Report Number: WUCS-92-38

1992-11-01

A Two-Level Flow Control Scheme for High Speed Networks

Fengmin Gong and Guru Parulkar

Many new network applications demand interprocess communication (IPC) services with guaranteed bandwidth, delay, and low. Existing transport protocol mechanisms have not been designed with these service objectives. Large bandwidth-delay products of high-speed networks also render the existing flow control mechanism inefficient. This paper presents the design, evaluation, and implementation of a two-level flow control scheme that can support efficient IPC for these applications in high-speed network environments.

A Two-Level Flow Control Scheme for High Speed Networks

Fengmin Gong and Gurudatta M. Parulkar

WUCS-92-38

November 1992

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

A Two-Level Flow Control Scheme For High Speed Networks[†]

Fengmin Gong
MCNC Center for Communications
gong@concert.net
Phone 919-248-9214
FAX 919-248-1405

Guru Parulkar
Computer and Communications Research Center
Washington University, St. Louis
guru@flora.wustl.edu
Phone 314-935-4621
FAX 314-935-7302

Abstract

Many new network applications demand interprocess communication (IPC) services with guaranteed bandwidth, delay, and loss. Existing transport protocol mechanisms have not been designed with these service objectives. Large bandwidth-delay products of high-speed networks also render the existing flow control mechanisms inefficient. This paper presents the design, evaluation, and implementation of a two-level flow control scheme that can support efficient IPC for these applications in high-speed network environments.

[†]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.

1. INTRODUCTION

The main function of *flow control* is to regulate the speed of data transmission between two communicating devices so that the maximum possible speed is achieved without overflowing the receiving device or any forwarding devices. For example, in the scenario shown in Figure 1 data is being transferred from a supercomputer to a workstation via a connection consisting of several switches and gateways. The underlying network is referred to as a very high speed internetwork (VHSI). Flow control is required to ensure that:

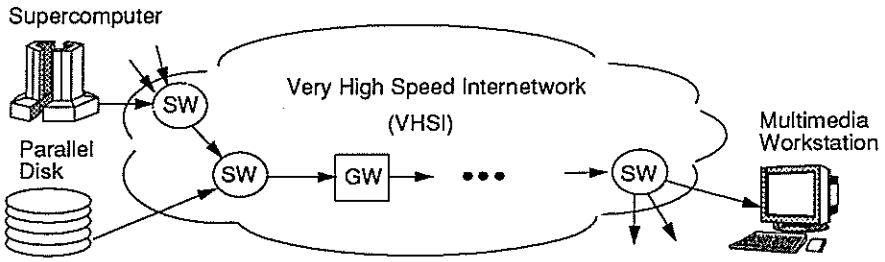


Figure 1: A Communication Scenario

- The supercomputer will not send data faster than the underlying VHSI can forward.
- The switches and gateways inside VHSI will forward data with minimum delay.
- The supercomputer will not overflow the buffer of the receiving application on the workstation.

While the flow control between neighboring switches inside VHSI can be handled efficiently using a *hop-to-hop* control mechanism, an *end-to-end* control mechanism is required to fulfill the other two functions. This paper focuses on the end-to-end flow control problem.

There are many proposed end-to-end flow control schemes and some of them have been implemented in transport protocols [10, 20]. For example, a sliding window scheme is used in Transmission Control Protocol (TCP) [13] and a rate control scheme is used in NETBLT [6]. However, the emerging high speed networks such as ATM and new distributed scientific computing applications have changed many assumptions on which the design of existing schemes has based. Examples of such assumptions are: the networks provide only unreliable datagram-oriented services, they have relatively small bandwidth-delay product, and the network applications are mainly electronic mails and file transfers. It is unrealistic to expect the existing schemes to support efficient communication in the new environment.

We have developed a flow control scheme that makes efficient use of the underlying high-speed networks to satisfy the communication requirements of applications. The design, evaluation, and implementation of the flow control scheme are the main subjects of this paper. The rest of this paper is organized as follows: Section 2 reviews the important issues and existing solutions for flow control; Section 3 describes in detail the two-level flow control scheme; Section 4 examines several important performance questions regarding the end-to-end flow control and presents the analysis and simulation results; Some implementation results are presented in Section 5; Finally, a summary is presented in Section 6. A much more detailed report of the flow control scheme along with an error control scheme is available in [14].

2. OVERVIEW OF ISSUES AND SOLUTIONS

This section provides an overview of the existing flow control schemes first, followed by discussions of several important issues which should be considered when designing a flow control scheme.

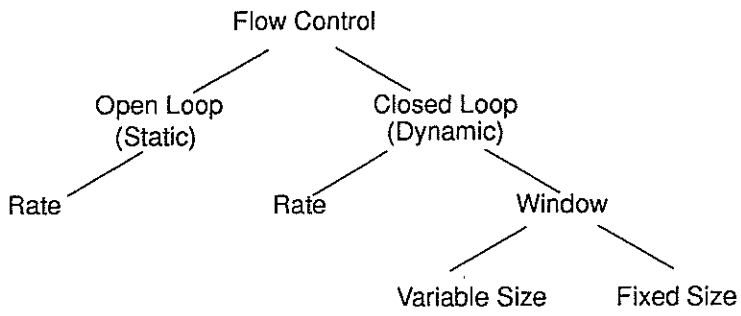


Figure 2: A Flow Control Taxonomy

2.1. Flow Control Taxonomy

The existing flow control schemes can be summarized using the taxonomy shown in Figure 2. From a control point of view, a flow control scheme can be classified as either *open loop* or *closed loop*. In an open-loop scheme, the way the flow is regulated depends only on a set of initial factors such as desired throughput and the available resources. In particular, the control function is static, and it does not change with any feedback information on the current state of the receiver or intermediate networks. A closed-loop scheme, on the other hand, has a dynamic control function that regulates the flow according to the feedback information from the receiver as well as the network nodes.

The only type of open loop scheme that has been proposed in the literature is *rate-based*. The main property of open loop rate control is that the rate is determined (possibly negotiated between end points and the underlying network) at connection setup time and remains fixed throughout the connection.

There are two different types of schemes for closed loop flow control: rate-based and window-based. The main difference between the rate control here and the one under open loop is that the rate can be dynamically adjusted according to the feedback from the receiver or the underlying networks. For example, transport protocols VMTP and NETBLT have proposed to adjust the flow rate according to the speed of an end receiver [5, 6]. In a window-based flow control scheme, the receiver specifies the maximum number of data units that the sender can transmit without being acknowledged. This maximum number of data units defines a *window*. The window is specified with reference to a sequence number space common to the sender and receiver. For example, Figure 3 illustrates a window of size W. For a sender, it means that packets with sequence numbers in the range $[n..n + W - 1]$ can be transmitted; for a receiver, it means that only packets in that range will be accepted. As data are received and consumed at the receiver, control messages are sent back to the sender to open up the window thus allowing more data to be transmitted. Even with a fixed window size, the receiver can exercise dynamic control over the sender by choosing when to advance the left edge (i.e., n in Figure 3) of the window. The window size can also be adjusted in addition to controlling the advancement of the window. This defines a window scheme with a variable window size. For example, a dynamic window scheme is used in TCP for congestion control purpose [17]. Next few paragraphs discuss several issues pertinent to end-to-end flow control.

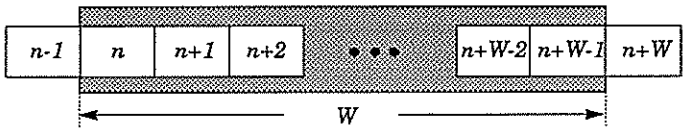


Figure 3: A Window Of Size W

2.2. Hop-to-Hop versus End-to-End Flow Control

Hop-to-hop control is clearly necessary for end-to-end communication because an end-to-end connection consists of many hops. However, end-to-end flow control function cannot be efficiently implemented by an hop-to-hop control mechanism. First of all, doing so will require each network node to keep complete and separate state information for every connection passing through it and to exchange control messages with its neighbor nodes. This represents a significant complexity increase that is detrimental to a fast

switch design. Secondly, when hop-to-hop control mechanism is used for end-to-end control purpose, throttling of the sending application is achieved by building up "back-pressure" from the receiver to the sender. That is, the receiver has to first slow down the network node directly connected to it, this node in turn slows down its neighbor, and so on until finally the sender is slowed down by its neighbor (another network node). When the main goal is speed matching between the two end applications, exchanging end-to-end control messages is more effective than relying on the "back-pressure" to build up hop by hop. Therefore, a separate end-to-end flow control mechanism should be used in addition to the hop-to-hop mechanism provided by the network.

2.3. Flow Control versus Congestion Control

Another network issue related to flow control is congestion control. Congestion is a condition under which the demand for network resources (e.g., processing cycles, buffer space, and output link bandwidth) at network nodes exceeds their capacities, resulting in a significant reduction in network throughput [18]. As pointed out by Jain in [18], congestion is affected by a wide range of control policies at the data link level, the network level, and the transport level. A successful congestion control solution is possible only if systematic design decisions are made at all these levels. Therefore, in the design of the proposed end-to-end flow control scheme, considerations will be made that will help the congestion control in underlying networks.

2.4. Rate Control versus Window Control

Rate control schemes have been widely proposed for congestion control in high speed networks. For example, the leaky bucket mechanism [1] and the virtual clock mechanism [26] are both variations of rate control. Rate control schemes have also been used for end-to-end flow control in an attempt to avoid performance problems observed with TCP's sliding window mechanism [5, 6]. With more ATM networks adopting rate-based flow and congestion control, it is also desirable to use a similar rate-based mechanism for end-to-end flow control so that direct negotiation for quality of service between the network and the end applications will be possible.

Despite much criticism, window control schemes remain the most widely used schemes for end-to-end flow control. It should be pointed out that several problems observed with TCP's windowing mechanism are not because of the windowing strategy itself, but rather due to the fact that windowing was used for

one too many functions; TCP windows are used for flow control, error control, and congestion functions. For example, TCP uses a single type of ACK packet to both acknowledge the correct reception of a data packet and to advance the sender's window. When the receiver tries to delay an ACK in an attempt to slow down the sender, an ACK timeout may occur and thus cause a false retransmission. Moreover, allowing each individual sender to modify its window in response to ACK delay has been reported to cause oscillation of load level in the network. This is a result of an improper mix of congestion control with end-to-end flow control.

Rate control alone is not adequate for efficient end-to-end flow control. The main reason is that specifying a rate cannot provide absolute protection against buffer overflow. Rate-based schemes require a "stop" signal to be delivered to the sender within a certain period of time in order to achieve this effect. On the other hand, when one uses window control alone, one is faced with a trade-off between tight control over the sender and high utilization of the connection, i.e., while a large window size is required to overcome the large bandwidth-delay product, a smaller window is needed for effective flow control [6]. In fact, Bovopoulos and Lazar have argued that, in order to derive an optimal window size for a window-based flow control, there must be a rate constraint [2]. The authors further suggested that an optimal flow control should be some combination of open loop rate control and window-based control. This lends strong support to the design of the flow control scheme to be presented in Section 3.

2.5. Data Granularity in Flow Control

Control data granularity refers to the smallest unit of data that the flow control mechanism acts upon. There are three possible levels of granularity that can be used in end-to-end flow control. They are byte, packet, and segment. Each of them is discussed next.

Byte granularity is more suitable for data communication inside a machine (e.g., for I/O) than for inter-machine communication. TCP uses byte granularity. The window size in TCP is specified in number of bytes. Since TCP/IP was designed to work assuming minimal support from the underlying networks, a transport level data block may be fragmented by the network at arbitrary byte boundaries. Byte granularity gives the maximum flexibility in fragmentation and reassembly of data blocks. This is probably the main motivation for using the byte granularity in TCP.

However, newer generations of networks all support a data unit called a "packet" with a minimum size that is typically much more than a byte. For example, a single ATM cell can carry 45 bytes of

data and with SAR protocol support, packets of up to 46080 byte each are possible over ATM networks; an Ethernet frame can contain data packets of up to 1500 bytes long. Assuming this new generation of underlying networks, the transport data granularity should at least be a packet in order to reduce the protocol processing overhead. Almost all newly proposed transport protocols use packet granularity [4, 5, 6, 20].

The third level of granularity that can be used for end-to-end flow control is a segment. A segment defines the smallest data unit that an application wishes to access independently. This implies that the buffer space for the application will be allocated and deallocated on a segment basis. Furthermore, most of the applications that demand high bandwidth communication also have very large segment sizes (kilobyte–megabyte). Therefore, using segment granularity as opposed to the packet granularity, the control overhead (including both local processing and the link bandwidth for control messages) can be reduced from a per packet basis to a per segment basis, thus increasing the efficiency of communication. For example, assuming the popular packet size of 1024 bytes and a modest segment size of 16 kilobytes, performing flow control using segment granularity reduces the processing overhead by roughly a factor of 16 compared to flow control on a per packet basis.

2.6. Flow Control Objectives

From the above discussions, the following objectives should be achieved by the new end-to-end flow control scheme:

- In order to receive performance guarantees by the network, the proposed scheme should control its data flow into the underlying network such that the rate agreement is not violated.
- For every pair of communicating applications, the sender should be throttled if necessary to avoid overflow in the receiver. This is a critical requirement for distributed pipeline applications.
- As far as possible, adverse interactions between flow control, error control, and congestion control functions should be avoided.
- Large control granularity should be used when possible to reduce the control overhead.

The proposed end-to-end flow control scheme is presented in the next section.

3. PROPOSED SOLUTION

This section first introduces the overall structure of the two-level flow control scheme and presents the rationale for using two levels of control. Then, formats of the packets used in the data transfer phase are described. Finally, it presents each level of the flow control scheme in detail.

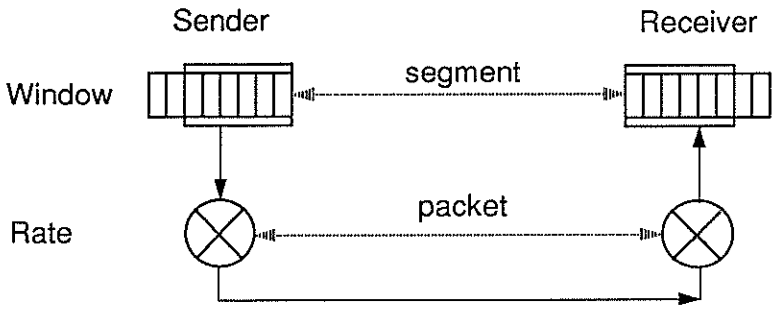


Figure 4: Two-Level Flow Control Structure

3.1. Two-Level Structure

Figure 4 illustrates flow control mechanisms as they reside in a sender and a receiver. The data flows from the sender to the receiver. At the lower level there is a rate control mechanism that regulates the data flow into the underlying network according to a rate specification. At a higher level, a simple window mechanism resolves whether a data segment is eligible for transmission through the rate control. As indicated by the dashed lines, rate control is on a packet basis while the window control uses segment granularity.

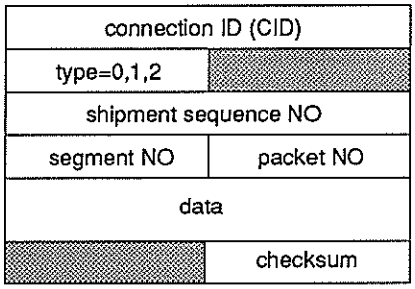
There are three important features of the scheme that should be emphasized:

Two Levels Of Control: A rate control is used in concert with a window control. The rate control ensures that the traffic generated by the local application conforms to a negotiated rate specification. Thus, the rate control makes the traffic into the underlying network more predictable so that the network can employ more efficient congestion control strategies. The window mechanism allows a receiver to control the speed at which the sender is transmitting data segments, according to the receiver's speed and buffer availability.

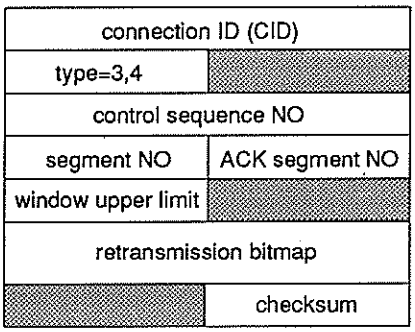
Two Levels Of Granularity: The rate control is at the packet level which ensures a finer granularity of control for data flowing into the network. The window control uses segment granularity. The larger segment granularity reduces the overall control overhead.

Avoidance Of Adverse Interaction: The proposed scheme uses special control messages for window advancement. These messages are different from the acknowledgments for reception of data segments. Therefore, even when the receiver needs to slow down the sender, it can still continue to send acknowledgments as necessary so that the sender will not generate false retransmission by confusing a "slow-down" request as a packet loss timeout, which is the case with TCP.

The rate control and window control are discussed in detail next.



(a) sender to receiver



(b) receiver to sender

Figure 5: Error Control Packet Formats

3.2. 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 5. The width of each row is assumed to be 4 bytes.

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's 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 in earlier segment transmissions. 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 a 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 (positive ACK) 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. The control sequence NO is used by the sender to detect and discard duplicate PACK packets.

SNAK Packet (type 4): A SNAK (selective negative ACK) 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 represents retransmission requests for those packets where there is a "0" in the corresponding position of the bitmap.

3.3. Rate Control

The proposed rate specification and the rate control operations are described in this section.

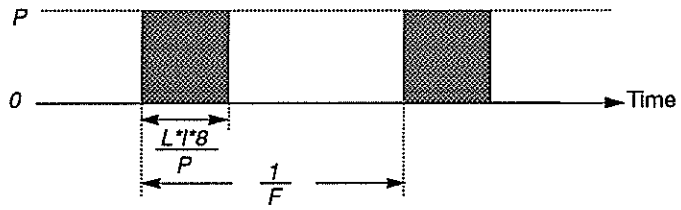


Figure 6: Bursty Rate Specification

Rate Specification

There are several rate specification schemes proposed in the literature. For example, an average rate calculated on a given time interval [26] or a peak rate, an average rate, and a burst factor [1]. The proposed scheme uses a simple bursty rate specification. Let l be the packet size in bytes and L be the burst length in packets. Assume that the peak rate of transmission is P bits/s and the desired burst rate is F bursts/s. Then, the rate specification R consists of three parameters as follows:

$$R = (P, F, L) \quad where\, F \leq \frac{P}{L \times l \times 8}$$

The average rate in correspondence with the bursty rate R is $A = F \times L \times l \times 8$. Figure 6 depicts this bursty rate specification in time domain.

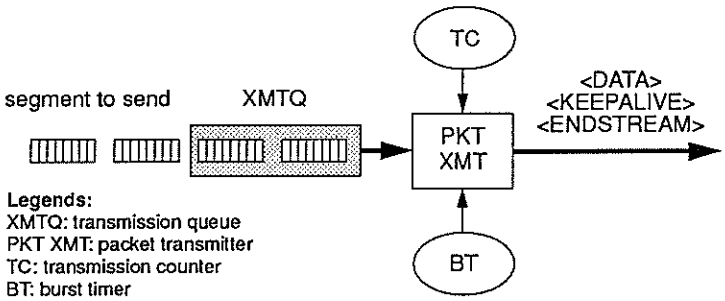


Figure 7: Rate Control Logic

Rate Control Operation

The main function of the rate control mechanism is to clock data packets out according to the rate specification. Figure 7 shows a detailed view of the rate control logic at the sender. The rate control mechanism consists of a burst timer (BT) with period $1/F$ and a transmission counter (TC) for controlling the burst length. The transmission queue XMTQ holds all packets eligible for transmission. The operation of the rate control is specified by the following simple steps:

Upon BT timeout:

1. reset timer $BT = 1/F$.
2. set $TC = 0$.
3. if $TC < L$ and XMTQ not empty,
   remove and transmit the first packet from the queue and increment TC;
   else if $TC < L$ and XMTQ is empty,
   transmit a keep-alive packet or an end-of-stream packet and then return;
   otherwise return.
4. go back to step 3.

Selecting Burst Length

There are two strategies for selecting the burst length L. First, assuming that the network loss is very low and that there will be very few retransmissions necessary, the burst length should be chosen to be the size of a segment ($L = s$). The bursty rate control then matches the way a segment is produced and consumed by application processes. For example, consider an intermediate stage of a distributed computing pipeline. When the stage first receives a data segment, it has to spend a certain amount of time processing the segment; meanwhile, there is nothing available for transmission yet; once it finishes processing the segment, the whole segment is ready to be transmitted to the next stage as a burst; the stage receives the next segment and the cycle goes on. The bursty transmission allows segments to be delivered as fast as possible when ready. This gives maximum flexibility for application processes to slow down or catch up, thus allowing efficient operation the pipeline.

However, if the expected number of packet retransmissions in each segment is not negligible, the burst length should be chosen to account for the expected retransmissions in order to meet the throughput requirement for the application. For example, if the expected number of packet retransmissions per segment is X, the burst length should be $L = s + X$. This way, a new segment can still be delivered in one burst even with X packet retransmissions of a previous segment, thus maintaining the desired throughput of F segments/second for the application.

3.4. Window Control

This section presents the window control scheme in detail. Specifically, it discusses how to determine the initial window sizes and what strategies to use for window control, and describes operations of the windowing mechanism.

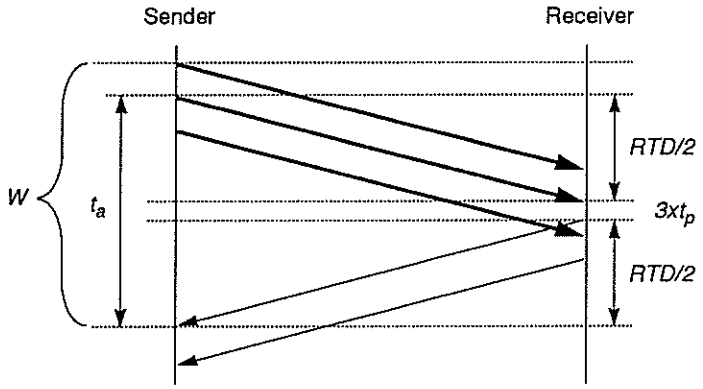


Figure 8: Acknowledgment Delay Diagram

Window Size Determination

Let the round trip delay on the connection be RTD and each segment consist of s packets. Also recall from Section 3.3 that each packet is l bytes long, and the peak transmission rate is P. Then, $t_p = l \times 8/P$ is the time for transmitting one packet. If the application desires a throughput of F segments per second, the corresponding average bit rate is $A = F \times L \times l \times 8$, assuming bursty rate control as described earlier.

From Figure 8, it is seen that the time elapsed from the moment a segment is transmitted till a window advancement message returns to the sender is $t_a = RTD + 3 \times t_p$, assuming that the overhead for the receiver to send an advancement message is approximately $3 \times t_p$. The equivalent number of segments that can be transmitted during this time is:

$$w_a = \frac{A \times t_a}{s \times l \times 8}$$

Therefore, even if one assumes there is no packet loss or corruption in the network, the initial window size W_0 has to be at least $(1 + w_a)$ in order to achieve the desired throughput. If the window size is less than this, the rate mechanism will be forced to be idle some of the time. Considering the possible loss and corruption in the network, W_0 will have to be much larger than $(1 + w_a)$ in practice. More detailed discussion of this issue will be provided in Section 4.

Window Control Strategy

Depending on how long the receiver is expected to slow down, it can choose to throttle the sender by delaying the advancement of the sender's window or by reducing the window size, or both.

In case there is a temporary delay in the receiver, for example, due to contention at a file server, the receiver only needs to withhold the window advancement message until the delay is over. On the other hand, if the receiver expects to have a long period (e.g., more than several RTD) of slow-down due to additional load on the system, it can reduce the window size to some value $W_1 < W_0$. Assume that the expected slower throughput is F_1 and $F_1 < F$, the new window size W_1 can be computed as:

$$W_1 = \frac{F_1}{F} \times W_0$$

This same adjustment formula can be used to increase the window size should the processor recovers from the slow-down, the only difference being that the new throughput F_1 is higher than the old throughput F.

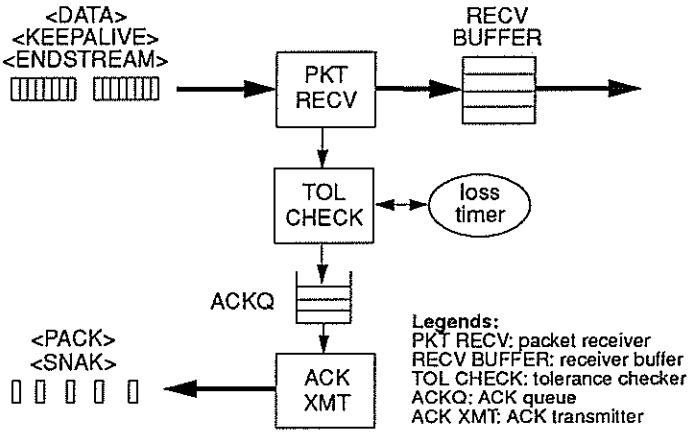


Figure 9: Receiver Logic – Window Control

Receiver Window Operations

There are window control functions performed at both the sender and receiver. We describe the receiver operations first because the receiver is actively controlling the window. The receiver function determines the position of the window in the segment sequence space and the size of the window. Figure 9 displays the receiver logic. The window control decision is made by the tolerance checker (TOL CHECK). The tolerance checker compares the packet loss in a segment against a loss tolerance specification for

the application; if the tolerance is not satisfied, a retransmission request bitmap will be generated for requesting retransmission from the sender; if the tolerance is satisfied, the segment will be accepted and the receiving window will be updated as described in the next paragraph.

There are three key variables for window control at the receiver: R_l the lowest segment number within the receiving window, R_h the highest segment number within the receiving window, and W the current window size. Only packets with segment numbers in the range $[R_l, R_h]$ can be accepted by packet receiver (PKT RECV). Let i and j be variables representing segment sequence numbers. The receiving window is updated as follows:

1. Adjust the window size W if there is a long-term change of processing speed in receiving application.
2. If for some segment i ($R_l \leq i \leq R_h$) such that all segments j ($R_l \leq j \leq i$) are successfully accepted by the application,
   set $R_l = i + 1$ and $R_h = \max(R_h, R_l + W - 1)$;
   otherwise return.

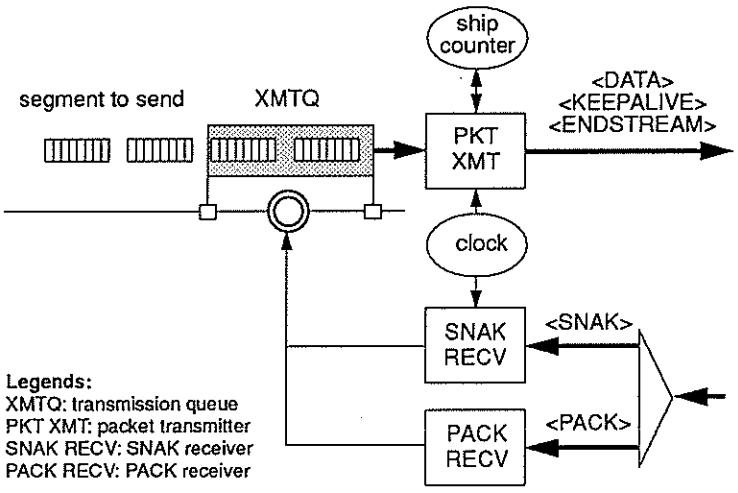


Figure 10: Sender Logic – Window Control

Sender Window Operations

The window control function at the sender keeps track of the receiver's window advancement messages and updates the window at the sender as necessary. This function is carried out by both the PACK receive logic and the SNAK receive logic at the sender, as shown in Figure 10. Two variables, S_l and S_h, specify

the lower and upper edge of the sending window. Only packets of the segments with segment numbers in the range $[S_l, S_h]$ are eligible for transmission by the rate-controlled transmission mechanism.

There are two fields, the ACK segment NO and the window upper limit in a PACK or SNAK packet that are used for window control (refer to Figure 5). Whenever a PACK or SNAK packet is to be sent to the sender from the receiver, the current R_l value is copied into the ACK segment NO field and the R_h value into the window upper limit field.

When a PACK or SNAK packet correctly arrives at the sender the first time, the corresponding logic (PACK RECV or SNAK RECV) will release all the positively acknowledged segments from its buffer, and then do the following two steps for window control:

1. set $S_l = \max(S_l,$ ACK_segment_NO).
2. set $S_h = \max(S_h,$ window_upper_limit).

4. PERFORMANCE STUDY

Given the design of the two-level flow control scheme, two important performance questions need to be examined. (1) How does the window control perform given that the rate is guaranteed? (2) Is it necessary for the size of the receiver's buffer to be as large as the window? These questions are explored in this section. But first, we define a set of performance measures to be used in this study.

4.1. Performance Measures

From applications' point of view, the two most important performance measures of a transport protocol are *throughput* and *delay*. Given two communicating processes p_1 and p_2, throughput from p_1 to p_2 is traditionally defined as the number of data segments transferred from p_1 to p_2 in one unit time. Clearly, this definition is inherently an average measure, and it is dependent on the application behavior. For example, the same protocol mechanism may produce different throughput on different occasions when the application process changes its speed from time to time. 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 that is 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.

Transport protocol deals with end-to-end delay as opposed to the hop-to-hop delay that concerns network level protocols. A transport level connection may be supported by many network hops consisting of packet switches and gateways. Without loss of generality, a one-way data flow from a sender to a receiver is considered here. 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*. For example, if the segment transmitting time is t_s and the round trip delay on the connection is RTD, the ideal end-to-end delay for the segment will be $(t_s + RTD/2)$, assuming propagation delays in both directions are the same on the connection. It should be noted that the above delay definition is from the receiver's point of view. If it were defined from the sender's point of view, the ideal end-to-end delay for the example would be $(t_s + RTD)$. The former definition is chosen because it reflects exactly when the segment is available for consumption at the receiver.

From the above basic definition of end-to-end delay, two commonly used delay measures can be defined. *Average End-to-End Delay* is the average of end-to-end delays over all the segments. *Maximum End-to-End Delay* is the maximum of end-to-end delays among all the segments. While the average characterizes the overall delay performance, the maximum gauges the worst case behavior that is very important to many applications with strict time constraint.

4.2. Evaluation Approaches

Both analysis and simulation are used in the study of the flow control scheme. Due to the complexity of the flow control process, only approximate analyses for average delay and throughput efficiency are pursued. The purposes of the analyses are: first, to gain insights through the derivation of simple performance expressions; and second, to verify the simulation model. Discrete event simulation is used for more accurate study of the various aspects of the flow control.

Delay Analysis

Given the average end-to-end delay definition in Section 4.1, it can be expected to be rather insensitive 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 and also possibly retransmission, 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 packet 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.
- Application requires 100% reliable delivery of all segments.

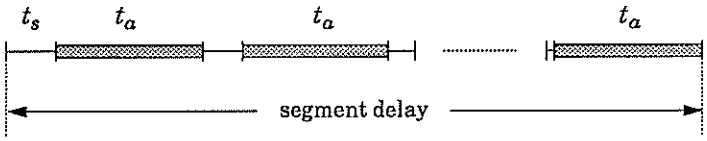


Figure 11: A General Segment Delivery Scenario

Figure 11 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:

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

[‡]A packet group refers to all the packets associated with a segment. For example, a segment is itself a packet group when it is first transmitted. During a retransmission all packets to be retransmitted for a segment define a new packet group.

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 in Section 4.1 from the receiver's view point, 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 4.3. It should be mentioned that a similar analysis has been presented in [19] 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, and the current derivation is simpler.

Throughput Efficiency Analysis

Accurate throughput analysis for selective retransmission scheme with large windows is a very difficult task. The difficulty stems mainly from the complex overlap and interaction between transmission of new segments and retransmission of lost packets. In general, full analysis of error control schemes using selective ACK is lacking in the literature. Existing studies [19, 27] 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 [20] and for analysis [11]), and it is the main inspiration for the throughput analysis to be presented. However, there are several significant differences between the SNR error control scheme and the proposed scheme that require different treatment in the throughput analysis.

The following assumptions are used in the analysis:

- A packet is lost if it is either corrupted or dropped in the underlying network. Each packet 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, where $t_a = RTD + 3 \times t_p$. The delay for the receiver to send out an acknowledgment after receiving the last packet of a segment is assumed to be $3 \times t_p$.
- Application requires 100% reliable delivery of all segments.
- The probability $(1-(1-p^2)^s)$ of requiring more than one retransmission for each packet is negligible.

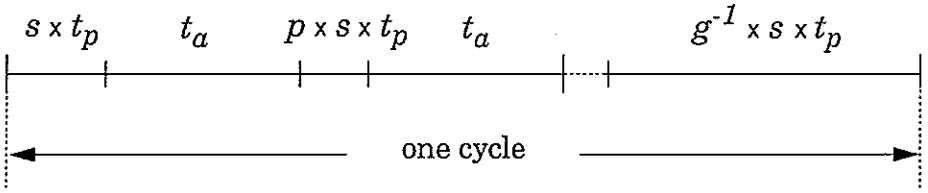


Figure 12: A Regenerative Cycle

Let $g = [1 - (1 - p)^s]$ denote the probability that a segment will require at least one packet retransmission. Activities on the connection follow regenerative cycles shown in Figure 12. Each time period, from left to right, as marked by vertical bars in the cycle is explained as follows:

1. The beginning of a cycle is marked by the transmission of a new segment which will have some packets lost.
2. After t_a time, a retransmission request (i.e., SNAK) comes back and $(p \times s)$ lost packets are retransmitted.
3. According to the assumptions, all retransmissions will be successful and a positive acknowledgment (ACK) will be received in another t_a time; Depending on the actual size of the window, the two corresponding time periods (labeled as t_a in the figure) for acknowledgment may be utilized for transmitting new segments or left idle.
4. If more new segments are transmitted during the waiting periods for acknowledgment, some of the lost packets may have to be retransmitted in the period marked by the dotted line.
5. After recovery from the earlier losses, approximately g^{-1} segments can be delivered without any loss, until the next packet loss which will start the next cycle.

Once the regenerative cycle is identified, throughput efficiency can equivalently be defined as the ratio of the actual number of segments delivered in one cycle, to the ideal number of segments that could be delivered if there were no losses and the window size were infinite.

Let W be the number of segments in a window and $w_a = t_a/t_s$. Also, let ThE be the throughput efficiency. An expression for ThE with a window size W such that $1 < W \leq (w_a + 1)$ is derived first. 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 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 total time for g^{-1} segments is $\frac{(w_a+1)}{W}g^{-1}t_s$. Ideally, exactly $\frac{(w_a+1)}{W}g^{-1}$ segments could 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 with the scheme in one cycle and the denominator is the number of segments deliverable under the 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 from Section 4.2:

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

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

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

Numerical results obtained using these expressions will be shown in Section 4.3.

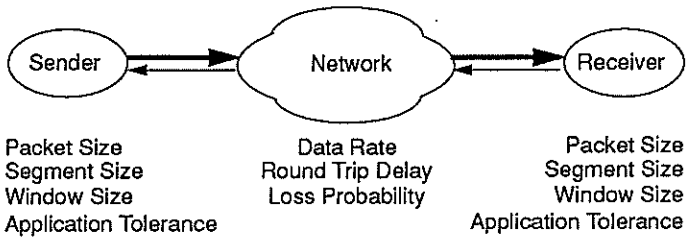


Figure 13: Simulation Configuration

Discrete Event Simulation

There are many aspects of flow control that require a detailed simulation study. For example, the effect of significant packet losses and the consequence of advertising windows larger than buffer sizes. A set of discrete event simulation programs has been developed for this purpose. The simulation configuration is shown in Figure 13.

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. However, these effects can be minimized by transmitting a large amount of data ($\geq 10^7$ packets) and by assuming very fast processors with large memories.

All aspects of the proposed flow control scheme are modeled in the simulation except for the following simplifying assumptions:

- 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 and none of the control packets will ever be corrupted or lost.
- 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, where $t_a = RTD + 3 \times t_p$.

4.3. Numerical Results

This section presents and discusses a selected set of results from the simulation study. More detailed results have been reported in [14].

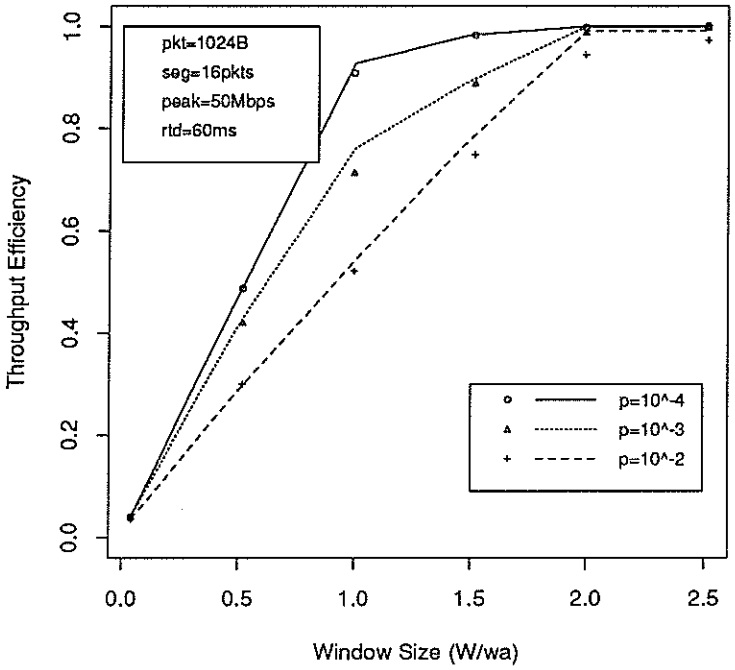


Figure 14: Throughput Efficiency – Analysis vs. Simulation

Window Size Requirement

Figure 14 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 considering the very high bandwidth achieved.

- The throughput efficiency expressions derived in Section 4.2 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.

Buffer Requirement

In an end-to-end communication, the sending buffer size always has to be at least as large as the window size because there can only be as many outstanding segments as there are sending segment buffers. The receiving buffer, on the other hand, can have a different size than the end-to-end window size. In particular, it is desirable to use a smaller receiving buffer if possible due to the constraint of physical memory sizes. For example, given a transcontinental connection with a bandwidth of 1 Gbps and a round trip delay of 100 ms, the bandwidth-delay product is 1 Gbps $\times$100 ms = 12.5 megabyte. According to the result above, a window size of about $2.5 \times 12.5 = 31.25$ megabytes is needed in order to achieve very high throughput. Even by today's standards this is a significant memory requirement for a workstation.

Indeed, under error-free conditions, the receiving buffer requirement is usually much smaller than the window size. Assume that the application reserves a buffer area for communication protocols to directly put new data in and for itself to do the processing. Then, two segment slots are sufficient for continuous receiving and processing operation. When a segment is received in one buffer slot, the application works on the segment while the protocol receives the next segment to the second buffer slot. The receiving and the processing finish at the same time and the buffer slots are switched, the cycle repeats. However, when there are errors or when the speeds of the protocol and the application do not match perfectly, data segments may have to be dropped or overwritten if no additional buffer slots are provided. An interesting question is: how effective is it to advertise larger windows for achieving higher performance? Two different acknowledgment strategies can be used for operating with over-advertised windows.

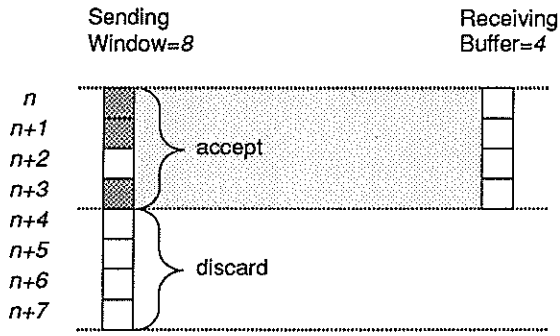


Figure 15: Hard ACK Strategy

4.3.1. Hard ACK and Soft ACK Strategies

Hard ACK and soft ACK are two types of acknowledgment strategies that can be used when the receiver is advertising a window larger than the actual receiving buffer available. These strategies have been considered for TCP extensions but were only briefly discussed on the end-to-end communications mailing list recently. To this date, there has been no performance study done to evaluate the effectiveness of these strategies, to the best of the author's knowledge. This research need is addressed through simulation studies in this dissertation. The hard ACK and soft ACK strategies will be defined first and all results should be interpreted with respect to these definitions.

Let $(n, n+1, \ldots n+W-1)$ be the current window of size W that the receiver has advertised to the sender. Let B be the actual buffer size and $B < W$. Figure 15 illustrates how *hard* ACK strategy works with $W = 8$ and $B = 4$. It is seen that the receiver simply discards any segments with segment numbers outside the range $[n..n+3]$. All segments accepted can be acknowledged with certainty.

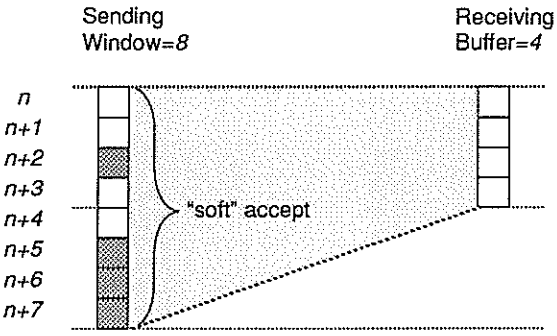


Figure 16: Soft ACK Strategy

The *soft* ACK strategy, however, will save a segment into the receiving buffer as long as it is inside the current window $[n, n+1, \ldots n+7]$, and there is space available for the segment, as shown in Figure 16.

Because of retransmission and resequencing inside networks, a scenario like the one shown in the figure can occur. In this case, segments $n+2$, $n+5$, $n+6$, and $n+7$ arrived at the receiver before segments n, $n+1$, and $n+3$, and filled up the buffer. In order to deliver the next contiguous block of data to the application, some of the saved segments (e.g., $n+5$, $n+6$, and $n+7$) will have to be discarded to make room for segments n, $n+1$, and $n+3$. This means that when those out-of-order segments are received, they could only be acknowledged with a special message indicating their successful arrival at the receiver but the sender should not release those segments until further acknowledgment is received. The name soft ACK reflects exactly this special requirement.

It is clear that the soft ACK strategy requires proper specification of a *replacement policy* in order to be used. Specifically, when a segment needs to be discarded from the receiving buffer and there are multiple segments to choose from, how does one decide which segment to discard? Recall that the receiving buffer contains multiple segments which consist of many packets each. The following soft ACK algorithm is used in this study:

Upon successfully receiving a packet with segment number N_{in}:

1. If the packet belongs to a segment already in the buffer, accept it.
2. If the packet is new and there is free segment slot, accept the packet and reserve a slot.
3. If the packet is new and the buffer is all reserved, find the segment with the largest sequence number N_{old} in the buffer; if $N_{in} < N_{old}$ then discard segment N_{old} from the buffer and save the new packet in its slot; otherwise, discard the new packet.

Three simulation experiments have been conducted to achieve two goals: (1) to quantify the advantage of advertising larger windows and (2) to compare quantitatively the hard ACK and soft ACK strategies. The simulation configuration is the same as the one used in error control evaluation (see Section 4.2).

In the first experiment, the end-to-end window was set to be the same size as the receiving buffer all the time. The system was simulated for buffer sizes of $0.25, 0.5, 1.0$, and 2.5 times the bandwidth-delay product w_a. For the second experiment, the simulator implemented the hard ACK strategy and the same set of buffer sizes were simulated, except that the end-to-end window was fixed at $2.5 \times w_a$. The third experiment repeated the set of conditions for experiment 2 but with the soft ACK strategy. In all three experiments, reliable simulation results were obtained by simulating 10 million packets. The results of these experiments are summarized in the next few paragraphs.

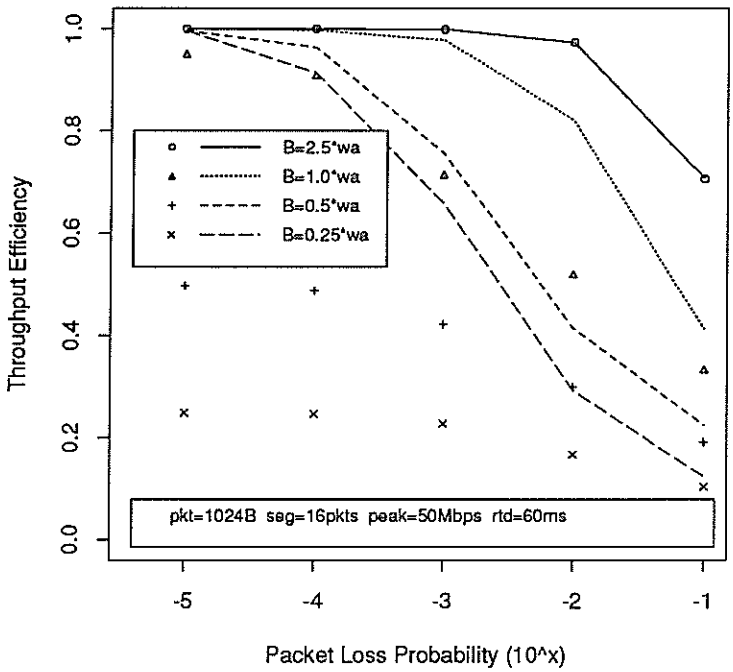


Figure 17: Advertising Larger Windows

4.3.2. Larger Window Advantage

Figure 17 compares the throughput efficiencies achieved when the end-to-end window size is the same as the receiving buffer size and when larger windows are used. The results from advertising larger windows are plotted in different line styles, while discrete points represent the results achieved when window sizes were kept the same as the receiving buffer sizes. In the case of advertising larger windows, the strategy used is hard ACK and the window is fixed at $2.5 \times w_a$. The vertical axis represents the throughput efficiency and the horizontal axis is the logarithmic value of random packet loss probability. Clearly, by advertising windows larger than the actual buffer size, significantly higher throughput is achieved. This holds for the whole range of loss probabilities studied and for buffer sizes as small as 1/4 of the bandwidth-delay product. In particular, over 70% higher throughput is achieved by advertising the large window for a buffer size of $0.25 \times w_a$ when loss probability is 10^{-5}.

The key advantage of advertising larger windows is that it allows the sender to continue transmitting data to fully utilize the connection. Although a segment may be dropped when it finds no buffer space when arriving at the receiver, such an event is not expected to occur very often because data will normally arrive successfully in sequence at the receiver and get consumed immediately, thus freeing more buffer space.

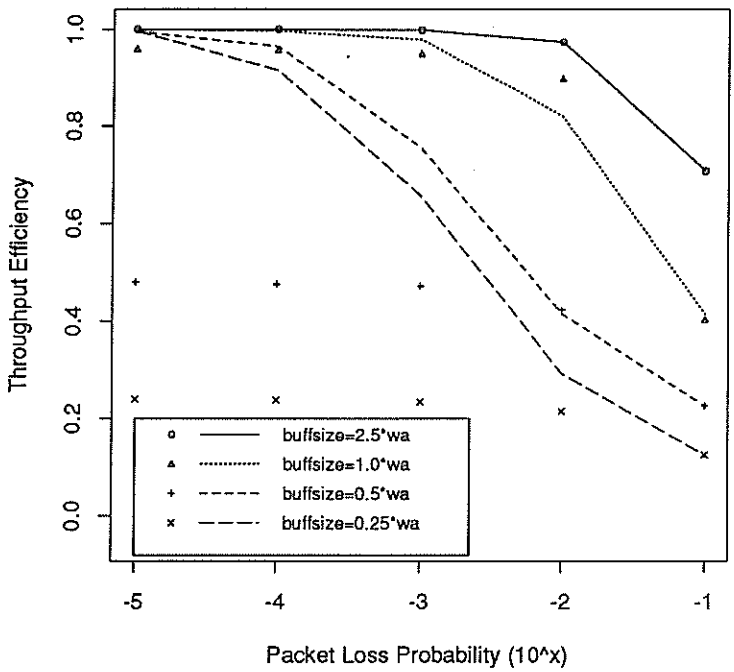


Figure 18: Hard ACK versus Soft ACK - Throughput

4.3.3. Hard ACK versus Soft ACK Performance

Plots in Figure 18 and 19 show comparisons between the hard ACK strategy and the soft ACK strategy. Figure 18 compares the throughput performance and Figure 19 shows the maximum segment delay. At all times, the end-to-end window size is fixed at $2.5 \times w_a$ and only buffer sizes are varied. The results from the hard ACK strategy are plotted using lines and the soft ACK results are shown as discrete symbols.

From Figure 18 it is seen that when the buffer size is equal to the window size $2.5 \times w_a$, the two strategies perform exactly the same (the solid line and the circles match) because there is no discard of segments at the receiver. But as the actual buffer size decreases, performance of the soft ACK strategy deteriorates dramatically. The most significant drop in throughput occurs when the buffer size is reduced to $0.5 \times w_a$. The performance difference between the two strategies remains significant until the loss probability approaches 10^{-1}, by that time the hard ACK performs as bad as the soft ACK due to the tremendous loss.

With the maximum segment delay (Figure 19), the hard ACK strategy also outperforms the soft ACK strategy for loss probabilities up to 10^{-2}. Again, the two strategies perform identically when the receiving buffer is as large as the end-to-end window, just as expected. However, the soft ACK strategy

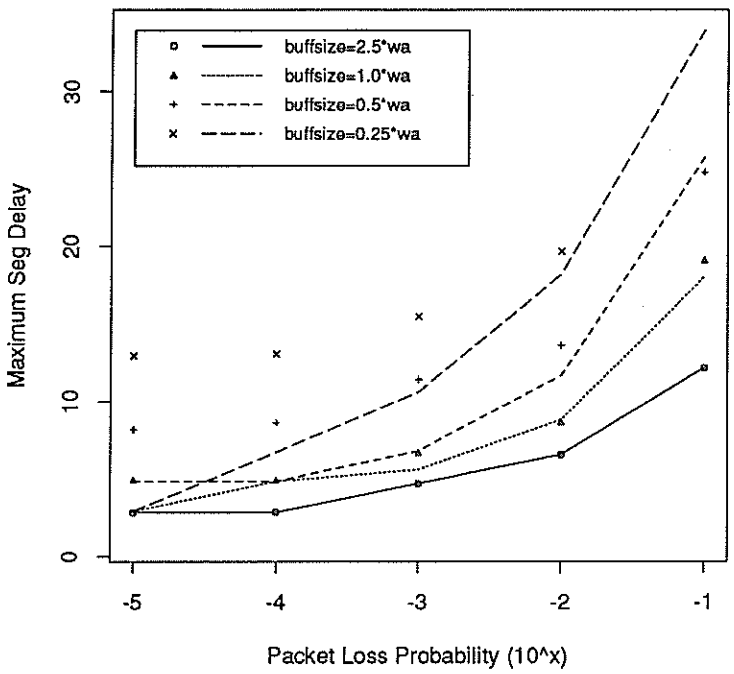


Figure 19: Hard ACK versus Soft ACK - Maximum Delay

seems to give lower maximum delay when the loss probability is very high, as seen for the loss probability of 10^{-1} in the plot. It should be noted that with the loss probability of 10^{-5}, the plot shows a higher maximum delay for buffer size $0.5 \times w_a$ than for buffer size $0.25 \times w_a$. This is believed to be an artifact of insufficient simulation time for this extremely low loss case.

Why is the soft ACK strategy performing worse than the hard ACK? A careful examination of the soft ACK operation revealed the following behavior:

> When the window size is larger than the receiving buffer size, a packet loss or corruption in the network can lead soft ACK into a state in which, periodically a number of segments will be first saved into the buffer but only to be replaced (thus discarded) later by segments with smaller sequence numbers. This state will persist even if there is no more loss in the network.

Note that this behavior not only causes spurious retransmissions, but also delays retransmission requests. Therefore, it leads to lower throughput efficiency and longer delay. This abnormal behavior also explains why the soft ACK performance is poor even with very low loss probability.

Intuitively, if the buffer size is very close to the window size, the soft ACK could possibly offer a better performance than the hard ACK strategy. But this margin seems to be so narrow that it is not visible within the range of parameters simulated. It is possible that by using a more elaborate policy,

the abnormal behavior can be avoided and a better performance may be achieved using the soft ACK strategy. However, it is doubtful that more complex policies will allow the soft ACK to achieve much better performance than the hard ACK strategy.

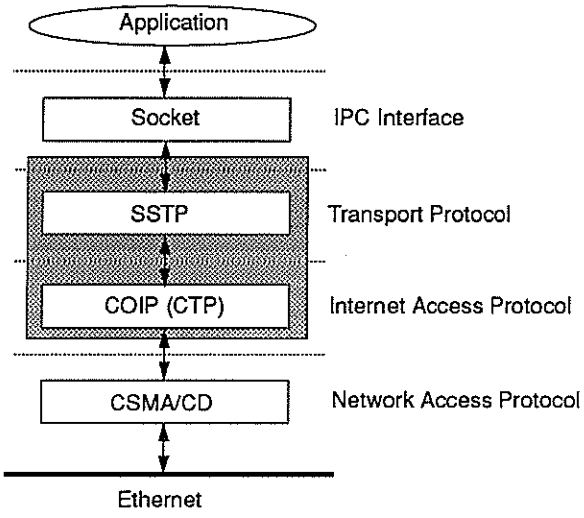


Figure 20: Protocol Hierarchy

5. IMPLEMENTATION

The proposed flow control scheme has been implemented along with an error control scheme as part of a segment streaming transport protocol (SSTP) inside the SunOS 4.0.3 kernel. SSTP is built on top of a connection-oriented internet protocol (COIP) designed by Cranor [8]. Figure 20 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; and CTP is supported by the Ethernet protocol CSMA/CD.

Extensive trace data has been collected that verified the flow control function. We also measured the throughput performance of the SSTP implementation using both custom software and a kernel probe technique developed by Papadopoulos [22]. The protocol processing delay results are summarized in Table 1. 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 1: 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. Since the control mechanisms of SSTP have been designed to maintain efficient operation even in large bandwidth-delay product networks, SSTP is expected to perform much better than TCP in high-speed network environments. More detailed results of the implementation are reported in [14].

6. SUMMARY

This paper has presented a two-level flow control scheme that can provide efficient support for new distributed scientific computing applications that exchange a stream of data segments in high speed networks. The rate control ensures that the data sources (application processes) do not use more bandwidth than requested at the connection setup. This helps the underlying network to perform effective resource management and provide guaranteed services for applications. The window control provides end-to-end speed matching at the segment level and it is decoupled from the congestion control complication. The window advancement message is also separated from the error control acknowledgment to avoid interference with one another.

This paper also explored in detail a number of performance questions about the two-level scheme. Using the loss-load model, the importance of a rate control mechanism at the transport level has been demonstrated. The size requirements of the end-to-end window as well as the receiving buffer have been studied with analysis and extensive simulations. The results suggest: (1) with the two-level scheme an end-to-end window size of 2.5 times the bandwidth-delay product is sufficient for achieving nearly ideal throughput efficiency; (2) the receiving buffer size does not need to match the window size to attain

very high performance; (3) the hard ACK strategy is far superior to the soft ACK strategy not only in performance, but also in the implementation complexity. Our measurement results show that even the primitive SSTP/COIP implementation performs significantly better than the well-crafted TCP/IP protocol.

References

[1] Akhtar, Shahid, *Congestion Control in a Fast Packet Switching Network*, Wash. U. CS Dept., M.S. thesis, St. Louis, Dec. 1987.

[2] Bovopoulos, Andreas D. and Aurel A. Lazar, "The Effect of Delayed Feedback Information on Network Performance", *Annals of Operations Research* 36 (1992) 101–124.

[3] 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.

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

[5] 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.

[6] Clark, David D., Mark L. Lambert, and LiXia Zhang, "NETBLT: A High Throughput Transport Protocol", *SIGCOMM '87 Symposium: Frontiers in Computer Communications Technology (Computer Communication Review)*, Vol. 17, No. 5, ACM, New York, 1987, pp. 353–359.

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

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

[9] Cranor, Charles D. and Gurudatta Parulkar, "An Implementation Model for Connection-Oriented Internet Protocols", to appear in Proceedings of IEEE Infocom '93.

[10] 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.

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

[12] 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.

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

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

[15] Gong, Fengmin and Gurudatta Parulkar, *An Application-Oriented Error Control Scheme For High Speed Networks*, Washington University Computer Science Department, techical report WUCS-92-37, St. Louis, November 1992.

[16] 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.

[17] Jacobson, V., "Congestion Avoidance and Control", *ACM SIGCOMM '88 Symp.*, Stanford, CA, Aug. 16–19, 1988, pp. 314–329.

[18] Jain, Raj, "Congestion Control in Computer Networks: Issues and Trends", *IEEE Network Magazine*, May 1990, pp. 24–30.

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

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

[21] 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.

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

[23] Papadopoulos, Christos and Gurudatta Parulkar, "Experimental Evaluation of SunOS and TCP/IP Protocol Implementation", to appear in Proceedings of IEEE Infocom '93.

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

[25] Williamson, Carey L. and David R. Cheriton, "Loss-Load Curves: Support for Rate-Based Congestion Control in High-Speed Datagram Networks", *Proc. ACM SIGCOMM '91*, Zurich, Switzerland, Sept. 3-6, 1991, pp. 17–28.

[26] Zhang, LiXia, "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks", *Proc. ACM SIGCOMM '90*, Philadelphia, PA, Sept. 1990, pp. 19–29.

[27] 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.