Appeared in ACM Multimedia Systems Journal, March 1999

A Progressively Reliable Transport Protocol For Interactive Wireless Multimedia

Richard Han
IBM T.J. Watson Research Center
rhan@watson.ibm.com

David Messerschmitt
University of California at Berkeley
messer@eecs.berkeley.edu

Abstract: We propose a *progressively* reliable transport protocol for delivery of delay-sensitive multimedia over Internet connections with wireless access links. The protocol, termed "Leaky" ARQ, initially permits corrupt packets to be leaked to the receiving application and then uses retransmissions to progressively refine the quality of subsequent packet versions. A Web server would employ Leaky ARQ to quickly deliver a possibly corrupt first version of an image over a noisy bandlimited wireless link for immediate display by a Web browser. Later, Leaky ARQ's retransmissions would enable the browser to eventually display a cleaner image. Forwarding and displaying corrupt error-tolerant image data: (1) lowers the perceptual delay compared to fully reliable packet delivery, and (2) can be shown to produce images with lower distortion than aggressively compressed images when the delay budget only permits weak forward error correction. Leaky ARQ supports *delaying* of *re*-transmissions so that initial packet transmissions can be expedited, and *cancelling* of retransmissions associated with "out-of-date" data. Leaky ARQ can be parameterized to partially retransmit audio and video. We propose to implement Leaky ARQ by modifying Type-II Hybrid/"code combining" ARQ.

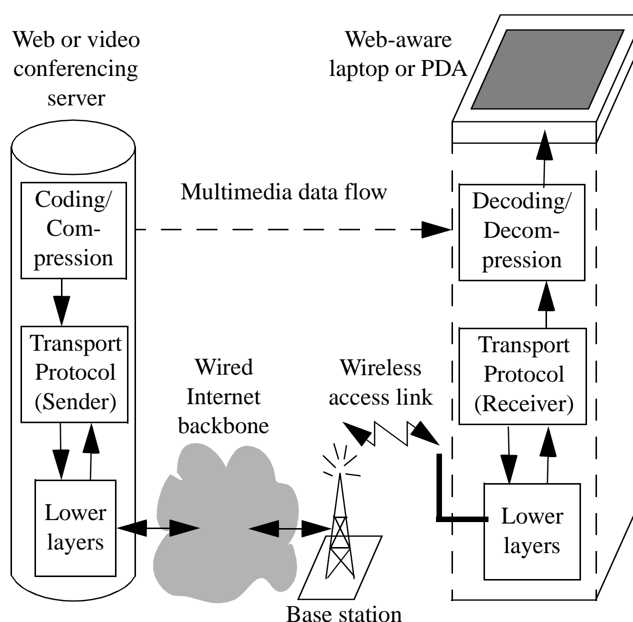

Figure 1. General system architecture supporting wireless access to multimedia on the Internet and Web.

1. Introduction[1]

Portable "network computers" (Berkeley's InfoPad [1] and Xerox PARC's MPad [2]), Web-cognizant PDA's [3], and Web-aware laptops with wireless access to the Internet demonstrate the dramatic convergence of portability, connectivity, and multimedia. Figure 1 illustrates a typical system architecture supporting wireless access to distributed multimedia. Web-based image browsing or interactive audio/ video conferencing generates multimedia data that is encoded by the server and then transported to the receiver by an end-to-end protocol that provides some form of error protection over an Internet connection. The likely scenario of a terminating wireless access link concatenated to a wired Internet backbone is pictured.

Delay-sensitive applications like interactive video conferencing that operate over a wireless access link pose a new challenge to the networking community, namely how to provide sufficiently rapid packetized image delivery to the end user in a manner that is also sufficiently reliable, given the constraints of a noisy wireless bottleneck. Interactive audio/ video conferencing applications typically require delivery of data within about 100-200 ms [4][5]. The desired goal of point-and-click interactivity for Web-based image browsing

1. The material in this paper was presented in part at the SPIE Multimedia Computing and Networking Conference, 1996. This work is based on the first author's Ph.D. thesis. This work was supported by funding from DARPA.

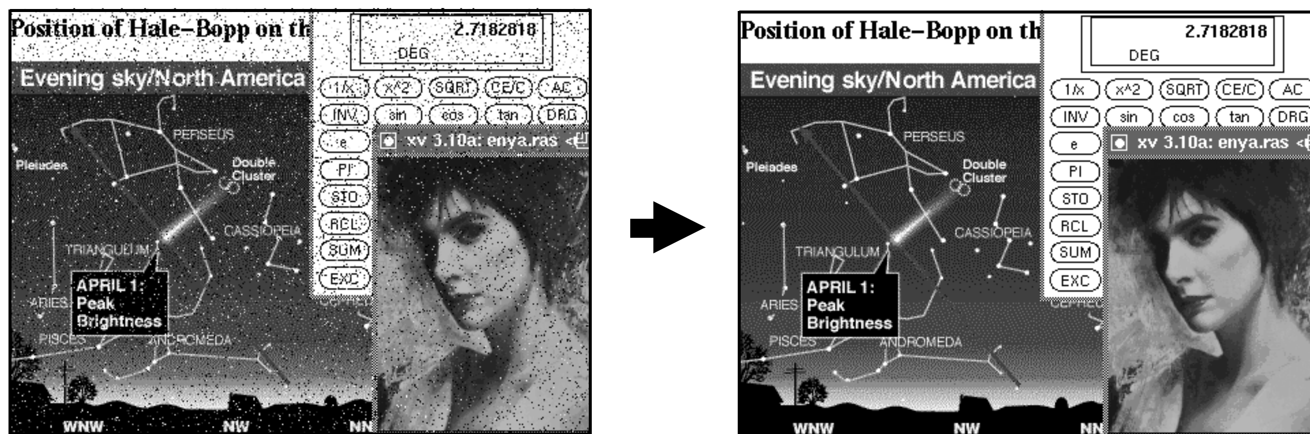

Figure 2. Progressively reliable packet delivery for Web-based image browsing over a wireless access link. An initial version of a packetized image is delivered with corruption to lower the perceived delay. The 8 bits/pixel colormapped image on the left is corrupted at 1% bit error rate (BER). The protocol successively refines each noisy packet, so that the Web browser can eventually present an image with fewer errors. The degree of refinement is parametrizable by the application.

requires similarly rapid packet delivery in order to provide the end user with truly immediate response.

In this paper, we first determine the latency costs of traditional error protection schemes like retransmission-based protocols and forward error correction (FEC) over wired and especially wireless links. In Section 2.1, we quantify the delay due to retransmissions over a wireless access link by deriving a minimum-delay bound. In Section 2.2, we consider the impact of FEC on transmission delay. Given sufficiently tight delay constraints, the combination of error-tolerant compression and delivery of corrupt packets is offered in Section 2.3 as a low-latency solution for interactive applications operating over wireless access links.

In Section 3, we propose end-to-end progressively reliable packet delivery [6], which initially permits delivery of a corrupt packet, and subsequently delivers increasingly reliable versions of that packet. In addition, several features are proposed to enhance the performance of progressively reliable packet delivery. These permit the application to control how long retransmissions are delayed, to cancel retransmissions associated with “out-of-date” data, and to partition its data into multiple flows for packet scheduling by the underlying transport protocol. In Section 4, we consider how to implement the property of successive refinement within a progressively reliable transport protocol. We call such a protocol *Leaky* ARQ since corrupt packets are permitted to be “leaked” to the receiving application.

In Figure 2, we illustrate how a Web-based image browser would make use of progressively reliable packet delivery. The browser would receive corrupt packets containing image data that has been coded by the Web server to be error-tolerant, and would immediately display this visual information, rather than wait for reliable packet delivery. As Leaky ARQ successively refines the quality of subsequent packet versions, the application is able to present versions of the displayed image with successively fewer errors.

2. The case for error-tolerant compression and delivery of corrupt packets

For delay-sensitive applications, the latency introduced by traditional error protection techniques like retransmission-based protocols and FEC becomes intolerable when the interactive delay bound is exceeded. In this section, we analyze the latency costs of both Acknowledgment-Repeat-Request (ARQ) protocols and FEC in wired and wireless environments. We conclude by offering a low-latency solution based on error-tolerant compression and delivery of corrupt packets.

2.1 The latency cost of retransmission-based ARQ protocols

The latency cost of ARQ protocols arises from separate wired and wireless loss phenomena.

2.1.1 Latency over the wired Internet

Images within Web pages are presently transferred over the Internet using the hypertext transfer protocol HTTP [7], which is built on top of TCP [8], the Internet’s reliable transport protocol. Retransmission-based ARQ protocols such as TCP can incur significant delay, due to large roundtrip times and congestion-induced packet loss over the Internet. Roundtrip times on the order of a few hundred milliseconds have been observed over wide-area Internet connections [9]. In the same study, packet loss rates on multi-hop Internet connections were found at times to exceed 10%. Given such

roundtrip delays and packet losses, it is questionable whether reliable delivery over the wired Internet can be depended upon to consistently deliver packets within the interactive latency bound. Consequently, interactive audio/video conferencing applications that operate over the Internet have chosen to employ unreliable packet delivery in order to achieve real-time transport [10][11]. For these delay-sensitive applications, communicating information quickly at the cost of unreliable delivery is subjectively preferable to communicating information reliably at the cost of delivery that is too slow.

2.1.2 Latency over noisy wireless bottlenecks: a minimum-delay bound

Latency introduced by retransmissions becomes an even greater problem over a wireless link, which will likely be the weakest link in the connection both in terms of limited bandwidth and high bit error rate (BER). In this section, our goal is to determine the average amount of time required to reliably transmit a packetized finite-length image over a fixed-BER fixed-bandwidth channel.

Our analysis is simplified if we develop a lower bound on latency over all protocols, so that we don't have to analyze each ARQ protocol individually. Standard ARQ protocols include Go-Back-N (GBN), Selective Repeat (SRP) [12], and TCP[8]. Quantifying the latency incurred by a complex adaptive protocol like TCP can be a difficult task. Fortunately, it has been shown that a form of SRP called ideal SRP bounds the throughput performance of all other repetition-based non-hybrid ARQ schemes like GBN, Stop-and-Wait [13], and TCP (assuming constant retransmission timeouts) [14]. Throughput in this context is a measure of the efficiency of the protocol, i.e. what percentage of the time the protocol spends sending new packets rather than retransmissions. Most ARQ protocols are windowed protocols that allow multiple packets or their retransmissions to be propagating toward the receiver during any single roundtrip time. Many windowed protocols respond to a lost packet by retransmitting other packets besides the lost packet. This can lead to unnecessary retransmissions, thereby lowering the throughput. In contrast, SRP only retransmits the specified lost packet. Therefore, no other single-copy ARQ protocol can improve upon the efficiency of SRP. We do not consider multi-copy strategies [15], and stutter-based enhancements to these protocols [16] [17] in the following analysis, because these techniques represent crude forms of FEC (i.e. repetition coding) for which there are more efficient hybrid FEC/ARQ protocols whose limitations are discussed in Section 2.2.

We begin by quantifying the average number of retransmissions experienced by a *single* packet that is reliably delivered by ideal SRP. Ideal SRP assumes transmitter buffers and receiver buffers have infinite length. Ideal SRP also assumes that the sender's window size is larger than the roundtrip time, so that the transmission pipe can be continuously filled with multiple distinct packets, assuming continuous transmission at the sender.

First, we need to know the average number of retransmissions (including the first transmission) per packet generated by ideal SRP. Let a packet consist of a block of K bits. Assuming independent Bernoulli trials and a fixed BER, the probability P_g of transmitting one K-bit packet through the wireless link without any errors is $P_g = (1-BER)^K$. Assign the random variable N_{packet} = the number of trials until the first good packet is delivered. For ideal SRP, the number of retransmissions for each packet is independent of the number for any other packet. Therefore, 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 N_{packet}, i.e. the average number of trials until the first good packet is received, is given by

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

Our next objective is to determine the latency cost of reliably transferring a complete image that is fragmented into many smaller packets by ideal SRP. Suppose we fragment the original image size I by a factor F, so that each packetized fragment contains ($I/F + H$) bits, where H is the number of header bits per fragment. The expected number of retransmissions for each fragment is closely related to Equation (2-1), and is given by

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

A lower bound estimate for the time it takes to reliably transmit a multi-fragment image is given by the product of the number of fragments F, the average number of transmissions per fragment $E[N_{fragment}]$, and the packet transmission time per fragment $PTT_{fragment}$. In Figure 3, we illustrate how this lower bound estimate is obtained. Our lower bound corresponds to summing all the shaded $PTT_{fragment}$ intervals. Since there are $F*E[N_{fragment}]$ shaded intervals, then the product $F \cdot E[N_{fragment}] \cdot PTT_{fragment}$ gives an estimate of the time needed to reliably transmit a multi-fragment image. Further, this summation of $PTT_{fragment}$ intervals is a lower bound because it does not count the interstitial spaces that appear during retransmission (e.g. after packet fragment F_2 is retransmitted in the figure). These interstitial spaces appear because, for any finite burst of packets corresponding to a fragmented image, the tail end of the packet burst will not have sufficient volume to keep the downstream pipe completely full.

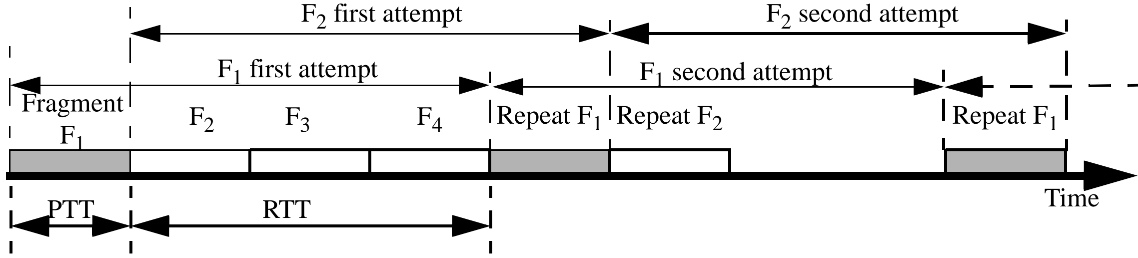

Figure 3. A lower bound estimate of the time it takes to reliably transmit a burst of packet fragments (F_1 through F_4) is obtained by summing the shaded packet transmission times *PTT*. Each time slot is counted at most once in the sum. This is a lower bound because empty slots are not counted. Fixed *PTT*'s and fixed roundtrip times *RTT*'s are pictured. In this example, only packet fragments F_1 and F_2 are retransmitted, while F_3 and F_4 are delivered without loss.

Therefore, if we define T_{image} as the time needed to reliably transmit a packetized image via ideal SRP, then the average overall time for reliable image transmission is given by

$$E[T_{\text{image}}] \geq F \cdot E[N_{fragment}] \cdot PTT_{fragment} \quad (2\text{-}3)$$

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

Equation (2-3) is the key result of this section's analysis. This lower bound on the image transfer delay is found to be a separable function of *BER* and *BW*, so that the sources of latency due to noise and bandwidth can be easily analyzed. This minimum-delay bound increases exponentially as the *BER* increases, and decreases inverse linearly as the wireless bandwidth *BW* increases.

In Figure 4, we construct a log-log plot of Equation (2-3)'s lower bound on $E[T_{image}]$ as a function of the image fragmentation factor *F*. We substitute the following values: a raw *BER* of 10^{-2} for wireless fades [18][19], header size H = 100 bits, wireless BW = 500 kbit/s (in the middle range of wireless transmission rates [20]), and image size I = 20000 bits (about the size of many small compressed images).

The behavior exhibited in Figure 4 is explained as follows. First, partition Equation (2-3) into a "*BER* factor" $(1-BER)^{-\left(\frac{I}{F}+H\right)}$ and a "*BW* factor" $\frac{I+F\cdot H}{BW}$. 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 causing 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 overhead from the header *H* for each ultra-small payload begins to dominate, causing the lower bound on latency to rise.

There is an intermediate packet size or fragmentation factor that produces the minimum delay. In order to find the optimal fragmentation value, we take the derivative of Equation (2-3) with respect to *F*, obtaining a quadratic in *F* which we then set equal to zero, $\frac{\partial}{\partial F}(equation(2-4)) = 0$. Our optimal F_{opt} is given by

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

where we have eliminated the second root. A different approach to deriving the optimal packet length has been described that is based on utilization instead of delay [12]. Substituting our chosen values into Equation (2-3), we obtain an $F_{opt} = 324$, which agrees with our plotted minimum near F=300. Therefore, for our set of assumed parameters, the optimal packet size (payload + header) is about 162 bits and the minimum delay is about 530 ms, or about half a second.

Given a real-time delivery bound of about 200 ms, then we are far from meeting our objective of interactivity for multiple reasons. First, the half second delay is a lower bound due to ideal SRP. All of the transport protocols surveyed in [21], including TCP, practice either some form of GBN, or some non-ideal form of SRP (e.g. finite receiver and transmit buffers), and will therefore incur a higher latency cost than ideal SRP. Second, the half second delay is a lower bound since the interstitial intervals were ignored in the estimate. A tighter minimum-delay bound has been derived that shows that ignoring these interstitial intervals can significantly underestimate the latency when the roundtrip times are large [22]. Third, the half second delay was obtained at the optimal packet length, which according

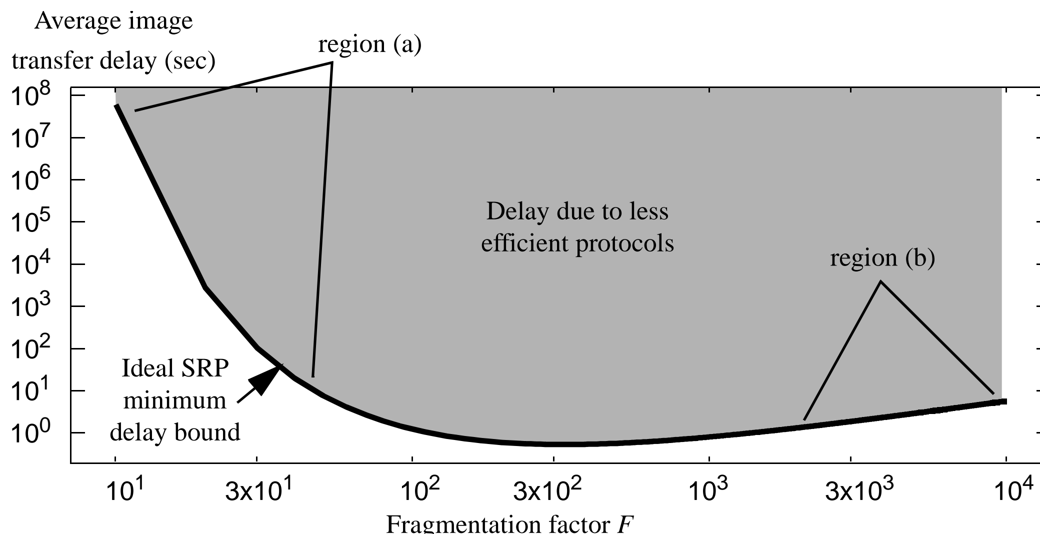

Figure 4. Log-log plot of the lower bound on the average delay experienced by reliable delivery of a packetized image over a bandlimited and noisy channel. $E[T_{image}]$ from Equation (2-3) is plotted as a function of the image fragmentation factor F (image size = 20 kbits, BER = 10^{-2}, BW = 500 kbit/s, and header size H = 100 bits). For very large packets (region (a)), delay is exponentially dominated by bit corruption. For very small packets (region (b)), delay is dominated by header overhead. In between, the optimal packet size causes a minimum delay of about half a second, which exceeds the interactive latency bound. ARQ protocols less efficient than ideal SRP will incur more delay (shaded region).

to Equation (2-4) requires knowledge of the *BER* and *BW* at the sender. In a real-world implementation, knowledge of the wireless bandwidth, and especially of the current *BER* of the wireless link, may not be available, so that chosen packet length will operate at a suboptimal point on the curve that will suffer a much higher penalty in latency.

2.1.3 Improvements to TCP for wireless links

Several proposals to improve TCP's wireless performance have been described [23]. One well-known problem with TCP is that, in a wireless environment, it mistakes packet losses caused by bit corruption for packet losses caused by network congestion, unnecessarily initiating congestion-avoidance mechanisms like window throttling and "slow start". While the various approaches improve the performance of TCP, ultimately their performance is still bounded by the ideal SRP analysis of the previous section. Even though it is possible to lower the delay incurred by TCP using Indirect-TCP and "snoop" TCP, the latency will still not be low enough to achieve the real-time objective of interactivity during wireless fades.

2.2 The latency cost of forward error correction

The latency introduced by open-loop FEC via overhead and interleaving delay can also violate interactive delay bounds. FEC is typically implemented as a block code and/or convolutional code, and is often used in link-layer protocols to improve reliability [19]. Error protection is achieved by adding redundancy to the source's data. For delay-sensitive image data, the delay budget can limit the amount of FEC overhead that can be applied.

Consider the example of Web-based image browsing or interactive video conferencing over wireless links. These applications require roundtrip response times of about 200 ms between user input (e.g. mouse action for Web browsing, or voice input for interactive video conferencing), and visual feedback. Roundtrip times *RTT* over just the wired portion of the Internet were earlier observed to exceed 100 ms for wide-area connections. In this case, the delay budget remaining for wireless image transmission would only be about 100 ms. Digital cellular standards like GSM and IS-95 use convolutional FEC coding and have determined that FEC overhead factors ranging from two to three are needed to adequately protect against cellular fading [24]. Assuming a minimum factor of two is required to adequately protect image data, then the largest amount of image data I_{max} that could be transmitted within the delay bound D is given by $I_{max} = (BW/2) \times (D - RTT)$. Given D=200ms, and RTT=100ms, then I_{max}=0.05*BW.

However, we must also include the interleaving delay introduced by FEC encoding and decoding over a wireless link. Interleaving is a key component of conventional digital cellular standards like IS-95 and GSM, where it is used to spread error bursts to lower the probability that the FEC code is overwhelmed by a concentration of errors. The one-way

delay due to interleaving is at least an extra 20 ms in IS-95 [25] and 37.5 ms in GSM [26]. The total interleaving delay consists of four components: interleaving and deinterleaving delays in the forward direction, and their counterparts in the reverse direction. All four components will contribute to the roundtrip time *RTT*. For example, if we assume 20 ms of interleaving delay in each direction and 2*x* FEC, then I_{max}=0.03**BW*. Even at 1 Mbit/s, only relatively small ~3.8 KByte images could be transmitted within the interactive latency bound. In contrast, if no FEC is applied, then both overhead and interleaving terms can be eliminated, and I_{max}=0.1**BW*. At 1 Mbit/s, Web-based images of size up to ~12 KByte can be communicated interactively.

There is also the potential that end-to-end FEC overhead will be suffered on the wireless link as well. Burst erasure correcting codes have been suggested as an end-to-end solution for mitigating packet losses due to congestion on the wired backbone [19][27][28]. In the event that erasure codes are implemented end-to-end, then erasure-based FEC overhead will be suffered over the wireless link, and consequently must be factored into the overall calculation of the delay budget along with the wireless FEC overhead. Even worse, the error correction provided by erasure codes is on the scale of packets, and is likely to be relatively ineffective against wireless bit errors. Consequently, erasure-based FEC overhead will be suffered over the wireless link without improving error-correcting performance.

Hybrid ARQ protocols that use FEC to lower the number of retransmissions have been shown to achieve a higher throughput than ideal SRP [13]. Consequently, FEC can improve the delay performance of reliable protocols over noisy links [22]. However, hybrid ARQ protocols that wish to adequately protect the first transmission of a packet must still suffer the FEC overhead and interleaving delay calculated above. The ARQ retransmissions will merely add delay beyond what has already been observed for open-loop FEC.

2.3 Low-latency solution: error-tolerant compression and delivery of corrupt packets

Rather than devote the few bits in a tight delay budget to aggressive FEC to protect heavily compressed images, we offer the alternative of error-tolerant image compression as a means to achieve lower distortion image delivery as well as low-latency delivery. We first address the claim of lower distortion, and then address the claim of low-latency delivery.

For wireless links with sufficiently low bit rates, even heavily compressed images may be of high enough volume that it is not possible to apply adequate FEC without violating the interactivity bound. For example, if the wireless bit rate is 1 Mbit/s and the delay budget for image transmission and FEC is 100 ms, then a heavily compressed 10 KByte image will take 80 ms to transmit. Our delay budget would prohibit the application of 2*x* FEC on such an image.

In these circumstances, when there are sufficiently tight constraints on how much FEC can be applied, and when wireless fading is sufficiently severe, then it has been shown quantitatively and qualitatively that error-tolerant compression and decoding of images results in lower end-to-end distortion than aggressive compression [29]. In that study, the overall sum of the source and channel coding rates was constrained to be constant in a very noisy environment, while the proportion of bits devoted to source coding (e.g. image compression) and channel coding (e.g. FEC) was varied. At low SNR's/high BER's, the combination of aggressive compression and aggressive FEC produced images with worse objective and subjective quality than the approach of error-tolerant compression. The intuition is that the constraint on the overall coding rate prohibited the application of sufficiently strong FEC, thereby rendering aggressively compressed data useless during severe fading. The constraint on the overall coding rate effectively acted as an interactive delay constraint and/or complexity limitation on FEC. Therefore, when interactive bounds prevent the application of sufficiently strong FEC over a wireless link, then it is better (i.e. lower distortion) to leave redundancy in the image via error-tolerant compression than to strip out this image redundancy via aggressive compression, only to add back FEC redundancy because aggressive FEC is required.

These observations are part of a larger body of literature known as *joint source/channel coding* (JSCC). This theory advocates error-tolerant compression, unequal error protection (UEP), source-cognizant FEC decoding and application-level error concealment, when complexity and delay constraints are sufficiently tight, and when channels are severely fading and non-stationary (e.g. wireless) [22].

Several authors have established that error-resilient coding of images can achieve compression down to the range of 0.5-1.0 bpp using DCT [30][22], subband [31][32], and VQ [33] source coding techniques, and yet still tolerate 10^{-2} BER despite the lack of any FEC error protection on the coded image bits. Robust compression reduces bandwidth while tolerating errors by employing fixed-length lossy quantization of images or their transforms. Robust compression excludes lossless statistical compression techniques like variable-length Huffman and arithmetic coding that introduce extreme error sensitivity and consequently require powerful FEC.

A protocol that delivers corrupt packets bearing error-tolerant image data gives an interactive multimedia application a reasonable chance of meeting its real-time delivery bound during wireless fades. Only the header of each packet needs to be communicated error-free, not the entire packet payload. In most cases, headers are relatively small compared to payloads. Consequently, the expected number of retransmissions $E[N_{packet}]$ from Equation (2-2) becomes only a function of the header length *H*, and Equation (2-3)

can be rewritten to obtain a new lower bound on the image transfer delay:

$$E[T_{\mathrm{image}}] \geq \frac{1}{(1-BER)^{H}} \cdot \frac{I + F \cdot H}{BW} \qquad (2\text{-}5)$$

Assume the same parameters as Section 2.1: H=100 bits, BER=1%, BW=500 kbit/s, I=20 kbits. The optimal fragmentation value that minimizes latency is F_{opt}=1, i.e. it is faster to send the image as one large packet than as many smaller fragments. This is because more fragments imply more header overhead. Substituting these values in Equation (2-5), we find that the lower bound on delay at F=1 is 110 ms, about five times faster than the half-second latency calculated for reliable ideal SRP in Section 2.1. At F=20 (1000 payload bits/packet), the lower bound on delay is 120 ms. This reduction in the minimum-delay bound to the 100 ms range suggests that header-only reliability could achieve delivery that is reasonably close to the interactive latency bound.

Because the header is a relatively small proportion of the overall packet, then it is possible to apply a high-redundancy FEC code on the header alone without inflicting a large overhead penalty. For example, applying a powerful block code with 4x overhead factor to the header alone will essentially reduce $E[N_{packet}]$ to 1 at 1% BER. Therefore, Equation (2-5) evaluated at F=20 predicts a minimum delay of ~56 ms. Judicious application of high-redundancy FEC on the header only, combined with tolerance of corrupt packet payloads, enables packets to be delivered by the interactive latency bound.

In order to fully realize the benefits of end-to-end error-tolerant compression and decoding, the components of the underlying network will need to forward certain corrupt packets to the destination. Intermediate wired backbone routers and intermediate wireless data-link protocols should not discard delay-sensitive packets with corrupt payloads when those packets are marked as containing error-tolerant data. In the case of single-hop wireless access to the Internet pictured in Figure 1, the data-link protocol should forward corrupt packets to higher layers at the receiver rather than discard all such packets on the mistaken assumption that they cannot be used.

The choice between error-tolerant compression or aggressive compression will depend on several factors that influence the delay budget analysis. These factors include the limit of compression (rate-distortion or subjective) for a specific media, the available wireless bandwidth, the delay-sensitivity of the media, and the severity of channel errors. For example, the delay budget for interactive audio over digital cellular links may in certain cases suggest that aggressive compression should be practiced, since the audio is relatively low-volume even before compression and the wireless bandwidth may be sufficiently high to permit aggressive FEC. The approach taken by IS-95 is to aggressively compress speech and aggressively apply FEC [25]. However, the IS-54 digital cellular TDMA standard takes a different approach, and practices error-tolerant compression of audio combined with UEP, and also permits certain speech bits to be corrupt (“class-two” bits) [34]. For interactive video and Web-based image browsing, this section’s delay budget analysis favors error-tolerant coding and forwarding of corrupt packets for certain wireless links.

3. Defining the application-transport interface for progressively reliable packet delivery

In this section, we identify the basic features of a progressively reliable transport *service*, i.e. the properties that define the socket interface between the application and transport layers, which we call the application-transport interface (ATI). Progressively reliable packet delivery:

- allows corrupt packets to be delivered to the receiving application in increasingly reliable fashion
- allows the application to parameterize how long retransmissions should be delayed, thereby achieving a gain in capacity by traffic smoothing;
- permits the application to cancel “out-of-date” retransmissions
- allows the application to specify multiple flows, that are serviced by the underlying protocol’s packet scheduler according to per-flow delay and loss parameters

3.1 Basic properties

The bursty nature of Web-based image browsing suggests that forwarding of corrupt packets will generate image artifacts that persist indefinitely, or persist at least until some new user action causes the noisy still image to be overwritten. Delivery of multiple versions of a corrupt packet without ensuring improving reliability can also lead to a screen presentation whose quality degrades or fluctuates with channel conditions. Consequently, we require that a progressively reliable transport service ensure that the multiple versions of a corrupt packet delivered to the receiving application improve in reliability. Figure 2 demonstrated the effect of Web-based image browsing using progressively reliable packet delivery.

Forwarding of noisy application data can cause the receiving application to lose synchronization unless the sending application frames its data into application data units (ADU’s) and the underlying transport service preserves ADU message boundaries end-to-end. Consider a system in which the sending application partitions its data into ADU’s, also called application-level framing [35], and embeds a length field within each ADU written to the sender’s socket

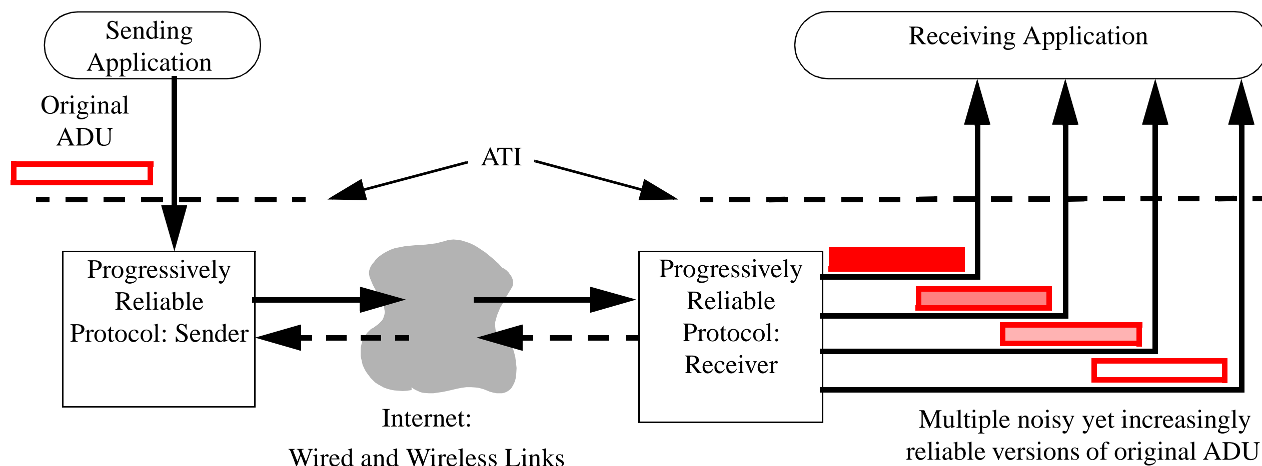

Figure 5. End-to-end progressively reliable packet delivery exhibits four basic properties as seen through the socket interface. The sending application frames its data into application data units (ADU's). At the destination, the receiving application observes that 1) corrupt ADU's are forwarded, 2) multiple noisy versions of each ADU may be forwarded, 3) these multiple versions of the same ADU will improve in reliability over time (statistically fewer errors with each successive version), and 4) different ADU's may arrive out of order (not shown).

buffer. The receiving application extracts ADU boundaries from the receiver's socket buffer by reading the length field in each ADU. However, delivery of noisy ADU's can cause an ADU's length field to be corrupted. In this case, the receiving application will lose track of where an ADU ends and the next ADU begins. Thus, application-level framing alone is insufficient to keep the receiving application synchronized. Additional framing assistance is required from the transport service in the form of read/write messaging primitives. A ReadMessage() primitive that returns data in discrete ADU units means that the receiving application will never lose track of the boundaries of variable-length corrupt ADU's. Also, a WriteMessage() primitive will enable the sending application to specify ADU boundaries to the transport service without having to embed application-level length fields in each ADU.

Given an ADU framework, the four core properties of a progressively reliable transport service are illustrated in Figure 5. For each ADU transferred across the source ATI, multiple noisy versions of each ADU may be delivered across the receiver ATI. In the simplest case, only one error-free ADU is forwarded to the receiving application, due to a lack of bit corruption. In the most general case, multiple retransmissions by the underlying progressively reliable transport protocol may be triggered, leading to multiple noisy ADU versions being delivered to the receiving application.

The multiple retransmissions undertaken by the underlying progressively reliable transport protocol do not constitute a traffic penalty when compared to a reliable protocol. This is because these retransmissions would have to be attempted anyway by a conventional ARQ protocol under the same channel conditions. In fact, progressively reliable packet delivery offers a means to more effectively utilize traffic capacity by allowing retransmissions to be delayed according to the user's subjective tolerances, thereby smoothing the traffic presented to the network as described in the next section.

A multiple-delivery transport service can be seen as a superset of both single-copy unreliable service and single-copy reliable service. For these special cases, the number of deliveries is configured to one, and the quality of that one delivery is set appropriately. We propose that the progressively reliable transport service be parametrizable so that the application endpoints can specify the number of ADU versions to be forwarded.

The requirement that these multiple noisy ADU versions improve in reliability means that each succeeding ADU version should be guaranteed to be delivered with statistically fewer errors than preceding ADU versions. The property of statistically improving reliability reduces, but does not eliminate, the number of occurrences in which ADU version $n+1$ has more errors than the previously forwarded ADU version n. Statistically improving reliability ensures that on average each new ADU version will have fewer errors than its predecessors. Section 4 describes how this property is achieved via successive refinement of noisy packets. Probabilistic guarantees do not preclude the isolated instance in which a newer ADU version may have more errors than a predecessor. We propose that the application endpoints be able to parametrize not only the number of ADU versions, but also the acceptable quality or noise level in each of the forwarded ADU versions, especially of the final version.

The final property defining this transport service is that separate ADU's may arrive "out of order", though the con-

cept of order must be redefined for a multi-copy context. In the absence of noise, ADU's will be reordered by the network. Given a noisy channel, the ADU delivery stream presented to the receiving application will consist of a mixture of multiple noisy versions of different ADU's, so that the traditional definition of "order" for single-copy delivery services must be modified. Suppose we define that order is preserved in a multi-copy sense when the original sequence of ADU's presented to the source ATI matches the order in which *final* ADU versions were delivered across the destination ATI. Given this definition, it is still the case that order will not be preserved, since the final version of ADU *X* may arrive after the final version of ADU *Y*, even though ADU *X* was initially transmitted before ADU *Y*. The application should be built to tolerate such out-of-order ADU delivery.

3.2 Delaying retransmissions

A key method for improving the performance of progressively reliable packet delivery is to allow the application to control when retransmissions are sent, so that traffic can be smoothed and network capacity can be used more effectively. For bursty Web-based image browsing, our subjective experimentation has determined that it is subjectively tolerable to delay sending the retransmission-based redundancy needed to clean up a noisy image, provided that the end user already has available an initially noisy version of that image for immediate interaction. A progressively reliable transport protocol can exploit this subjective phenomenon to quickly transmit delay-sensitive traffic, and to gradually send delay-tolerant retransmissions later whenever the volume of delay-sensitive traffic is sufficiently low.

The conventional assumption of reliable transport protocols is to retransmit data as soon as congestion conditions warrant. However, Figure 6(b) shows that retransmissions for a leading image burst can lead to slower delivery of a trailing image burst. In contrast, we observe that certain applications may prefer to make use of an initially noisy version of an image and consequently can tolerate relaxed delivery of retransmission-based redundancy. Figure 6(c) and (d) show how delaying retransmissions can facilitate rapid initial delivery of possibly noisy image bursts. Retransmissions are translated in time to minimize interference with delay-sensitive transmission of the initial version of each packet.

In order to determine subjectively how long retransmissions could be delayed for Web-based image browsing, we implemented a simple emulation of progressive reliability within a modified X windows server. An initially noisy version of all screen activity was written to the frame buffer and displayed immediately. After a minimum fixed delay, the final error-free version of the screen would be written to the frame buffer for display. This adjustable minimum wait time emulated the effect of delaying retransmissions.

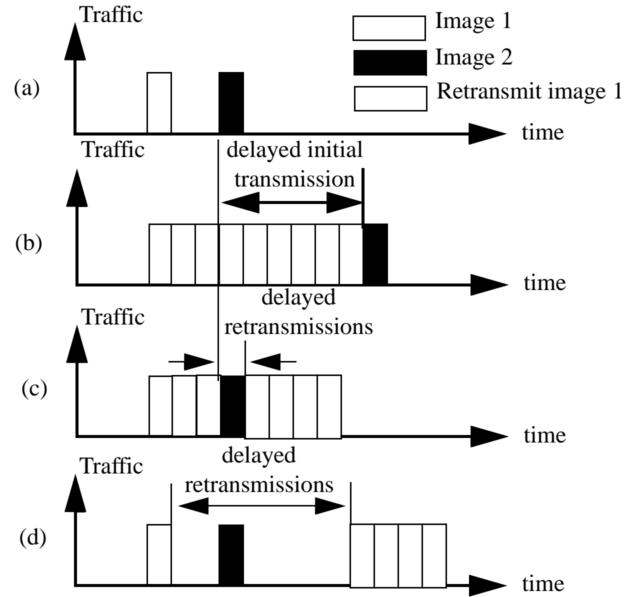

Figure 6. Traffic shaping of retransmissions: (a) original image arrivals (b) retransmissions of image 1 prevent immediate transmission of image 2 (c) delayed retransmissions of image 1 expedite transmission of image 2 (d) retransmissions of image 1 are delayed due to subjective user tolerances.

Our experiments 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 the 100 ms to 1-2 seconds range produced subjectively annoying side-effects.

The first option is to retransmit within 100 ms, but Section 2 indicated the difficulty of fully reliable delivery within such a tight bound.

The second option is to start retransmitting refresh redundancy at the first opportunity possible, whenever there is no delay-sensitive data in the transmit buffer, as shown in Figure 6(c). This opportunity will likely occur within a few seconds after the initial delivery. However, when retransmissions were delayed in the 100 ms to 1-2 seconds range, the act of cleaning up the noisy first version of an image disrupted the user's reading of text. This subjective "flicker" effect forced the reader to stop and then start reading again, thereby losing the reduction in perceived latency offered by fast delivery of a noisy 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 third option, shown in Figure 6(d), best suited the end user's subjective tolerances. We found that delivery latencies on the order of 5 seconds were tolerable provided that the user had an initial version of an image with which to interact. Such delays were found to be short enough to "quickly" clean the screen yet long enough to avoid "snake" and "flicker" effects. We propose that the transport service provide hooks into the progressively reliable protocol so that the application can parameterize how long the protocol should wait before initiating retransmissions.

In past work, retransmissions have been delayed for network-related factors such as congestion and wireless fading rather than for the application's subjective latency and distortion tolerances. For example, TCP's retransmission timeouts are increased as congestion over the network connection increases queueing delays [8]. The effect is to delay TCP's retransmissions in response to network congestion. In addition, it has been proposed that data-link retransmissions be delayed for users who are experiencing wireless fades [36]. Over a time-multiplexed broadcast wireless downlink, retransmissions to users experiencing deep fading are delayed while retransmissions to other users with a cleaner channel are not delayed. The most efficient realization of a progressively reliable protocol would clearly integrate information provided by user tolerances, congestion and fading.

3.3 Cancelling retransmissions of out-of-date ADU's

Our progressively reliable transport service also provides a way for applications to *cancel* retransmissions associated with *out-of-date* or stale ADU's. Multimedia data that has been sufficiently delayed within the transport layer of the sender can become out-of-date. If image data that was initially displayed with errors at the receiver is retransmitted after a delay of many seconds, then fast image browsing will cause the image data awaiting retransmission at the sender to become obsolete. Similarly, retransmissions of interactive video conferencing frames can be delayed by no more than about five frames before becoming out-of-date, assuming a playback point of about 200 ms. Retransmitting out-of-date image data will waste bandwidth and slow down delivery of other time-sensitive image data over the same low-bandwidth wireless access link. Therefore, the ability to cancel unnecessary retransmissions should accompany the ability to delay retransmissions of packets.

We propose that the cancellation primitive be implemented at the transport service interface through a generic *ADU label*. In Figure 7, the sending application marks each ADU with a label. The transport protocol receives these labeled ADU's through the socket buffer interface. As each new ADU arrives, there is *implicit* permission to stop retransmission of any older ADU residing in the transmit buffer that shares the same ADU label. Both the data and

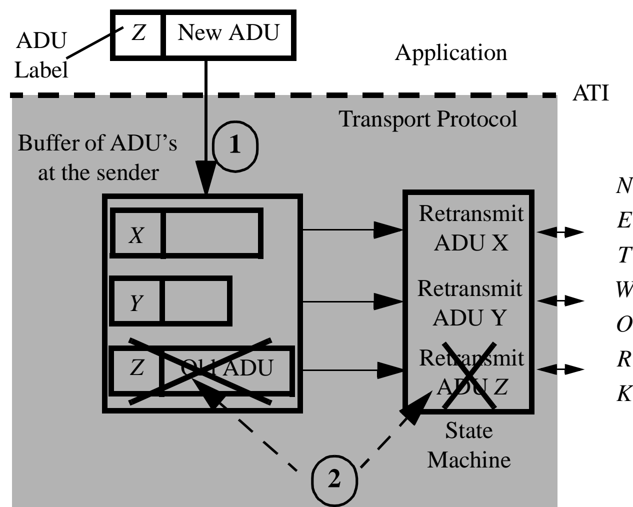

Figure 7. Retransmissions of out-of-date ADU's are *cancelled* using ADU labels. (1) The transport protocol receives the new ADU with label *Z*. (2) Retransmission of the previously cached ADU *Z* is stopped and its old data and state are purged. The protocol then initiates transmission of the new ADU *Z* (not shown).

state associated with the out-of-date ADU are purged. The protocol then initiates transmission of the newer ADU.

Our choice is to associate ADU labels with an implicit cancellation mechanism rather than provide a separate function that must explicitly be called by the application to cancel data. This implicit linkage of labelling and cancellation simplifies a common case encountered by the multimedia application. Newly-generated multimedia is often related spatially and/or temporally (e.g. regions within an image) to older data in the sender's cache. It is natural to cancel transmission of related older data before transmitting the newer data. Since application-level relationships are already expressed through labelling of ADU's, then it is natural to combine cancellation of an older ADU with the arrival of a newer ADU of the same label. We note that explicit cancellation can be emulated through our implicit cancellation mechanism by having the application send a zero-length ADU that only contains the label of the ADU to be cancelled.

Even though ADU labels express application-level relationships, the application's internal semantics are largely hidden from the protocol. ADU labels communicate the minimum information necessary to achieve cancellation. Consequently, the design of the protocol is decoupled as much as possible from the application's design. Given an ADU label, the protocol has sufficient information to identify which packet of application data needs to be cancelled. No further information, such as the data type (audio/video/image) or compression format, is required from the application to identify which retransmissions should be cancelled.

The implicit linkage of cancellation and labelling does not limit the application's freedom to define how ADU labels are interpreted. We give two examples below of how a graphical Web browser can define ADU labels so that they cancel out-of-date retransmission of block-based regions within an image, or arbitrarily-shaped objects within an image. ADU labels can also be used by continuous media applications to eliminate retransmissions of out-of-date speech samples and out-of-date video frames, in effect implementing partial retransmission of audio and video (see Section 3.5).

Figure 8 illustrates how graphical Web browsers could employ ADU labels to implement region-based and/or object-based cancellation of overwritten regions within a scene. In Figure 8(a), an image is subdivided into blocks and the application tags each block with a region-based ADU label. When new activity indicates that a portion of the image should be redrawn, then the sending application generates new ADU's for the affected blocks and transfers these labeled ADU's to the transport service. The underlying transport protocol checks for any queued ADU's that match the incoming ADU labels and stops retransmitting any out-of-date queued ADU's, which in this case correspond to overwritten image blocks. For example, in Figure 8(b), the shuttle's launch causes the block with ADU label *S* to be redrawn. When the sending side of the transport protocol receives the newest ADU with label *S* and finds a matching ADU with label *S*, it terminates retransmissions of the old ADU and initiates transmission of the new ADU corresponding to image block *S*.

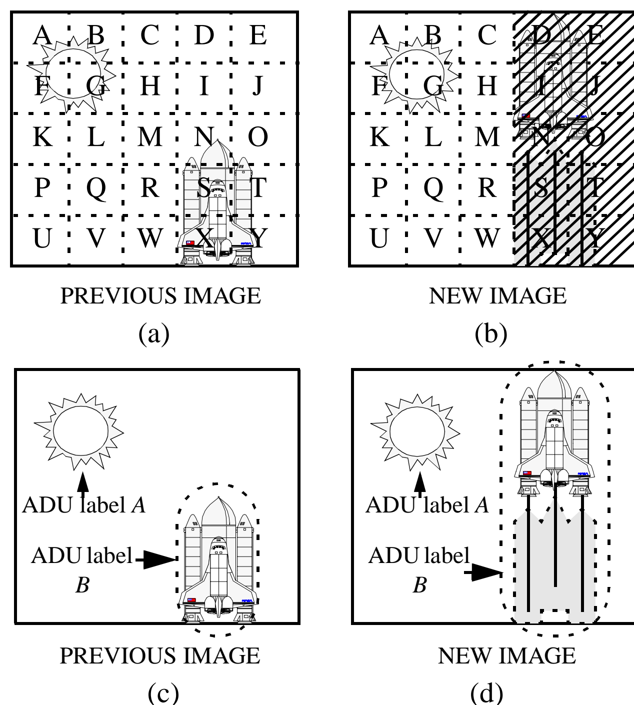

Figure 8. Ways to use ADU labels to implement cancellation: (a) Block-based ADU labelling (b) Retransmissions of out-of-date (shaded) blocks are cancelled by new image data. (c) Object-based ADU labelling (d) Retransmissions of out-of-date objects (e.g. object *B*) are cancelled.

An alternative use of ADU labels is to cancel retransmissions of semantic objects in an image, rather than a block. In Figure 8(c), the image is partitioned using object-based ADU labels, and the shuttle is assigned ADU label *B*. As the shuttle launches, the sending application generates a new ADU *B* that aggregates the launching shuttle and its exhaust flames, as shown in Figure 8(d). The protocol cancels retransmissions associated with the old ADU *B* and initiates transmission of the new ADU *B*. When the new ADU *B* arrives at the receiver, the end user will see the launching shuttle overwrite the old picture of the stationary shuttle.

We emulated the effect of block-based cancellation in Figure 8(a) within our experimental X windows server. We divided the screen into 16x16 blocks, and implemented cancellation for each block. A state machine for each block controlled whether the initially noisy version or the final error-free version of each block's data was written to the frame buffer. Whenever a new image update for a block was received, the state for that block was updated to the "sent initial version" state and a noisy version of that fresh data was written to the appropriate block in the frame buffer. Also, the timer for triggering transmission of the error-free version was reset to zero. The state machine transitioned from the "sent initial version" state to the "sent final version" state after the timer aged beyond the minimum wait time. This also caused the final error-free or refresh version to be written to the frame buffer. An out-of-date refresh version could never be written, since new image data always triggered a state transition back to "sent initial version", effectively cancelling transmission of out-of-date data. Our modified X server intercepted all pixels normally rendered into the frame buffer and diverted them through our state machine, so that all interactions with the display were subject to the cancellation mechanism.

Given this experimental framework, we found that delaying the error-free version by five seconds required cancellation to maintain reasonable response time for a wide range of routine screen interactions. For example, activities such as paging within a text editor or image browsing occur at a pace faster than once every five seconds and frequently invoked cancellation of a cached page or image that had not yet been refreshed. Window manipulation activities like restacking, resizing, and moving of windows also occurred with sufficient rapidity that large unrefreshed areas of the screen were frequently overwritten by new activity, thereby invoking block-based cancellation. When a window was dragged across the screen during an opaque move, each block on the screen that was in the path of that move was redrawn at a rate of many times per second. Such rapid redrawing triggered the cancellation mechanism. These

examples illustrate that even routine interactive operations require careful cancellation if wireless bandwidth is to be conserved and unnecessary retransmissions are to be avoided.

Cancellation was also continuously invoked for video. We observed that if the refresh version of a video frame was delayed longer than one frame time (typically one-fifth to one-tenth of a second on our workstations), then the cancellation mechanism prevented the refresh version of any video frame from ever being transmitted. By setting the refresh delay to five seconds, then each succeeding video frame in effect cancelled retransmission of the previous frame.

When an ADU is cancelled at the sender, the receiving end of the protocol must be informed of this cancellation in order to purge the receiver's state associated with the out-of-date ADU. Otherwise, the receiver's stored state would grow without bound. A convenient mechanism for informing the receiver is simply to propagate the ADU label along with the ADU to the receiving end of the protocol. The arrival of an ADU with a piggybacked ADU label purges out-of-date state at the receiver in the same manner as its arrival at the sending side of the protocol purged out-of-date state. For each ADU label, the receiving end should store the most recent sequence number (assigned by the sending side of the protocol) that has been received for that ADU label. This eliminates forwarding of stale ADU's to the receiving application should ADU's with the same label arrive out-of-order.

Forwarding of ADU labels also enables the protocol to implement parametrizable partial reliability of the final version of an ADU. As mentioned in Section 3.4, one of the parameters available for the application to specify is the level of distortion in the final version of an ADU delivered to the receiving application, i.e. the final version of an ADU is allowed to contain errors. Different levels of partial reliability can be approximated by varying the upper bound on the number of retransmissions attempted for each ADU. Once the upper bound has been reached, then the ADU label is propagated to the receiver to cancel receiver state and terminate further requests for retransmissions.

In related work, there exists a precedent among transport protocols for enabling retransmissions to be cancelled. The Xpress Transfer Protocol (XTP) provides a "no-error" mode that allows certain retransmissions to be suspended [37]. The sending side of XTP marks packets with a NOERR bit to tell the receiving side not to request retransmission for these packets. This example provides one means, though not necessarily the definitive mechanism, for achieving fine-grained cancellation of retransmissions. Most other reliable transport protocols, like TCP, do not allow retransmissions to be stopped once data has been passed to the protocol, since this would not preserve the order of packets guaranteed by reliable stream delivery.

3.4 Flow-based scheduling of packets and their retransmissions

When an application permits retransmitted information to tolerate more latency than the first transmission of an ADU, then the protocol is in effect implementing packet scheduling of two flows. The first flow consists of delay-sensitive original application data, and the second flow consists of delay-tolerant retransmitted packets. This two-flow model can be naturally extended to accommodate a general number of application-defined flows that are jointly scheduled by the protocol for transmission according to their per-flow delay tolerances and Quality-of-Service (QOS) parameters. The advantage of employing a multi-flow scheduling policy that is cognizant of variations in delay sensitivity is improved utilization of limited traffic capacity, since provably optimal scheduling disciplines like Earliest-Due-Date (EDD) [38] can be applied instead of simpler policies like first-in-first-out that are unaware of an application's delay sensitivities.

In Figure 9, we depict a transport protocol that includes a multi-flow packet scheduler that distinguishes between variations in delay tolerances within a flow (i.e. initial transmissions as distinct from retransmissions) as well as delay variations across multiple flows. The scheduler chooses which packet to transmit next from a pool of "eligible" packets. Initial transmissions of packets are immediately eligible for scheduling at their corresponding flow's delay bound (e.g. for flow 1, the scheduler would apply delay bound d_1). Eligibility of *re*transmissions for scheduling is determined by the state machine and depends upon several factors. The state machine must have been informed through an ACK/NAK or a time-out that a retransmission is needed. Further, congestion control and flow control policies will influence the state machine's decision to trigger a retransmission. Also, incoming ADU's may cause the state machine to cancel specified retransmissions at any time. Once the state machine has determined that certain packets must be retransmitted, then these retransmissions are added to the pool of eligible packets and are scheduled according to their corresponding flow's retransmission deadlines (e.g. by d_2 for flow 1). The uniqueness of this approach is that retransmissions are treated as schedulable entities with their own delay bounds.

While a wide variety of flow-based scheduling algorithms, also called service disciplines, have been proposed [39], the class of scheduling policies that *guarantee* delay bounds may not be suitable for progressively reliable packet delivery. Given well-behaved sources and admission control at all access points into a wired backbone network, then it is possible to guarantee end-to-end delay bounds (deterministic or statistical) for certain scheduling disciplines operating at each internal network switch, like EDD, but not for other service disciplines like Fair Queueing and Virtual Clock [40]. However, delay guarantees cannot be made if the scheduler makes its decision based on incomplete information at the

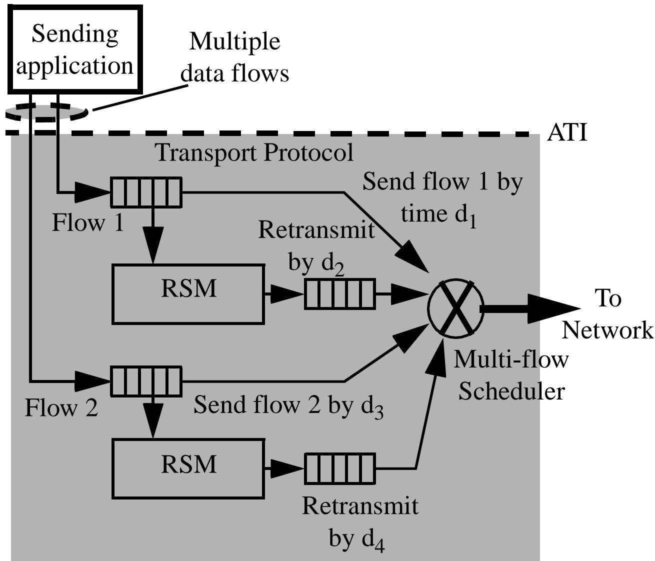

Figure 9. The protocol's scheduling policy distinguishes between the varying delay sensitivities d_i of retransmissions and initial packet transmissions within a flow, as well as variations across multiple application-defined flows. RSM = retransmission state machine.

sender, i.e. by considering only progressively reliable packet traffic and neglecting transmission of TCP/UDP traffic from the same host. Further, if the operating system does not support real-time execution of processes/threads, then the scheduler cannot ensure that delay bounds are not violated. Moreover, the wired Internet backbone will continue to be best-effort for some time to come. Consequently, the QOS scheduling required to guarantee end-to-end delay bounds will not necessarily be supported by intermediate network switches. Finally, the wireless access link will introduce generally unpredictable delay, since random fluctuations in channel fading will introduce fluctuating traffic due to retransmissions. At best, a statistical characterization of the fading may provide a statistical bound on delay, but this seems uncertain given the wide variety of wireless links and causes for fading. For these reasons, our view is that the protocol's scheduler should not guarantee end-to-end delay bounds. Instead, negotiated QOS parameters are interpreted as useful indicators of delay sensitivities that enable the protocol to provide differential service across multiple application-defined flows.

The sender's scheduling discipline should also enforce *fair* sharing of bandwidth among the multiple applications' flows if progressively reliable packet delivery is to be supported. We constructed a two-flow scheduling model of progressive reliability within our modified X server. A noisy version of an image was written immediately to the frame buffer, simulating transmission of delay-sensitive data via a first flow called the interactive flow. The noisy image was eventually overwritten with a clean version of that image, simulating delay-tolerant retransmissions of image data via a second flow called the refresh flow. We observed the well-known problem that a naive priority-based scheduling policy that always gave higher priority to the interactive flow over the refresh flow would, under certain conditions, unfairly *starve* the lower-priority flow. For example, we attempted to run typical bursty windowing applications (e.g. file editing, paging, scrolling, moving/resizing/restacking windows) while streaming video. We found that the continuous video generated enough delay-sensitive traffic so that the refresh traffic for the bursty windowing applications was starved for bandwidth. Error-induced artifacts were never removed, or removed too slowly, in the non-video portion of the screen. A desirable scheduling policy for this scenario would support a fair trade-off between delay-tolerant cleanup of the non-video portion of the screen and delay-sensitive transmission of video.

We introduce a *flow header* to identify individual substreams within the general data stream. 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 that first-time transmissions can tolerate, how much initial corruption can be tolerated, 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). 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.

Cancellation of retransmitted ADU's depends only on the ADU label, and not on the flow header. For example, an ADU with label *X* may be directed along flow 1 by the sending application. Later, the application may wish to send a newer ADU also labeled *X* along another flow 2 and cancel the older ADU in flow 1. Cross-flow cancellation can be applied to hierarchically coded images (say progressive JPEG) that are partitioned into multiple layers that are then sent via different flows serviced with variable QOS. The application can send low frequency coefficients first along a delay-sensitive flow 1, and then later transmit the full set of frequency coefficients along a separate delay-tolerant flow 2. Arrival of the latter set of coefficients can be designed to cause cancellation of retransmissions of the first set, even though the two sets reside in different flows. By basing cancellation exclusively on ADU labels, and not on flow headers, we enable the application to cancel ADU's across flows.

In related work, other end-to-end protocols have supported a limited degree of parameterization or QOS configuration, e.g. OSI/TP4 and XTP [21]. XTP supports intra-

protocol scheduling based on priorities via its "sort" field [37], but this is susceptible to the starvation problem we encountered earlier.

3.5 Partial retransmission of video and audio

The versatility of progressively reliable packet delivery permits it to be configured to implement partial retransmission of continuous media like audio and video. For streaming applications, any video frames or audio samples that arrive after the designated playback point due to network delays will be discarded as out-of-date. Any retransmissions that arrive prior to the playback point can still be used. Depending on the tightness of the playback bound, no retransmissions, a few, or many retransmissions may be attempted. For interactive video conferencing, the playback point is on the order of a couple hundred milliseconds, essentially precluding end-to-end retransmissions. For unicast video playback from a remote video database, the playback point can be on the order of minutes after a frame was initially transmitted. Consequently, a large number of retransmissions may be attempted and video can be streamed over a fully reliable protocol like TCP. Between these two extremes lie other video applications like "live" unicast video that have playback points on the order of seconds to tens of seconds. For these applications, a partial retransmission strategy that stops retransmitting after a finite number of attempts can improve the quality of the displayed video without violating delay constraints.

We implement partial retransmission of continuous media by employing out-of-date cancellation. As a sequence of video frames (or speech samples) is transmitted, we conceptually form a fixed-size sliding window at the sender whose duration is equivalent to the playback delay bound. Frames within the window are immediately retransmitted if necessary. As each new frame is generated, the window slides forward by one frame, incorporating the newest frame into the retransmission-eligible window while also cancelling any retransmissions of the oldest frame that was just eased out of the window. The end result is that retransmissions are only attempted on frames within the window, and terminate after a finite number of attempts once the playback point is reached.[1]

Another aspect of this cancellation scheme is that, if the user pauses the video sequence to view a frozen frame for editing purposes, then the progressively reliable protocol automatically sends the retransmission redundancy needed to remove any errors in the still frame. This simplifies the design of the video application, which no longer needs to explicitly send the paused frame again via a reliable protocol in order to remove any frozen artifacts.

In related work, partial retransmission of wireless audio has been described by several authors [41][42][43]. End-to-end partial reliability at the transport layer has also been proposed [44][45]. One distinction between these partially reliable approaches and progressive reliability is our notion of forwarding an initially noisy packet version followed by successive refinement of this noisy version over time to remove channel-induced errors. Another difference is our ability to delay retransmissions to minimize conflict with delivery of delay-sensitive data within a flow-cognizant scheduling paradigm.

3.6 Progressive image transmission

Another intriguing application for our transport service involves integrating progressive image transmission with progressively reliable packet delivery. We implemented a simple progressive colormap scheme within our experimental two-flow X server. Initially, every block on the screen would be drawn in monochrome. Eventually, every block would be displayed with full color depth. Combining this type of progressive source coding with progressively reliable packet delivery, essentially a type of progressive channel coding, we achieved the following effect: blocks would initially be displayed in noisy black-and-white form, and would eventually be displayed in an error-free color form. This emulated composite progressive system proved to be interactive even when bandwidth was tightly constrained (<100 kb/s) and the BER was severe (1%). Under these same conditions, the normal full-color X server became noticeably non-interactive due to the limited bandwidth. In these experiments, we did not emulate FEC within our X server, nor did we emulate corruption of packet headers. More information about our experimental setup and observations can be obtained from [22].

4. Selected implementation issues for Leaky ARQ

In this section, we discuss the minimal set of architectural modifications that must be made to an ARQ protocol to support progressively reliable packet delivery. At the minimum, a progressively reliable protocol must support the two basic properties of successive refinement of corrupt packets and forwarding of corrupt packets. In Section 4.1, we consider the options for realizing successive refinement within an ARQ protocol. In Section 4.2, we observe that forwarding corrupt packets requires a mechanism that distinguishes between corrupt headers and corrupt payloads. We conclude with a discussion on the feasibility of building progressive reliability on top of UDP or on top of a combination of TCP and UDP. Other implementation issues such as time-out estimation, NAKs vs. ACKs, congestion control, flow control,

1. The idea of using a circular cancellation scheme within our progressively reliable protocol to partially retransmit audio was first observed by Sanjoy Paul, currently at Lucent Bell Labs, in a conversation with the first author.

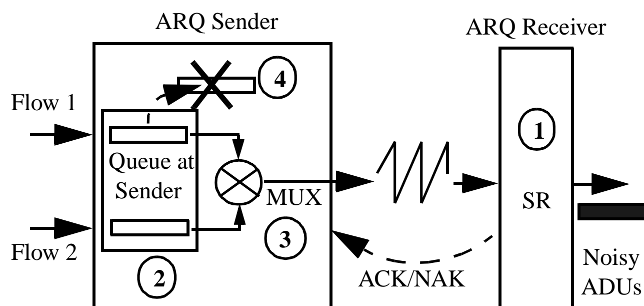

Figure 10. Leaky ARQ is a progressively reliable protocol that alters a traditional ARQ protocol as follows: (1) at the receiver, noisy yet successively refined (SR) packets are delivered to the receiving application. At the sender, (2) retransmissions are delayed (3) multiple flows are scheduled with different delay objectives, and (4) out-of-date retransmissions are cancelled.

and state machine design for both the transmitter and receiver are beyond the scope of this section. The issues of fragmentation and reassembly of ADU's to preserve end-to-end application-level framing were discussed in [22].

Our first observation is that an ARQ protocol's retransmission loop needs to be modified at the receiver to permit intermediate versions of a corrupt packet to be leaked to the receiving application, hence the name *Leaky* ARQ. The next step is to ensure that these noisy packet versions share the property of statistically improving reliability. In order to determine whether the current version of a packet has fewer channel-induced errors than any previously decoded version of that packet, the receiver must maintain some per-packet state or history. Caching dirty versions of decoded packets at the receiver maintains history and also enables us to exploit standard *packet combining* ARQ techniques to achieve progressively improving reliability. We will explain packet combining strategies in the next section. If the more advanced functions of delaying retransmissions, cancelling retransmissions, and flow-based scheduling are also desired, then additional modifications need to be made to the sending side of a standard ARQ protocol. Figure 10 identifies the major modifications to a standard ARQ protocol architecture that are needed to fully realize all of the features proposed for progressive reliability.

4.1 Achieving successive refinement via "packet combining"

We leverage off of traditional *packet combining* ARQ techniques to achieve successive refinement. Packet combining is a technique that uses the history of corrupt retransmissions for a given packet to obtain a better estimate for decoding that packet than the memoryless estimate provided by a single isolated retransmission. Rather than discard noisy packets and their noisy retransmissions at the receiver, packet combining ARQ typically caches these packets at the receiver to be used for future decoding. Packet combining is classified into time-diversity combining and code combining, both explained below.

Caching of a packet's received history reduces the number of retransmissions needed to decode a given packet, and more importantly for our purposes enables successive refinement. For example, consider the common case in which a protocol retransmits copies of the original packet, also called repetition coding. If the receiver caches all noisy copies, then bit-by-bit *majority-logic decoding* can be applied as each new retransmitted copy arrives. For each bit, the majority-logic decoder will decode the value 0 or 1 that has the most occurrences, i.e. that is in the majority [46]. Suppose two noisy copies of a packet (due to the initial transmission and the first retransmission) are cached at the receiver, and a third corrupt copy arrives due to a second retransmission. The packet version produced by majority-logic decoding of these three packets is statistically more likely to have fewer bit errors than any of the individual copies. This simple example shows that packet combining allows the receiver to converge more quickly to an error-free representation of a packet, thereby reducing the number of retransmissions required for decoding each packet. This example also indicates that packet combining is capable of guaranteeing that the most recent estimate of a packet has statistically fewer errors than previous estimates. For this reason, packet combining ARQ serves as the basis for successive refinement in Leaky ARQ.

Majority-logic decoding is a simple example of a broader class of packet combining approaches called *time-diversity combining* [47]. In time diversity combining, decoding of a given packet from multiple cached versions is performed on a bit-by-bit or symbol-symbol basis. For example, given repetition coding and Viterbi decoding, then a soft-decision diversity combiner could compute a weighted energy average derived from the analog pre-detection values of each bit within the multiple received copies of the ADU, and feed these symbol averages into a Viterbi decoder [48]. Repetition coding with majority-logic bit-by-bit decoding is an example of hard-decision diversity combining.

Code combining is an even more powerful form of packet combining than time-diversity combining. In code combining, retransmissions contain FEC redundancy rather than merely repetitions of the original packet's information. A well-known example of code combining is *Type-II Hybrid ARQ* [13]. In this hybrid ARQ protocol, retransmissions alternately contain the original payload (say P_1) or the convolutional parity check (say P_2) computed from the FEC code and P_1. The receiver either caches a corrupted P_1 and then uses the next retransmission P_2 to correct errors in the cached P_1, or caches a corrupted P_2 to correct any errors in the next P_1. Type-II Hybrid ARQ cannot, strictly speaking, guarantee statistically improving reliability since only one

prior retransmission is cached. A sequence of two severely degraded packets P_1 and P_2 can cause a decoded packet version to have more errors than previous versions, so that the number of errors will vary according to channel conditions and not converge towards zero.

Strictly speaking, the entire received packet history must be stored in order to guarantee that future packet versions will be decoded with statistically fewer errors than the previous versions. *Maximum-likelihood (ML) code combining* is the general form of code combining that operates over multiple cached packets to ensure that packets have statistically fewer errors over time. The FEC redundancy in the Nth transmission is conceptually concatenated to the FEC redundancy cached at the receiver from the N-1 previous transmissions (including the original payload). The concatenation of payload plus FEC redundancy forms an FEC codeword that can then be decoded into a single error-corrected payload. A version of ML code combining using fixed-length retransmissions of the same size as the original payload was introduced in [49]. An alternative approach employs variable-length retransmissions that contain *incremental* FEC redundancy generated by rate-compatible punctured convolutional (RCPC) codes [50]. In this scheme, individual retransmissions may not contain sufficient information to independently reconstruct the original message. A hybrid approach that initially retransmits incremental redundancy, followed later by fixed-length payload-size repetitions has also been proposed [51]. A comparison of the efficiency of diversity combining vs. code combining has been made for the case of repetition coding with multiple copy decoding [52]. ARQ protocols that implement packet combining are also called *memory ARQ* [53] [54].

We propose that Leaky ARQ's successive refinement feature be implemented via the simplest form of code combining embodied in Type-II Hybrid ARQ. While it is desirable to cache as much of the received packet history as possible, memory constraints at the receiver may limit our ability to statistically guarantee successive refinement, hence our compromise in choosing limited code combining. Also, end-to-end Type-II Hybrid ARQ is readily implemented given the availability of software implementations of Reed-Solomon FEC coders and decoders on the Internet. In addition, the more advanced techniques of incrementally redundant transmission require maximum-likelihood decoding at the receiver, i.e. a Viterbi decoder. Since Leaky ARQ operates as an end-to-end transport layer protocol, then it is unlikely that it will have access to soft symbol information. Hard-decision Viterbi decoding based on Hamming distance path metrics will have lower performance than soft-decision decoding. Moreover, Viterbi decoding would have to be implemented in software and operate in reasonable time at the receiver.

4.2 Separating header and payload error detection

Corrupt packet payloads can only be forwarded to the receiving application if their corresponding packet headers are decodable. To distinguish between a corrupt packet payload and corrupt packet header at the receiver, we need separate error detection mechanisms for the transport protocol data unit's (TPDU) header and TPDU payload (i.e. encapsulated ADU, possibly fragmented). For this reason, we calculate two checksums per TPDU, the first on the TPDU header only, and the second on the TPDU payload alone. When a TPDU arrives at the receiver, error detection is applied on the header first. If the computed header-only checksum matches the embedded header-only CRC, then the header is error-free and the TPDU payload is successively refined via packet combining. At this point, the receiving side of the protocol calculates the second checksum exclusively on the successively refined TPDU payload and compares it to the embedded payload-only CRC. If the payload-only checksums match, then the transport protocol knows that this TPDU has been delivered without errors, and can update the receiver state machine appropriately as well as inform the sender via acknowledgment that this TPDU has been correctly received. If payload-only error detection fails, then the noisy packet is cached at the receiver for packet combining. In either case, the successively refined TPDU is forwarded to the receiving application even if there are still errors in the payload after packet combining.

4.3 Leaky ARQ as an application-level protocol

In this section, we discuss strategies for implementing Leaky ARQ as an application-level protocol above the existing infrastructure provided by TCP and UDP.

One approach that achieves progressive reliability is to send an initial version of an image via UDP, and a subsequent reliable version via TCP. This (TCP+UDP) solution suffers from several problems that lead to decreased efficiency over wireless links. 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-of-date image data. Retransmissions of TCP packets will uncontrollably steal bandwidth from newly arrived UDP packets, increasing the delay of time-sensitive UDP data. Hence, (TCP+UDP) cannot promise consistent interactivity. Also, TCP can waste scarce wireless resources trying to reliably transport out-of-date information. TCP does not permit us to identify and "cancel" retransmissions of stale data.

Another approach is to build progressive reliability on top of UDP alone, thereby avoiding the difficulties with TCP. An application-level progressively reliable protocol could be

implemented above UDP, much like the application-level Real-Time Protocol (RTP) used by multicast video tools [10]. However, UDP checksumming must be turned off in order to support forwarding of corrupt packet payloads to the application-level protocol. UDP checksums are calculated over the IP pseudo header plus the entire UDP datagram (UDP header + UDP payload) [8]. Turning off UDP checksums can cause the application-level protocol to receive UDP datagrams with corrupt UDP headers and/or corrupt IP pseudo headers.

Error detection must be performed on three layers of header fields, including the TPDU header, UDP header, and IP pseudo-header. Leaky ARQ is responsible for calculating a checksum over the TPDU header. Leaky ARQ's checksum can also be calculated over the UDP header and IP pseudo-header. This approach places the entire burden of header validation on Leaky ARQ.

An alternative is to partition the error detection responsibility. Leaky ARQ could make use of "UDP-lite" [55], a form of UDP that calculates the UDP checksum over the UDP header and IP pseudo header only. In this scenario, UDP-lite acts as an initial filter that only passes UDP payloads that had error-free headers (UDP and lower) up to Leaky ARQ. The task remaining for Leaky ARQ is simply to calculate the checksum on the TPDU header. This UDP-lite approach has several limitations. Both UDP sender and UDP receiver stacks would have to be modified to incorporate header-only checksumming. An additional complication is that header-only UDP checksums may lead to the failure of some pure-UDP applications which were designed to handle packet loss, but not necessarily packet corruption. A potential solution is to calculate header-only checksums exclusively for Leaky ARQ packets, and to practice conventional UDP checksumming for all other UDP datagrams.

If header-only error detection is implemented exclusively at the Leaky ARQ layer, without the benefit of UDP-lite, then the Leaky ARQ receiver must have access to the source IP address and source UDP address. Fortunately, these two fields can be obtained from the socket interface via address structures that are returned with each received UDP datagram [56].

5. Conclusions

We have introduced the notion of progressively reliable packet delivery for delay-sensitive multimedia transmitted over Internet connections with a wireless access link. Traditional ARQ protocols were shown quantitatively to incur exponential delay over low bandwidth high BER links and therefore were unlikely to meet interactive delay bounds. FEC was shown to introduce delay due to overhead and interleaving that could violate interactive delay bounds given sufficiently bandlimited channels, severe fading, and multimedia that is of high volume even after compression. For interactive video conferencing or Web browsing over wireless links, the approach of error-tolerant compression was observed to provide a lower-distortion solution than the more traditional combination of aggressive compression and aggressive FEC. Delivery of corrupt error-tolerant packets was also shown quantitatively to be able to meet interactive delay bounds. In this context, we proposed a progressively reliable transport service that permits corrupt packets to be delivered to the receiving application in increasingly reliable fashion. Progressively reliable packet delivery also allows multimedia applications to delay retransmissions, cancel out-of-date retransmissions, and define multiple flows that are scheduled by the protocol with variable delay constraints. An example was given to illustrate how the protocol could be parameterized to implement partial retransmission of video/audio. The protocol was termed Leaky ARQ because corrupt packets could be leaked to the receiving application. We discussed how to achieve successive refinement by modifying packet combining techniques like Type-II Hybrid ARQ. Finally, we observed that implementing Leaky ARQ as an application-level protocol above UDP requires header-only error detection on UDP headers and IP pseudo-headers.

6. Acknowledgments

We wish to thank Richard LaMaire and Srini Seshan at IBM T.J. Watson Research Center for their helpful reviews.

7. References

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

[2] 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.

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

[4] I. Wakeman, "Packetized Video -- Options for Interaction Between the User, the Network and the Codec," *The Computer Journal*, vol. 36, no. 1, pp. 55-67, 1993.

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

[6] 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.

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, *HTTP/1.1*, RFC 2068, January 1997.

[8] W. Stevens, *TCP/IP Illustrated, Volume 1*, Addison Wesley, 1994.

[9] J. Bolot, "End-to-End Packet Delay and Loss Behavior in the Internet," *SIGCOMM*, pp. 289-298, 1993.

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

[11] T. Turletti, C. Huitema, "Videoconferencing on the Internet," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 340-352, June 1996.

[12] A. Tanenbaum, *Computer Networks, 2nd edition*, Prentice-Hall, 1989.

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

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

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

[16] D. Towsley, "The Stutter Go Back-N ARQ Protocol," *IEEE Transactions on Communications*, vol. 27, no. 6, pp. 869-875, June 1979.

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

[18] D. Weissman, A. Levesque, R. Dean, "Interoperable Wireless Data," *IEEE Communications Magazine*, vol. 31, no. 2, pp. 68-77, February 1993.

[19] E. Ayanoglu, S. Paul, T. LaPorta, K. Sabnani, R. Gitlin, "AIRMAIL: A link-layer protocol for wireless networks," *ACM Wireless Networks*, vol. 1, no. 2, pp. 47-59, February 1995.

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

[21] 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.

[22] R. Han, "Progressively Reliable Packet Delivery for Interactive Wireless Multimedia," Ph.D. Thesis, University of California at Berkeley, Dept. of Electrical Engineering and Computer Sciences, May 1997.

[23] 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.

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

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

[26] M. Mouly, M. Pautet, *The GSM System for Mobile Communications*, 1992, ISBN 2-9507190-0-7.

[27] E. Biersack, "Performance Evaluation of Forward Error Correction in ATM Networks," *SIGCOMM*, pp. 248-257, 1992.

[28] A. Albanese, J. Blomer, J. Edmonds, M. Luby, "Priority Encoding Transmission," TR-94-039, International Computer science Institute, Berkeley, CA, August 1994.

[29] W. Xu, J. Hagenauer, J. Hollmann, "Joint Source-Channel Decoding Using the Residual Redundancy in Compressed Images," *IEEE International Conference on Communications (ICC)*, vol. 1, pp. 142-148, 1996.

[30] 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.

[31] 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.

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

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

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

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

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

[37] W. Strayer, B. Dempsey, A. Weaver, *XTP: The Xpress Transfer Protocol*, Addison-Wesley, 1992.

[38] L. Georgiadis, R. Guerin, A. Parekh, "Optimal Multiplexing On a Single Link: Delay and Buffer Requirements," IBM TJ Watson Research Center, Research Report RC 19711 (97393), Aug. 1994. Short version appeared in Proc. INFOCOM 94.

[39] C. Aras, J. Kurose, D. Reeves, H. Schulzrinne, "Real-Time Communication in Packet-Switched Networks," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 122-139, January 1994.

[40] H. Zhang, S. Keshav, "Comparison of Rate-Based Service Disciplines," *SIGCOMM*, vol.21, no.4, pp.113-121, 1991.

[41] E. Malkamaki, "Performance of the Burst-Level ARQ Error Protection Scheme in an Indoor Mobile Radio Environment," *IEEE 44'th Vehicular Technology Conference*, vol. 3, pp. 1412-1416, June 1994.

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

[43] Y. Hayashida, N. Sugimachi, M. Komatsu, Y. Yoshida, "Go-Back-N System With Limited Retransmissions," *Eight Annual International Phoenix Conference on Computers and Communications*, pp. 183-187, 1989.

[44] B. Dempsey, W. Strayer, A. Weaver, "Adaptive Error Control for Multimedia Data Transfers," *International Workshop on Advanced Communications and Applications for High Speed Networks*, pp. 279-288, 1992.

[45] R. Marasli, P. Amer, P. Conrad, "Retransmission-Based Partially Reliable Transport Service: an Analytic Model," *INFOCOM*, vol. 2, pp. 621-629, 1996.

[46] 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.

[47] 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.

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

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

[50] 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.

[51] 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.

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

[53] P. Sindhu, "Retransmission Error Control with Memory," *IEEE Transactions on Communications*, vol. 25, no. 5, pp. 473-479, May 1977

[54] 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.

[55] Steve Pink at SICS, personal communication.

[56] W. Stevens, *UNIX Network Programming*, Prentice Hall, 1990.