PROGRESSIVELY RELIABLE PACKET DELIVERY FOR INTERACTIVE WIRELESS MULTIMEDIA

by

Richard Yeh-Whei Han

Memorandum No. UCB/ERL M97/36

26 May 1997

PROGRESSIVELY RELIABLE PACKET DELIVERY FOR INTERACTIVE WIRELESS MULTIMEDIA

by

Richard Yeh-Whei Han

Memorandum No. UCB/ERL M97/36

26 May 1997

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Abstract

Progressively Reliable Packet Delivery For Interactive Wireless Multimedia

by

Richard Yeh-Whei Han

Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences

University of California at Berkeley

Professor David G. Messerschmitt, Chair

In this dissertation, we propose a progressively reliable end-to-end network protocol for delivery of delay-sensitive visual multimedia over a wireless channel with a time-varying bit error rate and limited bandwidth. Interactive applications like Web-based image browsing which operate over a wireless access link to the Internet require immediate delivery of image data to the receiver in order to support genuine interactivity. Wired Internet connections incur roundtrip delays approaching the interactive latency bound. Reliable protocols that retransmit lost or corrupt packets over an Internet connection that also includes a noisy wireless link will be unable to deliver an image by the interactive latency bound. We derive a lower bound on the latency incurred by an ideal retransmission-based ARQ protocol in a noisy bit error rate (*BER*) wireless channel, and show that full reliability will cause the transport latency to far exceed the interactive latency bound at 1% *BER*. We also show that, even by adding powerful Reed-Solomon forward error correction (FEC) codes with redundancy rates that double or triple the bandwidth, the latency penalty due to ARQ retransmissions is too high to achieve the interactive latency bound at 3% *BER*.

Applications that accept unreliable packet delivery, e.g. packet loss and packet corruption, have a more reasonable chance of achieving the interactive latency bound over a noisy channel. Practical complexity and delay constraints on FEC and compression cause the traditional approach of aggressive compression and aggressive FEC to be unable to deliver images quickly enough at severe *BER*'s. Under these conditions, we show that the joint source/channel coding approach of error-resilient image coding/decoding combined with forwarding of corrupt packet data can continue to support interactive image display. We demonstrate that error-tolerant image coding can reconstruct images at 3% *BER* and simultaneously compress images down to 0.75 bits/pixel via lossy quantization only.

We propose a progressively reliable end-to-end protocol designed for rapid yet asymptotically reliable delivery of delay-sensitive imagery. A possibly noisy initial version of a packet is forwarded to the receiver quickly to allow the end user to interact immediately with an initially noisy image. For bursty multimedia applications like Web-based image browsing, the noisy still-image needs to be cleaned of any persistent artifacts. Therefore, the protocol follows its initial delivery with multiple increasingly reliable deliveries of each packet, leveraging off of the retransmission mechanism of the protocol. We call this progressively reliable protocol "Leaky ARQ". We identify through X server simulation three additional performance-enhancing functions for Leaky ARQ: delaying retransmissions by many seconds; cancelling out-of-date retransmissions; and fine-grained scheduling of application data through the use of flows. Finally, we show how Type-II Hybrid ARQ protocols, also called packet combining or memory ARQ protocols, can be modified to implement Leaky ARQ.

David G Messerschmitt May 20, 1997

Professor David G. Messerschmitt, Chair Date

Acknowledgments

I take great pleasure in thanking the many people who contributed to my education here at Berkeley. First, I would like to thank my advisor, Professor David Messerschmitt, for his guidance and advising philosophy. The academic freedom and flexibility which he gave me to pursue the dissertation topic of my choosing were invaluable to making my graduate experience at Berkeley an extraordinarily positive one. Perhaps the most valuable lessons I have learned are the importance of questioning underlying assumptions, and how this questioning can dramatically reveal new vistas of research.

I'd also like to thank Professor Bob Brodersen for including me in a project with the scope and historical impact of the InfoPad project. My experience with a system-level project like InfoPad was invaluable in giving me new insight into "Big Picture" system design issues, a new understanding of design issues faced in other related technical fields, improved experience working in teams, and an enhanced appreciation of industry's interaction with academic research.

I'd like to thank Professor J. George Shanthikumar for serving on my dissertation and qualifying exam committees. I'd like to thank Professor Jan Rabaey for serving as the chair for my qualifying exam committee. I'd like to thank Professor Randy Katz for helping me find my current position at IBM T.J. Watson Research Center.

I would like to thank my good friends and fellow graduate students, Louis Yun, Allen Lao, and Tom Truman, for their substantial advice and encouragement during my time at Cal. I'd also like to thank my fellow graduate students and InfoPad staff for their help and friendly feedback: Paul Haskell, Wan-Teh Chang, Praveen Murthy, Yuan-Chi

Chang, Craig Teuscher, Sam Sheng, Jeff Gilbert, Brian Evans, Johnathan Reason, Mike Hu, Shankar Naryanaswamy, Brian Richards, Jennie Chen, Renu Mehra, Tony Stratakos, Dave Lidsky, Lisa Guerra, William Li, Ivy Hsu, Shih-Fu Chang, Hari Balakrishnan, Venkat Padmanabhan, Fred Burghardt, and Mike Williamson.

I'd like to thank my parents, whose love and encouragement have sustained me through the most trying times of the Ph.D. program.

I'd also like to thank the taxpayers of America for supporting my education through an NSF Graduate Fellowship and ARPA's funding for the InfoPad project. Hopefully, they got their money's worth.

Berkeley, California *Rick Han*

May 1997

Contents

1

Introduction

"...Yet all experience is an arch wherethrough gleams that
untravell'd world whose margin fades forever and forever as I move."
— *Ulysses*, by Lord Tennyson

In this dissertation, we consider the design of an end-to-end network protocol for delay-sensitive visual multimedia delivered over an Internet connection that includes a wireless access channel with a time-varying bit error rate and relatively limited bandwidth. Interactive applications like Web-based image browsing which operate over a wireless access link to the Internet require immediate delivery of image data to the receiver in order to support genuine interactivity. Such delay-sensitive imaging applications can be designed to tolerate channel distortion in the form of packet loss and packet corruption. By tolerating some channel distortion in a reconstructed packetized image, delay-sensitive applications can reduce the perceptual latency seen by the end user. Error protection schemes like forward error correction (FEC) and/or reliable retransmission-based protocols (also known as ARQ protocols) are designed to mitigate distortion caused by channel noise, but will introduce additional transport latency either through retransmissions or bandwidth expansion. By accepting less than fully reliable packet delivery, delay-sensitive applications don't have to suffer the latency penalty associated with fully reliable packet delivery over a noisy wireless channel. This dissertation develops a *progressively reliable end-to-end protocol* that delivers an initial possibly noisy version of a packet to the receiver quickly, followed by multiple increasingly reliable versions of that packet. Progressively reliable packet delivery allows the end user to trade off subjective priorities related to the two dimensions of quality/distortion and delay by initially supporting low-latency high-distortion delivery of a packetized image, while ultimately supporting low-distortion high-latency reconstruction of a packetized image.

1.1 Wireless access to interactive visual multimedia

The expanding demand for Internet connectivity, the growth of multimedia computing, and the rising expectation of portable access to information are symptoms of the trend towards wireless access to multimedia information "anytime, anywhere". As part of this increasing integration of computing, communications, and portability, personal computer laptops can now access the Internet via wireless modems, gaining access to a vast array of visual multimedia services such as Web-based images, text/graphics, and video. Similarly, portable "network computers" (Berkeley's InfoPad [90], Xerox PARC's MPad [68], Digital's Web-aware PDA [7]) have been prototyped recently and offer an alternative paradigm for wireless access to networked visual multimedia. Both the laptop and portable network computer models attest to the great appeal of combining *portability*, *connectivity*, and *multimedia* into a single computing and communications device. Previously, personal computers (multimedia), workstations (networked multimedia), laptops (portable multimedia), and personal digital assistants (portable with limited multimedia) supported one or two of

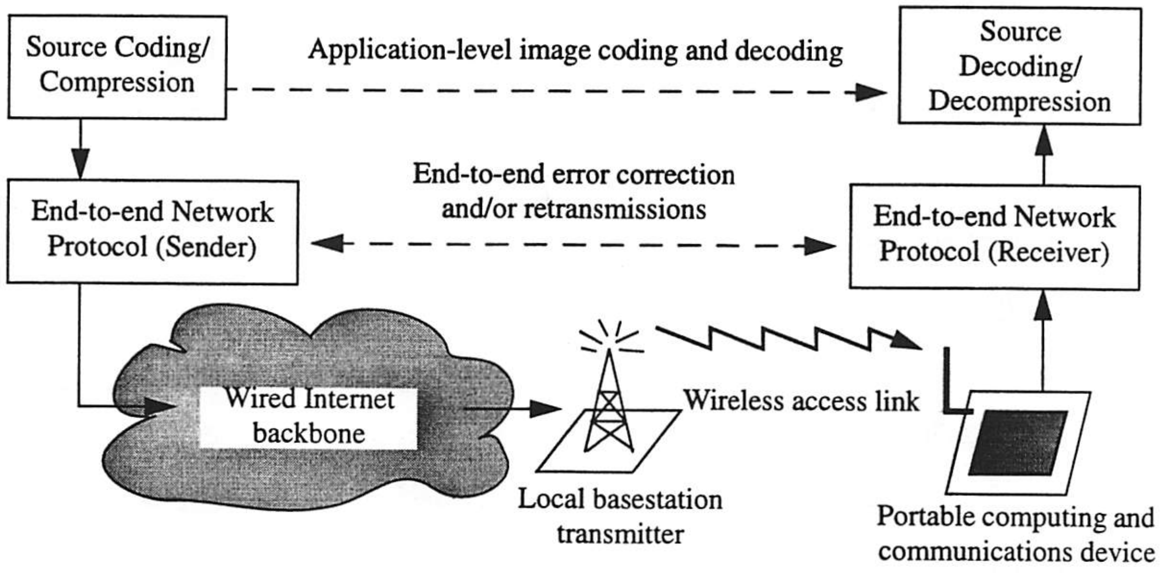

Source Coding/ Compression
Application-level image coding and decoding
Source Decoding/ Decompression
End-to-end error correction and/or retransmissions
End-to-end Network Protocol (Sender)
End-to-end Network Protocol (Receiver)
Wired Internet backbone
Wireless access link
Local basestation transmitter
Portable computing and communications device

Figure 1.1 General system supporting wireless access to visual multimedia. Packetized visual multimedia flows from the source coder through the end-to-end network protocol that implements end-to-end error protection, over an Internet backbone and concatenated wireless access link, to the portable multimedia receiver for decoding and display.

these individual elements, but failed to realize the potential of combining all three characteristics into a single computing and communications paradigm.

In a system which integrates portability, connectivity, and multimedia, the end user expects rapid response time and sufficiently reliable communication. These expectations conflict with the reality of delivering high bandwidth image-based data across a connection that includes a wireless link with a relatively low bandwidth and relatively high noise level. The overall source-network-receiver system that supports wireless access to distributed visual multimedia is pictured in Figure 1.1. The multimedia application or source coder generates images which are compressed, possibly in an error-resilient manner, and sends the coded image data to the underlying network, e.g. transport protocol, for end-to-end packet delivery. The transport protocol is responsible for ensuring that the packetized data arrives at the receiver with sufficient reliability and sufficient speed. The transport protocol has the option of applying end-to-end error protection, either in the form of FEC and/or retransmission-based ARQ protocols. The wireless channel represents the weakest link in the overall connection both in terms of limited bandwidth and relatively severe bit error rates (*BER*'s). The design of an end-to-end network protocol must recognize the impairments posed by this weakest link, both in terms of its limited bandwidth and its heavy channel noise.

The scope of this dissertation is confined to designing an end-to-end protocol for packet delivery of bursty high-bandwidth delay-sensitive multimedia across connections containing a bandlimited wireless link with a time-varying and at times severe *BER*. Our primary application of interest is interactive Web-based image browsing across a wireless access link, though we mention later how other applications based on video and audio can also utilize our proposed protocol. In the remaining paragraphs of this section, we outline the limiting assumptions that are made with regard to the type of source application considered, the end user's quantitative expectations in terms of delay and distortion, and the quantitative assumptions regarding the wireless link's bandwidth and *BER*.

Our end-to-end protocol is primarily designed for the class of interactive multimedia applications which require remote delivery of still images. This protocol is not designed to deliver unrendered text or command-based graphics, though it could be used to deliver images with pre-rendered text/graphics. In this class of distributed imaging applications, we include interactive Web-based image browsing, remotely rendered applications like Framemaker, and portable network computers which download bitmapped frame buffer updates across the network like the InfoPad. In each of these examples, images are periodically rendered at the source and downloaded to the receiver, thereby exhibiting bursty traffic behavior. While our primary objective is supporting rapid delivery of delay-sensitive bursty multimedia, we shall mention in Chapter 5 how the proposed protocol can be parameterized to deliver continuous media audio and video.

In the image browsing application described above, the end user has subjective expectations that are stringent in terms of delivery latency but are more relaxed in terms of the permissible distortion introduced by channel noise. For real-time video/audio conferencing applications, the maximum acceptable roundtrip delay cannot exceed about 200 ms [41][132], though some would argue for a tighter roundtrip bound. For point-and-click bursty-media applications like interactive image browsing, genuine responsiveness will impose a similar though likely tighter requirement of about 100 ms roundtrip delay. This constraint on delay plays a key role in the design of our progressively reliable protocol. On the other hand, the subjective tolerance for channel distortion can actually be quite high for natural images. We demonstrate in Chapter 4 that channel *BER*'s of about 3% are still subjectively tolerable for natural images which have been compressed over 10:1 from 8 bits/pixel down to 0.75 bits/pixel. Together, the stringent delay requirement and the tolerance for channel distortion help motivate the decision to forward corrupt information as a means of lowering the perceptual latency within our protocol design.

Wireless channels range from indoor picocellular wireless access links to outdoor microcellular channels with mobile access from automobiles. The key digital performance parameters of a wireless channel are its bandwidth and its *BER*. In addition, the variation of the *BER* with time can also affect protocol design. Bit errors arise from analog impairments in the wireless channel. A single user transmitting its data over a wireless link can suffer from three roughly independent analog noise phenomena: shadowing, path loss, and multipath fading [125]. Shadowing occurs when the line-of-sight signal is blocked by an object such as a mountain or building or even a person. Path loss occurs because the power of the signal falls off exponentially with distance. Multipath fading occurs when multiple reflections of a transmitted signal arrive at the receiver and add destructively due to phase shifts. In addition to these single user phenomena, in cellular systems there are multiple users whose transmissions generate intercell and intracell interference for other users. Frequency reuse in TDMA cellular systems introduces co-channel interference from other cells. Also, imperfect isolation of users introduces adjacent-channel interference both within and outside of the user's cell. Direct sequence CDMA spread spectrum systems are also well known to be interference-limited, in the sense that increased transmission power dedicated to one user will add interference noise to all other users. Finally, for indoor picocellular systems, an individual user may also have to deal with prolonged shadowing due to slow-moving/static objects (e.g. people, cubicle walls, etc.) interfering with line-of-sight transmission. Indoor channels do not have to deal with Doppler effects characteristic of outdoor mobile cellular systems. All of these phenomena create time-varying bursts of errors.

An end-to-end protocol should be designed to handle this heterogeneity in behavior over a variety of wireless access channels. However, our work makes some limiting assumptions with regard to the channel bit rate, and the channel *BER*. Our discussion is confined to wireless channels whose bit rates range from about 100 kbit/s to 1-2 Mbit/s. This range of bit rates offers a reasonably sized image some chance of achieving the 100-200 ms interactive latency bound. In addition, this range of bandwidth is likely to represent the typical wireless access link of future cellular networks. Currently, second-generation digital cellular standards like IS-54 digital TDMA, IS-95 CDMA, and Europe's GSM system support voice and data services at bit rates up to 13 kbit/s, though the raw channel rates can be more than double these values due to error correction overhead [94]. Current digital cordless standards, which support mobility in a more limited manner than cellular systems, can sustain even higher bit rates. For example, Europe's CT-2 system delivers voice and data over a 32 kbit/s cordless link [29]. Japan's Personal Handiphone System (PHS) supports voice at 32 kbit/s and data services up to 64 kbit/s. The European DECT standard transmits voice at 32 kbit/s, and data up to 384 kbit/s over one connection [121]. Several vendors have begun offering wireless LAN products whose radios operate at hundreds of kbit/s to one Mbit/s per user [29]. In the future, widely deployed global wireless access systems will likely offer a similar range of bit rates. For example, plans are underway in Europe for a third-generation Universal Mobile Telecommunications System (UMTS) that supports cellular, cordless and LAN access through a ubiquitous wireless access network operating at rates of at least 2B+D ISDN (144 kbit/s), and possibly higher (up to 2 Mbit/s).

Our second assumption is that wireless channels average at least 10^{-2} *BER* within a fade. Designers of digital cellular systems have often cited 10^{-2} *BER* as a typical design point for wireless audio [31][149]. In addition, the packet loss rate has been measured at 1-2% under realistic fully loaded conditions in the IS-95 DS-CDMA digital cellular system after rate-$\frac{1}{2}$ convolutional FEC coding [69]. Our design philosophy is to support continuous interactivity with possibly noisy packetized image data at *BER*'s up to and slightly exceeding 10^{-2}.

1.2 Joint source/channel coding for delay-sensitive wireless data

Given the system assumed in Figure 1.1, the traditional approach is to separate the design of compression/decompression algorithms, also called source coding and decoding, from the design of FEC/ARQ error protection, also called channel coding and decoding. For example, the still image coding JPEG standard is designed independently from the underlying network. JPEG does not care whether the underlying error protection is a linear Reed-Solomon block code, a convolutional code, or a retransmission-based ARQ protocol with error detection. Conversely, an ARQ protocol like the Internet's TCP has been designed independently of any source compression standard. This independent source and channel coding design philosophy is illustrated as the top picture in Figure 1.2.

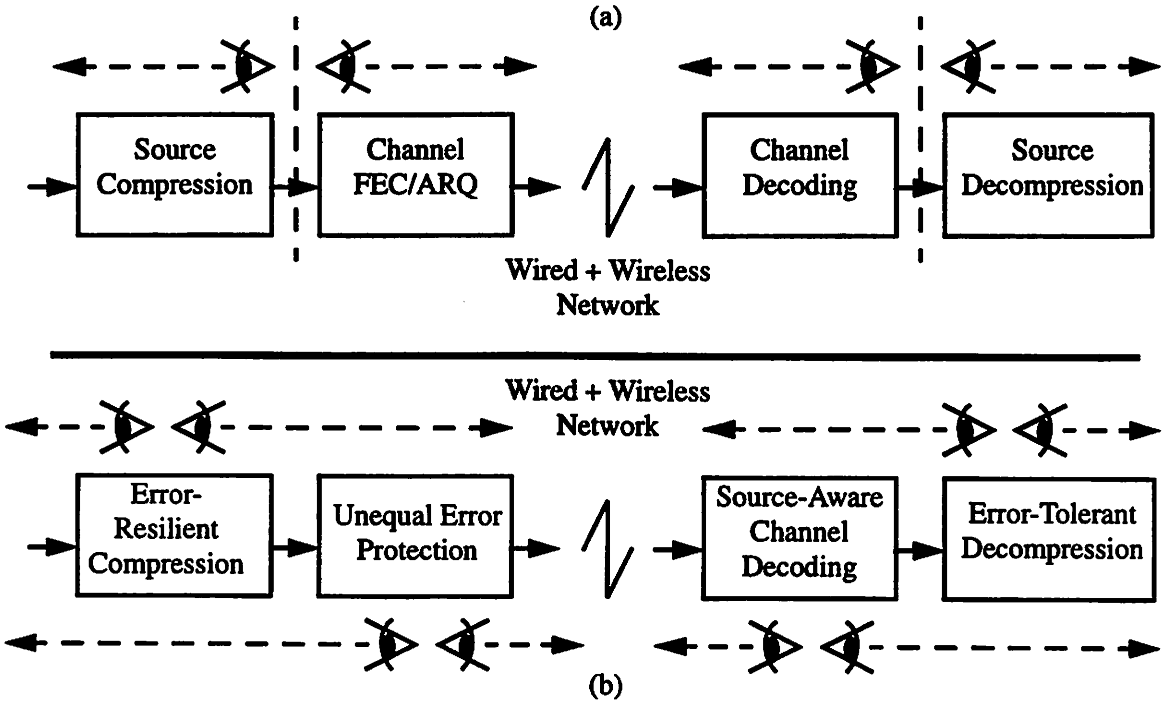


Figure 1.2 Joint source/channel coding (JSCC) (bottom figure (b)) compared to independent/separated source and channel coding (top figure (a)). In JSCC, compression and decompression algorithms "see" the channel's characteristics, e.g. its typical *BER*, and are designed to be robust. Channel FEC/ARQ coders and decoders "see" the source's statistics, and apply unequal error protection to the data. In independent source and channel coding, source coding/decoding only sees the input data, not the channel. FEC/ARQ is designed with only the channel in mind, and is oblivious to the source's statistics.

The channel coding and decoding modules are designed with only the characteristics of the network or channel in mind, i.e. they "see" only the channel impairments, and don't "see" anything about the source's statistics, thereby simplifying the task of the network protocol designer. Conversely, the source coding and decoding modules only "see" the data they are compressing, and are designed without regard to whether the compressed data is being sent over a wireless channel, thereby simplifying the task of the compression algorithm designer.

The traditional design philosophy of independent source and channel coding is justified theoretically by *Shannon's separation theorem* [27]. The theorem states that transmission of a source (e.g. image) through a noisy channel (e.g. wireless link) can be made arbitrarily close to reliable (arbitrarily close to zero probability of transmission error) as long as the source's information rate/entropy is less than the information-theoretic channel capacity. Consequently, the source's only duty is to aggressively compress the data below the channel capacity, without regard to any other properties of the channel. Assuming a suitably compressed source, the channel coder can design its FEC/ARQ error protection independent of any knowledge of the source. The separation theorem implies that no loss in performance, as measured by the reliability of the reconstructed data at the receiver, is suffered by a separated source and channel coding approach.

In Chapter 2, we quantify the latency performance of a traditional ARQ protocol that has been designed independently of source statistics. Two lower bounds on the delay that would be suffered by data requiring fully reliable packet delivery over a wireless channel are derived. At 10^{-2} *BER*, we show that even an ideally efficient retransmission-based protocol, called ideal SRP, cannot deliver its data within the interactive latency bound of 200 ms.

In Chapter 3, we quantify the minimum redundancy required by a traditional FEC linear block code, again designed independently of the image source, in order to adequately protect a sequence of data blocks corresponding to an image. We quantify how much more redundancy is required by binary BCH codes than optimal binary linear codes. For non-binary Reed-Solomon (RS) codes, we show that at 3×10^{-2} *BER*, the minimum redundancy required will double the bandwidth of the source.

Further, we analyze Type-I Hybrid ARQ protocols which combine FEC and ARQ together as a form of hybrid error protection. Assuming that RS codes perform the FEC function, and assuming that ideal SRP performs the ARQ function, then we show that no sufficiently powerful RS codes of length $N<1024$ output symbols could be found which are capable of reliably delivering a full image by the interactive latency bound of 100 ms at 3% *BER* for a Type-I Hybrid ARQ protocol.

The analytical results from Chapters 2 and 3 help motivate our search for alternatives to conventional FEC and ARQ channel coding schemes. In Chapter 4, we examine an alternative method for delivering delay-sensitive data over a wireless channel called *joint source/channel coding* (JSCC). First, we observe that the separation theorem was derived under the following three assumptions:

- stationary memoryless channels
- unconstrained complexity of compression/decompression algorithms and FEC/ARQ error protection techniques
- unconstrained delay in the operation of source and channel coders and decoders

Since our application of interest is interactive image browsing over time-varying wireless access links with practical limitations on compression efficiency and FEC error correction power, then *each one of these assumptions of the separation theorem is violated.* Portable access to distributed visual multimedia, which has the potential of evolving into a fairly common paradigm, calls into question the traditional philosophy of separating the design of compression algorithms from the design of network FEC/protocols.

In addition, the appropriate quality criterion in our case is a subjective dual function of delay and distortion. For example, progressive image transmission is a technique which exploits the human user's subjective tolerance for significant distortion in an initial version of an image provided that the end user knows that the image will improve eventually in quality over time. Progressively reliable packet delivery discussed in Chapter 5 represents another example of progressivity. The human user's changing tolerance for distortion over time is difficult to measure in quantitative terms. The separation theorem makes no comment on such qualitative evaluation metrics, relying instead only on quantitative measures of performance such as the loss probability which fail to capture the subjective complexity motivating our theme of progressivity.

When one or more of the theorem's assumptions are not satisfied, an approach called joint source/channel coding can be shown to outperform separately designed source and channel coders. The overall JSCC system is illustrated in the bottom portion of Figure 1.2. The JSCC channel coder and decoder are designed with knowledge of the source as well as the channel, i.e. they "see" the source's statistics and the variation in error sensitivity of different source bits and incorporate this knowledge into the design of an unequal error protection (UEP) codec. Similarly, the JSCC source coder and decoder are designed with knowledge of the channel as well as the source, i.e. they "see" the channel's error statistics and incorporate that knowledge into the design of an error-tolerant compression/decompression scheme.

The literature has shown that when compression algorithms are constrained in complexity and required to perform their operations quickly, then residual redundancy is left in an imperfectly compressed image. Imperfect compression leaves certain bits more sensitive to errors than others. Errors in sensitive bits will have a disproportionate effect on the distortion in a reconstructed image. Given imperfect compression, Section 4.3.2 and Section 4.3.3 cite references which show that JSCC channel coders and decoders that have knowledge of the source's statistics, e.g. UEP channel coders and source-cognizant channel decoders, produce images with lower end-to-end distortion than independent channel coders and decoders which have no knowledge of the source's statistics.

The literature has also shown that when the channel FEC coder and decoder are limited in error correction power due to constraints on complexity and the requirement that they perform their task quickly, then there is some benefit to backing off on aggressive compression and intentionally leaving some residual redundancy in an image. Section

4.3.4 cites the work by Xu, Hagenauer, and Hollman [155] which shows that error-tolerant image coding is more beneficial than a separated system of aggressive compression and aggressive FEC in terms of lowering the end-to-end image distortion when the channel is very noisy and the error correction power is constrained.

Given error-tolerant image compression, then we claim that packets with corrupt bits should not be thrown away. Current networks discard packets which have failed an error detection check, either at the data-link layer of a wireless link, or at the receiver endpoint of a connection. Discarding corrupt packets is done on the presumption that the payload data has been aggressively compressed and is therefore not useful at the receiver. However, we have just shown that there is some benefit to error-tolerant image compression in constrained-complexity constrained-delay systems. Given error-resilient image coding, then most of the payload is still useful at the receiver despite heavy channel noise. For example, a 1% *BER* applied to an 8bits/pixel grayscale image would invalidate only about 8% of the bits in any packet payload, leaving over 90% of the pixels as still usable.

We implemented a Discrete Cosine Transform (DCT) compression scheme which demonstrates the usefulness of error-tolerant compression and error-tolerant decoding of corrupt image data at high *BER*'s. The DCT compression scheme was able to compress the data from 8 bits/pixel (bpp) grayscale down to 0.75 bpp, a compression ratio of about 10:1. We were able to demonstrate that this error-tolerant encoding scheme could tolerate *BER*'s of 10^{-2} without error concealment, and $3\text{x}10^{-2}$ with error concealment. Our conclusion was that lossy quantization can achieve a reasonable compression ratio and still be error-resilient. Lossless statistical compression increases the error sensitivity and requires corresponding increases in FEC channel coding, thereby increasing the complexity of the system. Moreover, the limited strength of FEC in practical systems causes the aggressive compression/aggressive FEC approach to fail at heavy *BER*'s, while the error-resilient image coding is still able to communicate information by the interactive latency bound.

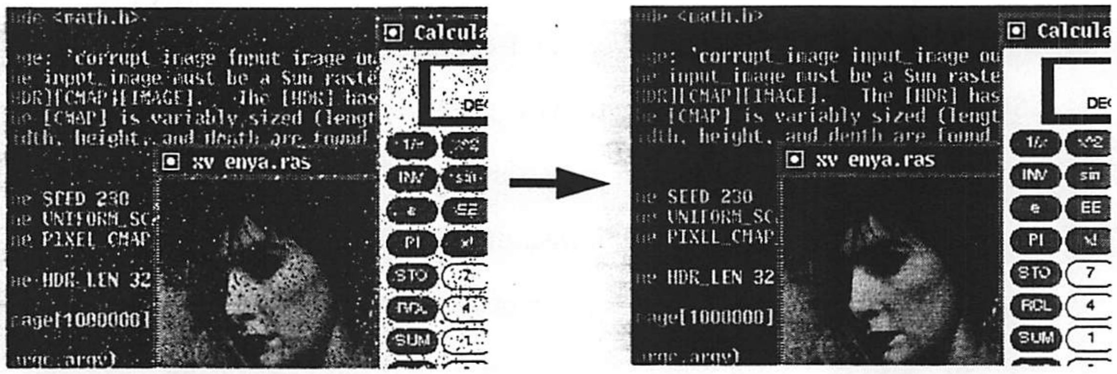
The end result of these observations is that payloads bearing error-resilient information are still useful at the receiver for decoding despite severe *BER*'s. In addition, in a JSCC system, UEP on packet payloads will leave some bits lightly error protected, resulting in possibly many packets being in error after UEP decoding. Therefore, in a JSCC system, it is essential that corrupt information be forwarded to the endpoint application for decoding. Given error-resilient data, the end-to-end distortion can actually be reduced by accepting corrupt packet data in comparison to throwing this corrupt data away.

Forwarding and processing of corrupt packet data can also be motivated from the perspective of delay, not just end-to-end distortion. An end user willing to tolerate noisy reconstructed images can receive and display corrupt information far faster than an end user who will only accept fully reliable delivery of images and therefore must wait for an error-free version due to retransmissions. The end user in a JSCC system will be able to interact as soon as possible with a noisy representation of an image over a wide range of *BER*'s.

The overall conclusion of Chapter 4 is that error-tolerant image coding, UEP, forwarding of corrupt error-resilient information, and application-level decoding of corrupt packet data together constitute an approach which is more likely to provide continuous interactivity to the end user over a wide range of *BER*'s than a system which practices aggressive compression, aggressive FEC, discards corrupt packets, and insists that an application process only error-free data.

1.3 Progressively reliable packet delivery

Our end-to-end protocol is tightly integrated into the JSCC design philosophy in order to realize the advantages of JSCC in terms of reducing the distortion and perceived latency of delay-sensitive image data encoded by constrained-complexity systems and delivered



```
de <math.h>
e: 'corrupt image input_image ou
e input image must be a Sun raste
DR][CMAP][IMAGE].   The [HDR] has
e [CMAP] is variably sized (lengt
dth, height, and depth are found
e SEED 230
e UNIFORM_SC
e PIXEL_CMAP
e HDR_LEN 32
age[1000000]
rgc, argv)
```

Figure 1.3 Progressively reliable packet transport delivers an initial possibly noisy version of a packet quickly to the receiver. Later, the protocol employs retransmissions to progressively improve the reliability of the delivered packet. An image-based multimedia application can use a progressively reliable protocol to display a noisy initial image (left) for immediate interactivity. Later, any persistent artifacts on the screen can be removed by displaying progressively cleaner image data provided by the protocol (right). The 8 bits/pixel color-mapped image is corrupted at *BER* 10^{-2}.

over very noisy wireless channels. Initially, our protocol forwards a possibly noisy first version of a packet to the receiver. For bursty multimedia, artifacts due to reconstruction of noisy data may persist on the screen for a very long time, until further user activity overwrites or redraws that portion of the screen. To remove long-term artifacts, our protocol delivers successively refined versions of a packet to the application by sending retransmission redundancy at some later time. Thus, each packet of image data is delivered to the application in a *progressively reliable* fashion.

A multimedia application like Web-based image browsing would employ a progressively reliable transport protocol to deliver a noisy initial version of an image for quick interactivity. The assumption is that the application will code its image data in an error-tolerant manner in order to make use of corrupt forwarded data at the receiver. Eventually, progressively reliable packet delivery will deliver a sufficiently clean version of an image to remove any persistent artifacts. The overall effect perceived by the end user is shown in

Figure 1.3. Essentially, the protocol is trading off the user's subjective priorities of delay and distortion. Low-latency high-distortion packet delivery is followed eventually by high-latency low-distortion delivery. The protocol permits the application to satisfy the end user's demand for continuous interaction with a possibly noisy initial image over a wide range of *BER*'s. In addition, the protocol allows the application to asymptotically satisfy the end user's desire for distortionless packet delivery.

In Chapter 5, we identify the following four essential properties of progressively reliable packet delivery:

- Corrupt packets are forwarded to the application
- Multiple versions of each packet are delivered
- The reliability of these multiple versions improves over time (statistically fewer errors with each successive version)
- Different packets are delivered out of order

Each of these properties has an impact on the design of the socket interface between the multimedia application and the protocol. Applications must frame their data into application data units (ADU's). In addition, the error detection applied on the ADU header must be distinct from the error detection applied to the ADU payload. In this way, the protocol can distinguish between packet corruption, i.e. useful ADU's with noisy payloads and error-free headers, and packet loss, i.e. useless ADU's with errors in the header. Furthermore, the property of increasing reliability requires that the receiver have memory so that it can recall the level of noise in previous versions of ADU's before delivering a newer version with fewer errors.

Also in Chapter 5, we discuss three key additional features of progressive reliability which improve its end-to-end delay performance. The transport latency of the first possibly noisy version of an ADU can be reduced by:

- delaying ADU retransmissions to minimize conflict with time-critical delivery of the initial version of an ADU
- cancelling retransmissions associated with out-of-date ADU's
- flow-based QOS scheduling of the sender's traffic by delay and reliability constraints

The first possibly corrupt version of an ADU is classified as delay-sensitive, while the later retransmissions employed to refine the first version are classified as delay-tolerant. By choosing when to send retransmission redundancy, it is possible to translate retransmissions to a time when there is little or no delay-sensitive traffic. Delay-tolerant retransmissions can afford to be shifted around in time with little or no subjective impact. Therefore, the progressively reliable protocol biases its delivery toward delay-sensitive initial delivery of ADU's, and transmits the retransmission redundancy so as not to conflict with initial delivery.

Furthermore, bursty multimedia has the property that new images often overwrite old images. Because our progressively reliable protocol retains data in order to delay ADU retransmissions, some ADU's may contain out-of-date image data. Our progressively reliable protocol allows the user to apply correlation labels to ADU's, so that the protocol can identify when a packet has become stale. This allows the protocol to cancel any further retransmissions of stale data, freeing up the connection to deliver other packet data.

Finally, flow-based scheduling permits the application to define more than just a single stream of data. Instead, a multimedia application can generate a hierarchically coded image, and send different parts of this image progressively along multiple substreams. Each substream has its own quality-of-service (QOS) requirement in terms of delay and reliability. The progressively reliable protocol incorporates a scheduler which can distinguish between the delay requirements of different substreams, as well as the separate delay requirements of the initial and final versions of data within the same substream.

An X windows server was modified to test the validity of these ideas. The X windows server was modified to implement a progressively reliable encoder/decoder and a random bit error channel model. The effect seen on the screen was similar to Figure 1.3, in which any portion of the screen was drawn in an initially noisy fashion, followed eventually by cleanup of artifacts. The X windows server provided a real-time platform to test our ideas concerning the viability of progressive reliability as a concept. We found that we could tolerate the presence of channel distortion in the initial version and still conduct interactive image browsing. Through experimentation, we discovered that the responsiveness of the system could be improved by delaying retransmissions and cancelling stale retransmission data. We constructed a dynamic software tool called a "performance manager" which allowed the user to dynamically adjust the various bit rates, *BER*, and delay parameters within the modified X server during run-time.

As part of our work on the X windows server, we also implemented a progressive colormap source coding algorithm. This permitted some experimentation with the relationship between progressive source coding and progressively reliable packet delivery, i.e. progressive channel coding. The initial version was allowed to suffer not only channel distortion due to bit errors, but was also encoded in a way that introduced source coding distortion. For example, a typical sequence of events would be that the initial version would not only be noisy, but would also be drawn as a black and white version of a color image. The followup redundancy would overwrite the noisy b/w version with a clean color image. We found that the overall effect was tolerable for natural images as long as the luminance magnitude, i.e. the brightness level of the image, was preserved between the original and final versions. More sophisticated versions of progressive source coding were not attempted in this thesis.

We also experimented with video within our modified X server. The cancellation function proved especially useful and demonstrated how continuous media could exploit a progressively reliable protocol.

In Chapter 6, we discuss end-to-end implementation issues regarding a progressively reliable transport protocol. Because such a protocol initially forwards corrupt information, and later cleans these errors up, we call our proposed progressively transport protocol "*Leaky ARQ*". We observe that Leaky ARQ can be implemented as a form of *memory ARQ*, also called *packet combining*, of which *Type-II Hybrid ARQ* is the most well-known example. While implementing Leaky ARQ as a true transport protocol will optimize the protocol's performance, migrating a new layer 4 protocol in the Internet faces some practical challenges. As an alternative, we describe how Leaky ARQ could be implemented within the context of the Internet on top of UDP, though certain modifications to UDP would be required. Finally, we consider the impact of fragmentation, non-sliding windows, and acknowledgments on the design of Leaky ARQ.

2

Latency Bounds For Reliable Network Protocols

In this chapter, we investigate the latency performance of end-to-end network protocols which reliably deliver visual multimedia over connections that include a wireless access link. Multimedia applications, such as Web-based image browsing, often require rapid delivery of individual images in order to convey a genuine sense of interactivity to the application's user. Our goal is to quantify the time necessary to reliably deliver a finite burst of packets corresponding to a single image over a noisy connection. In general, network connections consist of a sequence of concatenated communication links, each characterized by its own distinctive behavior. We consider the special case in which one of these links is a wireless access link.

Network protocols that provide end-to-end reliability, i.e. reliability over a multi-hop connection, are also known as Automatic-Repeat-Request (ARQ) protocols. ARQ protocols implement a closed-loop retransmission scheme that guarantees reliable delivery of packets (within error-detection limits).

In this chapter, we quantify the cost in latency associated with fully reliable packet delivery. First, we consider the latency cost for reliable delivery of a single packet. Later, we investigate the latency cost for reliable delivery of a burst of packets corresponding to a packetized image. Two estimates of the image transfer latency are derived: a "loose" lower

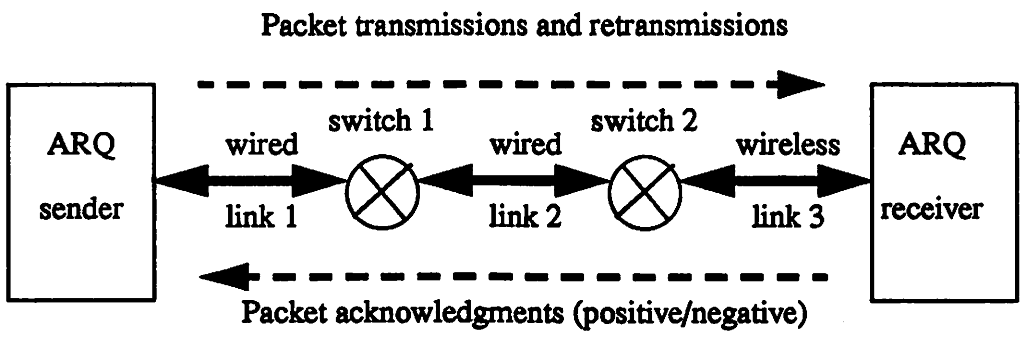

Packet transmissions and retransmissions
ARQ
sender
wired
link 1
switch 1
wired
link 2
switch 2
wireless
link 3
ARQ
receiver
Packet acknowledgments (positive/negative)

Figure 2.1 End-to-end fully reliable delivery over multiple concatenated network links (some wired, some wireless) via closed-loop retransmission-based ARQ protocols.

bound and a "tight" lower bound. Both bounds are based on the idealized Selective-Repeat (SRP) retransmission protocol, the most efficient ARQ protocol. These lower bounds on delay show that the minimum time to reliably deliver an image over a high bit error rate channel can be so large that real-time delivery is largely precluded. The ARQ analysis of this chapter serves as the basis for analyzing the hybrid FEC/ARQ protocol of a later chapter.

2.1 Introduction to ARQ protocols

Retransmission-based ARQ protocols ensure fully reliable delivery of individual images over connections which include a noisy wireless channel. Each of these closed-loop techniques sends and resends packets until they have been correctly acknowledged by the receiver. Figure 2.1 illustrates a sender transmitting packets and their repetitions in the forward direction across a multi-hop connection that includes a wireless link. In the reverse direction, the receiver sends the acknowledgments that correspond to the transmitter's packets. Some well-known examples of ARQ protocols include Stop-and-Wait (SW), Go-Back-N (GBN), Selective Repeat (SRP) [129], and the Internet's reliable Transmission Control Protocol (TCP) [25], each of which will be discussed later in this chapter.

In ARQ, after a packet is transmitted to the receiver, the sender waits for an acknowledgment for that packet from the receiver. If the receiver sends back a positive acknowledgment that the packet has been correctly received, then the protocol proceeds onwards to send a new packet waiting in the sender's queue. If the receiver sends back a negative acknowledgment (NACK) that the packet was corrupted, or if the sender times out waiting for any acknowledgment, then the sender retransmits the original packet. This describes a simple Stop-and-Wait (SW) protocol.

Windowed protocols like GBN and SRP improve upon the SW protocol by permitting multiple packets (i.e. the sender's window size $W \geq 1$) to propagate toward the receiver. Rather than wait the full roundtrip time until a single packet is acknowledged before sending the next packet, SRP tries to keep the transmission pipe full by continuously transmitting multiple distinct packets.

SRP's response to the loss of a packet is to retransmit only the lost packet. For other windowed protocols like GBN and TCP, loss of a given packet can trigger retransmissions of packets besides the lost packet. For example, in GBN, a packet loss causes the protocol to go back in the packet sequence to the lost packet, and start retransmitting sequentially from the position of the lost packet. Some packets may wind up being retransmitted multiple times even though they have already been correctly delivered to the receiver. These unnecessary retransmissions lead to inefficient management of the connection's bandwidth. In contrast, the number of unnecessary retransmissions in SRP is zero, just as for SW. Unlike SW, SRP permits a window size $W \geq 1$. The combination of windowed performance and zero unnecessary retransmissions make SRP the most efficient ARQ protocol possible. Moreover, the number of retransmissions in SRP for a given packet is independent of the behavior of all other packets. This leads to a simplified analysis of the expected or average number of retransmissions per packet.

A special form of SRP called *Ideal* SRP assumes transmitter buffers and receiver buffers have infinite length. Ideal SRP also assumes that the sender's window size is larger than a single packet's roundtrip time (more precisely, its equivalent measured in number of fixed-length packets). This assumption allows the downstream pipe to be continuously filled or *saturated* with multiple distinct packets.

How do we quantify the latency cost of a particular retransmission protocol over a multi-hop connection that includes a noisy wireless link? Our approach is to view the wireless link as the limiting factor in the connection both in terms of noise and bandwidth. Hence, we confine our protocol analysis to this single link. This formulation of the problem permits two interpretations. First, our analysis could be interpreted to apply to a link-layer ARQ protocol operating over a single wireless link. Alternatively, our analysis could be interpreted as a lower bound on performance of an end-to-end protocol, in which only the impact of the wireless link is studied, while the contributions to latency caused by the other wired links have been abstracted out of the problem.

Our strategy is to develop a lower bound on the image transfer latency that applies over all protocols, rather than investigate each protocol individually. From our previous discussion, we observed that SRP is the most efficient protocol among all possible repetition-based non-hybrid ARQ protocols like GBN and SW [78]. Optimal efficiency translates into minimal delivery time. Therefore, by basing our latency analysis of reliable protocols on ideal SRP, we will have determined a lower bound on latency over all protocols. Any other protocol will incur a higher cost in delivery time. Ideal SRP has the additional property that it is mathematically tractable. We will apply this bounding strategy to both our single-packet and multi-packet analyses of ARQ protocols.

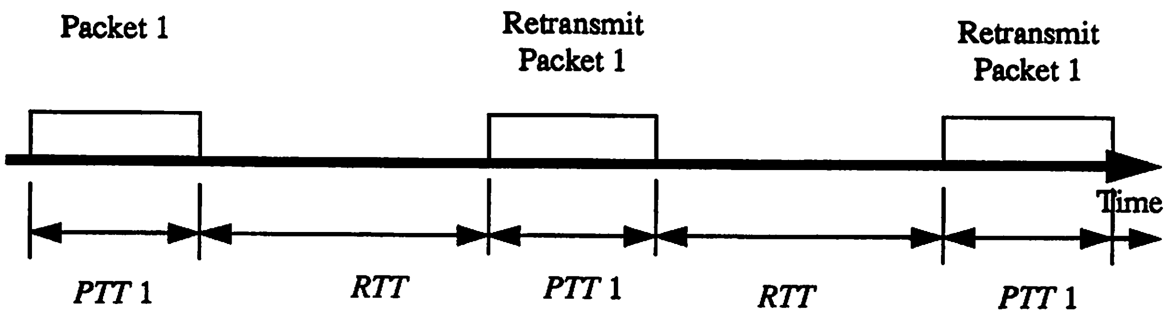


Figure 2.2 A typical retransmission protocol is shown. Each (re)transmission of a packet takes time *PTT+RTT*, where *PTT*=packet transmission time due to the packet's length, and *RTT*=roundtrip time that the sender has to wait after sending the packet before an acknowledgment could arrive. In general, multiple packets may propagate simultaneously toward the receiver.

2.2 Quantifying the single-packet latency of ideal SRP

In this section, we quantify the average latency experienced by a single packet reliably delivered by an ideal SRP ARQ protocol over a noisy channel. A typical retransmission-based ARQ protocol is shown in Figure 2.2. Let a packet consist of a block of K bits. Assuming independent Bernoulli trials and a fixed Bit Error Rate (BER), the probability of a K-bit error-free packet is $P_g = (1 - BER)^K$. If we define a random variable N_{packet} = # trials until the first good block is delivered, then N_{packet} has a geometric distribution, i.e. $P[N_{packet} = i] = (1 - P_g)^{i-1} \cdot P_g$, for i positive integers. The expectation of a geometric N_{packet} is well known, so that the average # of trials until the first good packet is received is given by

$$E[N_{packet}] = \frac{1}{P_g} = \frac{1}{(1 - BER)^K} \tag{2-1}$$

For ideal SRP, the number of retransmissions for each packet can be considered to be independent of all other packets. A conservative estimate of the average delay experienced by a single fixed-length packet is simply the product of the average number of transmissions $E[N_{packet}]$ and the time required per packet. This conservative estimate ignores contention at the sender among different packets, which will delay each packet beyond our conservative estimate. However, as we shall see in the next section, this contention among different packets is automatically taken into account during our analysis of the delay of a group of packets corresponding to an image.

We can now obtain the average delay per packet. Define *PTT* as the fixed packet transmission time due to the packet's size. Let *RTT* be the additional roundtrip time that the sender has to wait *after* each packet is sent *until* its acknowledgment returns to the sender (stating whether or not the packet has been correctly received). In general, *RTT* is a stochastic quantity consisting of at least the forward and reverse direction network queueing delays along the multi-hop connection, the protocol processing and operating system delays at the connection endpoints [22], and the reverse and forward direction propagation delays. Also, even though a sender may be ready to retransmit a packet, congestion at the sender due to a large number of packets awaiting transmission will add stochastic queuing delay to *RTT*. It can be easily verified that the typical range of *RTT* across a multi-hop Internet connection varies from tens of milliseconds to many hundreds of milliseconds. Let us assume that *RTT* is fixed, so we can focus our analysis on the retransmission policy. *PTT* and *RTT* are pictured in Figure 2.2.

Define T_{packet} as the time between the commencement of packet transmissions and the reception of the packet's positive acknowledgment from the receiver. Our conservative estimate of the average delay per packet is given by

$$E[T_{packet}] = E[N_{packet}] \cdot (PTT + RTT) \qquad (2\text{-}2)$$

In order to gain some intuition about the effects of bandwidth, probability of bit error, and header overhead on the single-packet latency $E[T_{packet}]$, we temporarily make some simplifying assumptions. First, let us for the time being assume *RTT*=0. This means that our estimate of the single-packet latency (*PTT*) will be a lower bound, since it will not incorporate many sources of delay arising from *RTT*. We will return to the impact of *RTT* on latency in Section 2.4, where we evaluate a tight lower bound on the latency of a packetized image for several real-world values of *RTT*.

Also, let us choose the bandwidth and probability of bit error *BER* to represent the weakest links in the connection, with the wireless link being the most likely limiting factor in both instances. If we define *BW* = wireless bit rate, *H* = # header overhead bits/packet, and *P* = # payload bits in the packet, then we can approximate the average delay until delivery of the first error-free packet as

$$E[T_{packet}] \geq E[N_{packet}] \cdot PTT = \frac{1}{(1-BER)^{(P+H)}} \cdot \frac{P+H}{BW} \tag{2-3}$$

Equation (2-3) summarizes the impact of bandwidth, bit error rate, and header overhead on the single-packet latency of ideal SRP. It also represents a lower bound on the expected delay.

Table 1. Minimum average delay estimates (*RTT*=0, ideal SRP) from Equation (2-3) for a single packet as a function of packet payload size *P*

Payload size(bits)	10	100	500	1000
lower bound estimate of $E[T_{packet}]$	0.665 ms	2.99 ms	0.499 sec	139 sec

In Table 1, we illustrate a sampling of the mean delays $E[T_{packet}]$ given by Equation (2-3) as a function of payload size *P*, assuming a raw *BER* of 10^{-2} (a typical design point of digital cellular systems [28][149]), header size *H* = 100 bits (TCP has at least 20 bytes of header when uncompressed, UDP has 8 bytes of header), and wireless *BW* = 500 kbit/s (on the high end of current wireless bit rates [94]). The results indicate that large packets that are longer than several hundred bits experience prohibitively large non-real-time delay. Since the *BER* is 10^{-2}, then on average large packets on the order of several hundred bits will contain multiple bit errors, forcing many retransmissions and thereby increasing the delivery time. Also, the single-packet delay rises exponentially as a function of the payload size *P*. This exponential behavior suggests that a large image, on the order of many tens of thousands of bits, should be packetized into many small packets instead of a few large packets over a noisy wireless link. In the next section, we will examine in more detail how this exponential behavior influences multi-packet image delivery.

While our protocol analysis has focused on latency performance, previous work on ideal SRP has focused on characterizing SRP's throughput performance (also called efficiency or utilization). The throughput η is defined as the percentage of time spent transmitting *new* packets [78][145]. High throughput implies that less time is spent on retransmissions ($E[N_{packet}]$ is small) and/or waiting for an acknowledgment or time-out, and proportionately more time is spent on transmitting packets correctly the first time. Conversely, if the expected number of transmissions per packet $E[N_{packet}]$ is large, then the throughput will be low. Clearly, $\eta \propto \frac{1}{E[N_{packet}]}$, where the precise relation depends on the protocol. Therefore, the single packet delay is closely related to the throughput through $E[N_{packet}]$. In fact, our estimate of the delay has leveraged off of previous throughput derivations in order to obtain $E[N_{packet}]$. We will have more to say in the next section about the relationship between throughput and our loose estimate of multi-packet image latency.

Finally, single packet latency analyses of SW and GBN have been performed in [116][134]. Single packet latency analyses of SRP have been performed in [78][145]. Other authors have undertaken a queueing analysis of the average wait time for a single packet in SRP [72][117]. Our derivation of Equation (2-3) has been designed to emphasize the intuitive dependence of latency upon *BER*, *BW* and overhead *H*, which will aid in understanding the multi-packet SRP analyses of the next two sections.

2.3 A loose lower bound on the image transfer latency of ideal SRP

In this section, we extend our single-packet latency analysis of ideal SRP to consider a long image block that may need to be fragmented into multiple smaller packets for delivery by the network protocol. We develop a "loose" lower bound that estimates the image transfer latency. The loose bound provides insight into the contributions to delay from the bandwidth, *BER*, and header overhead factors. We show how our loose bound is related to the throughput. The loose bound is also used to determine a closed-form solution that specifies the optimal way to fragment an image so that delay is minimized.

For an image fragmented into multiple packets, the average delivery time for the image is not simply the sum of the individual packet delivery times, due to *overlapping* packet deliveries. Suppose T_i is defined as the interval between the start of packet P_i's transmissions and final reception of an acknowledgment of error-free delivery for that packet. Ideal SRP fills any given packet's *RTT* interval with the transmissions and repetitions of other packets, in an effort to keep the downstream pipe completely filled. Therefore, an estimate for the expected delay of a multi-packet burst which sums $E[T_i]$ over the individual packets will overestimate the delay, since many overlapping times would be counted multiple times. This overlapping is illustrated in Figure 2.3, in which packet P_1's retransmissions overlap with packet P_2's repetitions.

Figure 2.3 suggests several ways to estimate the delivery time for a packet burst that avoids overlapping summations. A fairly conservative estimate would simply sum the total number of packet transmission times *PTT*. By counting the number of *PTT* slots occupied during the transmission of a group of packets corresponding to an image, we obtain a loose lower bound estimate of the average image transfer delay. Note that such an estimate automatically incorporates the delay due to contention for bandwidth among different

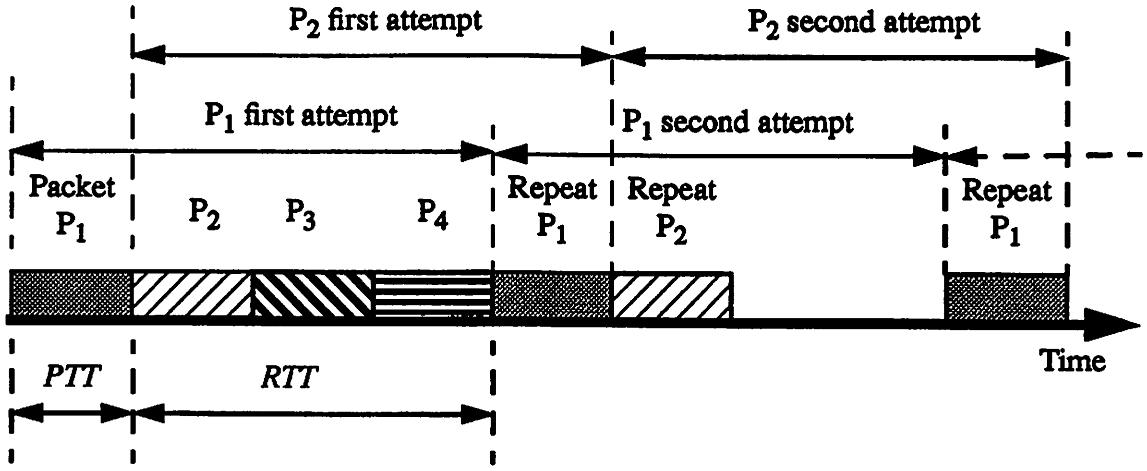


Figure 2.3 Retransmissions of different packets overlap in time. Define T_i as the time between first transmission and final acknowledgment of packet P_i. The expected delay *E*[packet burst] of packets P_1 through P_4 is not the sum of each packets' $E[T_i]$, since such an estimate would recount the same intervals due to overlapping T_i. *PTT* = (fixed-length) packet transmission time. *RTT* = (fixed-length) roundtrip time after packet is sent until its acknowledgment returns.

packets at the sender. Since packets are interleaved, and ideal SRP tries to keep the transmission pipe filled with new transmissions and retransmissions, then the *PTT* time slots by which a given packet is delayed due to contention are occupied by transmission of other packets. The contention-based delay for each packet is automatically included by counting the number of occupied *PTT* slots for the entire image burst.

Our loose lower bound estimate ignores the interstitial space (e.g. between the second transmission of P_2 and the third transmission of P_1 in the figure) caused by the protocol's inability to keep the transmission pipe filled when the amount of data left to transmit is less than *RTT*. In the next section, we will pursue analysis of a tighter delay estimate that accounts for the interstitial space or idle time.

Continuing with our loose estimate of the average image delay, we define T_{image} as the time to transmit a packetized image burst, including fragment-based retransmissions. Let

F be the number of fragments that an image is divided into. Define $E[N_{fragment}]$ as the expected number of retransmissions for each fixed-length fragment.

The loose lower bound on delay is obtained by counting the amount of "air time" when packets are actually transmitting over the channel. This is equivalent to counting the number of packet transmissions *PTT* that occur, i.e. the time occupied by the shaded packets in Figure 2.3. Each fragment will be repeated on average $E[N_{fragment}]$ times. Therefore, $E[N_{fragment}] \cdot PTT$ is a conservative and non-overlapping estimate of the reliable per-packet delivery time. The non-overlapping characteristic means that the per-packet delivery times can be summed across the F packets. In contrast, the overlapping estimate of per-packet delay $E[N_{fragment}] \cdot (PTT + RTT)$ from Equation (2-2) would have precluded summation of the individual delays. Since all F packet fragments are assumed to be the same length, then $E[N_{fragment}]$ is the same across all packets, then the loose estimate of the average image delay is given by the product of the number of fragments per image, the average number of retransmissions per fragment, and *PTT*, i.e. $F \cdot E[N_{fragment}] \cdot PTT$.

If we fragment the original image size I by a factor F, where H = number of header bits then each packetized fragment will be size $(I/F + H)$ bits. Then Equation (2-1) is revised into

$$E[N_{fragment}] = \frac{1}{(1-BER)^{\left(\frac{I}{F} \div H\right)}} \qquad (2\text{-}4)$$

The loose estimate of the average reliable image delay is therefore given by

$$E[T_{\text{image}}] \geq F \cdot E[N_{fragment}] \cdot PTT_{fragment} \qquad (2\text{-}5)$$

$$= \frac{F}{(1-BER)^{\left(\frac{I}{F}+H\right)}} \cdot \frac{\frac{I}{F}+H}{BW} = \frac{1}{(1-BER)^{\left(\frac{I}{F}+H\right)}} \cdot \frac{I+F \cdot H}{BW}$$

Equation (2-5) neatly summarizes the various contributions to delay. It can be partitioned into a *BER factor* $(1 - BER)^{-\left(\frac{I}{F}+H\right)}$ and a *BW factor* $\frac{I+F \cdot H}{BW}$ to emphasize the independent contributions to delay from the channel noise and the limited channel bandwidth respectively. The header overhead is shown to affect both factors.

This loose estimate is a lower bound on latency in two ways. First, it ignores the interstitial spaces shown in Figure 2.3 caused by the inability of a finite image burst to keep the protocol's transmission pipe completely filled. The interstitial spaces appear near the tail end of the image burst, causing partially empty transmission pipes. Second, it is a lower bound over all protocols, because we have assumed ideal SRP, which minimizes the number of unnecessary retransmissions, and hence minimizes average transmission time of a packet.

Table 2. Minimum average delay estimates for fragmented ideal SRP delivery of a 20 kbit image as a function of the image fragmentation factor *F*.

Fragmentation factor F	10	20	50	100	1000	10000
non-header payload size of packet (bits)	2000	1000	400	200	20	2
"*BER factor*"	1E9	6E4	152	20.4	3.34	2.79
"*BW factor*"	0.042	0.04	0.05	0.06	0.24	2.04
lower bound estimate of $E[T_{image}]$	6E7 sec	2.78E3 sec	7.61 sec	1.22 sec	0.802 sec	5.69 sec

In Table 2, we calculate a sampling of Equation (2-5) 's lower bound on $E[T_{image}]$ for a fragmented image as a function of the fragmentation factor F. The parameters BER, H, and BW have the same values as before, and we assume an image size $I = 20000$ bits (say a 200x200 pixel small color image compressed to 0.5 bits/pixel).

Table 2 shows that the delay decreases exponentially as F increases, reaching a minimum-delay value F_{opt} for the fragmentation factor (on the order of F=1000), before rising

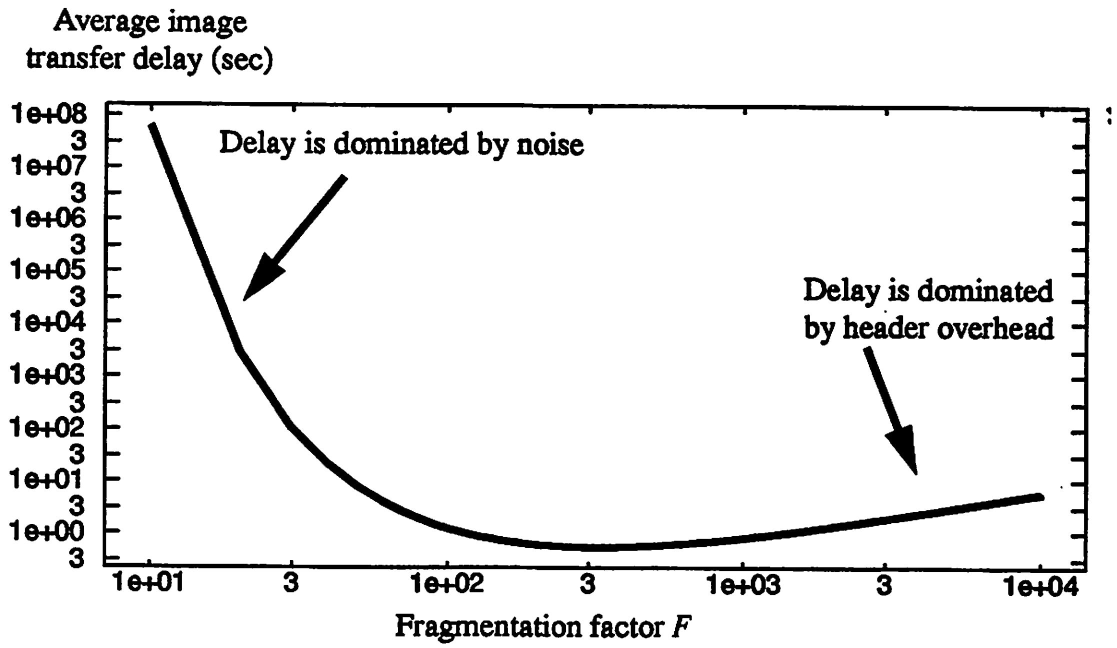


Figure 2.4 Log-log plot of average delay of the image $E[T_{image}]$ vs. image fragmentation factor F using Equation (2-5). Image size = 20 kbits, $BER = 10^{-2}$, BW = 500 kbit/s, and header H = 100 bits. A minimum delay of about half a second emerges around a fragmentation factor of ~300.

again for large *F*. We plot Equation (2-5) as a function of the image fragmentation factor *F* in Figure 2.4. The behavior of Figure 2.4 is governed by the two dominant factors due to *BW* and *BER*. The *BER factor* declines exponentially as a function of *F*, while the *BW factor* increases only linearly with *F*. Hence, for very small *F* (large packets), delay is astronomical due to the *BER factor* (many retransmissions). However, as *F* increases (smaller packets), the retransmission delay due to the *BER* recedes exponentially, so that smaller packets help to decrease image transfer latency. As *F* becomes very large (in the extreme, 1 bit payloads!), the *BW factor* causes the latency to rise again, due to the overhead caused by the header *H* in comparison to the ultra-small payloads.

For the sake of completeness, we next derive a closed-form solution for the minimum-delay fragmentation value. First, we take the derivative of Equation (2-5) with respect to *F*. This results in a quadratic in *F* which can then be set equal to zero, $\frac{\partial}{\partial F}(Equation(2-5)) = 0$. Our optimal *F* is given by

$$F_{opt} = \frac{I}{2} \cdot \ln(1 - BER) \cdot \left(-1 - \sqrt{1 - \frac{4}{H \cdot \ln(1 - BER)}} \right) \tag{2-6}$$

where we have eliminated the second root. F_{opt} is shown to be a function of the image size I, BER, and header overhead H.

Substituting default values, we obtain $F_{opt} = 324$. This value can be verified by consulting Figure 2.4, which yields a graphical minimum near F=300. At this value F_{opt}, our optimal packet size (payload + header) is about 162 bits. The minimal-delay payload is somewhat surprisingly smaller than the header overhead. This is because the protocol needs to generate very small packets in order to avoid the exponential delay contributed by the channel noise, even at the cost of suffering significant header overhead.

The minimum delay at F_{opt} is ~0.53 seconds. If we estimate our real-time delivery bound to be about 500 msec, then we are just barely meeting our objective. If we assume a more realistic bound of 100 msec is needed to support "instantaneous" interactivity, then ideal SRP will fail to meet the interactive delay objective by almost half a second even under the most optimistic assumptions.

Previous work that derived the optimal packet length has been based on taking the derivative of a throughput equation with respect to packet length in order to maximize throughput (for SW [129] and GBN [116]). The maximal-throughput packet length is found to be primarily a function of BER and header overhead H, though this depends on the protocol. This is confirmed by our SRP analysis, in which the optimum payload size $\frac{I}{F_{opt}}$ is found to be only a function of BER and H. Another time-based derivation of optimal packet length minimizes the expected "wasted time" but does not consider header overhead [89].

Finally, we relate Equation (2-5) to previous work on ideal SRP that has focused on characterizing the protocol's throughput performance η_{SRP}, defined as $\eta_{SRP} = \frac{PTT}{E[T_{packet}]}$ [78][145]. Throughput analyses observe that any given packet's roundtrip time *RTT* is filled with the transmissions and repetitions of other packets. Therefore, $E[T_{packet}]$ should only count the actual "air time" necessary to transmit and retransmit a given packet, i.e. $E[T_{packet}] = E[N] \cdot PTT$. Hence, η_{SRP} is given by $\eta_{SRP} = \frac{PTT}{E[N] \cdot PTT} = P_g$. The effective channel bit rate available to the sender is $\eta_{SRP} \cdot BW$. A simple estimate for the delay of an image *I* obtained from the throughput would be $\frac{I}{\eta_{SRP} \cdot BW} = \frac{1}{P_g} \cdot \frac{I}{BW} = F \cdot \frac{1}{P_g} \cdot \frac{I/F}{BW}$, which is essentially Equation (2-5) minus the overhead. Thus, we see that our time-based derivation is closely related to throughput expressions. Our delay-based approach explicitly shows the dependence of latency on *BW*, *BER*, and *H*. In addition, our time-based reasoning reveals that Equation (2-5) (and the equivalent throughput-derived delay) underestimates the delay, foreshadowing our analysis of the next section. Reliance on previous throughput-based derivations would have missed the effect of a partially empty pipe for finite image bursts.

2.4 A tight lower bound on the image transfer latency of ideal SRP

We develop in this section a more complete estimate of a protocol's delivery time for a finite image burst that also incorporates a measure of the idle time caused by a partially empty transmission pipe. In Figure 2.5, we depict how a large burst of packets corresponding to an image can initially keep the pipe full, under certain assumptions. However, the tail end of the burst eventually causes interstitial spaces to appear in the delivery stream. This tail effect occurs regardless of the burst size, and is a function of the roundtrip time *RTT*. Our goal is to quantify the contribution of this tail effect to the overall latency.

Define *RTT* as the roundtrip time after a packet transmission until its acknowledgment returns, and *PTT* as the fixed-length packet transmission time. We assume *RTT* is a constant value in the upcoming derivation. Define the total roundtrip time *TRTT* as

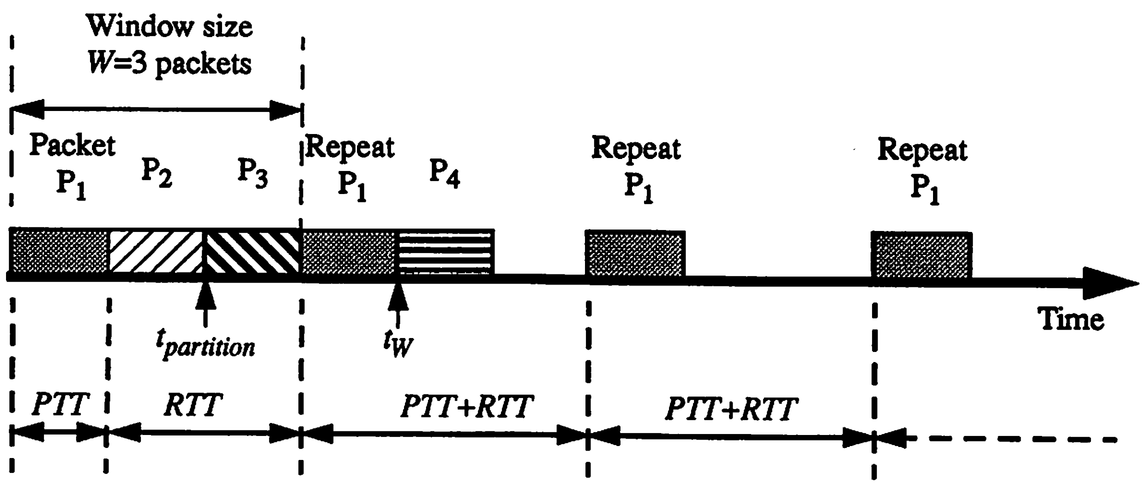


Figure 2.5 A finite burst of packets cannot keep the protocol's transmission pipe completely full. Assume that the window size *W* (# packets allowed in the pipe) and image burst of length *I* packets are large enough to initially keep the protocol's pipe full, i.e. $W \geq PTT + RTT$ and $I \geq PTT + RTT$, where *PTT* and *RTT* are measured in packet units. Eventually, the final *PTT+RTT* packets will be delivered through a partially empty pipe. In the figure, *I*=4, *W*=3, *PTT*=1, and *RTT*=2 equivalent packet lengths. All packets except P_1 are delivered correctly the first time. The significance of $t_{partition}$ and t_W is explained later in this section.

$TRTT = PTT + RTT$. Define the sender's window size W as the number of packets that are allowed to be simultaneously propagating toward the receiver in the transmission pipe. We optimistically assume that the fixed window size is large enough so that the protocol can completely fill any roundtrip time with multiple packets. Hence, W needs to exceed the number of packets that could fit into $TRTT$, i.e. $W \geq \left\lfloor \frac{PTT + RTT}{PTT} \right\rfloor$.

Furthermore, we assume that only one version of each packet is in the pipe in any given total roundtrip interval. For example, two retransmissions of a given packet are not allowed simultaneously in the pipe. Certain ARQ protocols do indeed fill the pipe with multiple copies of a given packet [13][62][133]. We do not consider such protocols, and instead note that ideal SRP satisfies our restriction that only one version of each packet may be allowed in the transmission pipe.

Each of these packets is formed by fragmenting an image into F packets, where each packet is of length $\frac{I}{F} + H$, counting header overhead H. Given ideal SRP with sufficiently large windows, then two scenarios are possible. Either the image size I is large enough to allow ideal SRP to initially fill the transmission pipe, or the image is too small to ever fill the pipe. In the first scenario, there are enough distinct packets from the fragmented image to fill the transmission pipe during the first *TRTT*. This condition can be written as $F \cdot PTT \geq TRTT$, or equivalently $\frac{PTT + RTT}{F \cdot PTT} \leq 1$. Let us define α as the ratio of the total roundtrip time to image burst size:

$$\alpha = \frac{PTT + RTT}{F \cdot PTT} \qquad (2\text{-}7)$$

2.4.1 Image burst is large enough to initially fill the transmission pipe

We begin by analyzing the case in which $\alpha \leq 1$, namely the image is large enough to initially fill the pipe during one complete roundtrip time *TRTT*. For example, Figure 2.5 portrays a situation in which $\alpha \leq 1$. The packetized image burst (4 packets long) is larger than *TRTT* (equivalent to 3 packets). The packetized image is able to initially fill the pipe, but only until one of the four packets is correctly received. At that point, the final *TRTT* (3) packets would be unable to keep the pipe filled.

As the example suggests, one approach to obtaining a tighter latency bound would be to try to partition our analysis into two non-overlapping estimates: calculate the delay accumulated while there is enough image data to keep the pipe full/saturated; then estimate the delivery time of the final "tail" packets through a non-saturated pipe. Without loss of generality, we can set the window size to $W = \left\lfloor \frac{PTT + RTT}{PTT} \right\rfloor$, which is the minimal window size that still permits a full pipe, provided there is a sufficient amount of source data.

2.4.1.1 Delay due to the first *F-W* packets that keep the pipe full

We observe that the final W packets will be unable to keep the pipe full. Since the image is fragmented into F packets, then F-W packets are transmitted under saturated pipe conditions. These F-W packets are also the first to be correctly delivered. Each of the first F-W correctly received packets will be retransmitted on average $E[N]$ times, where $E[N]$ is given by Equation (2-4) . Since the pipe is full, then the expected time per packet is $E[N] \cdot PTT$, and the total delay for the F-W packets is given by $(F - W) \cdot E[N] \cdot PTT$. This expression matches the conservative *PTT*-only estimate of the previous section (see Equation (2-5)) for the first F-W packets.

2.4.1.2 Delay due to the last *W* packets over a partially empty pipe

As the packetized image flows through the pipe, eventually the first moment in time will arrive when precisely W packets are remaining in the sender's queue and only one of these packets still awaits its first transmission. We shall label this time t_W. We contend that there exists a critical instant $t_{partition}$ related to t_W that can serve as the partitioning point which cleanly separates analyses of saturated and non-saturated pipes. First, we observe that some of the W packets remaining in the tail may already have a history of several transmissions by t_W. For example, in Figure 2.5, t_W represents the first time that precisely W=3 packets are left in the sender's queue and one packet (P_4) has yet to start transmitting. At this instant, packet P_1 has already been transmitted twice, and P_3 has been transmitted once. This history of unsuccessful transmissions could complicate the analysis if there was memory (in the probabilistic sense) in the ARQ process, so that the probability of successful transmission varied depending upon a packet's repetition history. Fortunately, for any of the W packets, the number of attempts N_i (i=1..W) until the first successful transmission is governed by a geometric distribution, which is known to be memoryless [154], i.e. $Prob[\langle N_i = (M+J) | N_i \geq J \rangle] = Prob[N_i = M]$. Therefore, a packet

P_i's history of J prior unsuccessful transmissions does not affect its future number M of unsuccessful retransmissions until the first successful attempt. Moreover, the memoryless property identifies this future distribution as still being geometric.

Based on the memoryless argument, our approach to analyzing the packet tail is to ignore previous unsuccessful attempts and start our analysis as if W packets had just arrived in the sender's queue at time $t_{partition}$. We still need to relate $t_{partition}$ to t_W. Figure 2.5 shows that if we backtrack in time from t_W by one roundtrip time RTT, then the final group of W packets can be viewed as initiating their transmissions at $t_{partition} = t_W - RTT$. The memoryless property permits us to ignore any unsuccessful attempts prior to $t_{partition}$, so that the final W packets appear from a probabilistic perspective as if they're starting transmissions for the first time.

In the final group of W delivered packets, there will always be one "last" packet which is the final one to be correctly delivered. Let N_{last} be defined as the number of transmissions experienced by this last packet. First, we would like to determine its distribution $Prob[N_{last} = M]$. From the distribution, we can find the average number of transmissions $E[N_{last}]$, which is useful for deriving the delivery time of the entire burst of W packets.

If N_i (i=1..W) represent the number of repetitions for each of the independent and identically distributed W packets, then $N_{last} = max(N_1, N_2, \ldots, N_W)$. Most basic probability texts show how to obtain the distribution of N_{last} from the distributions of N_i. We mention the salient steps here. The generic geometric random variable N is used to represent the i.i.d. geometric N_i. First, we find the cumulative distribution function for N_{last}:

$$Prob[N_{last} \le M] = Prob[N_1 \le M, \ldots, N_W \le M] = \{Prob[N \le M]\}^W \tag{2-8}$$

Next, we obtain the probability distribution of N_{last} from the c.d.f.:

$$\begin{aligned} Prob[N_{last} = M] &= Prob[N_{last} \le M] - Prob[N_{last} \le M-1] \\ &= [Prob[N \le M]]^{W} - [Prob[N \le M-1]]^{W} \end{aligned} \tag{2-9}$$

Let P_g represent the probability of an error-free $(I/F+H)$-bit packet, defined by $P_g = (1-BER)^{I/F+H}$. The c.d.f. of N is given by

$$Prob[N \le M] = \sum_{i=1}^{M} (1-P_g)^{i-1} \cdot P_g \tag{2-10}$$

Finally, the expected value of N_{last} can be found from Equation (2-9) and Equation (2-10) :

$$E[N_{last}] = \sum_{M=1}^{\infty} M \cdot Prob[N_{last} = M] \tag{2-11}$$

The expected time required to transmit the last packet is simply the product of the expected number of repetitions and the total roundtrip delay *TRTT* per repetition, i.e. $E[N_{last}] \cdot (PTT+RTT)$. This expression also represents the delivery time for the entire set of *W* packets, because the last packet by definition is delayed the longest and therefore spans the delivery periods of each of the other *W*-1 packets.

Note that by using the concept of a "last" packet to focus our analysis, we have avoided the possibly complex task of having to quantify the interstitial spaces that arise in a partially empty pipe. In addition, we have assumed that the protocol does not try to fill the interstitial spaces with multiple copies of a packet, as stutter ARQ is designed to do [13][62][128][133]. To support multi-copy retransmission, ideal SRP would have to be modified to recognize when a pipe is about to become idle, and would then have to initiate a specific stuttering strategy. This would likely increase $E[N_{last}]$, but would also manage to reduce the latency, because the additional repetitions actually improve the speed of delivery over a partially empty pipe in a single-user environment. We do not pursue this possibility further.

2.4.1.3 Overall tight bound estimate on image transfer delay

Adding the two contributions to delay from the saturated pipe and the non-saturated pipe from the preceding two subsections, we obtain the following estimate of the delivery latency for large images:

$$E[T_{\text{image}}] \geq (F - W) \cdot E[N] \cdot PTT + E[N_{last}] \cdot (PTT + RTT) \qquad (\alpha \leq 1) \qquad \text{(2-12)}$$

The right hand side of Equation (2-12) is a lower bound on the image delay for several reasons. First, we have neglected any retransmissions prior to $t_{partition}$ associated with the final W packets, which will contribute to the delay. Second, we assumed that the window size was large enough to permit a full pipe, which may not happen in practice. Third, we failed to include in Equation (2-12) how long the "last" packet has to wait in the sender's queue after $t_{partition}$ before it starts its retransmission cycle. Since the "last" packet could turn out to be the last in the sequence of W packets to begin transmitting after $t_{partition}$, then this packet will have to wait up to $(W - 1) \cdot PTT$ seconds (almost one full *TRTT*) before its retransmission cycle can start. The term $E[N_{last}] \cdot (PTT + RTT)$ only measures the delay due to the retransmission cycle, and fails to capture this waiting time.

Equation (2-12) is a *tighter* bound on the image transfer latency than Equation (2-5) because the effect on delay due to a partially empty pipe is included. Also, the assumption of ideal SRP makes Equation (2-12) a lower bound over all protocols.

We should note that Equation (2-12) measures the delay between the start of image transmission and the final return to the sender of the last packet's acknowledgment. The perceived delivery time measured from sender to receiver will differ because the final return trip (actually, there may be several attempts to return the last packet's acknowledgment on the reverse channel) of the last acknowledgment is not included in the sender-to-receiver delivery time. We do not pursue this alternative definition of image transfer latency.

2.4.2 Image burst is too small to ever fill the transmission pipe

When $\alpha > 1$, then the image size is too small to initially fill the pipe during the first *TRTT*. The partially empty pipe suggests again that we apply a "last-packet" latency analysis. First, let $W = \left\lfloor \frac{PTT + RTT}{PTT} \right\rfloor$, so that the window size *W* is large enough to permit a full pipe, even though there is not enough image data. There exists a "last" packet among the *F* image fragments which is the final one reliably delivered. Define the number of transmissions of this last packet as $N_{last(\alpha > 1)} = max(N_1, N_2, \ldots, N_F)$. Following the same reasoning as before, the distribution of the number of transmissions N_{last} for $\alpha > 1$ is given by

$$Prob[N_{last(\alpha > 1)} = M] = [Prob[N \le M]]^F - [Prob[N \le M - 1]]^F \qquad (\alpha > 1) \qquad (2\text{-}13)$$

$Prob[N \le M]$ can be obtained from Equation (2-10). $E[N_{last(\alpha > 1)}]$ can be obtained from Equation (2-13). Thus, the delay for a small image satisfies the following:

$$E[T_{\text{image}}] \ge E[N_{last(\alpha > 1)}] \cdot (PTT + RTT) \qquad (\alpha > 1) \qquad (2\text{-}14)$$

The right hand side of Equation (2-14) is a lower bound for many of the same reasons that Equation (2-12) was a lower bound. However, Equation (2-14) differs from Equation (2-12) in two ways: the full-pipe term $(F - W) \cdot E[N] \cdot PTT$ has been eliminated; and the distribution of N_{last} differs from $N_{last(\alpha > 1)}$, thereby subtly affecting the second term $E[N_{last}] \cdot (PTT + RTT)$. Together, Equation (2-12) and Equation (2-14) constitute the "tight" bound on the image transfer latency.

2.5 A comparison of the tight and loose lower bounds on latency

How much more accurate is the "tight" bound of Section 2.4 than the "loose" bound of Equation (2-5) ? To answer this question, we first fix the parameters to their default values: I=20 kbits, wireless BW=500 kb/s, H=100 bits, and $BER = 10^{-2}$. For the loose bound, fixing these quantities means that the estimated delay is only a function of the image fragmentation factor F, which was graphed in Figure 2.4. In contrast, the delay of the tight bound is still a two-dimensional function of RTT and F. Our approach is to choose a few representative values of RTT, and then plot the delay given by the tight bound as a function of F for each fixed RTT. Such an approach permits the loose bound shown in Figure 2.4 to be directly compared with the tight bound on the same graph.

2.5.1 Average image transfer latency

Figure 2.6 shows three log-log curves that plot average image transfer latency as a function of fragmentation F. The latency curve L evaluates the loose bound described by Equation (2-5) at the default parameters, and is almost identical to Figure 2.4, except that only values $F \geq 50$ are displayed.[1] Two graphs of the tight bound, T_1 and T_2, are also shown corresponding to different values of RTT. T_1 is obtained by evaluating Equation (2-12) and Equation (2-14) at RTT=100 ms, while T_2 corresponds to RTT=10 ms. We also evaluated the tight bound at RTT=1 ms, though these results are not pictured because they follow the loose bound so closely as to be visually indistinguishable. We chose these values of RTT to roughly correspond to real-world network conditions. A value of 100 ms for RTT approximates the end-to-end roundtrip delay experienced by packets across a multi-

1. Small values of F<50 caused numerical convergence problems in the evaluation of the tight bound's $E[N_{last}]$ (see Equation (2-11)). For comparison purposes, both loose and tight bounds were evaluated over the same range of F>=50.

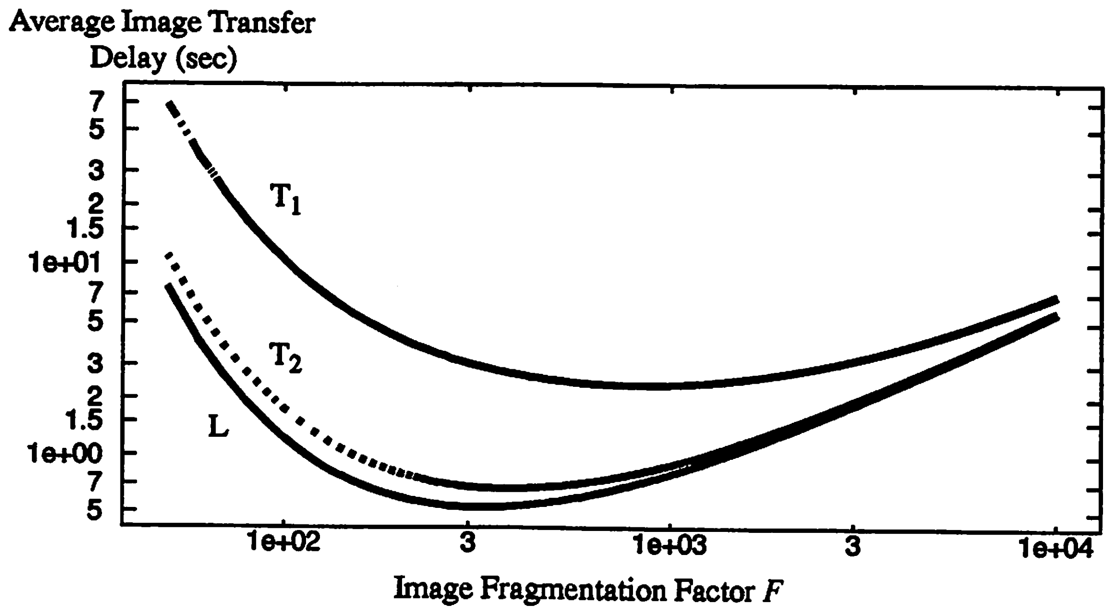


Figure 2.6 Log-log plot of the loose lower bound delay estimate (curve *L*) and two tighter bound delay estimates (curves T_1 (*RTT*=100ms) and T_2 (*RTT*=10ms)) as functions of the image fragmentation factor *F*. The loose curve *L* is obtained from Equation (2-5) and reproduces Figure 2.4, but over a slightly smaller range of *F*. The more accurate delay estimates shown in T_1 and T_2 are obtained from Equation (2-12) and Equation (2-14). The loose bound is shown to considerably underestimate the delay when the roundtrip time *RTT* becomes large relative to the size of the packetized image (including overhead).

hop Internet connection. A value of 1 ms for *RTT* approximates the roundtrip time experienced by packets at the data link layer, i.e. across a single wireless link.

Figure 2.6 shows that the difference between the loose lower bound L and the tight lower bound can become quite large for sufficiently large *RTT*. For example, a comparison of T_1 and L at the loose bound's minimum delay point ($F_{opt} \cong 300$) shows that there is a difference of approximately three seconds between the two estimates when *RTT*=100 ms. For such a large value of *RTT*, a considerable portion of the image will be delivered over a partially empty pipe. The loose bound will fail to count the latency introduced by this partially empty condition, and therefore will significantly underestimate the image transfer time compared to the tight bound.

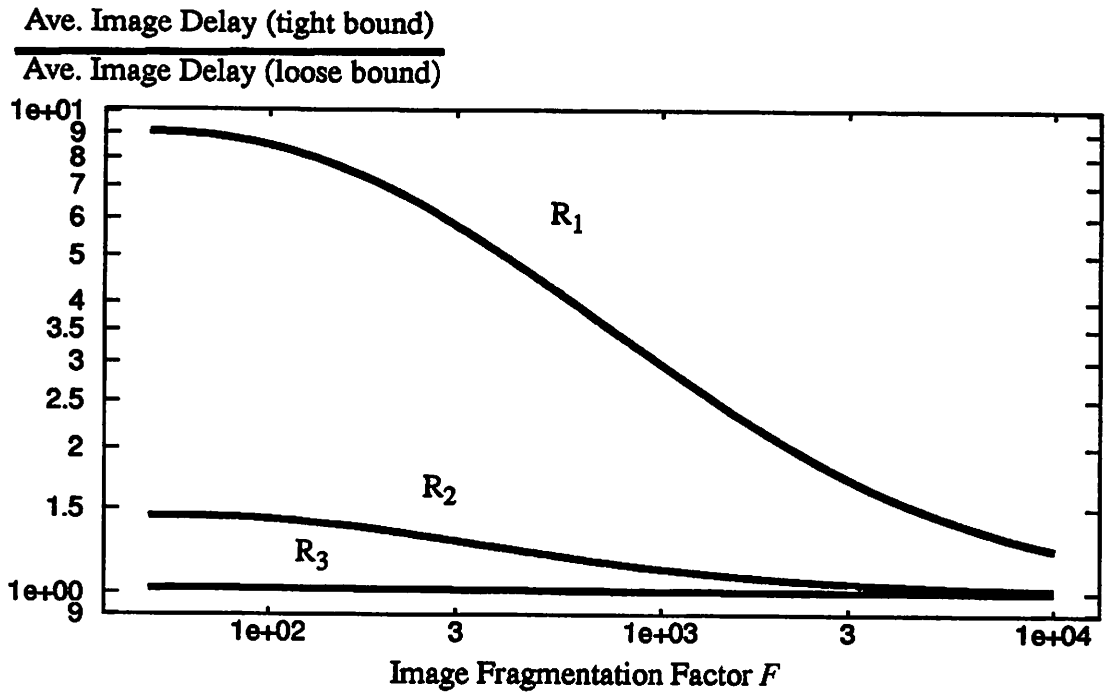


Figure 2.7 **A log-log plot of the ratio of the tight bound on image transfer delay to the loose bound on average image transfer delay as a function of image fragmentation *F*. The three curves R_1, R_2, and R_3 correspond to *RTT*=100 ms, 10 ms, and 1 ms respectively. R_1 shows that for large values of *RTT* (relative to the image size), the loose bound underestimates the delay by at least a factor of two for most reasonable values of $F<1000$. This figure is a normalized representation of Figure 2.6.**

2.5.2 Ratio of average image transfer latencies

An alternative and more revealing way of comparing the tight bound to the loose bound for various values of *RTT* is to normalize each of the curves with respect to the loose bound. We form the ratio of the tight bound's average image transfer delay to the loose bound's average image transfer delay at each point *F*, for fixed *RTT*. This ratio will measure the extent to which the tight estimate exceeds the loose estimate. In Figure 2.7, we plot this ratio as a function of *F*, and three values of *RTT*. Curves R_1, R_2, and R_3 correspond respectively to *RTT*=100 ms, 10 ms, and 1 ms. Figure 2.7 is directly related to Figure 2.6 since each point $R_i(F)$ is obtained by dividing $T_i(F)$ by $L(F)$.

Figure 2.7 summarizes the answer to our original question of how much more accurate the tight bound is than the loose bound. Curve R_1 shows that, for relatively large values of *RTT*, the loose bound underestimates the latency by at least a factor of two for almost all meaningful values of *F*. Consequently, end-to-end protocols with large *RTT* relative to the image size would find the tight bound a far more accurate prediction of the minimum cost in delay than the loose bound. In contrast, R_3 shows that, for relatively small values of *RTT*, the loose bound does as good a job as the tight bound in estimating the latency for all values of *F*, due to the negligible amount of idle time. Therefore, data-link protocols with a relatively small *RTT* would find the loose lower bound a useful tool for estimating their minimum image transfer latency, without having to resort to evaluating the tight bound.

2.5.3 Conditions under which the loose bound is inaccurate

The loose bound is considerably easier to evaluate than the tight bound, since $E[N_{last}]$ in Equation (2-11) is difficult to calculate. Therefore, we would like to use the loose bound as often as possible. However, we also need to know when the loose bound is inaccurate. Specifically, at what value of *RTT* causes the loose bound to start to perform "poorly"? As *RTT* increases, clearly there will come a point in which a significant portion of the image is delivered over a partially empty pipe, corresponding to a poor estimate by the loose bound. For example, for either of the cases *RTT*=100ms, and *RTT*=1ms, it is immediately clear from their normalized curves R_1 and R_3 respectively that either the loose bound is a good estimate (R_3), or it is a poor estimate (R_1), for all values of *F*. However, when *RTT*=10ms, there is some ambiguity. Curve R_2 shows that the loose bound is close to the tight bound for some values of *F* but not for others, i.e. it is not clear whether *RTT*=10 ms is "large enough" to start causing the loose bound to perform poorly.

To help us determine when *RTT* is large enough to affect the accuracy of the loose bound, we utilize the ratio α defined by Equation (2-7). Recall that α is the ratio of *TRTT* to image size (plus overhead). For values of $\alpha \geq 1$, then the roundtrip time is so large as to cause the entire image to be delivered over a partially empty pipe. Therefore, for $\alpha \geq 1$ the loose bound will significantly underestimate the delay. As α gradually decreases below one, the percentage of the delay caused by the non-saturated pipe gradually decreases, and the loose bound becomes more and more accurate. Let us define "poor" performance as any ratio of the tight bound to the loose bound which exceeds a factor of 1.5x to 2x. Therefore, our goal is to find the approximate value $\alpha_{\mathbf{indicator}}$ for which the tight bound is about 1.5x-2x of the loose bound.

The ratio α is a function of F and RTT. Table 3 tabulates α for the three values of RTT represented in Figure 2.7 over a sufficiently comprehensive range of F. For RTT=100ms,

Table 3. Values of the indicator α are shown as a function of image fragmentation factor F and roundtrip time RTT.

Image Fragmentation Factor F	RTT (ms)		
	100	10	1
50	$\alpha = 2.0$	.22	.04
300	1.0	.10	.013
500	.72	.073	.0091
3000	.16	.016	.0019
5000	.096	.0098	.0012

$\alpha \geq 0.2$ for values of $F < 3000$. Over this range of F, the tight bound exceeds the loose bound by at least a factor of two according to curve R_1. For RTT=10ms, $\alpha \geq 0.2$ for values around $F \leq 50$. Near this range of F, the tight bound exceeds the loose bound by a factor of about 1.5 according to curve R_2. Thus, in answer to the question concerning when the loose bound begins to perform poorly, a rough rule of thumb would be that if the total roundtrip time $TRTT$ exceeds about $\frac{1}{5}$ of the raw "image plus overhead" transmission time, then there is a high likelihood that the loose lower bound will considerably underestimate the minimum latency, possibly by a factor of 2x or higher.

2.6 Limitations of practical end-to-end ARQ protocols

In this section, we discuss several of the limitations to performance experienced by practical realizations of end-to-end ARQ protocols. First, real-world protocol implementations like TCP often cannot approach the throughput of ideal SRP. Second, in practice it may be difficult to determine the optimal packet fragmentation value. Third, protocol efficiency may be limited by small window sizes that are less than the roundtrip time, or by finite buffer sizes at the receiver.

Practical end-to-end fully reliable protocols will in general deliver data more slowly than ideal SRP. For example, the end-to-end reliable protocol used over the Internet, TCP, has been shown to have a throughput bounded by the performance of GBN (for the case of constant retransmission time-outs) [30][33]. Since GBN's throughput is bounded by SRP [78], and since throughput is inversely proportional to the expected number of transmissions $E[N]$ [78] and $E[T_{image}]$, then TCP will incur a higher latency cost than SRP. Moreover, all of the other real-world implementations of transport protocols besides TCP that are surveyed in [32] practice either some form of GBN, or some non-ideal form of SRP (e.g. finite receiver buffers, finite transmit buffers, and limited window sizes [78]). Therefore, all current real-world implementations of transport protocols ultimately incur a higher latency cost than the ideal SRP analyzed here.

Next, recall that the optimal fragmentation derived in Equation (2-6) is a function of both the BER and the image size I. Due to lack of feedback from the channel, the BER may not be known. Or due to a lack of timeliness in feeding back the BER, F_{opt} may be an inaccurate estimate that is not updated with sufficient rapidity. Furthermore, F_{opt} is also a function of the overall image size I. Unless the interface to the protocol specifically allows the application to define arbitrary boundaries on the stream of packets that is sent to the reliable protocol, then the underlying reliable stream protocol won't be able to understand the notion of an "image" nor of "image size". For example, TCP's reliable stream delivery service views the data that it is transporting as a stream of bytes, and has no sense of the embedded semantics of the data it is delivering. Hence, there is no means for TCP to extract image boundaries.

Another limitation occurs if the protocol's window size is less than the roundtrip time measured in packet lengths. Under these circumstances, the windowing mechanism is artificially choking the pipe, thereby forcing interstitial space to appear in every *RTT* interval. For example, previous implementations of TCP have had an upper bound on their window sizes that is too small to keep the pipe full over satellite links that have a high bandwidth-delay product [109]. Other concerns include finite receiver buffer space, which can cause buffer overflow and packet loss at the receiver, thereby causing retransmissions and reducing throughput. A throughput analysis of finite buffer SRP has been performed [79].

2.7 On proposals to improve the performance of wireless TCP

To complete our examination of reliable protocols, we consider some efforts to improve TCP for wireless links. The considerable latency costs of conventional ARQ protocols over connections with wireless access links have been documented in previous sections. TCP suffers from all of these problems and more. Another problem specific to TCP which lowers its throughput and increases its delivery latency is that, in a wireless environment, TCP mistakes packet losses caused by bit corruption for packet losses caused by network congestion. TCP responds to any packet loss by initiating congestion-avoidance mechanisms like window throttling and "slow start". However, if the packet losses were caused by bit corruption, then there is no congestion on the link, and TCP is unnecessarily lowering its throughput and increasing its image transfer latency in response to "phantom" congestion. Several approaches to fix this problem and improve TCP's wireless performance have been proposed, and are summarized in [5].

Each proposal to improve TCP has its own drawbacks. One approach, called Indirect TCP, violates end-to-end semantics, and requires intermediate proxies to be running throughout the wired/wireless network interface to translate between TCP and the local wireless transport-layer protocol. A second approach, called the "snoop" protocol, also requires TCP-aware snoop agents to operate throughout the network at the wired/wireless basestation interface, though in a less visible fashion than Indirect TCP. The snoop protocol also relies on the assumption that return acknowledgments will pass through the same snoop agents. The presence of multiple forward and return data paths underlying a single TCP connection may result in the snoop agent failing to filter acknowledgments properly, thereby failing to hide packet losses from the TCP sender, which will react as before in a congestion-avoidance manner.[1]

A third approach to improving TCP's wireless performance relies upon *data-link layer* protocols to improve the error performance of the wireless link, thereby helping out end-to-end protocols like TCP [4]. End-to-end protocols like TCP are responsible for reliably delivering a packet across multiple concatenated links. A data-link layer protocol only provides error protection (via forward error correction (FEC) and/or ARQ) over a single link. Conceptually, a link-layer protocol operates underneath the end-to-end protocol in the protocol stack, possibly for each link in the connection. We will have more to say on the use of FEC in the next chapter, and on link-layer issues in a later chapter.

For now, we consider several other issues concerning link-layer and end-to-end protocol interactions. One reference has pointed out that there can be inefficient coupling

1. This observation is gleaned in part from conversations with Venkat Padmanabhan at Berkeley.

between TCP and link-level retransmissions [30]. In addition, we ask: will wireless data-link layer protocols be able to handle the heterogeneity of source data in an efficient manner? For example, typically video and audio are delivered using the unreliable User Datagram Protocol (UDP) over the Internet because TCP is too slow for these stream-based media. Other forms of data (e.g. file transfers using the *ftp* application-level protocol, email using *smtp*, and Web-related transfers using *http*) are all delivered over the reliable TCP protocol. If wireless link-layer protocols apply uniform error protection to all of these different forms of data, then this can result in significant latency penalties and/or inefficiencies. Uniform FEC will overprotect some forms of data and/or simultaneously underprotect others. Link-layer ARQ reliability applied uniformly to all data can significantly slow down UDP-based video.

Ideally, wireless data-link layer protocols would like to be able to distinguish between different forms of data, and apply unequal error protection (UEP) to these data types to improve link-layer protocol efficiency. To what granularity should the data-link layer be informed about these different types of source data? One approach would be for the link-layer protocol to simply differentiate between UDP and TCP packets, and thereby provide a coarse two-tier approximation to UEP. This approach requires the link layer to understand IP header semantics at the minimum, which technically violates Internet layering principles. For IPv4, the data-link protocol must verify first that IP version 4 is being used, and then to inspect the *PROTOCOL* field in the IPv4 header to determine if UDP or TCP is embedded in the IP payload [25]. But for IPv6, determining which upper layer protocol is being used can be more complicated due to IPv6 header extensions. The IPv6 *NEXT_HEADER* field can be used to specify either various IPv6 header extensions, or upper layer protocols like TCP/UDP [109]. In the worst case, multiple header extensions create a linked list of *NEXT_HEADER* fields that must be searched to its tail before the correct upper layer protocol can be identified.

Beyond these difficulties, link-layer protocols that implement protocol-granularity UEP also fail to realize that not all UDP data should be treated the same. For example, a variety of video coding techniques transmitted over UDP are possible, including MPEG-2 video, H.261 video, as well as various hierarchically coded/layered multicast video. Each of these different video coders may generate multiple streams, with each stream requiring a different degree of FEC. Multiple coded forms of audio may be delivered over UDP, some of which may require little to no FEC, while others may require heavy FEC. Many other non-audio and non-video applications may also be transmitting on top of UDP. Protocol-granularity FEC would apply the same error protection to each of these very different forms of data, and suffer the same problems as uniform error protection.

Finer granularity application-specific UEP is somewhat problematic because no standard mechanism exists for communicating application-level quality-of-service (QOS) information down through the Internet protocol stack (i.e. through the transport layer and IP layer) to the data-link protocols. IPv6 "flow" labels [57] partially solve this problem, but several issues remain unclear. It is unclear what role the transport layer should play in mapping application-level QOS to IPv6 flows. Also, it is unclear to what extent IPv6 routers will be able to store per-flow QOS state information. In addition, the mechanisms for providing the data-link protocol with access to this flow-based IP state information do not currently exist. We discuss some of these issues in Chapter 7.

In summary, a transport layer protocol's reliance on link-layer protocols to solve most of its error problems over a wireless link can result in a variety of inefficiencies. The lack of a practical mechanism for passing application-level QOS information through the transport layer down to the data-link layer forces the link-level protocol to practice either uniform error protection, or protocol-granularity UEP, both of which can be very inefficient.

Finally, even if TCP is modified to incorporate elements of Indirect TCP and/or the snoop protocol, the resulting increase in speed for TCP over wireless channels will not ultimately be sufficient to support strictly real-time delivery for instantaneous interactivity. TCP's latency performance will still be lower bounded by that of ideal SRP. We showed earlier that ideal SRP incurs a latency of about half a second at 10^{-2} BER under the loose bound estimate. Therefore, if our delay budget is 100 ms, i.e. we need to get our image across the connection in less than 100 ms for interactivity purposes, then TCP as currently realized with sliding windows and cumulative acknowledgments will be unable to meet this latency objective.

2.8 Summary and conclusions

In this chapter, we have derived two lower bounds which quantify the minimum latency required by the most efficient ARQ protocol, ideal SRP, to reliably deliver a fragmented image over a noisy *BER* bandlimited link. Since our derivations characterize the channel generically through its parameters, then our analysis can be applied both to an individual wireless link (i.e. data-link layer protocols) as well as to end-to-end connections which include a wireless access link (i.e. transport protocols).

The loose lower bound is useful for the intuition it provides, and also for its ease of calculation. We have shown by using the loose lower bound that ideal SRP can take a minimum of half a second to deliver a 20 kbit image over a 500 kbit/s channel at 10^{-2} *BER*.

The tight lower bound is more accurate than the loose lower bound because the delay due to a partially empty transmission pipe is also counted. This also makes the tight lower bound more difficult to calculate. The tight lower bound should be used in place of the loose lower bound when the total roundtrip time is a significant fraction (exceeding about $\frac{1}{5}$) of the time it would take to transmit an image (plus protocol overhead) once at the channel bit rate. Under these conditions, much of the image is delivered over a partially empty transmission pipe, which the loose lower bound fails to measure. For end-to-end protocols with a *RTT* of approximately 100 ms or more, the tight lower bound is a better estimate by at least a factor of about two than the loose lower bound. For link-layer protocols with a *RTT* on the order of 1 ms, then the loose lower bound will be adequate.

Also, we have listed practical limitations which will increase the image transfer latency beyond the half second calculated by the loose lower bound. A real-world transport protocol which implements a scheme like GBN will be more inefficient than SRP. Moreover, non-ideal constraints such as finite receiver/transmit buffers and finite window sizes will increase the delay beyond the estimated bounds, even if SRP is the protocol of choice. Finally, our loose lower bound delay curve shows how imperfectly-sized fragmentation can lead to increased latency.

These limitations are all suffered by the current implementation of TCP. In addition, proposed fixes to TCP for wireless access have their own limitations, some of which we have outlined here.

In our analysis, we have not addressed the use of multi-copy stutter ARQ techniques which could be used to lower the latency incurred by reliable delivery of a fragmented image. In addition, our analysis assumes fixed-length payloads and headers, and does not account for variable-length packets. Finally, we have not considered in this chapter the impact on the image transfer latency of adding forward error correction (FEC) to protect against wireless bit errors. In the next chapter, we quantify the minimum FEC redundancy needed to adequately protect a fragmented image for a given *BER* for both open-loop systems and closed-loop ARQ systems.

3

Minimum Redundancy Evaluation of Linear Block Codes

Our primary interest is the design of digital communication systems to deliver interactive visual multimedia rapidly and with sufficient reliability across a noisy wireless channel. Our primary motivation for studying error protection techniques is to quantify the increase in delay associated with adding error protection. Extra reliability often requires the transmission of redundancy information in addition to the original data. This added redundancy increases the transmission delay.

In this chapter, we first quantify the minimum amount of redundancy required by open loop error protection techniques to adequately protect a fragmented image from the transmission errors introduced by a noisy channel. Open loop error protection, also called Forward Error Correction (FEC), lacks the feedback channel used by ARQ protocols to guarantee end-to-end reliability. The scope of our work is confined to FEC via linear block codes.

We derive an inequality which shows the minimum amount of error correction redundancy that is required by linear block codes in order to keep the loss rates of transmitted images acceptably low. We develop an algorithm based on this inequality that systematically determines the minimum code length for a wide range of input block sizes. Variations of this algorithm are applied to each of the following classes of codes: optimal (maximum code distance) binary linear block codes; binary cyclic codes; binary BCH codes; and Reed-Solomon (RS) codes. For binary BCH linear block codes, we deduce that the amount of FEC redundancy needed to adequately protect an image can triple the bandwidth for moderately long block codes. For RS codes, our analysis shows that the FEC overhead can double the bandwidth under the same channel conditions.

We complete our FEC analysis with an investigation of Type-I Hybrid ARQ protocols. Such hybrid protocols combine FEC with retransmissions. Assuming RS codes are employed by Type-I Hybrid ARQ protocols, then we show that no RS codes of length less than 1023 output symbols can be found which will adequately protect a fragmented image and deliver that image within the interactive latency bound of 100 ms at a severe 3% *BER*.

Consider the open-loop FEC system shown in Figure 3.1 that provides end-to-end error protection across a multi-hop connection which includes a noisy wireless link. The FEC block coder adds redundancy to each input block of data bits. These extra error protection bits are used at the FEC decoder to correct any bit errors that may be encountered during transmission across multiple concatenated links. Since FEC is an open-loop error protection technique, then no guarantee can made that a packet will arrive with full reliability; the lack of a feedback channel means that the sender cannot know whether the FEC has been sufficient to error-correct the data into a reliable packet. Following our

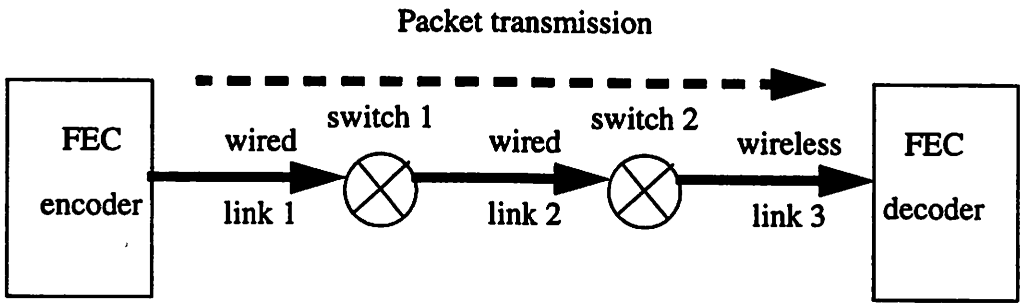


Figure 3.1 End-to-end open-loop error protection via FEC over multiple concatenated network links (some wired, some wireless). There is no guarantee of completely reliable delivery.

approach to analyzing ARQ protocols, we focus on the impairments introduced by the weakest link, e.g. the wireless link, and idealize the noise and packet losses introduced by the other links of the connection. As noted in the previous chapter, one consequence of this assumption is that both end-to-end and link-layer interpretations of our FEC analysis are possible.

3.1 A review of decoding techniques for linear block codes

In this section, we review two techniques for decoding linear block codes: maximum likelihood decoding; and bounded-distance decoding. Practical decoding of moderately long to long block codes is almost always a form of bounded-distance decoding. The probabilistic performance of bounded-distance coding has been characterized, and we shall apply those results to determine the minimum redundancy necessary to achieve a given loss rate for a packetized image across a noisy channel.

The encoding and decoding processes for block codes have easy geometric interpretations. For every K input symbols, the (N,K) block coder generates a block of $N>K$ output symbols. We assume binary input values for now, i.e. that the symbols are bits. There are 2^K input K-bit messages, so that a one-to-one mapping will produce 2^K valid output N-bit codewords or vectors. These codewords are scattered throughout the N-dimensional linear space, which contains 2^N total vectors. In general, this K-bit -> N-bit mapping can be nonlinear. That is, the 2^K codewords in general do not have to form a linear subspace of the N-dimensional linear space. However, almost all practical block codes are linear, i.e. they do form linear subspaces (over finite fields) within the N-dimensional space [152]. The property of linearity simplifies the task of encoding vectors, which becomes a straightforward matrix multiplication of the input K-bit code vector with a K x N generator matrix that completely describes all characteristics of the linear block code [79].

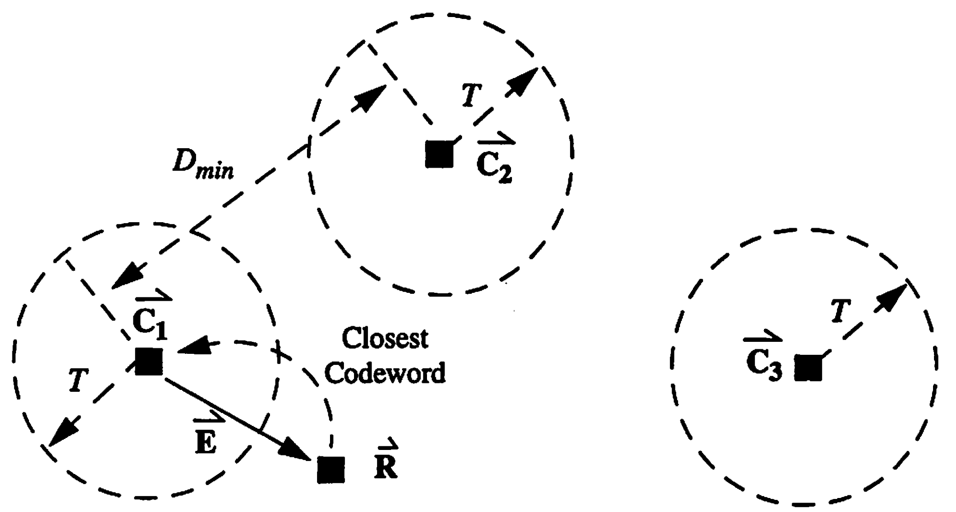


Figure 3.2 ***Maximum likelihood decoding*** **of linear block codes is equivalent to minimum distance decoding over a binary symmetric channel. The received vector $\vec{R}$ is mapped to the nearest codeword. An alternative form of decoding, called *bounded-distance decoding*, only guarantees minimum distance decoding of $\vec{R}$ when the number of errors satisfies $\|\vec{E}\| \le T$, i.e. only if $\vec{R}$ lies inside a T radius sphere. If the received vector $\vec{R}$ lies outside any of the codeword spheres, then bounded-distance decoding will make no further effort at correction.**

3.1.1 Maximum-likelihood decoding

The decoding process maps the received N-bit vector $\vec{R}$ to the proper codeword. However, a noisy channel will cause bit errors, so that the received vector is the sum of the original codeword $\vec{C}$ and the error pattern $\vec{E}$ (with 1's in the position of the bit errors), i.e. $\vec{R} = \vec{C} + \vec{E}$. The decoder's task is to find a codeword $\vec{C}$ from $\vec{R}$ that maximizes the conditional probability of correct decoding $f(\vec{R}|\vec{C_i})$ over all codewords $\vec{C_i}$, also called the maximum-likelihood (ML) decoding procedure. Assuming a binary symmetric channel (BSC) that corrupts/flips bits independently of one another, then it can be shown that ML decoding is equivalent to *minimum distance or nearest-neighbor decoding* [153]. ML decoding maps the received vector $\vec{R}$ to the "nearest" codeword. The distance between any two code vectors is measured as the number of vector indices in which the two vectors have different bit values, also called the *Hamming distance*. The minimum Hamming distance between any two code vectors is called the *minimum code distance D_{min}*. For linear block codes, D_{min} is also equivalent to the Hamming distance from the zero vector to the codeword closest to the origin. The *Hamming weight* of a vector $\vec{v}$ is defined as the number of 1's in that vector, i.e. the distance between the vector and the origin, namely the norm of the vector $\|\vec{v}\|$. Therefore, D_{min} can also be interpreted as the minimum Hamming weight codeword of the (N,K) linear block code.

In Figure 3.2, we illustrate several of these concepts. Minimum-distance decoding would map $\vec{R}$ to the nearest neighboring code vector $\vec{C}_1$. Also, if codewords $\vec{C}_1$ and $\vec{C}_2$ happen to be the two closest code vectors among all possible pairs, then the distance between them is labelled as the minimum code distance D_{min}.

The minimum code distance is a key property of block codes which affects their error correction and detection performance. We focus first on the error correction problem. Geometrically, it is clear that if the number of errors (i.e. Hamming weight of the error pattern) is less than about $\frac{D_{min}}{2}$, then for any transmitted codeword, the received vector $\vec{R}$ will be unambiguously closer to the original codeword than any other codeword. Conceptually, we can form a sphere around each codeword that has radius about $\frac{D_{min}}{2}$. If the error patterns are of Hamming weight less than about $\frac{D_{min}}{2}$, then the received vector $\vec{R}$ will not have changed enough to escape the sphere. Under these conditions, the received vector is guaranteed to be unmistakably corrected to the original codeword.

More precisely, define the error correction power of an (N,K) block code $T = \left\lfloor \frac{D_{min}-1}{2} \right\rfloor$, where the "floor" function $\lfloor a \rfloor$ calculates the greatest integer less than or equal to a. Every error pattern with Hamming weight $\|\vec{E}\| \le T$ can be unambiguously corrected to the original codeword by the geometric reasoning presented above [153]. The radius-T error correcting spheres centered around each codeword are shown in Figure 3.2.

If the error pattern has Hamming weight $\|\vec{E}\| > T$, then $\vec{R}$ will either lie in another T-sphere, or in the interstitial space outside of all T-spheres. If the received vector lies in another T-sphere, then minimum distance decoding will map $\vec{R}$ to the incorrect codeword associated with this other T-sphere. Given an independent error channel in which the probability of bit error is less than one-half, then the distribution of the number of errors in a block will be biased towards fewer rather than more errors. Consequently, choosing the closest codeword will help more often than it will hurt. So despite the possibility of incorrect decoding, the probability of incorrect decoding is relatively low, and minimum distance decoding still maximizes the overall likelihood of correct decoding because the probability of few errors is high and the probability of many errors is low.

If $\vec{R}$ lies outside of all T-spheres, then either the received vector is unambiguously closest to a single codeword, or it is equidistant from two or more codewords. In the former case, nearest neighbor decoding will produce a unique result. In the latter case where there is ambiguity, ML decoding must choose between two or more equally likely codewords.

Error detection also lends itself readily to a geometric interpretation. An error is detected whenever a received vector is not a codeword. An error goes undetected whenever the error pattern transforms the original codeword into another incorrect codeword. For example, in Figure 3.2 the vector $\vec{R}$ would be detected as in error. However, an error pattern could be added to $\vec{C_1}$ causing $\vec{R}$ to be identical to $\vec{C_2}$, leading to an undetected error. Figure 3.2 shows that all error patterns with Hamming weight $\|\vec{E}\| \le D_{min} - 1$ can be detected. If the number of errors is strictly less than D_{min}, then it is impossible to transform the two closest codewords into one another by adding the appropriate error pattern. By extension, no other pair of codewords which are separated even further can be transformed into one another under these conditions. Thus, if the number of errors $\|\vec{E}\| \le D_{min} - 1$, every received vector with non-zero errors can be detected.

For linear block codes, ML decoding can be thought of as a table lookup operation: for each received vector $\vec{R}$, consult the table to find the pre-computed nearest codeword $\vec{C_i}$. Rather than a two-column table lookup approach, the linearity of the block code permits an array-based ML decoding procedure called *standard array decoding* [141]. We mention this technique only to note that both table lookup and standard array decoding of (N,K) linear block codes require the storage of 2^N vectors. Therefore, memory requirements can be prohibitively large for block lengths N>20.

A variation of standard array decoding, called *syndrome decoding* [79], further exploits the linearity of the code to achieve a significantly smaller storage requirement. Here, a syndrome $\vec{S}$ is calculated from the received vector $\vec{R} = \vec{C} + \vec{E}$. The syndrome has the property that it is only a function of the error pattern $\vec{E}$. Also, multiple error patterns can map to the same syndrome, creating a group of error patterns per syndrome called a coset. Each coset has a minimum Hamming weight error pattern called the coset leader $\overrightarrow{E(\overline{S(\vec{R})})}$. Hence, syndrome decoding can also be thought of as a table lookup operation: calculate the syndrome; then consult the table to find the coset leader associated with this syndrome. Provided that the decoder tries to decode an output codeword for every possible syndrome (i.e. every possible coset leader is used), then syndrome decoding is equivalent to ML decoding.

For an (N,K) block code, it can be shown that there are only 2^{N-K} distinct syndromes, a significant memory savings over standard array decoding. However, even syndrome decoding can result in large storage requirements. For example, for a ($N = 90$, $K = 60$) linear block code, up to $2^{N-K} \cong 1$ Gigaword of memory needs to be stored in the syndrome lookup table, making syndrome decoding impractical for codes with large redundancy [24][153]. In practice, ML decoding is attempted only for a few restricted cases: either for codes with a small amount of redundancy; or for certain special classes of linear block codes that exhibit structural properties that make them amenable to certain ML decoding algorithms. For example, table-lookup decoding of single-error-correcting Hamming codes [79] is an ML decoding technique. Majority logic decoding of Reed-Muller codes [152] and Meggitt decoding of the (23,12) Golay code [24] are both examples of fast, practical ML decoders. However, such codes typically have small error correcting power, and/or poor performance (large redundancy for the number of errors corrected).

3.1.2 Incomplete bounded-distance decoding

Most practical methods for decoding long block codes with large error correcting ability sacrifice ML performance for lower complexity decoding, opting instead for what is called (non-ML) *incomplete bounded-distance decoding* [24][87][152][153]. Bounded-distance decoders only promise to correct all errors up to *L*, where $L \leq T$, and *T* is the maximum permissible error correction limit $T = \left\lfloor \frac{D_{min} - 1}{2} \right\rfloor$. Error sequences with more than *L* errors ($\vec{R}$ lies outside of any *L*-sphere) are not corrected, returning only information that an error has been detected. This corresponds to applying syndrome decoding to only some of the coset leaders, i.e. only coset leaders whose Hamming weight is less than or equal to *L* are used for error correction. In Figure 3.2, true bounded-distance decoding would correspond conceptually to correcting only those received vectors who fall within an *L*-radius sphere (not shown) enclosed by the larger *T*-radius sphere (shown). Thus, bounded-distance decoders do not always correct up to the maximum permissible radius *T* determined by the code's natural distance D_{min}. For the rest of this section, we will assume that bounded-distance decoders have been designed to correct up to the maximum permissible number of errors *T*.

The popular and highly efficient Berlekamp algorithm for decoding binary BCH cyclic codes, and the closely related Berlekamp-Massey algorithm for decoding *Q*-ary Reed-Solomon cyclic codes, are bounded-distance decoders [9][152]. Peterson's solution for cyclic codes is also bounded-distance [152].

First, we characterize the performance of bounded-distance decoding of (*N*,*K*) linear block codes. Given an *N*-bit received vector $\vec{R}$, then the probability of correctly decoding $\vec{R}$ is the probability that $\vec{R}$ lies within a *T*-sphere, i.e. that the error pattern has less than or equal to T errors. The number of error patterns with i errors in N bits is $\binom{N}{i}$. Therefore, if we define the *BER* as the probability of bit error, and assume a BSC memoryless error model, then the probability, P_{CD}, of correctly decoding an N-bit vector is given by [87]

$$P_{CD} = Prob\begin{bmatrix} Correct \\ Decoding \end{bmatrix} = \sum_{i=0}^{T} \binom{N}{i} \cdot BER^i \cdot (1 - BER)^{N-i} \tag{3-1}$$

Next we consider how ML decoding would modify Equation (3-1) . Since ML decoding can correct some error patterns with Hamming weight > T, then the summation for P_{CD} would include some additional terms. Hence, P_{CD} for ML decoding would be larger than the quantity shown in Equation (3-1) . As stated earlier, each syndrome and its corresponding coset leader is needed for ML decoding. Correct decoding occurs if and only if the error pattern matches the coset leader. Hence, if we can find the distribution A_i of the Hamming weights of the coset error pattern leaders, i.e. A_i is the number of coset leaders with weight i, then $P_{CD} = \sum_{i=0}^{N} A_i \cdot BER^i \cdot (1 - BER)^{N-i}$ [79].

In the rest of our derivation, we assume non-ML bounded-distance decoding of the general class of linear codes. Even though Berlekamp's iterative bounded-distance decoder only applies to the subclass of *cyclic* linear codes, we extend the bounded-distance analysis of Equation (3-1) to the general class of linear codes because Berlekamp's method represents the primary means of computationally efficient decoding of the large powerful efficient linear codes of interest to us. In some special cases, ML decoding of weak, small, and/or specially structured codes may still be possible and will outperform bounded-distance decoding. Bounded-distance analysis also has the advantage that knowledge of the internal structure and weight distribution of the code's coset leaders is not necessary, as evidenced by Equation (3-1) .

3.2 Bounded-distance decoding of a fragmented image encoded by optimal binary linear block codes

Given bounded-distance decoding, we would like to relate the probability of correctly decoding a single N-bit packet to the probability of correctly sending and decoding a much larger image. Following the same reasoning that we applied for ARQ protocols in the previous chapter, we fragment an I-bit image into $\lceil I/K \rceil$ fragments, each K bits long. Each K-bit fragment is encoded using an (N,K) linear block code.

The probability of correctly sending the entire image over an unreliable link is the probability that each of the individual K-bit fragments was decoded correctly after error correction. There are approximately I/K fragments, and assuming a BSC error model, then $Prob\begin{bmatrix}Correct\\Image\end{bmatrix} = [P_{CD}]^{I/K}$, where P_{CD} is the probability of correctly decoding an N-bit fragment defined in Equation (3-1) . Suppose we desire that the probability of correctly transmitting an image equal or exceed some subjectively tolerable threshold P_{CI}, so that $[P_{CD}]^{I/K} \geq P_{CI}$. Combining this equation with Equation (3-1) , we obtain the inequality

$$\sum_{i=0}^{T} \binom{N}{i} \cdot BER^{i} \cdot (1 - BER)^{N-i} \geq [P_{CI}]^{K/I} \qquad (3\text{-}2)$$

Equation (3-2) is the key expression which relates the error correction power T of the (N,K) linear block code to the other parameters of interest, namely the image size I, probability of bit error BER, and desired probability of correct image transmission P_{CI}. For our purposes, N, K, and T are the only parameters necessary to characterize the error performance of a linear block code. Under our previous "weakest" link assumptions, Equation (3-2) only captures the effect of corruption along a single noisy link, and does not attempt to capture the effect of packet losses caused by congestion along other links in the connection.

Equation (3-2) is used to construct a "minimum-redundancy" function of K. For each K, we want to find the (N,K,T) binary linear code with the smallest amount of redundancy $N-K$ that still manages to satisfy Equation (3-2), i.e. its error correction power T is sufficiently large to achieve the desired rate of successful image transmission. For a fixed K, a systematic procedure for determining the minimum-redundancy code would be to start from $N = K+1$ and increment N, testing at each $N > K$ for a code which satisfies Equation (3-2). The first suitable code found will be the minimum redundancy code for this value of K.

As N is incremented, at each value of $N>K$ it is sufficient to evaluate Equation (3-2) at a single value of T, namely at the strongest possible correction power $T_{opt}(N,K)$ for the given (N,K) pair. If the test passes, then we will have found at least one code at this (N,K) pair which is sufficiently powerful to satisfy the inequality and can stop further testing. This $(N,K,T_{opt}(N,K))$ code will have the minimum redundancy. If the test fails at $(N,K,T_{opt}(N,K))$, then no other codes with the same values of N and K and weaker error correction ability $T < T_{opt}(N,K)$ will be able to satisfy Equation (3-2), and we can continue to increment N.

The key to finding the strongest possible error correction power $T_{opt}(N,K)$ for a given (N,K) pair is to observe that the error correction ability T has an upper limit that is a function of N and K. Recall that T is a function of D_{min}, $T = \left\lfloor \frac{D_{min}-1}{2} \right\rfloor$. The minimum spacing D_{min} between any two codewords is upper bounded by N and K. For example, a simple upper bound would be N, because the distance between any two codewords cannot exceed the number of bits N in the vector. Thus, the minimum code distance D_{min} has an upper bound as a function of N and K, $D_{min} \le f(N,K)$, and T also has an upper limit. Provided that tighter more realistic upper bounds can be found for D_{min} and T, then T_{opt} can be evaluated directly at these upper bounds.

A variety of bounds on D_{min} have been derived for linear block codes, and usually are expressed in the form of $D_{min} \le f(N, K)$. No linear block code can have a D_{min} which exceeds these bounds. In later sections, we'll investigate how much the additional imposition of cyclic and BCH structure on linear codes reduces D_{min} below these upper bound values. Below, we list some of the more important upper bounds on D_{min} [9][87][153]. For now, we assume that the number of levels per symbol $Q = 2$, i.e. that we are dealing with binary linear block codes. Later, in our analysis of Reed-Solomon codes, we consider codes for which $Q>2$.

- *Hamming Sphere-Packing Bound*

$$\sum_{i=0}^{T} \binom{N}{i} \cdot (Q-1)^i \le Q^{N-K}$$

- *Plotkin Bound*

$$D_{min} \le N \cdot \frac{Q^{K-1} \cdot (Q-1)}{Q^K - 1}$$

- *Singleton Bound*

$$D_{min} \le N - K + 1$$

- *Griesmer Bound*

$$\sum_{i=0}^{K-1} \left\lceil \frac{D_{min}}{Q^i} \right\rceil \le N$$

References [9] and [153] plot these bounds on the same graph and show the different regions where they produce the smallest D_{min}. Some bounds are tighter for high rate codes ($\frac{K}{N}$ close to one), while other bounds are tighter for low-rate codes ($\frac{K}{N}$ close to zero). Since the D_{min} of every binary linear code obeys each of these four upper bounds, then the maximum permissible D_{min} will also obey the tightest (lowest) of these upper bounds. We define the tightest of these bounds as $\boldsymbol{D_{min}(N,K)}$ = min (D_{min}'s obtained from the Singleton, Plotkin, Hamming, and Griesmer bounds).

$T_{opt}(N,K)$ is obtained from the tightest overall bound $D_{min}(N,K)$. Since there may be other even tighter bounds than the four listed above, then it is possible that $D_{min}(N,K)$ represents a fairly loose bound, and in actuality no (N,K) binary linear codes exist with a minimum code distance of $D_{min}(N,K)$. By evaluating $T_{opt}(N,K)$ at $D_{min}(N,K)$, we are optimistically choosing the theoretically best set of parameters $(N,K,T_{opt}(N,K))$, even though a binary linear block code may not exist at these upper bounds.

Now, we can summarize the complete algorithm for determining the minimum-redundancy function of K over all binary linear block codes. Let $N_{min}(K)$ represent the minimum number of output bits for which Equation (3-2) is satisfied for a given K. The minimum amount of redundancy will be $N_{min}(K) - K$. Assume that I, BER, and P_{CI} are fixed. Algorithm I finds $N_{min}(K)$ over all binary linear codes based on the previous list of upper bounds.

Algorithm I:

1. Select a range $[K_{min}, K_{max}]$ for K. Initialize $K = K_{min}$-1.
2. Iterate $K = K$+1. Initialize $N = K$.
3. Iterate $N = N$+1.
4. Find the optimal $D_{min}(N,K)$ = min (D_{min}'s obtained from the Singleton, Plotkin, Hamming, and Griesmer bounds). Let
$$T_{opt}(N, K) = \left\lfloor \frac{D_{min}(N, K) - 1}{2} \right\rfloor .$$
5. Is Equation (3-2) satisfied at (N,K,T_{opt})? If no, go back to step 3.
6. If yes, then we have obtained the lowest $N_{min}(K)$ that creates a large enough D_{min} to satisfy Equation (3-2). Go back to step 2 until K exceeds K_{max}.

The inner loop finds the minimum code length for a fixed K, while the outer loop iterates over a range of K. Given $N_{min}(K)$, then we can plot the redundancy ratio $\frac{N_{min}(K)}{K}$ as a function of K. The redundancy ratio measures the minimum bandwidth expansion factor required by the best binary linear block codes to adequately protect a fragmented image over a noisy link.

The results from Algorithm I for $K<100$ are shown in Figure 3.3, in which three separate curves are pictured. All three plots assumed an image size I equal to 20 kbits. The "Baseline" function assumed a BER of 10^{-2} and a desired probability of correct image transmission P_{CI} of 0.999 (i.e. the user desires that at most 1 out of every 1000 images is undecodable due to bit errors). The "Worse BER" curve assumed a BER of $3\text{x}10^{-2}$ and a P_{CI} of 0.999, while the "Relaxed P_{CI}" curve assumed a BER of 10^{-2} and a P_{CI} of 0.99 (i.e. the user tolerates loss of at most 1 out of every 100 images due to corruption). We evaluated the left hand side of Equation (3-2) both by direct Binomial sums and by Poisson approximations to Binomial sums in order to verify the numerical precision of the our results.

Considering the baseline function alone, we find that an input packet of size 10 bits would require at least about three times expansion factor in bandwidth, equivalent to a minimum code length of $N = 30$ output bits. An input packet of size $K = 100$ bits would require at least about 1.5 times more bandwidth, equivalent to a minimum code length of $N = 150$, and a minimum redundancy of 50 redundancy bits.

Generally, for each curve, as the size of the K-bit input word increases, the redundancy ratio $\frac{N_{min}(K)}{K}$ required to satisfy Equation (3-2) decreases. This trend states that the relative cost in overhead required to successfully error-protect a fragmented image is significantly smaller for longer linear block codes than for shorter linear block codes. Since all the codes in Figure 3.3 achieve the same error correction performance (i.e. they all just

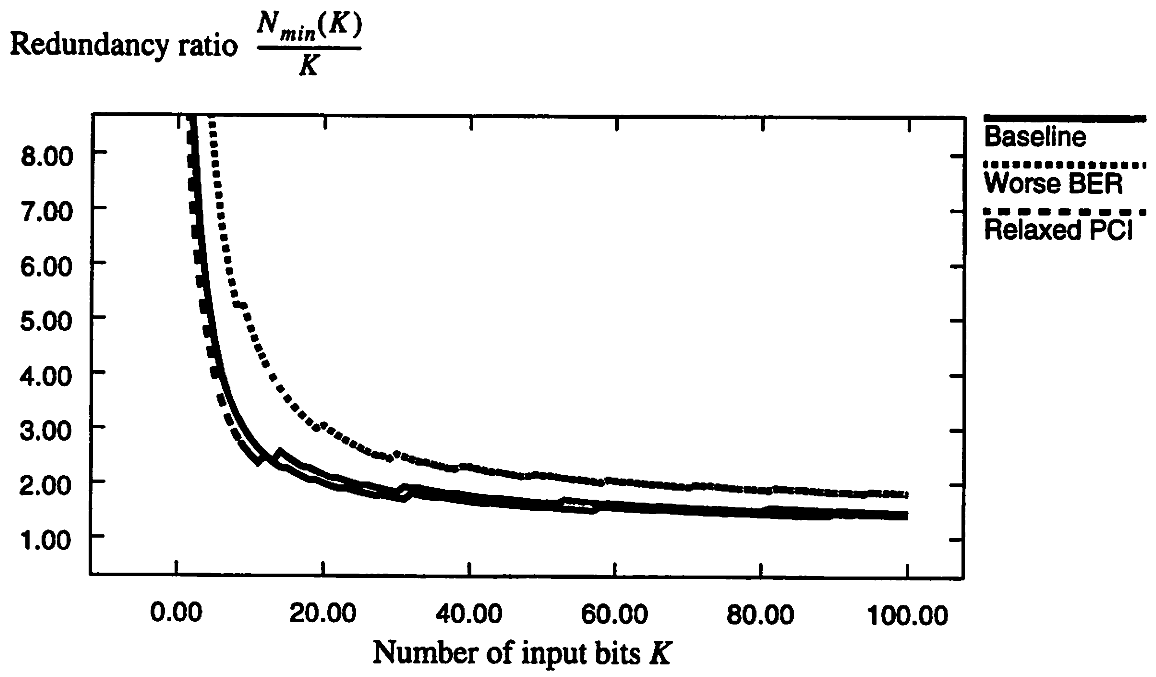


Figure 3.3 **The redundancy ratio $\frac{N_{min}(K)}{K}$ is plotted vs. the number of input bits K for different *BER*'s and different desired probabilities of correct image transmission P_{CI}. The redundancy ratio measures the minimum expansion factor required by the strongest binary linear codes to adequately error-protect a fragmented image transmitted across a noisy channel. $N_{min}(K)$ is the minimum number of output bits obtained by running Algorithm I. The Baseline, "Worse *BER*", and "Relaxed P_{CI}" curves are the middle, top and bottom functions respectively.**

barely satisfy Equation (3-2)), then the designer should choose the longer (N,K) linear codes in order to achieve reasonably low levels of overhead.

Since we are searching over all binary linear codes, then any other sub-class of binary linear codes, such as the binary cyclic linear codes and binary BCH cyclic linear codes, will require an even higher bandwidth expansion factor. Therefore, each of the curves shown in Figure 3.3 represents a lower bound (for its set of parameters) on how much redundancy is required to successfully error-protect and transmit a fragmented image across a noisy wireless link.

The "Worse BER" curve tests the sensitivity of Equation (3-2) and Algorithm I to variations in *BER*. The probability of bit error is increased to 3×10^{-2} from the baseline case of 10^{-2}, while P_{CI} remains fixed. The resulting dotted (top) curve shows a significant increase in the redundancy requirement. An input packet of size $K = 10$ bits would require at least about 4.5x redundancy, or a block code of length N greater than about 45 bits. An input packet of length $K = 100$ bits would require at least about 2x redundancy, or N greater than about 200 bits. Thus, for $K = 100$, the increase in *BER* leads to a hefty 50% increase in redundancy, requiring a (200,100) code instead of a (150,100) code.

The "Relaxed P_{CI}" curve tests the sensitivity of Equation (3-2) and Algorithm I to variations in the desired probability of correct image transmission P_{CI}. The *BER* remains the same as the baseline case, while the P_{CI} threshold is relaxed from 0.999 to 0.99, i.e. the permissible image loss rate is $\frac{1}{100}$ instead of $\frac{1}{1000}$. The result is plotted as the third dashed curve lying just below the baseline function. Figure 3.3 indicates that this extra tolerance for image errors does not substantially decrease the required power of the error correction code. In fact, a similar curve (not shown) evaluated for $P_{CI} = 0.9999$ also showed very little variation from the baseline function. At $K = 100$ bits, large variations in the probability of image loss ($1\text{-}P_{CI}$ varies from 0.01 to 0.0001) result in only small changes in code size ($N = 142$ and 151 at the respective bounds). Conversely, a very small increase in the redundancy ratio, at a fixed *BER*, can dramatically lower the probability of a lost image.

3.3 Bounded-distance decoding of a fragmented image encoded by binary BCH codes

In this section, our goal is to quantify how much more redundancy is required by binary BCH linear codes to adequately protect a fragmented image than the general binary linear codes analyzed in the previous section. BCH codes are of great practical importance because a fast bounded-distance decoding algorithm has been developed by Berlekamp for these codes, as noted in Section 3.1. Binary BCH codes form a subset of the class of binary cyclic codes, which are themselves a subset of the general class of binary linear codes. Therefore, the largest code distance that binary BCH codes can attain is limited by the upper bounds on D_{min} for general binary linear codes for a given N and K.

Practical values of D_{min} for binary BCH codes over a wide range of N and K show that indeed some error correction power is lost by imposing cyclic structure and then BCH structure on general binary linear codes. BCH codes are designed using a lower bound on the minimum code distance called the *BCH design distance* δ [153]. Up to T errors are corrected, where $T = \left\lfloor \frac{\delta - 1}{2} \right\rfloor$ and the code's true minimum distance satisfies $D_{min} \geq \delta$. Even though the true minimum distance D_{min} resulting from the BCH coding structure may be greater than δ, the bounded-distance decoder is unable to exploit this extra distance, and only corrects up to T errors. The effective code distance seen at the output of the receiver's decoder is therefore only δ. Hence, fast decoding of binary BCH codes can waste some error correction power, resulting in lower performance and/or higher overhead than general binary codes. The constraints on D_{min} and T suggest that a minimum-redundancy function of K for binary BCH codes will lie above the curves shown in Figure 3.3.

The cumulative efforts of several authors have resulted in an exhaustive tabulation of the minimum distance for all binary cyclic and binary BCH codes with odd N<128[1] as a function of N and K [17][98][99]. These tables show that binary BCH codes do not exist for some (N,K) combinations. These tables also show that there can exist more than one binary BCH code for a given (N,K) pair. The binary BCH code with the largest D_{min} for a fixed N and K (if there is more than one code) should be the candidate tested at each step

1. Only odd-N codes are considered because the construction properties of primitive and non-primitive BCH codes result in odd-N overall block lengths [87].

in the minimum-redundancy search algorithm, for the same reason that the binary linear code with the largest permissible $D_{min}(N,K)$ (taken from the upper bound) was tested at each step in Algorithm I.

The procedure employed to obtain the minimum code length function $N_{min}(K)$ for binary BCH codes very much resembles Algorithm I from the previous section. The only significant change is to step 4. Rather than computing the largest $D_{min}(N,K)$ for a fixed N and K from the upper bounds, we search the tables to find the largest D_{min} over all binary BCH codes with the same values of N and K (e.g. there are thirty-four (85,45) binary BCH codes but only two (111,75) binary BCH codes)[1]. Table lookup is a somewhat laborious procedure and is one of the reasons that we confined our search range of K to $35<K<46$. The other reason is that the tables only list block lengths up to 127. As we will show, for some of the K in this range, we were unable to find any binary BCH codes with $N<128$ of sufficient power to satisfy Equation (3-2) . If we were to test values of $K>45$, the search algorithm would with increasing likelihood stop at $N<128$ without being able to find a sufficiently powerful binary BCH code. Though such a determination is useful, it is less informative than obtaining a precise value for $N_{min}(K)$.

Our minimum-redundancy evaluation algorithm for binary BCH codes is summarized below. Again assume that I, BER, and P_{CI} are fixed.

Algorithm II:

1. Choose a range $[K_{min}, K_{max}]$ for K. Initialize $K = K_{min}-1$.
2. Iterate $K = K+1$. Initialize $N = K$.
3. Iterate $N = N + 1$.
4. Consult the reference tables to see if an (N,K) BCH code exists. If not, return to step 3. If so, then let D_{min} = the minimum distance of the

1. For completeness, $(N,K-1)$ expurgated codes (generated by the polynomial $(1+X)g(X)$ and listed as columns D2 and DB2 in [98][99]) derived from (N,K) codes are included into our calculations.

code with the largest *BCH design distance* among all (N,K) binary BCH codes. Let $T = \left\lfloor \frac{D_{min} - 1}{2} \right\rfloor$.

5. Is Equation (3-2) satisfied at (N,K,T)? If not, go back to step 3.
6. If yes, then we have obtained the lowest $N_{min}(K)$ such that there exists a binary BCH code with a large enough D_{min} to satisfy Equation (3-2) . Go back to step 2 until K exceeds K_{max}.

The inner loop finds the minimum code length for a fixed K, while the outer loop iterates over a range of K, just as for Algorithm I. The above procedure can be applied to binary cyclic codes as well, with the small modification in step 4 that D_{min} is set to the minimum distance of the code with the largest minimum distance among all binary cyclic codes with the same N and K.

Because the reference tables are limited to block lengths of $N<128$, then for our particular situation step 3 should be modified to iterate only up to $N = 127$. If $N \geq 128$, then the algorithm should declare that no binary BCH code with length $N<128$ could be found for this K which was sufficiently powerful to satisfy Equation (3-2) . The algorithm should then return to step 2 and iterate K. When the algorithm fails to find a sufficiently powerful binary BCH code, then the limited conclusion is that $N_{min}(K) \geq 128$ for this particular K. Under these conditions, plotting $N_{min}(K)$ as a function of K, or even the redundancy ratio $\frac{N_{min}(K)}{K}$ as a function K, becomes more difficult since it is not certain what value should be substituted for $N_{min}(K)$ for these points. For this reason, we choose to display $N_{min}(K)$ in tabular rather than graphical format.

Algorithm II is performed over a range $35<K<46$ for both binary cyclic codes and binary BCH codes. Assuming values of $BER = 10^{-2}$, $P_{CI} = 0.999$, image size $I = 20$ kbits, then the minimum code length $N_{min}(K)$ results are listed in Table 4. The middle column lists the minimum number of output bits required by odd-N binary cyclic codes, and the right column lists $N_{min}(K)$ for binary BCH codes. The left column does the same for the best possible (D_{min} evaluated at the upper bounds) binary linear codes, using the same values generated earlier by Algorithm I to construct Figure 3.3.

Table 4. The minimum number of output bits $N_{min}(K)$ generated by Algorithm II for all binary cyclic (middle column) and binary BCH codes (right column) satisfying $35<K<46$, N odd, and $N<128$. Let $BER = 10^{-2}$, $P_{CI} = 0.999$, $I = 20$ kbits. For comparison, $N_{min}(K)$ generated by Algorithm I for optimal (D_{min} at upper bounds) binary linear codes is shown in the left column. The error correction power T of the minimum-length ($N_{min}(K),K$) code is shown in parentheses.

	Minimum code length $N_{min}(K)$		
Number of input bits K	Optimal Binary Linear Codes	Binary Cyclic Odd-N Linear Codes, $N<128$	Binary BCH Linear Codes, $N<128$
36	66 ($T = 7$)	73 ($T_{cyclic} = 7, T_{opt} = 9$)	105 ($T_{BCH} = 10, T_{opt} = 17$)
37	67 (7)	85 (8,12)	93 (8,14)
38	69 (7)	93 (8,14)	93 (8,14)
39	70 (7)	79 (7,9)	117 (10,20)
40	71 (7)	79 (7,9)	105 (8,16)
41	72 (7)	105 (9,16)	none found for $N<128$
42	73 (7)	105 (8,16)	127 (14,22)
43	74 (7)	105 (8,15)	127 (14,21)
44	76 (7)	89 (8,10)	none found for $N<128$
45	77 (7)	89 (8,10)	none found for $N<128$

We interpret the table as follows. At $K = 40$, the minimum-redundancy search over all (N,40) optimal binary linear codes employing Algorithm I produces a (71,40) binary linear code with an error correction power $T = 7$ bit errors. A minimum-redundancy search employing Algorithm II over odd-N binary cyclic codes produces a (79,40) code that can correct $T = 7$ errors. Algorithm II applied to binary BCH codes produces a minimum-redundancy (105,40) code that can correct $T = 10$ errors. The error correction capability corresponding to the ($N_{min}(K),K$) code is shown in parentheses. For binary cyclic and

BCH codes, an additional parameter T_{opt} is shown paired with T, in the form (T,T_{opt}). T_{opt} corresponds to the theoretically optimal error correction ability that could be obtained from the upper bounds on minimum code distance for this $(N_{min}(K),K)$ pair.

The imposition of cyclic structure and BCH cyclic structure on binary linear block codes causes the minimum code distance of cyclic and BCH codes to shrink below the theoretically best upper bounds on D_{min} for each N and K. This weakens the error correction ability of (N,K) cyclic and BCH codes relative to optimum-distance (N,K) codes. For example, the minimum-redundancy (79,40) cyclic code can correct up to $T_{cyclic} = 7$ errors, while the optimal (79,40) binary code with the maximum permissible D_{min} can correct up to $T_{opt} = 9$ errors. The loss in error correction performance is even more marked for binary BCH codes. The table shows that the minimum-redundancy (105,40) BCH code that satisfies Equation (3-2) can correct up to $T_{BCH} = 8$ errors, while the optimal (105,40) binary code derived from the upper bounds should be able to correct up to $T_{opt} = 16$ errors.

As we move from left to right across columns in Table 4, the weaker error correction performance of binary cyclic and BCH codes relative to optimal binary linear codes causes an expansion in minimum code length $N_{min}(K)$ and hence increases the overall cost in overhead. For example, at $K = 40$, the (71,40) optimal linear code incurs an overhead of 31 bits, and a redundancy ratio $\frac{N_{min}(K)}{K}$ equal to 1.8. The best cyclic (79,40) code requires an additional 8 bits of redundancy over the binary linear code, and a redundancy factor of about 2.0. The best binary BCH (105,40) code requires an additional 26 bits of overhead over the best cyclic code, and an overall bandwidth expansion factor of 2.6. The same pattern is exhibited throughout the selected range of K.

Table 4 indicates that optimal binary linear codes require about a doubling of the bandwidth for this range of K in order to adequately protect a fragmented image. The performance of binary cyclic codes varies widely, occasionally approaching that of the optimal codes(e.g. $K = 36$) but other times matching the poor efficiency of binary BCH codes (e.g.

$K = 38$), though usually lying somewhere in between (e.g. $K = 42$). For binary BCH codes, we can conclude that bounded-distance decoding of moderately long binary BCH codes requires a minimum bandwidth expansion of about 2.5 in order to satisfy Equation (3-2) for the chosen range of K, though the more common case resulted in a tripling of bandwidth.

Another reason for the high overhead cost of binary BCH codes, besides their relatively weak error correction performance, is the relatively sparse distribution of existing binary BCH codes in (N,K) space. Binary BCH codes do not exist for all possible N and K. For example, the reference tables show that at $K = 41$, binary BCH codes only exist for $N=$ 51, 55, 63, 65, 95, and 105 in the range $N<128$. At $K = 36$, binary BCH codes only exist for $N = 57, 63, 65, 69, 71, 73, 77, 85, 105, 119$, and 127 in the range $N<128$. While one of the perceived advantages of BCH codes is that there is an ample selection of block lengths and code rates [153], for our purposes the scattering of codes is sparse enough to cause large jumps in calculated redundancy. When Algorithm II was applied at $K = 36$, the (85,36) binary BCH code failed the test in step 5 of Algorithm II. The next largest code available for testing was the (105,36) binary BCH code, which also turned out to be the minimum-length code. The sparse distribution of BCH codes in this case permitted a gap of 20 bits to arise between tested codes and in general contributes to the sudden jumps in $N_{min}(K)$ as we move from left to right in a row in the table.

The scattering of BCH codes in (N,K) space also is responsible for the unevenness of $N_{min}(K)$ as we move down a column in Table 4. As K increases, the minimum code length should also increase. However, $N_{min}(K)$ is not a monotonically increasing function of K due to the sparseness property, though the general trend is towards lengthening $N_{min}(K)$. In contrast, for the optimal linear codes, $N_{min}(K)$ (and by extension $\frac{N_{min}(K)}{K}$) is a relatively smooth monotonically non-decreasing function of K, since the optimal codes were assumed to exist for each N and K, $N>K$, and were derived from well-behaved mathematical bounds.

Table 4 does not provide sufficient information to determine whether longer BCH codes are necessarily better (i.e. have a higher rate $\frac{K}{N}$ for the same error correction performance) than short BCH codes. While $N_{min}(K)$ roughly increases with K, we cannot conclude from the table that the bandwidth expansion factor $\frac{N_{min}(K)}{K}$ decreases with increasing K for binary BCH codes in the same manner as we had found for optimal binary codes (depicted in Figure 3.3). A number of factors interfere with our ability to make a definitive statement. First, the sampling range of K is relatively small. In addition, for several values of K no sufficiently powerful BCH code could be find for $N<128$. This uncertainty reduces even further our set of available data points. Moreover, the unevenness of $N_{min}(K)$ also leads to an fluctuating redundancy ratio $\frac{N_{min}(K)}{K}$. For example, the (105,36) and (127,43) minimum-length codes both triple the bandwidth, while the (93,37) code in between only expands the bandwidth by a factor of 2.5, making it difficult to show that $\frac{N_{min}(K)}{K}$ decreases as K increases. Finally, it is known that BCH codes are asymptotically weak, in the sense that for a fixed rate $\frac{K}{N}$, the ratio of $\frac{D_{min}}{N}$ for BCH codes approaches zero in the limit as block length N increases [24][153]. For these reasons, our data does not support a conclusion that longer binary BCH codes are necessarily better than short binary BCH codes.

Next, we consider the sensitivity of Algorithm II to variations in the parameters P_{CI} and BER. We confine ourselves to analyzing binary BCH codes only. Four cases are considered and presented in tabular form in Table 5. The baseline case with parameters $BER = 10^{-2}$ and $P_{CI} = 0.999$ was same one used to generate Table 4.

If we relax P_{CI} to 0.99, and keep BER at 10^{-2}, then we obtain the far right "Relaxed P_{CI}" column. If we are willing to tolerate an image loss rate of 1 in 100 instead of 1 in 1000 (baseline P_{CI}), then a significant reduction in overhead can be achieved for many values of K, though not all. For example, the best baseline BCH code at $K = 42$ has $N_{min}(42) = 127$. By relaxing the P_{CI} objective, the new $N_{min}(42)$ can be reduced to 93. At this higher image loss rate, the binary BCH code with the smallest redundancy ratio is the (73,36) code, which would require at the minimum a doubling of bandwidth.

Table 5. For binary BCH codes only, test the sensitivity of $N_{min}(K)$ generated by Algorithm II to variations in *BER* and P_{CI}. All codes satisfy $35<K<46$, N odd, and $N<128$. Let $I = 20$ kbits. *BER* and P_{CI} vary across columns.

	Minimum code length $N_{min}(K)$			
Number of input bits K	Baseline: $BER = 10^{-2}$, $P_{CI} = 0.999$	"Worse *BER*" $BER = 2\text{x}10^{-2}$, $P_{CI} = 0.999$	"Worse *BER* & Relaxed P_{CI}" $BER = 2\text{x}10^{-2}$, $P_{CI} = 0.99$	"Relaxed P_{CI}" $BER = 10^{-2}$, $P_{CI} = 0.99$
36	105	127	105	73
37	93	NF	NF	85
38	93	NF	NF	93
39	117	NF	NF	105
40	105	NF	NF	105
41	None Found for $N<128$	NF	NF	NF
42	127	127	127	93
43	127	127	127	93
44	NF	NF	NF	117
45	NF	NF	NF	NF

If the *BER* is increased slightly to $2\text{x}10^{-2}$, and we keep P_{CI} at its baseline value of 0.999, then we obtain the "Worse *BER*" column. No sufficiently powerful binary BCH codes with $N<128$ could be found for most K in this column. In comparison, for only a few values of K in the baseline column does $N_{min}(K)$ exhibit the label "None Found". Therefore, similar to our conclusion for the optimal codes in Section 3.2, the minimum code length for binary BCH codes is extremely sensitive to small increases in the probability of bit error at these levels of *BER*. Under these slightly noisier conditions, the binary BCH code with the smallest redundancy factor is the (127,43) code. In order to adequately protect an image at this *BER* using binary BCH codes, the minimum overhead cost would require approximately a tripling of bandwidth.

3.4 Bounded-distance decoding of a fragmented image encoded by Reed-Solomon codes

Reed-Solomon (RS) codes constitute an important subset of *Q*-ary (non-binary) BCH codes. Reed-Solomon codes derive much of their importance from the fact that the fast Berlekamp-Massey decoding algorithm can be applied to them. The relative ease of hardware implementation of RS coders and decoders has led to their wide application in consumer products like compact discs and to their use over deep-space and satellite communication links [152]. RS codes are also popular because of burst error correction and erasure detection properties.

The non-binary nature of RS codes means that Equation (3-2) needs to be revised to account for symbol-based error correction, where there are *M* bits per RS code symbol, i.e. the number of possible symbols $Q = 2^M$. Again assuming a BSC error model, then the probability of symbol error, or symbol error rate *SER*, is given by $SER = 1-(1-BER)^M$. Some texts have used *Q*-ary symmetric error channel models instead of the BSC model, where the probability of transitioning to any symbol other than the original symbol is given by *p*, and $SER = p(Q\text{-}1)$ [87][152]. We do not use this model, and instead continue to adhere to the BSC model used by other texts [46][100]. For *Q*-ary codes, Equation (3-2) is transformed into the following relation:

$$\sum_{i=0}^{T} \binom{N}{i} \cdot SER^{i} \cdot (1-SER)^{N-i} \geq [P_{CI}]^{(K \cdot M)/I} \tag{3-3}$$

Next, we evaluate this equation for various values of N, K, and M, applying a procedure similar to the one used for binary BCH codes. One key property of RS codes is that the code length N is directly related to the finite field size Q by $N = Q\text{-}1$ [153]. We consider the standard non-extended non-shortened case where $Q = 2^M$ and $N = 2^M\text{-}1$. Another key property of RS codes is that their minimum code distance D_{min} is determined directly by the following relation: $D_{min} = N\text{-}K\text{+}1$. Thus, unlike binary BCH codes, there is no need to consult tables to find D_{min}, because the D_{min} of RS codes can be computed precisely from the Singleton bound [153] (see Section 3.2).

Our minimum-length evaluation algorithm for RS codes is summarized below.

Algorithm III:

1. Choose a range $[K_{min}, K_{max}]$ for K. Initialize $K = K_{min}\text{-}1$.
2. Iterate $K = K\text{+}1$. Find J, the first power of 2 such that $2^J\text{-}1 > K$. Initialize $M = J\text{-}1$.
3. Iterate: $M = M\text{+} 1$. Let $N = 2^M\text{-}1$. (We are increasing N by powers of 2).
4. $D_{min} = N\text{-}K\text{+}1$. Let $T = \left\lfloor \frac{D_{min} - 1}{2} \right\rfloor$.
5. Is Equation (3-3) satisfied at $(N(M),K,T)$? If no, return to step 3.
6. If yes, then we have obtained the lowest $N_{min}(K)$ for this K that creates a large enough D_{min} for an RS code to satisfy Equation (3-3) . Return to step 2 unless K exceeds K_{max}.

The results of our iterative evaluation algorithm are plotted in Figure 3.4, for the parameters $BER = 10^{-2}$, a relaxed $P_{CI} = 0.99$, $I = 20$ kbits. The stairstep pattern is a direct consequence of the sparseness of the available code lengths (separated by powers of two). Note that on each "stairstep" the point K farthest to the right just before the step jump is optimal in terms of reducing the overhead ratio for the constant "stairstep" code length N.

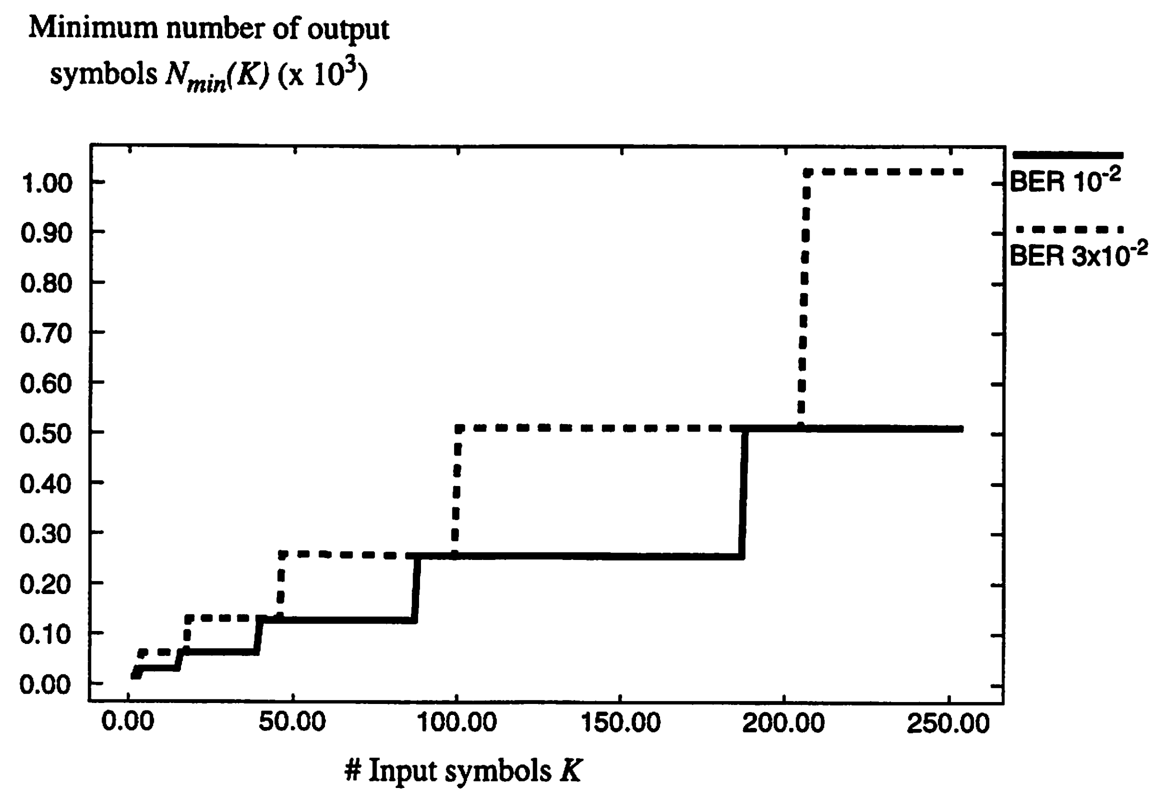


Figure 3.4 Minimum number of output symbols $N_{min}(K)$ required by Reed-Solomon codes to satisfy Equation (3-3) , i.e. deliver images of size 20 kbits at an image success rate P_{CI} exceeding 0.99 over a 10^{-2} *BER* link (solid plot). A second dotted plot is also shown for *BER* = $3\text{x}10^{-2}$. RS redundancy = *N-K*.

For 256-ary RS codes ($N = 255$), the minimum-length code with the lowest redundancy ratio was found to be the (255,175) code. Therefore, in order to deliver our image data with acceptably low image loss rates, about 1.5 times bandwidth expansion is required. In general, Figure 3.4 indicates that longer RS codes more effectively use redundancy, resulting in lower redundancy ratio $\frac{N_{min}(K)}{K}$. We also note that standard primitive RS codes have a relatively sparse set of available code lengths that is even more thinly distributed than binary BCH codes, an observation that inflates the RS redundancy ratio.

Next, we consider the sensitivity of RS codes to variations in *BER*. The second dotted curve in Figure 3.4 evaluates Equation (3-3) for *BER* = $3\text{x}10^{-2}$, at the same P_{CI} and image size I as the lower curve. The best code for the stairstep length of $N = 511$ is about a (511,200) code, or about a 2.5x redundancy code. Similarly, the best (255,*) RS code is about a (255,100) code, again about 2.5x redundancy. RS codes exhibit the same sensitivity to small increases in probability of bit error at this level of *BER* as binary linear block codes and binary BCH codes. We can arrive at roughly the same conclusion by inspecting the family of "waterfall" curves [8][46] which plot the output *BER* performance of a family of RS codes, say the (64,*) RS codes, as a function of the input *BER*. Because the curves are tightly packed around 10^{-2} input *BER*, a slight increase in the input *BER* can cause force a large jump in code redundancy in order to preserve the same output *BER*. While these waterfall curves are useful in confirming our intuition, they only consider the effect of single-packet delivery, and do not consider the impact of multi-packet image delivery captured in Equation (3-3) .

We would also like to compare RS code performance to binary BCH performance. Since there are M bits per symbol, then a (255,175) RS code is in fact a (255*8,175*8) = (2040,1400) binary code, though arithmetic operations are performed on a different finite field. Consequently, these RS codes are considerably longer than the (N<128) binary BCH codes considered in the previous section. Comparing similarly sized codes, we find that at $K = 11$ input symbols, the minimum-length RS codes are of length 31, so that they have 5 bits per symbol. The (31,11) symbol-based code is equivalent to a (155,55) binary code, and the overall cost in overhead approximately triples of the bandwidth. In comparison, at $K = 45$, minimum-length binary BCH codes require at least a (128,45) code for a several different combination of *BER* and P_{CI}. So both binary BCH codes and RS codes require a minimum tripling of bandwidth for moderate-length codes when the image size is several tens of thousands of bits, the *BER* is near 10^{-2}, and the subjectively acceptable image loss rates are less than 1 in 100.

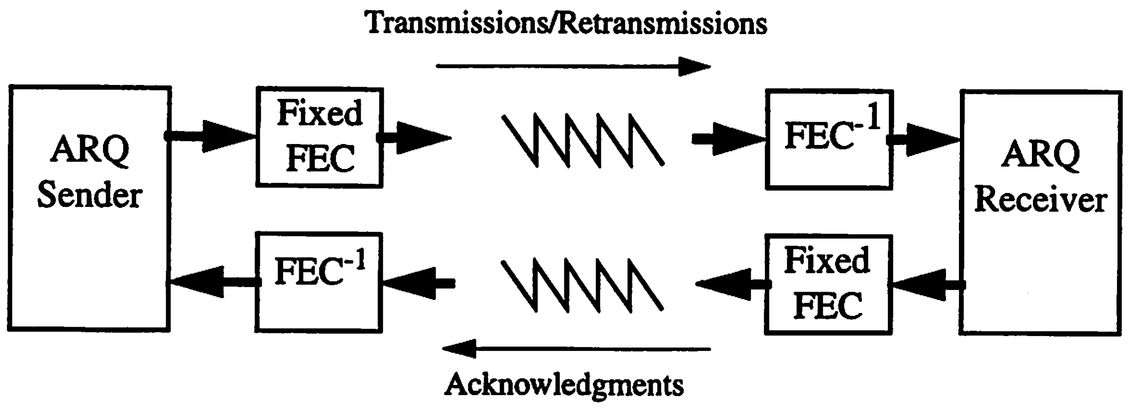


Figure 3.5 Type-I Hybrid ARQ. A fixed-rate FEC coder is embedded within the retransmission loop of a conventional ARQ protocol. The FEC coder helps to correct wireless bit errors, and can be applied in both the forward and reverse directions.

3.5 Minimum redundancy evaluation of a Type-I Hybrid FEC/ARQ protocol

So far in this chapter, we have considered only open-loop FEC systems based on linear block codes which lack feedback from the receiver. In this section, we consider a hybrid FEC/ARQ closed loop error protection scheme that integrates closed-loop ARQ protocols with FEC.

A straightforward way of combining FEC and ARQ, called *Type-I Hybrid ARQ*, is to embed an FEC coder operating at a fixed coding rate within the retransmission loop of any repetition-based ARQ protocol [78]. The Type-I Hybrid-ARQ protocol is shown in Figure 3.5, where FEC is applied in both directions over the wireless link. Our goal is to determine the minimum redundancy required by the inner FEC coder in order to deliver an image reliably across the noisy link within the interactive latency bound.

3.5.1 Optimal binary codes

The performance of a Type-I Hybrid ARQ protocol will depend on the choice of retransmission protocol as well as the type of FEC employed. Our choice of protocol is ideal SRP due to its optimal latency performance. For the type of FEC scheme, we choose the class of optimal binary linear codes studied in Section 3.2 because they minimize the redundancy over all binary linear codes. Together, these two choices form an ideal T-I H-ARQ protocol that serves as a lower bound on the amount of redundancy required for all T-I H-ARQ protocols. In addition, these two choices are both mathematically tractable, and we can reapply the equations from Chapter 2 and Section 3.2 here. We must also decide whether to evaluate the T-I H-ARQ protocol at the loose latency bound or the tight latency bound of the previous chapter. Again, we make the most optimistic choice, namely the loose lower bound represented by Equation (2-5) , in order to maintain the claim that this section's analysis produces a lower bound on redundancy.

Analyzing T-I H-ARQ requires some modifications to Equation (2-5) in order to account for the embedded fixed-rate FEC coder. Equation (2-5) represented a lower bound on the minimum amount of delay suffered by retransmission-based protocols over noisy channels. Two components contributed to the delay in Equation (2-5) : a *BER factor* caused by multiple retransmissions due to bit errors; and a *BW factor* due to the wireless channel's limited bit rate. The *BER factor* dominated the delay when the protocol's packets were large enough relative to the *BER* to cause multiple bit errors per transmitted packet. If FEC were applied to combat wireless bit errors, then the effective *BER* affecting each packet sent by the protocol could be reduced to the point that the number of retransmissions caused by link noise would no longer dominate the protocol's delivery latency. However, any increase in error correction also expands the bandwidth. Both effects must be accounted for in our T-I H-ARQ analysis.

First, we investigate how FEC affects the *BER* factor. If we assume ideal SRP as the retransmission protocol for T-I H-ARQ, then from the previous chapter the number of attempts A required to send one reliable version of an N-bit packet through a noisy BSC is geometrically distributed, i.e. $P[A = i] = (1 - P_g)^{i-1} \cdot P_g$, where P_g is the probability of correctly receiving the *N*-bit packet after one transmission. We found that $E[A] = \frac{1}{P_g}$, and later used this result to derive Equation (2-5) , the loose lower bound on the average amount of time required for reliable image transmission. We summarize the bound here as $E[T_{\text{image}}] \geq F \cdot E[A_{fragment}] \cdot PTT_{fragment}$.

In Chapter 2, we assumed that no FEC was applied to each *J*-bit packet, so that the *BER* of the channel directly affected each packet, and $P_g = (1 - BER)^J$. Assume that an image *I* is fragmented into *F* fragments. Define the size of each image fragment $J = \frac{I}{F}$, *H* = header overhead, and the number of input bits to the FEC encoder $K = J + H$. In T-I H-ARQ, each *J*-bit image fragment will be protected by an *(N,J+H)* code. Therefore, P_g must be modified to account for the error correction abilities of the code.

The probability of correctly receiving *J+H* bits is the probability of correctly decoding an *N*-bit packet, which is given by Equation (3-1) under the assumption of bounded-distance decoding and BSC errors. We repeat Equation (3-1) below, and change P_g to the following value:.

$$P_g = Prob\begin{bmatrix} \text{Correct Decoding} \\ \text{of} \quad (N-bit) \quad \text{packet} \end{bmatrix} = \sum_{i=0}^{T} \binom{N}{i} \cdot BER^i \cdot (1 - BER)^{N-i} \qquad (3\text{-}4)$$

Next, we investigate how FEC affects the *BW* factor. We need to account for the bandwidth expansion caused by FEC. This problem is easily solved by changing the packet transmission time *PTT* to $\frac{N}{BW}$. Thus, Equation (2-5) is transformed into the following form for Type-I Hybrid-ARQ:

$$E[T_{\text{image}}] \geq F \cdot E[A_{fragment}] \cdot PTT_{fragment} \tag{3-5}$$

$$= F \cdot \frac{1}{P_g} \cdot \frac{N}{BW} = \frac{1}{\sum_{i=0}^{T} \binom{N}{i} \cdot BER^i \cdot (1-BER)^{N-i}} \cdot \frac{F \cdot N}{BW}$$

The number of output bits $N = \frac{I}{F} + H + R$, where R equals the number of redundancy bits added, and $N>K$. Summarizing, our independent variables are image size I, header length H, wireless bandwidth BW, wireless probability of bit error BER, the number of input bits K, the number of output bits N, and the choice of linear block code. F is a dependent variable of I, H and K. The dependent redundancy variable R is a function of N and K. The dependent error correction power variable T is a function of N, K, and also of the linear code chosen.

A systematic procedure for determining the minimum redundancy is to follow the double loop approach developed in previous sections for analyzing open-loop FEC systems. Fix I, H, BW, BER, and fix the choice of linear block code to be the class of optimal maximum-permissible-D_{min} binary codes. The remaining independent parameters that can be varied are K and N. Iterate K in the outer loop, and iterate N in the inner loop. By fixing K, we fix the fragmentation factor also, since $K = \frac{I}{F} + H$. For each fixed K, increment N and test to see if the latency predicted by the loose lower bound in Equation (3-5) for this optimal (N,K) code falls below the desired interactive latency threshold $L_{interactive}$. If not, continue to increase N until the minimum-redundancy code of length $N_{min}(K)$ is found that produces a small enough image delivery latency. The algorithm is detailed below.

Algorithm IV:

1. Choose a range $[J_{min}, J_{max}]$ for J. Initialize $J = J_{min}-1$. J represents the number of bits per image fragment, i.e. $J = \frac{I}{F}$.

2. Iterate $J = J+1$. Set the number of input bits $K = J+H$. Initialize $N = K$.

3. Iterate $N = N + 1$.

4. Find the optimal $D_{min}(N,K) = \min$ (D_{min}'s obtained from the Singleton, Plotkin, Hamming, and Griesmer bounds). Let $T_{opt}(N, K) = \left\lfloor \frac{D_{min}(N, K) - 1}{2} \right\rfloor$.

5. Calculate the delay from the loose lower bound in Equation (3-5) for (N,K,T_{opt}). Does the estimated delay fall below the desired interactive latency threshold $L_{interactive}$? If not, go back to step 3.

6. If yes, then we have obtained the lowest $N_{min}(K)$ that has sufficient error correction to reduce the delay predicted by Equation (3-5) below $L_{interactive}$. Go back to step 2 until J exceeds J_{max}.

Algorithm IV is performed with the following parameters: I = 20 kbits, BW = 500 kbits/s, $BER = 10^{-2}$, header size H = 100 bits, and the desired interactive latency bound $L_{interactive} = 100ms$. For these parameters, the resulting behavior of $N_{min}(K)$ partitions the K axis into three regions. In the first region, the header overhead is so large relative to the size of each image fragment that the delay predicted by Equation (3-5) exceeds $L_{interactive}$ regardless of whether FEC is applied or not, and therefore $N_{min}(K)$ does not exist for these K. In the second region, the error correction is insufficient to reduce the delay below $L_{interactive}$, and again no finite $N_{min}(K)$ can be found for these K. Only for the third region can a sufficiently powerful code be found which manages to lower the image transfer delay below the interactivity threshold. We examine in detail each of these three regions of K below.

Since $K = \frac{I}{F} + H$, then it follows that when the number of image bits $\frac{I}{F}$ is very small relative to the header overhead H, then the resulting bandwidth expansion can inflate the image delivery time above the desired limit $L_{interactive}$. For example, if there are 50 bits of image data and 100 bits of header per K-bit packet prior to FEC, then the header overhead essentially triples the bandwidth. Since the original 20 kbit image requires a minimum of 40 ms to transmit over the 500 kbit/s channel, then the packetized image with overhead will require 120 ms, exceeding our objective $L_{interactive}$ without even considering FEC. Thus, Algorithm IV need not even be executed for these K.

To find the valid range of K over which FEC can be applied, we define a bandwidth expansion factor $\beta = \frac{I/F+H}{I/F}$. If $\beta \cdot \frac{I}{BW} > L_{interactive}$, then bandwidth expansion alone due to header overhead per fragment will exceed our desired delay. Substituting parameter values, we find that when the size of each image fragment $I/F \le 66$, then the header overhead per fragment pushes the image transfer delay above $L_{interactive}$. Equivalently, when $K \le 166$ then Algorithm IV need not be run.

In the range $K > 166$, the header overhead is a small enough percentage of each image fragment to permit FEC on each fragment. We executed Algorithm IV in the range $K > 166$ and found that $N_{min}(K)$ could not be obtained for values of N near 166. Specifically, evaluation of Equation (3-5) reveals that for the range $166 < K < 193$, no optimal binary linear (N,K) block code could be found such that the loose lower bound $E[T_{\text{image}}] \le L_{interactive}$. An investigation as to why the algorithm did not produce a finite $N_{min}(K)$ in this narrow region of K shows that the error correction power of any (N,K) optimal binary code is too weak to lower the image transfer delay below the interactivity threshold. We come to this conclusion both from theoretical analysis of Equation (3-5) as well as empirical inspection of the generated values of delay. For a fixed K, we observe that, as N increases, the delay $E[T_{\text{image}}]$ decreases exponentially, since the improving error correction ability helped reduce the retransmission BER factor $E[A] = \frac{1}{P_g}$ in Equation (3-5). Eventually, the delay reached a minimal value before starting to rise again for large N, when the linear dependence upon the $\frac{N}{BW}$ factor in Equation

(3-5) begins to dominate. This behavior is very similar to Figure 2.4, and the resemblance is not surprising since they are both governed by the same loose lower bound equation. If $E[T_{image}]$ exceeds $L_{interactive}$ at the value of N which minimizes the delay, then it will be impossible to find any optimal binary linear (N,K) block code for this value of K that delivers an image within the $L_{interactive}$ bound. An empirical inspection of the values of delay generated by Algorithm IV verified this predicted behavior as a function of N for the range of input values $166 < K \leq 193$.

In the range $K > 193$, Algorithm IV was able to find a finite minimum code length $N_{min}(K)$ for each K. In Figure 3.6, we plot two redundancy ratios as a function of K, the traditional $\frac{N_{min}(K)}{K}$ as well as a second metric $\frac{N_{min}(K)}{K-H}$. The first metric $\frac{N_{min}(K)}{K}$ measures the bandwidth inflation factor caused by an $(N_{min}(K),K)$ optimal binary block code embedded within the T-I H-ARQ protocol. The second metric measures the total amount of bandwidth expansion due to both the header overhead as well as the FEC overhead. By comparing the two curves, we can deduce the proportion of overhead due to headers and the proportion due to block coding redundancy.

At $K = 250$ input bits, there are 100 header bits and 150 image-related payload bits. The lower curve predicts that a minimum of about 1.1 times bandwidth inflation is required, corresponding to a minimum output code length of about $N = 280$ bits, or a (280,250) code. The upper curve represents the ratio of N to the number of payload bits, and equals $\frac{280}{150} = 1.9$ at $K = 250$.

When both block coding redundancy and header overhead are considered, the upper curve shows that roughly a doubling of bandwidth is required by a T-I H-ARQ protocol in order to deliver an image reliably within the interactive latency threshold. Comparing the two curves, we see that most of the redundancy is due to header overhead, rather than error correction redundancy. For example, at $K = 250$ bits, the per-packet header overhead is 100 bits, while the per-packet FEC overhead is only 30 bits.

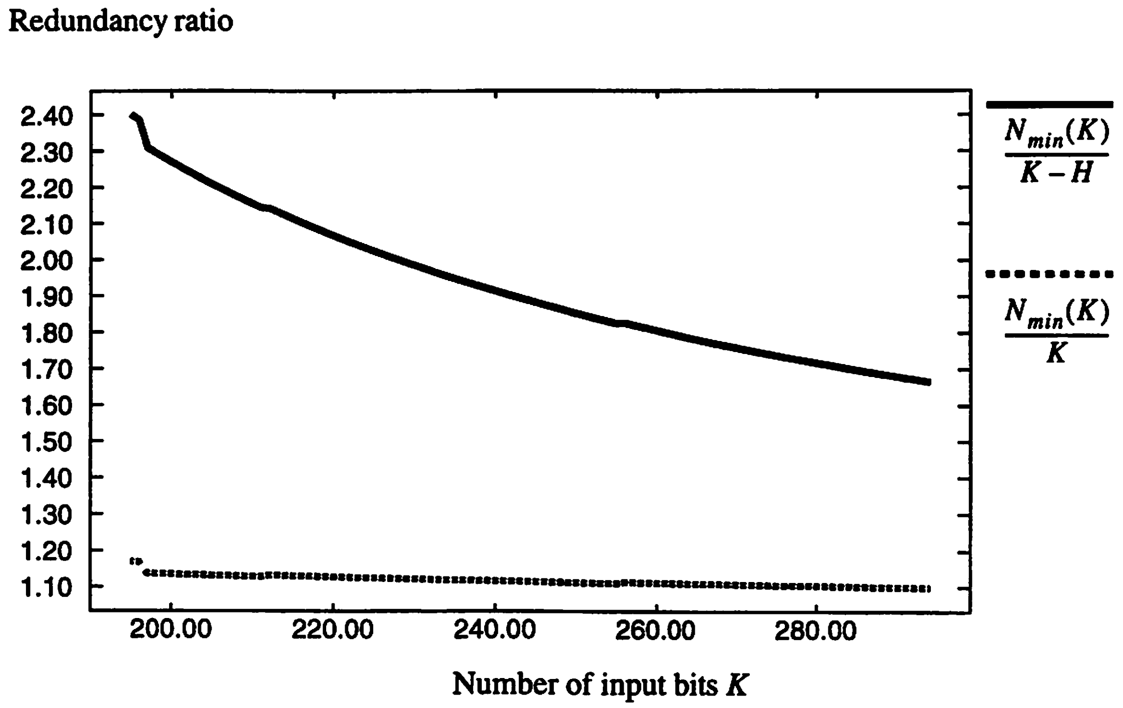


Figure 3.6 The minimum bandwidth expansion factor for a Type-I Ideal SRP Hybrid ARQ protocol is plotted as a function of the number of input bits *K*. The upper curve shows the inflation caused by the combined overhead of headers and FEC redundancy. The lower curve shows the bandwidth inflation due to FEC redundancy alone. The curves were generated by Algorithm IV.

The closed-loop redundancy rates of Figure 3.6 can be compared with the open-loop redundancy rates obtained for the best binary linear block codes in Chapter 2. First, the algorithm of Section 3.2 needs to be applied to the appropriate range of K>193. Furthermore, this algorithm, as well as Equation (3-2) , needs to be modified to account for the effects of header overhead, in order to achieve a fair comparison. We can redefine N in Equation (3-2) to equal K+H+R, where R consists of the block code's redundancy bits. Also, the algorithm is modified to start from K+H input bits, instead of K bits. The results of the new header-cognizant open-loop analysis can then be used for comparison. For example, for K=200 (and $BER = 10^{-2}$, $P_{CI} = 0.999$), the open-loop system would require about 70 redundancy bits, at an image loss rate of 10^{-3}, and a delay of

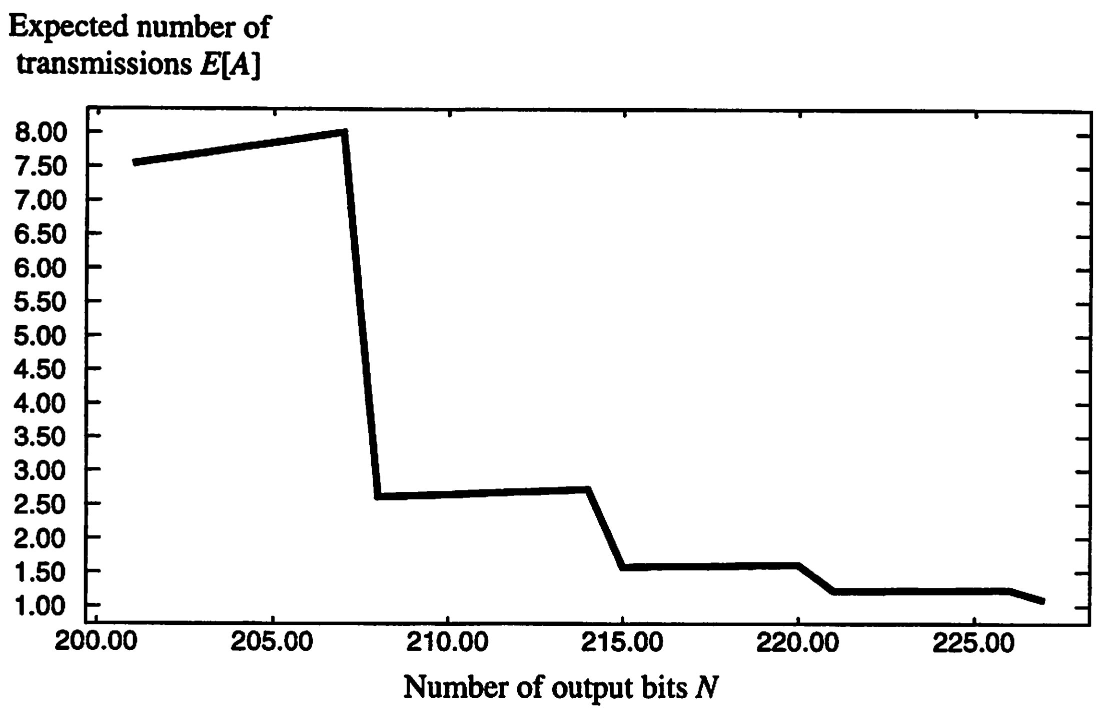


Figure 3.7 The expected number of transmissions $E[A]$ for a Type-I Hybrid ARQ protocol falls off rapidly as the size N of the error correcting $(N,200)$ code is increased. $E[A] = \frac{1}{P_g}$ (assumes ideal SRP). P_g is defined in Equation (3-4).

$F \cdot \frac{N}{BW} = \frac{20K}{200-100} \cdot \frac{260}{500K} = 104ms$. In comparison, at $K = 200$, the closed-loop T-I H-ARQ system requires 27 redundancy bits, transmits images with complete reliability, and delivers the image with a delay of ~100 ms. Thus, the closed-loop T-I H-ARQ system exhibits an improvement over the open-loop FEC system in all respects: lower redundancy rate, higher reliability, and faster delivery.

To see how the T-I H-ARQ protocol is managing to achieve such a low FEC redundancy rate of around 1.1 to 1.2, we examine Equation (3-5). Error correction coding is responsible for reducing the number of retransmissions $E[A]$ required by the protocol. For a fixed point $K = 200$, we plot $E[A] = \frac{1}{P_g}$ as a function of the code length N in Figure 3.7. As N is allowed to increase, the error correction power of the optimal binary code should improve, and the number of retransmissions $E[A]$ should correspondingly fall. Figure 3.7 shows that $E[A]$ falls very rapidly, below a value of 1.5 within the first 30 redundancy bits. An optimal (227,200) block code will create a sufficiently clean channel so that on average only ~1.1 repetitions are needed per fixed-size packet to send one clean version of that packet through the dirty link. Since $E[A] = \frac{1}{P_g}$ for ideal SRP, then an alternative interpretation is that the probability of error-free decoding of a corrupt packet will rise above 90% by adding about 30 error protection bits. Thus, FEC coding is able to dramatically reduce $E[A]$ with only a relatively minor cost in overhead.

The stairstep appearance of Figure 3.7, and the slight rising slope in each step, can be attributed to the behavior of the error correction variable T. Recall that T is derived from upper bounds on D_{min}, which happen to change slowly as N is increased. For a fixed K, as N increases the error correction power T will remain constant over a long interval of N between increments. Each increment in T corresponds to the ability to correct one additional error, and results in a large improvement in the probability of correct packet decoding, hence the visible drop in $E[A]$. Also, all codes along a stair step share the same T, so that the longest codes (farthest to the right in the figure) will be the least efficient, thereby causing the slight upwards slope along each stair step.

While Figure 3.6 suggests that about a doubling of bandwidth is necessary to support T-I H-ARQ delivery over a noisy link, there are several ways to reduce this bandwidth cost and make T-I H-ARQ delivery more efficient. First, we note that the upper curve $\frac{N_{min}(K)}{K-H}$ decreases as the code length increases, which indicates that longer block codes can help reduce the overhead penalty. Second, our analysis of T-I H-ARQ has assumed that an (N,K) code is only applied once per packet, and therefore the header overhead is suffered for each (N,K) encoding operation. It is possible for different sections of a packet to undergo separate (N,K) encodings, so that the header overhead is confined to a single (N,K) encoding, or equivalently the header overhead is thinly distributed across multiple encodings. Under these conditions, then the overall bandwidth expansion may be much smaller than the doubling derived here.

Finally, we observe that several factors which we ignored in our analysis will tend to make it more difficult for T-I H-ARQ protocols to meet the interactive latency bound. For example, we used the loose lower bound on latency to analyze a T-I H-ARQ protocol. For finite length image bursts, the tight bound on latency derived in Section 2.4 may be a much better estimate of the image transfer delay. The higher delay estimate will further limit how much FEC redundancy can be added before the interactive latency bound is exceeded. Also, our derivations assumed that the very best binary linear codes with the maximum permissible D_{min} were used. If instead binary BCH codes are employed, then we recall from Section 3.3 that the error correction performance can suffer, leading to a bandwidth inflation effect. In addition, if the retransmission scheme of the T-I H-ARQ protocol is not SRP, or has finite window and/or buffer sizes, then the T-I H-ARQ protocol will have greater difficulty meeting the interactive delay bound. Finally, T-I H-ARQ protocols operate with fixed FEC, and therefore have a well-known problem handling the bursty errors of time-varying wireless channels. We will discuss this classic problem in more detail in Chapter 6.

3.5.2 Reed-Solomon codes

In this section, we analyze RS codes within a T-I H-ARQ context. In particular, we are interested in the minimum-redundancy RS code for a given K which can deliver a fragmented image within the delay bound of 100 ms at severe BER's. For RS (N,K) codes, there are M bits per symbol, and the symbol error rate SER differs from the BER, i.e. $SER = 1-(1-BER)^M$. However, the rest of the analysis is comparable to the previous subsection's analysis for optimal binary codes. Therefore, assuming the same parameters definitions as before, we can rewrite the expected image transfer delay as follows:

$$E[T_{image}] \geq F \cdot E[A_{fragment}] \cdot PTT_{fragment} \tag{3-6}$$

$$= \frac{I}{K \cdot M} \cdot \frac{1}{\sum_{i=0}^{T} \binom{N}{i} \cdot SER^{i} \cdot (1-SER)^{N-i}} \cdot \frac{N \cdot M}{BW} = \frac{\frac{I}{BW} \cdot \frac{N}{K}}{\sum_{i=0}^{T} \binom{N}{i} \cdot SER^{i} \cdot (1-SER)^{N-i}}$$

Equation (3-6) was evaluated at $3\text{x}10^{-2}$ *BER*, an image size of 20 kbits, and a bandwidth of 500 kbit/s. The results for a selected few values of *K* are shown in Table 6. We

Table 6. Minimum-redundancy RS codes embedded within a T-I H-ARQ protocol which can deliver an image within 100 ms.

Minimum-redundancy RS (*N*,*K*) code	*BER*	
	10^{-2}	$3\text{x}10^{-2}$
Input Symbols *K* = 50	(63,50)	None Found (NF)
100	(127,100)	NF
150	(255,150)	NF
200	(255,200)	NF

only searched RS code lengths in powers of two (corresponding to primitive RS codes) up to *N*=1023, equivalent to 10 bits/symbol. At 3% *BER*, RS codes with less than length 1023 output symbols are basically unable to deliver the fragmented image by the interactive latency bound of 100 ms. Basically, we are near the failure point for RS codes. This means that comparatively large percentages of FEC redundancy must to be added in order to reduce to an acceptably low level the expected number of retransmissions per fragment $E[A_{fragment}]$, which depends on the *BER*. Unfortunately, increasing the FEC redundancy inflates the bandwidth faster than retransmission delay can be reduced, so that the overall image transfer delay is never reduced below the interactive latency bound.

For example, consider in Table 7 what happens at K=100 input symbols. At N=127, we find that $E[A_{fragment}]$ is 217, while the *overhead factor* $\frac{I \cdot N}{BW \cdot K}$ only registers 50 ms of delay. However, because the FEC is insufficiently strong, the total image delay is on the order of tens of seconds due to the exorbitant number of retransmissions. Next, at N=255, the number of retransmissions has been dramatically lowered to one, but the bandwidth is expanded by a factor of ~2.5. Since our 20 kbit image will take 40 ms to transmit through a 500 kbit/s channel, then the overall latency will just exceed 100 ms due to the FEC bandwidth expansion. At N=511, there is no reduction in the number of retransmissions. Instead, a latency penalty of 204 ms is paid in terms of bandwidth expansions, so that it will be impossible to deliver the image with redundancy rates of 5 times or higher. The values in the column farthest to the right do not match the product of the middle two columns because of integer truncation effects not taken into account by the overhead factor.

Table 7. Contributions to delay in a T-I H-ARQ protocol employing RS (*N*,100) codes at 3% *BER*. Retransmissions and expanded bandwidth/FEC overhead both contribute to delay.

	Number of retransmissions $E[A_{fragment}]$	FEC Overhead factor (ms)	Total Image Delay (sec)
$N = 127$	217	50	15.8
255	1.00	102	0.128
511	1.00	204	0.227
1023	1.00	409	0.409

Some final qualifying comments should be made. In our analysis of RS codes for T-I H-ARQ protocols, we have not considered the effect of header overhead, which complicated our earlier study of optimal binary codes for T-I H-ARQ protocols in Section 3.5.1. Also, while we have concluded that RS codes are unable to meet the stringent 100 ms latency bound at 3% *BER*, Table 7 shows that a (255,100) RS code can lower the latency to 128 ms, which may be close enough for interactivity purposes. In this case, two to three times bandwidth expansion would be suffered. Finally, while the analytical results of this subsection assume a constant average *BER*, real-world wireless links will produce a time-varying *BER* which depends on the depth and frequency of fades. Though 1% *BER* is arguably accepted as a design point for digital cellular voice, we lacked the measurement data to determine whether 1-3% *BER* constituted a likely or unlikely occurrence over a wide variety of indoor and outdoor wireless links.

3.6 Summary and conclusions

In this chapter, we have derived an expression (Equation (3-2)) that determines the (N_{min},K) linear block code with the minimum redundancy which keeps the loss rate for a fragmented image below a desired threshold probability. For practical reasons, this expression assumed bounded-distance decoding of FEC linear block codes, instead of maximum-likelihood decoding. This expression was evaluated for three classes of linear block codes: optimal binary codes; binary BCH codes; and Reed-Solomon codes.

We found that moderately-sized optimal codes of input length K<100 bits required an output length N at least ~1.5 times greater than K, i.e. a minimum-redundancy (~150,100) code was required to deliver a fragmented image across a 10^{-2} *BER* channel at an image loss rate of less than about 1 in 1000. At $3x10^{-2}$ *BER*, the output length N was multiplied by a factor of about 2, i.e. a (~200,100) optimal code was required for adequate protection.

For binary BCH codes over a small range of K, we quantified how the imposition of BCH structure weakens the error correction ability compared to optimal binary codes. For example, at $K = 40$ input bits, the minimum-redundancy optimal binary code for 10^{-2} *BER* which adequately protects the fragmented image is a (71,40) code (~1.75 times bandwidth expansion), while the minimum-redundancy binary BCH code at the same *BER* and desired loss threshold is a (105,40) code (over 2.5 times bandwidth expansion).

For Reed-Solomon codes, we found similarly that at least a doubling of bandwidth was required at severe *BER's*. For example, at K = ~200 input symbols, the minimum-redundancy RS code for a channel *BER* of 3% was a (511,~200) code. This doubling or tripling of bandwidth was characteristic of all minimum-redundancy RS codes in the range of K<255 input symbols. In essence, RS codes are near their point of complete failure at 3% *BER*.

Finally, we have performed an analysis of a Type-I Hybrid ARQ protocol which combines reliable ARQ delivery with FEC. Our analysis of RS codes from this chapter is combined with our latency evaluation of ideal SRP protocols from the previous chapter. Using the loose lower bound on latency, we found that no RS codes of length N<1023 could be found at 3% *BER* for a variety of K which could lower the overall image transfer latency below the desired interactivity threshold of 100 ms. The FEC overhead required to reduce the delay due to retransmissions didn't fall fast enough to compensate for the increased delay due to bandwidth expansion.

In view of the failure of practical FEC codes at severe *BER*'s, we consider in the next chapter an alternative technique which combines error-tolerant encoding and decoding of image sources, forwarding of corrupt packet data by the underlying network, and unequal error protection (UEP-based FEC) to convey delay-sensitive multimedia quickly to the destination over very noisy channels.

4

Error-Tolerant Encoding and Decoding for Delay-Sensitive Wireless Multimedia

Our primary objective is the design of end-to-end communication systems for sufficiently rapid and sufficiently reliable transport of interactive visual multimedia across connections that include wireless links. One of the major subjective criteria that must be satisfied is that the visual information needs to be communicated quickly in order to give the human user viewing the image at the receiver a genuine feeling of interactivity.

In this chapter, we consider how the constraint of low latency delivery forces delay-sensitive multimedia applications to accept less than completely reliable packet delivery. Real-time video/audio conferencing applications accept unreliable packet delivery over the wired Internet in order to achieve low-latency packet delivery. We assert that the same principle of accepting increased channel distortion (i.e. image distortion due to channel impairments) for lower latency applies to interactive Web-based image browsing. Given the additional constraint of wireless access, we show how error-tolerant image coding, unequal error protection (UEP), source-cognizant channel decoding, and application-level decoding of corrupt error-resilient packet data are beneficial for delay-sensitive multimedia operating over wireless links both in terms of lower end-to-end distortion as well as lower perceptual delay. Our contention is that continuous interactivity with a noisy screen whose distortion varies with channel conditions is subjectively more preferable than intermittent interactivity depending on channel conditions with a constant quality error-free image for delay-sensitive applications.

4.1 Decreasing transport latency by tolerating packet loss

Distributed interactive multimedia applications and the network protocols that support them are designed to provide subjectively acceptable performance under the resource constraints posed by the network connection. The subjective performance of the joint application-protocol system is typically evaluated in terms of the perceived delivery time (either roundtrip response time or one-way delay) and the quality of the received data. For interactive image-based applications, the performance metrics are the perceived visual response time, i.e. how soon the image is displayed, and the perceived end-to-end distortion introduced by image quantization coding and transmission errors, i.e. how "good" does the image look. When the network connection includes a wireless link, then image coding algorithms and packet delivery protocols are faced with the challenge of meeting the subjective latency and distortion objectives within the constraints of limited wireless bandwidth, a high bit error rate, and time-varying noise behavior.

4.1.1 Reliable Internet packet delivery can exceed the tolerable latency bound

For delay-sensitive visually-based multimedia applications, reliable packet delivery can be too slow to meet interactive latency bounds, and unreliable packet delivery alone can introduce too much long-term distortion into the reconstructed image. Closed-loop retransmission-based protocols implement fully reliable end-to-end packet transport but invariably introduce latency over noisy and/or congested communication links. Distributed multimedia applications like interactive video and audio conferencing require roundtrip response times of less than about 100-200 ms [41][132]. Our measurements of average roundtrip packet latencies on the Internet[1] range from ~5 ms on our own LAN, to ~50 ms Berkeley-Southern California, to ~150 ms Berkeley-MIT, and ~300 ms Berkeley-London, where the observed delays will vary with distance (propagation delay), congestion (queueing delay in network switches), etc. For many connections even one retransmission will cause the overall delay to exceed the 100-200 ms delay bound. We observed peak packet loss rates of ~15% for the Berkeley-Southern California, Berkeley-MIT and Berkeley-London connections, which again vary with congestion conditions. Our measurements suggest that packet losses happen frequently enough on the statistically multiplexed Internet that fully reliable packet delivery cannot be depended upon to deliver a sufficiently large number of packets below the latency bounds required by conferencing applications. In addition, most practical reliable protocols like TCP preserve the order of delivery of packetized information, a property also known as stream-based delivery. If a particular packet is lost and must be retransmitted, then all subsequent packets that have already been correctly received will also be delayed until the lost packet is correctly delivered, thereby adding resequencing delay to all the packets already cached at the receiver.

4.1.2 Unreliable packet delivery for real-time video conferencing applications

Faced with these network constraints, designers of audio/video conferencing applications have chosen to forgo reliable packet delivery, and instead have opted to transmit real-time continuous media over unreliable packet delivery protocols [83]. Low-latency delivery is achieved at the cost of compromising on the reliability of the delivered data. Impairments into the transported image caused by uncontrolled channel-related distortion, e.g. packet loss, are tolerated. Error concealment algorithms are used to mitigate the subjective effect of lost video packets [39][70].

1. Measurements taken using the UNIX "ping" utility, sending one 6400 byte packet every second, for 100 seconds.

Developers of interactive Web-based image browsing have also exploited this trade-off between distortion and latency. Such delay-sensitive applications are also willing to compromise on the amount of distortion introduced into an image in order to obtain faster perceptual response time. For example, many Web browsers now allow text and images in a Web page to be only partially loaded before the user can jump to a new Web page. In principle, the user is willing to accept higher distortion in the reconstructed Web page in order to maintain a higher degree of interactivity with the displayed information. Some older versions of Web browsers (e.g. Mosaic [56]) did not have this ability to interrupt the loading of a graphics-intensive Web page. Frustrated users were either forced to wait for reliable delivery of the entire possibly image-laden Web page, or they could use an option that *a priori* suppressed delivery of any image. Neither option was particularly attractive.

The same subjective trade-off of interactivity and distortion has been applied to the images that are embedded within a Web page. Web-based image browsers have employed progressive source coding methods, also called progressive image transmission (PIT) algorithms [138], in order to lower the perceptual delay seen by the end user, at the cost of tolerating greater initial distortion. Both partial loading of Web pages and progressive image coding represent examples in which distortion is introduced by some form of lossy quantization of the original source data in order to improve the perceived interactivity.

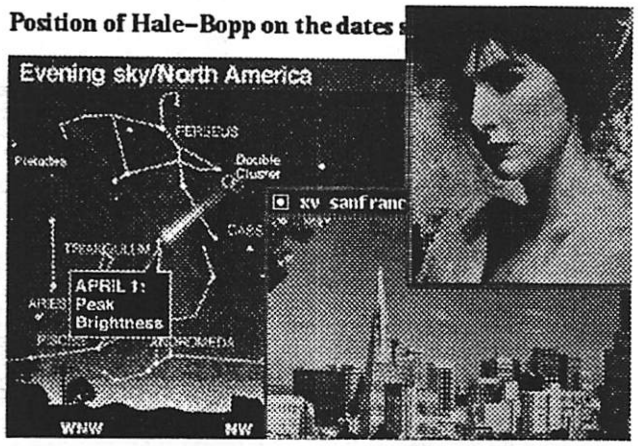
4.1.3 Unreliable packet delivery for interactive Web-based image browsing

While progressive image coding can help reduce the perceived delay, current versions of Web browsers implement this progressivity on top of a reliable packet delivery protocol [115]. As noted earlier, real-time video conferencing applications have already come to the conclusion that reliable packet delivery cannot dependably meet their subjective delay constraints over multi-hop Internet connections and have therefore opted for unreliable packet delivery. Since interactive bursty-media applications like image browsing share essentially the same 100-200 ms delay constraints as video conferencing applications, and generate the same volume of traffic over the duration of a single image burst, then *it is our contention that a distributed delay-sensitive image browsing application can only hope to provide consistent interactivity via an unreliable packet delivery protocol over an end-to-end Internet connection*[1]. Previous work has also suggested that image response time can be improved by tolerating Internet packet loss [137]. Over noisy access wireless links, it is even more uncertain whether reliable packet delivery can hope to provide consistent interactivity. Again unreliable packet delivery appears to be the most reasonable option for delay-sensitive media.

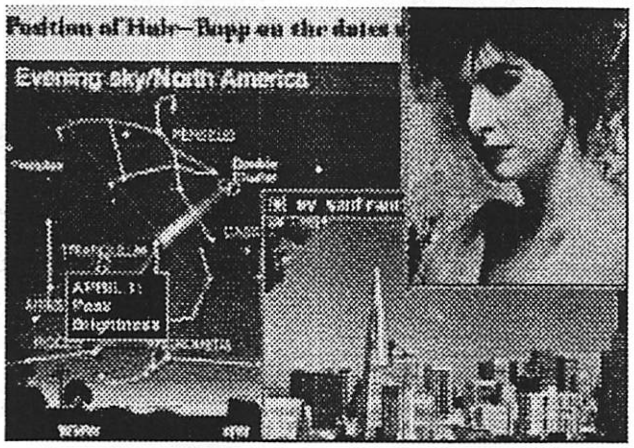
Properly designed image encoding and packetization algorithms combined with error-concealment algorithms should permit interactive image browsers to tolerate the effect of channel distortion introduced by an unreliable packet protocol. In Figure 4.1, we demonstrate how an intelligent packetization algorithm can help mitigate the effect of lost packets. First we explain the encoding process, before describing the packetization algorithm. An 8 bits/pixel grayscale image is divided into 8x8 blocks. Each block is transformed into the frequency domain using a Discrete Cosine Transform (DCT). The resulting DCT frequency coefficients $F(u, v)$ are then quantized to the number of bits shown in the quantization matrix in Figure 4.1. The quantized DCT image is logically transmitted across a channel which introduces packet losses. An inverse DCT transform is applied to the quantized frequency coefficients at the receiver in order to reconstruct the quantized DCT image.

The middle image shows the effect of lossy quantization on DCT coefficients (no packet losses). The quantization matrix removes high frequency components (also called zonal coding), which appears to be reasonably tolerable for natural images (Enya, San

1. It should be clear that unreliable packet delivery is intended for transport of images and not text.

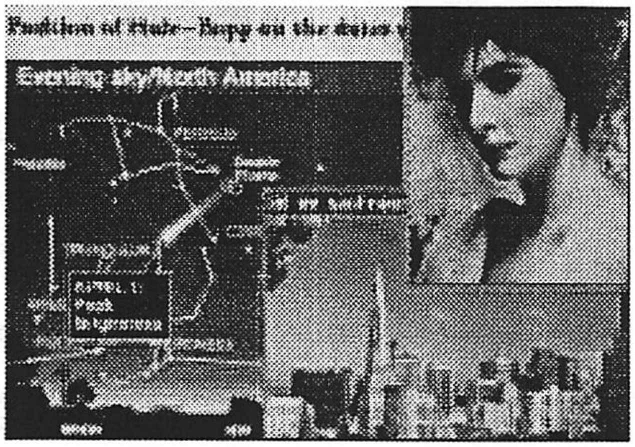


Initial 8 bits/pixel grayscale image



Quantized DCT image @ 0.75 bpp. Each DCT frequency coefficient is quantized to the number of bits shown. (DC coefficient is in upper left corner)

$$\begin{bmatrix} 8&5&4&4&3&2&0&0\\ 5&3&2&2&0&0&0&0\\ 4&2&0&0&0&0&0&0\\ 2&0&0&0&0&0&0&0\\ 2&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$



Quantized image after packet loss. Lost coefficients are marked with 1's.

$$\begin{bmatrix} 0&0&0&1&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0 \end{bmatrix}$$

Figure 4.1 The effects of quantization and packet loss on a DCT-coded grayscale image are shown. The original 8 bits/pixel image (top) is quantized to 0.75 bits/pixel (middle image) using the quantization matrix for frequency coefficients on the right. The bottom image suffers from packet loss of some of the quantized DCT coefficients. Frequency coefficients from the same 8x8 block are placed in different packets. The coefficients corresponding to the 1's in the drop matrix are assumed to be the only ones lost during transmission. Packet loss of these few AC coefficients (reconstructed at their average value) does not appear to be too objectionable for DCT-coded natural images. Both natural and embedded text/graphic images are shown.

Francisco) but does appear to adversely affect the quality of embedded text/graphics images with high frequency edges (Hale-Bopp). The lowest frequency DC coefficient was quantized uniformly. All other AC frequency coefficients were non-uniformly quantized according to the following rule. We calculate each coefficient's mean $\mu(u, v)$ and average absolute deviation from the mean $AAD(u, v) = \sum_{allblocks} \frac{1}{N} \cdot |F(u, v) - \mu(u, v)|$, where N is the total number of blocks in the image. Two quantization levels were reserved for extreme variations of AC coefficients at $\pm 5 \cdot AAD(u, v)$ while the remaining quantization levels were uniformly distributed between $[2.5 \cdot AAD(u, v), -2.5 \cdot AAD(u, v)]$. This simple non-uniform quantization scheme improved the quality of the reconstructed image over uniform quantization of both AC and DC coefficients. This is because non-uniform quantization is based on the input signal's statistics and generally produces a smaller error variance in the quantized signal than uniform quantization [60]. More elaborate optimal Lloyd-Max non-uniform quantization was not attempted [102]. The matrix does not contain any 1-bit quantization components because we found that a noticeable amount of graininess due to quantization noise was introduced throughout the image. The fixed quantization matrix results in 0.75 bits/pixel, or about 11:1 compression.

If all DCT frequency coefficients from the same block are placed in the same packet, then a lost packet will cause "holes" to appear in the reconstructed image for each block that was contained in the lost packet. Our subjective experimentation has shown that spreading the effect of channel noise across an image is more acceptable than concentrating the distortion into "holes" in the displayed image. A more intelligent packetization algorithm would distribute the frequency coefficients from the same block across different packets. Similar approaches have been developed for packet video [131] and for packetized still image delivery [137]. A single packet loss will cause some coefficients from many blocks to be lost, but each block will hopefully have enough surviving coefficients to reasonably estimate the affected image block. We can emulate the effect of packet loss on such a system by dropping sets of coefficients. Our error concealment algorithm consisted of replacing the lost frequency coefficients with a pre-computed average value based on image statistics that differs for each coefficient. However, analysis of the mean and average absolute deviation of each frequency coefficient showed that all the AC coefficients had a mean value very close to zero. The error concealment strategy can be simplified and made image-independent by zeroing lost AC coefficients.

The lower image in Figure 4.1 illustrates how dropping the four coefficients indicated in the dropping matrix affects the reconstructed image. As expected, the results are generally blurrier due to the removal of even more high frequency information. However, the natural images are largely intact and decipherable. Figure 4.1 demonstrates that packet loss can indeed be tolerated when DCT coefficients are intelligently interleaved across different packets, though the method is only effective when the lost components are higher frequency AC coefficients and not the lowest frequency DC coefficient.

Another method of compensating for packet loss is to code the image in such a way that geographical information is naturally spread out into each packet due to the coding technique. For example, subband image coding separates an image into subsampled frequency subbands, and is usually applied more than once to generate multiple layers of multiresolution encoded information [102]. Each frequency subband in each resolution layer naturally contains geographical elements from every region of the image. Normal sequential packetization will naturally place these subbands into different packets. Loss of a single packet that contains one or more subbands will create an effect that resembles Figure 4.1 in the sense that no "holes" will appear in the reconstructed image, only a general blurring effect as high frequency subbands are lost. Subband coding with sequential packetization is useful for dealing with packet loss that corresponds to high frequency subbands, but not for the lower frequency "LL" subband. Hence, packetization algorithms have been proposed that scatter components of the "LL" subband across different packets

[76]. Error concealment algorithms can then be applied to further improve the error resilience of subband encoding and intelligent packetization.

Finally, even though subband coding and DCT frequency coefficient interleaving try to avoid the creation of "holes" in the reconstructed image, some elaborate error concealment algorithms deal directly with this problem of lost image blocks. Advanced interpolation operations that use surrounding blocks are capable of reconstructing a good approximation of both the DC component and the high frequency edges in lost blocks [157].

4.1.4 Erasure codes

We describe briefly an FEC technique called erasure coding that has been proposed to combat Internet packet loss. Erasure coding is a form of FEC in which B blocks of user data (each block of length W words) are coded into N packets (each packet of length W words), where $B \leq N$. The erasure code has the property that the original B blocks of user data can be recovered from any B coded packets. Therefore, up to N-B packets can be lost, or erased, without affecting recovery of the original B blocks of data.

While erasure coding has been proposed to cope with packet loss on the Internet [2], it is ineffective in dealing with packet corruption over a noisy wireless link. Many of the erasure-coded packets will likely suffer some bit corruption when the BER is high. The error correction ability of erasure codes is focused on locating and correcting erasures. Consequently, much of the ability to correct corrupt bits is sacrificed. Corrupt packets that have been strictly erasure-coded cannot be decoded at the receiver due to the assumption that received packets will be error-free. The corrupt packets would either need an additional layer of FEC to protect against bit corruption, or perhaps a very long Reed-Solomon code which could simultaneously handle both error correction and erasure detection and correction. In the absence of such error correction, erasure-coded packets would be highly ineffective over an Internet connection with a wireless access link.

4.2 On the traditional separation of compression algorithm design from network/error protection design

In this section, we consider the traditional approach of separating the design of compression algorithms (source coding) from the design of the network's protocols and other error protection functions (channel coding). This approach is also called independent source and channel coding. The background provided here will help us compare joint source/channel coding to independent source and channel coding over wireless links in the next section.

The rationale behind the principle of separate source and channel coding is derived from *Shannon's separation theorem,* also called the *channel coding theorem* [27]. The theorem states that transmission of the source (e.g. image) through a noisy channel (e.g. wireless link) can be made arbitrarily close to reliable (arbitrarily close to zero probability of transmission error) as long as the source's information rate/entropy is less than the information-theoretic channel capacity. The source coder's primary responsibility is to compress the input data below the channel's capacity, and the channel coder's primary responsibility is to adequately protect the compressed data up to the arbitrary limit of channel distortion tolerated by the source. Thus, the source coder's compression algorithm can be designed independently of all characteristics of the downstream channel except its capacity. Assuming that the post-compression entropy is less than the channel capacity, then the channel coder can design its FEC and/or ARQ protocols independent of any information about the upstream source except its maximum tolerance for channel distortion. The separated source and channel system is depicted in Figure 4.2. One consequence

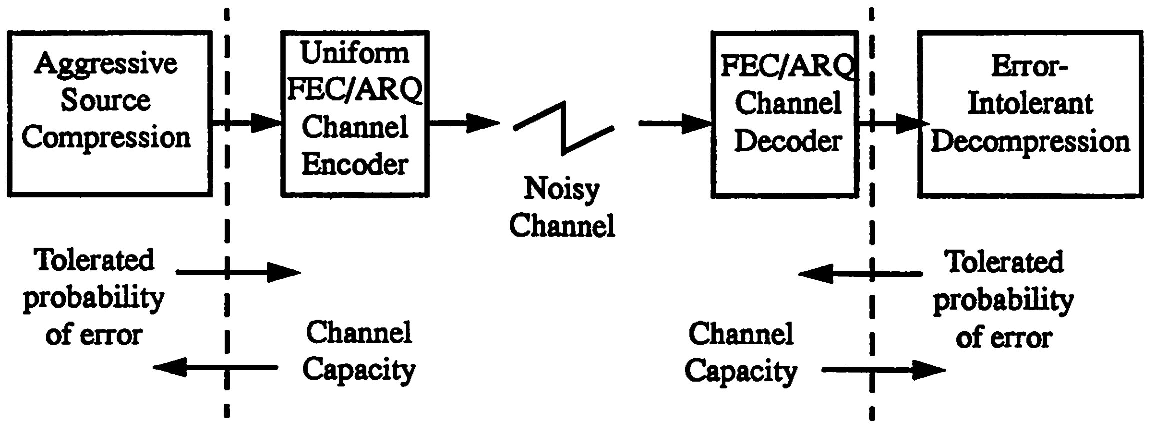

Aggressive Source Compression
Uniform FEC/ARQ Channel Encoder
Noisy Channel
FEC/ARQ Channel Decoder
Error-Intolerant Decompression
Tolerated probability of error
Channel Capacity
Channel Capacity
Tolerated probability of error

Figure 4.2 The four components of an independent source and channel coding system: an aggressive source compression algorithm; a uniform FEC/ARQ channel encoder; an FEC/ARQ channel decoder; and source decompression. If reliable transmission is the only concern, then ideally the only channel information used in the source coder's design is the channel capacity. The only source information used in the channel coder's design is the tolerated probability of transmission error.

of this separability is that there is no need to design error tolerance into the compression algorithm, since the channel coder can already guarantee an arbitrarily small probability of transmission error using FEC and/or ARQ if the source entropy is less than the capacity.

While the separation theorem addresses the issues of information rate (source compression followed by channel coding redundancy) and distortion (source coding introduces controlled distortion while channel coding protects against uncontrolled distortion), it does not deal with the third dimension of delay. In order to guarantee an arbitrarily small probability of transmission error, the separation theorem basically assumes that the source coder/decoder and channel coder/decoder can all be arbitrarily complex and that all can take an arbitrarily long time to implement compression and error protection respectively. Practical compression algorithms and network FEC/ARQ mechanisms are designed independently of one another under the separation principle, but typically also have to operate under delay constraints imposed by the user. The approach taken to minimize the overall delay in such de facto separated systems is to have the source coder and channel coder independently minimize their individual contributions to delay.

From the source coder's perspective, delay is minimized by minimizing the bit rate of the compressed data. Lossy quantization and lossless statistical coding are applied in tandem to reduce the information rate of the coded source down to the minimum rate which is still subjectively tolerable. For many images, the subjectively tolerable distortion limit is taken as the Just-Noticeable Distortion (JND) threshold [20], which is the threshold below which reconstruction errors are rendered imperceptible. The theoretically minimum bit rate corresponding to a given distortion limit is called the rate-distortion limit [47][102]. In practical systems, minimizing the bit rate subject to the JND threshold helps to minimize the transmission delay and also hopefully compresses the source's bit rate below the channel's capacity so that the separation theorem holds. A consequence of minimizing the bit rate at or near the rate-distortion limit corresponding to the JND threshold is that corrupt information tends not to be very useful at the receiver.

From the channel coder's perspective, delay is minimized by increasing the operating speed of network switches, increasing the bandwidth of the network trunk lines (e.g. optical fiber), expanding the buffer sizes in network switches (which reduces packet loss and therefore retransmission-based latency), improving signal processing techniques to make each link in the connection appear less noisy, and designing stronger realizable FEC codes that operate at minimal redundancy.

The independent source and channel coding approach has been implicitly extended to wireless access to the Web [77]. In this approach, images are heavily compressed for delivery over the wireless link in order to reduce the transport latency over a bandlimited wireless link. The implicit assumption is that heavy FEC will keep the channel relatively clean so that heavily compressed error-sensitive data are still useful at the receiver. In the next section, we show that heavy compression/heavy FEC does not necessarily equate

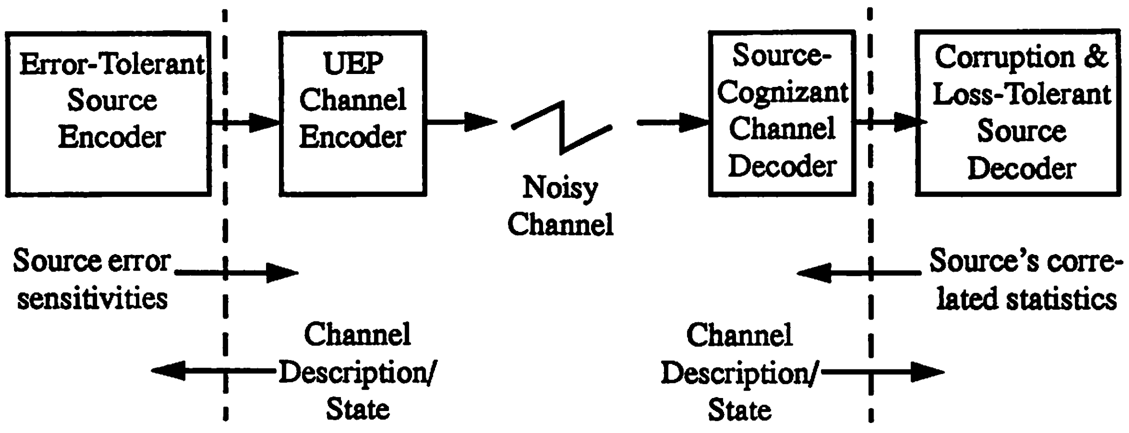

Error-Tolerant Source Encoder
UEP Channel Encoder
Noisy Channel
Source-Cognizant Channel Decoder
Corruption & Loss-Tolerant Source Decoder
Source error sensitivities
Source's correlated statistics
Channel Description/ State
Channel Description/ State

Figure 4.3 The four components of a joint source/channel coding (JSCC) system: an error-tolerant source encoder; an unequal error protection (UEP) channel encoder; a source-cognizant channel decoder; and an application-level source decoder that tolerates packet loss and processes corrupt information. The source shares information about its statistics and error sensitivities to the channel encoder and decoder. The channel shares information about itself to the source encoder and decoder. In the simplified case, a static description of the channel is sufficient for designing the source coder and error concealment algorithms at the source decoder. Compare to Figure 4.2

with low latency delivery over a heavy *BER* wireless link. Constraints on complexity and the demand for speedy operation limit the power of FEC, causing FEC to fail and resulting in a large image transfer delay at a certain *BER*. For the same *BER*, error-tolerant image coding/decoding can still deliver an image successfully with smaller delay since retransmissions are unnecessary.

4.3 Joint source/channel coding

In this section, we investigate the motivation for each of the four components of joint source/channel coding (JSCC) shown in Figure 4.3. Given complexity and delay constraints on the design of FEC channel coding as well as compression algorithms, and given the often severe raw *BER* suffered over nonstationary wireless links, then error-tolerant coding combined with loss-tolerant and corruption-tolerant decoding of noisy payload information can be shown to produce images with lower end-to-end distortion than the classic approach of aggressive image compression, aggressive FEC channel coding, and decoding of only error-free information. We demonstrate that robust image coding can achieve a reasonable degree of compression and yet can be tolerant to high *BER*'s in the absence of any FEC on the compressed data. Therefore, corrupt packets bearing error-resilient data are inherently useful and need not be thrown away. Over wireless links, the combination of robust image coding and application-level decoding of corrupt packets can help reduce the end-to-end distortion as well as reduce the perceptual delay

4.3.1 Introduction

The separation theorem discussed in Section 4.2 was derived under the following three assumptions:

- stationary memoryless channels
- unconstrained complexity of the compression/decompression algorithms and FEC/ARQ error protection techniques
- unconstrained delay in the source and channel coders and decoders

If any of these assumptions underlying the separation theorem are violated, then it becomes questionable whether the best design policy is completely separate source and channel coding. Recent work has suggested that care should be exercised before applying the separation theorem to nonstationary channels [142]. Wireless links that suffer time-varying shadowing and time-varying co-channel interference fit into this category of non-stationary channels. Moreover, since our application of interest is interactive image browsing over time-varying channels with constraints on the complexity and delay of compression and network FEC/ARQ, then each one of the three assumptions of the separation theorem is violated.

In addition, the appropriate quality criterion in our case is a subjective dual function of delay and distortion. For example, progressive image transmission is a technique which exploits the human user's subjective tolerance for significant distortion in an initial version of an image provided that the end user knows that the image will improve eventually in quality over time. Progressively reliable packet delivery discussed in Chapter 5 is a similar example which introduces significant channel-induced distortion into an initial image, and follows that up with a cleaner image at some later time. Such subjective tolerances and their associated metrics are not addressed by the separation theorem.

An approach called *joint source/channel coding*(JSCC) can be shown to outperform separate source and channel coding when one or more of the theorem's assumptions are not satisfied. The JSCC channel coder is designed with some knowledge of the source's statistics in mind, and conversely the JSCC source coder is designed with some awareness of the channel's error statistics. In comparison, the only knowledge shared between independently designed source and channel coders are the channel capacity and tolerable channel distortion limit.

The four major components of a JSCC system are pictured in Figure 4.3. A robust source encoder generates error-tolerant data, which is then passed to a channel coder which implements unequal error protection (UEP), protecting the more sensitive source bits with stronger error protection. After corruption from the transmission channel, the FEC channel decoder attempts to reconstruct a best estimate of the noisy data using in-depth knowledge of the source statistics. Finally, a corruption-tolerant and loss-tolerant source decoder operates on the possibly noisy decoded bits. The information shared between the source and channel is also shown, and will be described later as we discuss each of the components. Note that there is no constraint in Figure 4.3 on the layer at which the channel encoders and decoders should be placed. This permits data-link and physical layer source-cognizant channel encoding and decoding that can provide some form of lower level variable quality-of-service to the application's data. In addition, end-to-end FEC/ARQ can also be practiced simultaneously if desired.

In this section, we discuss how the practical constraints of complexity, delay and non-stationary wireless channels motivate the development of each of the four components of JSCC. We contend that a JSCC system that practices robust image coding and forwards corrupt error-tolerant information to the receiver is a better means for achieving low-latency and sufficiently reliable delivery of visual multimedia across time-varying wireless channels than the traditional separated approach of aggressive compression and aggressive channel coding.

4.3.2 Unequal error protection for imperfectly compressed data

Standard compression algorithms typically are unable to fully compress images, speech, and video down to the rate-distortion limit, due to constraints on the complexity and speed of practical software and/or hardware encoders and decoders, and also due to imperfect knowledge of the input image/speech/video statistics. This means that the compressed data still contains some statistical correlation after coding. For example, the JPEG image compression standard still leaves some residual redundancy after aggressive lossy and lossless compression. Differential encoding of DC DCT coefficients does not completely remove correlation among neighboring DC coefficients [113]. There is also residual correlation between DC and AC DCT coefficients in the same block [135].

Also, most motion-compensated DCT video coding standards like MPEG-2 and H.261 leave residual redundancy as a compromise for practical implementation. For example, computing DCT's on an 8x8 block basis is done for the sake of implementation simplicity. However, ideally a DCT should be computed on the entire image to achieve the greatest compression, since this will most comprehensively exploit spatial redundancy and pack energy in the lower frequency coefficients. In addition, the motion compensation used to generate differential frames typically only uses a single frame to predict the current frame. Ideally, we would like the entire sequence of video frames, or at least two or three frames, in order to generate a more accurate motion-compensated prediction. Again, complexity constraints force the video coder/decoder to compromise, resulting in residual redundancy. Even if there were no complexity constraints, there is still a delay constraint for real-time applications. In real-time video conferencing, the motion-compensated predictor cannot queue up more than a few frames, which is about 60-90 ms worth of delay, before a coded frame must be transmitted in order to meet the conferencing application's latency bound of a couple hundred milliseconds. This delay constraint forces the compression algorithm to compromise and leave residual redundancy in the coded video.

The residual source redundancy left by imperfect compression means that some codewords are more sensitive to channel errors than other codewords. In this distortion sense, some bits are perceptually more important than others. *All bits are not equal.* A natural approach to channel coding is to place stronger error protection on the perceptually more important source bits and weaker error protection on the perceptually less important source bits. This approach to channel coding is called *unequal error protection* (UEP). Since the channel coder has explicit knowledge of the source's different error sensitivities, then clearly UEP is a form of JSCC.

Recent studies have compared the performance of UEP linear block codes to more traditional equal error protection (EEP) block codes when both are applied to compressed images with residual redundancy transmitted across noisy channels [38][126]. UEP and EEP block codes with approximately the same redundancy rate are applied to the same coded image. The error-protected images are then corrupted at the same *BER*. Both objective measures (peak signal-to-noise (PSNR) ratios) and subjective measures (reconstructed image quality) show that the JSCC approach of UEP outperforms the EEP channel coding approach that is designed independently of the source's statistics.

While digital block and convolutional codes offer one way to implement UEP, another way to support variable reliability is to adjust the analog power of the signal according to the error sensitivity of the data. Several authors have shown that a QAM signal constellation that is designed with knowledge of the distribution of a vector quantized Markov source (i.e. a source with residual redundancy) considerably lowers end-to-end distortion compared to a standard system that employs independently designed VQ and QAM modulation signal sets [80][140]. Again, JSCC via UEP outperforms independent source and EEP channel coding for practically compressed sources.

UEP has been implemented or proposed as a channel coding solution for several real-world systems. Working examples of UEP can be found in both the GSM and IS-54 digital cellular systems. The speech coder in both systems generates two classes of audio data that are error-protected differently. In both systems, the perceptually most important audio bits are protected with a rate $\frac{1}{2}$ convolutional error correction code, while the perceptually least important bits are left completely unprotected [85]. Another example of UEP can be found in a proposal for the design of broadcast digital HDTV [104]. Hierarchical modulation is proposed as a means of providing UEP for multiresolution broadcast of digital HDTV, so that users far away from a broadcaster can still obtain a coarse version of the HDTV signal while users nearer to the broadcaster can obtain a complete "coarse+fine" HDTV signal.

4.3.3 Source-cognizant channel decoding for imperfectly compressed data

In addition to JSCC UEP *encoders*, JSCC channel *decoders* have also been designed to exploit a source's residual redundancy. Recent studies have shown that sophisticated soft-decision Viterbi decoders that exploit knowledge of the source's correlated behavior can substantially improve the quality of a reconstructed image compared to traditional Viterbi decoders that ignore all source characteristics [50][96][114]. Sources with residual redundancy exhibit memory or correlated behavior, much like the output of convolutional coders with memory. Hence, the path metric of Viterbi decoding, normally optimized for decoding sequences with convolutionally encoded memory, can be modified to incorporate additional knowledge of source sequences with memory. Certain sequences of source codewords may be closely correlated due to spatial or temporal relationships, so that certain paths through the trellis will be more likely than others once the source statistics are taken into account. Again, the conclusion is that JSCC outperforms separately designed source and channel coders and decoders when there is residual redundancy to exploit at the decoder.

4.3.4 Error-tolerant image coding over noisy channels with constrained-complexity FEC

In the previous section, residual redundancy after compression was viewed as an unavoidable necessity dictated by practical constraints on complexity and delay. Recent work has added the constraint of very noisy channels to the constraints on delay and complexity. Xu, Hagenauer, and Hollmann show that it is in fact desirable to purposely leave residual redundancy in a coded image over very noisy channels rather than aggressively compress the data to remove all possible redundancy [155]. The authors compare a baseline JSCC approach to a second approach which practices further compression and FEC on the compressed data. In the first JSCC approach, a quantized DCT encoder leaves some redundancy in the AC coefficients. A UEP channel encoder is applied to the quantized data, and specialized source-cognizant Viterbi decoding helps reconstruct the corrupted image data. In the second more traditional approach, the AC coefficients in the quantized

DCT data are further compressed using Lempel-Ziv variable-length coding, and then rate $\frac{1}{4}$ convolutional error protection is applied. At the decoder, standard Viterbi decoding helps reconstruct the noisy image data. In order to compare the two techniques, the overall sum of the source and channel coding rates in bits/symbol is constrained to be constant across both the JSCC and independent source and channel coding approaches. The primary difference between the two techniques is that the JSCC approach devotes a higher proportion of bits to source coding than channel coding. The reconstructed images show that *the traditional approach of aggressive compression and aggressive FEC suffers considerably more channel distortion over very noisy channels than the JSCC approach of moderately compressed image coding, UEP, and source-cognizant channel decoding given the same overall encoding rate.*

The intuition behind this result is that the constraint on the overall encoding rate limits the aggressiveness of FEC, causing the channel coding to fail frequently at noise levels at which error-tolerant coding is still comparatively beneficial. For example, the failure point of practical linear block codes (e.g. binary BCH and Reed-Solomon codes) limited to lengths less than about a few thousand bits long and code rates greater than about $\frac{1}{4}$ is approximately $5\text{x}10^{-2}$ *BER*. Around this range of *BER*, the "waterfall" curves for linear block codes show that the post-error correction *BER* is little better than the original channel *BER* [87] (and sometimes worse!). For example, the failure point for the (63,47) RS code employed by the Cellular Digital Packet Data (CDPD) standard's MAC layer [122] is approximately $2\text{x}10^{-2}$ *BER*. We believe the same reasoning can be extended to practical convolutional codes which are limited by small constraint lengths (5 for GSM, 6 for IS-54, 9 for IS-95 CDMA) and code rates exceeding $\frac{1}{2}$ on the forward link in all conventional digital cellular systems [105]. For block codes, once the *BER* nears $5\text{x}10^{-2}$, then the compressed data will essentially become unusable. On the other hand, for channel error rates of this approximate severity, error-tolerant coding can ideally still exploit residual source redundancy, i.e. 95% of bits are viewed as ideally still being useful at 5×10^{-2} *BER*, and 99% of bits at 10^{-2} *BER*. Consequently, for multimedia applications that desire continuous operation over a wide range of *BER*'s and for practical networks that have limited FEC capability to combat relatively severe fading on wireless channels, the approach of error-tolerant image coding integrated with JSCC channel coding and decoding offers the lowest-distortion solution for encoding and decoding multimedia over wireless links.

Finally, we observe that there are two ways to achieve a reduction in bit rate: lossy compression and lossless statistical compression. Lossy image coding achieves its compression through dropping of perceptually unimportant information and through fixed-length scalar and vector quantization of the remaining information. This approach limits error propagation. Lossless compression removes statistical correlation in the data and is synonymous with variable-length arithmetic or Huffman coding. This approach will increase the error sensitivity of the data. Error-tolerant coding typically will concentrate on lossy techniques and limit the contribution from lossless compression. For example, the general approach to error-tolerant DCT or subband coding will drop perceptually unimportant higher frequency coefficients or subbands, and then apply fixed-length vector quantization to the remaining lower frequency coefficients or subbands. In the next section, we demonstrate an example of error-tolerant DCT compression.

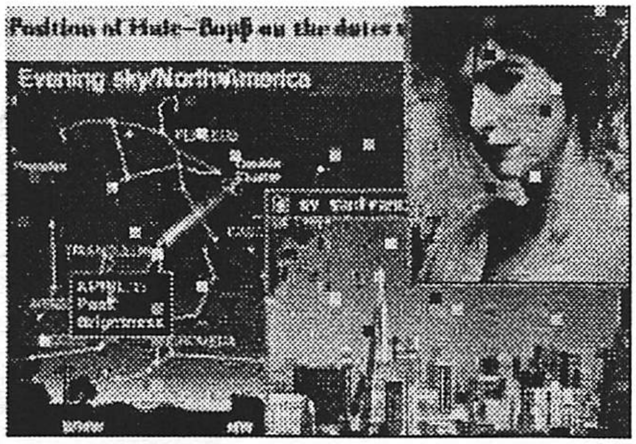
4.3.5 Application-level decoding of corrupt image data for noisy channels

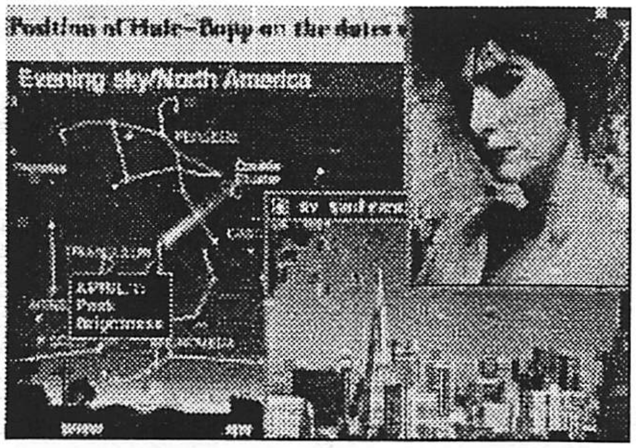
The previous section demonstrated the desirability of error-tolerant image coding when the channel *BER* is severe and practical FEC is constrained in its error correction ability. For a source that has been encoded in an error-tolerant manner, the error-resilience of the coded data will support decoding of corrupt packet payloads at the receiver. In a noisy packet, only a fraction of the codewords are typically in error while the majority of codewords are free of bit errors. The error-resilience of the coded data will localize any visual artifacts caused by decoding of corrupt codewords. The robustness of the data prevents errors from propagating. As a result, error-tolerant source decoders can extract a great deal of useful information, based on the error-free segments of the payload, from a noisy packet.

For example, a compression scheme that generates fixed-length codewords after quantization would allow an error-tolerant decoder to make use of nearly all of the error-free codewords in a noisy payload. Consider the image from Section 4.1 that is transformed by a DCT and non-uniformly quantized using the fixed bit allocation matrix shown in Figure 4.1. This is an example of robust image compression, because some residual redundancy is left within the coded image (e.g. neighboring DC coefficients are not differentially encoded). We map the non-uniform quantization levels onto binary codewords using a natural binary code (NBC)[1] [60]. For example, if two bits are devoted to a frequency coefficient, then the lowest magnitude level is assigned to '00' and the highest to '11'. All coded bits are corrupted at 10^{-2} *BER*, a commonly referred to design point for cellular voice [50][149]. No FEC is applied. At the receiver, the error-tolerant decoding procedure uses every received fixed-length codeword, whether corrupt or not, in the inverse DCT.

The resulting noisy reconstruction is shown as the top image in Figure 4.4. The major visible artifacts in the image correspond to corruption of the DC coefficients (not to lost packets). Despite the heavy *BER*, lack of FEC, and error sensitivity due to compression down to 0.75 bpp, our error-resilient image coding technique permits the decoder to reconstruct a reasonable approximation to the transmitted image. Errors are localized due to the fixed-length coefficients, which limit the ability of errors to propagate throughout the entire image. For example, an error in one DC coefficient does not propagate to cause

1. Alternative mappings that would produce less distortion are the Folded Binary Code (FBC) and Gray code [60].


0.75 bpp quantized DCT image corrupted @ *BER* 10^{-2}. Corruption of DC coefficients is responsible for the worst artifacts.


Corrupted image @ *BER* 10^{-2} with simple error concealment. Sudden isolated jumps in DC coefficients are assumed to be in error and are replaced by a local DC average.

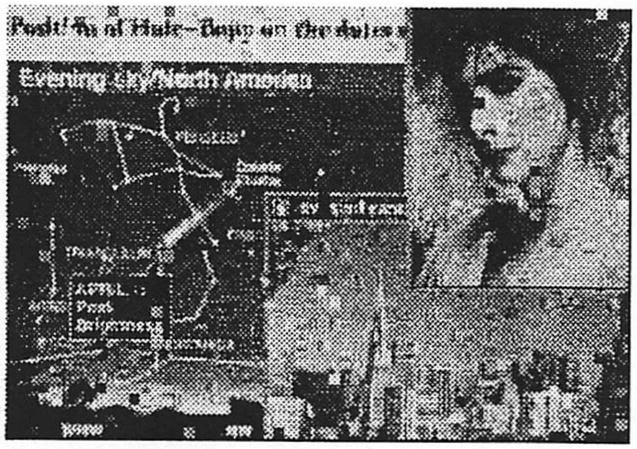

Corrupted image @ *BER* 3×10^{-2} with simple DC error concealment.

Figure 4.4 A quantized DCT image is *corrupted* at *BER* 10^{-2} (top image). The image is encoded by a DCT followed by the fixed quantization matrix shown in Figure 4.1 and the frequency coefficients are then corrupted. The most objectionable artifacts are due to corruption of DC coefficients. A simple error concealment algorithm based on DC interpolation can eliminate most of the worst artifacts. If the difference between a DC coefficient and each of its eight neighbors exceeds a threshold, then the DC coefficient is replaced with an average, thereby removing completely isolated jumps in DC values.

errors in other DC or AC coefficients. This localization of errors permits the application-level error-tolerant decoder to extract all of the error-free codewords. Consequently, a reasonable facsimile of the transmitted quantized DCT image can be reconstructed despite the heavy *BER*.

If corrupt packets bearing error-resilient data were discarded by the underlying network, then all of the error-free codewords in each noisy packet would be thrown away. Typically, the proportion of usable error-free codewords will far exceed the fraction of corrupt codewords in a noisy packet. For example, if each DCT coefficient were quantized at 8 bits/coefficient, then the overall coefficient error rate would be about 8% given 1% channel *BER*. If each coefficient were quantized according to the bit-allocation matrix of Figure 4.1, then the overall coefficient error rate is closer to 4% given 1% channel *BER*. In either case, well over 90% of the AC and DC coefficients in each noisy packet are still usable on average by the decoder at a 1% *BER*. Channel distortion will be needlessly introduced into the reconstructed image if any coefficients, especially DC coefficients, are unnecessarily thrown away by the underlying wireless network. Hence, we contend that corrupt packets identified as containing error-resilient data should be forwarded by the underlying wireless network to the application for error-tolerant decoding.

A survey of the literature on robust image coding confirms that error-resilient images corrupted at 10^{-2} *BER* are still presentable despite the lack of any FEC error protection. This has been shown for a variety of DCT [139], subband [18][107], and VQ [58] source coding techniques that achieve 0.5-1.0 bpp compression. One property common to each of these error-resilient compression algorithms is that source-based redundancy is distributed throughout the coded data. An alternative approach to error-resilient coding is to concentrate source-based redundancy in a few resynchronizing codewords whose job is to stop error propagation. For example, JPEG image coding first compresses the data, and then allows the insertion of uniquely identifiable byte-aligned reset markers between groups of blocks of entropy-coded DCT coefficients in order to stop error propagation [95]. Other forms of synchronizing codewords have been described for standards like Fax, H.261 video and MPEG video [74].

While the corrupted image in Figure 4.4 conveys a significant amount of information despite the visual artifacts, its presentation can be improved significantly by applying error concealment. Our knowledge of image statistics, specifically spatial correlation, can be combined with the noisy received data to detect and mask erroneous image artifacts. For example, we know that neighboring DC coefficients are typically highly correlated. Sudden isolated changes in the DC coefficient are statistically unlikely, and can be interpreted as being erroneous.

A simple error concealment mechanism resembling median filtering was applied to the DC coefficients only. If the difference between a DC coefficient and each of its eight closest DC neighbors exceeds a given threshold, then we replace the DC coefficient with an average of the eight neighboring blocks' DC coefficients. We applied this error concealment filter to the corrupted image and the result is shown as the middle image in Figure 4.4. Our elementary filtering removes most of the worst artifacts from the image, leaving only a few DC artifacts and various AC-related artifacts. More advanced concealment algorithms could be applied to detect and correct irregularities in AC coefficients, to further reduce the effect of channel distortion. Error concealment algorithms have also been applied to heavily compressed JPEG images, and, in conjunction with resynchronizing markers, have been shown to significantly improve the quality of the reconstructed image in the presence of channel noise [74][148]. We observe that throwing away corrupt packet payloads will not only discard decodable information, but will also hinder error concealment by reducing the amount of error-free information available at the receiver for masking operations like spatial interpolation.

Several precedents exist for processing of corrupt information in today's digital wireless systems. As we noted in Section 4.3.2, both the IS-54 and GSM digital cellular TDMA standards employ UEP on compressed audio data [36][85]. The perceptually more important speech bits are protected via a rate $\frac{1}{2}$ convolutional coder, while the perceptually least important speech bits have no error protection, not even error detection. Therefore, there is no means for the speech decoder to know whether the perceptually least important bits are corrupt or not. Hence, possibly noisy data will be used in the audio reconstruction algorithm, much like noisy frequency coefficients were used in our inverse DCT example.

Application-level decoding of corrupt information will likely represent the primary practical means of realizing the benefits of JSCC on the decoding side of a noisy connection. Source-cognizant channel decoding described in Section 4.3.3 depends upon the channel FEC decoder having in-depth knowledge of the source's statistics. However, the multiple protocol layers separating the channel decoder from the application-level source decoder will likely prevent the channel decoder from obtaining explicit knowledge of the source's statistics. However, application-level decoding of corrupt packet data can be applied as a surrogate to source-cognizant Viterbi decoding due to the similarities in function. Both approaches attempt to reconstruct the best possible estimate of the transmitted image data given a noisy received data stream and given in-depth knowledge of the source's statistics. The difference is that source-cognizant Viterbi decoding operates on soft-decision information, which is more efficient, while application-level decoding of corrupt packets operates on hard-decision bits. The resemblance is close enough for us to contend that application-level processing of corrupt packet payloads can help to minimize end-to-end distortion in a JSCC system in much the same manner as source-aware channel decoding of robust data.

4.3.6 Forwarding corrupt packet data to enhance the perceived interactivity

Our analysis of JSCC in preceding sections of this chapter has established that the combination of error-tolerant encoding, UEP, and error-tolerant decoding of corrupt packet data reduces the end-to-end distortion compared to an independent source and channel coding approach when the wireless channel is very noisy and both FEC and compression are constrained in their complexity and delay. One of the major implications of the JSCC approach is that the perceived interactivity of delay-sensitive distributed multimedia applications will be relatively continuous over a wide range of *BER*'s, while the distortion suffered by packetized images will be vary according to channel conditions. Forwarding of corrupt packet data to the application for decoding allows the end user to interact almost immediately with a potentially noisy image, instead of having to wait for an error-free version of the image to arrive[1]. The end user will perceive a relatively constant transport delay due to the continuous interaction. The impact of time-varying nature of the wireless channel will be on the quality of the displayed image, which will degrade/improve gracefully as channel conditions worsen/improve.

Compare this continuous interactivity/variable distortion approach to a wireless system whose focus is on low-distortion delivery. Such a system throws away corrupt packet data and relies on either link-layer or end-to-end retransmissions to recover discarded packets. The end user must wait for the underlying network to reliably deliver each packet. Such a system provides constant minimum-distortion service at the cost of variable delay over a time-varying wireless channel. The JSCC design philosophy of forwarding corrupt packets in order to provide continuous interactivity/variable distortion is arguably a better fit for delay-sensitive applications than the intermittent interactivity/minimum-distortion.

1. Forwarding of corrupt packet data is only designed for natural or pre-rendered artificial images, and not for plain text or graphics drawing commands.

First, we want to establish that forwarding packets with corrupt payloads reduces the delay close to the interactive latency bound of several hundred milliseconds. Recall from Chapter 2 that we calculated the loose lower bound on the image transfer latency using ideal SRP. An image of size I bits was divided into F packet fragments, each of length $\frac{I}{F}$ bits. Define the channel bandwidth as BW, probability of bit error as BER, and header size per packet as H. Let $E[N_{packet}]$ represent the number of times on average that a fixed-size packet must be retransmitted. Let PTT_{packet} represent the time required to transmit a packet (payload + header bits) once over a link. In Chapter 2, we showed that the average amount of time to reliably transmit a packetized image $E[T_{\text{image}}]$ was lower bounded by

$$\begin{aligned} E[T_{\text{image}}] &\geq F \cdot E[N_{packet}] \cdot PTT_{packet} \\ &= \frac{1}{(1-BER)^{\left(\frac{I}{F}+H\right)}} \cdot \frac{I+F \cdot H}{BW} \end{aligned} \tag{4-1}$$

By forwarding packets with corrupt payloads, we are in effect requiring that the ARQ protocol only needs to ensure that the *headers* on packets are free from errors. Therefore, the BER factor that triggers the retransmissions is only dependent on the header size, not the payload size $\frac{I}{F}$. The revised average delay required to reliably transmit an image that only needs the *headers* on all packets to be error-free is given by

$$E[T_{\text{image}}] \geq \frac{1}{(1-BER)^{H}} \cdot \frac{I+F \cdot H}{BW} \tag{4-2}$$

Assume that the image size I is 20 kbits, channel BW is 500 kbit/s, and header size H is 100 bits, the same parameter values we chose in Chapter 2. Also, assume that the payload length $\frac{I}{F}$ is 1000 bits (i.e. the number of fragments F=20) and the channel BER is 10^{-2}. For these values, the image transfer delay for error-free header/corrupt payload delivery given by Equation (4-2) is calculated as 120 ms. By forwarding packets with possibly noisy payloads yet error-free headers, we are able to deliver the entire image close to the interactive latency bound of a couple hundred milliseconds despite the heavy *BER* and without any assistance from FEC. Though practical limitations will push the delay above this ideal lower bound, we contend that forwarding of corrupt-payload header-valid packets will allow image transfer latencies of close to the interactive latency bound over very noisy channels without the help of FEC.

Next, we would like to quantify the reduction in delay achieved by forwarding corrupt information in comparison to throwing away corrupt packets and retransmitting the discarded packets. By taking the ratio ρ of Equation (4-1) to Equation (4-2), we can compare the latency reduction obtained by forwarding corrupt packet payloads (requiring only reliable header transmission) to the approach which requires that both payload and header be error-free. This ratio ρ is given by

$$\rho = \frac{E[T_{\text{image}}](header + payload)}{E[T_{\text{image}}](header - only)} = \frac{1}{(1 - BER)^{I/F}} \qquad (4\text{-}3)$$

Note that ρ only depends on the *BER* and payload length $\frac{I}{F}$. We tabulate ρ as a function of $\frac{I}{F}$ and *BER* in Table 6.1. The reduction in latency can be quite dramatic, exceeding

Table 6.1: Factor by which delay is reduced by forwarding corrupt information, plotting ρ from Equation (4-3) for different *BER* and payload lengths.

ρ factor by which delay is reduced		Payload length *I/F* (bits)			
		10	100	1000	10000
BER	5×10^{-3}	1.1	1.7	1.5×10^{2}	5.9×10^{21}
	10^{-2}	1.1	2.7	2.3×10^{5}	4.4×10^{43}
	5×10^{-2}	1.7	1.7×10^{2}	1.9×10^{22}	--

a factor of 100 for various combinations of *BER* and payload length. For example, consider the point 10^{-2} *BER* and payload length $\frac{I}{F}$ of 1000 bits. For this point, the delay calculated earlier as 120 ms is a factor of $2.3\text{x}10^{5}$ less than the latency that would have been suffered if the entire payload and header were required to be error-free. The table provides some perspective on how much faster the response time is by forwarding corrupt information.

4.4 Implications of forwarding corrupt data on the design of data-link layer error protection

Forwarding corrupt error-resilient packets has several implications on the design of FEC channel coding at the data-link layer. First, while the payload may contain error-resilient data, the header is still very sensitive to errors. While we have shown that in the absence of any FEC it is still possible to transfer a complete image by the interactive latency bound despite a heavy *BER*, FEC on the header can help improve that probability of successful packet transmission with a minimal cost in overhead. The overhead cost of FEC redundancy is only suffered by the relatively small header, not the entire payload. This means that very strong FEC block/convolutional codes with coding rates of $\frac{1}{3}$ or even less can be applied with little expansion of the bandwidth.

Also, in order to distinguish payload-based corruption from header-based corruption, the protocol needs at least an error detection mechanism that operates just on the protocol header. As we mention below, it is also helpful to know whether the payload is in error at the channel decoder. Therefore, both header error detection, usually in the form of a cyclic redundancy code (CRC), and some form of payload error detection (though it need not be in the explicit form of a CRC) are required at the data-link level in order to implement forwarding of corrupt packet payloads.

In a sense, header-only FEC is a form of UEP, since the header is error-protected while the payload is not. The JSCC approach permits finer-granularity UEP-based FEC on the payload as well. However, care should be taken in choosing the type of FEC decoding to avoid the catastrophic effect of multiplying errors. Recall from Chapter 3 that an (N,K) linear block decoder attempts to find the code vector in N-dimensional space that is closest in Hamming distance to the noisy received vector. If the wrong code vector is chosen, then even though the transmitted code vector and the received code vector may be neighbors and differ by only one bit in N-dimensional space, the corresponding decoded K-dimensional codeword may differ by many bits from the original K-dimensional input codeword. Once the FEC decoder fails, then the effect is catastrophic, causing a multiplicative effect on the number of channel errors in the decoded corrupt packet. For example, transform-based decoding of RS codes can completely scramble the decoded packets if the RS decoder fails to correct the sequence [46]. Even though the corrupt packets contain robust encoded data, this multiplicative effect is still highly undesirable.

Fortunately, systematic RS codes offer one solution that permits error correction and forwarding of corrupt packets whose bit error rate is no worse than the channel BER. Systematic block codes encode a K-bit input word by appending N-K redundancy bits on to the original K-bit input word. Therefore, at the receiver it is possible to extract the original possibly noisy K-bit codeword even if the N-dimensional RS decoding fails. For non-systematic codes, the original codeword is not visible after encoding, and can only be extracted by decoding the N-dimensional received vector. With reasonable likelihood, RS codes can detect when their error correction ability has been exceeded. Therefore, once a RS decoder detects that error correction failure, then a suitable policy is to forward the original K-bit packet recovered from the noisy N-bit received codeword, rather than forwarding the possibly noisier K-bit vector from the output of the RS decoder. Under this policy, the forwarded corrupt packet will have no worse BER than the channel BER.

We observe that link-level retransmissions are not necessarily incompatible with forwarding of corrupt information. Several authors have described partial retransmission strategies which do not provide full reliability, but do make some effort to provide a cleaner version of each corrupted packet [69][103]. If partial retransmission is still unsuccessful after a finite number of attempts, then the policy is to typically discard dirty packets. Within a JSCC system, this policy can be modified to forward onwards to the application the final corrupt packet leftover after partial retransmission.

Also, while it may be tempting to employ a few link-layer retransmissions, we caution that over certain cellular links this can lead to large increases in delay. For example, the interleaving delay is near 40 ms each direction on the GSM system [85]. Given that roundtrip latencies of many multi-hop Internet connections are already at or near the interactive latency bound of about 200 ms, then there is very little margin left for additional retransmissions. If the retransmission loop at the link-level is very tight, then partial link-layer retransmissions can be employed to use up the rest of the delay budget for time-critical data. However, each retransmission over a GSM link will incur 80 ms roundtrip delay.

Finally, we observe that forwarding corrupt information across a noisy wireless link is ideal for low-latency packet transport, it can leave artifacts in a reconstructed still image. The end user of such a bursty media application would prefer that the screen presentation ultimately be cleaned of artifacts that were initially tolerated for the sake of interactivity. In the next chapter, we propose an end-to-end progressively reliable protocol which initially utilizes the corrupt packet payloads forwarded by the data-link layer yet eventually sends enough end-to-end redundancy to clean up noisy received data. In effect, the partitioning of responsibilities between the data-link layer and the end-to-end protocol is to make the link-layer responsible for header-valid packet delivery and the end-to-end protocol responsible for eventually error-free delivery of payload data.

4.5 Summary and conclusions

First, we have described how real-time distributed audio/video conferencing applications have been designed to tolerate unreliable packet delivery (e.g. packet loss) over the wired Internet in order to deliver their data quickly. We have shown that end-to-end roundtrip times on the Internet can exceed 100-200 ms and packet loss rates can range up to 15%, which suggest that reliable delivery via retransmission-based end-to-end protocols can take too long to transport packets for delay-sensitive applications. We have shown that bursty imaging applications, such as interactive Web-based image browsing, can also be designed to tolerate significant packet loss for the sake of rapid delivery.

The same trade-off of accepting increased channel distortion (i.e. image distortion due to channel impairments) for lower latency can be applied to the situation of wireless access to the Internet. Rather than waiting for reliable delivery over a noisy wireless link, the user can interact immediately with a possibly corrupt image. For delay-sensitive applications, we contend that consistent interactivity with a possibly noisy displayed image is subjectively preferable to intermittent interactivity with an error-free image. Thus, over wireless access links, the perceived latency can be lowered not only by tolerating packet loss, but also by tolerating packet corruption.

We outline the theoretical joint source/channel coding (JSCC) framework which supports the notion of forwarding and displaying corrupt data. We call into question the traditional separation of image compression design from network protocol/FEC design, also called independent source and channel coding, that is based on *Shannon's separation theorem*. This information-theoretic theorem was derived based on the assumptions of stationary memoryless channels, unconstrained complexity in the design of FEC encoders/decoders and compression/decompression, as well as unconstrained delay in the operation of each of these elements. Since our application of interest is interactive Web-based image browsing across time-varying wireless access channels with real-world limitations on complexity, then each of these assumptions is violated in practice. Moreover, the separation theorem is based on quantitative metrics like the loss probability, while the theme of progressivity developed in this thesis is motivated largely by a subjective performance metric which is a complex function of perceptual delay and perceived image distortion.

As an alternative, we outline the framework of *joint source/channel coding*, in which image encoders and decoders are designed to be error-tolerant, and FEC/ARQ channel encoders and decoders are designed with some knowledge of the image source's statistics in mind. The literature shows that the JSCC approach reconstructs images with lower distortion than the independent source and channel coding approach at severe *BER*'s. An important outcome of the JSCC approach is that the underlying network should not throw away corrupt packets containing error-tolerant image data, but instead should forward packets with corrupt payloads to the destination for decoding/error concealment. We demonstrate that error-tolerant image coding can achieve reasonable compression via lossy quantization alone (variable-length statistical compression is left out) as well as tolerate *BER*'s up to at least 3% - the same approximate *BER* at which RS codes that tripled the bandwidth were determined to fail in the previous chapter. Corrupt yet robust image data that is forwarded quickly to the receiver can be displayed immediately, thereby improving the perceived interactivity of the system in comparison to error-free delivery.

Finally, having established the advantages of forwarding corrupt information in a JSCC framework, we explore the implications on the design of data-link layer protocols of forwarding corrupt data. In particular, we note that systematic FEC codes are especially useful over noisy wireless links, because if the error correction fails, the systematic portion of the payload can be forwarded without magnifying the post-decoding *BER*, as would occur for decoding failures in non-systematic FEC codes.

In the next chapter, JSCC and forwarding of corrupt packet data serve as the foundation for the concept of progressively reliable packet delivery.

5

Progressively Reliable Packet Delivery

In this chapter, we introduce progressively reliable packet delivery. The observations made in the previous chapter concerning the usefulness of processing corrupt packetized image data over wireless channels serve as the starting point. We showed that forwarding and displaying error-tolerant corrupt image data can reduce the delay and distortion of the reconstructed image over very noisy channels given constraints on the complexity and delay of compression algorithms and FEC/ARQ channel coding. Forwarding corrupt information allows the end user to interact immediately with a possibly noisy image. For bursty multimedia applications like Web-based image browsing, subjectively harmful artifacts can persist indefinitely on the screen if no mechanism is available to clean up these initial artifacts. Hence, we add the idea of eventually reliable packet delivery to the idea of initially forwarding and displaying corrupt packetized image data to obtain the overall concept of progressively reliable packet delivery.

Following the overview, Section 5.2 identifies the essential properties that describe the progressively transport service, or socket interface, seen by a multimedia application and supported by the underlying progressively reliable transport protocol. These properties include forwarding corrupt information, delivering multiple noisy versions of a given packet, ensuring that the delivered versions are increasingly reliable, and delivering different packets out-of-order. Section 5.3 describes our modifications to an X windows server that implement a simple version of progressive reliability which adheres to each of the essential properties. Section 5.4 proposes that three performance-enhancing functions be added to the transport service interface in order to improve the end-to-end delay performance of the underlying progressively reliable protocol. These functions were motivated by experimentation and subjective evaluation within the X server emulator. These three new functions allow the application to parameterize the protocol to delay retransmissions, cancel out-of-date, or stale, retransmissions, and partition application data into multiple flows which are each served with a different quality of service by the protocol. Finally, in Section 5.5 and Section 5.6, we discuss three examples of how multimedia applications would utilize progressively reliable packet delivery. Progressive image transmission for bursty multimedia is tested in conjunction with progressively reliable packet delivery within our modified X server. In addition, we consider how continuous video and audio can parameterize a progressively reliable protocol to suit their needs, and implement this parameterization for the case of video.

5.1 Overview of progressive reliability

The advent of wireless access to the Internet poses a new challenge to interactive multimedia applications like Web-based image browsing originally designed to operate over end-to-end wired Internet connections. Such delay-sensitive applications require fast packet transport in order to provide the end user with genuine interactivity. Over a noisy wireless link that introduces bit errors, there is a fundamental trade-off between the desired reliability of a packet and the desired speed of delivery. A design philosophy which emphasizes error-free delivery of packets will have to incur variable packet latency over a time-varying wireless channel. The number of retransmissions generated by an

ARQ protocol in order to guarantee error-free delivery will vary according to the severity of the channel noise, thereby causing the delivery latency to fluctuate.

An alternative design philosophy is to back off on the guarantee of complete reliability and apply only FEC to mitigate the effect of channel errors. Such an open-loop approach supports distortionless delivery up to the point of failure of the error correction code. As we pointed out in Chapter 4, the failure point for this "brittle" FEC approach is about $5\text{x}10^{-2}$ *BER* for rate $\frac{1}{4}$ linear block codes, and about 3% *BER* for higher rate $\frac{1}{2}$ practical FEC codes. From Figure 3.4 in Chapter 3, an RS (255,127) code with 8 bits/symbol fails around $3\text{x}10^{-2}$ *BER*. Such an open-loop approach also supports constant transport delay up until the failure point of the code. However, when heavy channel noise causes the FEC to fail, packets are catastrophically lost, and the delay becomes exponentially large.

The JSCC design philosophy outlined in Chapter 4 consisted of error-tolerant image encoding, UEP, forwarding of corrupt packet data, and error-tolerant image decoding of noisy received packets. JSCC supports constant transport delay by forwarding corrupt packet data. Moreover, as we showed in the previous chapter, images can still be reconstructed at $3\text{x}10^{-2}$ *BER* given error-tolerant compression to 0.75 bpp. Therefore, the JSCC design philosophy supports continuous interactivity over a wider range of *BER*'s than practical FEC systems. Constant transport delay is again achieved at the cost of variable channel distortion. However, in comparison to the FEC approach that fails catastrophically, the JSCC approach allows the quality of the image to gracefully degrade as a function of *BER*, over a wider range of *BER*'s than practical FEC systems.

Our JSCC approach of forwarding and displaying corrupt error-tolerant information has been tailored for delay-sensitive multimedia applications like interactive Web-based image browsing. Channel distortion is tolerated initially in order to transmit a quick first version of an image to the end user. However, for such bursty-media applications, artifacts caused by channel noise can remain on screen indefinitely, until some user action over-

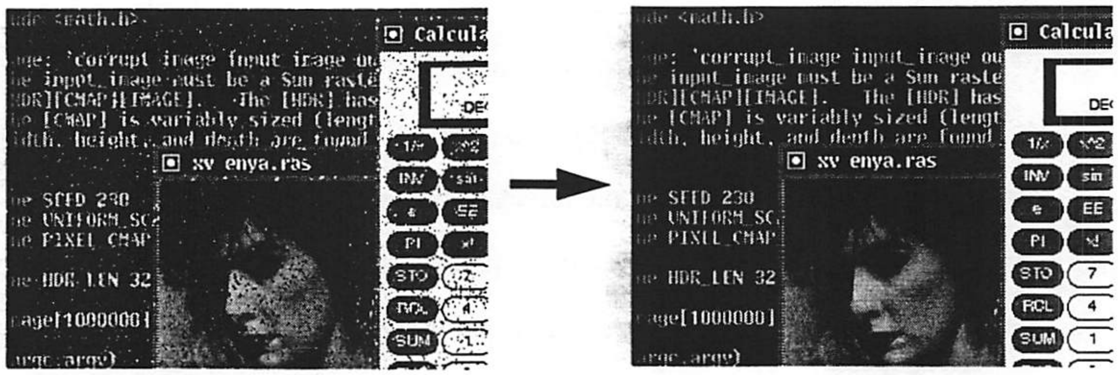


Figure 5.1 Progressively reliable packet transport delivers an initial possibly noisy version of a packet quickly to the receiver. Later, the protocol employs retransmissions to progressively improve the reliability of the delivered packet. An image-based multimedia application can use a progressively reliable protocol to display a noisy initial image (left) for immediate interactivity, and later remove any persistent artifacts on the screen by displaying progressively cleaner image data (right). The 8 bits/pixel colormapped image is corrupted at *BER* 10^{-2}.

writes the displayed noisy image. In contrast, for continuous video, artifacts may not persist either because the next frame overwrites the current frame, or because error concealment has been applied to compressed video to reduce error propagation, e.g. by interpolating from the previous frame or by leaky motion compensation [54]. In order to clean up these artifacts and provide eventually distortionless packet delivery, we observe that an asymptotically reliable cleanup mechanism based on retransmissions can be combined with forwarding of corrupt packet data to form a progressively reliable end-to-end protocol for packet delivery.

In Figure 5.1, we demonstrate the perceptual effect seen by the end user of a progressively reliable JSCC system. An application which employs a progressively reliable protocol would code its image data in error-tolerant fashion using lossy compression techniques like fixed-length quantization. The packetized image data would be forwarded to the receiver even if corrupted by an intervening wireless channel. At the receiver, the application decoder would display a possibly noisy first version of an image immediately. Over time, the progressive function of the protocol will eventually deliver a distortionless version of each packet, thereby cleaning up any artifacts on the screen. It should be clear that progressive reliability is designed to be used with coded pixel-mapped images, and not with text or graphics commands, though progressive reliability could be used to deliver images with pre-rendered embedded text/graphics.

Progressively reliable packet delivery can be seen as a form of progressive channel coding, analogous to how progressive image transmission is interpreted as a form of progressive source coding. In both cases, an initial high-distortion version of the image is transmitted and displayed quickly at the receiver in order to lower the perceptual delay observed by the end user. And in both cases, ultimately enough redundancy is sent to remove the distortion in the initially "noisy" displayed image. The difference of course is that progressive source coding introduces controlled distortion into the image while progressive channel coding introduces uncontrolled distortion.

Progressively reliable packet delivery offers an alternative to completely reliable and unreliable packet services over noisy time-varying channels. Fully reliable delivery will suffer variable delay, while unreliable delivery will suffer variable distortion and will leave persistent artifacts. Progressive reliability offers an intermediate solution adapted initially to meet the latency requirements of the delay-sensitive data, and adapted ultimately to satisfy the subjective distortion requirements of the data.

While this technique is primarily designed for natural images, ultimately some images will include embedded text/graphics. Therefore, we have included embedded text/graphics in Figure 5.1 to illustrate the impact of channel noise on these artificially rendered objects. The reader can also refer to Figure 4.1 and Figure 4.4 to study the impact of channel noise at 10^{-2} *BER* on compressed images for both natural and embedded artificial images.

5.2 Essential properties of a progressively reliable service interface

In this section, we identify the core properties which define a progressively reliable *service interface* between the multimedia application and the underlying network transport protocol that implements progressive reliability. At this point, we are not defining the precise implementation details of how a progressively reliable end-to-end protocol could be built, which will be left for the next chapter. Instead, we define the essential list of services and functions across the application/transport interface (ATI) which an application subscribing to a progressively reliable packet delivery service would expect to be provided. This distinction between transport *service* and transport *protocol* is an important one [129], and is evident in real-world systems. For example, the UNIX socket service interface is distinguished from the implementation details of the underlying transport protocol [123]. Applications that set up a reliable stream (in-order) socket connection across the Internet do not need to understand the protocol implementation details about how TCP supports reliable stream service. Finally, the service features outlined in this section only define the bare minimum ATI. The performance of progressively reliable packet delivery can be enhanced by adding an extra set of primitives to the ATI that allow the application to choose when and if retransmission redundancy should be sent. These important "added-value" ATI functions are described in Section 5.4.

In Figure 5.2, the progressively reliable service interface, or ATI (dotted line), is distinguished from the underlying transport protocol that implements progressive reliability. The application sends a packet across the source ATI to the transmitting end of the progressively reliable protocol. At the receiver, the application must be prepared to process multiple noisy versions of each packet which improve in reliability over time across the

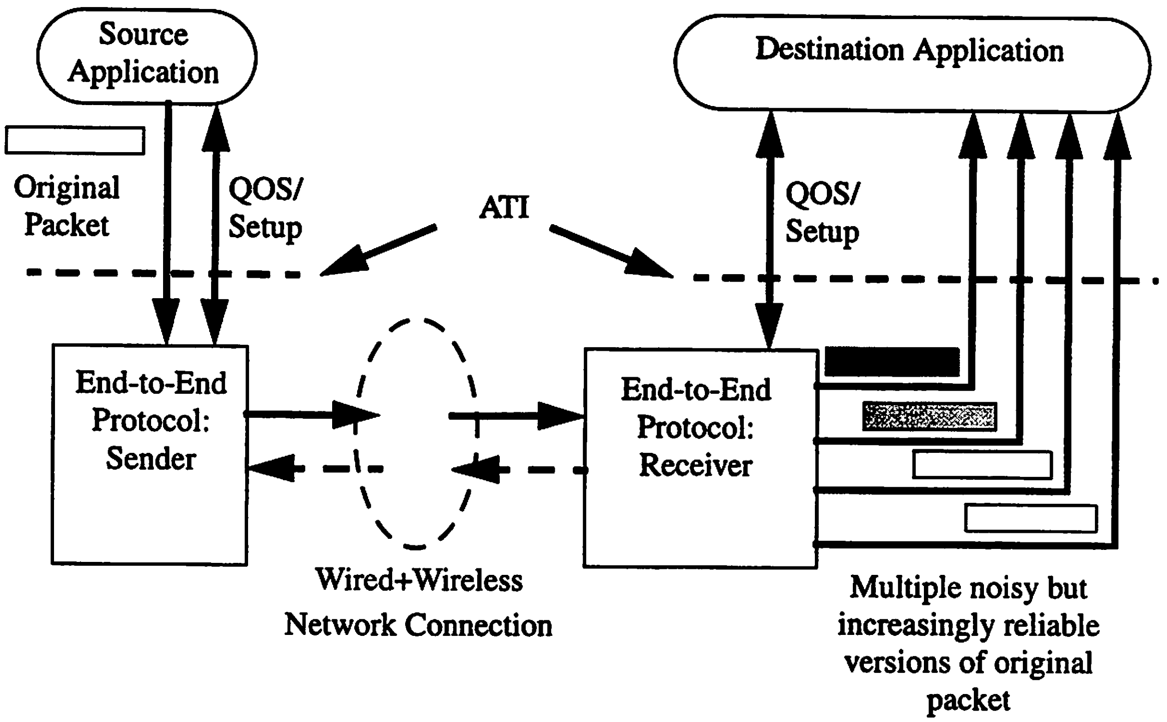

Source Application
Original Packet
QOS/ Setup
ATI
Destination Application
QOS/ Setup
End-to-End Protocol: Sender
End-to-End Protocol: Receiver
Wired+Wireless Network Connection
Multiple noisy but increasingly reliable versions of original packet

Figure 5.2 The end-to-end progressively reliable protocol provides four core services to the application through the application/transport interface (ATI): 1) corrupt data is forwarded to the application 2) multiple versions of each packet are delivered to the application 3) these multiple versions are increasingly reliable (fewer errors with each successive version) 4) different packets will arrive out of order (not shown). Other added-value functions which improve the efficiency of progressively reliable packet delivery (not shown) are described in Section 5.4.

destination ATI. Therefore, the essential properties of a progressively reliable packet delivery service are:

- Corrupt packets are forwarded to the application
- Multiple versions of each packet may be delivered
- The reliability of these multiple versions improves statistically over time (fewer errors with each successive version)
- Different packets may be delivered out of order

In the rest of the section, we discuss the implications of each of these properties on the design of the progressively reliable service interface.

5.2.1 Forwarding corrupt information

At the sender, the multimedia application transfers data through the source ATI to the underlying protocol for transmission. At the receiver, the transport protocol must support forwarding of possibly corrupt received data across the destination ATI to the application decoder.

5.2.1.1 Application-level framing

Suppose that the application sends a sequence of two images through the source ATI. At the receiver, assume that the two possibly corrupt images are forwarded to the application through the destination ATI. Also, assume there is no support from the transport service to help identify the original image or packet boundaries. Given a received sequence of noisy variable-length images, the application decoder can lose track of where an image begins and ends. The lack of framing support from the transport service means that the application must embed its own application-level header within the payload so that the length of the image can be extracted at the receiver. However, this length field can still be corrupted, even taking into account error correction, as shown in Figure 5.3. Once the application-level header is corrupted, then all subsequent information is lost because the decoder cannot identify the length of the current image, and therefore can no longer identify where subsequent images begin and end.

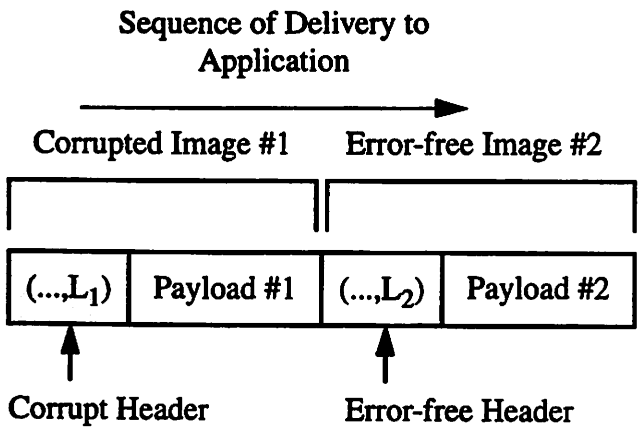


Figure 5.3 Lack of framing support from the transport service means the application can lose track of where a message begins. Once the length field L_1 is corrupted, then all subsequent messages (images in this example) cannot be identified.

In order to avoid losing synchronization with the data, the application decoder needs the transport service to preserve application-specified boundaries end-to-end. The source application must first parse its data into a sequence of variably-sized messages, called application data units (ADU's). This segmentation process is also called *application-level framing* (ALF) [23]. ALF does not exclude the possibility that the underlying transport protocol will fragment the ADU into smaller pieces in order to transmit the data. Indeed, we shall discuss later the role of ADU fragmentation in the design of a progressively reliable protocol. For now, we note that the ADU is the basic unit of data exchange across both the source and destination ATI's and a necessary function to support forwarding of corrupt packetized image data.

In the absence of corruption, reconstruction of an ADU in its entirety end-to-end despite fragmentation by the underlying transport protocol is relatively straightforward. Consider the real-world example of IP delivery of a UDP datagram. IP encapsulates a UDP datagram with an IP header. The combined IP header and UDP datagram is called an "IP datagram" [124]. The IP datagram may be divided into smaller IP fragments. Each IP fragment contains a header field specifying the length of the entire IP datagram. Assuming a fixed IP header length, then *each IP fragment is implicitly aware of the original length of the higher layer packet entity, or UDP datagram.* At the receiver, IP reassembles the IP datagram, and then strips off the IP header to reconstruct the UDP datagram. By analogy, the ADU represents the higher layer packet entity in our scenario which may be fragmented by the transport layer. Since each transport layer fragment contains implicit information on the original ADU's length within the fragment header, then the transport protocol is equipped to identify ADU boundaries and reconstruct each ADU in its entirety at the receiver.

5.2.1.2 ADU header and ADU payload

In the presence of bit corruption, preserving ADU boundaries end-to-end becomes a little more complicated. Preserving ADU boundaries is equivalent to communicating each ADU's length to the application decoder free from errors. Our approach is to first partition the ADU into an *ADU header* and an *ADU payload*. The ADU header will contain error-sensitive information like the ADU length, as well as error-sensitive image coding information like the image width or starting (x,y) coordinates. The ADU payload will contain error-tolerant coded image data, such as vector quantized pixels/blocks, frequency coefficients, or subbands. In addition, at the receiver, we propose to forward an ADU only if its ADU header is error-free. This combination of ADU partitioning at the sender and header-valid forwarding of ADU's at the receiver will permit corrupt ADU payloads to be passed to the application decoder without suffering our earlier problem of lost synchronization due to corrupt length fields.

Separating error-sensitive data from error-tolerant data supports the implementation of UEP on the ADU. Heavy FEC can be confined to the ADU header, thereby avoiding a large cost in overhead if the entire ADU were protected by heavy FEC. Similarly, minimal FEC can be applied to the error-tolerant ADU payload, thereby avoid possibly intolerable channel distortion if insufficient FEC were applied to the entire ADU. More generally, the FEC coding rate can be varied throughout the ADU payload.

At the sender, a CRC error detection checksum is appended to the ADU header and the combination is encoded via FEC. At the receiver, FEC decoding is followed by error detection on the post-correction ADU header. The entire ADU (header and payload) is forwarded to the application for further decoding only if the ADU header is error-free. In part, this is because an error in the ADU header essentially renders the information in the packet useless, e.g. if the (x,y) starting point of an image in the ADU header is corrupted, then the rest of the image data in the ADU payload will be drawn in the wrong location.

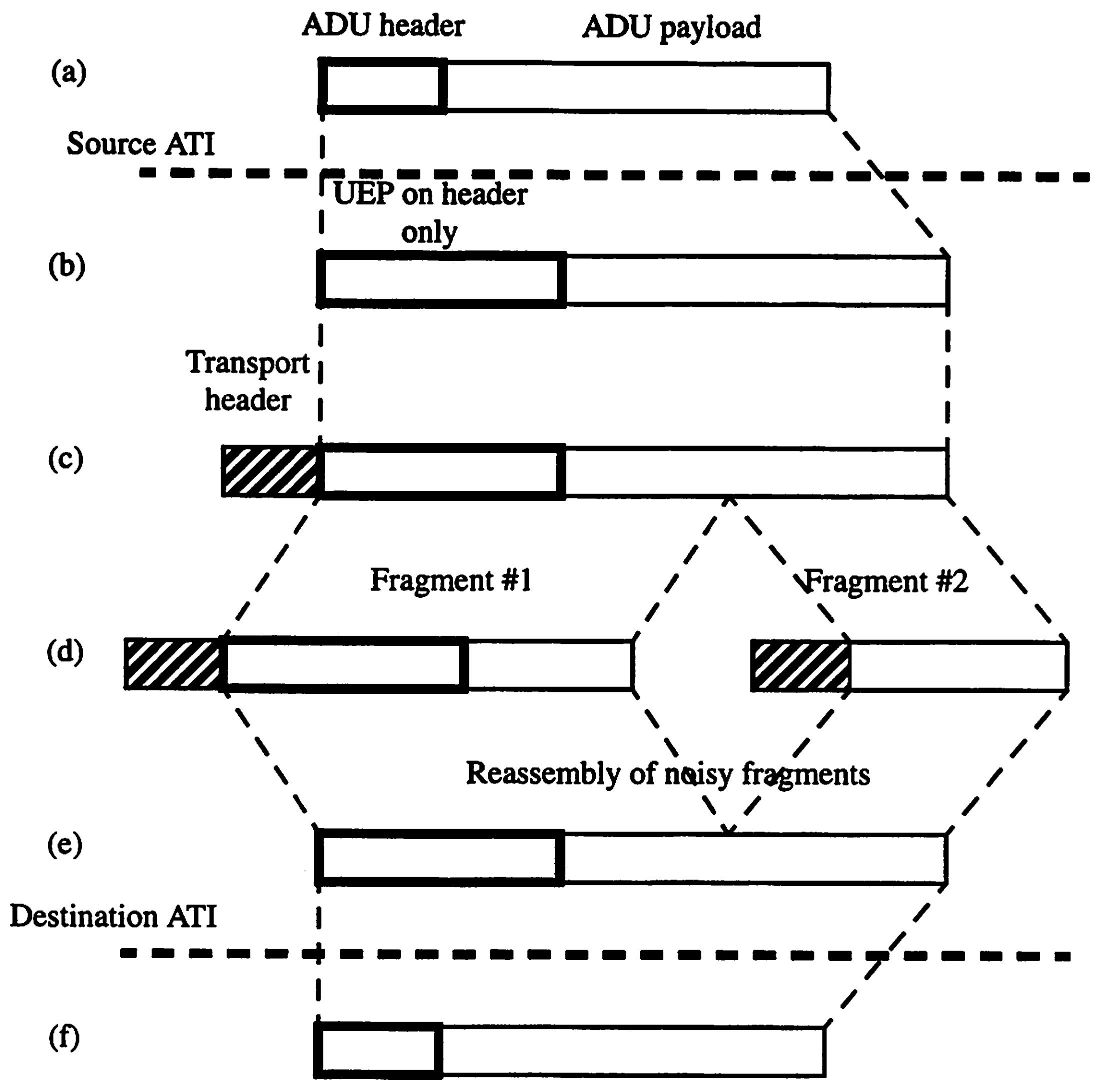


Figure 5.4 **A method for forwarding corrupt packet data which keeps the receiver synchronized: (a) the application sends messages called application data units (ADU's) to the socket buffer. These ADU's are partitioned into an ADU header (error-sensitive data) and an ADU payload (error-tolerant data). (b) UEP is applied by the transport protocol. Header-only FEC can be applied on both the ADU header (shown) and the transport header (not shown) (c) a transport header is appended to the expanded ADU (d) the transport "datagram" is fragmented for network packet delivery (e) Noisy fragments are reassembled (f) If after error correction decoding the ADU header is free from errors, then the ADU header and noisy ADU payload are forwarded to the receiving socket buffer.**

More importantly, forwarding only ADU's with clean headers allows the application decoder to remain synchronized with the ADU boundaries, because the ADU length field is cleanly reproduced at the receiver without any errors. The application decoder will perceive an alternating sequence of error-free ADU header and possibly noisy ADU payload, from which the ADU boundaries can always be extracted. Figure 5.4 shows how the concepts of application-level framing, ADU headers, ADU payloads, and header-only error detection can be combined to implement forwarding of corrupt packet payloads to the receiver in a manner that still preserves ADU boundaries end-to-end.

As part of the transport service of forwarding corrupt packet data, it is necessary to define the length of the ADU header. This length can either vary with each ADU, or can be fixed during setup of the progressively reliable connection. We propose that the ADU header, unlike the payload, be a fixed length that is configurable during setup of the progressively reliable connection. This approach simplifies the FEC decoding required for each ADU header. Variable-length ADU headers can be emulated by creating multiple connections that differ in the length of their ADU headers.

Table 7 summarizes the various transport service elements and the source and destination ATI boundaries across which they are defined. All three of the elements discussed in this section - the ADU payload, ADU header, and ADU length field - propagate end-to-end. The other elements will be described in Section 5.4.

Table 7. A listing of transport service elements and the ATI's across which they are defined

Source ATI	Across Network	Destination ATI
ADU payload	ADU payload	ADU payload
ADU header	ADU header	ADU header
ADU length	ADU length	ADU length
ADU correlation label	ADU correlation label	
ADU flow header	ADU flow header	

5.2.2 Multiple versions of each ADU

Another key property of a progressively reliable packet delivery service is that multiple noisy versions of a single ADU may be forwarded to the application. The application must be prepared to process multiple versions of the same ADU. This multiple-copy property is fundamentally different from the single-copy services offered by today's Internet. For example, TCP implements single-copy reliable stream service, while UDP supports single-copy unreliable datagram service. In both of these services, each message sent into the system produces at most one output message. In contrast, progressively reliable packet delivery generates possibly more than one output version of each input message.

The transport service should allow the application to parameterize progressive reliability to suit its needs. What constitutes "sufficient reliability" for one application may differ sharply from the tolerance for distortion of another application. The application should be able to parameterize whether the final version of each packet is completely free of errors, or whether the final version can contain some residual distortion. In the former case, retransmissions will be attempted indefinitely, much like a conventional ARQ protocol. In the latter case, the application should be able to specify an upper bound on the number of retransmission attempts. For example, certain delay-sensitive continuous-media applications like audio and video are not likely to benefit from indefinite retransmissions, and therefore progressive reliability should stop trying after a specified number of attempts.

5.2.3 Increasingly reliable versions of each ADU

Forwarding of multiple corrupt versions of an ADU to the application decoder without any further filtering has the undesirable property that the quality of more recent ADU versions may actually be worse than earlier ADU versions. The quality of an ADU which is forwarded directly to the application will depend entirely upon the severity of the channel noise during the time that this particular version of the ADU was retransmitted over the channel. This direct dependence on the channel noise can lead to a displayed image whose quality actually degrades over time despite retransmissions.

In order to avoid random fluctuations in ADU quality, a *progressively reliable* transport service needs to ensure that the multiple versions are statistically more reliable over time. The ADU versions delivered to the application no longer correspond directly to the data that is retransmitted over the noisy channel. Instead, there is an intervening filter implemented by the transport protocol which *successively refines* an ADU as more information brought by retransmissions arrives at the receiver. The end user will perceive that successive versions of an ADU have statistically fewer and fewer bit errors. This effect is pictured in Figure 5.2 as packet versions that have progressively lighter shades.

The requirement of monotonically improving reliability (in a probabilistic sense) means that the receiver must have some form of *memory* at the receiver that tracks the history of each packet. The decoding process can only ensure that the most recent ADU version has fewer errors than previous versions if it can remember the noise level of one or more of the previous noisy ADU versions. This knowledge is obtained by storing previous ADU versions, or by storing some noise metric associated with these previous ADU versions. ARQ protocols which exhibit this feature of saving previous packet versions to improve decoding are called *memory ARQ*, of which the *Type-II Hybrid FEC/ARQ* protocols are the best known example [78]. Another name for memory ARQ is *packet combining*, since ADU's are successively refined by combining the noisy data associated with several previous packet retransmissions. In order to make clear how successive refinement can be carried out by employing memory at the receiver, we discuss some examples below.

The simplest two-tiered scheme for ensuring that ADU's improve in reliability is to forward only one initial noisy version of an ADU and then wait until a final error-free version can be delivered to the application. The receiver keeps a single bit of memory per

ADU which remembers whether a noisy version has already been forwarded to the application decoder. Once an initial corrupt ADU has been delivered, then any subsequent retransmitted noisy ADU versions are discarded. The transport protocol waits until the ADU is successfully retransmitted before forwarding a final error-free ADU version to the application.

A more advanced approach would cache recent versions of the actual packets, rather than keeping an indirect noise metric. For example, a simple scheme would cache the two most recent corrupt packet versions, and combine them with the current received version. Majority-logic decoding can be applied bit-by-bit on these three packets. Given an odd number of packets, then for each bit decode a '1' if there are more ones, or a '0' if there are more zeros. This example illustrates one of the major features of memory ARQ, namely that by storing the received packet history, the protocol can more quickly converge to a clean representation of a packet. Consequently, fewer retransmissions are needed when a protocol employs packet combining compared to a protocol which ignores the received packet history.

Note that by replacing older ADU versions with more recent ADU versions, the majority-logic decoding example does not strictly ensure improving reliability. A necessary condition for successive refinement requires that the entire history used to decode previous versions of an ADU be preserved so that subsequent ADU versions can be decoded with increasing reliability. A recombining algorithm which caches only a partial history of received packet versions will be unable to ensure monotonically improving reliability. Unfortunately, this caching requirement can lead to an unbounded need for buffer space at the receiver. As a practical compromise, limited memory at the receiver may not be able to strictly ensure monotonically increasing reliability, but for engineering purposes the property of increasing reliability can be closely enough approximated.

In the next chapter on protocol implementation issues, more in-depth consideration of successive refinement based on incremental transmission and combining of FEC coding redundancy is presented.

5.2.4 Out-of-order delivery of different ADU's

Another property of progressively reliable packet delivery is that different ADU's will be delivered out of order. Since multiple corrupt versions of each ADU are delivered to the application, then the sequence of ADU's presented to the application will contain an interleaved mixture of multiple noisy versions of multiple distinct ADU's. For example, even though ADU A is transmitted before ADU B, the protocol may deliver a sequence like {noisy A, noisy B, error-free B, error-free A} to the application decoder. If both A and B represent images to be displayed in the same region on screen, and B represents the newer of the two images, then out-of-order multi-version delivery could result in an error-free version of image A overwriting the newer error-free version of image B. An application which subscribes to this progressively reliable delivery service must be designed to tolerate out-of-order delivery of ADU's. This suggests that application-level sequence numbers should be embedded within ADU headers to determine the most up-to-date ADU. For example, video sequences label images by frame number. This semantic content would be embedded within the ADU header, and would be completely transparent to the transport service.

However, as we shall discover in Section 5.4, by making the progressively reliable transport service partially aware of ordering precedents, we can improve the performance of the underlying transport protocol. We will provide a means for the application to identify out-of-date information to the end-to-end protocol. Therefore, in our example, the protocol would automatically stop trying to deliver the overlapped portion of the error-free version of image A, because it would know that the overlapping portion of image B supersedes the overlapped region of A in relevance.

5.3 An X windows server as a real-time simulator

In this section, we describe the X windows server which we modified to test in real-time our ideas concerning progressive reliability. Our goal is to emulate the visual quality and interactivity that an end user would experience operating an interactive multimedia application across a noisy channel that forwards corrupt error-resilient data to the application decoder for immediate display. The X windows server [82] provides a real-time platform that permits us to evaluate the subjective impact of channel noise on the quality of reconstructed images as well as on the perceived responsiveness of the system.

We modified a color X11R6 windows server which employed 8-bit/pixel colormaps. The X server typically receives drawing commands like DrawText(), DrawRectangle(), and PutImage() from X client programs like editors, terminals, Netscape, Framemaker, and a host of other applications. The X server translates these commands into pixel form and then draws the rendered pixel-mapped image into the computer's physical frame buffer for display on-screen.

5.3.1 Overall structure of the progressively reliable X server

Our task is to insert a channel model, a progressively reliable encoding module, and a progressively reliable decoding module into the X server. Figure 5.5 summarizes our modifications to the X windows server, which we describe in detail below.

Every rendering action which draws pixels into the frame buffer must be first intercepted and diverted into our software coding modules rather than being allowed to write directly into the physical frame buffer. We created a virtual frame buffer[1] (*vfb*) and redirected all pixel-writing actions to the *vfb*. Each time a drawing command intends to update

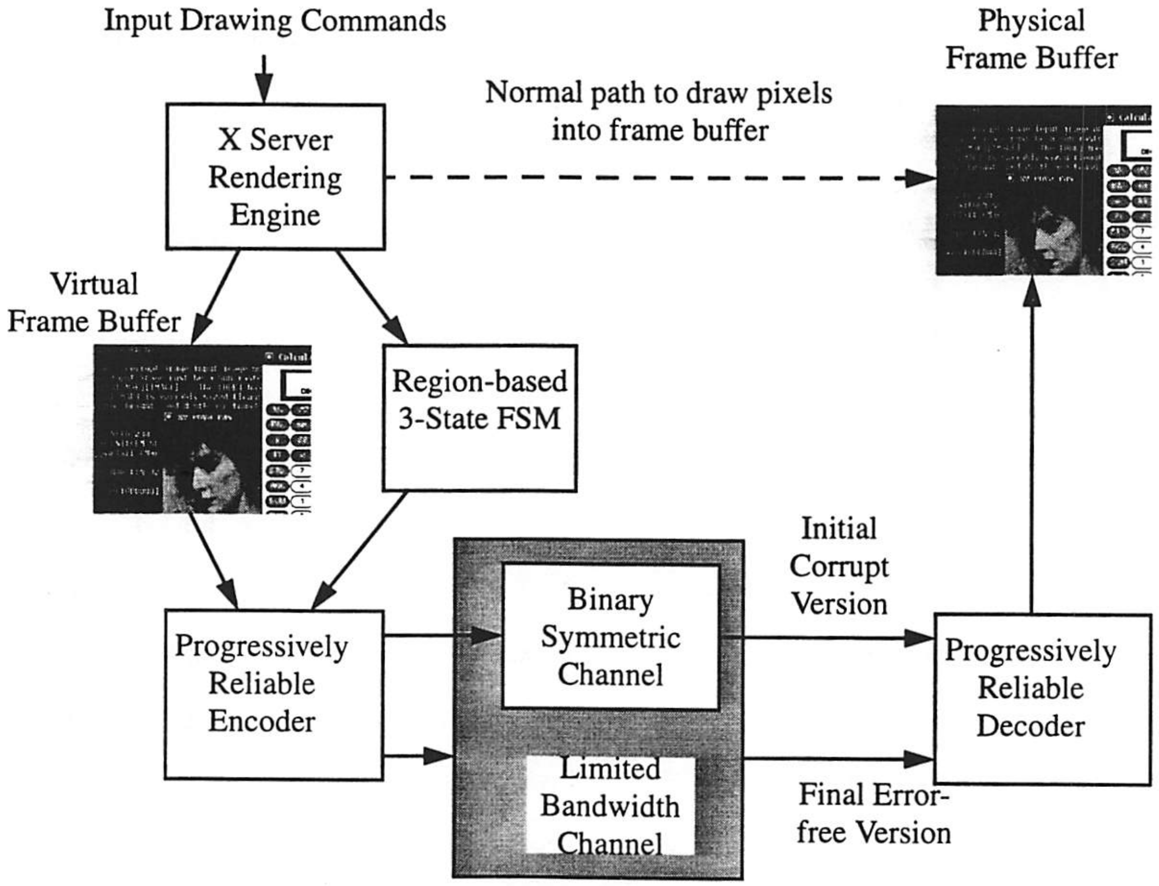


Figure 5.5 X windows server simulation environment for testing the subjective impact on the end user of progressively reliable packet delivery. All normal pixel rendering operations are diverted from the physical frame buffer to a virtual frame buffer. Newly updated blocks are recorded by a 3-state finite state machine (FSM). The progressively reliable encoder generates an initial and final version of each image. The initial version is corrupted using a random BSC error model. The final error-free version is sent at some later time. Both versions are constrained by a limited bandwidth channel

a region of the screen, the resulting rendered image is deposited in the appropriate position in the *vfb*. Equally important, a block-based state matrix is updated to record each block or region in the *vfb* which was recently overwritten. We can label this state as "new data" which is waiting to be transmitted over our noisy channel model. This state-based infor-

1. X11R6 has a virtual frame buffer extension, so we certainly take no credit for the idea. We implemented our own *vfb* in order to embed a state matrix specifically tailored to our block-based progressively reliable encoder. We would also like to credit Brian Richards of the InfoPad project for modifying our color server to work with the monochrome InfoPad. Also, we'd like to credit Jeff Gilbert of the InfoPad project for implementing a really jazzy multithreaded split color X server with *vfb* subsequent to our initial crude implementation.

mation is used by the progressively reliable encoder to send updates to the actual physical frame buffer.

As implemented in UNIX, the X windows server waits in idle mode until there is some new input data to process from X clients or from user actions (e.g. from a mouse/keyboard). The X server also has a self-wake time-out option which is invoked by prolonged inaction so that a screen saver can be started. We modified the X server to execute the progressively reliable encoder every time the X server wakes up, whether it is due to a time-out or new input. When there is no input data, then the time-out option permits us to continue to render images in a progressively reliable fashion. We set the time-out to 30 ms in order to achieve reasonably continuous operation of the progressively reliable encoder/decoder.

Each time the progressively reliable encoder is invoked, it checks the block-based state matrix for any blocks that are in the "new data" state. Our implementation avoided a brute-force search of all blocks by keeping a separate list of recently updated blocks, which could be quickly consulted. The encoder sends all blocks in the list immediately over the simulated error channel. The individual pixels in these initial block versions are corrupted by a binary symmetric channel model, which introduces random bit errors with probability *BER*. While a true fading wireless channel will introduce correlated bursty errors, we did not simulate this effect, though it should be straightforward to replace our memoryless BSC with a Markov channel model. We simulate the delay due to limited channel bandwidth that would be experienced by the corrupt block and then draw the noisy image block into the physical frame buffer. The state of the block is then changed from "new data" to "sent the initial version".

The progressively reliable encoder then searches the state matrix for blocks which have "sent the initial version". For all such blocks, the intent is to eventually send an error-free version which overwrites the noisy version displayed earlier. Afterwards, the state of the block is changed from "sent the initial version" to "sent the final version (quiescent)". All blocks in this final quiescent state are left undisturbed by the encoder, since they have already gone through the iterations of noisy and then error-free transmission to the receiver.

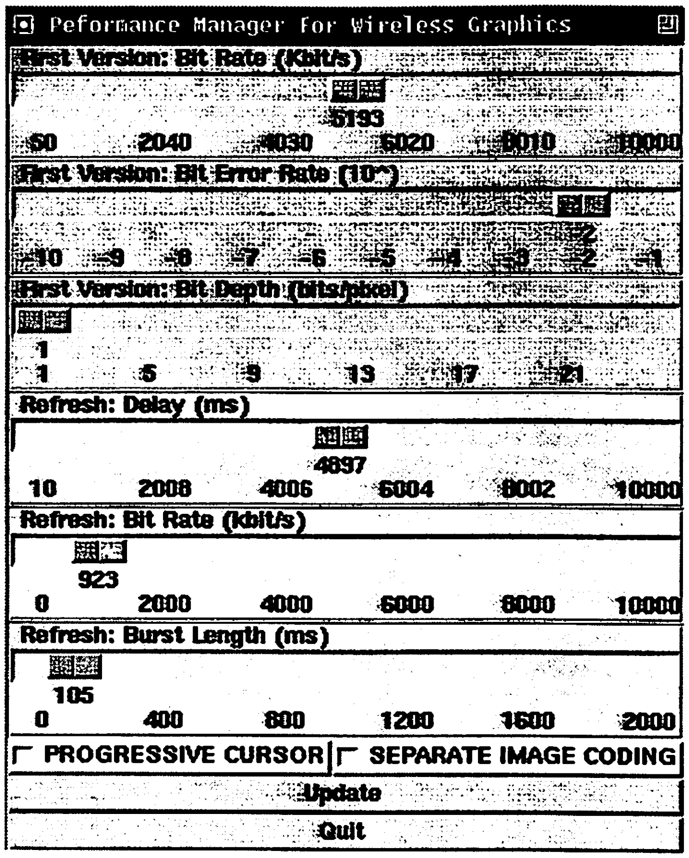
This overall effect perceived by the end user is that an initially noisy version of an image is displayed, followed by a refresh process which cleans up pixel errors at some later time. This two-tiered approach provides a crude approximation to progressive reliability. It emulates a protocol which forwards a single first noisy version of an ADU followed by a single final reliable version of an ADU to the receiver. This simple example was discussed in Section 5.2.3 as a means of guaranteeing increasing reliability.

The decision process that determines which of the "sent the initial version" blocks should be transmitted may delay eligible blocks for a variety of reasons. In order to avoid unnecessarily delaying time-critical data, we limit the number of reliable "refresh" blocks which can be transmitted by any invocation of the encoder. This avoids clogging the connection with a large number of reliable blocks, which would interfere with immediate delivery of delay-sensitive data. In a sense, we are biasing the modified X server to deliver delay-sensitive data with higher priority than delay-insensitive cleanup data. This limitation on the volume of reliable data is implemented as an upper bound on the refresh rate. This upper bound is parameterizable by the end user.

In addition, a minimum wait time parameter is associated with each block in the "sent the initial version" state. Only blocks in this state whose minimum wait times in the *vfb* have been exceeded are eligible for transmission. Thus, a minimum delay is established between the time that the initial version is transmitted and the time the final error-free version is transmitted. For example, if the wait time is set to zero, then the system defaults to immediately sending the pertinent blocks. By including a generally parameterizable minimum delay, we can adjust the gap in time between display of the initially unreliable image and display of the final reliable image. We will use this minimum delay parameter in Section 5.4 to establish the rationale for spreading retransmission-based redundancy in time.

Once the decision has been made to send a final reliable block, this block is delayed by the limited bandwidth channel, but is not corrupted with bit errors in order to emulate reliable packet delivery. Our assumption is that the delivery times for the refresh versions are so relaxed that it is not necessary to emulate the effect of FEC expansion on the delivery time of the final version. That is, if the reliable version takes many seconds to deliver, then the additional latency burden of FEC that may increase the overall latency by several more second will be largely imperceptible given that the delay is already many seconds long. Similarly, we assume that FEC is only applied on the header of all packets associated with first-time delivery of a noisy image. We assume that the increase in delay caused by FEC on the header only is too small to warrant emulation by our modified X server system. Also, we assume that the FEC is sufficiently strong on the header to reconstruct all packets, and hence all images, at the progressively reliable decoder, without packet loss.

The modified X server, with progressively reliable encoding/decoding and channel model is able to perform in real-time subject to certain qualifications. By diverting pixel writes into the *vfb*, we were unable to take advantage of the hardware graphics acceleration that the X server typically employs on SUN workstations. This means that all lines, text, and graphics, along with natural images and block copy functions, had to be rendered in software. Software rendering slowed our simulation down to slightly less than real-time for large screen updates, in which the redrawing of the screen could be seen, rather than appearing instantaneous as in a hardware accelerated system. This effect was mild enough that the system could be called interactive for most practical purposes.



```
Peformance Manager For Wireless Graphics
First Version: Bit Rate (Kbit/s)
                    5193
50     2040     4030     6020     8010     10000
First Version: Bit Error Rate (10^)
                                         -2
-10  -9  -8  -7  -6  -5  -4  -3  -2  -1
First Version: Bit Depth (bits/pixel)
1
1     5     9     13     17     21
Refresh: Delay (ms)
                    4897
10     2008     4006     6004     8002     10000
Refresh: Bit Rate (kbit/s)
  923
0     2000     4000     6000     8000     10000
Refresh: Burst Length (ms)
105
0     400     800     1200     1600     2000
PROGRESSIVE CURSOR | SEPARATE IMAGE CODING
Update
Quit
```

Figure 5.6 A Tcl-based performance manager hooks into the X windows server, allowing real-time variation of test parameters. For the initially noisy version of an image, the channel BER, channel bandwidth, and bit depth/pixel can be varied. For the final reliable/refresh version of an image, the minimum delay between initial and final version, as well as the refresh bit rate and maximum refresh burst length can be varied.

5.3.2 "Performance manager" tool for real-time evaluation

We took advantage of the interactivity of our modified X server to create a software tool, which we call a "performance manager", that communicates via a Tcl-DP UDP socket with the X server during run-time. The performance manager allows the end user to dynamically control parameters within the color X server while it is running. For example, for the initial noisy version of an image, the user can vary the channel *BER*, the channel bit rate, and the average bit depth/pixel. For the final "refresh" or error-free version of an image, the user can vary the refresh bit rate, refresh burst length, and minimum wait time between the initial and final versions. The Tcl interface includes scrollbars for dynamic adjustment of test parameters and is illustrated in Figure 5.6.

By varying the bit rates of the first and final versions, we can investigate the trade-off between devoting bandwidth to initial delivery of noisy delay-sensitive images, also called the initial substream, and devoting bandwidth to final reliable delivery of images, also called the refresh substream. We used this tool to characterize the refresh burst length threshold at which the perceived interactivity of the system was noticeably slowed down. Holding the bandwidth of both initial and refresh substreams constant at about 2 Mbit/s, we varied the maximum refresh burst length or duration, a critical parameter which controls how many refresh image blocks can be transmitted upon each invocation of the progressively reliable encoder. If there are a large number of reliable blocks awaiting transmission, then a large refresh burst duration will occupy the channel with refresh blocks for a long time and prevent time-critical initial substream data from being displayed quickly.

The refresh burst duration indirectly establishes a relative proportion of the channel bandwidth that is devoted to the refresh substream. This proportion varies depending upon the number of initial images blocks that are eligible for transmission at the time of encoder operation. When a large burst of initial data needs to be transmitted during a particular invocation of the progressively reliable encoder, then the refresh burst length limits the refresh substream to a small percentage of the channel bandwidth.

Our experimentation showed that small refresh burst lengths limited to less than about 30 ms were able to provide reasonable interactivity provided that the refresh bit rates exceeded 1 Mbit/s. The general intuition is that refresh bursts should be broken up into very fine-granularity segments so that possibly long bursts of initial substream data can be interleaved between short refresh packets. The specific numbers produced by the simulation are limited by the precision of the bit rate monitors and timers within the X server that cap the transmission rate or measure the delay. These monitors and timers perform imperfectly in a non-real-time operating system like UNIX, in which other processes on the workstation can arbitrarily steal CPU cycles. Specific values for such parameters as refresh bit rate, refresh burst duration, and minimum wait time between first and final image versions should instead be interpreted as indicating a general range of operation.

The number of "sent the initial version" blocks which can be transmitted per invocation of the encoder can be calculated as the product of the refresh burst duration and the refresh bit rate divided by the number of bits per block. Updates of this value are passed during run-time to the progressively reliable encoder by the Tcl performance manager.

Using this tool, we also observed that if not enough transmission bandwidth is devoted to refresh activity, then certain areas on the screen will take a very long time to clean up the artifacts, on the order of half a minute to a minute, rather than in seconds. For example, one of our earliest policies for limiting refresh activity was to suppress delivery of refresh blocks if there was any amount of activity in the initial substream. The rationale for this approach was that bursty user activity would eventually lead to intervals of no new image data, allowing the final reliable image versions to be drawn. However, we observed that a video stream which generated a continuous sequence of new initial image versions for transmission would starve the refresh bit stream. The lesson learned was that a subjective balance must be maintained between interactivity and distortion, in which delay-sensitive data is given priority for transmission, but not to the complete exclusion of reliable delivery of data. This starvation issue is closely related to the concept of *fairness* developed for scheduling policies in operating systems [119] and packet switches [159].

The X server checks for any new updates on parameters controlled by the performance manager every time it is woken by a time-out related to "prolonged" inaction. Since we changed the time-out value from minutes to 30 ms, then the updates actually occur quite quickly. However, if there is continuous new input (e.g. video), then it is possible that the X server never times out, and therefore never registers the newest updates sent by the performance manager. In the current implementation of the X server, new parameter values from the performance manager can only be registered between video sessions.

5.4 Enhancing the performance of progressive reliability

In this section, we extend the set of essential functions described in Section 5.2 to include new service primitives which improve the end-to-end delay performance of the progressively reliable transport service. In particular, the perceptual delay of an image, as measured by the delivery latency of the first version of an ADU, can be reduced by:

- delaying ADU retransmissions such that conflict with delivery of the initial version of an ADU is minimized
- cancelling retransmissions associated with out-of-date data
- flow-based scheduling of the sender's traffic by delay and reliability constraints

These primitives allow the application and underlying transport protocol to cooperate to reduce the amount of traffic sent over a wireless channel, and to ensure that delay-sensitive source traffic, such as the first version of an ADU, be given priority in transmission over delay-insensitive traffic, such as the retransmitted versions of an ADU. The rationale behind each of these primitives was developed based on experimentation with our modified X server simulator.

5.4.1 Delaying retransmissions

The progressively reliable transport protocol can control when follow-up redundancy in the form of retransmissions are transmitted. The protocol shapes its transmitted traffic pattern by delaying retransmissions so that they don't conflict with the initial transmission of an ADU's information. Delivery of the initial version of an image is delay-sensitive, since the end user desires to interact immediately with that image. Therefore, the sender side of the protocol prioritizes its transmission to transmit the delay-sensitive traffic first. The redundancy sent at a later time by the progressively reliable protocol to successively refine noisy ADU's has a comparatively relaxed latency requirement. Therefore, these retransmissions can be delayed in order to accommodate immediate delivery of delay-sensitive initial transmissions.

We implemented a simple mechanism within our modified X server which delayed the final refresh version of an image from being sent until after a minimum wait time had expired. The effect emulated by this policy is that all retransmissions are initiated no sooner than a minimum fixed delay. Given a bursty multimedia source, we reasoned that this simple mechanism would more often than not translate retransmissions to a time window which is unoccupied by new delay-sensitive data. If the time window is occupied by delay-sensitive traffic, then the other throttling procedures like the refresh burst length and refresh bit rate on the refresh substream will in effect delay retransmissions even more than the minimum wait time. This "Refresh Delay" parameter is adjustable during run time by one of the scrollbars in the performance manager shown in Figure 5.6.

Our experiments with the X windows server and performance manager revealed that it is perceptually acceptable either to send the retransmission redundancy many seconds after the initial delivery, or within about 100 ms of the initial version, but that any delivery of the reliable version in between 100 ms to a couple of seconds after the initial version produced subjectively annoying side-effects. The most attractive option is to send enough retransmission redundancy so that a final reliable version can be reconstructed quickly at the receiver within about 100 ms. However, in the presence of severe channel noise, even retransmissions protected by practically constrained FEC will fail, so that the fully reliable image will be unable to be delivered by the interactive latency bound.

A second option is to start retransmitting refresh redundancy at the first opportunity possible, whenever there is no delay-sensitive data left to send. This opportunity will usually occur within a few seconds after the initial delivery. However, we found that there were several annoying subjective side-effects which occurred when retransmissions were delayed by between 100 ms to 1-2 seconds. First, for images containing rendered text, as the end user started to read the text, the refresh redundancy overwrote the noisy version with the clean version. This subjective "flicker" effect forced the reader to stop and then start reading again, thereby obviating the perceptual reduction in delay gained by early noisy delivery of an image. To a lesser degree, the "flicker" effect also interrupted viewing of natural images. Second, as our mouse cursor was moved around the screen, a "noise trail" was left in its wake. By sending the refresh redundancy within 1-2 seconds after the initial noisy version, the noise trail would chase the mouse cursor across the screen like a "snake" in a video arcade game. This "snake" effect proved to be subjectively distracting.

Our experimentation suggested that the end user could tolerate very long delivery latencies for the reliable version of an image, so long as the user had an initial version of an image with which to interact. Delays on the order of 5 seconds and more were found to be readily acceptable. Subjectively, the user's desire to eventually see the visual presentation free from artifacts places an upper bound on the delay of the final version. We found that about a 5 second cleanup delay was fast enough to satisfy this distortion criterion and long enough to avoid the "snake" and "flicker" effects mentioned previously.

Given the end user's tolerance for considerably relaxed delivery of the reliable follow-up redundancy for an image, then our progressively reliable protocol can smooth the traffic that it presents to the network, scheduling retransmissions to minimize their impact on delay-sensitive delivery of initial packet versions.

Consider in Figure 5.7 how a traditional ARQ protocol responds to a high *BER* noisy channel. A sequence of application data bursts corresponding to separate images is passed to the underlying transport protocol for packet delivery (a). A traditional ARQ protocol will concentrate retransmissions of a given packet as closely as possible to the original transmission of that packet. The presumption is that the user desires reliable delivery as soon as possible, so there is no reason to delay retransmissions.

This grouping of retransmissions in time slows down with the delivery of data in the tail end of the sequence (b). The last of the three image bursts is delayed by retransmissions associated with the previous two image bursts. The end user will perceive the screen to inconveniently freeze as the underlying network protocol is occupied with retransmitting older image data, while the most recent image waits in a queue at the sender. In contrast, our progressively reliable protocol can translate retransmissions in time so that each new image does not have to wait in the sender's queue (c). An initial version of each new image is transported quickly to the receiver, and any needed retransmissions are sent later. If the overall perceptual latency of image delivery is measured by the first appearance of an image, noisy or not, then Figure 5.7 indicates that our retransmission-shifted progressively reliable protocol will on average have a lower perceptual delay than a conventional ARQ protocol which concentrates retransmissions.

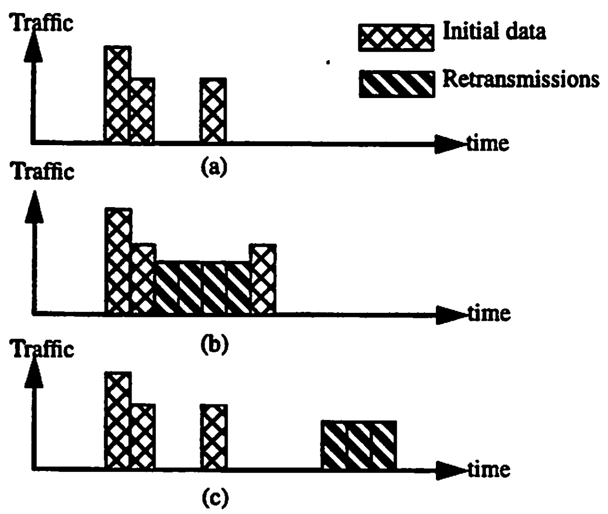


Figure 5.7 Traffic shaping of retransmissions: (a) Initial traffic burst (b) retransmissions delay some initial data over noisy channel (c) retransmissions are delayed so they don't slow delivery of the initial burst.

Thus far, delaying retransmissions has been motivated by our subjective observations of the end user's tolerance for considerable delivery latency in the low-distortion version of an image provided an initially noisy image is quickly delivered. In the past, protocols have delayed retransmissions for other reasons, including congestion avoidance and channel state-dependent scheduling. For example, part of the congestion avoidance policy of TCP involves exponentially backing off on (e.g. doubling) the time-out value for retransmitted segments [25]. The effect is to delay retransmissions so that they don't add congestion to the already overcrowded network. Recent work has also proposed to effectively delay the retransmissions of a data-link protocol based on the wireless channel state of a particular user [10]. The assumption is that the data-link protocol groups the packets from different users into a single statistically multiplexed stream and attempts to sequentially deliver the packets in this stream. A packet that is transmitted to a user with a very noisy channel will have to be continuously retransmitted. Given a sliding window protocol, then other users' data will be queued up waiting for this bad-channel packet to be delivered reliably, so that the window can be allowed to slide forward. The authors propose to adapt the retransmissions to the state of the channel, and essentially delay retransmissions to those users with a bad channel, on the presumption that these retransmissions will not get through error-free anyway, and will only slow down other users' data. Thus, we see that translating retransmission redundancy in time is not a new phenomenon, but has previously been based only on channel considerations. In contrast, our proposal to delay retransmissions is based on the end user's subjective tolerances.

We propose that the transport service socket interface provide hooks into the progressively reliable protocol so that the application can parameterize how long the protocol should wait before initiating retransmissions of the follow-up image redundancy. This crude translation approach does not fully prevent conflict with delay-sensitive data, once retransmission data are eligible to be sent. Our observations of the modified X server revealed that additional limitations on the refresh burst length and refresh bit rate were necessary to preserve interactivity. A more sophisticated approach to providing a fair balance between interactivity and the pace of cleanup of visual artifacts is to treat the initial/final trade-off as a packet scheduling issue. Initial header-valid packet versions are labeled with a small delay budget, while final versions are labelled with a large delay bound. Given this partitioning of delay sensitivity, packet scheduling algorithms like Earliest-Due-Date/Least Slack [158] can implement the desired fairness policy. In general, we propose that any fair packet scheduling policy implemented by the protocol be parameterizable to suit the individual multimedia application.

5.4.2 Cancelling "out-of-date" retransmissions

In addition to delaying retransmissions, our progressively reliable protocol also provides the application with the ability to identify out-of-date or stale data, and to stop or cancel retransmissions at the sender-side of the protocol associated with this out-of-date data. The ability to annihilate stale retransmissions helps reduce the amount of bandwidth devoted to unnecessary retransmissions of packet data that are no longer useful at the receiver. Over a bandlimited wireless link, this bandwidth management tool permits more timely delivery of delay-sensitive data because a sliding window protocol doesn't have to occupy itself with sending the final reliable version of an outdated image, and instead can devote itself to immediate delivery of useful first-time ADU versions.

Experimentation with our modified X server demonstrated the importance of using retransmission cancellation in conjunction with delayed retransmissions for bursty multimedia. Many multimedia applications like Web-based image browsing often overwrite the same region of the screen with new image data. If the protocol delays sending the reliable retransmission redundancy by many seconds for an image, then the queued image data at the sender can become out-of-date due to the arrival of new user data passed through the socket buffer which invalidates the queued data. For example, if the final reliable version of an image is delayed by five seconds, the amount found to be subjectively preferable in the previous section, then a user who is browsing rapidly through a remote image database at a rate of an image every few seconds would consistently cause the protocol's queued image data to become stale. Moreover, the interactivity of the system would be compromised since in many cases retransmissions associated with stale packet data would commence just as the end user has chosen a new image and desires its immediate display.

In order to improve the interactivity of an application which subscribes to progressively reliable packet delivery, we propose that a retransmission cancellation function be provided to the application through the transport service interface. We propose that this function be implemented by labelling related packets with a *correlation label*. In this way, new ADU packet arrivals given to the protocol for delivery can be used to target, eliminate, and replace old information at the sender's queue.

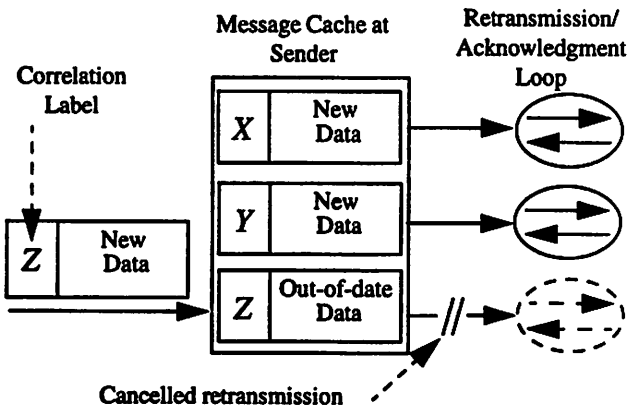


Figure 5.8 Packet annihilation uses correlation labels to stop out-of-date retransmissions. The incoming message with correlation label *Z* cancels retransmission of any previous message with the same label.

Figure 5.8 illustrates how packet cancellation would work across the transport service interface. The application segments its data into ADU's for transmission as before. Each ADU is assigned a correlation label by the application. The sender-side of the protocol inspects the correlation label (Z) of the incoming ADU, and checks if it is currently retransmitting any information with the same correlation label. If so, then the protocol stops retransmitting this old data immediately, and removes this out-of-date data from the sender's queue, replacing it with the new ADU's payload.

The modified X server emulated this cancellation function through its three-state FSM. Recall from Figure 5.5 that each 16x16 block on the screen is described by one of three states: "new data", "sent initial version," and "sent final version (quiescent)". As new image data arrives, the states corresponding to the redrawn blocks are updated to "new data", regardless of whether their previous state was "sent initial version" or "quiescent". In addition, the affected blocks in the *vfb* are redrawn with new data overwriting or replacing the previous image. If the previous state was "sent initial version", then newly drawn image blocks are in effect cancelling the retransmissions associated with the previous image.

In this implementation of the cancellation function, the (x,y) drawing location of a block implicitly serves as a correlation label which identifies this block as out-of-date, stops retransmission of stale data by purging this block's associated state, and replaces the old ADU (i.e. block) with the new ADU's information. Application of correlation labels on a geographic block basis as implemented by the X server is depicted in Figure 5.9 (a) and (b). Only those blocks which require redrawing will trigger annihilation of stale retransmissions associated with overwritten.image data.

Recall from Section 5.2.4 that we suggested correlation labels would be useful to cancel an out-dated image *A* with a new image *B*. Given the context of the block-based correlation labels described in this section, then it should now be clear that only the overlapped portion of *A* that is overwritten by image *B* will trigger annihilation of stale retransmissions. The non-overlapped portion of *A* will continue to be delivered in asymptotically reliable fashion. Hence, images A and B need not coincide in order for correlation labels to be effective. In Figure 5.9 (b), the new launched shuttle image completely overlaps the old unlaunched shuttle image. Therefore, out-of-date retransmissions of the entire old image will be halted, and no non-overlapped portion of the image remains to be delivered with progressive reliability.

We verified that this block-based implementation of the cancellation function performed as expected within the X server. For interactive bursty multimedia, our experimentation showed that the combination of FSM, *vfb*, and (x,y) correlation labels always stopped retransmission of out-of-date reliable redundancy. For example, we delayed retransmission of the final reliable version of an image by five seconds. Recall that our progressively reliable encoder operates on pixel-mapped images, and not commands.

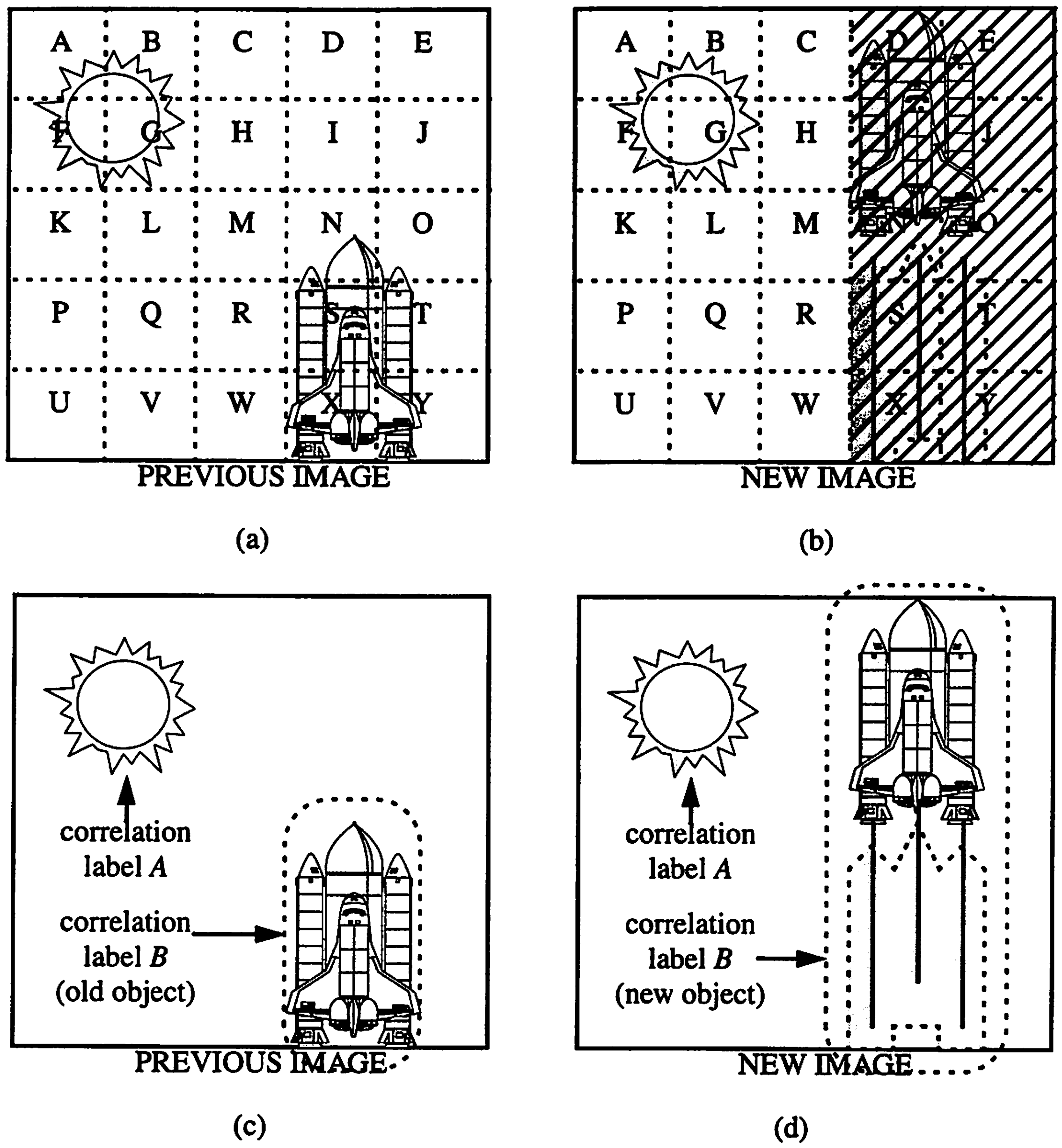


Figure 5.9 Possible uses of correlation labels by a multimedia imaging application. (a) The screen/image is subdivided into blocks, each designated by a unique correlation label. (b) Retransmissions associated with out-of-date (shaded) blocks are cancelled and new image data are transmitted in their place. (c) Object-based correlation labels allow the application to semantically identify objects within an image. (d) When an object changes position/size, the application passes a new ADU (e.g. with label B) to the protocol to automatically cancel retransmissions associated with the out-of-date object. In addition, the new object data is transmitted in place of the cancelled object's ADU, resulting in the new object being drawn and overwriting the old object.

Therefore, paging in a text editor causes new pixel-mapped images with rendered text to be redrawn over the same region of the screen. Frequent paging at a rate faster than one page every five seconds should result in cancellation of retransmissions associated with the overwritten out-of-date images. We verified that this indeed took place. Also, by waiting longer than the minimum wait time of five seconds, the final reliable version of an image was appropriately drawn, indicating that the FSM was functioning as expected.

An alternative way of applying correlation labels is to associate each label with a semantically-defined object within the image. As the object is moved, resized, or redrawn, each new ADU with that object's correlation label cancels any retransmissions associated with stale drawing information for the object. For example, in Figure 5.9 (c) and (d), correlation label *B* associated with the shuttle object is used to stop retransmissions that would draw the unlaunched shuttle. In their place, new "launched shuttle" drawing data is transmitted which renders the shuttle in a new screen position and overwrites the old shuttle image. Object-based ADU's can be adapted to track moving objects and erase the trail of rendered images left by the moving object. These could be applied to animation sequences. For example, a distributed video game could draw a moving snake on screen via object-based correlation labels. In addition, the correlation labels pose no restriction on the shape of the drawn image data, so that these objects can be arbitrarily shaped, and need not be block-based as in our X server implementation.

Correlation labels are not only applied at the sender; they are also propagated to the receiving end of the transport protocol. This has the advantage of ensuring that out-of-date ADU's are not forwarded to the receiving application. If only source-based annihilation were carried out, then it is possible for a stale ADU to arrive at the receiver after a more up-to-date ADU, due to network delay jitter. Without correlation labels, the receiving end of the transport protocol would be unable to identify stale ADU's, and hence would forward out-dated information to the application. The destination application would have to implement its own sequencing scheme to detect stale ADU's. We can eliminate out-of-date delivery at the receiver by forwarding correlation labels to the receiver's transport layer so it can identify and discard stale ADU's as they arrive. Another potential benefit of propagating correlation labels is that stale ADU's could possibly be purged at any point along the connection, provided that sufficiently intelligent transport-aware agents were present in the network as suggested in [5]. Since the labels are already available at the receiver, then no additional cost is incurred by forwarding these labels to the receiver's transport layer. We previously indicated the propagation of correlation labels across the network connection in Table 7.

Though not shown in Figure 5.8, our transport service also permits an ADU to annihilate *multiple other ADU's* that have different correlation labels. The source application might have multiple hierarchically coded versions of an image being serviced by the transport layer. When a new image is transmitted, the application would like to simultaneously cancel any of these out-of-date ADU's. Or an image may be packaged as a macroblock, and the application would like to annihilate a collection of smaller images that were previously packetized into ADU's. In each case, we conveniently provide a primitive that cancels multiple other ADU's. The convention is that each ADU may contain a list of correlation labels. The first label is associated with the parent ADU. Any stale ADU's with correlation labels on the list are annihilated. For example, in Figure 5.8 the newly arriving ADU identified with the correlation label Z could also contain other labels, e.g. both Z and X. In this case, the newly arrived ADU will stop retransmissions on both ADU's Z and X queued in the sender's buffer. In addition, since Z is the first correlation label listed, then the newly arrived ADU will be queued at the sender under the parent label Z, and not X.

From a practical perspective, it is not certain whether the list of correlation labels should also be propagated along with the ADU to the receiving end of the transport layer to cancel any remaining stale data. While we proposed propagating correlation labels to the receiver when there is only one correlation label per ADU, the disadvantages of complexity and overhead for propagating a list of correlation labels to the receiver for each ADU may overwhelm the advantage gained in consistency by cancelling out-of-date at all points along the connection.

5.4.3 Flow-based scheduling

Rescheduling retransmission traffic to minimize conflict with delay-sensitive initial ADU deliveries represents a form a scheduling which discriminates between initial and final versions of a packet. Scheduling initial transmissions and final retransmissions can be interpreted as a two-flow scheduling problem. This scheduling problem can be generalized to include scheduling multiple flows of data generated by a single application.

Traffic capacity can be improved by noting that multimedia applications often generate heterogeneous data which have inherently different requirements for timely delivery and end-to-end error protection. Recently, multimedia compression algorithms have started to encode their data streams into multiple flows, or layers, where each flow requires a different end-to-end QOS. Some examples of flow-based coding include asynchronous delay-cognizant video coding [75][106] and multilayered multicast video coding [84]. In addition, hierarchical and progressive image transmission techniques can easily be adapted to multi-substream transport services. Hence, our protocol should be able not only to schedule initial transmissions and retransmission redundancy differently, but should also permit flow-based multiplexing. This flow-based approach can result in retransmission redundancy from one flow being serviced with different urgency than another flow's retransmissions.

To support traffic shaping of retransmissions, quality-of-service (QOS) configuration information is communicated between the application and the underlying transport protocol across the ATI. Some socket interfaces have already begun to support QOS [11]. Other protocols also support some degree of QOS configuration at the transport level, as noted in [26] and for OSI/TP4 [32]. A full discussion of QOS (setup, admission control, profiles, etc.) is beyond the scope of this thesis, but we mention those items most relevant to our progressively reliable wireless protocol. QOS parameters control the scheduling of retransmission redundancy with initial transmissions but also permit finer grades of service within the data stream.

We introduce a *flow header* to identify individual substreams within the general data stream; these substreams at the transport level resemble IPv6 flows [57]. The flow header alone completely characterizes the service required by that data; no knowledge of the internal semantics of the packetized data is required. The per-flow QOS parameters that we have found most useful include how much scheduling delay the first-time transmissions can tolerate, how much initial corruption can be tolerated (i.e. how much FEC overhead should be applied to the payload), how soon retransmissions can begin after confirmed initial delivery, how much multiplexing delay that retransmission redundancy can tolerate, and when to terminate retransmissions (e.g. after a finite number of trials, or after a finite amount of time, or after a sufficiently reliable packet has been received). Nearly every QOS parameter is specified either in terms of delay or reliability. We do not explicitly support "priorities" across the ATI, since we believe that priorities and classes can be derived from delay-based and reliability-based QOS parameters. The QOS parameters are passed to our progressively reliable protocol, which can then employ an appropriate scheduling policy to decide what packet to transmit next over the connection.

The protocol's scheduling policy, in combination with the behavior of the wireless channel, will dictate whether guaranteed service can be provided to the application across the ATI. Deterministic guarantees of bounded delay cannot be made for certain multiplexing disciplines like Fair Queueing and Virtual Clock, but can be made for others like EDD for well-behaved sources practicing admission control over backbone networks [3][158].

Even if a service discipline can guarantee delay bounds at the source, the wireless access link will introduce uncontrollable delay. In wireless systems, delay is caused by random shadowing and random multipath effects, leading to bit corruption, packet loss and uncontrollable retransmission delay. At best, a statistical characterization of the fading may provide a statistical bound on delay, but this assumes that the channel is predictable. We prefer to view QOS parameters as passing useful hints from the application onto our progressively reliable protocol to help improve non-guaranteed performance. The protocol's scheduler will decide how best to meet these QOS objectives, but may occasionally fail to satisfy them.

Flow headers are needed at the receiver by the transport layer, but not necessarily by the receiving application. Over the network, transport-level flow headers are not necessary, since QOS is properly provided by IPV6 flow headers. But at the receiver, the transport layer will need flow headers to identify per-flow QOS parameters, like the ADU header size, and the residual reliability criterion (which determines when to generate an acknowledgment to cut off the source's retransmissions). However, the receiving application does not require the flow header. Since a flow is defined by its QOS parameters only, then it is possible for multiple compression algorithms, or perhaps the same compression algorithm operating at different quantization depths, to transmit different forms of encoded data over the same flow. In order to differentiate and correctly decode each distinct data type within the same flow, the destination application would need to agree *a priori* with the source application on an application-level protocol that would label data so that the proper decoding algorithm would be invoked. Hence, application-level identifiers would already provide the functionality to properly extract coded data, and transport-level flow identifiers would not need to be passed across the destination ATI. Since there may still be a reason for passing flow headers to the receiving application, such as reducing labelling overhead, then we do not dogmatically rule out passing flow headers to the application, even though Table 7 would seem to recommend that.

Note that correlation labels can operate across flows, cancelling other flows' data. A hierarchically coded image (say progressive JPEG) may have partitioned its layers into different flows. A correlation label need not be constrained to only annihilate within its parent ADU's flow. For example, suppose a given image is hierarchically coded into three layers I, II, and III, and each layer is transmitted along a separate flow. Suppose also that the three different layers also have three separate correlation labels A,B, and C, respectively. If layer I of each new hierarchically coded image contains all three correlation labels A,B, and C, then the retransmissions for all layers of the previous image will be cancelled across different flows.

Though it may appear from the previous example that there is a one-to-one mapping between source coding layers and flows, this is not necessarily the case. It should be stressed that flows are a QOS concept, not a source coding concept. Therefore, each of the three layers in our hierarchically coded image could have been sent along the same flow, thereby receiving the same level of service from the underlying network protocol.

5.5 Progressive source and channel coding

Our progressively reliable packet protocol can be interpreted as a form of progressive channel coding. As an analogy, progressive image transmission (PIT) constitutes a form of progressive source coding. A natural research question is how the two concepts can be integrated into a single system which achieves the advantages of both techniques in terms of improved interactivity at the cost of high initial distortion, and in terms of eventually low-distortion delivery. The work outlined in this section is a preliminary attempt to address this issue of combined progressive source and channel coding. Specifically, we modified the X server to incorporate a progressive source coding algorithm for images, in addition to the progressively reliable encoder described previously. We implemented a rudimentary progressive colormap algorithm to explore the impact of progressively reliable delivery on progressive source transmission.

5.5.1 Limitations of conventional PIT algorithms

We begin by outlining some of the limitations of traditional PIT algorithms [138] when applied to images that include embedded text/graphics. Again, it should be clear that our modified X server is emulating progressively reliable transmission and delivery of rendered images, and not of plain text nor graphics commands. In general, our interactive image browsing application emulated by the modified X server will display both natural images and images with embedded text/graphics. A PIT algorithm should be chosen so that it can support both types of visual data.

Many PIT algorithms depend on a pyramidal decomposition of the image, performed either in the multiresolution [143], quantization [71], or transform [35] domains. Pyramidal coders produce a multi-layered representation of an image. For example, a "coarse" initial version of an image could be represented by the highest layer in the hierarchy, while the "fine" version of an image could be represented by the lowest layer in the pyramid with the fullest definition of the image. Only a few such pyramidal decompositions are suitable for windows-based graphics, since the initial representation needs to support reasonably legible text. In addition, this initial version needs to be error-tolerant, as discussed previously.

Human users' subjective and objective requirements for windowed graphics form an important consideration in the choice of the PIT technique. Subjective image quality and perceptual delay must be satisfied by any PIT method. In terms of image quality, interactive graphics requires a reasonably high degree of fidelity in the coarse image if PIT is to be successful. This is true because much of the windowing environment is artificially and procedurally generated and rendered. Text is an important component of all such applications. Normally text appears as a simple foreground and background two-color mixture that is sufficiently contrasting to appear legible. Hence text consists of relatively high-frequency information with extremely "important" information content to the human user. Therefore, any PIT technique supporting windowed graphics should also support a coarse representation which must be substantially legible

In view of these characteristics, not all PIT techniques are suitable for windowed graphics. For example, transform-based progressive methods [35][95] like the JPEG algorithm do not support graphics well. Most frequency domain approaches send low frequency information first, resulting in blurry illegible initial versions. JPEG includes two options for progressive transmission of DCT coefficients, namely spectral selection and bit-plane separation. In both cases, observations of the initial coarse reconstructions of progressive JPEG images reveal two phenomena. The low-frequency-only content smears the visual data per block. One alternative could be to reverse the order of coefficient transmission, and communicate high-frequency information first. However, such an approach tailored for text would clearly backfire for natural images, which must also be supported within the windowing milieu.

Progressive multiresolution coding offers an alternative to progressive DCT frequency coding. Multiresolution coding can be defined either in wavelets/subband domain, or spatial resolution domain. Subbands are filtered frequency bands which have been downsampled since they have reduced frequency content [102]. Subbands can be recursively filtered and subsampled, to create a multi-layered hierarchy of low frequency to high frequency subbands. Unfortunately, this type of multiresolution algorithm, while performing well for natural images, poorly supports text legibility when only the lowest frequency coarsest resolution subbands are viewed first.

Spatial subsampling techniques also suffer from the same inability to support legible reproduction of text in the coarse low-resolution version. Progressive pyramidal subsampling schemes in the spatial domain, while designed for natural images, are not well-suited for the highly artificial windows environment [44][71].

5.5.2 Progressive colormap algorithm

Instead, we have settled on a simple spatial domain technique similar to tree-structured VQ [102] which is robust to errors, can favorably depict text, can be applied to both natural and artificial images, and is easily implementable in the X server environment.

5.5.2.1 Description of algorithm

Arguments in favor of spatially-based VQ coding are its relative robustness [127][136], compatibility with the X server's internal representation, and its ability to confer initial text legibility. Colormaps represent a form of per-pixel VQ, since the (Red, Green, Blue), or RGB, color space is quantized into a few color vectors. We create a palette of "coarse" colors, represented by a coarse colormap smaller than the default colormap. For each pixel, the graphics server will initially transmit the index of the closest "coarse" color to the pixel's color. Later, the final set of bits will arrive, specifying one of the full colors in the default colormap. In Figure 5.10, we picture how this progres-

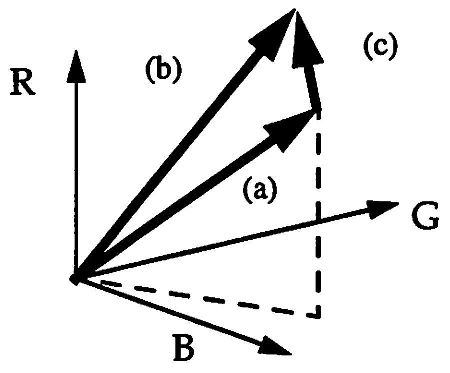


Figure 5.10 Progressive colormap algorithm in RGB space. (a) initial color vector for each pixel (b) final color vector for each pixel (c) displacement vector between initial and final versions.

sive colormap scheme works in RGB color space for each pixel. Since the initial vector could be corrupted by noise, then our assumption was the final colormap value for each pixel would contain the full information vector (b), rather than the displacement vector (c).

Instead of defining a separate "coarse" colormap, the default colormap could be tree-structured, so that the tree's coarse elements inherently define our "coarse" palette. We did not implement this extension. Also, we did not extend pixel-VQ to two-dimensional VQ, which can achieve greater compression without compromising robustness, at some cost in legibility.

In our case, we have chosen a palette of 16 "coarse" colors, or 4 bits of information for the coarse colormap. Early color monitors were limited to approximately 16 colors. Color quantization attempts in the range of approximately 16 colors appear to give subjectively reasonable reconstruction for certain natural and most artificial images [16][45]. Hence the size N_c=16 of the coarse colormap was recognized as the lower bound capable of coarse image definition

The next stage is to properly choose this set of N_c=16 colors. Fundamentally, this involves the issue of color quantization, i.e. choosing a smaller set of colors to represent a much larger color space. A wide variety of algorithms are available, including the frequency and Median Cut algorithm [55], several variance-based methods [146][147], and others [6][93]. Each of these methods seeks to partition RGB space by using a single image's statistics. Many of these techniques are computationally intensive, and hence ill-suited to our latency-constrained environment.

An interesting alternative is to choose the set of coarse colormap data independent of image statistics. Again, a wide variety of algorithms can be designed, many of which take advantage of certain characteristics of the human visual system(HVS). One algorithm proposes the uniform quantization of UVW luminance-chrominance space that is psychovisually uniform, based on HVS characteristics [73]. However, by simple experimentation, we found that a simple heuristically-defined palette of "likely" colors, based upon our accumulated experience of human usage patterns for both natural and artificial images, was subjectively reasonable. This is primarily due to the extremely limited size N_c=16 of the static color palette.

Given a quantized set of colors, the final step in color quantization is the mapping of the original image to the quantized image [146][147]. To do this, each color must be mapped in some sense to its "closest" representative color, which is defined by its tristimulus components or luminance and chrominance components.

In our case, given the size and constitution of the coarse colormap, the final step in color quantization is to map each color in the default colormap to one of the 16 coarse colors. Many techniques choose to minimize over an RGB Euclidean distance measure. It is well-known that equal distances in RGB space do not reflect equal "just noticeable" subjective color differences [97]. Hence, in order to emphasize certain HVS tendencies, a luminance-chrominance color space was chosen as the basis for comparison.

The color metric chosen here minimizes the weighted YUV-distance between a given color and the set of coarse colors:

$$\min_{c=1..16}(0.6|c_Y - i_Y| + 0.2|c_U - i_U| + 0.2|c_V - i_V|)$$

where i is the index into the default colormap, with color components i_Y, i_U, and i_V, and c is the coarse color index. The weighting factors reflect the HVS characteristics of the eye, which is more sensitive to spatial variation in intensity than variation in chromaticity [6]. While emphasizing nearness in luminance, the color metric also attempts to resolve among colors with approximately equivalent luminance.

It has been recognized that this final mapping of original image to quantized image requires a computationally intensive search, especially for large quantization maps [1]. Most of the time in color quantization is taken up by the latter search and remapping phase, and not by the initial quantized color determination phase. In the worst case, each pixel must be compared to all quantized colors before an optimum match can be made. However, the relatively small quantized set size of 16 should decrease search latency considerably, in comparison to the normal search size of 256.

More importantly, this search needs only to be run *once* when the original color is first *allocated* in the default colormap, and thus need not be run for every pixel referencing this color. Observe that the colormap scheme requires all colors to first be allocated before they can be used. Since all further references to the color are by index, then the search need only be conducted once. This is in sharp contrast to the normal problem of mapping a 24-bit RGB image into a 256-color colormap, where each original pixel stores a 24-bit color instead of an index, so that the search must be conducted anew for each pixel. In fact, the colormap scheme defers all of this quantization work to the application. Thus, the colormap scheme allows us to effectively removes the worst contribution to coding latency from color quantization.

The overall drawing process implemented by the X server is encapsulated in a simple lookup table, pictured in Figure 5.11. When a pixel uses a value in the default 256-color colormap, then the X server draws that pixel into the virtual frame buffer, and in addition does a quick lookup to find the index for the 4-bit coarse color. This coarse color is drawn immediately into the physical frame buffer. The effect seen by the end user is a progressive transmission algorithm, within an initial color for each pixel closely followed by a final color. From the source's perspective, the 4-bit coarse color is first transmitted to the receiver, and followed by a later progressive transmission of the final 8-bit color vector.

5.5.2.2 Lessons learned

We constructed a simple two-substream example to gain some intuition about the subjective behavior of asymptotically reliable and progressively coded graphics that trades off reliability, delay and bandwidth across substreams. We modified an X11R6 windows

	Colormap Value	Coarse Color	(R,G,B) Color
Coarse colormap	0	0	$(R,G,B)_0$
	⋮	⋮	⋮
	N_c-1	N_c-1	$(R,G,B)_{Nc-1}$
Rest of the colors in colormap	N_c	[0,N_c-1]	$(R,G,B)_{Nc}$
	⋮	⋮	⋮
	N_f-1	[0,N_c-1]	$(R,G,B)_{Nf-1}$

Figure 5.11 Lookup table that implements progressive colormap algorithm. For each pixel, the X server saves the actual colormap value in the virtual frame buffer, and writes the coarse colormap value into the physical frame buffer. At some later time, the full colormap value stored in the *vfb* is sent through the progressively reliable encoder to be displayed on screen. Let N_c = # coarse colors, N_f = # total colors. In our implementation, the coarse colormap is embedded within the N_f (=256)-color colormap, occupying the first N_c positions.

server to implement the progressive colormap algorithm and asymptotic reliability using two substreams. A simulated channel introduces errors and constrains the bandwidth.

We experimented with several depths of initial colormap size. First, we found that coarse colormaps of 16 colors were ill-suited for progressive transmission of natural images. The coverage of the 16 colors over RGB space was too sparse to permit accurate representation of both luminance, and hue/chrominance color components. Consequently, both the luminance and color of the image would change between the initial and final representation. As a compromise, we fixed the coarse colormap to only allow grayscale values. This would limit the change between initial and final versions of an image to only a color change, rather than a luminance change as well. We observed that this change significantly improved our PIT algorithm.

We also experimented with a 1-bit/2-color black and white coarse colormap. We found that text could be easily reproduced in legible form by this two-tone initial colormap. This is because the foreground and background colors of a text-based application are usually chosen by the user to appear as highly contrasted, i.e. the luminance difference between text and background is extreme. While natural images were less well reproduced, by and large the essential subjective features of most such images were still easily discernible. By expanding to 2-4 bit grayscale coarse colormaps, the coarse versions of natural images were able to start conveying fine detail in addition to coarse detail reasonably well.

Interactivity of the progressive server at 100 kbit/s was much improved over the normal X server. Bursty activity like menu pulldown is conveyed quickly, thanks to the low-quantization black and white initial version. We modified a second X server to incorporate rate-limiting only, with no progressivity either in the source or channel coding domains. This "normal" X server rate-limited at the same channel capacity experienced significant interactivity problems by trying to send the fully quantized color version. At higher channel capacities exceeding 1 Mbit/s, the improvement in interactivity was less noticeable.

Unreliability of the initial image was tolerable despite a severe *BER*. Even text-based images tolerated the noise, thanks to the robustness of the progressive colormap scheme. We found that the pixel error rate at the limit of text legibility for noisy 12-point text was about 3%. However, this legibility threshold depends on pixel resolution, pixel depth, and text size, in addition to the channel noise, so it should not taken as an absolute. It was derived by corrupting pixels for an 8-bit ~100 dpi display until a normal shell terminal's 12-point text was no longer readable. Natural images could tolerate higher *BER*'s.

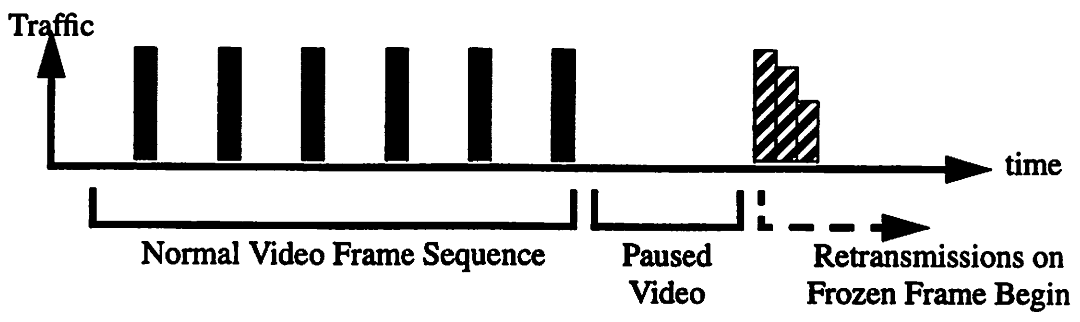
We found that the end user soon became accustomed to the transition effect from initial to final image. The most pronounced effect occurs for heavy quantization, i.e. the monochrome to color transition. We found this crude system usable, though certainly not ideal. Clearly, more clever PIT algorithms could mitigate this transition effect. We also tried grayscale "coarse" colormaps, which reduced the transition effect. Color-based "coarse" colormaps were found to perform slightly worse subjectively than their grayscale counterparts. We believe this to be a consequence both of the eye's greater sensitivity to luminance than chrominance (plus our mapping metric's biases toward a luminance match) and of the lack of dynamicity in the "coarse" colormap.

Another conclusion we came to is that two substreams insufficiently partition the graphics source. For example, we ran mpeg_play to see how video performs. We observed a pronounced *annihilation effect* due to the 5 second minimum delay -- the reliable version is never sent thanks to the fast overwriting action of the video stream. However, we were forced to watch a black and white video presentation indefinitely! Clearly, the progressive colormap algorithm should not be applied uniformly to all applications' data. To avoid this, in the future we believe that data-dependent or application-dependent partitioning of the graphics stream is needed, which is one of the motivating reasons for the development of the flow labels described in Section 5.4.3.

To experiment with application-dependent source coding within the X server, we included two hooks into the X server through the Tcl-based performance manager. First, in Figure 5.6 we show that a "separate image coding" button is incorporated into the performance manager. With this button, we were able to turn off progressive colormap coding for video sequences, and natural images in general. For example, we could edit text in one window, which would be sent in progressive colormap fashion, while the video in another window would be playing at full color depth. The overall effect was far more pleasing than applying a single coding scheme to all applications' data.

In addition, we also incorporated a "progressive cursor" button in the performance manager in order to remedy another subjective artifact introduced by uniform progressive colormap coding. We found that if progressive colormap coding is applied to cursor redrawing, then as the mouse cursor is moved across the screen, it leaves not only a noise trail due to progressive reliability (which produced the "snake" effect described in Section 5.4.1), but it also leaves a much more objectionable source coding trail. In the case of a black and white coarse colormap, the blocks redrawn in black and white due to the passing of the mouse cursor stood out in stark contrast to the surround color versions. To fix this problem, we added state in the progressive encoder of the X server to detect when a cursor was being drawn. If a mouse cursor was drawn into an affected block, then the previous state was used to decide what depth the mouse cursor should be drawn at. If the previous block state was "sent final data", then we know that the final color version is being displayed for that block at the receiver. In this case, the mouse cursor is drawn a full colormap depth rather than coarse colormap depth. If instead the previous block state was "sent initial data", then we assume that the receiver is viewing a coarse colormap version of that block, and therefore the least objectionable approach is to redraw the cursor-affected block at coarse colormap depth. The overall effect is that the cursor again leaves only a noise trail, and no longer leaves a black and white, or grayscale, trail in its wake. If the cursor is viewed as a separate application, then we see again that application-dependent coding can reap large subjective benefits in terms of perceived quality.

Finally, we note that our progressive colormap scheme is quite simple, and that more sophisticated progressive source coding schemes are possible that reproduce both embedded text and natural images well, tolerate errors well, and achieve a larger degree of compression in their first version. For example, we did not experiment in-depth with wavelet image bases which might be able to capture text and natural images well. The objective would be to find a transform that packs much of its energy into high and low frequency components, but not much into middle frequency components.



Traffic
time
Normal Video Frame Sequence
Paused Video
Retransmissions on Frozen Frame Begin

Figure 5.12 Video is delivered over a progressive reliable protocol. While video is streaming continuously, progressive retransmissions are shut off by combining delayed retransmission redundancy with the ADU cancellation function. Each new frame effectively cancels any attempt to retransmit the previous frame's data. When video is suddenly paused, the progressively reliable protocol naturally kicks in its retransmissions to clean up artifacts on the final frozen frame.

5.6 Other application examples

In this section, we consider how continuous multimedia applications like video and audio can parameterize our progressively reliable protocol to deliver their data sequences. While progressively reliable packet delivery was designed with bursty multimedia in mind, both video and audio can make use of two of progressive reliability's functions, controlling when retransmissions are sent (Section 5.4.1) and whether retransmissions are initiated at all (Section 5.4.2).

5.6.1 Video

Video exhibits a natural cancellation property. As each new frame is displayed, the previously visible frame is overwritten. We observe that a progressively reliable protocol which implements cancellation of stale ADU's and delayed retransmission of ADU redundancy could be parameterized to deliver video unreliably without having to invoke progressive retransmissions on each frame. Consider the example shown in Figure 5.12. Video is passed to the protocol as a sequence of image frames. Suppose the video application tells the protocol to delay retransmissions on a given frame by about 3 frame times, or equivalently about 100 ms. Suppose that the video application subdivides its image into blocks, and identifies each block in a frame by a separate correlation label (defined in Section 5.4.2). Since a new video frame arrives every 30 ms, then the correlation labels cause each new block associated with the next frame to cancel retransmissions of each corresponding block in the old frame. Integrated over all blocks, each new frame effectively cancels retransmissions of the old frame. In this way, a progressively reliable protocol can be parameterized to shut off progressive retransmissions and deliver video without requiring each noisy frame to be cleaned up by reliable follow-up redundancy.

Many video applications allow the user to pause the video in order to capture and edit/composite a single image. As soon as the video is paused, retransmissions will no longer be cancelled by the next frame. Following the minimum wait time, which is 100 ms in our example, progressive retransmissions will automatically be initiated to clean up the artifacts in the freeze-frame image, as shown in Figure 5.12. The application need not establish a separate TCP connection to transmit the fully reliable freeze-frame image. Thus, when a video application sends a continuous sequence of images, the progressively reliable protocol behaves like an unreliable datagram protocol and suppresses retransmissions. When the video application is paused, the progressively reliable protocol behaves like a reliable protocol and employs retransmissions to deliver a reliable version of the freeze-frame image.

5.6.2 Audio

The cancellation and delayed retransmission functions of our progressively reliable protocol can also be parameterized to deliver continuous media audio. The intent is to customize the progressively reliable protocol to partially retransmit audio until the desired latency bound expires[1], e.g. due to conversational interaction. In some connections, the

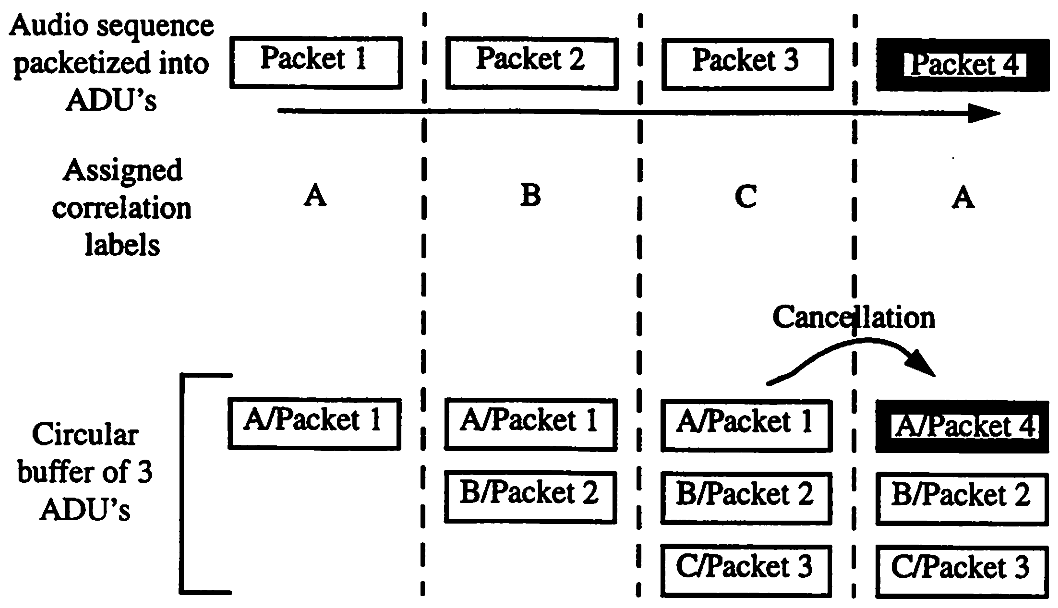

Audio sequence packetized into ADU's
Packet 1
Packet 2
Packet 3
Packet 4
Assigned correlation labels
A
B
C
A
Cancellation
Circular buffer of 3 ADU's
A/Packet 1
B/Packet 2
C/Packet 3
A/Packet 4

Figure 5.13 Circular cancellation of audio packets. The progressively reliable protocol attempts retransmissions for each audio packet up until the conversational latency bound is exceeded. Suppose that by the time packet 4 is ready to be transmitted, any further retransmissions of packet 1 will arrive too late at the receiver. Therefore, packet 4 can be used to cancel retransmissions of packet 1, and packet 5 to cancel retransmissions of packet 2, and so on in circular fashion. The protocol is parameterized so that retransmissions of an audio packet are sent immediately, rather than being delayed as for interactive image browsing.

propagation and queueing delays may still be small enough to leave slack in the latency budget for several end-to-end retransmission attempts.

In Figure 5.13, we illustrate how circular annihilation of progressively reliable packetized audio works. Consider an audio sequence of four numbered packets or ADU's. Suppose that by the time packet four is ready to be sent that any further retransmissions of packet one will arrive too late at the receiver, i.e. after the desired latency bound has been exceeded. The time interval during which partial retransmissions can be attempted is

1. This idea of using a circular cancellation scheme for audio was proposed by Sanjoy Paul, currently at Lucent Bell Labs, in a conversation with the author.

therefore three packet lengths in our example. Within this window of three packets, each new audio packet displaces the oldest of the three queued packets, cancelling retransmissions associated with the oldest packet. For example, packet four will be labelled by the application with the same correlation label *A* as packet one, packet five with the same correlation label B as packet two, etc. This enables the protocol to stop retransmitting packet one(two) as soon as packet four(five) has been passed to it through the transport service interface. Cancellation proceeds in a circular fashion through the correlation labels, transitioning from *A*->*B*->*C*->*A*->*B*->...

The same approach described here for audio can also be applied to partially retransmit a frame of video. However, unlike the example of video in the previous section, retransmissions in this section are tightly concentrated within a couple hundred millisecond interval, and the cancellation function is applied over multiple frames spread in time. By analogy with the audio example above, video frame four would cancel video frame one. In contrast, the previous section delivered video by parameterizing progressive reliability to delay retransmissions by many seconds, and cancellation was employed for each successive frame, i.e. frame two cancels frame one, frame three cancels frame two, etc. The two examples recounted here illustrate how parameterization of a progressively reliable protocol allows multiple applications to flexibly adapt this protocol to suit their individual needs.

5.7 Summary and conclusions

In this chapter, we have discussed the following: our definition of progressively reliable packet delivery; the fundamental properties which characterize a progressively reliable delivery service; our modified X windows server that emulates in real-time a crude version of progressive reliability; three added-value functions which improve the latency performance of a progressively reliable protocol; and examples of how multimedia applications (image, video, voice) could exploit a progressively reliable transport service.

Progressively reliable packet delivery delivers an initial possibly corrupt version of a packet to the receiver immediately. Later, successively refined versions of that packet are delivered with statistically fewer and fewer errors. An interactive Web-based image browser would exploit progressively reliable packet delivery to display an initial possibly noisy version of an image. Later, the displayed image would gradually be cleaned of any persistent artifacts caused by channel noise.

There are four key properties to a progressively reliable service/socket interface as seen by the application. First, corrupt information is forwarded to the application. This requires that the application frame its data into application data units (ADU's). Further, to permit forwarding of ADU's with corrupt payloads, there need to be separate error detection checksums for the ADU header and ADU payload. Second, multiple versions of each ADU are forwarded. Third, these multiple versions have the property that they are monotonically increasing in reliability, i.e. each successive ADU version is guaranteed to have statistically fewer errors than previous versions. Fourth, the application must be able to tolerate out-of-order delivery of different ADU's.

We modified an X windows server to implement progressive reliability in real-time. We also created a Tcl-based "performance manager" tool which could dynamically adjust a variety of delay, bandwidth, and *BER* parameters as the X server was running.

Through experimentation, we discovered three additional functions which can be incorporated into the socket interface to help parameterize the underlying progressively reliable protocol and improve its latency performance. First, retransmissions of an ADU that are used to provide successive refinement can be delayed so that they don't conflict with initial delivery of an ADU. Subjectively, once the end user has a noisy initial image version with which to interact, then the final reliable version of that image can be retransmitted with a more relaxed delay objective, on the order of many seconds. This flexibility can be exploited by our protocol to give priority to delivering the delay-sensitive first version of an ADU. Second, retransmissions that have been delayed by the protocol can also be cancelled if they are associated with out-of-date image data. This unique feature permits an application sender to stop the protocol from retransmitting old image data that is no longer useful to the receiver. We propose the use of "correlation labels" to implement cancellation of stale ADU data. Third, we propose that applications be allowed to subdivide their data into multiple flows, which are then scheduled differently according to their delay and reliability quality-of-service (QOS) parameters by a progressively reliable protocol. Flow-based scheduling is a natural extension of progressive reliability, which inherently makes a two-flow scheduling distinction between the virtual flow of initial delay-sensitive ADU versions and the virtual flow of final relaxed-delay ADU retransmissions.

Finally, we describe several examples of how multimedia applications can take advantage of progressively reliable packet delivery. Progressive image transmission (i.e. progressive source coding) can be combined with progressively reliable packet delivery (i.e. progressive channel coding), and we describe a simple progressive colormap algorithm which we implemented within the X server. In addition, video applications can make use of the cancellation function to allow each succeeding frame to cancel retransmissions of data associated with previous frames. Finally, audio applications can make use of the cancellation function to create a circular buffer, which partially retransmits packets within a sliding window of eligible packets.

In the next chapter, we discuss some issues related to implementing a progressively reliable end-to-end protocol, which we call Leaky ARQ.

6

Leaky ARQ Implementation Issues

In this chapter, our intent is to consider how a progressively reliable transport protocol could be implemented to support the progressively reliable transport services described in the previous chapter. The survey of implementation issues presented here is not intended to be a rigorous treatment of the topic. Instead, specific implementation options are targetted and some of the trade-offs associated with these options are outlined. In Section 6.1, we introduce Leaky ARQ and outline the modifications to a traditional retransmission-based ARQ protocol needed to support the essential properties and added-value functions of the progressively reliable transport service interface. In Section 6.2, we consider how progressive reliability, particularly the successive refinement property, can be implemented as a true transport protocol by appropriately modifying Hybrid FEC/ARQ protocols. In Section 6.3, we detail some of the advantages and difficulties of implementing progressive reliability as an end-to-end protocol lying above TCP and UDP in the Internet stack. Finally, we consider other implementation issues such as fragmentation of ADU's, non-sliding windows, and acknowledgments.

6.1 Leaky ARQ: a progressively reliable end-to-end protocol

Recall from the previous chapter that the essential properties that a progressively reliable end-to-end protocol must support at the transport service interface are:

- Corrupt ADU's are forwarded to the application
- Multiple versions of each ADU are delivered
- The reliability of these multiple ADU versions improves over time (fewer errors with each successive version)
- Different ADU's are delivered out of order

In addition, the progressively reliable protocol should support the three following performance-enhancing functions:

- delaying ADU retransmissions such that conflict with delivery of the initial version of an ADU is minimized
- cancelling retransmissions associated with out-of-date ADU's
- flow-based scheduling of the sender's traffic by delay and reliability constraints

A retransmission-based ARQ protocol can be modified at the receiver to "leak" forward corrupt ADU's. We call such a protocol "Leaky ARQ" [51]. The retransmissions contain the reliable redundancy which allow Leaky ARQ to improve the reliability of the delivered data over successive versions. At the sender, Leaky ARQ needs to be modified to support delaying of retransmissions, cancellation of retransmissions, and scheduling of multi-flow application data. Figure 6.1 summarizes each of the major modifications that need to be made to a traditional ARQ protocol in order to support progressively reliable packet delivery. In order to support unordered delivery of packets, cancellation of specific data, and successive refinement of packets, then the internal structure of Leaky ARQ should be implemented as an acknowledged datagram protocol.

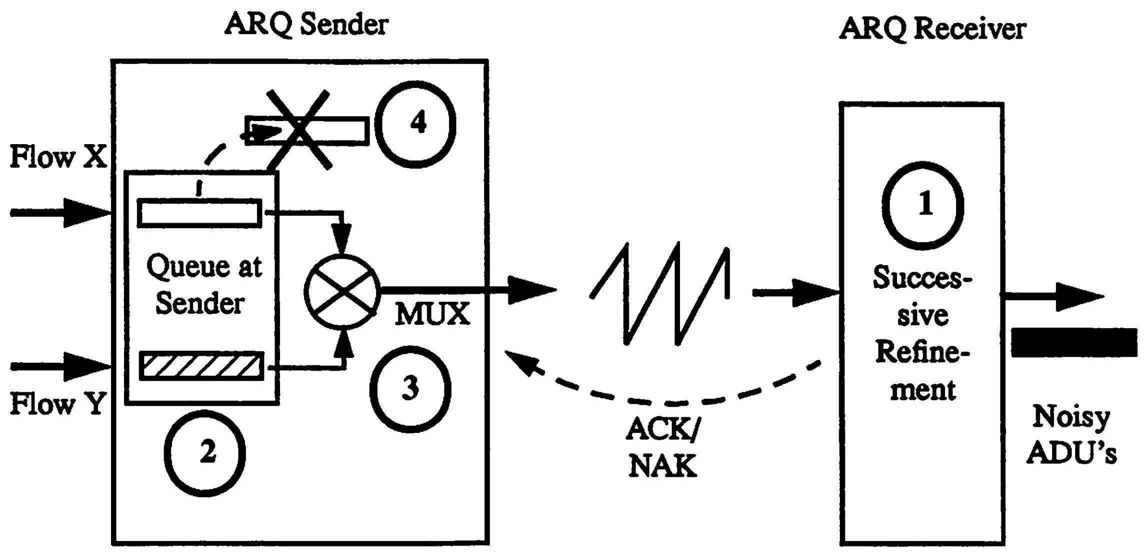


Figure 6.1 **Leaky ARQ is a progressively reliable packet delivery protocol that modifies a traditional ARQ protocol in four ways: (1) at the receiver, noisy packets are initially forwarded to the multimedia application and then successively refined to remove errors (2) retransmissions are delayed at the sender's queue, while initial transmissions are sent immediately (3) multiple flows are scheduled with different delay objectives (4) out-of-date retransmissions can be annihilated at the sender.**

6.2 Implementing Leaky ARQ as a hybrid FEC/ARQ transport protocol

In this section, we consider implementing Leaky ARQ as a true transport protocol, i.e. as a third alternative to either TCP or UDP for the Internet. Implementation as a true transport protocol gives us the flexibility to employ a sophisticated hybrid FEC/ARQ scheme for implementing the successive refinement property. Hybrid FEC/ARQ protocols combine a retransmission-based ARQ protocol with FEC, either in the form of linear block codes or convolutional codes. In the following subsections, we discuss Type-I Hybrid ARQ, adaptive Type-I Hybrid ARQ, and Type-II Hybrid ARQ.

Hybrid FEC/ARQ protocols are applied in situations where only one form of error protection, e.g. FEC or ARQ, by itself would be insufficient to meet the end user's reliability and delay objectives. In the absence of FEC, a retransmission-based network ARQ protocol can take too long to reliably deliver an error-free packet to the destination. We quantified in Chapter 2 the minimum latency introduced by an ideal protocol over a noisy wireless link and showed that the delivery time for an error-free packetized image far exceeded the interactive latency bound of 100 ms. Similarly, in the absence of ARQ-type feedback, open-loop FEC can introduce considerable overhead in order to adequately protect an image over a noisy wireless link. As we showed in Chapter 3, fixed open-loop FEC can double or triple the bandwidth. Moreover, practical limitations on the complexity of FEC limit its effectiveness at high *BER*'s. A hybrid FEC/ARQ protocol can achieve higher throughput and/or lower overhead compared to ARQ or FEC operating alone. In Table 8,

Table 8. A summary of digital error protection techniques

	Description	Examples
Conventional ARQ	Retransmit lost packets	SRP, GBN, SW, TCP
Open-loop FEC	Linear block codes, convolutional codes	Reed-Solomon codes
Stutter ARQ	Conventional ARQ + multiple transmissions of unacknowledged packets	see Chapter 2 references
Type-I Hybrid ARQ	Fixed FEC within a conventional ARQ retransmission loop	TCP over digital cellular with fixed convolutional coding
Adaptive Type-I Hybrid ARQ	Adaptive FEC within a conventional ARQ retransmission loop	see Chapter 6 references
Type-II Hybrid ARQ (also called Memory ARQ or Packet Combining)	At sender, initially transmit packet with little/no FEC, only send FEC redundancy if original packet was corrupted. At receiver, save noisy packets and decode/error correct by combining previous history of noisy received packets.	see Chapter 6 references

we summarize in advance the major forms of digital error protection surveyed so far, and also those to be explained later in this section. These comprise the menu of error protections options available for the design of Leaky ARQ. We shall see that Leaky ARQ can be interpreted as a modified form of Type-II Hybrid ARQ.

6.2.1 Limitations of Type-I Hybrid ARQ over time-varying channels

A Type-I Hybrid ARQ protocol is defined as having a FEC coder/decoder pair with fixed coding rate embedded within the retransmission loop of an ARQ protocol. For example, if an end-to-end ARQ protocol like TCP [25] is employed over an IS-54 digital cellular wireless link that has implemented a fixed rate $\frac{1}{2}$ convolutional FEC code [36], then together this TCP-FEC combination conceptually forms a Type-I Hybrid ARQ protocol, even though the FEC and ARQ were designed separately and operate at different layers. For T-I H-ARQ protocols, we showed in Chapter 3 that header overhead and FEC redundancy can together double the bandwidth under certain assumptions, though FEC redundancy alone only added about 15% redundancy. However, we pointed out several optimistic assumptions in our derivation that would make it more difficult for T-I H-ARQ protocols to meet the interactive latency bound, either because of protocol inefficiencies, increased overhead when non-optimal binary codes are considered, or severe *BER*'s that cause the FEC to fail.

A potentially more objectionable problem with T-I H-ARQ is its relative inefficiency over non-stationary fading wireless channels with time-varying probabilities of bit error. Since the coding rate is fixed for T-I H-ARQ protocols, then when the *BER* is too low, the FEC overhead is excessive and unnecessary. When the *BER* is too high, the error correction is insufficient. For example, practical digital cellular systems design the error correction to protect against heavy *BER*'s, resulting in a doubling and even tripling of the number of transmitted bits due to FEC overhead [29][36][101]. The fixed overhead will be wasteful when the channel is in a relatively benign low *BER* state, which is often the case [78]. If the protocol designer tries to compromise and reduce the overhead cost suffered during good channel conditions, then the code will fail to sufficiently protect the data during most wireless fades. Once FEC fails, then the repetition-based ARQ protocol will also fail because most retransmissions will be corrupted by bit errors [8]. Under these conditions, reliable packet delivery will suffer the exponential delays which we observed in Chapter 2.

To help analyze the performance of T-I H-ARQ protocols over time-varying channels, researchers have developed stochastic channel models. Real-world wireless channels exhibit correlated or bursty errors, and hence exhibit a form of memory, unlike the random memoryless BSC. The basic Gilbert-Elliott two-state Markov model assumes that the wireless channel transitions back and forth between a "good *BER*" state and a "bad *BER*" state. The transition probabilities control how often a given state is entered and exited. More sophisticated multi-state Markov models [21][42], as well as other channel characterizations like "gap error" models, have also been developed based on empirical measurements of different types of wireless links, and are summarized in [67]. Previous work has demonstrated that Markov-modeled channels are much better at predicting the probability of decoding error for open-loop FEC block codes over actual wireless channels than the memoryless BSC model [34][92]. Similarly, Markov models have been applied in the study of the throughput performance of non-hybrid ARQ protocols like ideal SRP, GBN, and SW over channels with memory [19][48][81].

The performance of T-I H-ARQ protocols in bursty error wireless environments has been investigated by a number of authors [43][52][59][61][150]. Each of these studies differ in their choice of ARQ protocol, FEC type, underlying analog modulation technique, and assumed channel fading behavior. The general consensus appears to be that a T-I H-

ARQ protocol can outperform ARQ or FEC operating alone, but is still subject to the overhead problem mentioned above.

6.2.2 Adaptive Type-I Hybrid ARQ

Recognizing the fixed overhead costs suffered by a T-I H-ARQ scheme when the channel is relatively clean, several authors have proposed heuristics to adapt the depth of FEC channel coding to the level of noise, or the probability of bit error, posed by the time-varying channel [40][66][108][144]. Key design parameters of adaptive T-I H-ARQ protocols include what metric(s) (e.g. *BER*, analog received signal strength indicator (RSSI), analog signal-to-noise rate (SNR)) to use for adaptation, the granularity of adaptation rate, the states of the metric that trigger adaptation, and the level of FEC overhead that is applied for a given value of the metric. While such adaptive hybrid protocols hold great promise, there are some practical limitations to consider. A partitioned communications system design may hide the physical layer's received analog "soft" symbol values from higher layer "hard-decision" bit decoding. Even if the analog RSSI/SNR values are known to the upper layers, the channel may be changing too rapidly for these values to represent the current state of the channel. Also, the raw *BER* of the channel may be difficult to infer from the received bit stream, because failure of FEC decoders can cause the number of errors to be unknown in a packet, or even worse can magnify the number of errors in a packet. One author has proposed deriving the *BER* from correctly decoded packets only, and ignoring incorrectly decoded packets with too many errors. Such a metric is precise only up to the *BER* failure point of the error correcting code [118].

6.2.3 Type-II Hybrid ARQ incorporates variable-rate FEC

The progressively reliable protocol design is arguably best accomplished by basing it on another type of adaptive hybrid ARQ protocol called the Type-II Hybrid ARQ protocol [78]. In this type of scheme, redundancy is sent *incrementally* per packet as dictated by the channel noise conditions. At the sender, the original packet is transmitted with little or no FEC. If the original packet cannot be decoded free from errors, then the sender transmits more FEC redundancy bits, rather than simply repeating the original packet as practiced by conventional ARQ. At the receiver, noisy packet versions are saved rather than discarded. The possibly noisy FEC redundancy is *combined* with the noisy original packet in order to decode the packet. Each time the decoded version is in error, incrementally more redundancy is conveyed to the receiver, which continues to store a history of received payload and error correction bits in order to successively refine the decoded packet.

By sending FEC redundancy only when the channel *BER* is sufficiently severe, the T-II H-ARQ approach is able to adapt to time-varying channel conditions, and therefore is substantially more efficient than a fixed FEC T-I H-ARQ protocol. In addition, caching of a packet's received history for packet combining further improves the performance of T-II H-ARQ by reducing the number of retransmissions needed to decode a given packet. Another outcome of packet combining is that the most recent estimate of a packet's transmitted bits is guaranteed to be statistically less error-prone than previous estimates, at the cost of greater memory and processing at the receiver, a characteristic we will use to implement progressive reliable delivery of images. The T-II H-ARQ approach is also called *memory ARQ* [86][120], and/or *packet combining*.

A well-known example of T-II H-ARQ is Lin & Yu's protocol, which alternates between sending the original packet (say P_1) of length N bits and the N error correction bits (say P_2) generated by a $(2N,N)$ linear block code [78]. The receiver either caches a corrupted P_1 and then uses the newly arrived P_2 to correct the cached P_1, or caches a corrupted P_2 to correct the next P_1. This approach has been generalized to cache and combine more than two versions of a packet. Generalized *maximum-likelihood code combining* of multiple cached packets is described in the literature [14][15].

Both of the aforementioned techniques only consider the special case where the retransmissions containing the FEC redundancy are all the same length as the original payload. An alternative is to send FEC redundancy incrementally at granularities other than strictly the payload length. Rate-compatible punctured convolutional (RCPC) codes support incrementally redundant transmission and maximum-likelihood code recombination and have been proposed for use in T-II H-ARQ protocols [49]. In addition to supporting incremental transmission, RCPC codes also support unequal error protection (UEP), i.e. different sections within a packet may be protected with varying degrees of error correction. Moreover, the diverse range of error correction supported by RCPC codes can be generated by a single encoder and a single decoder. A mixed approach which integrates payload-length repetition with incremental redundancy was proposed in [65]. The performance of several T-II H-ARQ schemes over Markov-modeled channels has been studied in [63][91][112].

Another form of packet combining called *time diversity combining* that is less powerful than code combining has also been discussed in the literature [151]. In time diversity combining, decoding of a given packet from multiple cached versions is performed on a bit-by-bit or symbol-by-symbol basis. For example, if there are three cached versions of a packet at the receiver, then conceptually each bit can be thought of as having been encoded with a (3,1) error correcting code, i.e. the redundancy is limited to error correcting a specific bit or symbol. In comparison, code combining permits redundancy bits to apply across all bits in the original packet. For the same percentage of overhead, the conceptually simpler time diversity combining is weaker than code combining. An example of digital "hard-decision" time diversity combining is majority-logic bit-by-bit decoding. Suppose that all retransmissions are the same length as the original packet, and that the receiver stores an odd number of received versions of each packet. For each bit, a majority-logic decoder will decode the value 0 or 1 that has the most occurrences, i.e. that is in the majority [12]. An example of an analog "soft-decision" time diversity combiner would be computing a weighted energy average representation for each symbol in a packet. That is, given multiple received copies of a packet stored in terms of their analog received symbol values, compute an averaged representation of each symbol in a packet. This averaged analog packet representation can then be fed into a maximum-likelihood soft-decision Viterbi decoder [53]. An advantage of this analog technique is that the post-decoding output will not fail catastrophically and magnify errors, as would occur for hard-decision decoding of non-systematic linear block codes. A comparison of the efficiency of diversity combining vs. code combining is found in [66] for the case of repetition coding with multiple copy decoding.

Note that both time-diversity combining and code combining algorithms which do not preserve the complete history of received packets will be unable to guarantee statistically improving reliability. As discussed in Section 5.2.3, a necessary condition for successive refinement requires that the entire history used to decode previous versions of an ADU be preserved so that subsequent ADU versions can be decoded with increasing reliability. For example, Lin & Yu's T-II H-ARQ alternating parity protocol only preserves the most recent corrupt packet at the receiver, and is therefore unable to strictly guarantee improving reliability. Unfortunately, the caching required to retain the entire history can lead to an unbounded need for buffer space at the receiver. As a practical compromise, limited memory at the receiver may not be able to strictly ensure monotonically increasing reliability, but for engineering purposes the property of increasing reliability can be closely enough approximated.

T-II H-ARQ schemes differ from adaptive T-I H-ARQ protocols in three areas: whether receiver memory is used for packet combining; the metric used for adaptation; and the rate of FEC adaptation. A common characteristic of all T-II H-ARQ protocols, whether they apply code combining or time diversity combining, is that noisy packets are saved and the received history is used to enhance the power and efficiency of error decoding. Adaptive T-I H-ARQ schemes do not employ this kind of memory at the receiver.

T-II H-ARQ methods also use a very simple error detection metric to estimate the channel state; whether a packet has been detected in error or not is the only channel information used by the protocol to adapt the level of FEC. In comparison, adaptive T-I H-ARQ techniques can measure the channel state by a variety of more sophisticated analog (SNR, RSSI) or digital (*BER* within a time window) metrics. Yet, the error detection metric can be quite effective when used in conjunction with memory at both the transmitter and receiver of a T-II H-ARQ protocol. The sender remembers how many retransmissions of a packet have occurred and how much FEC redundancy has been incorporated in these retransmissions. The sender can integrate this memory with the feedback from the receiver about whether the latest version of the reconstructed packet still has errors to adjust the incremental amount of redundancy incorporated in the next retransmission. Combining error detection with memory is arguably as effective a metric as using a sophisticated channel estimator without memory. Moreover, from a practical perspective, error detection is considerably easier to implement than other analog SNR/RSSI or windowed *BER* metrics.

T-II H-ARQ protocols adapt the level of FEC on a per-packet granularity. Both the channel estimation (error detection) and the decision whether to send more/no FEC are carried out each time a packet is transmitted. In contrast, adaptive T-I H-ARQ schemes have a great deal of latitude in choosing how often to estimate the channel state, under what conditions to change the level of FEC, and to what degree FEC should be increased/decreased. Most proposals for adaptive T-I H-ARQ that we listed in the previous section continuously update the channel state estimate for each transmitted packet, but differ on the criteria and depth for adaptation.

Performance comparisons of adaptive T-I H-ARQ and T-II H-ARQ protocols are available in [64][111], which show that incrementally redundant transmission plus code combining help T-II H-ARQ outperform even adaptive T-I H-ARQ protocols in terms of throughput. Table 8 summarizes the different forms of digital error protection that we have reviewed so far.

Leaky ARQ can be implemented by modifying any of the time-diversity combining, maximum-likelihood code combining, and T-II H-ARQ schemes described above at the receiver to forward noisy intermediate representations of a packet to the application. Such memory ARQ protocols naturally refine the packets stored at the receiver. Rather than hiding these intermediate representations from the application, these memory ARQ protocols can be modified to forward intermediate noisy representations of packet data. Delivery of multiple noisy yet increasingly reliable versions of a packet is a natural outcome obtained by leveraging off of the successive refinement implemented by packet combining. In addition to modifying the receiver to forward corrupt information, the memory ARQ sender can also be modified to implement delayed retransmissions, cancellation of out-of-date data, and flow-based scheduling. In this way, T-II H-ARQ protocols represent a useful template for constructing our Leaky ARQ progressively reliable protocol.

6.3 Implementing Leaky ARQ above the transport layer

While implementing Leaky ARQ as a true transport protocol ideally maximizes its performance potential, the reality of propagating a new transport protocol throughout the Internet may force Leaky ARQ to be implemented as an application-level protocol built on top of UDP and/or TCP. We discuss in the following subsections some of the disadvantages of layering Leaky ARQ on top of UDP and/or TCP.

6.3.1 Limitations of the combination of TCP and UDP

An end-to-end progressively reliable protocol can be layered on top of the combination of TCP and UDP. The initial version of an image can be sent via UDP, and a reliable version can be transmitted at some later time via TCP. Unfortunately, once a packet is sent to TCP, there is no way to stop TCP from sending that packet reliably to the receiver. This poses two problems: we cannot stop TCP retransmissions from conflicting with the more urgent delivery of the UDP image data (for immediate interactivity); and we cannot stop retransmissions of *out-dated* image data. First, retransmissions of TCP packets will uncontrollably steal bandwidth from newly arrived UDP packets, increasing the delay of time-sensitive UDP data. Hence, the combination of TCP and UDP cannot promise consistent interactivity. Second, TCP can waste scarce wireless resources trying to reliably transport *out-dated* information. Web-based image browsers and remotely rendered bitmapped editors like Framemaker overwrite the same screen location frequently with newly delivered image data, so that visual data can become stale or out-of-date. After rapid image browsing or paging/scrolling, the protocol's buffers may contain a sizable backlog of stale image data awaiting reliable delivery, especially if the channel is in a long fade. TCP does not permit us to identify and "cancel" this stale redundancy, i.e. to stop further retransmissions. Our protocol remedies both of these problems by cancelling stale retransmissions, and by rescheduling redundancy/retransmissions so as not to conflict with initial delivery of data. Finally, we note that TCP only provides full reliability. Image applications may be able to tolerate a small degree of lasting residual errors. Therefore, our protocol should support less than fully reliable asymptotic service. As we noted in Chapter 5, retransmissions can be terminated after a finite number of trials, or a finite amount of time.

6.3.2 Limitations of UDP

To avoid the limitations of TCP outlined in the previous subsection, an application-level end-to-end protocol could implement progressive reliability entirely on top of UDP, much like the application-level Real-Time Protocol (RTP) used by multicast video tools [83]. However, UDP checksumming will have to be turned off in order to support forwarding of corrupt packet payloads. Once UDP checksums are turned off, then an application-level protocol will have great difficulty confirming the validity of the IP and UDP headers for received datagrams.

The UDP checksum is computed over the IP pseudo header plus the entire UDP datagram (UDP header plus UDP payload) [25]. IP routers often don't calculate checksums; IPv6 doesn't even have IP-level checksums! [57]. Normally, UDP checksumming is relied upon to eliminate packets that may have been accidentally routed to the application after corruption of their IP and/or UDP headers. The application-level protocol that receives UDP datagrams won't have access to the source IP address or the source UDP address (the destination IP address and destination UDP ports can be inferred). Once UDP checksumming is turned off, then the application won't be able to verify whether the datagram has suffered IP and/or UDP header corruption, even if the application-level protocol implements its own error detection policy.

A solution practiced by designers of UDP is to incorporate the IP pseudo header into checksum calculations, thereby violating Internet layering principles. An analogous solution for an application-level protocol would require UDP to forward both the IP pseudo header as well as the UDP header to the application-level protocol for header verification. A final difficulty is that once UDP is modified to turn off checksumming, other applications using UDP may not expect corrupt packet delivery, even though this is a clearly specified option of UDP. Some poorly designed UDP applications may have built-in assumptions that tolerate out-of-order and lost UDP datagrams, but because they assume

UDP checksumming, they do not tolerate bit corruption within UDP payloads. Note that building progressive reliability on top of the combination of TCP and UDP would suffer many of these same problems.

A new version of UDP, dubbed "UDP-lite", is in the process of development[1] at the time of writing this dissertation. In this version of UDP, the UDP checksum is only calculated on the UDP and IP headers. The UDP payload is not included in the checksum calculation. A progressively reliable transport protocol could be built on top of UDP-lite, since only header-valid payloads will be forwarded from UDP-lite to the progressively reliable protocol. Corruption will be confined to the UDP payload. Application-level Leaky ARQ will need to implement its own error detection on the Leaky ARQ protocol headers embedded within the UDP-lite payload. In order to improve the chances of header-valid reception, FEC should be applied on headers throughout the protocol stack. Leaky ARQ can apply FEC on its own protocol headers, but will be unable to protect lower layer UDP-lite and IP headers. Therefore, UDP-lite can improve the chances of header-valid transport over wireless links by implementing end-to-end FEC on the combination of UDP header and IP pseudo-header. Linear block codes are ideally suited for FEC on fixed-length packet headers.

Implementing an end-to-end protocol as an application-level library of functions on top of UDP has been discussed in the literature [88]. Leaky ARQ built on top of UDP-lite would require a similar set of design trade-offs. Of course, the socket interface between Leaky ARQ and UDP-lite will introduce an additional level of indirection which will reduce the efficiency of Leaky ARQ.

1. From conversations with Steve Pink, currently with the Swedish Institute SICS, on the end-to-end mailing list in late 1996.

6.3.3 SACK-TCP

Current implementations of TCP employ cumulative acknowledgments. Recently, a new proposal to improve the performance of TCP has been made which replaces cumulative acknowledgments with selective acknowledgments, called SACK-TCP [110]. A selective-repeat (SRP) TCP sender combined with selective acknowledgments provides the fundamental structure to implement a progressively reliable acknowledged datagram protocol required by Leaky ARQ (observation by K.K. Ramakrishnan of AT&T Labs). The ability to selectively identify specific fragments of a data stream is essential in order to practice successive refinement or packet combining on these packet fragments, and to support forwarding of multiple noisy versions of an ADU to the application.

6.4 Implementing transport services within Leaky ARQ

In this section, we survey several other issues related to Leaky ARQ implementation, including fragmentation of ADU's, acknowledgments, and non-sliding windows. Other important design issues such as a complete time-out strategy, NAKs vs. ACKs, adaptive resizing of congestion and flow control windows, and state machine design for both the transmitter and receiver are not addressed in this dissertation.

6.4.1 Fragmentation of ADU's

Application-level framing constrains leaky ARQ to operate as an acknowledged datagram protocol. Since Chapter 2 argued that fragmentation of large ADU's into smaller packets for transmission and reassembly can dramatically reduce transport latency, we would like to fragment an ADU into smaller datagrams. In many respects, ADU fragmentation is similar to the IP datagram fragmentation problem [25].

Internal to the protocol we define an *ADU fragment*. Figure 6.2 shows that each fragment's header will contain at least ADU-related information (flow header, correlation

ADU sequence #	ADU flow header	ADU correla-tion label(s)	ADU length	fragment offset pointer	fragment header length	fragment total length	fragment header CRC	fragment payload CRC

Figure 6.2 A subset of the header fields defined for each fragment. The fragmented ADU payload is concatenated to the end of the fragment header.

labels, ADU length, ADU sequence number), fragment-related fields (fragment offset, fragment total length, fragment header length, fragment payload CRC), and a fragment header CRC that is calculated over all the non-CRC header fields. Recall from Figure 5.4 that the ADU header contains both application-dependent error-sensitive data, like (x,y) location coordinates, in addition to control information like the ADU's length field. The protocol will extract only the control fields from the ADU header that are needed in the fragment header to reassemble the fragmented ADU at the receiver. These fields are the first four starting from the left shown in Figure 6.2, and do not include application-specific data such as location information. The rest of the fields are fragment-specific. FEC is applied on the header at a depth which is dependent upon the prenegotiated depth associated with that application's flow. FEC may also be applied on the fragment payload depending on how successive refinement is implemented and the prenegotiated depth associated with that application flow.

Retransmissions, acknowledgments, and code combining can all be performed on fragments rather than ADU's. The various retransmission options are shown in Figure 6.3, where an additional distinction is made between retransmission with and without incremental redundancy. The initial transmission contains sufficient information to reconstruct the payload, since the initial payload may be clean of errors and not require retransmission at all. Subsequent retransmissions may only contain incremental parity check bits.

As the receiver tries to reassemble an ADU from its fragments, the receiver first error corrects the fragment header and then compares a computed fragment header CRC to the received fragment header CRC (computed only on all non-CRC header fields). If the fragment header is error-free, then the ADU sequence number in conjunction with the fragment offset pointer are used to identify the individual fragment. The fragment's payload undergoes packet combining and the successively refined fragment is cached with the other fragments of the same ADU, some of which may be corrupt and others clean. As each fragment is successively refined, the reconstructed fragment is compared to the fragment payload CRC (computed on the original fragment payload, not the current fragment payload, which may only contain incremental redundancy used for payload combining). If the fragment payload is error-free, or if the refined fragment payload is sufficiently reliable, then the fragment is acknowledged as having been correctly received. Since each fragment payload converges toward a more correct representation due to packet combining, then the ADU payload will also converge towards an error-free representation, fulfilling our promise of successive refinement of ADU's.

The ADU length field determines ADU framing boundaries. This helps the receiver identify when all the fragments for an ADU have been received. The fragment total length

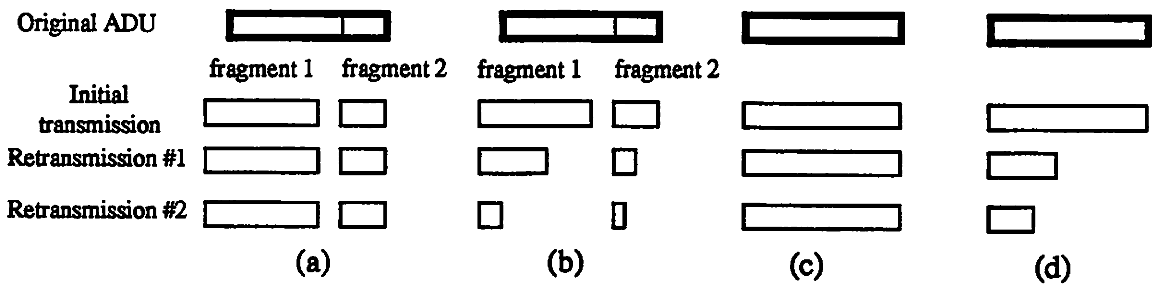


Figure 6.3 Retransmission granularity. (a) Fragment retransmission without incremental redundancy (IR) (b) Fragment retransmission with IR (c) ADU retransmission without IR (d) ADU retransmission with IR.

field only describes the size of the fragment, not the size of the ADU. Without further information, the receiver would not know at what point all data for an ADU had been received. IP solves this problem by using a "more-fragments" bit in conjunction with the fragment offset. However, the IP approach can create a complicated reconstruction problem if one or more segments are lost. The receiver will be unable to determine the size of the ADU for variably-sized segments, or for fixed-size segments when the final (variably-sized) segment is lost. This unnecessarily delays delivery of the initial corrupt version of the ADU. Hence, we explicitly specify the size of the ADU.

Variable-sized headers are supported by the fragment header length field. Observe that once any of the fragments of an ADU have been received with valid headers, then much of the information contained in the fragment header is redundant. For example, per-ADU header fields like the ADU header, ADU length, flow header and correlation labels need only be correctly transmitted once. Provided that a valid fragment header has already been received, then only the ADU sequence number plus the per-fragment header fields are needed to uniquely identify the fragment (along with port numbers, host addresses, etc.). This minimal information is used to define a minimal fragment header with a fixed well-known size. When the sender sets the fragment header length field to this minimal size, then it is signalling that a compact header is being sent, instead of a full-fledged header. In this way, retransmissions need only contain compact headers. This mechanism can be further extended to transmit the ADU header only in the first fragment, instead of each fragment.

6.4.2 Non-sliding windows

The sliding windows characteristic of most ARQ protocols can result in decreased performance for bursty visual multimedia. In TCP, the sender is restricted not just by the window size W of outstanding unacknowledged bytes, but also by the sliding property of the sender's window. The sender cannot transmit beyond W bytes past the lowest unacknowledged byte. If W-1 of the previous words have been correctly received, but the oldest sequence number has not been acknowledged, then the window stays fixed on this sequence number until it has been acknowledged. Hence, available bandwidth is underutilized even as additional data may be queued waiting for transmission. The analogy holds for ADU's as well. The Leaky ARQ window merely specifies the number of bytes outstanding at any given time, and not any additional ordering relationships among ADU's.

6.4.3 Acknowledgments

Since we're supporting fragmentation of ADU's, acknowledgments must be able to identify individual fragments. In addition, acknowledgments should distinguish between reception of a valid header with a dirty payload and a valid header with a sufficiently reliable payload. Such a distinction would enable abbreviated retransmission headers and incremental redundancy. Once the transmitter has been informed that the first header-valid payload has been reconstructed with errors, then it can begin sending incrementally coded redundancy and/or shortened retransmission headers. Without this additional precision, the sender would not know when to send the smaller-sized packets; retransmissions would be limited to segment-sized granularity and retransmission headers would repeatedly carry possibly unnecessary fields.

6.5 Summary and conclusions

In this chapter, we have presented Leaky ARQ, a progressively reliable end-to-end protocol, designed for conveying delay-sensitive image data quickly across noisy wireless links, yet with eventual reliability. We have illustrated what modifications need to be made to conventional ARQ in order to support Leaky ARQ. We have shown that Leaky ARQ can be implemented by modifying Type-II Hybrid ARQ protocols (also called memory ARQ and/or packet combining) to forward corrupt intermediate renditions of a packet. We have described briefly the drawbacks of implementing Leaky ARQ on top of UDP, possibly in combination with TCP. Finally, we have discussed other implementation issues such as fragmentation of ADU's, acknowledgments, and non-sliding windows.

7

Future Work

In this dissertation, we have proposed a progressively reliable end-to-end network protocol, which we call Leaky ARQ, for delay-sensitive high-bandwidth visual multimedia delivered over a noisy and limited bandwidth wireless channel with time-varying bursts of errors. Leaky ARQ forwards corrupt information to the application and eventually delivers a sufficiently reliable final version of all forwarded data. We have justified Leaky ARQ's forwarding of corrupt information from a theoretical joint source/channel coding point of view. We have also discussed practical implementation issues regarding Leaky ARQ. In this chapter, we describe some of the more interesting theoretical and practical challenges uncovered by our research into the concept of progressively reliable packet delivery.

7.1 Theoretical "Holy Grail" challenges

Information theory only addresses the issues of rate and reliability/distortion and ignores the dimension of delay and subjective quality. The channel coding theorem assumes that arbitrarily close to zero distortion delivery is possible provided the entropy rate of the source lies below the channel capacity, and assumes that source and channel encoders and decoders are unconstrained in delay, i.e. they can take arbitrarily long to per-

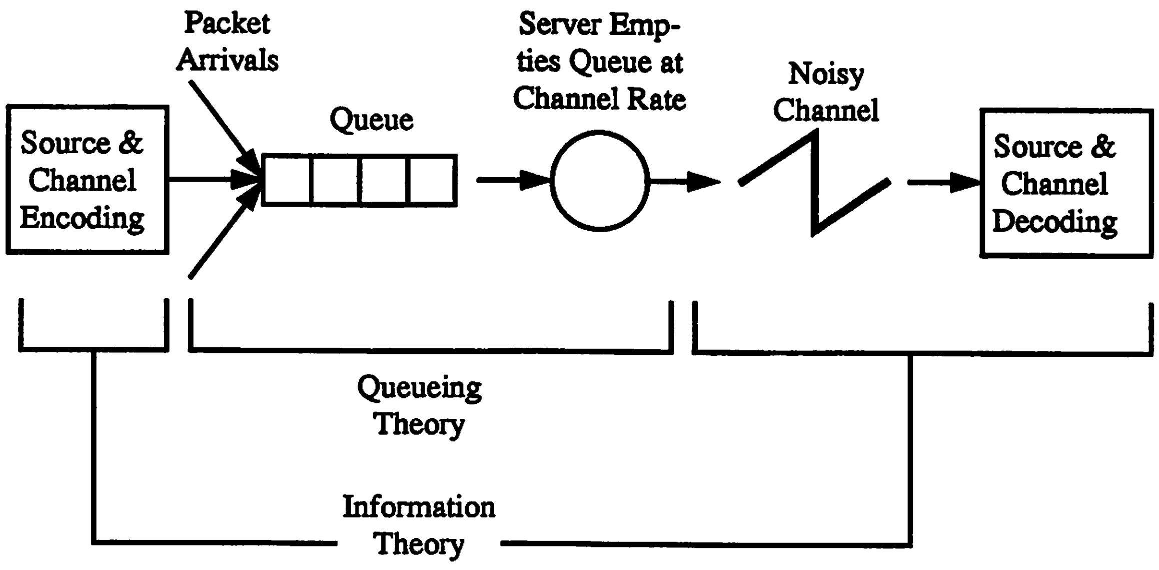

Packet Arrivals
Server Empties Queue at Channel Rate
Noisy Channel
Queue
Source & Channel Encoding
Source & Channel Decoding
Queueing Theory
Information Theory

Figure 7.1 Information theory and queueing theory address different issues. Information theory addresses the two elements of distortion/reliability and information rate, but ignores the third element of delay. Queueing theory addresses the issues of delay and information rate, but ignores the third dimension of unreliable noisy channels. The next great "Holy Grail" for information and communication theorists is to develop a single theory which encompasses delay, reliability, and rate. Such a theoretical framework should be able to characterize progressive source and channel coding, which combine each of these three dimensions.

form their tasks. Thus, information theory essentially ignores the critical role of delay in the performance of the communication system.

On the other hand, queueing theory only concerns itself with delay/waiting time and the rates of arrival (source) and servicing (channel), but ignores the unreliability introduced by the downstream channel. In Figure 7.1, we illustrate how queueing theory and information theory address different issues. No comprehensive theory exists which encompasses simultaneously all three elements of rate, reliability/distortion, and delay. Recently, an initial attempt has been made to combine the two theories [130].

Given the theme of progressivity in this dissertation, we observe that progressive source coding and progressive channel coding represent techniques which combine all three elements of delay, distortion, and information rate into a single practical paradigm. Any all-encompassing theory for delay, reliability, and information rate should be able to characterize the theme of progressivity within its framework. We believe that a single coherent theory which combines all three dimensions is the next "Holy Grail" for information and communication network theorists.

Progressive source coding, also called successive refinement of source data, has been characterized in information-theoretic terms in the literature [37]. This work establishes conditions under which successive source refinement is theoretically possible, i.e. that the rate-distortion limit is attained at each refinement step. Conceptually, over time, the user is moving up the rate-distortion curve. However, this work does not fundamentally integrate the notion of progressivity over *time*, and only considers successive refinement in the distortion domain. For example, we would like to know: can a given information rate suffering a given level of source coding distortion can be delivered within a given delay bound? Moreover, for progressive coding, we would like to know if a *sequence* of three-dimensionally constrained questions is achievable. Currently, even a single three-dimensionally constrained question is only answerable by means of heuristic guesswork.

Similarly, progressively reliable packet delivery is a form of successive refinement spread over time, though the distortion is introduced in a more uncontrolled fashion by channel noise. Given a metric for measuring the impact of channel distortion, we would again like to know if a given information rate suffering a given level of channel distortion can be delivered within a given latency bound.

In fact, since we have demonstrated via our X server experimentation that progressive image transmission can be combined with progressively reliable delivery, then our final question is: can a progressive sequence of (information rate, source+channel distortion, delay) points be achieved in (rate, reliability, delay) three-dimensional space? These are all questions for theorists to answer.

7.2 Lower-level infrastructure for Leaky ARQ

In the process of developing Leaky ARQ, we chose to analyze only a few design issues, and deferred many design decisions until the future. In this section, we conjecture about future research in the area of communicating link-level wireless state information up the protocol stack to the endpoints of a Leaky ARQ connection, and conversely communicating application-level source information down the protocol stack to the data-link level for selective UEP and scheduling of multi-flow multi-user application data over the wireless link.

7.2.1 Communicating QOS parameters down to IP and data-link layers

By emphasizing the virtues of joint source/channel coding (JSCC) in a constrained-complexity constrained-delay non-stationary wireless environment, we have been contending that equal error protection (EEP) is an inherently inefficient worst-case technique that is poorly matched to the applications' wide range of error and delay sensitivities. Consequently, an important component of the JSCC architecture is that QOS information is communicated up and down the protocol stack, so that UEP can be employed at the data-link layer for the wireless channel.

Both IPV6 flow headers as well as the Protocol ID bit field in IP can furnish the data-link protocol with some information about the error protection requirements of the application data encapsulated within an IP datagram. These two pieces of information provide part of the structure for communicating information from the transport layer down to the data-link layer. Ideally, we would like to be able to communication enough information down to the data-link layer of the wireless link so that fine-grained UEP via power control [156] and scheduling could be practiced over the wireless link. Open issues include determining what else needs to be in place to communicate application-level QOS parameters through transport, network layers down to the data-link, and determining what parameters will be most useful to the application at the connection endpoints.

7.2.2 Communicating channel state up to end-to-end protocols and application endpoints

Finally, as we noted in Chapter 5, Leaky ARQ can not only delay retransmissions due to the end user's subjective tolerances, but our progressively reliable protocol can also delay retransmissions in response to channel conditions. Moreover, feedback of the channel state to Leaky ARQ can also help the protocol distinguish between corruption-induced loss and congestion-induced loss. This is one of the key practical problems which limits TCP over wireless links, due to TCP's implicit assumption that all packet losses are due to congestion. We would like to avoid repeating the same mistake for Leaky ARQ, i.e. we wish to avoid building in the assumption that all losses are due to bit corruption. Feeding back channel statistics to the channel endpoints can help Leaky ARQ distinguish between congestion and corruption.

In addition, it may also be helpful to feedback channel conditions all the way up to the application. A wireless-aware multimedia application may be able to adapt its image/speech/video coding in response to varying channel conditions. Or a wireless-aware multimedia application may be able to change the parameterization of Leaky ARQ in response to a prolonged wireless fade.

Open research questions include: How should the channel state be passed back? What parameters are useful? For example, a running window average of the BER could be maintained by the data-link layer, or a windowed packet loss count. How often should channel updates be fed back? For example, channel state could be communicated only during connection setup, i.e. long-term channel statistics, or channel state could be updated frequently during the connection. We leave these questions as open research issues.

Bibliography

[1] T. Agui, M. Shimizu, M. Nakajima, "Speed-up Algorithm of Color Image Quantization," *Systems and Computers in Japan*, vol. 19, no. 3, pp. 73-78, 1988.

[2] A. Albanese, J. Blomer, J. Edmonds, M. Luby, "Priority Encoding Transmission," *Technical Report TR-94-039*, International Computer Science Institute, Berkeley, August 1994. See also http://www.icsi.berkeley.edu/PET/icsi-pet.html.

[3] C. M. Aras, J. F. Kurose, D. S. Reeves and H. Schulzrinne, "Real-time Communication in Packet-Switched Networks," *Proc. of the IEEE*, Vol. 82, No. 1, January 1994, pp. 122-138.

[4] E. Ayanoglu, S. Paul, T. LaPorta, K. Sabnani, R. Gitlin, "AIRMAIL: A link-layer protocol for wireless networks," *ACM Wireless Networks*, vol. 1, no. 1, pp. 47-59, February 1995.

[5] H. Balakrishnan, S. Seshan, R. Katz, "Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks," *ACM Wireless Networks*, vol. 1, no. 4, pp. 469-81, 1995.

[6] R. Balasubramanian, C. Bouman, J. Allebach, "New Results In Color Image Quantization," *SPIE Image Processing Algorithms and Techniques III*, vol. 1657, pp. 289-303, 1992.

[7] J. Bartlett, "Experience with a Wireless World Wide Web Client," *COMPCON 95*, pp. 154-7, March 1995.

[8] E. Berlekamp, R. Peile, S. Pope, "The Application of Error Control to Communications," *IEEE Communications Magazine*, vol. 25, no. 4, pp. 44-57, April 1987.

[9] E. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, 1968.

[10] P. Bhagwat, P. Bhattacharya, A. Krishna, S. Tripathi, "Enhancing Throughput Over Wireless LAN's Using Channel State Dependent Packet Scheduling," *INFOCOM 96*, vol. 3, pp. 1133-1140, 1996.

[11] M. Bickford, "WINSOCK 2: The New Face of Networked Applications," *Data Communications*, pp. 75-79, March 21, 1996.

[12] R. Cam, C. Leung, C. Lam, "A Performance Comparison of Some Combining Schemes for Finite-Buffer ARQ Systems in a Rayleigh-Fading Channel," *IEEE International Conference on Selected Topics in Wireless Communications*, pp. 88-92, June 1992.

[13] Y. Chang, C. Leung, "On Weldon's ARQ Strategy," *IEEE Transactions on Communications*, vol. 32, no. 3, pp. 297-300, March 1984.

[14] D. Chase, "Code Combining - A Maximum-Likelihood Decoding Approach for Combining an Arbitrary Number of Noisy Packets," *IEEE Transactions on Communications*, vol. 33, no. 5, pp. 385-393, May 1985.

[15] D. Chase, P. Muellers, J. Wolf, "Application of Code Combining to a Selective-Repeat ARQ Link," *MILCOM 1985*, vol. 1, pp. 247-252, October 1985.

[16] W. Chau , S. Wong, X. Yang, S. Wan, "On the Selection of Small Color Palette For Color Image Quantization," *SPIE Image Processing Algorithms and Techniques III*, vol. 1657, pp. 326-333, 1992.

[17] C. Chen, "Computer Results on the Minimum Distance of Some Binary Cyclic Codes," *IEEE Transactions on Information Theory*, Vol. IT-16, No. 3, pp. 359-60, May 1

[18] N. Cheng, N. Kingsbury, "The ERPC: An Efficient Error-Resilient Technique for Encoding Positional Information or Sparse Data," *IEEE Transactions on Communications*, vol. 40, no. 1, pp. 140-148, January 1992.

[19] Y. Cho, C. Un, "Performance Analysis of ARQ Error Controls Under Markovian Block Error Pattern," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 2051-2061, February/March/April 1994.

[20] C. Chou, C. Chen, "A Perceptually Optimized 3-D Subband Codec for Video Communication Over Wireless Channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 2, pp. 143-156, April 1996.

[21] J. Chouinard, M. Lecours, G. Delisle, "Simulation of Error Sequences in a Mobile Communications Channel with Fritchman's Error Generation Model," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 134-137, June 1989.

[22] D. Clark, V. Jacobson, J. Romkey, H. Salwen, "An Analysis of TCP Processing Overhead", *IEEE Communications Magazine*, pp. 23-29, June 1989.

[23] D. Clark, D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *ACM SIGCOMM 1990*, pp. 200-208.

[24] G. Clark Jr., J. Cain, *Error-Correction Coding for Digital Communications*, Plenum Press, 1981.

[25] D. Comer, *Internetworking with TCP/IP, Volume I, 2nd edition*, Prentice-Hall, 1991.

[26] P. Conrad, E. Golden, P. Amer, R. Marasli, "A Multimedia Cocument Retrieval System Using Partially-Ordered/Partially-Reliable Transport Service," *SPIE Multimedia Computing and Networking*,Proc. SPIE, Vol. 2667, pp. 136-147, January 1996.

[27] T. Cover, J. Thomas, *Elements of Information Theory*, John Wiley & Sons, 1991.

[28] D. Cox, "Wireless Network Access for Personal Communications," *IEEE Communications Magazine*, pp. 96-115, December 1992.

[29] D. Cox, "Wireless Personal Communications: What Is It?", *IEEE Personal Communications*, vol. 2, no. 2, pp. 20-35, April 1995.

[30] A. DeSimone, M. Chuah, O. Yue, "Throughput Performance of Transport-Layer Protocols over Wireless LANs," *GLOBECOM 1993*, vol. 1, pp. 542-549, November 1993.

[31] R. Dixon, *Spread Spectrum Systems With Commercial Applications, 3rd edition*, John Wiley & Sons, 1994.

[32] W. Doeringer, D. Dykeman, M. Kaiserswerth, B.W. Meister, H. Rudin, R. Williamson, "A Survey of Light-Weight Transport Protocols for High-Speed Networks," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 2025-2039, November 1990.

[33] B. Doshi, P. Johri, A. Netravali, K. Sabnani, "Error and Flow Control Performance of a High Speed Protocol," *IEEE Transactions on Communications*, vol. 41, no. 5, pp. 707-720, May 1993.

[34] A. Drukarev, K. Yiu, "Performance of Error-Correcting Codes on Channels with Memory," *IEEE Transactions on Communications*, vol. 34, no. 6, pp. 513-521, June 1986.

[35] E. Dubois, J. Moncet, "Transform Coding For Progressive Transmission of Still Pictures," *GLOBECOM 85*, volume 1, pp. 348-352, December 1985.

[36] EIA/TIA Interim Standard, Cellular System Dual-Mode Mobile Station-Base Station Compatibility Standard, IS-54-B, April 1992, Telecommunications Industry Association.

[37] W. Equitz, T. Cover, "Successive Refinement of Information," *IEEE Transactions on Information Theory*, vol. 37, no. 2, pp. 269-275, March 1991.

[38] K. Fazel, J. Lhuillier, "Application of Unequal Error Protection Codes on Combined Source-Channel Coding of Images," *IEEE International Conference on Communications (ICC) 90*, pp. 898-903, 1990.

[39] J. Feng, K. Lo, H. Mehrpour, A. Karbowiak, "Cell Loss Concealment Method for MPEG Video in ATM Networks," *GLOBECOM 95*, pp. 1925-1929, 1995.

[40] P. Feldman, V. Li, "An Adaptive Hybrid ARQ Protocol For Continuous-Time Markovian Channel," *MILCOM 87*, vol. 2, pp. 502-8, 1987.

[41] D. Ferrari, "Client Requirements for Real-Time Communication Services," *IEEE Communications Magazine*, pp. 65-72, November 1990.

[42] B. Fritchman, "A Binary Channel Characterization Using Partitioned Markov Chains," *IEEE Transactions on Information Theory*, vol. IT-13, no. 2, pp. 221-227, April 1967.

[43] J. Gauvreau, C. Despins, "Optimal Coding Rate of Punctured Convolutional Codes in Indoor Wireless TDMA Cellular Systems," *Third Annual International Conference on Universal Personal Communications (ICUPC)*, pp. 94-98, 1994.

[44] M. Goldberg, L. Wang, "Comparative Performance of Pyramid Data Structures for Progressive Image Transmission," *IEEE Transactions on Communications*, vol. 39, no. 4, pp. 540-548, April 1991.

[45] N. Goldberg, "Colour Image Quantization For High Resolution Graphics Display," *Image and Vision Computing*, vol. 9, no.5, pp. 303-312, October 1991.

[46] S. Golomb, R. Peile, R. Scholtz, *Basic Concepts in Information Theory and Coding*, Plenum Press, 1994.

[47] R. Gray, *Source Coding Theory*, Kluwer Academic Publishers, 1990.

[48] N. Guo, S. Morgera, "Frequency-Hopped ARQ for Wireless Network Data Services," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 8, pp. 1324-1337, October 1994.

[49] J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 389-400, April 1988.

[50] J. Hagenauer, "Source-Controlled Channel Decoding," *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2449-2457, September 1995.

[51] R. Han, D. Messerschmitt, "Asymptotically Reliable Transport of Multimedia/Graphics Over Wireless Channels," *SPIE Multimedia Computing and Networking*, Proc. SPIE, Vol. 2667, pp. 99-110, January 1996.

[52] J. Hanratty, G. Stuber, "Performance Analysis of Hybrid ARQ Protocols in a Slotted Direct-Sequence Code Division Multiple-Access Network: Jamming Analysis," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 4, pp. 562-579, May 1990.

[53] B. Harvey, S. Wicker, "Packet Combining Systems Based on the Viterbi Decoder," *MILCOM 1992*, vol. 2, pp. 757-762.

[54] P. Haskell, D. Messerschmitt, "Resynchronization of Motion Compensated Video Affected by ATM Cell Loss," *ICASSP 92*, vol. 3, pp. 545-548, 1992.

[55] P. Heckbert, "Color Image Quantization For Frame Buffer Display", *Computer Graphics* (*SIGGRAPH*), vol. 16, no.3, pp. 297-307, 1982.

[56] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html

[57] C. Huitema, *IPv6: the New Internet Protocol*, Prentice-Hall, 1996.

[58] A. Hung, T. Meng, "Error Resilient Pyramid Vector Quantization for Image Compression," *ICIP 94*, vol. 1, pp. 583-587, 1994.

[59] A. Jalali, G. Mony, L. Strawczynski, "Performance of Data Protocols for In-Building Wireless Systems," *First International Conference on Universal Personal Communications*, pp. 407-411, 1992.

[60] N. Jayant, P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, 1984.

[61] S. Jiang, "A Novel Error Control Scheme for CDMA Wireless LANs," *1994 IEEE Region 10's Ninth Annual International Conference, Theme: Frontiers of Computer Technology*, vol. 2, pp. 581-586, 1994.

[62] M. A. Jolfaei, "Stutter XOR Strategies: A New Class of Multicopy ARQ Strategies," *1994 International Conference on Network Protocols*, pp.56-62, 1994.

[63] S. Kallel, "Analysis of Memory and Incremental Redundancy ARQ Schemes Over a Nonstationary Channel," *IEEE Transactions on Communications*, vol. 40, no. 9, pp. 1474-1480, September 1992.

[64] S. Kallel, "Efficient Hybrid ARQ Protocols with Adaptive Forward Error Correction," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 281-289, February/March/April 1994.

[65] S. Kallel, D. Haccoun, "Generalized Type II Hybrid ARQ Scheme Using Punctured Convolutional Coding," *IEEE Transactions on Communications*, vol. 38, no. 11, pp. 1938-1946, November 1990.

[66] S. Kallel, C. Leung, "Efficient ARQ Schemes with Multiple Copy Decoding," *IEEE Transactions on Communications*, vol. 40, no. 3, pp. 642-650, March 1992.

[67] L. Kanal, A. Sastry, "Models for Channels with Memory and Their Applications to Error Control," *Proceedings of the IEEE*, vol. 66, no. 7, pp. 724-743, July 1978.

[68] C. Kantarjiev, A. Demers, R. Frederick, R. Krivacic, M. Weiser, "Experiences with X in a Wireless Environment," *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium*, pp. 117-128, 1993.

[69] P. Karn, "The Qualcomm CDMA Digital Cellular System," *Proceedings of the USENIX Mobile and Location-Independent Computing Symposium*, pp. 35-39, 1993.

[70] W. Keck, "A Method for Robust Decoding of Erroneous MPEG-2 Video Bit-streams," *IEEE Transactions on Consumer Electronics*, vol. 42, no. 3, pp. 411-421, August 1996.

[71] K. Knowlton, "Progressive Transmission of Grey-Scale and Binary Pictures by Simple, Efficient, and Lossless Encoding Schemes," *Proceedings of the IEEE*, vol. 68, no. 7, pp. 885-896, July 1980.

[72] A. Konheim, "A Queueing Analysis of Two ARQ Protocols," *IEEE Transactions on Communications*, vol. 28, no. 7, pp. 1004-1014, July 1980.

[73] B. Kurz, "Optimal Color Quantization for Color Displays," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 217-224, June 1983.

[74] W. Lam, A. Reibman, "An Error Concealment Algorithm for Images Subject to Channel Errors," *IEEE Transactions on Image Processing*, vol. 4, no. 5, pp. 533-542, May 1995.

[75] A. Lao, A, J. Reason and D.G. Messerschmitt, "Asynchronous Video Coding For Wireless Transport," *IEEE Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA., December 1994.

[76] H. Lee, J. Liu, A. Chan, C. Chui, "An Error Concealment Algorithm for Wavelet-Coded Images Over Packet-Switched Networks," *SPIE Multimedia Computing and Networking 1996*, vol. 2667, pp. 222-233, 1996.

[77] M. Liljeberg, T. Alanko, M. Kojo, H. Laamanen, K. Raatikainen, "Optimizing World-Wide Web for Weakly Connected Mobile Workstations: An Indirect Approach," *Second International Workshop on Services in Distributed and Networked Environments*, pp. 132-139, 1995.

[78] S. Lin, D. Costello, M. Miller, "Automatic-Repeat-Request Error-Control Schemes," *IEEE Communications Magazine*, vol. 22, no. 12, pp. 5-16, December 1984.

[79] S. Lin, D. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[80] F. Liu, P. Ho, V. Cuperman, "Joint Source and Channel Coding Using a Non-Linear Receiver," *IEEE International Conference on Communications (ICC) 93*, vol. 3, pp. 1502-1507, 1993.

[81] D. Lu, J. Chang, "Performance of ARQ Protocols in Nonindependent Channel Errors," *IEEE Transactions on Communications*, vol. 41, no. 5, pp. 721-730, May 1993.

[82] N. Mansfield, *The Joy of X: An Overview of the X Window System*, Addison-Wesley, 1993.

[83] S. McCanne, V. Jacobson, "VIC: A Flexible Framework for Packet Video," *ACM Multimedia*, pp. 511-522, November 1995.

[84] S. McCanne, M. Vetterli, "Joint Source/Channel Coding For Multicast Packet Video," *International Conference on Image Processing*, vol. 1, pp. 25-28, 1995.

[85] A. Mehrotra, *Cellular Radio: Analog and Digital Systems*, Artech House, Inc., 1994.

[86] J. Metzner, D. Chang, "Efficient Selective Repeat ARQ Strategies for Very Noisy and Fluctuating Channels," *IEEE Transactions on Communications*, vol. 33, no. 5, pp. 409-416, May 1985.

[87] A. Michelson, A. Levesque, *Error-Control Techniques for Digital Communication*, John Wiley & Sons, 1985.

[88] Y. Miyake, T. Kato, K. Suzuki, "Implemention Method of High Speed Protocol as Transport Library," *International Conference on Network Protocols*, pp. 172-179, 1995.

[89] J. Morris, "Optimal Blocklengths for ARQ Error Control Schemes," *IEEE Transactions on Communications*, vol. 27, no. 2, pp. 488-493, February 1979.

[90] S. Narayanaswamy, S. Seshan, et al, "Application and Network Support for InfoPad," *IEEE Personal Communications*, vol. 3, no. 2, pp. 4-17, April 1996.

[91] V. Oduol, S. Morgera, "Performance Evaluation of the Generalized Type-II Hybrid ARQ Scheme with Noisy Feedback on Markov Channels," *IEEE Transactions on Communications*, vol. 41, no. 1, pp. 32-40, January 1993.

[92] K. Olson, P. Enge, "Forward Error Correction for an Atmospheric Noise Channel," *IEEE Transactions on Communications*, vol. 40, no. 5, pp. 863-872, May 1992.

[93] M. Orchard, C. Bouman, "Color Quantization of Images," *IEEE Transactions on Signal Processing*, vol. 39, no. 12, pp. 2677-2690, December 1991.

[94] J. Padgett, C. Gunther, T. Hattori, "Overview of Wireless Personal Communications," *IEEE Communications Magazine*, vol. 33, no. 1, pp. 28-41, January 1995.

[95] W. Pennebaker, J. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.

[96] N. Phamdo, F. Alajaji, S. Chen, T. Fuja, "Source-Dependent Channel Coding of CELP Speech Over Land Mobile Radio Channels," *MILCOM 95*, pp. 1041-1045, 1995.

[97] W. Pratt, *Digital Image Processing*, John Wiley & Sons, 1978.

[98] G. Promhouse, S. Tavares, "The Minimum Distance of All Binary Cyclic Codes of Odd Lengths from 69 to 99," *IEEE Transactions on Information Theory*, Vol. IT-24, No. 4, pp. 438-442, July 1978.

[99] G. Promhouse, S. Tavares, "An Investigation of All Binary Cyclic Codes with Block Lengths Less Than 127," Masters of Science Thesis, Queen's University, Kingston, Ontario, Canada, 1977.

[100] M. Purser, *Introduction to Error-Correcting Codes*, Artech House, 1995.

[101] Proposed EIA/TIA Interim Standard, Wideband Spread Spectrum Digital Cellular System Dual-Mode Mobile Station-Base Station Compatibility Standard, April 21, 1992, Qualcomm.

[102] M. Rabbani, P. Jones, *Digital Image Compression Techniques*, SPIE Optical Engineering Press, 1991.

[103] G. Ramamurthy, D. Raychaudhuri, "Performance of Packet Video with Combined Error Recovery and Concealment," *IEEE INFOCOM 95*, pp. 753-761, 1995.

[104] K. Ramchandran, A. Ortega, K. Uz, M. Vetterli, "Multiresolution Broadcast for Digital HDTV Using Joint Source/Channel Coding," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 1, pp. 6-22, January 1993.

[105] T. Rappaport, *Wireless Communications: Principles & Practice*, Prentice-Hall, 1996.

[106] J.M. Reason, L.C. Yun, A.Y. Lao, D.G. Messerschmitt, "Asynchronous Video: Coordinated Video Coding and Transport for Heterogeneous Networks with Wireless Access," *Mobile Computing*, H. F. Korth and T. Imielinski, Ed., Kluwer Academic Press, Boston, MA., 1995.

[107] D. Redmill, N. Kingsbury, "Still Image Coding for Noisy Channels," *ICIP 94*, vol. 1, pp. 95-99, 1994.

[108] M. Rice, S. Wicker, "Adaptive Error Control For Slowly Varying Channels," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 917-926, February/March/April 1994.

[109] RFC 1323, *TCP Extensions for High Performance*, V. Jacobson, R. Braden, D. Borman, May 1992.

[110] RFC 2018, *TCP Selective Acknowledgment Options*, M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, October 1996.

[111] S. Sandberg, M. Pursley, "Retransmission Schemes for Meteor-Burst Communications," *Ninth Annual International Phoenix Conference on Computers and Communications*, pp. 246-253, 1990.

[112] T. Sato, M. Kawabe, T. Kato, A. Fukasawa, "Throughput Analysis Method for Hybrid ARQ Schemes Over Burst Error Channels," *IEEE Transactions on Vehicular Technology*, vol. 42, no. 1, pp. 110-118, February 1993.

[113] K. Sayood, J. Borkenhagen, "Use of Residual Redundancy in the Design of Joint Source/Channel Coders," *IEEE Transactions on Communications*, vol. 39, no. 6, pp. 838-846, June 1991.

[114] K. Sayood, F. Liu, J. Gibson, "A Constrained Joint Source/Channel Coder Design," *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 9, pp. 1584-1593, December 1994.

[115] H. Schulzrinne, "World Wide Web: Whence, Whither, What Next?," *IEEE Network*, pp. 10-17, March/April 1996.

[116] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison Wesley, 1988.

[117] N. Shacham, "Packet Resequencing Under Reliable Transport Protocols," *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, vol. 3, pp. 716-723, 1989.

[118] N. Shacham, "Adaptive Link Level Protocol for Packet Radio Channels," *IEEE INFOCOM 86*, pp. 626-635, 1986.

[119] A. Silberschatz, J. Peterson, P. Galvin, *Operating System Concepts, Third Edition*, Addison-Wesley, 1991.

[120] P. Sindhu, "Retransmission Error Control with Memory," *IEEE Transactions on Communications*, vol. 25, no. 5, pp. 473-479, May 1977

[121] O. Spaniol, A. Fasbender, S. Hoff, J. Kaltwasser, J. Kassubek, "Impacts of Mobility on Telecommunication and Data Communication Networks", *IEEE Personal Communications*, vol. 2, no.5, pp. 20-33, October 1995.

[122] M. Sreetharan, R. Kumar, *Cellular Digital Packet Data*, Artech House Publishers, 1996.

[123] W. Stevens, *UNIX Network Programming*, Prentice-Hall, 1990.

[124] W. Stevens, *TCP/IP Illustrated, Volume 1: the Protocols*, Addison-Wesley, 1994.

[125] G. Stuber, *Principles of Mobile Communication*, Kluwer Academic Publishers, 1996.

[126] N. Suzuki, T. Sasaki, R. Kohno, H. Imai, "An Error-Controlling Scheme According to the Importance of Individual Segments of Model-Based Coded Facial Images," *IEICE Transactions on Fundamental of Electronics, Communications and Computer Sciences*, vol. E77-A, no. 8, pp. 1289-1297, August 1994.

[127] K. Takahashi, N. Ishii, "Robustness of Data Compression Coding Schemes for Still Pictures Over Noisy Channels," *SUPERCOMM ICC*, vol. 3, pp. 1035-1042, 1990.

[128] H. Tanaka, "A Performance Analysis of Selective-Repeat ARQ with Multicopy Retransmission," *Fourth IEEE International Conference on Universal Personal Communications*, pp. 472-475, 1995.

[129] A. Tanenbaum, *Computer Networks, 2nd edition*, Prentice-Hall, 1989.

[130] I. Telatar, R. Gallager, "Combining Queueing Theory with Information Theory for Multiaccess," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 963-969, August 1995.

[131] A Tom, C. Yeh, L. Shaw, "Packet Video for Cell Loss Protection Using Deinterleaving and Scrambling, " *ICASSP 91*, pp. 2857-2860, 1991.

[132] J. Touch, "Defining High-Speed Protocols: Five Challenges and an Example That Survives the Challenges," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 5, pp. 828-835, June 1995.

[133] D. Towsley, "The Stutter Go Back-N ARQ Protocol," *IEEE Transactions on Communications*, vol. 27, no. 6, pp. 869-875, June 1979.

[134] D. Towsley, J. Wolf, "On the Statistical Analysis of Queue Lengths and Waiting Times for Statistical Multiplexers with ARQ Retransmission Schemes," *IEEE Transactions on Communications*, vol. 27, no. 4, pp. 693-702, April 1979.

[135] F. Tse, W. Cham, "A DC Coefficient Estimation for Image Coding," *Fifth International Conference on Image Processing and Its Applications*, pp. 569-573, 1995.

[136] E. Tsern, A. Hung, T. Meng, "Video Compression for Portable Communication Using Pyramid Vector Quantization of Subband Coefficients," *Proceedings of IEEE Workshop on VLSI Signal Processing*, VLSI Signal Processing VI, pp. 444-452, 1993.

[137] C. Turner, L. Peterson, "Image Transfer: An End-to-End Design," *Computer Communication Review/SIGCOMM 92*, pp. 258-268, August 1992.

[138] K. Tzou, "Progressive Image Transmission: a Review and Comparison of Techniques," *Optical Engineering*, vol. 26, no. 7, pp. 581-589, July 1987.

[139] V. Vaishampayan, N. Farvardin, "Optimal Block Cosine Transform Image Coding for Noisy Channels," *IEEE Transactions on Communications*, vol. 38, no. 3, pp. 327-336, March 1990.

[140] V. Vaishampayan, N. Farvardin, "Joint Design of Block Source Codes and Modulation Signal Sets," *IEEE Transactions on Information Theory*, vol. 38, no. 4, pp. 1230-1248, July 1992.

[141] S. Vanstone, P. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, 1989.

[142] S. Vembu, S. Verdu, Y. Steinberg, "The Source-Channel Separation Theorem Revisited," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 44-54, January 1995.

[143] M. Vetterli, J. Kovacevic, *Wavelets and Subband Coding*, Prentice-Hall, 1995.

[144] B. Vucetic, "An Adaptive Coding Scheme for Time-Varying Channels," *IEEE Transactions on Communications*, vol. 39, no. 5, pp. 653-663, May 1991.

[145] J. Walrand, *Communication Networks: A First Course*, Aksen Associates Incorporated Publishers, 1991.

[146] S. Wan, P. Prusinkiewicz, S. Wong, "Variance-Based Color Image Quantization for Frame Buffer Display," *Color Research and Application*, vol. 15, no. 1, pp. 52-58, February 1990.

[147] T. Watanabe, "A Fast Algorithm for Color Image Quantization Using Only 256 Colors," *Systems and Computers in Japan*, vol. 19, no. 3, pp. 64-71, 1988.

[148] V. Weerackody, C. Podilchuk, "Transmission of JPEG Coded Images Over Wireless Channels," *SPIE Wireless Data Transmission*, vol. 2601, pp. 157-168, 1995.

[149] D. Weissman, A. Levesque, R. Dean, "Interoperable Wireless Data," *IEEE Communications Magazine*, vol. 31, no. 2, pp. 68-77, February 1993.

[150] J. Whitehead, R. Krishnamoorthy, "FEC, ARQ, and Throughput of Radio LANs," *1994 IEEE 44th Vehicular Technology Conference*, vol. 3, pp. 1430-1434, 1994.

[151] S. Wicker, "Adaptive Rate Error Control Through the Use of Diversity Combining and Majority-Logic Decoding in a Hybrid-ARQ Protocol," *IEEE Transactions on Communications*, vol. 39, no. 3, pp. 380-385, March 1991.

[152] S. Wicker, *Error Control Systems for Digital Communication and Storage*, Prentice-Hall, 1995.

[153] S. Wilson, *Digital Modulation and Coding*, Prentice-Hall, 1996.

[154] R. Wolff, *Stochastic Modeling and the Theory of Queues*, Prentice-Hall, 1989.

[155] W. Xu, J. Hagenauer, J. Hollmann, "Joint Source-Channel Decoding Using the Residual Redundancy in Compressed Images," *IEEE International Conference on Communications (ICC) 96*, vol. 1, pp. 142-148, 1996.

[156] L. Yun, D. Messerschmitt, "Power Control and Coding For Variable QOS on a CDMA Channel," *IEEE MILCOM*, vol. 1, pp. 178-182, October 1994.

[157] W. Zeng, B. Liu, "Geometric-structure-based Directional Filtering for Error Concealment in Image/Video Transmission," *SPIE Wireless Data Transmission*, vol. 2601, pp. 145-156, 1995.

[158] H. Zhang, S. Keshav, "Comparison of Rate-Based Service Disciplines," *SIGCOMM '91*, vol.21, no.4, pp.113-121, 1991.

[159] L. Zhang, "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks," *Computer Communications Review*, vol. 20, no. 4, pp. 19-29, September 1990.