

# Adaptive Error Control for Multimedia Data Transfers

Bert J. Dempsey  
W. Timothy Strayer  
Alfred C. Weaver

University of Virginia  
Charlottesville, Virginia  
*bjd7p@virginia.edu*  
*wts4x@virginia.edu*  
*acw@virginia.edu*

**Abstract:** In this paper we suggest the need for a new class of transport service to handle the requirements of digital continuous-media data transfers and other applications that may desire to trade off data completeness for latency considerations. Specifically we define a Partially Error-Controlled Connection service under which the user submits application- and endsystem-specific parameters to coordinate the protocol's use of data retransmissions for error recovery with latency concerns. We implement a prototype of this service as a lightweight implementation enhancement to an existing protocol, the Xpress Transfer Protocol. Measurements of our implementation suggest that the PECC approach can provide for a service that limits the frequency and duration of data loss, which a non-error-controlled service can not do, while avoiding the delay-insensitivity of traditional error control in a fully reliable connection.

## 1. Introduction

As the paradigm for computing moves from centralized compute-servers to distributed systems, the experience base for bulk-data and transactional protocols is expanding. Current networks enable large quantities of data to be shared using application programs that make the underlying data delivery service transparent to the users. Windowing software environments for workstations decouple servers from displays, using many short messages and fixed-size page transfers to coordinate a session between the server and the display. Yet there are certain applications that produce data in a continuous stream for which neither bulk data transfer nor a transactional service is appropriate.

Multimedia describes the emerging network service that unifies into one provider the ability to serve reliable data exchanges and transactional interactions, as well as audio and video transfers. The term *digital continuous-media communication* ([1]) refers to data transfer where

the source produces an arbitrarily long stream of data at a fixed rate such that the data is intended for playback at the receiving end. It is this aspect of playback that prevents continuous-media data transfers from using conventional network services.

### 1.1. Continuous-media Data Transmissions

There are several characteristics that distinguish continuous-media data transmissions from other types of data transfers. Data is produced at a fixed or nearly fixed rate at the data source. For voice and video, both inherently analog data sources, the signals must be converted into some digital sampling. These samples are created once per sampling interval. Since the receiving device must recreate as nearly as possible the analog signals produced at the source, the samples are "played back" in the same order and at the same rate they were produced. Thus continuous-media data transmissions must be sequenced and delivered within a time constraint. Failing to do either causes the playback to deviate from the original.

Interestingly, though, the playback of certain types of continuous-media data does not require complete integrity of the data stream. Voice, for instance, can tolerate corruption before the playback becomes noticeably degraded to the human ear. Since humans are trained to filter noise from speech, even poor playback can be sufficient for a voice service. Video, by virtue of being divided into distinct frames, can tolerate losses of parts of frames, or even whole frames, before the video data loses its contiguity. Thus, tolerance of data loss is also a characteristic of continuous-media data transmissions, although exactly how much and in what density data loss can be tolerated depends on the application and data compression algorithms.

Since the data is being produced at a fixed rate, the throughput requirements for continuous-media data transmission are easily derived. For instance, when using standard Pulse Code Modulation techniques, uncompressed voice requires bandwidth of either 56 or 64 kilobits per second to maintain an appropriate sampling of

the analog data. Video, on the other hand, may require several megabytes per second if every frame is sent at a screen refresh rate of 30 times a second. Compression algorithms reduce this throughput requirement, but the aggregate of several streams over a single network can saturate the bandwidth of the underlying physical network. Consequently, specialized hardware has historically been used to transfer continuous-media data, specifically public branch exchanges (PBXs) and radios for voice, and television signal transmissions over cable (CATV) and the airwaves for video. Computer networks are designed for digital data transfers; continuous-media transmissions over computer networks require a rethinking of traditional data transfer techniques.

## 1.2. Continuous-media Data Transfers over Computer Networks

Traditionally networking protocols have been designed to provide data transfer services biased toward fully reliable and in-sequence delivery. Complex error detection and correction algorithms using data integrity checks, control messages, timers, and other forms of protection and detection are well-understood. Unreliable service supports low-overhead data delivery that is neither acknowledged nor sequenced. Continuous-media data transmission, due to both its time-constrained and loss-tolerant characteristics, does not match well with the services provided by either reliable or unreliable data delivery. Since reliability is either all or nothing, a continuous-media application using a reliable data delivery service may be blocked trying to recover from lost data, to the detriment of the playback quality. If the service is unreliable, the playback rate may be maintained but excessive data loss and/or out-of-sequence data may cause the playback to become incomprehensible.

Next-generation transport layer protocols are starting to provide some solutions to the requirements of this type of data delivery. Error control in these protocols is more flexible, being decoupled from data sequencing and other features such as flow and rate control. In particular, the Xpress Transfer Protocol ([2]) offers a “no-error mode” service that disables error control while maintaining all of the sequencing and flow control of a traditional connection-oriented service. Because XTP is a *transfer layer* protocol, having subsumed network layer responsibilities along with transport layer services into a single protocol, rate control is available on an end-to-end basis with the active participation of the intermediate nodes in the path, rather than on a hop-by-hop basis. Therefore, using rate control procedures, the available bandwidth can more easily be divided among the parties using the channel—

quite useful for applications whose rate of data production is known and fixed.

The problem of providing an appropriate service to continuous-media data transmission still exists, even with new protocol features. The choice remains as to whether or not to use a reliable service. XTP offers a no-error mode service to provide sequencing but no error recovery; this service, however, can not ensure that a certain quality of service is maintained. Conversely, in XTP there is an option to have data loss reported immediately, a feature not present in traditional window-based error detection. This *fast negative acknowledgement* is useful when out-of-sequence data most probably indicates lost data rather than a delayed packet. Recovery time is minimized since action is taken at the earliest moment. Yet this does not guarantee that the error recovery mechanisms will not impede the progress of the playback.

The type of error recovery mechanism appropriate for continuous-media data transfer must attempt to recover from lost data when it is possible to do so without impeding the playback. When error recovery would impede the playback, the mechanism must abandon the lost data in favor of fresh data. Clearly such a mechanism must be parameterized to accommodate the wide range of acceptable data loss for the various continuous-media applications and of playback timing factors relative to end-system architectures. This suggests the need for a service with quality of service parameters like those for traditional transport protocols in order to guarantee either maintaining a specified level of service or signaling the service user that a failure has occurred.

## 1.3. Paper Overview

The rest of this paper is organized as follows. First, we propose a Partially Error-Controlled Connection service and outline a lightweight implementation providing it. The algorithm underlying this service is next described, followed by a discussion of how to use the service. In order to understand the user-chosen parameters better, experimental results are then presented and discussed. We conclude with a summary and observations on the directions for further research.

## 2. Partially Error-Controlled Connection Service

In this paper we observe that, while the no-error mode is a useful service for applications where recovery of lost data may impede the progress of the communication, this service provides only an on-off switch. We propose a new service, which we term a Partially Error-Controlled Connection (PECC) service, that allows the application to parameterize error-control policy within

---

	no data loss	controlled data loss	arbitrary data loss

---

the underlying protocol in order to achieve responsiveness to the application-specific trade-off between message loss, on the one hand, and improved throughput and timeliness of message delivery, on the other.

Figure 1 shows the ways that two aspects of an end-to-end protocol—*sequencing* of the data and reliable delivery, or *completeness*—are combined to meet the needs of various paradigms. When the application requires fully in-order sequencing and no data loss, the service provides a reliable connection. Without the aspect of sequencing this service becomes an acknowledged datagram, and if no efforts are made to ensure delivery, the service degenerates to datagram. XTP fills the fourth option, arbitrary data loss with in-order delivery, by offering the no-error mode. The middle column is added for continuous-media data transfers, where data loss is acceptable if it is controlled. The proposed PECC service sits between a reliable connection service and the service defined by XTP’s no-error mode connection by providing sequencing while allowing for a controlled amount of data loss.

A PECC service exploits the trade-off of data completeness in favor of latency concerns that are found in time-constrained data transfers. The concept of a PECC class of service addresses the needs, identified by a number of researchers ([3][4]) for an expansion of the traditional quality of service parameters in order to provide for continuous-media applications. Specifically, PECC introduces mechanisms for specifying packet error tolerance, error recovery strategy, and certain end-system delay properties that drive the selective use of retransmissions for recovering packets that were lost in the network. Such recovery takes place when it does not conflict with latency requirements while completeness is sacrificed when recovery is not useful or too time-consuming. Thus,

the PECC service provides a time-constrained application with error control that avoids undesirable blocking if the amount and frequency of data loss in the network is at a level tolerable to the application. If the service guarantee is violated during a data transfer, the PECC service provides an indication to the user.

### 2.1. Implementation Strategy

The sender-driven architecture of the XTP protocol, and in particular its use of Automatic Repeat Request (ARQ) error correction, offers a convenient architecture for a low-overhead implementation of a controlled reliability service. Specifically, the XTP receiving endpoint can use more sophisticated error-reporting algorithms based on application-specific parameters, and the XTP sending endpoint need not be aware of this modification. If, based on the user-submitted parameters for error tolerance, gaps in the data stream are ignored, the XTP receiving endpoint reports this data as correctly received and the XTP sending endpoint will advance its state accordingly. There need be no PECC-specific code on the sending side.

This strategy of modified error-reporting algorithms at the data sink is used for implementing the PECC service described in this paper. In this way the PECC service is provided as a lightweight enhancement to an existing end-to-end protocol. It is lightweight in two ways. First, it does not require any change to the XTP protocol definition. The PECC mechanisms can be introduced selectively at individual XTP nodes such that full interoperability with other end-systems is preserved. There is even the capability for certain receiving endpoints in an XTP multicast connection to use the PECC mechanisms at the same time that other multicast receivers are not.

Second, the PECC code does not interfere with normal packet processing when a PECC-based transfer is error-free, nor does it add any processing burden to other data transfer services. The modifications made to a software implementation of XTP in which the PECC algorithm is embedded include an interface call to invoke the PECC behavior and a small amount of additional code inside the module for processing packets received from the network. Once a receiving XTP context is opened, the user invokes the PECC option with an interface call on the receiving side of the transfer. All other XTP interface routines are unchanged.

## 2.2. PECC Algorithm

The PECC algorithm is invoked at the receiving XTP endpoint whenever a packet received from the network is being processed and that packet contains data that is out-of-order. The algorithm is guided by four parameters in the PECC interface call: `fifo_ok`, `window_length`, `window_density` and `max_gap`. Since the PECC algorithm is intended for a continuous-media distribution service, the XTP receiver logically places the data received from the network into a FIFO buffer that is emptied by an isochronous reader. `fifo_ok` indicates the amount of data in bytes that must be available to the reader before the receiving XTP will request a retransmission of lost data. Since the reader empties its FIFO at a fixed rate, the depth of the FIFO feeding the reader translates directly into an amount of time after which the reader will have consumed the data currently in its FIFO. The `fifo_ok` variable represents the PECC service user's estimate of the amount of time needed for lost packets to be recovered by retransmission from the source without the reader's FIFO underflowing in the mean time.

`Window_length` and `window_density` are parameters concerning the tolerance for the frequency and duration of errors during the transfer. The service guarantee to the user is that no more than `window_density` bytes of data will be *skipped* in any interval of `window_length` data bytes. Skipping data refers to the XTP receiver advancing its state as though a group of missing data had been correctly received. The receiver's new state is eventually reported through normal protocol procedures to the transmitter, and the transmitter's state will advance accordingly. For the purposes of the PECC service the skipped data is regarded as having been placed in the receiving FIFO, though the exact actions upon skipping data are implementation-dependent.

Finally, `max_gap` represents the maximum number of bytes that will be skipped in any one pass through the

PECC algorithm. `max_gap` controls the rate, relative to new data packets arriving, at which data in the receiver's buffers is skipped.

The PECC service reports failure when the service guarantee on the spacing and size of skipped data must be violated in order for the data transfer to continue in a timely fashion. When a failure occurs, the PECC implementation provides the user with an indication of failure, skips the missing data, and continues the data transfer.

A schematic of the PECC algorithm is shown in Figure 2. The first decision point compares the current depth of the reader's FIFO to the value of `fifo_ok` in order to determine if there exists enough data in the reader's FIFO such that retransmission of the current "gap" in the data can be carried out in a timely manner. This gap in the data is defined to be the group of data between the highest sequence number<sup>1</sup> for contiguously received data and the next span of valid data in the receiver's buffers, which may be the packet currently being processed.

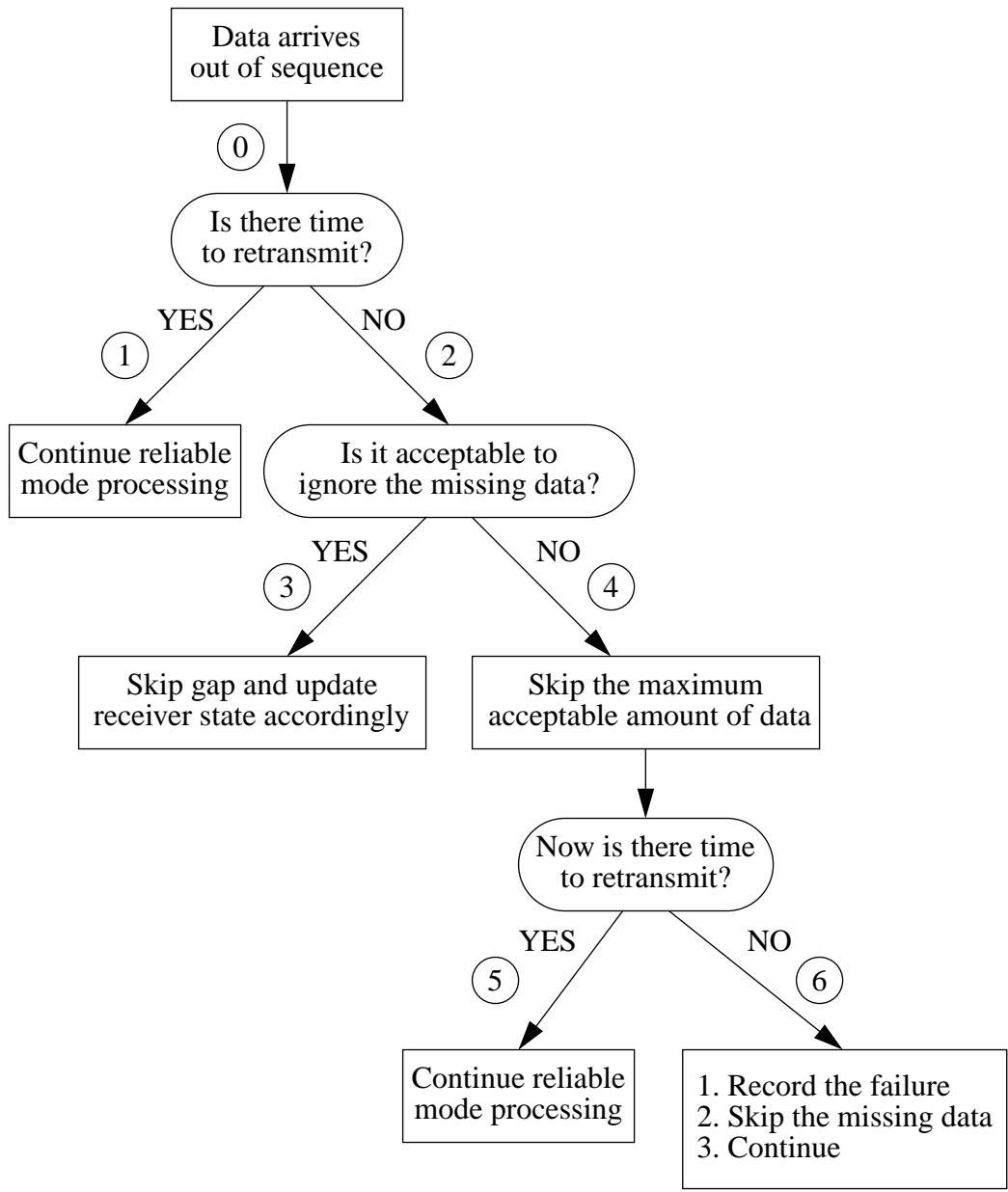
If the FIFO's depth is greater than `fifo_ok`, Arc 1 is followed, meaning that packet processing continues as it would in a reliable transfer. This processing includes the reporting of the gap in the receiver's buffers when the next control packet is sent from the receiver to the transmitter. Otherwise, Arc 2 is taken, and the second test in the PECC algorithm determines if the current gap in the receiver's buffers can be skipped.

The second test involves comparing the gap's size to the minimum of two values: (1) the value of `max_gap` and (2) the maximum amount of data that can be skipped in the current window. The latter is determined by the value of `window_length` and the amount of data that has been recently skipped. If the gap is smaller, then it is skipped (Arc 3 is taken), and packet processing continues under that assumption. Under XTP no-error mode transfers, all data up to the packet being currently processed is skipped whenever a packet with out-of-sequence data arrives.

If the gap in the data can not be skipped, Arc 4 is followed and the maximum acceptable amount of data, as determined by the value calculated in the second test, is skipped. The depth of the FIFO is now tested again to see if the addition of the skipped data has increased the FIFO's depth enough that there is now time for the additional lost data to be retransmitted. If so, Arc 5 is traced and packet processing continues with the new state. The gap has been partially skipped and the receiving FIFO artificially lengthened. As protocol processing continues,

---

<sup>1</sup>Sequence number



a request for the remaining part of the gap will be sent, when appropriate under XTP's procedures, to the transmitter.

If skipping the maximum acceptable amount of data at this point did not buy enough time to recover the remainder of the current gap, Arc 6 is followed. Since there is not time to recover the data, the rest of the gap is skipped, and a variable is used to record the fact that the PECC service guarantee on the size and frequency of gaps in the data stream has been violated. The data trans-

fer then continues on the optimistic assumption that the application may be able to tolerate this service lapse.

### 2.3. Using the PECC Algorithm

Determining how to set the PECC parameters for a certain behavior may be difficult since the PECC service is inherently concerned with real-time properties of the data transmission and packet loss in the network. Given a specific network environment and knowledge of the characteristics of a particular continuous-media stream, values for the PECC parameters can be determined, though

tuning based on a small number of test transfers may be helpful.

Error tolerance tends to be somewhat rigidly determined by the application, though subjective parameters such as the amount of data necessary for an acceptable screen refresh during motion video are sometimes a factor. The amount of time necessary to perform retransmissions (`fifo_ok`), on the other hand, is clearly an internal characteristic of the network and in general will not be known to the user. However, a rough bound should suffice since in most cases the granularity of the time necessary to perform retransmissions is fine relative to other time-constrained factors.

Consider as an example a voice transmission where the reader must be fed at a rate of 64 Kbits/s. In experiments conducted recently on the feasibility of using XTP running over FDDI to carry voice channels ([5]), it was observed that 512 or 1024 bytes per packet represented the most attractive packet size for carrying 8000 Hz voice samples. These figures derive from considerations regarding latency and the bandwidth necessary for a reasonable number of channels to be supported. In a local area network, if 10 ms is the (conservative) estimated time for the receiver to obtain a data retransmission, then the `fifo_ok` variable need specify only 80 bytes, a fraction of the data carried in each packet, as the required amount of data to be present in the reader's FIFO before data retransmission will be timely. Thus, in this type of LAN environment, conservative estimates for the `fifo_ok` variable are possible, affording the user confidence that the PECC service will indeed only use retransmission when it will not impede timely data delivery.

### 3. Experiments

The software XTP implementation in which the PECC algorithm is embedded runs as a user process on Sun-4 workstations with the User Datagram Protocol as its underlying network service provider. This version of the UVA XTP code uses the lightweight process library provided in the SunOs 4.1.1 distribution to handle the shared-memory communication between the logical layers of the UVA XTP architecture ([6]). Obviously, this XTP implementation environment is most useful as a tool for studying protocol workings, which is our purpose, rather than for actual continuous-media transfers. The experiments in this environment are not intended to represent performance data but to serve as a proof of concept for the PECC service and to provide insight by quantifying the interactions between PECC parameters in a particular network environment.

The experiment involves the timed transfer of 1000 buffers of user data. The buffers are transmitted using a non-blocking reliable SEND primitive defined in the UVA XTP interface. The last buffer is sent with a blocking reliable SEND primitive, after which the timer is stopped. The buffers are all transmitted across a single XTP connection, which operates under reliable, no-error, or PECC modes as determined by the configuration of options chosen. The reliable and no-error transfers use only XTP-defined mechanisms. For the PECC transfer, on the sending side the XTP connection is opened for reliable transfer while on the receiving side the PECC interface call activates the new error reporting algorithm.

Errors are injected into the data stream by having the transmitting side drop certain packets. The time between successive error bursts is determined by an exponentially distributed random variable as is the number of packets suppressed in each error burst. Burst durations are limited to 9 consecutive packets in order to keep overall data loss in the experiment to a small percentage of total data transferred. The net effect is to produce packet losses in the range of 5% of the total data transferred. While unrealistically high for a stable network, this error rate ensures that the PECC code is heavily exercised.

Our hypothesis is that a PECC service can provide an intermediate service between fully reliable and the XTP no-error mode connection. The first observation is that the PECC algorithm can mimic each of these bounding services. If the `fifo_ok` variable is set to 0, then there is always time enough to recover lost data since there is always at least 0 bytes in the reader's FIFO. In this case the algorithm in Figure 2 follows Arc 1 on every invocation. If the `fifo_ok` and `max_gap` variables are given very large values, then the equivalent of a no-error mode service is the result; the PECC algorithm always decides there is not time to retransmit (and hence traces Arc 2) and then always skips the missing data in full (traces Arc 3).

Table 1 shows the results of 1000-buffer transfers under various PECC configurations. The numbers in the first column express the transfer scenario in terms of the values of the four PECC parameters, respectively, `fifo_ok`, `window_length`, `window_density`, and `max_gap`. For convenience, these parameters are expressed in Table 1 in terms of application buffers, not bytes. The application buffers in Table 1 are 4 bytes. The transfer syntax used in the experiment forces out each application buffer onto the network, and, since the buffers are small enough to fit into a single XTP-level packet, an application buffer corresponds to an XTP packet crossing the network. A 1-10-3-3 configuration denotes that (a)

Scenario	Total transfer time (sec)	Total buffers skipped	Number packets dropped	Maximum buffers lost in a window	Failed runs (out of 15)
0-0-0-0	43.7 [41-49]	0 —	69.7 [41-100]	— —	0
1000-0-0-1000	38.0 [37-40]	66.2 [44-100]	66.2 [44-100]	— —	0
1-0-0-1	37.8 [37-39]	16.8 [14-20]	56.4 [39-69]	— —	0
3-0-0-3	38.3 [37-40]	40.3 [29-48]	59.8 [39-76]	— —	0
1-10-3-3	37.8 [37-39]	43.0 [32-58]	64.0 [40-87]	3.4 [3-8]	2
3-10-3-3	37.8 [37-41]	42.2 [37-53]	62.3 [49-81]	3.9 [3-8]	3
3-30-3-3	37.6 [37-38]	45.9 [28-59]	60.2 [38-79]	8.0 [3-11]	14
8-100-10-8	38.2 [37-41]	66.4 [48-89]	67.3 [48-90]	15.9 [12-22]	15

`fifo_ok` is set to the number of bytes in a single buffer, (b) no more than 3 buffers in every 10 will be allowed to be skipped, and (c) no gap of more than 3 buffers will be skipped in one iteration of the PECC algorithm. The parameter settings also ensure that no gap of more than 6 contiguous buffers will be skipped since the worst case would be a 6-buffer gap that fell on a window boundary.

The next four columns contain the average value over 15 1000-buffer transfers displayed above the minimum and maximum values in any individual transfer. The columns contain, left to right, the measured transfer time, the total number of buffers that were missing and for which no transmission was requested by the receiver (i.e., skipped), the total number of XTP packets lost in the underlying network, and the maximum number of buffers skipped in any one window of length `window_length` during a 1000-buffer transfer. The rightmost column identifies the number of runs during which the PECC service specified could not be met. A failed transfer may have experienced more than one time at which the service guarantee was violated.

The first row of Table 1 shows a PECC configuration that ensures fully reliable delivery. Since all dropped data

is being recovered through retransmissions, the total number of packets sent is slightly higher in these transfers, which results in the total packets dropped being slightly higher than the other PECC configurations. As expected, the average transfer time is longer than for the other PECC configurations, in this case about 6 seconds. This longer transfer time implies that the reader was blocked waiting on retransmissions at some points during the transfer.

The second row of Table 1 shows a PECC configuration that effectively provides a no-error mode (unreliable) service. For the no-error mode case, the amount of data lost is exactly the number of bytes contained in the buffers that are (artificially) lost in the network—that is, the receiver skips all gaps in the data stream. No-error mode transfers finish more quickly than reliable transfers, and all the other PECC configurations take essentially the same amount of time as the no-error configuration. This is not by accident since the PECC algorithm is specifically designed to suppress requests for data retransmission when such a request will cause the reader’s FIFO to underflow. Thus, in a sense, the PECC configurations only perform those data retransmissions that are free with

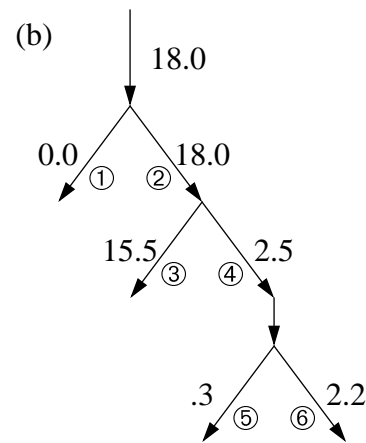
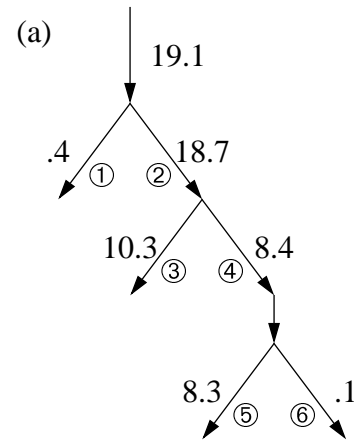
respect to total transfer time. Although we have not attempted here to measure interpacket delay, or jitter, the PECC algorithm's data recovery policies should also have the effect of controlling jitter as well as total transfer time.

Configurations such as 1-0-0-1 and 3-0-0-3 in which  $fifo\_ok$  is less than or equal to  $max\_gap$  and no window criteria is used provide a *threshold service*. All losses of less than  $max\_gap$  buffers are skipped immediately (Arc 3 is traced) while all larger gaps will have their first  $max\_gap$  buffers skipped and a possibility of retransmission of the remaining missing data. The latter is true since, after tracing Arc 4,  $max\_gap$  buffers will be skipped and there will then be at least  $fifo\_ok$  buffers in the FIFO (assuming no race condition with the FIFO emptying after the skip). Arc 5 is then followed. Thus, as seen in Table 1, the threshold service guarantee can always be enforced, and all runs were successfully completed.

These threshold services do not guarantee, however, that only  $max\_gap$  size gaps in the data will be skipped since each out-of-order packet may cause the first  $max\_gap$  buffers in a large gap to be skipped. Nonetheless, by thresholding at 1 buffer, the PECC configuration of 1-0-0-1 reduces buffer lost to an average of less than 1.7% of the total data transferred where a no-error mode connection for the same transfers would have lost, based on actual measurement of the number of dropped packets, 5.6% of the data. Thresholding at three contiguous buffers per skip raises the average skipped data to 4% where 5.9% is being dropped by the network.

The last four configurations in Table 1 introduce a window criteria to control the frequency and duration of errors. This also introduces the possibility of the PECC service reporting failure. Since the size of  $max\_gap$  equals or exceeds that of  $fifo\_ok$  in these configurations, the PECC algorithm reports failure only if the window density criteria reduces the number of buffers that can be skipped such that the receiver is unable to skip data that can not be recovered from the transmitter in a timely fashion.

The behavior of the PECC algorithm under configuration 1-10-3-3 is shown in Figure 3(a). The tree branches in this figure correlate to the diagram in Figure 2, and they are labeled with the average number of times they were followed during a single 1000-buffer transfer. On average for the runs using 1-10-3-3, the PECC algorithm was entered 19.1 times. Arc 1 was almost never taken, indicating that the reader in our test environment empties its logical FIFO fast enough that the FIFO is almost always empty. This is also evident from the almost iden-



tical performance of configurations 1-10-3-3 and 3-10-3-3.

At the second decision point in 3(a), the gap can often be skipped (Arc 3 is followed). When the gap is too large to be skipped, it is usually the case that at least 1 buffer's worth of data can be skipped, which is enough (since  $fifo\_ok$  is 1) to allow for a resumption of reliable mode packet processing. However, if 3 buffers have already been skipped in the 10-buffer window, it will be the case that no extra buffers can be skipped with impunity at this point. Arc 6 is then followed and a PECC service failure recorded.

As the size of the window is increased, the likelihood of long and/or multiple gaps occurring in a single window increases and the PECC service reports failure more often. Figure 3(b) shows the PECC behavior for the 8-100-10-8 configuration. Since  $fifo\_ok$  is 8 and our reader is very fast, it never happens that there is enough data in the FIFO to allow Arc 1 to be followed. At the second decision point, most gaps can be skipped since

Scenario					(out of 10)
0-0-0-0	53.5 [51-57]	0 —	77.6 [54-94]	— —	0
1000-0-0-1000	46.9 [46-48]	65.3 [59-78]	65.3 [59-78]	— —	0
.5-0-0-1	48.1 [47-50]	21.9 [16-34]	65.5 [49-75]	— —	0
.5-0-0-3	48.5 [46-51]	40.7 [32-48]	63.2 [39-80]	— —	0
.5-0-0-1000	47.5 [46-50]	57.4 [44-69]	65.8 [49-79]	— —	0
.25-10-3-3	47.5 [46-49]	42.7 [32-50]	71.2 [49-88]	3.4 [4-7]	1

`max_gap` is 8, and our error injection scheme limits the number of consecutively dropped packets to 9. Indeed, the 8-100-10-8 configuration recovers very few errors, approaching a no-error mode service in the amount of data skipped. Because the window of 100 buffers is large relative to the occurrence of errors in the data stream, failure at the last decision point in the PECC algorithm is relatively frequent, and in each of the 15 test runs, failure occurs at least once.

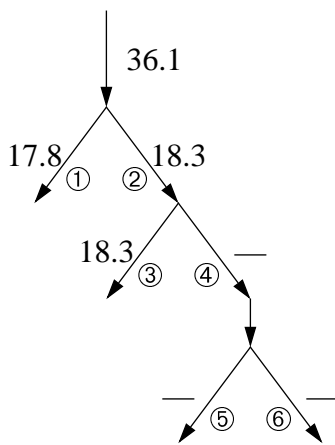
For Table 2 the basic experiment is altered in the following ways. The application buffers are now 4096 bytes; on the receiving side, the reader reads only 1024 bytes at a time and is forced to perform additional processing on each buffer in order to further slow the rate at which it empties its FIFO.

As with Table 1, the PECC configuration for reliable transfer shown in row 1 of Table 2 takes longer than the other PECC configurations. The other PECC instances are close to the no-error mode configuration (second row) in total transfer time. The burden on the reliable service in terms of total packets dropped is also noticeable since the reliable service requires more retransmissions, and at the XTP level two buffers no longer fit into a single packet.

The intent behind slowing down the reader and forcing it to read each transmitter buffer as 4 independent messages was to see the effect of the `fifo_ok` variable in the PECC algorithm. The configuration 0.5-0-0-1000

quantifies the effect. In this configuration, the only way that packets lost in the network can be recovered is if the PECC algorithm takes Arc 1, that is, the FIFO buffer at the receiver is greater than `fifo_ok` (2048 bytes). Otherwise, the unconstrained value for `max_gap` will result in all the data in the gap being skipped. Figure 4 shows the measured behavior of the algorithm under the 0.5-0-0-1000 configuration. While the algorithm went out Arc 1 roughly half of the time, Table 2 indicates that only about 10% of the buffers lost are recovered. These figures suggest that, under this configuration, many data retransmissions that were permissible when one out-of-sequence packet is being processed are effectively negated by ensuing data packets whose processing causes the data gap to be skipped.

The three configurations of the form 0.5-0-0- $x$  indicate the relative effects of the `max_gap` variable. When `max_gap` is 1, the PECC service recovers two-thirds of the data lost in the network, slightly worse on a percentage basis than the 1-0-0-1 configuration in Table 1. When `max_gap` is 3, the data recovered through retransmissions is down to one-third of all buffers lost by the network and, in the limit of an unconstrained `max_gap` value, this figure is about one-tenth. Thus, the `max_gap` parameter plays an important role in limiting the rate at which data is skipped, and, at least under the high error rates of our test environment, small changes in `max_gap` affect the service greatly.



Finally, the 0.25-10-3-3 configuration loses roughly 7% fewer buffers than the counterpart configuration 1-10-3-3 in Table 1, normalized to the experienced error rates. This figure correlates with the evidence from data on configuration 0.5-0-0-1000 that, under the conditions for Table 2, a small value for the `fifo_okv` variable allows for some data recovery that would otherwise be skipped if the reader's FIFO is always empty, which was the case for the runs in Table 1.

#### 4. Conclusions

In this paper we have addressed the need for a new class of transport service to handle the requirements of digital continuous-media data transfers and other applications that may desire to trade off data completeness for latency considerations. We have defined a Partially Error-Controlled Connection service and demonstrated that this service can be provided using a lightweight implementation enhancement to an existing protocol, the Xpress Transfer Protocol. Measurements of our implementation suggest that the PECC approach can provide a service responsive to end-system playback characteristics such that error control recovers some amount of data lost in the network and yet avoids the blocking that can occur with traditional fully reliable data delivery.

For applications that do not require 100% data completeness, selectively ignoring noncritical data loss can improve the throughput of a transfer while allowing the advantages of a connection-based service to be retained. Throughput increases may be especially significant in the case of multicast data transfers. In particular, XTP defines a multicast service that uses go-back-N retransmission at the transmitter. Since an isolated packet loss at one receiver normally results in a retransmission of that packet for the entire multicast group, the more sophisti-

cated error reporting in a PECC service can be expected to have a significant impact on the throughput performance of an XTP multicast in which some or all of the multicast receivers operate under the PECC rules. We intend to pursue this idea our XTP multicast implementation.

We also intend to migrate our PECC implementation to an FDDI-based platform where our XTP research efforts have already had some success with continuous-media transfers ([5]). Here we can explore variations and refinements to the current PECC algorithm—such as adding an error history component to the `max_gap` parameter and addressing FIFO overflow—within the context of actual continuous-media transfers.

In summary we believe that a mechanism by which error-recovery in the end-to-end protocol can be sensitive to application- and architecture-specific parameters will be an important component in any computer network system for carrying digital continuous-media streams.

#### 5. References

- [1] D. Ferrari, "Client Requirements for Real-Time Communication Services RFC 1193, November 1990.
- [2] *Xpress Transfer Protocol Definition: Revision 3.6*, Protocol Engines, Inc., Santa Barbara, California, December 1991.
- [3] B. Field and T. Znati, "Experimental Evaluation of Transport Layer Protocols for Real-Time Applications," *Proceedings of the 16th Conference on Local Computer Networks*, Minneapolis, Minnesota, October 14-17, 1991.
- [4] D. B. Hehmann, M. G. Salmony, and H. J. Stuttgen, "High-Speed Transport Systems for Multi-Media Applications," *IFIP WG 6.1/WG 6.4 International Workshop on Protocols for High-Speed Networks* H. Rudin and R. Williamson (Editors), Elsevier Science Publishers B.V. (North-Holland), Zurich, May 1989.
- [5] A. C. Weaver, J. McNabb, and J. Fenton, "A Feasibility Study of Digitized Voice Distribution via the Xpress Transfer Protocol," Technical Report, Department of Computer Science, University of Virginia, December 1991.
- [6] R. Simoncic, A. C. Weaver, and M. A. Colvin, "Experience with the Xpress Transfer Protocol," *Proceedings of the 15th Conference on Local Computer Networks*, Minneapolis, Minnesota, September 30-October 3, 1990.