Proc. Multimedia Computing and Networking, San Jose, CA, Jan. 29-31, 1996

Asymptotically Reliable Transport of Multimedia/Graphics Over Wireless Channels

Richard Y. Han and David. G. Messerschmitt

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

Abstract: We propose a multiple-delivery transport service tailored for graphics and video transported over connections with wireless access. This service operates at the interface between the transport and application layers, balancing the subjective delay and image quality objectives of the application with the low reliability and limited bandwidth of the wireless link. While techniques like forward-error correction, interleaving and retransmission improve reliability over wireless links, they also increase latency substantially when bandwidth is limited. Certain forms of interactive multimedia datatypes can benefit from an initial delivery of a corrupt packet to lower the perceptual latency, as long as reliable delivery occurs eventually. Multiple delivery of successively refined versions of the received packet, terminating when a sufficiently reliable version arrives, exploits the redundancy inherently required to improve reliability without a traffic penalty. Modifications to acknowledgment-repeat-request (ARQ) methods to implement this transport service are proposed, which we term "leaky ARQ". For the specific case of pixel-coded window-based text/graphics, we describe additional functions needed to more effectively support urgent delivery and asymptotic reliability. X server emulation suggests that users will accept a multi-second delay between a (possibly corrupt) packet and the ultimate reliably-delivered version. The relaxed delay for reliable delivery can be exploited for traffic capacity improvement using scheduling of retransmissions.

Keywords: Multimedia, Wireless, Graphics, ARQ, Reliability, Transport Protocol, Transport Service

1. MOTIVATION

As demand for wireless portable access to multimedia services (e.g. windows-based graphics and video) expands, the networking community is faced with a new challenge: how to provide rapid and sufficiently reliable end-to-end delivery of high bandwidth multimedia over the limited traffic capacity of multi-user wireless links. The Xerox PARC MPad system [1] and the Berkeley InfoPad [2] project are addressing this challenge. User expectations of high subjective image quality and real-time interactivity for certain applications conflict with the reality of traffic conditions on multiple access wireless links, specifically the heavy burst bit error rates (BER) and low bandwidth. Techniques to improve reliability via forward error correction (FEC), interleaving, and retransmission introduce latency, a problem that is more pronounced with limited bandwidth. This poses the problem that low latency (for interactivity) and high reliability (for subjective quality) are fundamentally incompatible under high traffic conditions. And yet, in contrast to data, we observe that graphics and video can typically be displayed with error-induced artifacts without undue penalty, if those artifacts do not persist indefinitely.

We propose to achieve low latency and high reliability simultaneously, without a penalty in traffic

capacity, but at the expense of introducing a new transport service primitive called *multiple delivery.* This primitive delivers multiple possibly corrupt but increasingly reliable versions of a packet to the receiving application, invisibly to the source. The application has the option of taking advantage of the earlier-arriving corrupt packets to lower the perceived latency, but eventually replaces them with the *asymptotically reliable* version. Coupled to this transport service, primitives are proposed to give finer-grained control over the trade-off between retransmissions and bandwidth-induced delay; these allow the application to tailor the service to more closely meet its joint latency and reliability targets without wasting bandwidth yet isolate the application from the protocol implementation. Multiple delivery can exploit the retransmissions of existing reliable transport protocols by forwarding intermediate corrupt packets, rather than by discarding this valuable and already available information, without traffic penalty. The protocol implementation can also be improved by *packet combining* to support successively improved reliability; this reduces the number of retransmissions (and thus traffic). Finally, multimedia datatypes typically do not require 100% asymptotic reliability, a property readily exploited (for higher traffic capacity) with packet combining by multiple delivery, but beyond traditional retransmission implementations.

1.1 Characteristics of windows-based multimedia

.Windows-based graphics includes operations like text editing, window moving, resizing, and restacking, menu pull-down, scrolling, and display of multimedia datatypes like images, video and animation. Their behavior is categorized as either event-driven and highly bursty (e.g. window manipulation, text editing) or continuous-media in nature. A trace of both forms of behavior-- pixel traffic measuring an X windows server writing into its frame buffer -- is illustrated in Figure 1. In this paper, we presume that graphics is treated as a pixel map, encoded and compressed much like video, as in the InfoPad. This has advantages such as platform independence and robustness to a high error-rate environment. The latter results because corrupted data can be profitably used to display a version with corrupted artifacts as illustrated in Figure 2. Pixel-coded graphics can appear quite legible despite the 10^{-2} raw BER that is typical of error bursts in wireless links [3][4]. Of course, the tighter the compression, the less friendly the coded data is to corrupt presentation, so we limit ourselves to those pixel-coded cases where corrupt information is still useful. However, we would not want such artifacts to be displayed permanently in static portions of the display, and hence require that much higher reliability be guaranteed asymptotically. This may or may not be complete reliability, such as the Internet suite's Transmission Control Protocol (TCP) [5] delivers for data (and would be desired for object-based graphics representations like X).

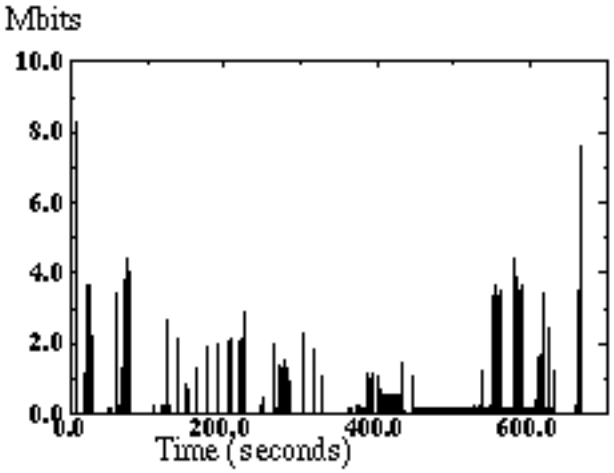

Figure 1. A trace of X server pixel traffic measured at 8 bits/ pixel rendered into a color frame buffer. Typical window manipulation (editing, paging, ...) lasts until ~350 sec. Then video in a small window is graphed from 350-550 sec. Mixed windows and video activity begins after ~550 sec.

Application use of corrupted information is almost universal in networks dedicated to audio, such as circuit switched networks like the public telephone network. Examples similar to the wireless access links we consider here include the IS-54 digital cellular TDMA telephony standard [6]. Speech is coded

using a VSELP vocoder, and the more important class-one bits are protected via a rate one-half convolutional coder, while the less important class-two bits have no error protection. The decoder uses both classes of bits to reconstruct the speech without regard to corruption, where the coding is specifically designed to be robust in the face of high error rates. The IS-95 DS-CDMA digital cellular standard also supports corruption-tolerant service because it recognizes that certain service options can satisfactorily handle some bit errors [7]. Both standards recognize that the heavy BER and limited bandwidth of the wireless link places a premium on recovering as much useful information as possible from data, even where it is corrupted.

Graphics and video differ from audio, however, in the characteristic that data can be persistent, visible to the user for an indeterminate period. Corruption-induced artifacts in such persistent data are much more objectionable than transitory artifacts in audio or in continually changing portions of a visual presentation. Thus, it is important to insure that persistent data is asymptotically reliable; that is, unchanging portions of the visual field are presented without visible artifacts even where the transport medium is extremely unreliable.

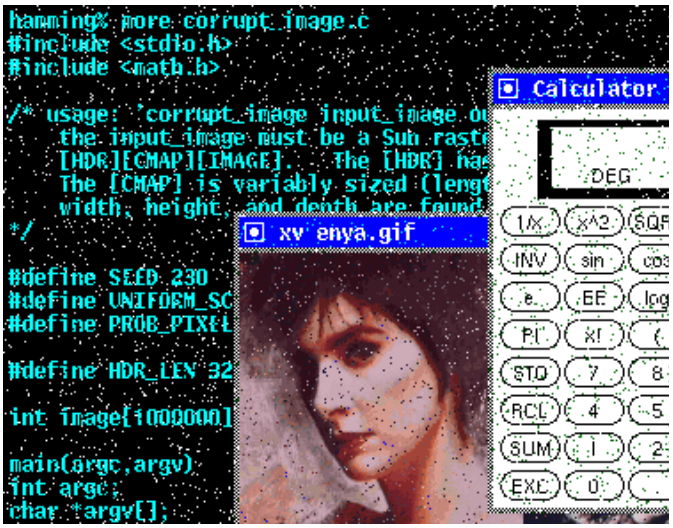

Figure 2. Corrupted Windows-based Graphics at $4x10^{-2}$ BER for 8-bit per pixel colormap data.

1.2 Estimate of single-packet latency

If we insist that only reliable data will be displayed, considerable latency is introduced by either FEC or retransmissions on a transport medium with a high bit error rate and limited bandwidth. As an example, consider a 1000-bit packet. Assume a worst-case burst BER of 10^{-2}, independent Bernoulli errors, a 1 Mbit/s wireless channel. The number of retransmissions N before the first packet arrival with full reliability has a geometric distribution, and the average number of retransmissions is E[N] = 23163. If instead only a 100-bit header requires reliable delivery, then E[N] falls to 2.7. Reliable delivery by repetition can cause intolerable delay on unreliable channels, but delivery of a corrupted packet can incur much less latency. This analysis optimistically assumes Selective-Repeat, where only lost packets are retransmitted. Selective-Repeat is the most efficient retransmission protocol for throughput, bounding all other retransmission schemes like Go-Back-N and Stop-and-Wait [8]. TCP has a throughput bounded by the performance of Go-Back-N and hence Selective-Repeat (for the case of constant retransmission time-outs) [9][10], and thus will do worse than the above analysis.

The total one-way latency consists of transmission, queueing, and endpoint processing delay. The transmission delay of 1 millisecond is additive to an average packet processing delay of approximately 10 ms (where we have conservatively scaled the results from [11] by ten to account for today's faster CPU's, though memory access times have not increased at the same rate). Hence, cumulative per-packet delay is at least 11 ms. To provide a 100 ms response latency, E[N] should be no more than 9. This requirement becomes more stringent once multi-hop queueing and propagation delays and back-channel acknowledgment delay are added, so that the upper bound on E[N] edges uncomfortably closer to unity. Forwarding corrupt packet payloads to the application gives a reasonable hope of meeting this latency bound, since only reliability of the header is required.

Block or convolutional FEC can be added, trading redundancy for lower decoded error rate and hence fewer retransmissions. The redundancy will increase traffic and delay, although not as much as the retransmissions they displace. Essentially error control using these codes is much more efficient than the "repetition coding" inherent in reliability protocols. For example, the Reed-Solomon(64,32) code [12] lowers the BER from 10^{-2} to 10^{-8} at the cost of doubling the bandwidth for independent errors, and E[N] essentially falls to unity. However, this does not consider the correlated burst errors caused by multipath fading, shadowing, and co-channel interference in cellular systems, and block codes do considerably worse in these correlated error environments [13][14][15] unless interleaving is used. However, burst error events on the indoor wireless channel caused by slow-moving interferers may last from hundreds of milliseconds up to several seconds, rendering interleaving infeasible for interactive applications.

1.3 Estimate of multiple-packet latency

While the single-packet latency gives a hint of design trade-offs, it is by no means the full story, because user activity typically results in bursts of multiple packets, and the latency to represent the full set of packets is the appropriate perceptual latency metric. For example, when the user of a graphics application pulls down a menu or updates a window, he or she does not perceive a single packet delivery but rather the display of a reasonable representation of the entire screen area affected by the one user action. Typical bandwidth-induced delays are shown in Table 1 for a fairly small image, say a 8-bit color-

Table 1: Latency due to limited bandwidth for 200x200 8-bit/pixel image

Channel rate (kbit/s)	10	100	1000
No compression	32 sec	3.2 sec	.32 sec
Compression by 10:1	3.2 sec	.32 sec	32 ms
Compression by 100:1	.32 sec	32 ms	3.2 ms

mapped 200x200 pull-down menu, as well as for a 10:1 and 100:1 compression ratio, at three bit rates (range taken from [16]) with no FEC redundancy or retransmissions. Little margin is available for additional FEC or retransmissions, except with the most aggressive compression assumptions and highest channel rate. (Note that compression typically magnifies the effect of a bit error, and hence makes the BER requirement more stringent.) A rate one-half FEC code will double the latency shown. For either 10:1 compression at 1 Mbit/s, or 100:1 compression at 100 kbit/s, this results in a bandwidth-induced latency of 64 ms. To meet our interactivity bound, this effectively means the majority of information needs to arrive correctly the first time, placing a great expectation on FEC, especially with correlated errors. (Interleaving to combat burst errors would increase the latency further.)

In summary, FEC is needed to keep the number of retransmissions reasonable on unreliable channels, but the latency cost of this redundancy for updating reasonable areas of the screen is excessive over many severely bandlimited wireless links. Alternatively, reliability could be increased at the expense of traffic capacity by reducing the interference. There seems to be no overall acceptable solution, due to the undesired coupling of latency and reliability requirements.

2. MULTIPLE-DELIVERY TRANSPORT SERVICE

Techniques for improving on the raw BER of an inherently unreliable wireless medium inevitably involve redundancy in the form of FEC or retransmissions. This redundancy results, in effect, in unreliable versions of a packet arriving early at the receiver. Such early versions can be exploited by passing them to the application for possible display, thereby reducing latency and improving interactivity, with the implication that the receiving application must be prepared for delivery of two or more (increasingly reliable) versions of the *same packet*. We call this a *multiple-delivery transport service*. The

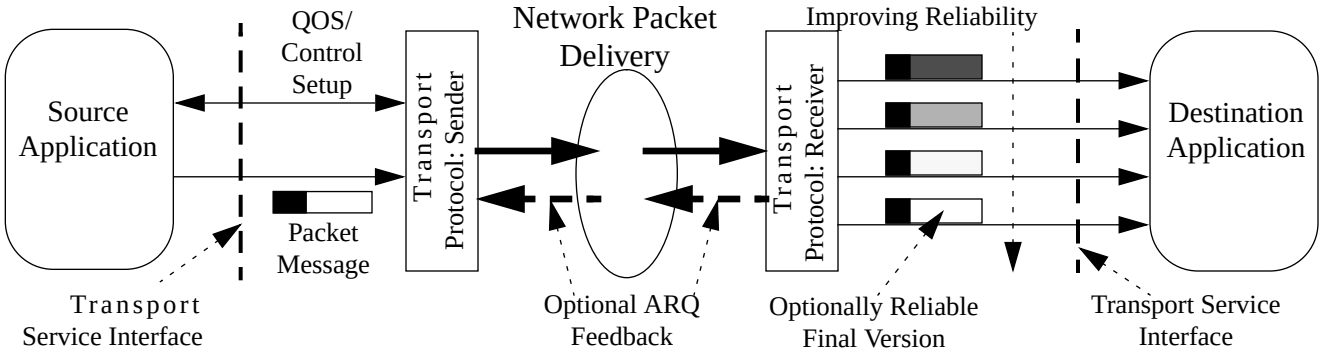

Figure 3. General Multiple-Delivery Transport Service Model: the transport layer delivers multiple versions of a packet, which improve in correctness over time, optionally ending with a reliable final version.

goal is for the latency perceived by the user to be determined by the arrival of the first header-intact packet, possibly with bit errors in the payload. The level of reliability of the last delivered version can then be designated independently of the user-perceived latency. Multiple delivery service can also be exploited for certain types of video coding [17], also to the benefit of lower perceptual delay for interactive applications like video conferencing and windows-based graphics.

The multiple-delivery transport service is illustrated in Figure 3. The application may receive more than one version of each packet, where each version is guaranteed to be higher quality than the previous (as measured statistically by BER or some other appropriate metric). From the application perspective, multiple delivery has three major properties: a corrupt packet payload can be delivered; multiple versions of each packet may be delivered; and the quality of these successive packet deliveries improves over time. Regarding the latter property, it would make no sense to define a service in which a packet is delivered with lower quality than earlier deliveries; rather, it would be preferable to skip the delivery.

Multiple delivery is a transport *service*, as distinguished from a transport *protocol* [18], since it defines only the behavior at the application-transport interface (ATI) between the application and transport, at both the source and destination. The service imposes no assumptions on how increasing reliability is achieved, whether by packet copies, error coded copies, or incremental redundancy. The transport service interface is analogous to the socket programming interface in UNIX [19].

Multiple-delivery transport service can subsume both the present single-copy unreliable delivery or single-copy reliable transport by simply configuring the number of deliveries to one, as well as the quality of that one delivery. The superset of greater-than-one delivery options offers the application greater flexibility in tailoring the transport service's resources, such as bandwidth and buffering, to efficiently meet its joint delay and reliability quality-of-service (QOS) objectives. The application could reproduce a similar service by simulcasting over both an unreliable and a reliable transport service, but at considerable expense in the use of resources. Similarly, an implementation based on UDP-like service [5] alone would require application-level primitives to support reliability. This forces each application to understand and program complex policies like adaptive window resizing and retransmission timeouts, resulting in suboptimal individual implementations. In addition, implementing the service within the transport allows the exploitation of special characteristics of transport media like wireless. (As a previous example, exploiting the TCP transport protocol over the final wireless link can substantially improve its

throughput [20], basically by eliminating many latency-inducing end-to-end retransmissions.).

Table 2: Multiple-delivery transport service elements.

Source to transport		Transport to destination	
Field	**Subfields**	**Field**	**Subfields**
Application data unit (ADU)	ADU header	Application data unit (ADU)	ADU header
	ADU payload		ADU payload
ADU correlation label		ADU sequence number	
ADU flow header			

To support multiple delivery, configuration information is communicated between the source application and the underlying transport protocol across the ATI, including such QOS parameters as reliability of the first (and especially last) delivery, the number of deliveries, the latency objectives for the first and last deliveries, etc. The chosen configuration must also be communicated to the destination application, so that delivered packets can be interpreted and their quality anticipated.

We now describe some particulars of the proposed transport service, listing elements passing across the interface in Table 2. These interface elements are described in turn below.

2.1 Application-level framing

Because of the delivery of corrupt packets, message-passing semantics is required between the transport layer and application layer. The source application must frame its data into a sequence of variable-sized messages, called *application data units* (*ADUs*) [21], and this framing structure must be preserved at the destination. One reason is that delivery of corrupt message versions to the receiving application can invalidate any embedded size information within an ADU, since the ADU's internal length field may be corrupted. Without any framing information or special support from the transport layer, once an ADU's internal length field is corrupted, the application loses track of where the next message begins and is unable to recover future ADU's from the stream of delivered bytes, as pictured in Figure 4. Another reason is that successive refinement operates on discrete ADU's and would make no sense if applied to a continuous stream. Finally, the notion of sequence numbers and correlation labels described in Section 2.2 and Section 2.3 also depend on the ADU concept. ADU framing need not be replicated precisely within the transport protocol, where segmentation and reassembly may occur.

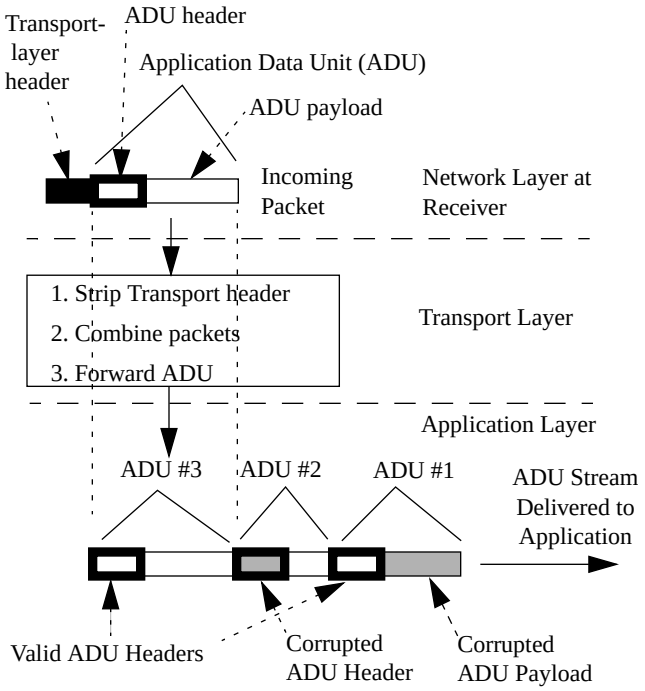

Figure 4. Application-level framing is needed to support forwarding of corrupt payloads. Lack of framing support from the transport service means the application can lose track of where a message begins.

Each ADU is partitioned into two elements: the *ADU header* and the *ADU payload*. Only the length and location of these elements is visible to the transport service. The ADU header accounts for a characteristic of continuous-media applications that an ADU typically contains information that, if

corrupted, will invalidate the use or display of that ADU (the header), and also information that can be corrupted without rendering the ADU useless (the payload). An example would be a video or graphics encoding, where information about the location on the screen is inviolate, but the representation of the pixel values is not. An ADU with a corrupt ADU header is not delivered to the destination application, while an ADU with a corrupt ADU payload may be delivered if it meets the QOS objective for that delivery. We propose that the ADU header, unlike the payload, be a fixed configurable length, which may be zero.

2.2 ADU sequence numbers

Internally, the transport service will associate a sequence number (defined implicitly by the order of delivery of source ADU's) with each source ADU, in order to keep track of retransmissions. We propose that these ADU sequence numbers be furnished to the destination application. This service, which comes at no cost, simplifies the application task in identifying the multiple deliveries of the same ADU, since they will have the sequence number, and may reduce the size of the ADU header.

2.3 Implicit annihilation of stale ADU's

.An optional primitive of the transport service is the annihilation of stale ADU's. This addresses a characteristic of continuous-media sources, such as graphics and video, that ADU's can become *stale*, meaning that they will not be used when delivered to the receiving application. Examples are regions of a graphics screen that have since been updated, or a video pause frame after the video resumes. Transport resources are saved if stale ADU's and all their associated state are purged by the transport in violation of QOS objectives. We propose therefore an *implicit annihilation* primitive, supported by an ADU *correlation label*. The convention is that, given two ADU's with the same correlation label, the one with a lower sequence number is presumed to be stale. For each ADU provided to the transport service, there is an implicit permission to purge (stops retransmission and destroy any state) any other ADU with the same correlation label, as shown in Figure 5. Anywhere within the transport service, stale ADU's can be identified by comparing the sequence numbers of all ADU's with the same correlation label without knowledge of application-specific semantics. The label can be propagated with the ADU for the entire connection, enabling the optional purging of stale ADU's with any transport link (although it is likely only a wireless access link would bother). Forwarding of the correlation label to the destination application is recommended, because it may obviate information that would otherwise consume space in the ADU header.

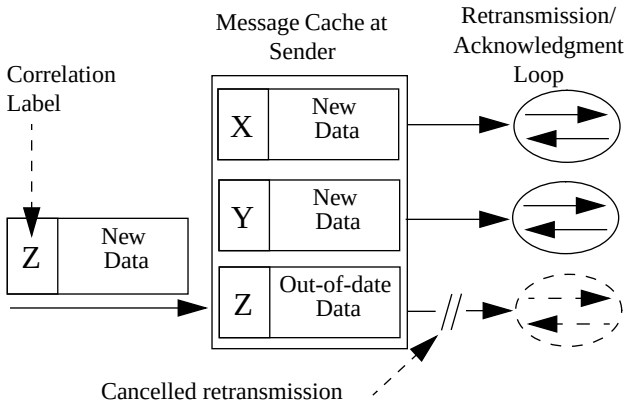

Figure 5. Implicit annihilation uses correlation labels to stop out-of-date retransmissions. The incoming message with correlation label Z cancels retransmission of any previous message with the same label.

As an example of the correlation label, it could be associated with a particular block region of the screen in a graphics-based application. Not only would the correlation label signal to the destination application the region of the screen represented by a given ADU, but it would also identify the temporally freshest representation. Since the transport service may still be consuming considerable resources to deliver a reliable version of a stale ADU, it is desirable to immediately purge that ADU

The destination application must be able to identify and deal with stale ADU's itself for two reasons.

First, the transport service is not obligated to purge stale ADU's. Second, even if it tries it may not always be successful because of transport delay jitter. Another reason for sequence numbers delivered to the destination application is therefore to allow the destination application to detect stale ADU's.

2.4 Traffic shaping of retransmissions

As will be mentioned in Section 3, the desirable spread in time between the earliest and latest delivery of ADUs can be surprisingly large for certain forms of multimedia. This implies that there is actually considerable flexibility in the timing of retransmissions. (This property is directly due to the multiple delivery service, as a single delivery service places a premium on rapid delivery for interactivity.) There is considerable benefit to scheduling retransmissions in accordance with traffic fluctuations. Done properly, this can result in traffic smoothing and a substantial increase in traffic capacity. An additional benefit on time-varying wireless access channels is that retransmissions can be rescheduled to coincide with favorable propagation conditions. Further details are given in [23]. In Figure 6, we illustrate how the transport might spread out retransmissions in time, rather than bunch them up so that they delay delivery of the initial data burst. As part of the configuration information passed across the ATI, the application can in the most general case specify when each retransmission should begin following the initial header-intact reception, basically by describing a function in time. A simpler approach would be to specify a retransmission deadline.

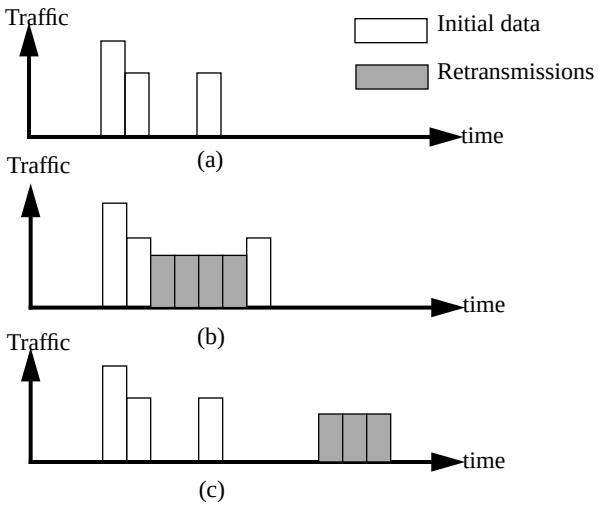

Figure 6. Traffic shaping of retransmissions: (a) initial traffic burst over ideal (high bandwidth/low BER) channel (b) retransmissions delay some initial data over non-ideal (low bandwidth/high BER) channel (c) retransmissions are delayed so they don't slow delivery of the initial burst.

2.5 Flow header

As described in [22], it is important for a source continuous-media application to partition ADU's into different service classes based on QOS requirements. These flows or substreams are the basis of joint source/channel coding on wireless access links, increasing traffic capacity. For this reason, we include a flow header, similar to IP V.6.

3. EXAMPLE: X WINDOWS SERVER

While a multiple delivery transport service has not yet been fully implemented, in the InfoPad project we are implementing aspects, as well as coordinated continuous media applications. One such application is asynchronous video [17], and another is a split X windows server for graphics. We describe briefly here the X server, and how multiple delivery service is helpful.

We have modified an X11 Windows Release 6 server to write an initially corrupted pixel image into a frame buffer, and later with some fixed delay D to write the reliable representation, thus emulating a two-delivery model within the application. We find that D less than one-half second is actually deleterious, resulting in an annoying subjective "flicker" effect in which the cleaning up of images distracts the user from inspecting the content. In fact, as the user is reading newly displayed text or perusing an image or menu, we find it subjectively preferable to send the reliable image much later, rather than as soon as possible after the initially corrupt image representation. Delays on the order of about 5 seconds were found to be a reasonable compromise between "flicker" and rapidly reliable delivery. The general

magnitude of the delay, on the order of multiple seconds, gives the delivery mechanism great flexibility concerning when to deliver the reliable version. The transport protocol can exploit this characteristic by spreading its retransmissions in time on a much greater time scale than previously appreciated.

There are many benefits to spreading retransmission redundancy over an interval of seconds. First, from Figure 1 windows-based manipulation generates large bursts of pixel data in a short time. By removing most of the retransmission and FEC redundancy during the initial burst, the first relatively unreliable delivery does not have to contend with retransmissions and FEC for bandwidth, increasing the perceived latency. This is especially advantageous when the transport protocol window is smaller than the burst size, so that retransmissions would otherwise consume bandwidth while much of the original burst's data awaits its first transmission. This substantial delay also makes it more probable that the data becomes stale, saving transport resources (presuming its annihilation). For example, window operations like paging, scrolling, or fast Web/image browsing all overwrite the same region of the screen relatively quickly, obviating the need for delayed reliable delivery. The same reasoning can be applied to video, which transmits at a rate of 30 frames/sec. No retransmissions are necessary until the user pauses the video, or for stationary background areas.

To emulate asymptotically reliable delivery, we implemented a refresh algorithm which periodically cleans the screen of all artifacts. This refresh algorithm acts as a (poorly tuned) application-level reliability protocol, and furnishes the second and final (reliable) delivery. We assumed a back-channel whose acknowledgments provide enough data to target our refresh data, so that we are not forced to deal with persistent open-loop refresh. For non-continuous bursts, the closed-loop refresh data roughly mimics the trace from Figure 1, except that bursts are translated forward in time, and somewhat reduced in size due to annihilation. For continuous media, this annihilation effect became especially pronounced, and refresh data is often not needed since the delay D between first and final versions is larger than the frame interarrival time. Overall, when comparing the interactivity of the original X server with that of the initial-followed-by-refresh X server, preliminary results indicate that grouping the retransmissions too closely with the initial transmission, in this case within half a second, started to affect the responsiveness of mouse-driven actions like window dragging and menu pulldown, especially when bandwidth was lowered to about 500 kbit/s. Since the refresh data emulates a retransmission caused by noise, this confirmed our intuition that traffic shaping of retransmissions would be beneficial in a more realistic burst-error environment. We plan to replace this application-level refresh algorithm with an implementation of an asymptotically reliable transport service embedded within an operating system kernel and test this within the InfoPad system.

4. PROTOCOL IMPLEMENTATION ISSUES: LEAKY ARQ

The implementation of a multiple-delivery transport protocol raises several interesting questions. How does the application utilize corrupt packets? How does the transport distinguish corrupt from lost packets? How does the transport achieve monotonic quality improvements with multiple deliveries? We discuss some implementation considerations here.

One configuration option for the multiple delivery transport service is asymptotic reliability: is the final delivery guaranteed to be correct, or is some low level of residual corruption acceptable? Is the maximum number of deliveries fixed, bounded by a configuration parameter, or indeterminate? If the underlying network layer offers QOS guarantees, the number of deliveries is fixed, and residual corruption of the last delivery is allowed, then packet combining (see Section 4.2) suffices without ARQ. Since continuous-media services like audio, video and graphics can tolerate some residual corruption,

this suggests that ARQ be avoided altogether for these applications. Nevertheless, ARQ may be valuable on time-varying wireless access links, where QOS guarantees will be difficult or expensive.

The other option in multiple delivery service is asymptotically guaranteed reliability, for which ARQ will be required and the number of retransmissions (although not necessarily deliveries) will be indeterminate. We call ARQ integrated with packet combining, forwarding of corrupt information, multiple deliveries, and guaranteed reliability *leaky ARQ*.

Classical ARQ retransmission schemes include Selective-Repeat, Go-Back-N, Stop-and-Wait, TCP, multi-copy strategies, and other hybrids [8] [24] [25] [26]. From Section 2, leaky ARQ must handle forwarding of corrupt payloads, application-level framing, successive refinement via packet combining, and implicit annihilation of stale frames. We concentrate on these features of leaky ARQ, but don't address other important issues such as a complete time-out strategy, NAKs vs. ACKs, adaptive resizing of congestion and flow control windows, and state machine design for both the transmitter and receiver.

4.1 Detecting ADU loss and corruption

An ADU that is never delivered is said to be lost, and if a delivered ADU has errors in the ADU payload it is said to be corrupted. The distinction is greatest in high BER environments like wireless, where loss can be due to errors in the ADU or transport service header (as well as congestion-induced buffer overflow) and corruption is due to errors in the ADU payload. Since reliable delivery is not provided by most dedicated circuit switched networks, there is ample previous work on achieving high subjective quality for continuous-media applications in the presence of residual errors. For congestion-induced loss, error robustness or error concealment algorithms [27] [28] are designed for packet video. To deal with packet loss (due to a corrupted header), the IS-54 digital cellular telephony standard incorporates a six-state decision model that changes its masking behavior according to how many consecutive packets have suffered the equivalent of corrupted-header loss. For corrupted payloads, IS-54 simply plays them to the user, and the voice compression algorithm is carefully designed to be robust to this corruption. Section 3 describes our own experimentation with corrupted graphics payloads.

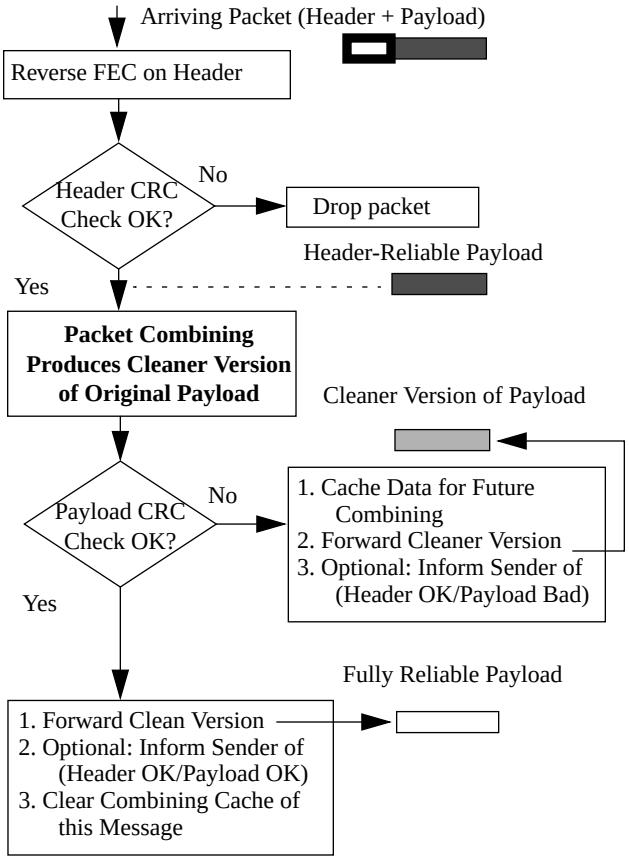

Figure 7. Error detection is performed separately on packet header and payload. Only header-reliable payloads are combined with cached data to create a cleaner version of the original packet.

Packet loss and corruption due to bit errors can be independently controlled by FEC applied separately to packet header (including ADU header) and ADU payload. To distinguish between a corrupted header and payload at the receiver, each segment requires a separate error detection mechanism, such as by a cyclic redundancy code (CRC). This is diagramed in Figure 7. Upon the first arrival of a packet, prior to any secondary transmissions, a CRC check is applied to the header, and the packet is discarded if in error. If the header is correct, error correction followed by error detection is

applied to the payload. If the payload is correct, the error-free ADU is forwarded to the application and all state associated with this sequence number is purged. A useful feature of the transport service is to inform the application whether the packet being delivered is corrupt, and certainly it should inform the application upon delivery of the last (most reliable) delivery.

If the ADU payload is corrupted, the ADU may be delivered to the application, if it meets QOS requirements. It is also cached for packet combining (see below).

4.2 Successive refinement via packet combining

A key property of the multi-version service model is that the packet versions are successively refined (improved in quality). Repetition coding, often used in ARQ protocols, does not have this property, and also wastes transport resources by discarding corrupted packets with useful information. Packet combining simply accumulates all the received information relevant to a given ADU, and makes use of that aggregate evidence in reconstructing the next estimate. This approach admits coding approaches more sophisticated than repetition coding, such as transmitting incremental parity check redundancy rather than retransmitting the raw ADU. The latest ADU estimate is guaranteed to be statistically higher in quality than previous estimates, at the cost of greater memory and processing requirements at the receiver.

Packet combining has been incorporated within single-delivery ARQ protocols, in a form often called *memory ARQ* [29] [30]. For example, *Type-II Hybrid ARQ* described in [8] alternates retransmissions between the original payload (say P_1) and a convolutional parity check (say P_2) for error correction. The receiver either caches a corrupted P_1 and then uses the next P_2 to correct the cached P_1, or caches a corrupted P_2 to correct the next P_1. This has been extended to *maximum-likelihood code combining* of multiple cached packets [31]. Both these techniques retransmit payload-length repetitions. An alternative is to retransmit redundancy incrementally using rate-compatible punctured convolutional (RCPC) codes and incrementally recombine received information [32]. Hence, individual retransmissions may not contain sufficient information to independently reconstruct the original message. An integration of [31] and [32] was proposed in [33]. Simpler (and less powerful) packet combining approaches use bit-by-bit combining, called *time diversity combining* [34]. For example, given repetition coding and Viterbi decoding, then a soft-decision diversity combiner would 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 this into the Viterbi decoder [35]. Repetition coding with majority-logic bit-by-bit decoding is an example of hard-decision diversity combining [36]. A comparison of the efficiency of diversity combining vs. code combining is found in [37] for the case of repetition coding with multiple copy decoding.

Leaky ARQ modifies “ARQ with packet combining” to forward intermediate renditions of the payload to the application. Since retransmissions may arrive much later than the initial header-intact version due to traffic shaping, then it is vital that these intermediate renditions be quickly delivered rather than be hidden. Moreover, there is no traffic penalty to forwarding these stored intermediate versions.

Packet combining at the receiver can take several forms. For example, the encoder can apply RCPC codes, thereby gaining both incrementally redundant transmission and maximum-likelihood decoding. The receiver can implement a hard-decision Viterbi decoder based on Hamming distance path metrics, or a majority-logic hard-decision diversity combining decoder. Since we are operating an end-to-end transport layer protocol, instead of a link-layer protocol, then it is unlikely that we will have access to soft symbol information; hence, any transport-level Viterbi decoder will operate on hard-decision information.

Packet combining can be performed either at the granularity of ADUs or segments (see Section 4.3). Figure 7 showed that the payload CRC check is performed after packet combining. If the payload CRC check is executed once per segment, clearly packet combining must be performed on a segment granularity. If the payload CRC check is executed once per ADU, then packet combining can occur either at the ADU or segment granularity; i.e. individual segments can be successively refined and delivered to the payload CRC check, or successive refinement can be done on an ADU basis prior to the payload CRC check.

Caching requirements for the receiver's packet combiner depend on the maximum number of outstanding segments that can be simultaneously unacknowledged, the depth with which retransmissions are stored, and the combining policy. Certain combining policies can minimize the storage needed per segment. The idea is to only store the most recent retransmission, or even better an accumulated best estimate, rather than all previous redundancy pertaining to this ADU. For example, Type-II Hybrid-ARQ [8] only stores the previous payload and combines it with the incoming payload. This sacrifices coding efficiency to achieve memory savings by ignoring the complete reception history. Time diversity combining accumulates energy per bit to achieve a single best estimate of the original payload. Full code combining requires caching of all received payloads.

4.3 Segmentation of ADU's

Because of application-level framing, leaky ARQ is essentially an acknowledged datagram protocol. These datagrams need not be the same size as the ADU, since segmentation and reassembly should be allowed. (In many respects, this message fragmentation is similar to the IP datagram fragmentation problem [5].) As long as individual segments converge toward a reliable representation, the ADU will similarly converge. Retransmission, acknowledgments, and code combining can all be done on the segment rather than ADU granularity. The various retransmission options are shown in Figure 8, where an additional distinction is made between retransmission with and without incremental redundancy. The initial transmission contains sufficient information to reconstruct the payload, since the initial payload may be clean of errors and not require retransmission at all. Subsequent retransmissions may only contain incremental parity check bits.

Internal to the protocol we define an *ADU segment*. Each segment's header will contain at least the ADU sequence number, segment number, and header CRC, in addition to the usual source and destination port ID's and segment length. In addition, the ADU correlation label can either accompany each segment, or propagate once per ADU, or not propagate at all with the ADU, depending on how carefully the transport layer is optimized to implicitly annihilate out-of-date data. For example, if the correlation label is allowed to propagate with the ADU, then retransmissions can be quenched not only at the source, but at any intermediate router which is caching and retransmitting transport-level packets, as

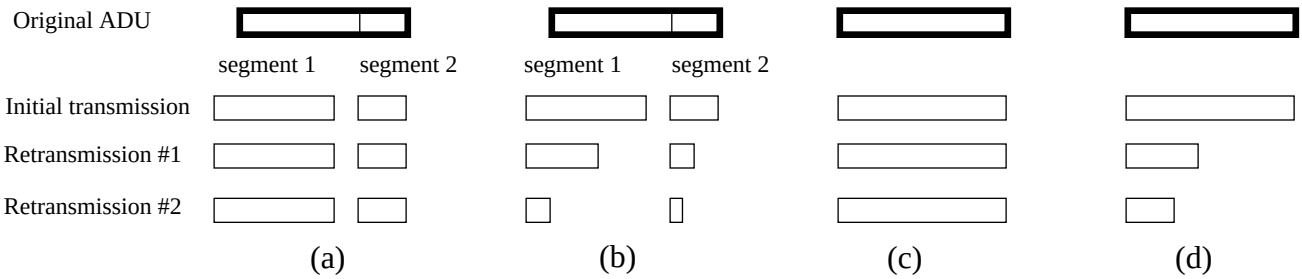

Figure 8. Retransmission granularity. (a) Segment retransmission without incremental redundancy (IR) (b) Segment retransmission with IR (c) ADU retransmission without IR (d) ADU retransmission with IR.

advocated in the snoop protocol TCP scheme [20]. When the protocol transmit side receives the annihilator, it stops retransmitting any other ADU with the same correlation label.

The payload CRC is also embedded within the segment header and allows the discard of segments with invalid headers. For asymptotically reliable delivery, an ADU payload CRC is needed to detect correct payloads. There can be a separate payload CRC for each segment, or more likely one payload CRC for the entire ADU.

Another field in the header indicates the overall ADU message length so that ADU framing boundaries can be determined. As for the payload CRC, the ADU length field can be distributed within each segment, or embedded only once per ADU. This length indicator can explicitly specify the total ADU length, or can follow the approach of IP datagrams, which use a "more-fragments bit" to define the boundary of a segmented ADU. However, the IP approach can create a complicated reconstruction problem if one or more segments are lost. The receiver will be unable to determine the size of the ADU for variably-sized segments, or for fixed-size segments when the final (variably-sized) segment is lost. This unnecessarily delays delivery of the initial corrupt version of the ADU.

Both the payload CRC and message length fields share several properties. First, both can be embedded once per ADU. For segmented ADU's, this means that the segment header will be variably-sized, since only one segment needs to contain the overall message length and/or payload CRC. In Figure 8(a), this would correspond to segment one carrying the per-ADU header information, while segment two carries an abbreviated header. Variable-sized headers can be supported by using an "options" field to extend the header when necessary (for comparison, IP datagrams contain an "IP Options" field).

The second property shared by these two fields is that neither needs to be carried within *re*-transmissions. Clearly both fields must be propagated until the first header-valid reception. However, after this event, overhead can be minimized by not including these fields in retransmitted headers. In order for the sender to know whether to retransmit a full header or a shortened header, the receiver must relay its state back to the sender. Since transmitter and receiver realize this ADU is now in a "clean header but dirty payload" state, there is no need for a field or bit in the header to signal that the header is abbreviated. Note that this second property leads to variable-sized headers across retransmissions, whereas the first property lead to variable-sized headers across segments. In the first case, we needed additional header fields to indicate abbreviated/expanded headers, while in the second case no additional header signalling is required.

4.4 Non-sliding windows

The sliding windows characteristic of most ARQ protocols can result in decreased performance for bursty windows-based multimedia.TCP's effective window size W, the number of outstanding bytes/words within the transport pipe, is a minimum of the flow control window (what the receiver can handle) and the congestion-control window (what the network can handle). In TCP, the sender is further restricted not just by the number W of outstanding unacknowledged bytes, but also by the sliding property of the sender's window. The sliding window constrains the sender to transmit no more than W bytes beyond the lowest unacknowledged byte. If W-1 of the previous words have been correctly received, but the oldest sequence number has not been acknowledged, then the window stays fixed on this sequence number until it has been acknowledged. Hence, available bandwidth is under-utilized even as additional data may be queued waiting for transmission. For leaky ARQ, a sliding window based on message numbers will unnecessarily delay a burst of data if the lowest ADU sequence number remains unacknowledged. To facilitate rapid delivery, the leaky ARQ window should merely specify the number of ADUs outstanding at any given time, and not additional ordering relationships among ADUs.

4.5 ARQ Acknowledgments

The granularity of acknowledgments directly depends on the granularity of the payload CRC check. If the payload CRC check is executed once per segment, Figure 7 shows that acknowledgments must be able to identify individual segments. If the payload CRC check is executed once per ADU, then acknowledgments can either identify only segments, only ADU's, or both segments and ADU's.

The relationship between the granularity of acknowledgments and the granularity of packet combining is complicated. For example, suppose segmented ADU's are supported and that the payload CRC check is done once per ADU. One option would be to acknowledge with ADU precision, while combining packets at the segment level. This results in an inefficiency similar to Go-Back-N since each acknowledgment may spur retransmission of cleanly reconstructed segments. Another option would be to combine packets at an ADU level, and acknowledge with segment precision. Ideally, this permits the receiver to direct retransmission of the most corrupt segments. However, this depends on the ability of the packet combiner to both correct and locate the spurious segments, which it may be unable to do. The most flexible arrangement would permit packet combining and acknowledgments with segment precision.

In order to support abbreviated retransmission headers and incremental redundancy, acknowledgments must contain tighter precision to distinguish between header-valid corrupt-ADU reconstruction and fully reliable ADU reconstruction. Once the transmitter has been informed that the first header-valid payload has been reconstructed with errors, then it can begin sending incrementally coded redundancy and/or shortened retransmission headers. Without this additional precision, the sender would not know when to send the smaller-sized packets; retransmissions would be limited to segment-sized granularity and retransmission headers would repeatedly carry possibly unnecessary fields.

4.6 Other issues

We have left many unresolved issues for future work. For example, what kind of feedback from the transport service to the application is useful and practical? Does QOS admission control, which pre-allocates network resources, preclude the need for dynamic feedback? In Figure 3, we included the most general case of feedback to the source application. Also, forwarding corrupt ADU's assumes that the underlying data-link level protocols do not implement fully reliable link-layer retransmission. There may be other assumptions built into the multiple-delivery transport service which constrain the underlying network transport. Further, how can the transport service be optimized to deal with concatenated backbone-and-wireless connections, or multicast connections? Also, we chose not to address the issue of crunched packet (transport, ADU, and/or segment) headers and their impact on protocol design. Finally, are the QOS parameters that the user sets at the transport service interface interpreted as hints by the transport protocol, or viewed as requirements?

5. CONCLUSIONS

We have presented a multiple-delivery transport service which is motivated by the desire to transmit windows-based multimedia over wireless access links with, simultaneously, low perceptual latency and asymptotic reliability. Multiple (possibly corrupt) versions of the original packet are delivered to the destination application, each with greater reliability than the last. A richer set of service primitives and implementation features is made available to more efficiently use wireless channel resources, including traffic shaping by scheduling retransmissions judiciously, and annihilation of stale data. For the specific case of a reliable multi-version transport service, we have described modifications to classical ARQ protocols needed to support forwarding of corrupt packet payloads, an approach we term leaky ARQ. These structural modifications include separate payload and header error detection CRC's, segmentation

leading to variably-sized headers, and packet combining which forwards intermediate corrupt reconstructions.

6. REFERENCES

[1] 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.

[2] E. Brewer, T. Burd, F. Burghardt, A. Burstein, “Design of wireless portable systems,” *COMPCON ‘95 Technologies for the Information Superhighway*, pp. 169-176, March 1995.

[3] D. Weissman, A. Levesque, R. Dean, “Interoperable Wireless Data,” *IEEE Communications Magazine*, vol. 31, no. 2, pp. 68-77, Feb. 1993.

[4] A. Baier, U. Fiebig, W. Granzow, W. Koch, P. Teder, J. Thielecke, “Design Study for a CDMA-Based Third-Generation Mobile Radio System,” *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 4, pp. 733-743, May 1994.

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

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

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

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

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

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

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

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

[13] D. Chase, L.J. Weng, “Multiple-Burst Correction Techniques for Slowly Fading Channels”, *IEEE Transactions on Information Theory*, vol. 22, no. 5, pp. 505-513, September 1976.

[14] J. Linnartz, A. Jong, R. Prasad, “Performance of Personal Communication Networks with Error Correction Coding in Microcellular Channels”, *1st International Conference on Universal Personal Communications*, pp. 308-313, 1992.

[15] P. Patterson, “Error Correction Performance of Block Codes in Binary, Burst Communication Systems”, *MILCOM 1983*, vol. 1, pp. 114-118, 1983.

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

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

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

[19] W. R. Stevens, *UNIX Network Programming*, Prentice-Hall, 1990.

[20] H. Balakrishnan, S. Seshan, E. Amir, R. Katz, “Improving TCP/IP Performance over Wireless Networks,” *Mobile Computing and Networking 95*, pp. 1-11.

[21] [5] D. Clark, D. Tennenhouse, “Architectural Considerations for a New Generation of Protocols,” *ACM SIGCOMM 1990*, pp. 200-208.

[22] P.Haskell and D.G.Messerschmitt, “In Favor Of An Enhanced Network Interface For Multimedia Services”, submitted to IEEE Multimedia Magazine.

[23] J.M. Reason, L.C. Yun, A.Y. Lao, D.G. Messerschmitt, ``Asynchronous Video: Coordinated Video Coding and Transport for Heterogeneous Networks with Wireless Access'', *Mobile Computing*, H. F. Korth and T. Imielinski, Ed., Kluwer Academic Press, Boston, MA., 1995.

[24] Y. Chang, C. Leung, "On Weldon's ARQ Strategy," *IEEE Transactions on Communications*, vol. 32, no. 3, pp. 297-300, March 1984.
[25] D. Towsley, "The Stutter Go Back-N ARQ Protocol," *IEEE Trans. on Communications*, vol. 27, no. 6, pp. 869-875, June 1979.
[26] M. A. Jolfaei, "Stutter XOR Strategies: A New Class of Multicopy ARQ Strategies," *1994 International Conference on Network Protocols*, pp.56-62, 1994.
[27] V. Rhee, J. Gibson, "Rate-constrained two-layer coding of H.261 video," *Twenty-Eighth Asilomar Conference on Signals, Systems and Computers*, vol. 1, pp. 514-518, Nov. 1994.
[28] D. Raychaudhuri, H. Sun, R. Girons, "ATM transport and cell-loss concealment techniques for MPEG video," *ICASSP 93*, vol. 1, pp. 117-120, April 1993.
[29] P. Sindhu, "Retransmission Error Control with Memory," *IEEE Trans. on Communications*, vol. 25, no. 5, pp. 473-479, May 1977
[30] J. Metzner, D. Chang, "Efficient Selective Repeat ARQ Strategies for Very Noise and Fluctuating Channels," *IEEE Trans. on Communications*, vol. 33, no. 5, pp. 409-416, May 1985.
[31] D. Chase, P. Muellers, J. Wolf, "Application of Code Combining to a Selective-Repeat ARQ Link," MILCOM 1985, vol. 1, pp. 247-252, Oct. 1985.
[32] J. Hagenauer, "Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications," *IEEE Trans. on Communications*, vol. 36, no. 4, pp. 389-400, April 1988.
[33] S. Kallel, D. Haccoun, "Generalized Type II Hybrid ARQ Scheme Using Punctured Convolutional Coding," *IEEE Trans. on Communications*, vol. 38, no. 11, pp. 1938-1946, Nov. 1990.
[34] S. Wicker, "Adaptive Rate Error Control Through the Use of Diversity Combining and Majority-Logic Decoding in a Hybrid-ARQ Protocol," *IEEE Trans. on Communications*, vol. 39, no. 3, pp. 380-385, March 1991.
[35] B. Harvey, S. Wicker, "Packet Combining Systems Based on the Viterbi Decoder," *MILCOM 1992*, vol. 2, pp. 757-762.
[36] 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.
[37] S. Kallel, C. Leung, "Efficient ARQ Schemes with Multiple Copy Decoding," *IEEE Trans. on Communications*, vol. 40, no. 3, pp. 642-650, March 1992.