmission of video streams generated based on H.263+ [7]. More information about this payload format and how it is used in the prototype is given later in this chapter.

Below we present some of the RTP terminology helpful in understanding the system operation described later in this chapter.

- *RTP timestamp* is a 32-bit unsigned integer carried in the RTP header, reflecting the sampling instant of the first octet in a RTP data packet. If more than one packet contains parts of the same video frame, then all those packets have the same timestamp value.

- *RTP session* is an association among a set of participant(s) communicating with RTP. For each participant, the session is defined by a particular pair of destination transport addresses (one network address plus a port pair for RTP and RTCP). The destination transport address pair may be common for all participants, as in the case of IP multicast, or may be different for each, as in the case of individual unicast network addresses plus a common port pair. In a multimedia session, each medium is carried in a separate RTP session with its own RTCP packets. The multiple RTP sessions are distinguished by different port number pairs and/or different multicast addresses.

5.1.3 MASH tookit

MASH is an extensible multimedia-networking toolkit developed by researchers at the University of California, Berkeley. It uses an object-oriented software architecture based on a split object model, where fine-grained, high-performance C++ objects are glued together into complex macro-objects using OTcl, a simple object oriented scripting language.

Performance critical components like protocol modules, media codecs, and rendering windows are implemented as parts of a class hierarchy in C++ and provide the basic mechanism for data generation, transmission and processing. The split model provides a near seamless integration of C++ and OTcl, that is, an object has methods and properties on either side of the C++/OTcl boundary. The *mash* programming model is event driven. Events are dispatched to objects either from the Tk event dispatcher or via method invocation from OTcl. A *mash* programmer can compose a complex application by arranging the objects into a data flow "pipeline", in which data generated by events is typically processed and passed on to the downstream object. The *mash* toolkit contains a large number of fine-grained building blocks including: audio/video device interface modules, audio/video software based codecs, RTP session objects, video/image rendering/dithering objects, RTP packet buffer pool and so forth.

Since the various building blocks in the toolkit are implemented as parts of a class hierarchy, multiple levels of abstractions can be supported. Additionally, new modules providing enhanced (or application-specific) functionality can be developed by extending the existing building blocks. Some of the existing building blocks important from the point of view of implementing a video application like our prototype are as follows.

- *EncoderModule* provides a number of common functions shared by many types of encoding objects. A video encoder for a particular compression scheme can be implemented by specializing this generic class. For example, an H.261 encoder can be developed by deriving a subclass "H261Encoder" from EncoderModule, where the derived subclass implements the H.261 compression algorithm.

- *Decoder* provides a number of common functions shared by many types of decoding objects. A video decoder for a particular compression scheme can be implemented by specializing this generic class.

- *VideoSession* provides the functionality defining the semantics of an RTP session involving video. Recall that an RTP session encapsulates the network (RTP) and control (RTCP) parts of the communication into a single entity. Thus the VideoSession object has been implemented to provide primitives for sending and receiving RTP/RTCP packets, so that it can be useful in various types of applications involving video transmission using RTP. RTCP packets are generated and consumed by this object, whereas, RTP packets are generated and consumed by video encoder/decoder.

5.2 System architecture

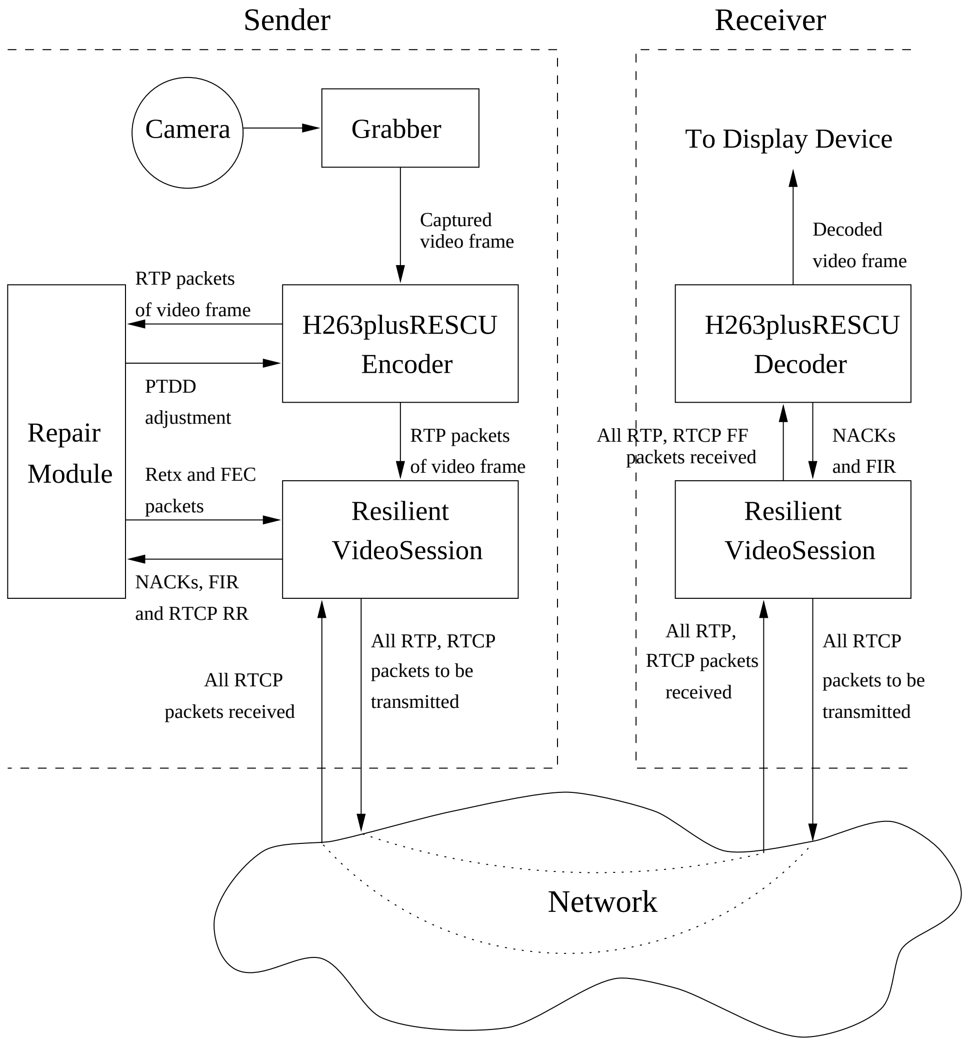

Figure 5.1: Block diagram of the system

Figure 5.1 shows a detailed block diagram of the system, its main components and the various data flows involved. This system, implemented by extending the *mash* toolkit, uses the existing toolkit objects for grabbing video frames, displaying the decoded data, and gathering network statistics etc. Following objects, seen in the block diagram, have been developed specifically for this prototype.

- *H263plusRESCUEncoder* and *H263plusRESCUDecoder* : H263plusRESCUEncoder is an H.263+ encoder, developed by extending the existing EncoderModule class and H263plusRESCUDecoder is a H.263+ decoder developed by extending the Decoder class. The actual compression/decompression code is derived from the source code of a public domain H.263+ codec developed by the Signal Processing and Multimedia Research group at University of British Columbia. The H263plusRESCUEncoder uses both the slice structured mode and the reference picture selection mode described earlier in this chapter. It also supports the use of B-frames. Currently, the slice size (in bytes) and the number of B-frames between consecutive P-frames is configured through the OTcl script of the application.

- *RepairModule* : It is an object developed to implement the lazy hybrid RESCU algorithm for loss recovery. This object is responsible for generating FEC packets for periodic frames, retransmitting lost packets, and adjusting PTDD. The RepairModule maintains a loss model similar to the one described in Section 3.2. This loss model is updated regularly, as RTCP reception reports are received.

- *ResilientVideoSession* : As mentioned in the previous section, the VideoSession object in the *mash* toolkit is responsible for transmitting RTP packets to and receiving them from the network. It also periodically sends the RTCP sender and reception reports. However, certain new (experimental, application-specific) RTCP packets (described later in this chapter) need to be generated and/or consumed by the encoder/decoder objects, instead of the session object. Hence, we implemented ResilientVideoSession object providing such an interface. It is derived from VideoSession, and helps the RepairModule in retransmitting lost packets, using round-trip delay estimates, processing replenishment requests etc.

The grabber captures live video frames from the camera at regular time intervals, which depend on the frame rate chosen by the user (sender) through the user interface. It delivers the captured video frame to the encoder (H263plusRESCUEncoder), which encodes it using the H.263+ video compression algorithm. The encoder performs the tasks of encoding a video frame and packetizing the encoded data. Once all the packets of a frame are generated, it delivers them to the session (ResilientVideoSession) object for transmission to the receiver. If the encoded frame is a periodic frame, FEC packets may need to be generated and lost packets may need to be retransmitted. Since the loss recovery algorithm is implemented by the repair module (RepairModule), the encoder delivers a copy of all the packets of every periodic frame to the repair module object in addition to the session object. Section 5.2.1 describes the generation of RTP packets containing H.263+ video and the associated packet formats in more detail.

For every periodic frame, the repair module determines the number of FEC packets, $NFEC$, needed to protect that frame and appropriate interval, Δ, between consecutive FEC packets. If $NFEC > 0$, the repair module sets a timer having a timeout value of Δ. When the timer expires, it encodes a FEC packet and hands it over to the session object for transmission. This may continue till $NFEC$ packets are encoded and transmitted for a periodic frame. However, if PTDD is adjusted due to retransmission, FEC transmission for a periodic frame may be aborted and the FEC timer will be deactivated. More detailed description of how the repair module performs all these tasks and the packet formats it uses is given in Section 5.2.4.

Received RTP packets containing H.263+ payload or FEC payload are delivered by the session object at the receiver to the decoder (H263plusRESCUDecoder), which decodes the packets and writes the decoded information into a buffer. When a packet of the next frame is received, this buffer is handed over for display. While receiving RTP packets from the sender, the decoder detects packet losses by looking at the gaps in the RTP sequence numbers of received packets. It generates a retransmission request (NACK) for every lost packet (in fact, as will be seen later, a single NACK can report several lost packets). When the decoder sees a repair (FEC or retransmitted) packet of a frame, it tries to recover the frame to which they belong to, using the notion of a *frame context* (explained in Section 5.2.2). Since NACKs as well as retransmitted packets can be lost, the decoder sets a retransmission request timer for each NACK. If a lost packet is not received after several attempts, the decoder stops sending further requests for retransmitting that packet, assuming that the lost packets belong to non-periodic frames. However, this assumption may not always hold; it is possible that the lost packets belong to periodic frames and all the retransmission attempts fail. Since unrecovered lost packets of periodic frames can cause error propagation, the decoder may request the sender for a replenishment (I-frame) in order to stop any error propagation. Additionally, the decoder and the session objects at the receiver cooperate on periodically sending RTCP reception reports containing statistical feedback on the quality of reception. Section 5.2.2 describes the decoder operation and the associated packet formats in greater detail.

All the feedback received at the sender (including RTCP reception reports, retransmission requests, and replenishment requests) is forwarded by the session object to the repair module. In response to a RTCP reception report, the repair module updates its loss model. In response to a replenishment request, it may instruct the encoder to transmit the next frame as an I-frame. In response to a retransmission request, it first checks whether the lost packet belongs to a periodic frame. If the lost packet belongs to a non-periodic frame, the retransmission request is silently ignored. However, if the lost packet belongs to a periodic frame, then depending on how many packet losses have already been reported for that frame the repair module may either choose to ignore the request, or process it. If it retransmits lost packets, then PTDD must be adjusted as per the lazy hybrid RESCU algorithm. Thus, the repair module must inform the encoder of any retransmission that calls for PTDD adjustment. Section 5.2.4 describes the processing of all this feedback information in more detail.

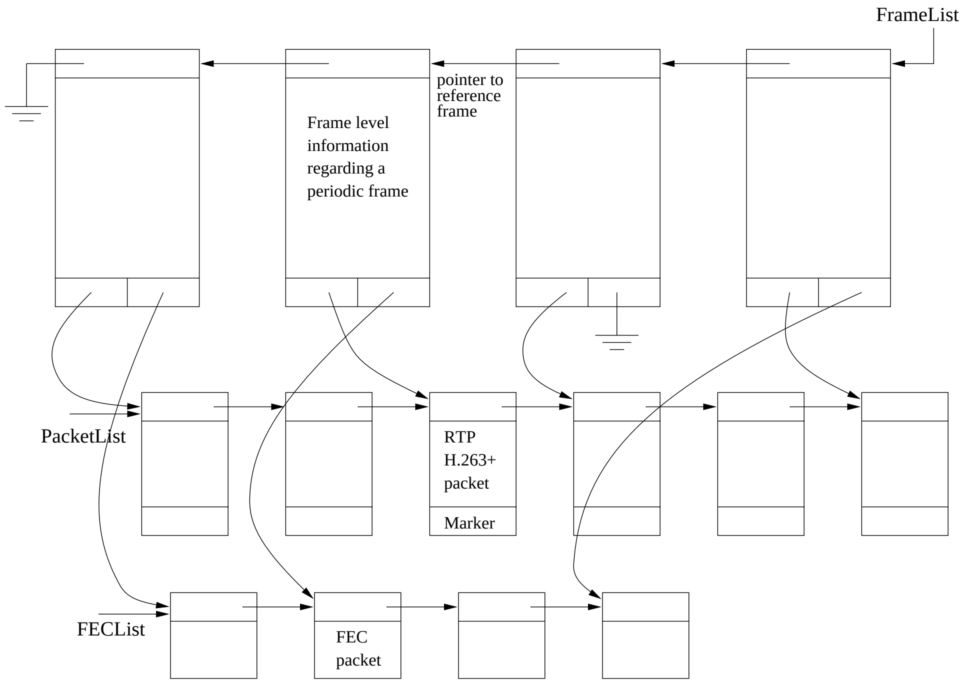
FrameList
pointer to reference frame
Frame level information regarding a periodic frame
PacketList
RTP H.263+ packet
Marker
FECList
FEC packet

Figure 5.2: FrameStore (FrameList, PacketList and FECList)

As mentioned earlier in this section, the repair module is responsible for generating FEC packets for periodic frames, retransmitting their lost packets, and adjusting PTDD.

In order to perform these tasks, it needs a data structure that allows it to store and access packets (video and FEC) belonging to periodic frames, and other "frame-level" information. A data structure, called "FrameStore", has been designed with these goals in mind. The three components of FrameStore, as seen in Figure 5.2, are as follows.

- *FrameList* : A singly-linked list, each node in which contains the following frame level information about one periodic frame.
  - Frame number
  - Frame encoding type (I or P)
  - RTP timestamp of the frame,
  - Number of data packets in the frame
  - Number of FEC packets for the frame, $NFEC$
  - Number of FEC packets already transmitted for the frame.
  - FEC timer
  - Number of lost packets
  - A pointer to the node containing information about the reference frame of the periodic frame.
  - A pointer to the first video packet of that periodic frame
  - A pointer to the first FEC packet of that periodic frame

  The head node of FrameList identifies the periodic frame transmitted most recently. It can be observed that starting at any node in this list, one can follow the links for reference frames and reach "all the ancestors" of the periodic frame represented by the node one started from. Also, at any point in time, FrameList contains only the nodes representing the ancestors of the periodic frame represented by the head node. Note that when PTDD needs to be adjusted due to retransmission, that is, some previously transmitted frame needs to be used as a reference frame, deletion of all the nodes (and the packets of the frames represented by those nodes) leading to that reference frame is all that is required. It can be further observed that the terminal node in FrameList represents the I-frame transmitted most recently. Note also, that at any point in time only the frame represented by the head node can have FEC packets being generated for it. Therefore, only that node can have an active FEC timer.

- *PacketList* : A singly-linked list, each node in which contains the following packet level information about a video packet belonging to a periodic frame.
  - RTP H.263+ video packet.
  - A marker to indicate that the packet was reported lost but was not retransmitted.
- *FECList* : A singly-linked list containing FEC packets encoded and transmitted for the periodic frames represented by nodes in FrameList.

It should be noted however that FrameList does not contain the actual reference frames. At the end of the decoding stage of a I or P-frame encoding, the encoder stores the decoded video frame in a buffer maintained by it. Since B-frames may need to use non-periodic P-frames as reference frames, the encoder must store periodic as well as non-periodic (I and P) pictures. Thus sometimes, due to insufficient buffer space in the encoder, it may not be possible to find a particular (distant) reference frame. In such a case, the encoder simply encodes the current frame as an I-frame.

5.2.1 Encoder

As outlined in Section 5.1, the main tasks of the encoder are to compress a video frame given to it by the grabber, select a proper reference frame (by consulting the repair module), determine whether a frame is a periodic frame, packetize the encoded frame and hand over the RTP packets containing the H.263+ video payload to the session object for transmission to the receiver. The different steps followed by the encoder in performing these tasks are described below.

When the grabber delivers a captured video frame to the encoder, the encoder first determines the coding type (I, P or B) to be used for that frame. The default coding type is P (except for the very first frame, which is intra-coded). The encoder consults the repair module to check whether it has received any replenishment (I-frame) request. If so, the coding type is set to I, otherwise, for a certain predetermined number of frames(configured through OTcl script) after every I or P frame, the coding type is set to B.

As the next step, if the coding type is P or B, the encoder determines a reference frame to be used. If the coding type is B, it selects the previously transmitted I or P picture as a reference frame. However, a B-frame also needs to use the next (future) P-frame. Therefore, the encoder stores the B-frame in a buffer so that it can be coded later when the future P-frame is coded and transmitted. If the coding type is P, the encoder selects the frame represented by the head node in FrameList (maintained in the repair module) as a reference frame. If the selected reference frame cannot be found in the frame buffers managed by the encoder, it simply changes the coding type from P to I.

The encoder then determines whether the frame (I or P) being encoded is a periodic frame. An I-frame is always designated as a periodic frame. A P-frame can be designated as a periodic frame if the FEC timer of the reference frame, the frame represented by the head node in the FrameList, is not active. Note that when we described the lazy hybrid algorithm in Section 3.5, PTDD was precomputed to accommodate FEC packets, and later readjusted when retransmission happens. From an implementation perspective it was easier to just check whether all the FEC packets are transmitted and not precompute the PTDD to accommodate FEC beforehand, hence this approach was adopted.

At this point, the encoder is ready to start encoding the video frame. It allocates an RTP packet for a maximum allowable packet size (set to 1024 bytes for the prototype). The encoder encodes the video frame, one macroblock at a time, and adds the coded data to the packet. When "slice size" or more bytes of encoded data has been added to the packet, it allocates a new RTP packet and starts adding encoded data to that packet. This continues till all the macroblocks in a frame are encoded.

As noted in Section 5.1.2, a video application, that needs to transmit H.263+ video using RTP, should use the corresponding RTP payload format for generating RTP packets containing H.263+ video payload. As per this payload specification, a section of an H.263+ compressed bitstream is carried as a payload within each RTP packet. For each RTP packet, the RTP header is followed by an H.263+ payload header, which is followed by a number of bytes of a standard H.263+ compressed bitstream. The size of the H.263+ payload header is variable depending on the payload involved and the layout of the RTP H.263+ video packet is given below.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    RTP Header                                               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    H.263+ Payload Header                                    ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    H.263+ Compressed Data Stream                            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The H.263+ payload header is structured as follows:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   RR    |P|V|   PLEN    |PEBIT|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RR, PLEN and PEBIT are not used in the RTP H.263+ packets generated by this prototype and therefore we do not describe them here. The 'P' bit, indicating start of a slice, is set in all the packets since each packet contains one slice. The 'V' bit indicates the presence of an 8 bit field containing information for Video Redundancy Coding (VRC), which follows immediately after the initial 16 bits of the payload header, if present. It is intended for use with the reference picture selection mode. The format of the VRC header extension is as follows:

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
| TID | Trun  |S|
+-+-+-+-+-+-+-+-+
```

TID and Trun are not used by the prototye. The 'S' bit indicates that the packet content is for a sync frame, and in our prototype this bit is used to indicate whether the frame to which that packet belongs is a periodic frame. Thus, for all the packets of periodic frames, both the 'V' and 'S' bits described above are set. This arrangement allows the receiver to determine if a particular video frame is periodic or not.

At this stage, all the RTP H.263+ video packets comprising the encoded video frame are ready and the encoder hands them over to the session object for transmission. If the encoded frame is a periodic frame, then it also delivers a copy of these packets to the repair module. The repair module appends the packets to the PacketList, creates a new node representing the periodic frame and inserts it at the head of FrameList. This new head node has the proper links to the reference frame node, first packet of frame etc. The repair module also determines the number of FEC packets, $NFEC$, needed to protect that frame and appropriate interval, Δ, between consecutive FEC packets. If $NFEC > 0$, the repair module also sets the FEC timer (in that node) to timeout after Δ units.

5.2.2 Decoder

Decoding video frames

As outlined in Section 5.1, received RTP H.263+ packets are delivered by the session object at the receiver to the decoder. The decoder decodes the packets and writes the decoded information into a buffer. When a packet of the next frame is received, this buffer is handed over for display.

Additionally, the decoder also plays an important role in recovering lost packets and reconstructing the video frames to which recovered packets belong. It can be observed that in order to support reconstruction of a video frame when its lost packets are recovered, decoder should be able to decode packets out-of-order. The decoder uses the notion of a *frame context* to implement out-of-order decoding. Since only those frames that are likely to be used as reference frames may need to be recovered, any I or P frame seen by the decoder is associated with a *context*, which contains following information.

- Frame number
- Reference frame number
- RTP timestamp of the frame
- Number of video packets received for the frame
- Number of FEC packet received for the frame
- A list of the video packets received for the frame
- A list of the FEC packets received for the frame
- Buffer to which the decoded information should be written

Frame number and Reference frame number are obtained from the "TR" and "TRP" fields in the picture header of a H.263+ video frame.

It is possible that a frame is decoded and displayed with distortion (due to packet losses) and the decoder is decoding a later frame when the repair packets (FEC or retransmitted) for the displayed frame arrive. When such "late" repair packets are received, the decoder performs a "context switch", that is it saves the context of the frame being decoded and restores the context of the old frame. If the repair packet is a video packet, then the decoder can immediately decode that packet and write the decoded information to the corresponding buffer. However, if the repair packet is a FEC packet, the decoder can either recover "all" the lost packets of that frame or will have to receive some more FEC/video packets to attempt recovery. Thus, the decoder checks if it has received enough packets (video and FEC together). If so, it feeds all the packets (video and FEC) of that frame to the FEC decoder (implemented as a part of the decoder), recovers lost video packets and decodes them one at a time.

Generating retransmission requests

While receiving RTP packets from the sender, the decoder detects if there have been any packet losses by looking at the gaps in the RTP sequence numbers of received packets. In general, the decoder cannot ascertain whether the lost packets belong to a periodic frame from gaps in packet sequence numbers. In fact, if the decoder knows the RTP sequence number of the first packet of a frame and the number of packets in that frame, it can identify the lost packets in that frame if it receives even a single packet of that frame. However, the current RTP payload format for H.263+ video [22] does not have the provision to send such information. Also, in the presence of burst losses when entire frames can be lost, this scheme may not work. Therefore, instead of trying to figure out whether to ask for a retransmission or not (as non-periodic frames need not be recovered), the decoder sends a retransmission request for every lost packet. It is then up to the repair module at the sender to determine whether the lost packets belong to periodic or non-periodic frames and decide which ones to retransmit.

Currently, RTCP can be used only to provide statistical feedback about reception quality to the sender. Recently, there have been some efforts towards providing a standard for sending explicit feedback in unicast streaming media [40]. The authors propose extending the RTCP specification to accommodate retransmission requests. RTCP is deemed a reasonable place for putting control packets into because it is already set up and no extra connection needs to be established. We have adopted this approach in designing the feedback scheme for our prototype.

The packet format used for sending retransmission requests is shown below.

```
 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=2|0| UNUSED  | PT=RTCP_NACK  |          length               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             FSN               |            BLP                |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

- The first two bits (V) indicate the current RTP version.
- PT defines the RTCP packet type which identifies the packet as being an RTCP NACK.
- The FSN corresponds to an RTP sequence number of a lost packet.
- The BLP allows for the reporting of losses of the 16 RTP packets immediately following the RTP packet indicated by the FSN. Denoting the BLP's least significant bit as bit 1, and its most significant bit as bit 16, then bit i of the bitmask is set to 1 if the sender has not received RTP packet number FSN+i (modulo 2^16) and the receiver is requesting its retransmission, and 0 otherwise.

Retransmitted packets or their requests can be lost. The decoder detects such losses through timeouts. It sets a retransmission request timer for each RTCP NACK packet. If a lost packet is not received after several retransmission requests, the decoder stops sending further requests for retransmitting that packet. Timeout values for retransmission requests should be based on estimates of round-trip delays (RTT) between receiver and sender. However, currently RTCP supports only the sender to maintain such an estimate. In [40], the authors propose the use of a "Forward Feedback" packet (described in Section 5.2.3) to be transmitted by the sender to convey the RTT estimate. The receiver can use this estimate to choose appropriate timeout values before sending duplicate requests.

Generating replenishment requests

If the lost packets of periodic frames remain unrecovered, it may cause error propagation. For each periodic frame, the decoder keeps track of the number of macroblocks that have been decoded and marks the frame as DIRTY or NOT DIRTY, depending on whether all the macroblocks have been decoded. Since the decoder may have only a limited number of frame buffers, older reference frames need to be replaced by newer reference frames. When a dirty periodic frame is being replaced, the decoder may need to request the sender for a replenishment (I-frame), so that irrecoverable losses do not cause error propagation. For this purpose, the decoder can send an RTCP Full-Intra Request to the sender using the following packet format. This packet format is a modified version of the one described in RTP Payload format for H.261 video [53].

```
 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=2|P| UNUSED  |  PT=RTCP_FIR  |           length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      32-bit RTP timestamp of the frame being flushed          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| FIR TR        |                   UNUSED                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- FIR TR carries the 8-bit unsigned number denoting the frame being flushed out. It is same as the “Frame number” available in the *context* of that frame.

5.2.3 Session

As outlined in Section 5.1, the session object is responsible for transmitting RTP packets to and receiving them from the network. It also periodically sends the RTCP sender and reception reports.

Generating reception reports

The session object at the receiver periodically sends RTCP receiver reports, giving a statistical feedback on the reception quality. In our prototype, we have relaxed the RTCP report intervals to 500ms as there is only one sender and one receiver. This relaxation allows the encoder to quickly respond to changing network conditions. While receiving the packets, the session object at the receiver also maintains an estimate of the average loss burst length. This estimate is included in the RTCP receiver report by adding a 32-bit application-specific extension. The burst length is reported in the MSB 16 bits, the LSB 16 bits not being used. Among the 16 bits of the burst length, 8 MSB bits carry the integer part of the burst length and the 8 LSB bits carry the fractional part of the burst length (expressed as a fixed point number with the binary point at the left edge of the field).

Generating forward feedback

Although the majority control/feedback traffic is generated at the receiver, the session object at the sender periodically sends RTCP sender reports (SR). Also, as the session object at the sender receives RTCP reception reports from the receiver, it updates its estimate of the round-trip delay between the two endpoints. As no mechanism is currently defined which allows the receiver to measure the RTT, the session object at the sender periodically sends its RTT estimate to the receiver in a RTCP packet so that it can employ appropriate retransmission timeouts. The following packet format used for this RTCP packet has been adopted from [40].

```
 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=2|P| UNUSED  | PT=RTCP_FF    |     maximum sequence number   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   RTT estimate                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- PT denotes the packet type for RTCP FF (for "forward feedback")
- maximum sequence number is the highest RTP sequence the sender has transmitted at the time it sends its RTT estimate; this allows the receiver to ensure that any estimate received is more current that its previously recorded estimate.
- RTT estimate is a 32-bit integer representing the sender's RTT estimate in milliseconds.

5.2.4 Repair module

We have mentioned earlier in Section 5.1 that the main tasks of the repair module are: generating FEC packets for periodic frames, retransmitting lost video packets, and adjusting PTDD in response to retransmission. First we describe how it generates the FEC packets followed by how it handles retransmission requests etc.

Generating FEC packets

For every periodic frame, the repair module determines the number of FEC packets, $NFEC$, needed to protect that frame and appropriate interval, Δ, between consecutive

FEC packets. $NFEC$ and Δ are estimated based on a loss model (similar to that described in Section 3.2) maintained by the repair module. If $NFEC > 0$, the repair module sets the FEC timer with a timeout value of Δ. When the timer expires, it encodes a FEC packet and hands it over to the session object for transmission. This may continue as long as the FEC timer is active. The FEC timer is deactivated when either $NFEC$ packets are encoded and transmitted for a periodic frame, or there has been a PTDD adjustment.

Our prototype uses a Vandermonde matrix based Linear Block Coding software FEC encoder and decoder [45], modified for on the fly encoding [16], for generating FEC packets. On the fly encoding allows one to generate FEC packets for a set of (data) packets one at a time, as and when needed. Each FEC packet protects all the data packets in a frame and is uniquely identified by an index number. That is, if any one (data) packet is lost, then it can be recovered provided the FEC packet and all the remaining data packets are received. Two FEC packets that have been generated using the same set of data packets as input will differ in content, if they have different indices. Thus we can repeatedly generate new FEC packets for the same frame by simply specifying new indices. Although an RTP payload format for generic FEC coding is available [46], it allows for one FEC packet to protect at the most 24 data packets. Since the number of packets in a video frame can be more than 24 (e.g., in an I-frame), we have chosen to develop a simple FEC packet format for our prototype. This packet format is as follows.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   RTP Header                                                ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   FEC Payload Header                                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   FEC Payload                                               ...
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The RTP timestamp carried in the RTP header of a FEC packet is same as that carried in the RTP H.263+ packets of the video frame to which the FEC packet belongs.

The FEC payload header contains information that will allow the decoder to use a FEC packet for recovering lost video packets. For the FEC decoder to make use of a FEC packet, it needs to know the number of video packets that have been used to generate the FEC packet, the index of the FEC packet and the indices of the data packets that have been successfully received (or recovered). Therefore, the following format is used for the FEC payload header.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Blocksize   |   FEC Index   |              BSN              |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

- Blocksize denotes the number of video packets for a periodic frame.
- FEC Index denotes the index of the FEC packet, starting from 0 and incremented by one for every additional FEC packet transmitted for the same periodic frame.
- The BSN (Base Sequence Number) field is the 16-bit RTP sequence number of the first packet that has been used to generate this FEC packet. Since packets are fed in sequence to the FEC encoder, the index of a video packet being used for decoding can be determined simply by subtracting the BSN from the sequence number of that data packet.

The session object at the sender transmits the FEC packets on a different layer (that is, to a different UDP port) from the one that is used for video packets. Also, the FEC packets have a separate RTP sequence number space, different from that of video packets. Since FEC packets are transmitted on a different layer, a receiver that is not FEC capable (that is, it does not understand this RTP FEC packet format) can interoperate with a FEC capable sender because it will not be subscribing to the FEC layer and will still be receiving the original video stream. Another important implication of using different layers for video and FEC packets is that, their RTCP packets are also received on separate UDP ports, which allows one to easily find out how the RTCP packet should be parsed.

Processing retransmission requests

When the session object at the sender forwards a received RTCP NACK to the repair module, the steps followed by it in processing the retransmission request are described in detail below. Since a single RTCP NACK can report multiple lost sequence numbers, these steps are performed for each of the lost sequence numbers reported.

Depending on the layer on which an RTCP NACK packet is received, the repair module tries to retrieve the RTP packet corresponding to the lost sequence number from either the PacketList or the FECList. If it cannot find the packet in the list searched, no more processing is required for that sequence number. Recall that both PacketList and FECList contain packets of periodic frames only. Thus, not finding a packet in those lists can be considered as an indication that the frame to which that packet belongs is not a periodic frame.

At this stage, the repair module knows whether the lost packet belongs to a periodic frame. If so, then FrameList must contain a node representing that frame. Since there is a one-to-one mapping between the RTP timestamp of a video frame and its corresponding node in the FrameList, the repair module uses the RTP timestamp to locate the corresponding node in FrameList. Before any further processing, the repair module first increments the number of lost packets of that periodic frame by one.

If the number of lost packets of the video frame has not exceeded the number of FEC packets already transmitted for that frame, then the repair module need not retransmit any packet at this time. This is based on the assumption that the receiver will have enough packets (video and FEC) to fully recover any lost video packets. However, this assumption may not hold if more packets are lost, in which case, lost video packets will have to be retransmitted. Therefore, if the lost sequence number is that of a video packet, the repair module sets the corresponding marker in the PacketList. However, no retransmission happens.

If the number of lost packets exceeds the number of FEC packets already transmitted for that frame, then the repair module must retransmit a lost packet. However, the selection of which packet to retransmit is influenced by whether the lost packet under consideration is a FEC packet or a video packet. If a video packet is lost, then the repair module selects the same packet for retransmission. However, since there is no point in retransmitting a lost FEC packet (in a unicast transmission), if a FEC packet is lost, the repair module selects a lost video packet of that frame for retransmission. If the repair module can find a video packet for that frame with its marker set, it selects that packet for retransmission and resets the marker. This arrangement allows a different lost video packet to be retransmitted if the repair module were to later receive NACK for another lost FEC packet.

The repair module delivers the packet selected for retransmission to the session object for transmission. At this stage, PTDD may need to be adjusted. That is, the next P frame should reference the frame whose packet was recently retransmitted. As noted earlier in this chapter, the repair module just needs to delete all the nodes (and the packets of the frames represented by those nodes) leading to that frame (effectively making its node the head node of FrameList). Also, the FEC timer for this node is deactivated.

Processing full-intra request

A full-intra request is an indication to the sender that the receiver is experiencing error propagation due to irrecoverable losses. Therefore, whenever such a request is received, the repair module makes a note of it and while encoding the next frame it instructs the encoder to encode that frame as an I-frame.

Processing RTCP reception reports

When the session object at the sender receives a valid (extended) RTCP receiver report containing burst length information in addition to other fields, it first updates its estimate of round-trip delay following the algorithm described in [48]. The other information pertaining to quality of reception is passed to the repair module, which then builds the loss model similar to that described in Section 3.2. Note that the current frame rate (which determines the frame interval, δ_f), given to the encoder by the grabber, is also used in building the loss model. The loss model, maintained in the repair module, is used in determining number of FEC packets for a periodic frame, interval between consecutive FEC packets etc.

Chapter 6

Concluding Remarks

Video transmission over lossy networks like the Internet is challenging because the quality of compressed video is very susceptible to packet losses. In this thesis we have focussed on designing dynamic hybrid loss recovery schemes to reduce error propagation due to packet loss in video transmission over the Internet. By carefully combining transport level recovery mechanisms such as FEC and retransmission into a hybrid recovery scheme, we can take advantage of their respective strengths. Also, a dynamic recovery scheme that adapts itself to the changing network conditions can not only improve the resilience to packet loss but can also reduce incurred bit overhead.

We proposed two dynamic hybrid recovery schemes:(1) exhaustive hybrid RESCU, in which the sender finds minimum PTDD that incurs smallest bit overhead, and also satisfies a desired recovery probability of a periodic frame. In order to be able to meet the desired threshold, the PTDD should be long enough to accommodate anticipated repair attempts and the associated delays due to both FEC and retransmission. (2) lazy hybrid RESCU, in which the sender chooses a PTDD that "proactively" provides only for recovery through FEC, and masks out retransmission delays in a truly "reactive" manner, if and when retransmission effects, without introducing additional playout delays.

We showed through simulation experiments designed based on actual Internet transmission tests that these dynamic hybrid recovery schemes provide better performance than existing error-resilient schemes, both in terms of video quality and bit rate. Even in a high motion video sequence when the performance gains of both the hybrid techniques is much attenuated, they nonetheless show better overall performance when compared to NEWPRED and Intra-H.261.

We also desribed implementation of a prototype interactive video transmission system comprised of one sender and one receiver, in which the sender transmits live, H.263+ compressed video to the receiver using RTP/UDP/IP protocols. Lazy hybrid RESCU algorithm is employed for loss recovery. The prototype system has been developed by extending the MASH toolkit. While developing this prototype we have taken care to follow existing standards as much as possible. We have also adopted the approach of other researchers in using RTCP for implementing a feedback scheme in unicast transmission.

In this thesis we have not investigated the integration of rate control schemes. We believe it is possible to incorporate rate control schemes based on frame rate reduction, quantization step-size adjustment etc. in the current framework, so that by reducing the bit rate in times of packet losses, congestion will not be aggravated due to increased overhead of protecting video frames.

Bibliography

[1] A. Albanese, J. Blomer, J. Edmonds, M. Luby, and M. Sudan. Priority encoding transmission. *IEEE Transactions on Information Theory*, 42(6), November 1996.

[2] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory*, 42(6):1732–1736, Nov 1996.

[3] E. W. Biersack. Performance evaluation of FEC in ATM networks. In *Proceedings of the ACM SIGCOMM*, pages 248–257, Baltimore, MD, August 1992.

[4] J. Bolot and A. Vega-Garcia. The case for FEC-based error control for packet audio in the internet. *ACM Multimedia Systems Journal (to appear).*

[5] J-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-based error control for internet telephony. In *Proceedings of the IEEE INFOCOM*, volume 3, pages 1453–1460, March 1999.

[6] J-C. Bolot and T. Turletti. Adaptive error control for packet video in the internet. In *Proceedings of International Conference on Internet Protocols*, volume 1, pages 25–28, Lausanne, September 1996.

[7] C. Bormann, L. Cline, G. Deisher, T. Gardow, C. Marciocco, D. Newwell, J. Ott, G. Sillivan, S. Wenger, and C. Zhu. RTP payload format for the 1998 version of ITU-R Rec. H.263 video (H.263+), IETF draft-ietf-avt-rtp-h263-video-p2.txt, May 1998.

[8] G. Carle and E. Biersack. Survey of error recovery techniques for ip-based audio-visual multicast applications. *IEEE Network*, pages 24–36, December 1997.

[9] S. Casner and V. Jacobson. Compressing IP/UDP/RTP headers for low-speed serial links (work in progress) IETF, RFC-2508, Feb 1999.

[10] D. Clark and L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM SIGCOMM*, pages 200–208, September 1990.

[11] M. Clark and K. Jeffay. Application-level measurements of performance on the vbns. In *Proceedings of IEEE Intl. Conf. on Multimedia Computiing and Systems, vol. 2*, pages 362–366, Florence, Italy, June 1999.

[12] M. Degermark, H. Hannu, L. Jonsson, and K. Svanbro. CRTP over cellular radio links (work in progress) IETF, RFC-2509, June 1999.

[13] B. Dempsey, J. Liebeherr, and A. Weaver. On retransmission-based error control for continuous media traffic in packet-switching networks. *Computer Networks and ISDN Systems*, pages 719–736, 1996.

[14] H. Deng and M. Lin. A type I hybrid ARQ system with adaptive code rates. *IEEE Transactions on Communications*, COM-46(2):733–737, Feb. 1995.

[15] T. Dorcey. Cu-seeme desktop videoconferencing software. *ConneXions*, 9(4), March 1995.

[16] D. Rubenstein et al. Improving reliable multicast using active parity encoding services (APES). Technical Report 98-79, UMASS, July 1998.

[17] S. McCanne et al. Toward a common infrastructure for multimedia-networking middleware. In *Proceedings of NOSSDAV*, pages 39–49, St. Louis, MO, May 1997.

[18] R. Fredrick. Network video(nv). Technical report, Xerox Palo Alto Research Center.

[19] M. Ghanbari and V. Seferidis. Cell-loss concealment in ATM video codecs. *IEEE Transactions on Circuits and Sys. for Video Tech.*, 3(3):238–47, June 1993.

[20] J. Hagenauer. Rate-compatible punctured convolutional codes (RCPC codes) and their applications. *IEEE Transactions on Communications*, 36(4):389–400, April 1988.

[21] ITU-T. Recommendation, H.263: Video codec for low bit-rate communications, 1996.

[22] ITU-T. Recommendation, h.263+: Video codec for low bit-rate communications, 1998.

[23] S. Kallel and D. Haccoun. Generalized type-II hybrid ARQ scheme using punctured convolutional coding. *IEEE Transactions on Communications*, 38(11):1938–1946, November 1990.

[24] S.K. Kaseta, J. Kurose, and D. Towsley. Scalable, reliable multicast using multiple multicast groups. In *Proceedins of ACM SIGMETRICS*, pages 64–74, Seattle WA, 1997.

[25] L. Kieu and K. Ngan. Cell-loss concealment techniques for layered video codecs in an ATM network. *IEEE Transactions on Image Processing*, 3(5):666–77, September 1994.

[26] T. Kinoshita, Nakahashi, and M. Takizawa. Variable bit-rate HDTV coding algorithm for ATM environments in B-ISDN. In *Proceedings of SPIE Conference on Visual Communications and Image Processing: Visual Communication*, pages 604–612, November 1991.

[27] LBC. Document, LBC-95-309 (ITU-T SG 15, WP 15/1), Sub-videos with retransmission and intra-refreshing in mobile/wireless environments, 1995.

[28] LBC. Document, LBC-96-033 (ITU-T SG 15, WP 15/1), An error-resilience method based on back channel signalling and FEC, 1996.

[29] C. Leicher. Hierarchical encoding of MPEG sequences using priority encoding transmission (pet). Technical Report 94-058, ICSI, November 1994.

[30] X. Li, S. Paul, P. Pancha, and M. Ammar. Layered video multicast with retransmission (LVMR):evaluation of error recovery schemes. In *Proceedings of the Sixth International Workshop on Network and Operating System Support for Digital Audio and Video, St Louis*, pages 161–172, May 1997.

[31] S. Lin and D. Costello. *Error control coding: fundamentals and applications.* Prentice-Hall Inc., Englewood Cliffs, NJ, 1983.

[32] H. Liu and M. El Zarki. Performance of video transport over wireless networks using hybrid ARQ. In *Proceedings of Proceedings of IEEE International Conference on Universal Personal Communications (ICUPC)*, pages 567–571, Boston, MA, October 1996.

[33] W. Luo and M. El Zarki. Analysis of error concealment schemes for MPEG-2 video transmission over ATM networks. In *Proceedings of the SPIE/IEEE Visual Communications and Image Processing*, Taiwan, May, 1995.

[34] S. McCanne and V. Jacobson. vic: a flexible framework for packet video. In *Proceedings of ACM Multimedia'95, San Francisco, CA*, pages 511–522, November 1995.

[35] Jorg. Nonnenmacher, Ernst Biersack, and Don Towsley. Parity-based loss recovery for reliable multicast transmission. *ACM Transactions on Networking*, 6(4):349–361, 1998.

[36] 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.

[37] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[38] M. Podolsky. A study of speech/audio coding on packet switched networks. Technical report, MS Thesis, Dept. of Electrical Eng. and Computer Sciences, Univ of California, Berkeley, December 1996.

[39] M. Podolsky, C. Romer, and S. McCanne. Simulation of FEC-based error control for packet audio on the internet. In *Proceedings of the IEEE Infocom*, volume 2, pages 505–515, San Francisco, CA, March 1998.

[40] M. Podolsky, K. Yano, and S. McCanne. A RTCP-based retransmission protocol for unicast streaming internet draft draft-podolsky-avt-rtprx-00.txt(work in progress), IETF, October 1999.

[41] G. Ramamurthy and D. Raychaudhuri. Performance of packet video with combined error recovery and concealment. In *Proceedings of the IEEE INFOCOM*, pages 753–761, April 1995.

[42] I. Rhee. Error control techniques for interactive low-bit rate video transmission over the internet. In *Proceeding of the ACM SIGCOMM'98*, Vancouver, Canada, Sept. 1998.

[43] I. Rhee. Retransmission-based error control for interactive video applications over the internet. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 118–127, Texas, Austin, June 1998.

[44] Injong Rhee and Srinath R. Joshi. FEC-based loss recovery for interactive video transmisssion. In *Proceedings of IEEE Intl. Conf. on Multimedia Computiing and Systems, vol. 1*, pages 250 – 256, Florence, Italy, June 1999.

[45] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *Computer Communication Review*, 27(2):24–36, Apr 1997a.

[46] J. Rosenberg and H. Schulzrinne. An a/v profile extension for generic forward error correction in RTP, internet draft draft-ietf-avt-fec (work in progress), IETF, July 1997.

[47] Dan Rubenstein, Jim Kurose, and Don Towsley. Real-time reliable multicast using proactive forward error correction. In *Proceedings of NOSSDAV*, Cambridge, UK, July 1998.

[48] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A transport protocol for real-time applications, January 1996.

[49] E. Steinbach, N. Farber, and B. Girod. Standard compatible extension of H.263 for robust video transmission in mobile environments. *IEEE Transactions on Circuits and Sys. for Video Tech.*, 7(6):872–81, December 1997.

[50] R. Storn. Modeling and optimization of pet-redundancy assignment for MPEG sequences. Technical Report TR-95-018, ICSI, Berkeley, CA, May 1995.

[51] R. Talluri. Error-resilient video coding in ISO MPEG-4 standard. *IEEE Communication Mag.*, 36(6):112–119, June 1998.

[52] Priority Encoding Transmission. Web page: http://www.icsi.berkeley.edu/pet/icsi-pet.html.

[53] T. Turletti and C. Huitema. RTP payload format for H.261 video streams, IETF RFC 2032, October 1996.

[54] M. Wada. Selective recovery of video packet loss using error concealment. *IEEE Journal on Selected Areas in Communications*, 7(5):807–814, 1989.

[55] R. Xu, C. Myers, H. Zhang, and R. Yavatkar. Resilient multicast support for continuous-media applications. In *Proceedings of the Sixth International Workshop on*

Network and Operating System Support for Digital Audio and Video, pages 183–194, St. Louis, May 1997.

[56] J. Zdepski and H. Sun. Error concealment strategy for picture-header loss in MPEG compressed video. In *Proceedings of High-Speed Networking and Multimedia Computing*, pages 145–152, San Jose, CA, Feb 1994.