Report Number: WUCS-94-27

1994-01-01

An Application-Oriented Error Control Scheme for High Speed Networks

Fengmin Gong and Gurudatta Parulkar

Many new network applications demand interprocess communication (IPC) services that are not supported by existing transport protocol mechanisms. Large bandwidth-delay products of high-speed networks also render the existing control mechanisms such as flow and error control less efficient. In particular, new error control schemes that can provide variable degrees of error recovery according to the applications requirements are needed. This paper presents the design, evaluation, and implementation of an application-oriented error control scheme that is aimed at supporting efficient IPC in high-speed networking environments. Our results show that the proposed error control scheme allows effective control of trade-off between... **Read complete abstract on page 2.**

An Application-Oriented Error Control Scheme for High Speed Networks

Fengmin Gong and Gurudatta Parulkar

Complete Abstract:

Many new network applications demand interprocess communication (IPC) services that are not supported by existing transport protocol mechanisms. Large bandwidth-delay products of high-speed networks also render the existing control mechanisms such as flow and error control less efficient. In particular, new error control schemes that can provide variable degrees of error recovery according to the applications requirements are needed. This paper presents the design, evaluation, and implementation of an application-oriented error control scheme that is aimed at supporting efficient IPC in high-speed networking environments. Our results show that the proposed error control scheme allows effective control of trade-off between the amount of error an application can tolerate and the amount of delay it suffers.

An Application-Oriented Error Control Scheme for High Speed Networks

Fengmin Gong and Gurudatta Parulkar

WUCS-94-27

October 1994

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

To appear in the ACM/IEEE Transactions on Networking.

To appear in ACM/IEEE Transactions on Networking

An Application-Oriented Error Control Scheme For High Speed Networks[†]

Fengmin Gong
MCNC
gong@concert.net
Phone 919-248-9214

Gurudatta Parulkar
Washington University
guru@flora.wustl.edu
Phone 314-935-4621

Abstract

Many new network applications demand interprocess communication (IPC) services that are not supported by existing transport protocol mechanisms. Large bandwidth-delay products of high-speed networks also render the existing control mechanisms such as flow and error control less efficient. In particular, new error control schemes that can provide variable degrees of error recovery according to the applications requirements are needed. This paper presents the design, evaluation, and implementation of an application-oriented error control scheme that is aimed at supporting efficient IPC in high-speed networking environments. Our results show that the proposed error control scheme allows effective control of trade-off between the amount of error an application can tolerate and the amount of delay it suffers.

1. INTRODUCTION

The new generation of high-speed networks will interconnect machines with very high bandwidth over long distances. There are two important characteristics for these networks: large bandwidth-delay product and non-negligible error conditions (losses in particular). The high data rates and long propagation delays result in very large bandwidth-delay products. This large bandwidth-delay product has a number of implications on error control strategy: (1) Time for end-to-end control actions (minimum of one round

[†]This work has been done at Washington University and it was supported in part by the National Science Foundation, and an industrial consortium of Ascom Timeplex, Bellcore, BNR, DEC, Italtel SIT, NEC, NTT, and SynOptics.

trip delay) becomes very significant with respect to the high data rate and therefore, frequent end-to-end control actions should be avoided. (2) It becomes very difficult to achieve efficient error control using only timer-based loss detection, because high data rates make it more difficult to set timer accurately and also make it costly to have an inaccurate timer. (3) It becomes very expensive to recover every packets in error through retransmission. Thus, we still need error control function, and the error control scheme should provide efficient error recovery in the face of large bandwidth-delay product.

At the same time, new applications such as network computing and visualization require high bandwidth and low latency communication with performance guarantees. They also deal with different types of data streams (e.g., voice and video streams, image sequence, and numerical data set) which have very different error tolerances. It is thus highly desirable to have a flexible service interface provided by the transport level that allows the application to accurately specify its error tolerance and thus avoid unnecessary long recovery delay.

There are four basic types of errors in network communication: out-of-sequence delivery, packet duplication, packet corruption, and packet loss. For error recovery, we will treat corrupted packets as if they were lost packets. Error control involves two basic steps: the detection of errors and the recovery from these errors. Of all the errors, packet loss is the most difficult to deal with and will be focused on in this paper.

Doeringer and his colleagues have done an extensive review of existing transport protocols, including their error control schemes [8]. Existing error control schemes have not addressed these issues efficiently. Here, we only point out the key error control features in these protocols. Standard implementation of TCP (transmission control protocol) [10] uses a cumulative ACK scheme with a loss timer at the sender side. VMTP [3] and SNR [17] have used selective ACK schemes with loss timers at the sender side, while NETBLT used a receiver side timer [6]. SNR also uses the idea of periodic state information exchange to minimize delay in error recovery. Delta-t [22] protocol used timers to implement an innovative connection management scheme that supports the so-called light-weight (or soft-state) connections, i.e., a connection is established upon the arrival of the first protocol data unit and will be torn down at the expiration of a timer after data transfer. A gap-based loss detection method was first used in a link level protocol CPM [1]. TP++ [2] supports a forward error correction (FEC) option, which forms a contrast with the other schemes that are purely retransmission-based. XTP provides the most extensive options (e.g., cumulative ACK and selective ACK) but calls for a very complex protocol engine for implementation [4, 5]. Due to their sole dependence on timers, these transport level error control schemes will not be very efficient in network environments with large bandwidth-delay product. Furthermore, none of the existing protocols support variable-degree error recovery according to the application error tolerance.

We have developed a new application-oriented error control scheme to address the need of efficient

error control in large bandwidth-delay product environments. The scheme is part of a Segment Streaming Transport Protocol (SSTP[1]). SSTP has been evaluated using analysis and simulation, and it has been implemented in software inside SunOS 4.0.3 kernel. This paper presents the design and main results for the error control scheme. A more detailed report on SSTP is available in [12].

The rest of the paper is organized as follows. Section 2 discusses how the proposed scheme addresses the key error control issues. Section 3 describes in detail the packet formats used as well as the operations of the error control scheme. Section 4 describes the performance studies conducted for SSTP and presents important results. Section 6 presents a summary.

2. PROPOSED ERROR CONTROL SOLUTION

Our objective for error control is to satisfy the applications' error tolerance with minimum retransmission overhead, so as to achieve high throughput and low end-to-end delay for a given network connection. We assume a connection-oriented network with resource reservation, which means better-controlled loss and out-of-sequence delivery characteristics.

There are three major issues concerning the error control: characterization of application error tolerance, detection of errors, and recovery from errors. Each of these issues is discussed next, along with a description of how it is addressed in SSTP.

2.1. Characterization Of Application Error Tolerance

In order to characterize the application error tolerance, we need to understand how an application uses data. In SSTP, a *segment* is defined as the smallest unit of data that an application accesses independently. Thus, a segment is the smallest meaningful unit of data for an application. A segment here should not be confused with a segment in TCP, where a segment is just a number of contiguous bytes and may not have any logical significance. We also assume that a sender will try to transmit a data segment to the receiver as soon as the segment is available for transmission and the receiver is willing to accept it. This mode of communication is called *streaming*. The segments are accepted in sequence[2].

The characterization of application's error tolerance should be powerful enough to describe detailed distribution of tolerable errors within a data segment. It should also be expressive enough to describe a wide range of error tolerances, from tolerating no error to tolerating all errors. Preferably, it should

[1] SSTP is not a complete transport protocol in that we have not specified packet formats or operations for transport connection setup and teardown. Our concentration has been on the design and performance study of the control mechanisms that can support efficient transfer of large data segments for imaging applications.

[2] We recognize that there are applications which do not necessarily require in-sequence delivery of segments. However, there are many that do and these applications are the main target for SSTP.

be simple and as much as possible compatible with the quality of service (QOS) characterization of the network, in order for efficient implementation.

Two traditional measures for the reliability of communication channels are bit-error-rate and packet-loss-rate. They have also been used in describing the reliability requirement of applications. These measures only specify a long-term average of bit error or packet loss rate which is hard to monitor and enforce on a per application basis.

In SSTP, we use a triplet specification (N, E, B) for characterization of application error tolerance, where N is the number of packets based on which the loss should be assessed and tolerance checked[3]; E specifies the maximum number of packets that is permitted to be lost among N; and B specifies the maximum length of an acceptable loss burst. When such a tolerance is specified by an application, the transport protocol will guarantee that the tolerance is satisfied through necessary retransmissions.

The most important advantage of the triplet specification is that it describes the loss with respect to a meaningful data unit of the application. Therefore, an application will be able to evaluate the impact of loss in a more precise context and make rational decisions on how to set its tolerance. We show by an example how the triplet specification can be used by applications. For applications such as computational fluid dynamics (CFD) simulation and visualization, the data set is often organized as a series of 2D slices (or arrays). Figure 1 shows an example where each slice is 1024 x 1024 bytes. Each slice is an independent data unit for the application and is defined as a segment. The segment is then divided into 1024 packets for transmission.

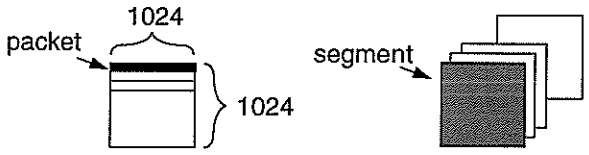

Figure 1: Application Data Set Structure

If each slice is an animation image, loss of 5 isolated scan lines may be perfectly acceptable. So a tolerance specification of $(1024, 5, 1)$ will ensure that the application will not suffer the recovery overhead as long as the underlying network is not violating this tolerance.

Another related advantage of the triplet specification is that an application has the flexibility of specifying a lower tolerance and letting the transport level do the recovery, or specifying a higher tolerance and dealing with the loss on its own. For instance, when an additional FEC scheme is used, the tolerance can be set so that retransmission is performed only as necessary for the FEC scheme to recover the loss.

[3] In general, N can be any positive integer. For many applications, we expect a segment to be a natural unit for defining error tolerance and thus we will assume N to be the segment size measured in number of packets in this paper.

In fact, researchers have proposed HDTV transmission and error concealment schemes that can take good advantage of our scheme [14, 20].

2.2. Error Detection

Data segments are divided into packets for transmission. Each packet contains two sequence numbers (i,j), where i is the segment number and j is the packet number within the segment. For normal transmission, all packets will be sent in the order of increasing segment numbers and again in increasing packet numbers within a segment. Unless being used explicitly as i and j, we will refer to (i,j) as one packet sequence number, which is the concatenation of i and j, in the rest of the paper.

Using Gap-Detection and Loss Timer

In order to perform recovery according to the tolerance of the application, the receiver has to make the final decision for accepting a segment. SSTP performs loss detection at the receiver side so that the amount of state-information exchange between the sender and the receiver is minimized. A gap-based detection mechanism is used at the receiver to achieve early detection of packet losses. The effectiveness of a gap-based detection scheme depends on the following two conditions:

- Data packets are always transmitted in a particular order at the sender.
- The underlying networks guarantee a maximum re-sequencing distance for all the packets transmitted.

Let the re-sequencing distance guaranteed by the network be m. Then, the receiver can deem a missing packet lost if another packet transmitted m or more packets after this packet has arrived. For example, consider the first-time transmission of segment i. The first packet of this segment will have a sequence number $(i,1)$. According to the above assumptions, all the packets of segment i missing at the receiver when any packet $(i+1,k)$ with $k \geq m$ arrives must have been lost. The gap-based scheme assumes guarantees for in-sequence delivery which is a more realistic assumption than guarantees of data rate. Schemes using only a timer at the receiver for loss detection will require stringent guarantees for the data rate.

Due to packet retransmissions, the segment and packet sequence numbers contained in each packet do not identify uniquely the order in which a packet is transmitted at the sender. Therefore, they cannot be used for gap detection. Figure 2 shows an example scenario. Although the receiver detects that there is a gap between the sequence numbers (3,2) and (8,3), it cannot determine for sure whether there is

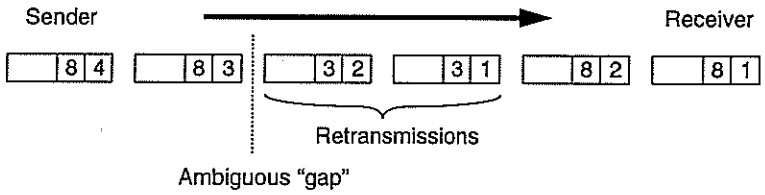

Figure 2: Sequence Number Ambiguity

any packet lost between (3,2) and (8,3). Because it is possible that some packets from segments 4 to 7 have been retransmitted and got lost, but also possible that the sender have not retransmitted them at all because the retransmission requests have not reached the sender yet. SSTP uses a shipment sequence number attached to each packet to solve this problem. Every packet, new or retransmission, is assigned a unique shipment sequence number according to the sequence it is sent out at the sender. Any gap larger than m in the shipment sequence number observed at the receiver indicates a loss of packets. When the sender temporarily runs out of data to send, it sends special control packets. These packets get their shipment sequence numbers from the same space of monotonically increasing sequence numbers. The special packets provide the necessary context for the receiver to detect losses in the segments transmitted earlier and also informs the receiver that the sender is idle but alive.

In addition to the gap detection mechanism, the receiver also uses a loss timer. The purpose of the timer is to detect very long burst of losses, in which there is no subsequent packets to enable detection of gaps. The duration of the loss timer is on the order of several segments transmitting time.

Overall, the combination of gap-detection with a loss timer ensures that early loss detection can be made. Early detections make quick recovery from losses possible.

2.3. Error Recovery

Once loss detection is made, recovery from the loss involves three steps: (1) The receiver has to determine which lost packets if any need to be requested for retransmission. (2) The request information needs to be passed to the sender. (3) The sender needs to verify the request and retransmit the necessary packets. These issues are discussed next.

Determining Retransmission List

The packets that have been successfully received are marked with a "1" in the corresponding positions of a packet-presence bitmap. At the time of a loss detection, retransmission request list needs to be generated from the packet-presence bitmap and the application error tolerance. Determination of the request list is a trivial issue in existing transport protocols such as TCP because all lost packets are

retransmitted regardless of application tolerance. In high speed networks, packet transmitting time (packet length divided by transmitting rate) and other local processing overhead are very small compared to the round trip delay, which is the minimum delay incurred in a retransmission round[4]. Therefore, the primary optimization goal for the determination of the request list should be to minimize the number of retransmission rounds. Following this strategy, when a burst of loss causes violation of application loss tolerance, the whole burst will be included in the request list for retransmission. The retransmission request list is also in the form of a bitmap, but with "0" indicating a request for retransmission of the corresponding packet and "1" otherwise.

Sending Acknowledgments

There are two pieces of information concerning the state of the receiver that must be available to the sender in order to perform retransmission:

- Which packets have been correctly received so that the buffer at the sender can be released.
- Which packets need to be retransmitted.

Since loss detection is made at the receiver in SSTP, the receiver has to send explicit information about what packets need to be retransmitted and which segments have been correctly received.

SSTP uses two types of acknowledgments. The first type is positive acknowledgment (PACK). A PACK is only used to inform the sender that certain segments have been accepted and can be released; A PACK can acknowledge a single isolated segment or it can acknowledge a contiguous block of segments if the block is the next expected in sequence. A selective negative acknowledgment (SNAK) is the second type of acknowledgment. It is mainly used to request packet retransmissions from the sender. However, it also carries a segment sequence number that informs the sender of the acceptance of all segments with sequence numbers below this number. SSTP transmits SNAK messages to the sender periodically to avoid long delays in the case of a SNAK loss. Periodic transmission of a SNAK stops when the corresponding segment is successfully accepted at the receiver.

Avoiding Premature Retransmission

By detection of gaps, the receiver can quickly detect the occurrence of losses. However, it is difficult for the receiver to determine exactly which packets are lost based on only the receiver's state information, particularly when the new and retransmitted packets are overlapping. This difficulty is avoided by allowing the receiver to determine only a superset of the lost packets and request for their retransmissions.

[4] A retransmission round consists of one retransmission for all requested packets of a segment.

The sender can filter out the premature requests because it knows exactly what have been transmitted. The sender always records the time at which each segment is transmitted. Upon receiving a retransmission request, the sender compares the current time with the time recorded for the last transmission of the segment. The sender ignores the request if the segment was transmitted within one round trip time in the past. Note that in order for any legitimate acknowledgment of a segment to come back, it takes more than one round trip delay on the connection.

Just like in any other schemes, SSTP senders maintain an estimate of the roundtrip delay for the connection. However, the estimate does not need to be very accurate since it is used in conjunction with request information from the receiver. Although we do not specify forward error control (FEC) as an explicit transport level option, an FEC scheme can be used effectively with our scheme by choosing appropriate error tolerance specifications.

3. PACKET FORMATS AND PROTOCOL OPERATIONS

This section first describes the data and control packet formats necessary for carrying out the transport protocol functions, particularly those of error control. Then, a detailed description of the operations of the error control mechanisms is provided.

3.1. Packet Formats

There are five types of packets used for data transport and error control. There are two basic packet formats as shown in Figure 3. The width of each row is assumed to be 4 bytes.

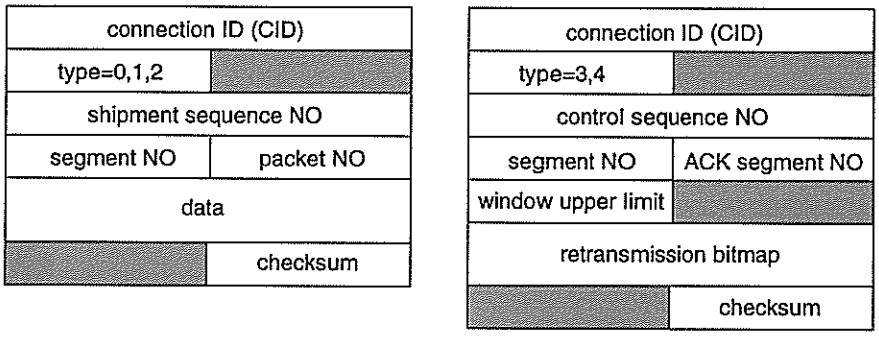

Figure 3: Error Control Packet Formats

All packets start with a 4-byte connection identifier field (CID) followed by a 2-byte type field and ends with a 2-byte checksum field. The CID identifies all the packets belonging to a connection. The type field contains a unique code number for each packet type. Of the five types, type 0, 1, and 2 are

for conveying information from sender to receiver; type 3 and 4 are used in the reverse direction. The checksum field is filled by the sender and then used by the receiver to detect packet corruption. The rest of the fields are summarized as follows:

Data Packet (type 0): Data packet is for carrying application data from the sender to the receiver. The shipment sequence number indicates the sequence at which the packet was last shipped from the sender and is used for loss detection. The segment number along with packet number identifies the position of this data block in the application data stream. The data itself is carried in the data field.

Keep-Alive Packet (type 1): Keep-alive packet is for informing the receiver that the sender is at a pause. Periodic transmission of this packet serves two purposes: (1) to prevent the receiver from closing the connection and (2) to provide the context (shipment sequence number, segment number, and packet number) for the receiver to detect losses. This format is the same as that of the data packet except that it does not have a data field.

EndStream Packet (type 2): End-of-stream packet is for informing the receiver of the end of segment stream. It also allows the receiver to make prompt detection of losses occurring in the last portion of the segment stream. This format is the same as that of the keep-alive packet.

PACK Packet (type 3): A PACK packet serves three purposes: (1) to acknowledge the acceptance of the segment specified by segment NO, (2) to cumulatively acknowledge all the segments with segment number below ACK segment NO, and (3) to optionally advance the sender's window by specifying a new window upper limit[5]. The control sequence NO is used by the sender to detect and discard duplicate PACK packets.

SNAK Packet (type 4): A SNAK packet is used to request retransmission of the packets specified by the retransmission bitmap and to cumulatively acknowledge all those segments with segment number below ACK segment NO. The most recently granted window limit is contained in field window upper limit to protect against the loss of the last window advancement message. The difference between this packet and the PACK packet is that it has a retransmission bitmap. This bitmap indicates retransmission requests for those packets where there is a "0" in the corresponding position of the bitmap.

[5] Although window control is not the focus of this paper, simple window control operations in SSTP will be explained as necessary when error control operations are presented. Details of the flow control are available in [12].

3.2. Error Control Operations

It is assumed that a connection between the sender and the receiver has already been set up. Again, application data consists of a sequence of segments, each having a size of s packets. The error control operations during the data transfer are described next.

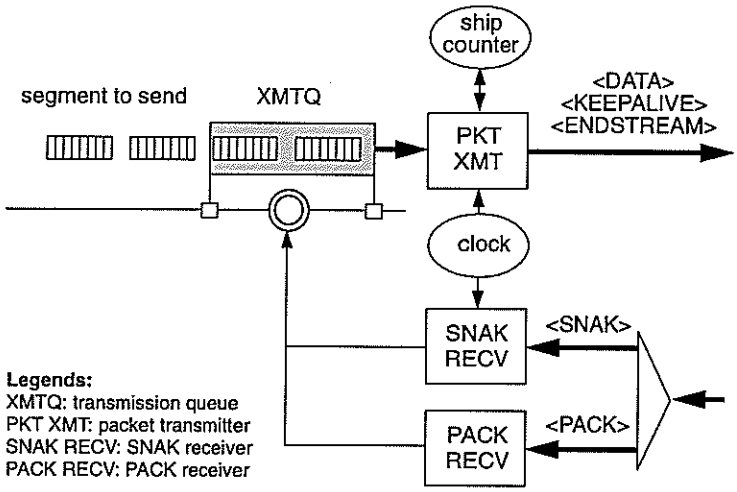

Figure 4: Sender Error Control Logic

Sender Operations

Main functions related to error control at the sender are:

- Schedule and transmit new segments.
- Process retransmission requests, schedule and retransmit requested packets.
- Process positive acknowledgments and release successfully delivered segments.

Figure 4 shows a logic diagram for the sender. There are three functional blocks for performing the error control functions, a packet transmitter (PKT XMT), a retransmission request receiver (SNAK RECV), and a positive acknowledgment receiver (PACK RECV). The transmission queue (XMTQ) contains all packets eligible for transmission. There is also a shipment counter for generating monotonically increasing shipment sequence numbers, and a time reference clock for retransmission control.

For each outstanding segment, there is also a data structure called segment control block, which contains information such as the status of the segment (e.g., in transmission or waiting for acknowledgment) and the packet retransmission bitmap. This data structure is not shown in Figure 4 for clarity reasons. The sender side functions are:

Packet Transmission: Transmission queue XMTQ is serviced in the order of increasing packet sequence numbers according to a given rate. For every packet leaving the transmission queue, transmission logic puts the current value of ship counter into the shipment sequence NO field of the packet and increments the counter. Whenever the queue becomes empty due to a pause of the application, a keep-alive packet is sent. An end-of-stream packet will be sent if the application has reached the end of its data stream. The keep-alive and end-of-stream packets carry the same segment number and packet number as the last data packet sent, but with their shipment sequence numbers incremented for each successive transmission. A 16 bit checksum is computed and appended to the checksum field before each packet is passed down to the lower level protocol for output.

Retransmission Request Processing: There is no retransmission unless it is explicitly requested by the receiver with a SNAK packet. Upon transmission of the last packet of a packet group[6], the sender records the current time (from the reference clock) in the control block of the corresponding segment. If a SNAK packet later arrives for this segment, the current time is compared with the previously recorded time and a retransmission is scheduled only if the last transmission for the segment was at least one round trip time ago. This way, spurious retransmissions can be reduced even if the receiver sends premature retransmission requests (SNAK packets). Packets to be retransmitted are scheduled into the same transmission queue as the first-time packets.

Positive Acknowledgment Processing: A segment is released from the buffer when it is acknowledged by the receiver. The acknowledgment for segment i can arrive at the sender in three different ways. The most common way is an explicit PACK packet with its segment number set to i. A second possible way is another PACK packet with a cumulative acknowledgment segment number (ACK segment NO) greater than i. Finally a segment i can also be acknowledged by a SNAK packet with a cumulative acknowledgment segment number greater than i. Every time the sender's window is advanced, data from new segments are packetized and scheduled into XMTQ for transmission.

Receiver Operations

There are three key error control functions at the receiver side:

- The receiver has to perform per packet processing that includes verification of checksum, check for duplicate packets and packets outside the window, and detection of gaps in the shipment sequence numbers.

[6] A packet group refers to all packets belonging to a common segment. For example, a segment is itself a packet group when first transmitted; all packets to be retransmitted for a segment define a packet group. A packet group is identified by the segment number.

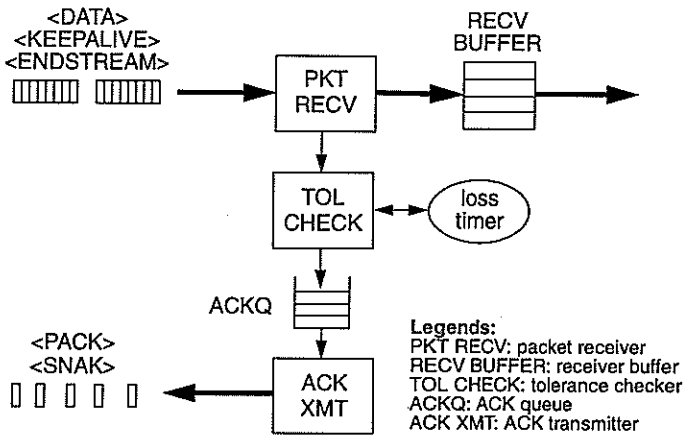

Figure 5: Receiver Error Control Logic

- Upon loss detection or reception of the last expected packet in a segment, the receiver needs to determine which packets, if any, to request for retransmission.
- Positive acknowledgments will need to be sent for the successfully accepted segments and retransmission requests (SNAK) have to be sent for those packets needing retransmissions.

These functions are carried out by three functional blocks as shown in Figure 5. The packet reception and gap detection function is performed by packet receiver (PKT RECV), the retransmission decision is made by the tolerance checker (TOL CHECK), and acknowledgments delivered by acknowledgment transmitter (ACK XMT). Also shown in the figure is a receiving buffer (RECV BUFFER) for re-sequencing data packets and segments, an acknowledgment queue (ACKQ) for keeping outstanding SNAKs, and a segment loss timer that detects long burst of losses.

There are several additional data structures that are omitted from Figure 5 but need to be introduced before describing the receiver operations. A variable L holds the highest segment number below which all of the segments have been acknowledged. A variable H records the maximum segment number of all packets correctly received so far. The receiver also maintains one segment presence bitmap (SEGMAP[W]) and W packet presence bitmaps (PKTMAP[W][s]), each for one segment in the window (i.e., the window size is W segments). By definition, PKTMAP[i][j] = 1 if and only if packet j of segment i has been successfully accepted; and SEGMAP[i] = 1 if and only if segment i has been accepted. A packet is successfully accepted if it is correctly received or deemed not necessary. A segment is successfully accepted if all the packets in the segment are successfully accepted. The receiver operations are summarized as follows:

Packet Reception and Gap Detection: The receiver (PKT RECV) operates according to the following steps upon reception of a packet:

1. Compute checksum for the packet and compare it with the value contained in the checksum field of the packet. Discard the packet if there is no exact match.
2. If this is a data packet, check whether the packet is within the current window. If not, discard it; otherwise, put the packet into the proper slot of the receiving buffer according to the segment and packet number. The packet is marked as *present* in the PKTMAP.
3. The tolerance checker is called to make acknowledgment decision under following conditions after receiving a packet:
   - If the data packet is the last expected of a segment, call tolerance checker for this segment.
   - If the shipment sequence number of the packet is more than m beyond the expected number, call the tolerance checker for all segments with sequence number $i \in [n_1, n_2 - 1]$. Note that n_1 is the segment number of the packet immediately before the gap, n_2 is the segment number of the current packet, and m is the maximum re-sequencing distance guaranteed by the underlying network.
   - If the keep-alive packet indicates that the last segment sent was n, but the receiver has only seen segments up to $H < n$, call tolerance checker for all segments with sequence number $i \in [H, n]$.
   - In the case of an end-of-stream packet, call tolerance checker for all the segments, from the next expected in sequence up to the last segment as indicated by the end-of-stream packet.
4. If the shipment sequence number of the packet is less than the next expected shipment sequence number, discard the duplicate packet; otherwise reset the loss timer.

Loss Timer Processing: Upon expiration of the loss timer, tolerance checker is called for all segments with sequence numbers in the range $[L, H+1]$ to make acknowledgment decisions. Recall that L is the next segment number yet to be accepted in sequence and H is the maximum segment number of all the packets correctly received so far. The loss timer is used for early detection of long burst of losses. Typically, the timer value should be on the order of several segments transmission time.

Checking Application Tolerance: Assume that the application tolerance is given as (N, E, B). When invoked, tolerance checker performs as follows:

1. If the application specifies 100% error tolerance (i.e., $E \geq N \wedge B \geq N$), all segments will be marked as *accepted* before going to step 4 to send acknowledgments.
2. If the application requires 100% error recovery (i.e., $E = 0 \vee B = 0$), no manipulation of the packet presence bitmap is necessary so skip to step 4 to send acknowledgments.

3. When the application has a partial tolerance, check the packet presence bitmap of each segment against the specified tolerance:
   - Start from the first packet and count the total number of missing packets as well as the number of missing ones with consecutive packet numbers.
   - If a missing packet does not cause violation of the tolerance it is marked as *present* in the bitmap, otherwise the whole burst of lost packets starting from the current one is left as *missing* and will be requested for retransmission later. This checking process continues with the next packet until all the packets in the segment are checked.
   - A segment is marked as *accepted* if there is no violation during the checking process.
4. After all segments have been checked, PACK or SNAK packets are scheduled for these segments as appropriate. If the newly accepted segments form the next contiguous block of data in sequence for the application, deliver the data to the application, set L to be the sequence number of the next segment to be accepted in sequence, and call ACK transmitter to send a PACK packet for the accepted block. If a segment is accepted but cannot be delivered to the application due to some incomplete segments ahead of it, a PACK packet is sent for only this segment. Finally, if a segment needs retransmission, call ACK transmitter to schedule a SNAK packet.

Acknowledgment Transmission: If a PACK packet needs to be sent, a new PACK packet will be created. The packet will contain the given segment sequence number, the current L value as its cumulative ACK sequence number, and the current window limit as its window upper limit. After a unique control sequence number is attached, the checksum is computed and the packet is transmitted by calling the lower level protocol function. In the case of a SNAK packet, the fields of the segment sequence number, the cumulative ACK sequence number, and the window upper limit are constructed the same way as for the PACK packet. However, the packet will contain an additional retransmission bitmap. The retransmission bitmap is simply a copy of the packet presence bitmap for the segment at the receiver. The SNAK packet is appended to the ACKQ for transmission to the sender. If the segment has been requested for retransmission before, a SNAK packet for the segment will be already in ACKQ. Then, the fields of the segment number, the cumulative ACK sequence number, the window upper limit, and the retransmission bitmap are updated with new values. SNAK packets are periodically transmitted from ACKQ to the sender. A SNAK packet stays in the queue until it is removed upon successful acceptance of the corresponding segment. A new control sequence number is attached and checksum computed every time before a SNAK packet is transmitted.

4. PERFORMANCE STUDY

This section first describes the objectives for our performance study and defines three performance measures to be used. Then, the analysis and simulation studies are summarized along with some results. It should be noted that in this study, no attempt will be made to compare the proposed scheme with a cumulative ACK scheme. The reason is that sufficient studies have been done by other researchers that have shown conclusively that selective ACK schemes are superior in performance than cumulative ACK schemes, especially in high bandwidth-delay product environments [9, 16, 23].

The performance study serves two main purposes. First, to obtain quantitative measures on how well the proposed scheme works in a typical high bandwidth-delay product network environment. Second, to gain insight into the working of the scheme so that the design can be improved and a better implementation can be achieved.

We use two performance measures, *throughput* and *delay*. To concentrate on the performance of the protocol mechanism itself, the maximum achievable throughput when application process is not a bottleneck is considered. Furthermore, a throughput normalized against the connection rate is used in order to obtain a direct measure of how efficient the mechanism can utilize the connection. Specifically, *Throughput Efficiency* is defined as *the ratio between the ideal segment transmitting time (segment size divided by the connection rate) and the actual time required on average to successfully deliver one segment.*

The *End-to-End Delay* of a segment is defined as *the time elapsed from the start of transmission at the sender till the successful acceptance of the segment at the receiver*. This delay definition is from the receiver's point of view. If it were defined from the sender's point of view, there will be additional time for an acknowledgment to reach the sender. The former definition is chosen because it reflects exactly when the segment is available for consumption at the receiver.

Following the basic definition of end-to-end delay, *Average End-to-End Delay* is defined as the average of end-to-end delays over all the segments and *Maximum End-to-End Delay* is the maximum of end-to-end delays among all the segments.

4.1. Evaluation Approaches

Both analysis and simulation are used in the study of the error control scheme. Due to the complexity of the error control process, only approximate analyses for average delay and throughput efficiency are pursued. The purposes of the analyses are to gain insights through the derivation of simple performance expressions and to verify the simulation model. Discrete event simulation is used for more accurate study of the various aspects of the error control. For example, the effect of bursty losses and the application-dependent error recovery is examined through simulations.

Delay Analysis

Given the earlier definition for average end-to-end delay, it can be expected to be much less sensitive to the window control function. This is because the delay is not measured until the transmission of a segment begins and the window only determines if the transmission of a segment can begin. Once a segment begins transmission, the delay caused by transmitting packets of another overlapping segment should be small compared to the acknowledgment delay. Therefore, the average end-to-end delay for a large number of segments is approximated by the average delay of a single segment transmitted in isolation.

Let the packet size be l bytes, the segment size be s packets, and the data rate be A bps (bits/second). The packet transmitting time is $t_p = (8 \times l)/A$ seconds. The following specific assumptions apply to this analysis:

- A packet is lost if it is either corrupted or dropped in the underlying network. Each new packet transmission can be lost independently with probability p.
- An acknowledgment (PACK or SNAK) always comes back to the sender t_a time after the transmission of the last packet of a packet group. Acknowledgment delay t_a is determined as $t_a = RTD + 3 \times t_p$. That is, it takes one round trip delay plus an assumed acknowledgment delay of $3 \times t_p$ in the receiver, for an acknowledgment to return to the sender. Note that redundancy in acknowledgment information transmission makes it reasonable to ignore PACK and SNAK losses in this analysis.
- Application requires 100% reliable delivery of all segments.

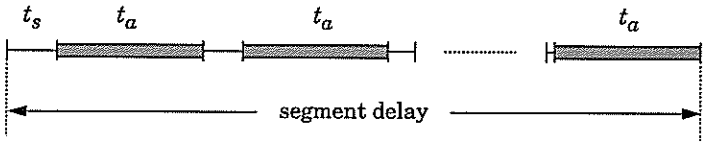

Figure 6: A General Segment Delivery Scenario

Figure 6 depicts a general scenario for the delivery of a segment. A segment is successfully delivered after at least one round of attempt. The first round consists of the transmitting time t_s for the whole segment followed by the waiting time t_a for the acknowledgment (the first shaded area in the figure); in each subsequent round, the lost packets from the previous round are retransmitted and another t_a has to pass by before an acknowledgment is received; no more attempt will be needed when a positive acknowledgment is received. Let D_s denote the average segment delay as shown in the figure, it contains clearly three parts:

$$D_s = Transmitting\ Time + Mean\ Retransmission\ Time + Mean\ Waiting\ Time$$

The segment transmitting time is simply $s \times t_p$. The second term, the mean retransmission time for the segment is $s \times t_p \sum_{i=0}^{\infty} i \times p^i \times (1-p)$. As each round has a constant waiting time t_a, the mean waiting time equals t_a times the mean number of rounds, which is $t_a \sum_{i=1}^{\infty} i \times [(1-p^i)^s - (1-p^{i-1})^s]$. Therefore:

$$\begin{aligned} D_s &= st_p + st_p \sum_{i=0}^{\infty} ip^i(1-p) + t_a \sum_{i=1}^{\infty} i[(1-p^i)^s - (1-p^{i-1})^s] \\ &= \frac{t_s}{1-p} + t_a \sum_{i=0}^{\infty} [1-(1-p^i)^s] \end{aligned}$$

Derivation of D_s assumed sender's point of view. Let D_r be the average delay as defined from the receiver's point of view, it is then related to D_s as follows:

$$D_r = D_s - \frac{1}{2}RTD - 3t_p$$

Numerical results from this expression are presented in Section 4.2. It should be mentioned that a similar analysis has been presented in [16] for a selective acknowledgment scheme that uses only packet granularity (i.e., there is no concept of a segment). In that analysis, expressions were derived for both average delay (mean) and the variance, while the main interest here is the mean value.

Throughput Efficiency Analysis

Full throughput analysis of error control schemes using selective ACK is very difficult. Existing studies (e.g., [16, 23]) have made simplifying assumptions such as absence of window flow control and no overlap between transmission and retransmission. These assumptions, though still allowing one to demonstrate the superiority of the selective ACK over the cumulative ACK, are too unrealistic for the operations of real schemes such as the one proposed. The approach used by Doshi and his colleagues in the analysis of SNR protocol is the only exception to this account (for the protocol see [17] and for analysis [9]), and it is the main inspiration for the throughput analysis to be presented.

In addition to those assumptions made for the delay analysis, we assume that the probability of requiring more than one round of retransmission for each segment, i.e., $(1-(1-p^2)^s)$, is negligible. With this assumption, we are able to identify regenerative cycles for the error control operation on a connection. Within one cycle, we first determine the actual number of segments delivered and the ideal number of segments deliverable under error-free conditions. The ratio of the actual number of segment to the ideal number of segment defines the throughput efficiency.

Let $g = [1-(1-p)^s]$ be the probability that a segment will require at least one packet retransmission. Let W be the number of segments in a window and $w_a = t_a/t_s$. Also, let ThE be the throughput

efficiency. First, we consider the case with $1 < W \leq (w_a + 1)$. In this case, the actual number of segments delivered in one cycle is:

$$N_{actual} \approx g^{-1} + W$$

because the window limits the number of segment transmissions to W during the whole time period $(2t_a + t_s + pt_s)$ and there are g^{-1} segments on average that can be delivered without loss. Now consider how many segments can be delivered under ideal condition. First of all, there can be $(2w_a + 1 + p)$ segments delivered during the time $(2t_a + t_s + pt_s)$. Since the smaller window forces the sender to wait from time to time during the transmission of the g^{-1} loss-free segments, it equivalently requires $\frac{(w_a+1)}{W}t_s$ time to deliver each segment. Thus, the actual total time for delivering g^{-1} segments is $\frac{(w_a+1)}{W}g^{-1}t_s$. Ideally, exactly $\frac{(w_a+1)}{W}g^{-1}$ segments would have been delivered. The ideal number of segments deliverable is therefore:

$$N_{ideal} \approx 2w_a + 1 + p + \frac{(w_a + 1)}{W}g^{-1}$$

and by definition:

$$\begin{aligned} ThE &\approx \frac{N_{actual}}{N_{ideal}} \\ &\approx \frac{g^{-1} + W}{2w_a + 1 + p + \frac{(w_a+1)}{W}g^{-1}} \quad 1 < W \leq (w_a + 1) \end{aligned}$$

Similarly, the following expression can be derived for $(w_a + 1) < W < (2w_a + 1)$:

$$ThE \approx \frac{w_a + \min(W - w_a, w_a(1-p)) + g^{-1}}{2w_a + 1 + p + \min(W - w_a, w_a(1-p)) + g^{-1}} \quad (w_a + 1) < W < (2w_a + 1)$$

Again, the numerator represents the actual number of segments delivered in one cycle and the denominator is the number of segments deliverable under ideal condition.

Note that due to the simplifying assumptions made, the identified cycle is only appropriate for deriving expressions with $1 < W < (2w_a + 1)$. For $W = 1$, using the average end-to-end delay D_s derived earlier:

$$ThE \approx \frac{t_s}{D_s}$$

When $W \geq 2w_a + 1$, the only reduction in throughput is due to retransmission of packets, therefore:

$$ThE \approx (1 - p)$$

Numerical results obtained using these expressions will be shown in Section 4.2.

Discrete Event Simulation

Discrete simulations are used to study the effect of transmission scheduling, application-dependent selective retransmission, and bursty packet loss. The simulation configuration is shown in Figure 7.

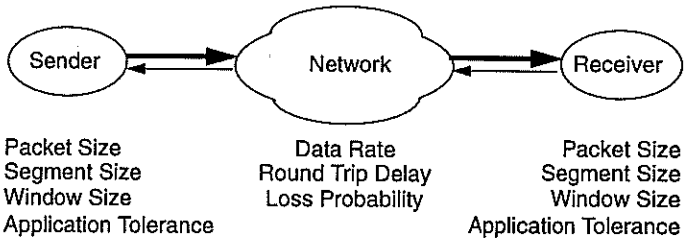

Figure 7: Simulation Configuration

The underlying network is modeled as a "black box" characterized by a connection data bandwidth, a round trip delay, and a packet loss process. The end system consists of two transport entities, a sender and a receiver. The end system parameters include packet size, segment size, window size, and application error tolerance. Performance measures of main interest are throughput efficiency, average end-to-end delay, and maximum end-to-end delay of a segment.

There are other parameters that can affect the performance of an error control scheme. For example, the total number of data segments to be transported and the amount of physical memory in the sender and receiver. We minimized these effects in the simulation by transmitting a large number of packets ($\geq 10^7$) and by assuming very fast processors with large memories.

The following assumptions are made for the simulation.

- The round trip delay (RTD) on the connection does not vary in the duration of data transfer and the delay on each direction is RTD/2.
- All control packets are sent "out-of-band" which means they do not consume the bandwidth of data connection.
- An acknowledgment (PACK or SNAK) always comes back to the sender t_a time after the transmission of the last packet of a packet group.
- Receivers always have sufficient buffer space to match the window size used, and therefore, no overflow can occur at receivers.

In the simulation, bursty loss process is modeled with two alternating states, good and bad, as shown in Figure 8. There is no loss in the good state while packet loss occurs with probability p_b in the bad state. The durations for each state is deterministic.

SNR Error Control Scheme

SNR is a transport protocol developed in AT&T Bell Labs [17]. The most important features of SNR include: (1) periodic and complete state exchange between sender and receiver and (2) selective retrans-

Figure 8: Bursty Loss Process

Features of Scheme	Proposed	SNR
Location for Loss Detection	Receiver	Sender
Loss Detection Method	Gap Detection & Timer	Timer
Error Control Unit	Segment	Block
Retransmission Strategy (Unit)	Selective (Packet)	Selective (Block)
Control Info Sender→ Receiver (Frequency)	Pause/End (When Idle)	Highest seq# (Periodic T_{in})
Control Info Receiver→ Sender (Frequency)	Cumulative ACK & Selective NAK (Periodic NAK)	Cumulative ACK & Selective NAK (Periodic T_{in})
Support for Variable Grades of Reliability	Application Dependent Recovery	Yes or No Error Control

Table 1: SSTP Scheme versus SNR Scheme

mission based on blocks (block=multiple packets). SNR design has greatly influenced the error control in SSTP. Also, key ideas of SNR have been accepted as part of the B-ISDN T1S1 standard. Doshi and his colleagues have also shown that SNR error control scheme outperforms those used in existing protocols [9]. Therefore, we choose to compare SSTP with SNR. Table 1 provides a qualitative comparison between the main features of the two schemes. SNR uses a single timer at the sender side for error detection but SSTP uses a gap detector along with a very loose timer at the receiver side; in SNR, the sender and the receiver continuously exchange state information at an interval T_{in} while SSTP sends separate control packets from the sender to the receiver only when there is no data to send, and sends periodic control packets from the receiver to the sender only when retransmission is required; finally, SNR allows selection of with or without retransmission whereas SSTP supports variable degrees of retransmission based on the application error tolerance. These differences in design will be reflected in the behavior of the protocol to be seen in the next section.

4.2. Numerical Results

This section presents and discusses some SSTP performance results.

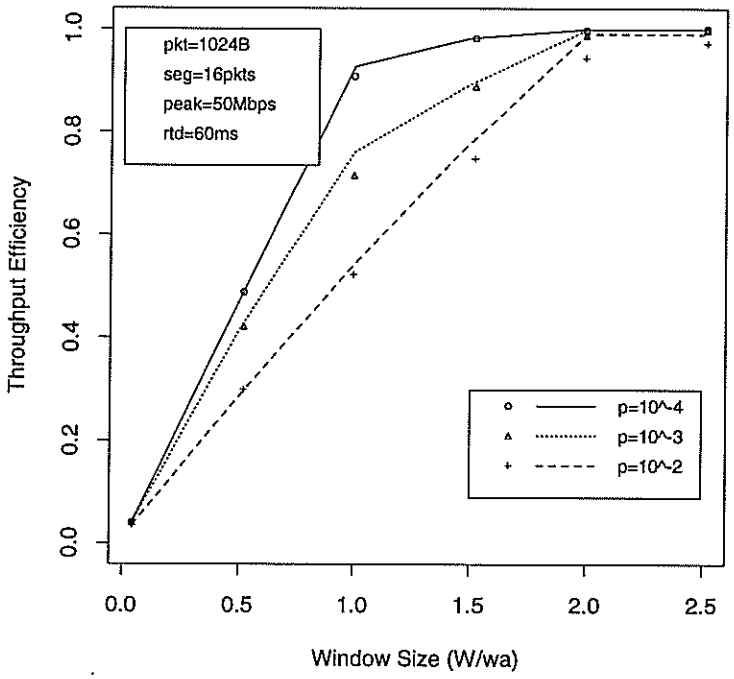

Figure 9: Throughput Efficiency – Analysis vs. Simulation

Throughput Efficiency with Random Loss

Figure 9 shows throughput efficiency against window size. The results from both the analysis and the simulation are shown for comparison. The connected lines correspond to the analysis and the discrete symbols are for simulation. The loss probability was varied to obtain a family of three data sets which are labeled with different symbols in the figure. The error control was providing 100% loss recovery. The following can be observed from this plot:

- With very small loss probability ($\leq 10^{-4}$), a window size of $(1+w_a)$ is sufficient for achieving close to maximum throughput efficiency. Because with no loss, a window of size $(1+w_a)$ is sufficient to keep the sender busy all the time.
- Larger window sizes are necessary to achieve the same efficiency when loss probability is higher. However, with loss probability of up to 10^{-2}, a window size of about 2.5 times bandwidth-delay product can achieve almost perfect throughput efficiency. The factor of 2.5 is in the same range as that found by Doshi et al. in SNR study. Although $2.5 \times w_a$ may correspond to a fairly large memory

requirement for large bandwidth-delay product networks, the factor 2.5 should be reasonable if very high bandwidth utilization is desired.

- The throughput efficiency expressions derived in the analysis seem to provide quite accurate prediction for loss probabilities up to 10^{-2}. But as expected, the analysis consistently predicts higher throughput due to the optimistic assumption about packet loss.

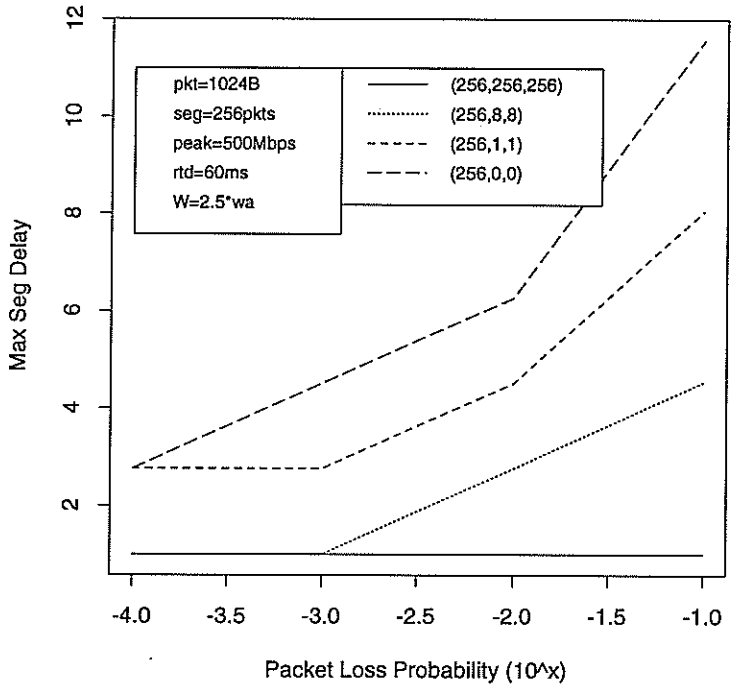

Figure 10: Maximum End-to-End Delay

Maximum Delay with Random Loss

To study the effect of packet loss on maximum segment delay and evaluate the effectiveness of the application-dependent recovery mechanism, another simulation experiment was conducted. In this experiment, the error control mechanism dynamically recovers from losses according to four different settings of application tolerance. The collected maximum delay is plotted against packet loss probability in Figure 10. The family of four curves corresponds to the four different error tolerances as shown in the legend. The results demonstrate that:

- When the application requires 100% loss recovery (i.e., tolerance=(256,0,0)), it suffers significant increase in maximum delay with increasing loss probability; on the other extreme, it is immune to the increased loss if it can tolerate all losses (i.e., tolerance=(256,256,256)) as one would expect.

- More importantly, it is seen that lower maximum delay can be achieved by tolerating more packet losses using the scheme across a wide range of tolerances. For example, with loss probability equal to 10^{-3}, an application can reduce its maximum delay from 154 ms to 94 ms by tolerating only one packet loss in a segment and the saving of 60 ms is a very significant amount of processing time.

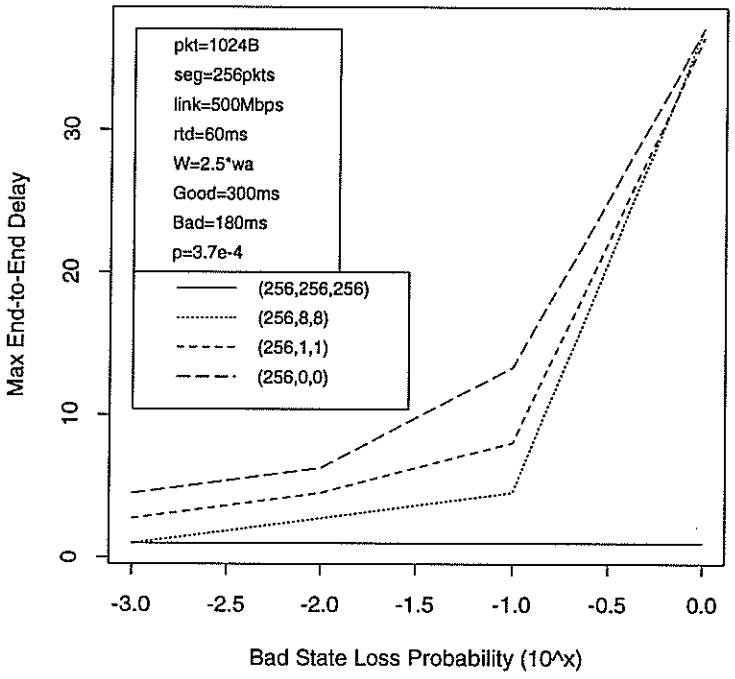

Figure 11: Maximum End-to-End Delay with Variable Tolerance

Application-Dependent Recovery with Bursty Loss

Figure 11 shows the trade-off between the loss tolerance and the maximum segment delay achievable under bursty losses. To maintain the readability of the plot, only the result for mean bad state duration 180 ms is included. The set of four curves represent the maximum delay vs. loss probability relationships for four different loss tolerances. In the legend, p represents the effective loss probability for the corresponding bad state duration, with bad state loss probability 10^{-3}. The effective loss probability is computed as $p = \frac{p_b d_b}{d_g + d_b}$, where d_g and d_b are the mean duration time for good and bad states respectively. The following can be observed from the plot:

- Similar to the results under random losses, as the application tolerance increases from $(256, 0, 0)$ to $(256, 256, 256)$ the maximum segment delay shows significant decrease accordingly.

- As the bad state loss probability approaches 1 the differences among the three lowest tolerance settings diminish. This is not surprising for the following reasons:
  - All three tolerances will be violated under the same condition because every packet will be lost once it is in a bad state.
  - Recall that the retransmission strategy will request retransmission for all the packets lost in a burst from the point the tolerance is violated, which include almost all the packets lost for all three tolerance settings.
  - Once the three tolerance settings lead to the same number of segment retransmissions, waiting time due the round trip delay affects them the same way.

Additional results with different bad state durations have shown that: (1) As the bad state duration increases, the trade-off between tolerance and delay becomes much more significant as an indirect result of increased impact of bursty loss on the maximum delay; (2) When the bad state duration is decreased, the reduction in maximum delay with increasing tolerance becomes less. However, with bad state duration as short as 15 ms the trade-offs with the same set of tolerances as used above are still appreciable.

Throughput-SNR vs. SSTP

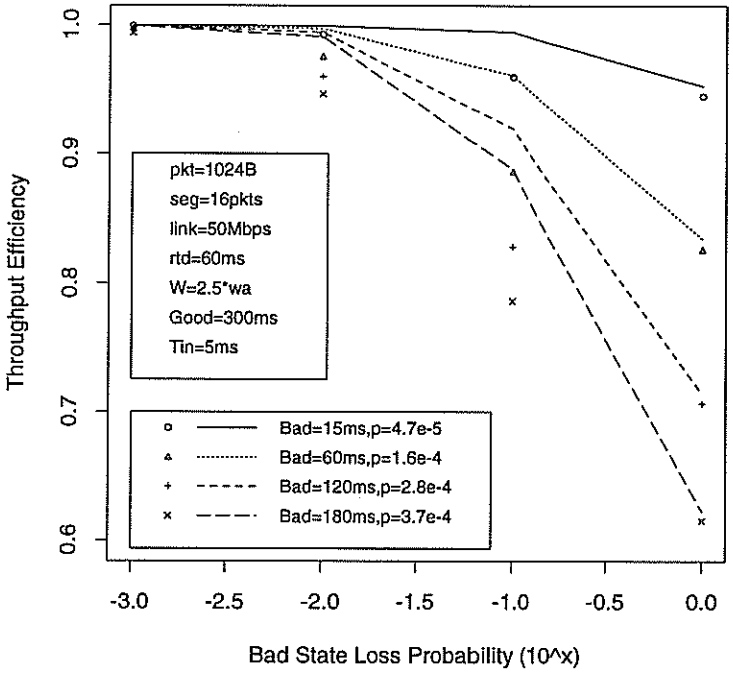

Figure 12: Throughput Efficiency Comparison

Figure 12 shows a comparison of throughput efficiency between SSTP (in line style) and SNR (by discrete points). There are four sets of data corresponding to four different mean bad state durations. As expected, there is little difference between the two schemes when bad state loss probability is very low ($< 10^{-3}$); but as bad state loss probability increases, SSTP performs better and better than SNR; the difference reaches a maximum before it starts to diminish as the bad state loss probability approaches 1. The diminishing difference is also expected, because when almost every packet is lost in the bad state, selective retransmission at the packet level has very little advantage over retransmission at the block (segment) level. It should also be noted that, the performance advantage of the proposed scheme increases with longer bad state duration.

Simulations have also been done for comparison of SNR and SSTP under random loss conditions. Throughput efficiency, average delay, and maximum delay were all collected for each of the simulations. All the comparison results showed superior performance by SSTP. This performance difference stems mainly from the fact that SNR performs retransmission on a per block (multiple packets) basis whereas the proposed scheme selectively retransmits only lost packets. We recognize that the main incentive for retransmitting larger blocks is to reduce the amount of state information for retransmission and to simplify retransmission logic. However, for demanding applications such as distributed computing with visualization, we believe that the additional complexity is well justified given the improvement in performance. The shorter delay in requesting for retransmission is another factor that contributes to the better performance for SSTP. In the case of the SNR protocol, transmission of the receiver state information to the sender is strictly time-driven. The minimum interval at which this occurs is T_{in}, determined as a fraction of the round trip delay RTD. In SSTP, sending of the first SNAK packet for a segment is driven by the detection of a loss. Following the example of Doshi et al., T_{in} is set to 5 ms for RTD = 60 in the SNR simulations. This results in an average delay in sending retransmission request of $T_{in}/2 = 2.5$ ms. However, SSTP assumes a delay for initiating a request (SNAK) that is equivalent to the transmission time for three packets ($3 \times t_p$). For all cases studied, this delay is less than 2.5 ms, and therefore, SSTP has a smaller loss recovery delay. Determining T_{in} solely based on RTD but not the average bandwidth of the connection does not seem sufficient for achieving efficient loss recovery.

5. IMPLEMENTATION

As mentioned earlier, Segment Streaming Transport Protocol (SSTP) has been implemented inside the SunOS 4.0.3 kernel. It is built on top of a connection-oriented internet protocol (COIP) [7, 15]. Figure 13 displays the protocol hierarchy used in the SSTP implementation. Applications use SSTP service through the standard socket interface. SSTP provides flow-controlled segment streams with variable degrees of reliability by building on the services provided by CTP (COIP test protocol); and CTP is supported by

the Ethernet protocol CSMA/CD.

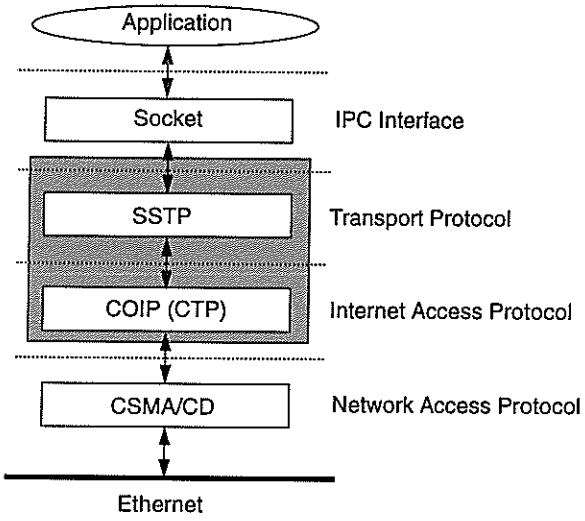

Figure 13: Protocol Hierarchy

Extensive trace data has been collected that verified the error control function. We also measured the throughput performance of the SSTP implementation using both custom software and a kernel probe technique developed by Papadopoulos [19]. The protocol processing delay results are summarized in Table 2. Protocol processing delays at both the sending and receiving ends are measured on a large number of packets. The resulting average is shown in the second column of the table. Note that this delay measure does not include the time for copying data from the application space to the kernel space or vice versa. In the last column, we also show the theoretical throughput corresponding to the given processing delay.

Sending/Receiving	Average Delay	Theoretical Throughput
Sending	273 μs	30 Mbps
Receiving	310 μs	26 Mbps

Table 2: Per Packet Processing Delay

It is worth noting that the corresponding theoretical throughput for TCP/IP has been found to be about 22 Mbps. Thus, SSTP/COIP is about 20% faster than existing TCP/IP implementation. While TCP/IP implementation has been carefully crafted over the years, very little effort has been made to optimize implementations of SSTP or COIP. Furthermore, efficiency of the SSTP/COIP implementations are limited by the mbuf-based memory management scheme and by the lack of efficient timer support in the operating system kernel. We expect significant performance improvement for SSTP/COIP with the removal of these constraints and with additional hardware assistance. In particular, we expect SSTP to

perform much better than TCP in high-speed network environments for imaging applications with error tolerance, because SSTP has mechanisms that ensure prompt detection and recovery from errors even in networks with large bandwidth-delay product and it deals with segments and packets which are much larger units and fixed per connection, which has lower processing complexity.

6. SUMMARY

This paper has presented an application-oriented error control scheme for applications such as distributed pipeline computing with visualization in high speed networks. The proposed scheme has four novel features: the triplet specification for loss tolerance allows accurate description of the application's reliability requirement and also lends itself to easy enforcement; the recovery scheme performs selective packet retransmission according to application loss tolerance, thus minimizing retransmission delay; it uses a gap-based loss detection method to ensure early loss detection; and the scheme also uses redundant retransmission requests by sending a request periodically from the receiver to the sender, which avoids long recovery delay in the face of a request loss.

Analysis and simulation of the error control scheme have also been presented. The results support the following conclusions: (1) The proposed scheme can achieve high throughput with reasonable buffer requirements and with a range of random and bursty loss conditions. (2) The maximum delay increases significantly with packet losses if the application expects 100% reliability. This leads to a large variance in segment delay. (3) The proposed application-dependent loss recovery scheme provides an effective means for controlling the trade-off between error tolerance and the maximum segment delay. (4) The proposed scheme gives superior throughput and delay performances than the SNR scheme which has been shown to outperform other proposed schemes. Our measurement results show that even the primitive SSTP/COIP implementation performs significantly better than the well-crafted TCP/IP protocol. In summary, our experience shows that more flexible error control services can be provided effectively through simple mechanisms. It also demonstrates that more efficient error recovery can be achieved through integrated consideration of the application requirements, the available network services, and the characteristics of the network environment.

References

[1] Brodd, W.D. and R.A. Donnan, "Data Link Control Improvements for Satellite Transmission", in J.L. Grange (ed), *Satellites and Computer Communication*, North-Holland Publ., Amsterdam, The Netherlands, 1983, pp. 201–213.

[2] Biersack, E.W., et al., "An Overview of the TP++ Transport Protocol Project", internal draft, Computer Communication Research Group, Bellcore, March 1991.

[3] Cheriton, David, "VMTP: A Transport Protocol for the Next Generation of Computer Systems", *SIGCOMM '86 Symposium: Communications Architectures and Protocols (Computer Communication Review)*, Vol. 16, No. 3, ACM, New York, 1986, pp. 406–415.

[4] Chesson, Greg, "Protocol Engine Design", *Proceeding of the Usenix Conference*, 1986.

[5] Chesson, Greg, et al., "XTP Protocol Definition", Revision 3.1, Protocol Engines, Inc., PEI 88–13, Santa Barbara, Calif., 1988.

[6] Clark, David D., Mark L. Lambert, and LiXia Zhang, "NETBLT: A Bulk Data Transfer Protocol", *Network Working Group RFC 998*, March 1987.

[7] Cranor, Charles D., *An Implementation Model for Connection-Oriented Internet Protocols*, Washington University Computer Science Department, Master thesis, May 1992.

[8] Doeringer, Willibald A., et al., "A Survey of Light-Weight Transport Protocols for High-Speed Networks", *IEEE Trans. Communications*, Vol. 38, No. 11, November 1990, pp. 2025–2039.

[9] Doshi, B. T., et al., "Error and Flow Control Performance of a High Speed Protocol", internal draft, AT&T Bell Laboratories, 1991.

[10] Department of Defense, "Transmission Control Protocol", *MIL-STD-1778*, 20 May 1983.

[11] Droms, Ralph E., et al., "Report from the Joint SIGGRAPH/SIGCOMM Workshop on Graphics and Networking", *Computer Communication Review*, Vol. 21, No. 2, 1991, pp. 17–25.

[12] Gong, Fengmin, *A Transport Solution for Pipelined Network Computing*, D.Sc. dissertation, Washington University Computer Science Department, St. Louis, December 1992.

[13] Gong, Fengmin, and Gurudatta M. Parulkar, "Segment Streaming for Efficient Pipelined Televisualization", *Conference Record, IEEE Military Communications Conference*, Vol. 3, October 11–14, 1992, San Diego, pp. 991–997.

[14] Joseph, K., et al., "MPEG++: A robust compression and transport system for digital HDTV," *Signal Processing: Image Communication* 4 (1992) 307-323.

[15] Mazraani, Tony Y. and Gurudatta M. Parulkar, "Specification of a Multipoint Congram-Oriented High Performance Internet Protocol", *Proceedings of the Ninth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'90)* IEEE Computer Society, Washington

D.C., June 1990, abridged from: Washington University Department of Computer Science, technical report WUCS-89-20, St. Louis, Aug. 1989.

[16] Mukherjee, Amarnath, *Analysis of Error Control and Congestion Control Protocols*, Ph.D. dissertation, Department of Computer Science, University of Wisconsin, November 1990.

[17] Netravali, Arun N., W. D. Roome, and K. Sabnani, "Design and Implementation of a High-Speed Transport Protocol", *IEEE Trans. on Communications*, Vol. 38, No. 11, November 1990, pp. 2010–2024.

[18] Parulkar, Gurudatta M., "The Next Generation of Internetworking" , *Computer Communication Review*, Vol. 20, No. 1, ACM SIGCOMM, New York, Jan. 1990, pp. 18–43, also: Washington University Department of Computer Science, technical report WUCS-89-19, St Louis, May 1989.

[19] Papdopoulos, Christos, *Remote Visualization on a Campus Network*, Washington University Computer Science Department, Master thesis, August 1992.

[20] Sun, H., et al., *Error Concealment Algorithms for Robust Decoding of MPEG Compressed Video*, Technical Report, NEC Computer and Communications Laboratories, TR# 93-C028-5031-1.

[21] Tanenbaum, Andrew S., *Computer Networks*, Prentice-Hall 1988.

[22] R. W. Watson, "The delta-*t* transport protocol: Features and experience," in *Proc. IFIP Workshop Protocols High-Speed Networks*, Rüschlikon, May 9-11, 1989, pp. 3–17.

[23] Zhou, Xiao You and Ahmed E. Kamal, "Automatic Repeat-Request Protocols and Their Queueing Analysis", *Computer Communications*, Vol. 13, No. 5, June 1990, pp. 298–311.