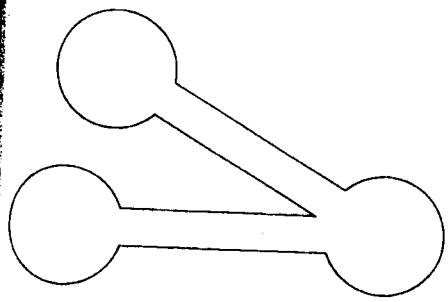
Mechanisms for a Reliable Timer-Based Protocol

John G. Fletcher and Richard W. Watson
Lawrence Livermore Laboratory, Livermore, California 94550, USA

Timer-based protocol mechanisms are developed for reliable and efficient transmission of both single-message and message-stream traffic. That is, correct data delivery is assured in the face of lost, damaged, duplicate, and out-of-sequence packets. The protocol mechanisms seem particularly useful in a high-speed local network environment. Current reliable protocol design approaches are not well suited for single-message modes of communication appropriate, for example, to distributed network operating systems. The timer intervals that must be maintained for sender and receiver are developed along with the rules for timer operation, packet acceptance, and connection opening and closing. The underlying assumptions about network characteristics required for the timer-based approach to work correctly are discussed, particularly that maximum packet lifetime can be bounded. The timer-based mechanisms are compared with mechanisms designed to deal with the same problems using the exchange of multiple messages to open and close logical connections or virtual circuits.

Keywords: Computer network, transport protocol, interprocess communication, host-to-host protocol, end-to-end protocol, timer protocol, connections, connection management, reliable communication, transaction communication, maximum packet lifetime, 3 way handshake, packet switching, network operating system



[1] This paper has been presented at the Computer Network Protocols Symposium, held in Liege (Belgium) in February 1978 and organized by the University of Liege. The permission to reprint this paper is gratefully acknowledged.

Computer Networks 2 (1978) 271–290

1. Introduction

This article discusses a number of important problems in the design of a reliable end-to-end (process-to-process) communication protocol (EEP) for a packet-switched network. Timer-based mechanisms are presented that appear to offer an efficient and effective solution to these problems.

One can usefully view the design of an EEP as consisting of three main problem areas, and one

John G. Fletcher was trained as a physicist specializing in general relativity. He received a B.S. from the George Washington University and an M.A. and Ph.D. from Princeton University. He has spent most of his working life at the Lawrence Livermore Laboratory, having joined the staff in 1962. About a dozen years ago his interests changed from physics to computer science. He has been one of the chief architects of the Laboratory's Octopus computer network and is currently a project leader working on Octopus. He also teaches computer science at the University of California, Davis.

Richard W. Watson received his B.S. from Princeton University in 1959, and his M.S. and Ph.D. from the University of California, Berkeley in 1962, and 1965. All degrees were in Electrical Engineering and Computer Science. During the years 1964–1966, he was a member of Stanford University's Computer Science Department. He has been affiliated with the University of California (Berkeley, Davis) as a lecturer part time since. From 1966–1971 he was associated with Shell Development Co. as Supervisor of Computer Science Research. From 1971–1976 he was with Stanford Research Institute's Augmentation Research Center as Assistant Director for Development. He is currently with the Computation Department at Lawrence Livermore Laboratory. His areas of research and development have involved graphics and man-machine interface, operating systems, and computer networking. He is author of the book *Timesharing System Design Concepts.*

We wish to acknowledge the useful discussions with V.G. Cerf, Y. Dalal, J.E. Donnelley, L.L. Garlick, R. Rom and J.B. Postel. We also wish to thank L.J. Sloan for his careful reading of the paper and valuable suggestions. The work reported here was sponsored by the United States Department of Energy, under contract No. W-7405-Eng-48.

would like to find as small and as simple a set of mechanisms as possible to solve the problems in all three classes. The areas are:

- Routing and interfacing: Issues in this area include addressing, routing, packet fragmentation at internetwork gateways, and reassembly [21].
- Assurance: Issues in this area include detection of and recovery from lost, damaged, or duplicate packets; detection and handling of packets that arrive out of sequence, and provision of reliable mechanisms for protection of control information [4,6,8,13].
- Efficiency: Issues in this area include the interaction of flow control and buffer-management strategies, header size, use of negative acknowledgment, packet size, and the numbers of packets that must be exchanged to reliably transmit one or more data packets [2,3,7,13,15,18,22].

This paper is concerned primarily with the assurance issues in an environment where it is desired to minimize the number of packets exchanged to reliably transmit one or more information packets. The network environment for which the timer-based mechanisms discussed here are being considered has the following characteristics:

- High-speed local internode connections using a variety of technologies. Technologies in use include low-delay store-and-forward nodes interconnected with 5–50 Kb serial links, point-to-point parallel 5–40 Mb links, and planned 50 Mb shared coaxial cable buses [12,16].
- The maximum packet lifetime (MPL) can be bounded and is appropriately small, as defined later.
- Planned connections among several local networks, and possible future connection to other government or public geographically distributed networks. (We assume maximum packet lifetime can be bounded if this protocol is to be used across networks, or that a gateway will translate between different protocols.)
- The need for packet fragmentation and reassembly because of the constraints of the different networks and the variety of internode link technologies.
- The possibility, with low probability, of out-of-sequence arrivals through store-and-forward subnetworks, lost or damaged packets, and creation of duplicates within the network.
- The creation of duplicates at the source due to the use of a positive acknowledgment mechanism for recovery from lost or damaged packets.

The computers between which it is desired to provide communication cover the range from microprocessors to supercomputers. The types of traffic to be exchanged include: messages between service machines and interactive terminals; small, medium, and large files; raster display frames; real time data collection; and single-message traffic between "user" processes and distributed "network operating system" services.

Of all the environmental factors listed above, the one that makes our needs different from those for which other EEPs have been designed is the desire to make use of the potential offered by the high-speed local network environment for the design of network-based distributed operating systems built around efficient reliable single and double (data and acknowledgment) message exchanges. Existing EEPs or those under design [4–7,13,14,19,20] deal with some of the reliability issues to be discussed by first exchanging two or more messages to reliably establish a virtual circuit or logical connection between communicating processes, then carrying out their data exchanges, and finally exchanging two or more messages to reliably close the connection. Several authors have expressed the desire for message-based protocols as opposed to connection-based protocols, but have not dealt with the assurance issues [1,10,15,17,24,25]. Belsnes has shown that to reliably send a single data message requires the exchange of five messages, unless state information is retained after logical connections are severed [2]. We felt that this packet overhead was not reasonable in an environment in which the exchange of single data messages might be very common and so began to examine the mechanisms and algorithms necessary to tradeoff the retention of state information for the exchange of several messages in order to achieve both reliable single-message and message-stream communication.

The organization of the paper is the following. Section 2 outlines the protocol context in which the specific mechanisms developed in detail later are embedded. It is included for possible reader interest and to provide a specific protocol environment for the timer mechanism as an example. Section 3 and 4 define the assurance problems that the timer-based mechanisms are designed to solve by showing a current solution using an explicit exchange of messages. Sections 5 and 6 present the timer-based mechanisms and show how these constitute another solution to the problems of Sections 3 and 4. Section 7 discusses the factors requiring practical bounds on timer intervals. Section 8 compares the timer-based mechanisms with the exchange of message mechanisms. Section 9

is the conclusion ar
algorithm for the tir

2. Protocol context

We briefly outlin
without justificatior
in the three areas l
reader a full concre
visualize the tir
mechanisms are i
decisions, and so oth

Packets consist c
address and contro
Header-only packets

Many of the r
(including those d
some of the assuran
those of packet fra
cal information to
uniquely indentifi
messages, message s
bits. We call the uni
the state informatio
end, these units ar
numbering them. T
number (DSN) of
view of the mecha
unit chosen as the b
preference for an
sequence-number o
The number of d
the packet header t
the data-sequence-n
contained in the pa
to use a hierarchic
associated routing s
rules for header fo
tion cannot be giv
mentation is as desc

To deal with the
to use the positive
mechanism for deal
packet checksum
dealing with damag
sequence numbers
cribed, for handlin
tions. The retransr
important to the ti
as duplicates can b

is the conclusion and the Appendix gives a detailed algorithm for the timer protocol.

2. Protocol context

We briefly outline our protocol design preferences, without justification, for solving the main problems in the three areas listed above in order to give the reader a full concrete protocol context in which to visualize the timer mechanisms. The timer mechanisms are independent of most of these decisions, and so other choices could be made.

Packets consist of two parts, a header containing address and control information, and a data part. Header-only packets are possible.

Many of the mechanisms developed to date (including those described here) for dealing with some of the assurance and efficiency issues, as well as those of packet fragmentation, assume that the logical information to be exchanged is broken up into uniquely indentifiable indivisible units such as messages, message segments, octets (8-bit bytes), or bits. We call the unit chosen an *element*. To minimize the state information required at each communicating end, these units are usually named by sequentially numbering them. This number is the data-sequence-number (DSN) of the element. From the point of view of the mechanisms to be described, the size of unit chosen as the basic element does not matter. Our preference for an element is the octet. The data-sequence-number of a packet names the first element. The number of data elements is also recorded in the packet header to enable the rapid computation of the data-sequence-number of the last data element contained in the packet. Our choice for addressing is to use a hierarchical cluster or area addressing and associated routing scheme. Detailed discussion of the rules for header formation during packet fragmentation cannot be given here. Handling the data fragmentation is as described for TCP in ref. [4].

To deal with the assurance issues, we have chosen to use the positive acknowledgment/retransmission mechanism for dealing with lost packets; an optional packet checksum over both header and data for dealing with damaged packets; and a combination of sequence numbers and timer approaches, to be described, for handling packet duplicates and transpositions. The retransmission interval and strategy are important to the timer mechanisms to be discussed, as duplicates can be generated at the sender node if an acknowledgment is delayed, for whatever reason, beyond the retransmission interval.

Our current view regarding protection of control information is that its meaning should be so formulated that its loss or duplication does not matter. This is accomplished by requiring it to fit one of three conditions: report status about the sender, report the sender's current view of its partner's status, or describe the data in the packet in some way [8]. Then, if a packet is lost under the first condition, status information, possibly more up-to-date, will appear in later packets and if duplicated it will just cause the overwriting with an identical value of some register content. In the second condition, loss or duplication will be detected at the receiver and the appropriate control information regenerated or discarded as appropriate. In the third case, loss or duplication is associated with the data, and the mechanisms for dealing with data loss or duplication will be invoked and lead to appropriate recovery. Control information requiring frequent communication will be in a fixed-field header that will contain sender and receiver addresses, the data-sequence-number (DSN), the acknowledge-sequence-number (ASN) acknowledging all preceding elements, some control flags, the checksum, and a window size for flow control [4,5, 14,18,19].

Listing of design choices in the efficiency area is beyond this paper except to repeat that a window flow-control scheme is used. Receiver buffering is assumed to be by logically endless buffers or by explicit buffer quantization known by the sender. Negative acknowledgments are allowed to speed retransmission. The number of packets required to reliably transmit one or more packets is minimized by the timer mechanisms.

The above brief summary should give a flavor for the overall protocol within which the timer mechanisms to be described in Sections 5 and 6 are to be used. Of the issues mentioned above, the only ones discussed here are those interrelating the assurance issues, the desire being to achieve both an efficient single message and a message-stream capability.

We want to allow both simplex and full-duplex communication. (Half-duplex is a special case of full-duplex.) During communication, each side maintains a connection record where it records the status of its end of the conversation and its best current knowledge of the state of its partner. Because of network delays, the latter is likely to always be a little out of date.

A simplex connection exists or is established when both sides have connection records with their partner's address recorded and one side has a connection record in which is recorded a valid next expected data-sequence-number of its partner. A full-duplex connection exists or is established when both sides have connection records in which are recorded their partner's addresses and both contain a valid next expected data-sequence-number. A connection may exist very briefly during the exchange of a data packet and its acknowledgment or over a prolonged period during the exchange of a sequence of packets.

We now proceed to discuss the duplicate-detection and connection opening and closing problems that the timer mechanism is designed to solve. In order to compare the timer mechanisms with existing multi-packet exchange mechanisms to solve these problems, the issues are introduced using the multipacket exchange mechanisms.

3. Reliably opening a connection, and duplicate detection

We want to provide the following forms of duplicate protection:

a) If no connection exists and the receiver is willing to receive, then no duplicate packets from a previously closed connection should cause a connection to be opened and duplicate data accepted by the next level (user) program. This is the "replay" problem, where a series of old duplicates could cause a connection to be opened and the data replayed to the receiver from one or more packets.

b) If a connection exists, then no duplicate packets from a previously closed connection should be acceptable within the current connection.

c) If a connection exists, then no duplicate packets from the current connection should be accepted.

Let us consider these problems in order.

Problem (a) results because there has to be some way to indicate that a new connection is being established and what its initial data-sequence number is to be. This "opening" indication is usually made by the use of a control flag in the packet header. A packet with such a flag "on" is a dangerous packet, particularly if it should contain data. In the three-way-handshake mechanisms, discussed below, we call this flag the "OPEN flag". The OPEN flag only appears on in the first packet sent over a connection.

In the timer-based mechanisms there is no OPEN flag, instead a packet starting a "run" of contiguous sequence numbers has a data-run-flag (DRF) on. More than one packet in a connection may have the DRF flag on, as described in Section 5. Problem (a) results, for example, if a previous connection had just closed, the receiver was in a state ready for a new connection, and the EEP had the rule that when data arrived, either in a packet with the OPEN flag on, or with appropriate data-sequence-numbers following a packet with the OPEN flag on, it was passed on to the next-level program; then the arrival of an old duplicate OPEN packet with data from a previous connection would be accepted and the data passed on to the next level. Further, even if the old OPEN packet did not contain data, but was followed by old duplicate data packets with appropriate contiguous sequence numbers to that contained in the old OPEN packet, then they would be accepted and passed on to the next level program as an undetected replay. These conditions result because the receiver has discarded all state information about a connection once it has closed the connection and thus has no way to recognize an incoming packet with the OPEN flag on as an old duplicate. When such a packet arrives, the receiver proceeds to open a connection. When the receiver is closed, receipt of an old duplicate data packet with the OPEN flag off can be ignored.

To guard against this problem two mechanisms have been previously proposed, the three-way handshake [6], and the unique socket address [20]. It is our view that the latter is just a modified version of the former as it effectively relies on an exchange of confirmation messages to detect the duplication. Therefore, we will limit our discussion to the explicit three-way handshake. The three-way hand-shake protects against replay by not allowing data to be passed to the next level program until the successful exchange of three messages. It works as follows: Assume that node A wishes to communicate with node B in a full-duplex conversation.

(1) A sends B a packet with the OPEN flag on and an initial data-sequence-number.

(2) B acknowledges the receipt of this sequence number in a packet with the OPEN flag on and containing its own initial data-sequence-number.

(3) A in a third message acknowledges receipt of B's initial data-sequence-number.

A could include data with the first or third messages, but it is only on arrival of the successful acknowledgment by "A" of "B's" initial data-

sequence-number that B
(user) level. The reason f
by assuming that the in
old duplicate. B has no
responds as above. Howe
ledgment of this old du
recognize that it does n
number that it has sent.

A can perform this r
closed with respect to I
opening a connection t
sequence-number larger
connections with B. Th
be chosen in a variet
mapping the value of
timer into the data-sequ
of this field must be c
the mapping will not w
packet lifetime. It is th
three-way-handshake ap
assumption about a bo
time in its choice of
length.

Given that A can rec
ledging a valid sequence
error or reset signal rat
of B's initial data-seque
absence of any state ir
tion, B checks whether
duplicate by asking A tc

If a connection is n
duplicate from a previo
the OPEN flag on, th
rejected.

There are a number
procedure above, such
three messages get lost
open simultaneously.
detail in refs. [2,6].

An alternative appr
three-way handshake is
priate previous sequen
period of time, so th
arrives either from an
within the "current" co
fact. In this case, no co
is required. We will
remembering something
time the Δt mechanism
on maximum packet lif
arise as to what to retai

sequence-number that B would pass data to the next (user) level. The reason for this restriction can be seen by assuming that the initial message from A was an old duplicate. B has no way of knowing this, so it responds as above. However, when A gets B's acknowledgment of this old duplicate opening packet, it can recognize that it does not acknowledge any sequence number that it has sent.

A can perform this recognition because either it is closed with respect to B or, if it is in the process of opening a connection to B, it picks a new initial data-sequence-number larger than any used on "recent" connections with B. This initial sequence number can be chosen in a variety of ways; for example, by mapping the value of a monotonically increasing timer into the data-sequence-number field. The length of this field must be chosen large enough such that the mapping will not wraparound within a maximum packet lifetime. It is thus important to note that the three-way-handshake approach does make an implicit assumption about a bound on maximum packet lifetime in its choice of data-sequence-number field length.

Given that A can recognize that B is not acknowledging a valid sequence number it can reply with an error or reset signal rather than an acknowledgment of B's initial data-sequence-number. In effect, in the absence of any state information from past connection, B checks whether or not the OPEN packet is a duplicate by asking A to confirm it.

If a connection is not yet established and an old duplicate from a previous connection arrives without the OPEN flag on, then the old duplicate will be rejected.

There are a number of subtleties with the opening procedure above, such as what happens if any of the three messages get lost, or if both A and B try to open simultaneously. These are discussed in some detail in refs. [2,6].

An alternative approach presented here to the three-way handshake is for B to remember an appropriate previous sequence number of A's for some period of time, so that when a duplicate packet arrives either from an "old" connection or from within the "current" connection, B can recognize this fact. In this case, no confirming exchange of messages is required. We will call the approach based on remembering something for some interval (delta) of time the Δt mechanism. It requires explicit bounds on maximum packet lifetime. Questions immediately arise as to what to retain; how to choose the length of time Δt for the retention; what reliability and efficiency differences exist between a three-way handshake and a Δt algorithm; how a Δt approach is used both with streams of data packets (i.e., long-term connections, such as for use with interactive terminals or transmission of large multipacket files) and with single messages or bursts of single messages; what happens if simultaneous opens are attempted; etc. We discuss these issues in the sections to follow.

Problem (b) exists when there is an open connection and a packet from a previous connection arrives. For the three-way handshake there are two cases: either the OPEN flag is on or it is off. If the OPEN flag is on, it is rejected immediately as it is recognized as being an old duplicate for this reason. If the OPEN flag is off, there is no way to distinguish this packet from a valid one for this connection if its data-sequence-number is within the receiver's window of acceptable DSN values.

The three-way handshake solves this problem by trying to choose an initial sequence number for a connection that makes this case acceptably improbable as discussed above. A number of difficulties can occur with this approach that can lead to the need to change sequence numbers in mid-connection. These problems are not discussed here. These difficulties were one motivation for the unique-socket-number mechanism [19,20] being developed, and are fully discussed in Refs. [9,13,23].

The Δt mechanism avoids these problems by having the receiver wait until all duplicates have died out before destroying its connection record. On recovery from a crash the receiver also waits a time necessary for packets from a previous connection to have died out before receiving again. This solution would also solve the problem for the three-way-handshake approach.

Handling Problem (c) of detecting duplicates from the current connection is simply done by comparing the data-sequence-number of the received packet against the sequence number maintained in the connection record of the last contiguous element received. This technique is used in both the three-way-handshake and Δt approaches. The full acceptance algorithm for the timer mechanism is given in the Appendix.

4. The closing problem

A connection has been closed if the connection record either no longer exists or could be discarded.

Assume that a full-duplex connection has been established, that an exchange of data has been taking place, and that one side A has no more data to send and wishes to close the connection. The main problem is to achieve a graceful close; namely A should stay open not only until it has received all information B wishes to send but also until it knows that B knows that its final data has been received, and vice versa. Other forms of closing are possible where one side unilaterally ends the conversation or will no longer receive. We are not concerned with these here. The following exchange will accomplish the graceful close in the three-way-handshake approach:

(1) A sends an indication to B that it is ready to close, i.e., has no more data to send.
(2) B acknowledges receipt of this signal in some reliable way and continues sending its data.
(3) B eventually indicates with a final data packet that it has no more to send.
(4) A acknowledges the last data element sent by B and acknowledges B's desire to close.
(5) B, on receipt of this acknowledgment, tells A to actually complete the close. B can destroy its record at this point.

Even though A knew earlier, after step 3, that both it and B had no more data to send, it cannot close, since its acknowledgment of B's final data might have gotten lost, causing B to retransmit. On receipt of this retransmission, had A closed, A would have either ignored the retransmission or returned an error indication of some kind. The net result in either case would have been to leave B in a state of confusion, where it does not know whether its last data had actually been received by A. The result of this possible confusion is that B might assume A had crashed before receipt of its last data and therefore open a new connection to resend the last data, thus causing duplication. With the above algorithm, if B's final "close it" message gets lost the worst that can happen is that A is left with an inactive connection record.

Since connection-record space may be an important resource to A, it can protect itself with an appropriate time-out after step 4. The length of such a time-out contains another form of implicit assumption about maximum packet lifetime and the length of time B will continue to try and retransmit if the acknowledgment(s) from A is (are) not getting through; a subtle point, but one dealt with explicitly with the Δt mechanism. Some protocol designers have assumed that the probability of A's acknowledgments getting lost is low and have not required the final message of step 5 above (6). We have included it in the interest of describing the safest algorithm. Thus, in the absence of retaining special state information, anouter threeway handshake is required for a graceful close from the point where the last data is sent and final closing can be performed.

The condition for a reliable, graceful close requires each side to know that the other side has gotten its final data and to allow for reliable retransmission if an acknowledgment does not arrive. The above condition can be met without a three-way handshake using a timer-based approach, if (a) the sender as well as the receiver has a timer and (b) appropriate rules are established governing use of data-sequence-numbers, use of control flags, and the restarting of the timers.

5. The Δt mechanism

The Δt mechanism is based on both sender and receiver maintaining state information long enough so that duplicates can be detected, information flow is smooth, and transmissions, retransmissions, and acknowledgments will have arrived at their destinations, if they are ever going to arrive. We must derive bounds on the interval values for the send and receive timers, and develop rules for when the timers get initiated, when a node can terminate a connection (delete its connection record), and how the sender should choose data-sequence-numbers. The rules for the Δt mechanism are quite simple, but their justification involves a number of subtle points.

The Δt mechanism developed here is based on the assumption that the data sent from a node A to node B can be considered essentially an infinite stream of elements. The goal is to deliver these elements in sequence, and without element losses or duplicates. If it is desired to impose additional structures on top of this element stream, such as that of logical messages, this can be done and is independent of the mechanism described here. We first define the control information used by the Δt mechanism and then develop the timer values and rules for manipulating that information. The control information exists in the packet headers and in the connection records and other information at the two ends. Some readers may find it useful to skim the Appendix at this point, where this state information is defined in more detail.

For the purposes of the Δt mechanism, a packet header contains two control flags, the data-run-flag

(DRF) and the a
sequence number
(DSN) and the
(ASN); and two si
elements and the
assume WIN is an
the ASN field of th
tain additional co
logical message an
here. The DRF, if
far as the sender is
as the beginning o
numbers. As far as
in packets with th
with what came b
indicates to the re
acknowledgment c
ARF, if on, indica
expecting a packet
in the ASN field is
the sequence num
the sender of the
implicitly acknow
elements. Note th
CLOSE flag. The I
a function similar
handshake. The di
only in the first p
way handshake,
many, even all,
Closing is handled

Let us now e
receiver must mai
must follow for h
for acceptance of i

5.1. Receiver rules

Because full-du
two simplex conve
to consider just a s
sending data to a
function of the re
from being accept
is to assure smooth

Consider that
connection record
Both A and B are
least one packet.
between A and
exchange and th

(DRF) and the acknowledged-run-flag (ARF); two sequence number fields, the data-sequence-number (DSN) and the acknowledged-sequence-number (ASN); and two size fields, the data length (LEN) in elements and the window (WIN) in elements. We assume WIN is an increment relative to the value in the ASN field of the packet. The header will also contain additional control flags for indicating end-of-logical message and for other purposes not of interest here. The DRF, if on, indicates to the receiver that as far as the sender is concerned the DSN can be treated as the beginning of a new *run* of contiguous sequence numbers. As far as the recevier is concerned, the DSN in packets with the DRF on need not be in sequence with what came before, although it may. Implicitly it indicates to the receiver that the sender has received acknowledgment of all previously sent elements. The ARF, if on, indicates that the sender of the packet is expecting a packet with DRF on and that the number in the ASN field is meaningless; otherwise the ASN is the sequence number of the next data element that the sender of the ASN expects to receive, and it implicitly acknowledges receipt of all previous data elements. Note that there is no OPEN flag and no CLOSE flag. The DRF in the Δt mechanism performs a function similar to the OPEN flag in the three-way handshake. The difference is that the OPEN flag is on only in the first packet of a connection in the three-way handshake, whereas the DRF may be on in many, even all, packets using the Δt mechanism. Closing is handled implicitly via timeouts.

Let us now examine what timer information a receiver must maintain and the rules that the receiver must follow for handling its timer, for closing, and for acceptance of incoming packets.

5.1. Receiver rules and timer bound

Because full-duplex conversations can be treated as two simplex conversations, it simplifies the discussion to consider just a simplex conversation from a node A sending data to a node B willing to receive data. The function of the receiver rules is to prevent duplicates from being accepted. The function of the sender rules is to assure smooth flow.

Consider that neither A nor B have an existing connection record for a conversation with the other. Both A and B are assumed to have buffer space for at least one packet. We will examine the exchanges between A and B, first for a single data message exchange and then for bursts and longer message streams, discussing the important design issues where appropriate. A sends a packet to B containing:

DRF = on
ARF = on
DSN = x (any value can be used)
ASN = anything
LEN = l

In an expected full-duplex conversation, A would also include a window size, WIN. The DRF is on because the DSN in the packet begins a new contiguous run of sequence numbers. The ARF is on because A has no elements to acknowledge. The value x chosen for DSN can be anything since it is assumed, due to the rules discussed below of the Δt mechanism, that there are no existing old duplicate packets from previous conversations between this pair of nodes; otherwise there would be an existing connection record.

For purposes of this discussion the receive part of the connection record for B containes three pieces of state information: the input window left edge, the receive timer, and the number of elements of buffer space available (the input window size). On receipt of A's message B creates a connection record and records $x + l$ as the value of the next expected DSN (input window left edge), decrements the input window size by l, passes the data to the next level (user) program, sets its receive timer (R_{timer}) to the appropriate value, and generates an acknowledgement packet.

How long should the interval be that is set into R_{timer}? As a minimum, it must be long enough to guarantee that any duplicate data that may have been generated by the sender A or by the routing network will have died out during the interval. We will use R to denote the time period during which the sender can retransmit and thus create duplicates. The time period during which the routing network can create duplicates and contain these duplicates of the original packet is the maximum packet lifetime, (MPL). To be precise, we are actually interested in the maximum data element lifetime within the routing network, but it is the same as MPL on the assumption that, if packet fragmentation occurs, all packets generated propagate the age of the original packet. These factors R and MPL establish a lower bound on the value R_{time} set into R_{timer} for conversations between any pair of nodes to guarantee duplicate detection:

$$R_{\text{time}} \geq R + \text{MPL}$$

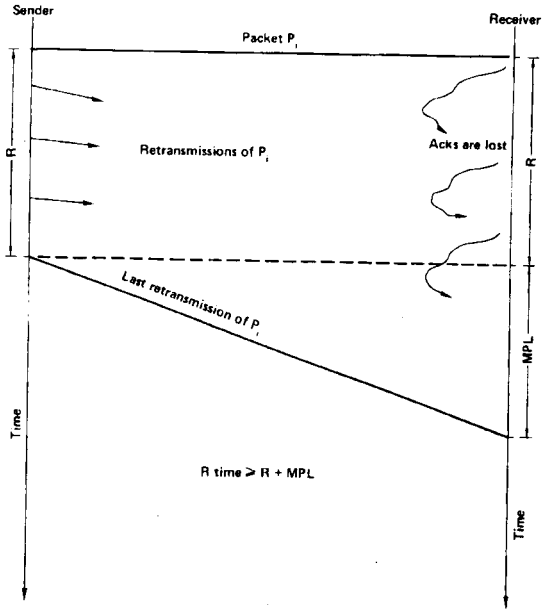


Fig. 1. Worst-case time for duplicate detection.

Later we discuss the terms that make up R. Fig. 1 shows the worst case where the original packet arrives "instantaneously" at the receiver, acknowledgements do not get through and the sender keeps retransmitting for time R, and the last retransmission takes MPL to arrive at the receiver.

For the Δt mechanism to work, R and MPL must be bounded. We claim that, for any packet switched network to achieve maximum transmission reliability, MPL must have a firm upper bound. If this were not the case, then there is the possibility that a packet from a previous connection with an acceptable DSN might exist. We have also seen that the three-way handshake does depend on implicit assumptions about MPL being bounded.

There are undoubtedly many mechanisms that could be used to bound a packet lifetime. One obvious one would be to bound the number of nodes through which a packet can travel and the maximum time spent in any node. These factors, combined with knowledge of the maximum packet size and minimum transmission rate, would allow the establishment of a mechanism for enforcing a maximum packet lifetime. Another mechanism for bounding MPL would be to use a relative time or aging field in the packet header that is incremented as the packet travels through the network. When the value of this field exceeds MPL, the packet would be destroyed.

There is one additional factor that must be bounded, as developed later. This factor is the maximum time from the arrival of a data element within the receiver protocol module until an acknowledgement is sent, assuming it had arrived in proper order. We call this the unacked time (UAT). The unacked time can be bounded by use of a timer on arrival at the receiver or by use of an aging field.

Having picked a lower bound on the value of R_{time} for duplicate detection purposes, we can see that it does not have to be a precise number as long as it is at least R + MPL. Tables would not have to be kept of R_{time} for all possible partners, but could be abbreviated for worst case between nodes in particular areas or clusters or for the entire network. Having discussed some issues affecting the value of R_{time}, we consider the rules that the receiver, B, must obey. Later we examine the term R in detail, but first we have to discuss sender timing considerations.

- *Rule 1:* Reset R_{timer} to R_{time} every time the receiver receives in proper order a data element with a sequence number not previously acknowledged.
- *Rule 2:* The receiver part of a connection record should be closed (the connection record considered as containing invalid information or destroyed) when R_{timer} goes to zero. All unacknowledged elements are discarded at this point. The reason why the receiver cannot maintain a record beyond R_{time} is discussed below when we develop the rules for the sender. The receiver must maintain the record at least R_{time} as shown in fig. 1 to provide duplicate detection.
- *Rule 3:* Data elements are accepted according to the following two conditions:
  1) If there is no valid connection record (R_{timer} equal to zero) then it is an implementation option whether or not a packet with the DRF off will be accepted. Such a packet is out of order and no data can be passed to the next level program or acknowledged until a packet with DRF on arrives. Both options are considered in the Appendix. The initial packet with DRF on can contain any DSN. Effectively a new receive connection record is put into use.
  2) If R_{timer} is nonzero, then only elements with sequence numbers within the receiver's input window are accepted. The DRF is ignored. The reason why many packets may have the DRF on are developed below. (The receiver may

optionally
Accepted ele
Acknowledg
place withir
later.
B acknowledges A

DRF = on
ARF = off
DSN = y (can be a
ASN = $x + 1$
LEN = O (we are
WIN = appropriat

The rule fo
and DSN in a hea
be chosen accord
them for a data
because the DSN
initial value for
assumption of a s
DSN will be the s
receiver. The ARI
a valid acknowled
in the header th
duplication as in
[6]. The LEN fiel
and the WIN field
relative to the val
receive. However,
duplex, B could
onto a packet con

A receiver shc
response packet
also when elemen
tance window. Th
sender know the
ing to clear up pc
delayed acknowle

5.2. Sender rules

We now need
state information
We also need to e
mission and for c
considered.

1) We want to
bursts of lo
(messages) v
tion to time
as be able

optionally ignore out-of-sequence packets.) Accepted elements are acknowledged.

Acknowledgement for an element must take place within a bounded interval as discussed later.

B acknowledges A's packet by sending:

DRF = on
ARF = off
DSN = y (can be any value)
ASN = $x + l$
LEN = O (we are assuming simplex connection)
WIN = appropriate value

The rule for choosing the values for the DRF and DSN in a header only packet is that they should be chosen according to the rules used for choosing them for a data packet. Therefore, the DRF is on because the DSN starts a new run. The value y is the initial value for DSN. It can be any value. On the assumption of a simplex connection the value in the DSN will be the same for all packets generated by the receiver. The ARF is off because the value of ASN is a valid acknowledgement. There are no fields or flags in the header that need protection against loss or duplication as in some other protocols such as TCP [6]. The LEN field is zero to indicate there is no data and the WIN field indicates how many more elements relative to the value in the ASN field B is prepared to receive. However, if the connection were to be full duplex, B could "piggy-back" the acknowledgement onto a packet containing data.

A receiver should be triggered into generating a response packet not only when it accepts data, but also when elements arrive that fall outside the acceptance window. The purpose of the latter is to let the sender know the receiver's current state thus attempting to clear up possible confusion arising from lost or delayed acknowledgment packets.

5.2. Sender rules and timer bound

We now need to examine the timer and related state information that the sender A must maintain. We also need to establish the sender's rules for transmission and for closing. Three problem areas must be considered.

1) We want to allow the sender the ability to send bursts of logically independent sets of elements (messages) without having to wait for a connection to timeout and close between each, as well as be able to send a sequential stream of elements, and have their sequencing maintained whether or not they are so widely separated in time that the connection has timed out. To accomplish this we need to develop a mechanism for a loose coupling between the sender's timer for sending, S_{timer}, and the receiver's receive timer, R_{timer}. We further need to develop rules for use of the DRF and the choice of DSN.
2) We want to establish the rules and timing considerations for handling the case where at least one data element has had its maximum number of retransmissions and there are some new data elements to transmit and/or some other data elements outstanding that have been transmitted, but not yet acknowledged.
3) We need to establish what information the sender must maintain in order to bound R, the interval during which retransmissions can take place.

We now discuss the first problem. One possibility for handling logical messages is to create a separate connection record with its own S_{timer}, R_{timer} and other state information at both sender and receiver for each message; in effect creating a separate connection record for each message. We rejected this approach as unwieldy and inefficient. Instead we choose to use only a single connection record and consider all elements sent from a sender to a receiver to be an infinite sequential stream. During the life of a connection, as viewed by the receiver, elements are only accepted if their sequence numbers fall within the receiver's acceptance window; the elements are sequenced by the receiver, and acknowledgment is only generated for an element when all preceding elements have arrived.

The price paid for this decision is that logical messages must also arrive in sequence. That is, all the elements of message 1 must arrive and be delivered before message 2 can be delivered, even if all the elements of message 2 arrive before all those of message 1. Additional simple mechanism can be built on that described here to remove this restriction, if desired. We could imagine both applications where sequencing of logical messages was important and where it did not matter. This fact coupled with the assumption that out-of-sequence message arrival would either be rare, or in the case of single-packet messages would normally occur with short spacing between them, lead us to feel that the extra mechanism required to support out-of-sequence message

arrival to be of questionable value.

According to Rule 3 above, for a sender to have an element accepted while the receiver's R_{timer} is running, it must have a sequence number within the receivers acceptance window. Because of network delay uncertainties, the sender can never know exactly when the receiver's timer has run out. Therefore, our goal is to find an initial value for a send timer, S_{timer}, and a set of rules for its resetting such that we can quarantee that the sender's S_{timer} continues to run so long as the receiver's R_{timer} is running. In this case the sender will maintain the sequence number state information needed to assure that an acceptable DSN will be placed in each packet as long as the receiver's R_{timer} is running. We must still assure that the packet will be accepted if the receiver's timer has timed out.

Since the receiver pays no attention to the DRF and only considers the element sequence number when its receive time is running, one might consider setting the DRF on in every packet, so that the packet would be accepted whether it arrived before or after the R_{timer} had run out. If one could assume that packets could not arrive out-of-sequence, this would work and eliminate the need for a DRF. (If it is always on, it is not needed.) However, there is the following sequence problem: If the receiver's timer had gone to zero, the sender had sent a burst of packets, and one of the later packets arrived at the receiver out-of-sequence, then the receiver would accept the elements in the packet and initialize its connection record with input window left edge equal to the value DSN + LEN contained in the first received packet. Thus, when the logically preceding packets arrive, containing elements with preceding sequence numbers, thay would be rejected as if they were old duplicates.

One solution to the problem would be to require a packet's elements to be acknowledged before allowing succeeding elements to be sent. This would clearly reduce throughput unnecessarily and therefore is rejected.

Therefore, a flag, the DRF, is needed to indicate the start of a run of sequence numbers so that when the receiver gets this flag, it can know that there are no "earlier" numbers in this sequence that it may receive.

The following rules for the sender's handling of its timer, choice of a DSN for a packet, and the setting of the DRF are now given.

- *Rule 4:* Reset the S_{timer} to S_{time} (derived below) whenever a packet containing data is sent or resent.
- *Rule 5:* If the sender's S_{timer} is nonzero, then choose as a value for DSN in new packets the value plus one of the sequence number for the last element sent (excluding retransmissions); otherwise any value of DSN can be used.
- *Rule 6:* If all preceding data elements have been acknowledged, then set the DRF on, otherwise set the DRF off. This is done for both new transmission and retransmissions.

Rules 4 and 5 guarantee that the value of DSN will be acceptable to the receiver when R_{timer} is nonzero. (The value of DSN does not affect acceptability when R_{timer} is zero.) This is so because S_{timer} is reset as each packet is transmitted and the initial value for S_{timer} is chosen to quarantee that it does not time out before the receiver's R_{timer}. Let us examine how the setting of the DRF by Rule 6 solves the problem cited above.

Assume that the receiver's timer has run out and that the sender transmits a burst of packets, the first one with the DRF on and the later ones with the DRF off. If one of those with the DRF off arrives first, then it can be rejected, or it can be considered an out-of-sequence packet and held, on the assumption that the packet with the DRF on will arrive shortly. This is an implementation choice. The algorithm in the Appendix covers both options. Rejecting a packet will eventually lead to its retransmission. In any case, element sequencing is maintained.

The above discussion considered this situation near the beginning of a run where the out-of-sequence arrival of packets was a potential problem. A question still remains whether or not a packet with the DRF off could arrive at the receiver after its timer had gone to zero. We demonstrate later, after discussing the solution to the third problem above and introducing Rule 7, that this cannot happen. The net result of these rules is that sequencing of elements is maintained even if the time between their transmission is such that the receiver's connection record may have timed out and require reinitialization.

We now develop the bounds for the send timer interval. The sender wants to keep its send part of the connection record; a) as long as the receiver's timer is running, so as to generate an acceptable stream of contiguous data sequence numbers, and b) long enough to assure that it will receive all acknowledgments that may arrive. Consider condition (a). The receiver sets its R_{timer} at the point that it receives a

new data element t
time for a packet
MPL. Therefore, c
val.

$S_{time} \geq MPL + R_{ti}$

as derived previous

$S_{time} \geq 2MPL + R$

Case (b) above req

$S_{time} \geq 2MPL + U$

as both the pack
each take MPL tc
the worst case
acknowledgment.
$R > UAT$, and th
gent condition on
tion on S_{time} is:

$S_{time} \geq 2MPL + R$

Let us now con
lier, namely the
considerations for
least one data eler
and there may be
some elements alr
been acknowledge
blem, namely the t

We now define
time during which
retransmit an elen
for any element,
longer transmit or
seems likely that
failure. Because tl
data is sent, the s
will then terminat
edgments arrive.
effect indicate th
have resumed fu
set to be any
sender and receive
some number of d
choosing the time
slightly larger than
packet and its acl
and B. In this case

$G = n * \delta t$.

The term G is c

new data element to be acknowledged. The maximum time for a packet to travel from sender to receiver is MPL. Therefore, case (a) requires a send timer interval.

$S_{\text{time}} \geq \text{MPL} + R_{\text{time}}$, where $R_{\text{time}} \geq \text{MPL} + R$,

as derived previously; so

$S_{\text{time}} \geq 2\text{MPL} + R$.

Case (b) above requires:

$S_{\text{time}} \geq 2\text{MPL} + \text{UAT}$,

as both the packet and its acknowledgment might each take MPL to get their destination and UAT is the worst case for the receiver to generate the acknowledgment. Below we demonstrate that $R > \text{UAT}$, and thus case (a) imposes a more stringent condition on the value of S_{time}. So the condition on S_{time} is:

$S_{\text{time}} \geq 2\text{MPL} + R$.

Let us now consider the second problem listed earlier, namely the establishment of rules and timing considerations for handling the situation when at least one data element has had its last retransmission and there may be some elements to transmit and/or some elements already transmitted that have not yet been acknowledged. We also consider the third problem, namely the bound on R.

We now define a "giveup" interval G that is the time during which the sender will continue to try and retransmit an element. When the interval G runs out for any element, the sender will give up; that is, no longer transmit or retransmit data elements, since it seems likely that the connection is experiencing a failure. Because the S_{timer} is refreshed only when data is sent, the send part of the connection record will then terminate after S_{time} unless new acknowledgments arrive. These new acknowledgments in effect indicate that the network and/or receiver have resumed functioning. The time G might be set to be any arbitrarily agreed value between sender and receiver, but is probably set by choosing some number of desirable retransmissions, say n, and choosing the time between retransmissions, δt, to be slightly larger than the average round trip time for a packet and its acknowledgment moving between A and B. In this case the value for G is:

$G = n * \delta t$.

The term G is one component in the total retransmission interval R, as discussed below.

How long should the sender wait after its last retransmission of an element until it closes the connection and reports the problem to its user program? For maximum reliability it should wait until the sender time interval has elapsed. This assures that, if the last packet and its acknowledgment are each taking no more than MPL to be delivered, (due to some temporary overloading or other factor of the network) then transmissions can continue, although slowly, and may return to normal when the cause of the delay goes away. If there is a more serious problem, and the sender closes after the send time interval has elapsed and then later restarts transmission over again, the receiver's timer will have gone to zero and no old duplicate packets will exist.

What we want is a rule that will:

1) Allow a reliable close in a definite time period if the receiver is down or the path to the receiver is effectively cut,
2) guarantees that no packet sent with the DRF off, that arrives at the receiver after a predecessor packet with the DRF on, can arrive after the receiver's timer has run out.
3) guarantees that if elements sent their maximum number of times are eventually acknowledged, even if in the worst case, then the immediate successor element will have at least one more retransmission; or stated differently, if all elements that have had their last retransmission are acknowledged, then normal transmission can be resumed.

Condition 1 just stated that the sender does not want to keep transmitting new packets and retransmitting old packets indefinitely once trouble is suspected.
Condition 2 assures that the sender's packets will not be unnecessarily rejected and lead to unnecessary retransmissions.
Condition 3 assures that the protocol can recover from intermittent network troubles such as periods of congestion or brief link failures. In fact the rule to be given reduces the aggravation of the network loading that might cause large delays. The rule that satisfies the conditions above is:

· *Rule 7:* When an element is given its last retransmission: suspend all transmissions and retransmission of other elements, and suspend the G interval timers for all elements awaiting acknowledgment. If the acknowledgments of all elements having had their last retransmission arrive before S_{timer} goes to zero, then allow normal transmission and retransmission to continue, and resume operation of the suspended G

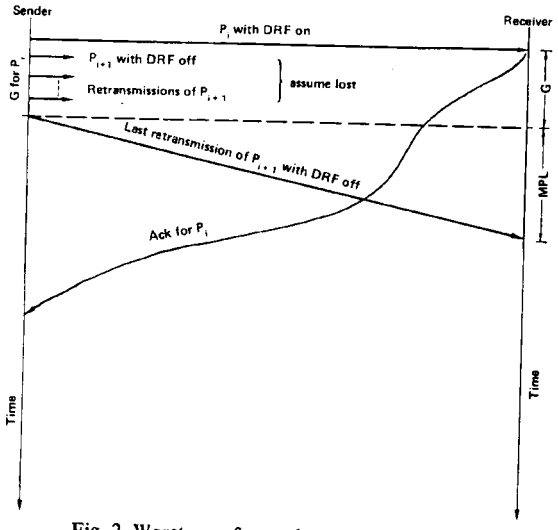


Fig. 2. Worst case for packet with DRF off.

interval timers. If no acknowledgment arrives before S_{timer} goes to zero, then send an appropriate indication to the next level program and close the connection. The send part of a connection can be normally closed when S_{timer} goes to zero.

We now need to demonstrate that the above three conditions are satisfied by this rule. Condition 1 is clearly satisfied as the value of the send timer is refreshed at the point of the last transmission or retransmission and not refreshed again unless all elements sent in that retransmission are acknowledged. To see that condition 2 is met consider the following argument based on fig. 2.

A packet P_i with the DRF on is transmitted and (in worst case for this problem) arrives "instantaneously" at the receiver, refreshing the receiver's timer. Shortly after P_i is sent a packet P_{i+1} containing additional elements is sent with DRF off. By Rule 7 packet P_{i+1} with the DRF off can be sent or resent only during the interval G after P_i was sent. Either the acknowledgment to P_i would have arrived within G and thus in retransmission of P_{i+1} DRF would have been on per Rule 6, or no acknowledgment of P_i would have occurred and all retransmissions would be suspended. Thus, the latest time after P_i was sent that a packet P_{i+1} can be sent with DRF off is G.

Now P_{i+1} could have taken MPL to arrive at the receiver from the time of its last transmission, which in worst case described in the previous paragraph is at time G after P_i was sent. Therefore, it will arrive at the receiver in time MPL + G after the time P_i was sent (and in worst case received) and the receiver's timer was set. The receiver's timer was set to $R_{\text{time}} = R + \text{MPL} > G + \text{MPL}$ (since R, the time during which retransmissions can take place, must be no less than G); thus, R_{time} is as long or longer than the worst case for a copy of P_{i+1} to arrive with DRF off. One can examine the case where additional packets P_{i+2}-P_{i+n} are sent during interval G after P_i to see the argument can be recursively applied relative to P_{i+1} and so on. Let us now consider consition 3 above.

The reason why a packet P_{i+1} might, without rule 7, have used up its retransmissions interval G before a predecessor P_i's elements were acknowledged is that the maximum round trip time for a packet and its acknowledgment, 2MPL + UAT, is likely in practice to be greater than G. Because the elements of P_{i+1} have their G interval times suspended when P_i's G interval lapsed, on receipt of the acknowledgment of the predecessor elements there will be some time left in P_{i+1}'s G interval. Thus P_{i+1} will have at least one retransmission left, and (barring network failure) it will arrive at the receiver after P_i has arrived. Therefore, we see that Rule 7 satisfies the three desirable conditions above for handling the closing of a send connection record.

Let us now return to the total retransmission interval R and determine its component terms as illustrated in fig. 3.

The worst case, longest retransmission interval, occurs if the G interval timer has been suspended. Using the example above, we can bound this suspend time. If P_{i+1} had been transmitted just before packet P_i's G interval ran out it would have essentially its entire G interval left when normal transmission resumed.

We claim the worst case that must be considered for duplicate arrival at the receiver occurs when the original P_{i+1} arrives at the receiver very close to the time that P_i arrives, the acknowledgment to P_i takes UAT to leave the receiver and MPL to return to the sender and then retransmissions of P_{i+1} continues for an interval G (its acknowledgments being delayed or lost). This implies that $R = \text{UAT} + \text{MPL} + G$. Thus, substituting into the expressions derived earlier,

$$R_{\text{time}} \geqslant \text{MPL} + R = 2\text{MPL} + \text{UAT} + G .$$

From the earlier discussion,

$$S_{\text{time}} \geqslant 2\text{MPL} + R = 3\ \text{MPL} + \text{UAT} + G .$$

Maintaining the value of MPL, UAT, and G for all

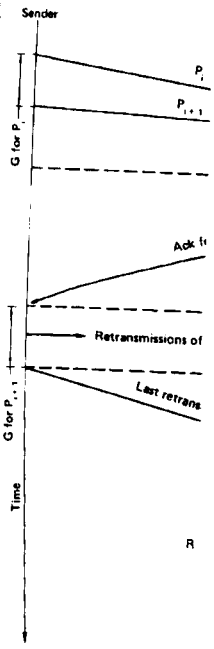


Fig. 3. D

possible pairs of p
essary burden. The

$\Delta t = \text{MPL} + \text{UAT} +$

so that each node
Δt, for conversatic
the same Δt), and

• *Rule 8:*

$R_{\text{time}} = 2\Delta t,$

as the appropriate
these values are la
slightly so, since M
of Δt. The reader
R_{time} and S_{time}
value of R_{time} lar
S_{time} must be cor
values used are
receiver pair.

The rule for clo

• *Rule 9:* A side n
timers both equal

Because the sen
end are logically s
opened and closed.

This completes

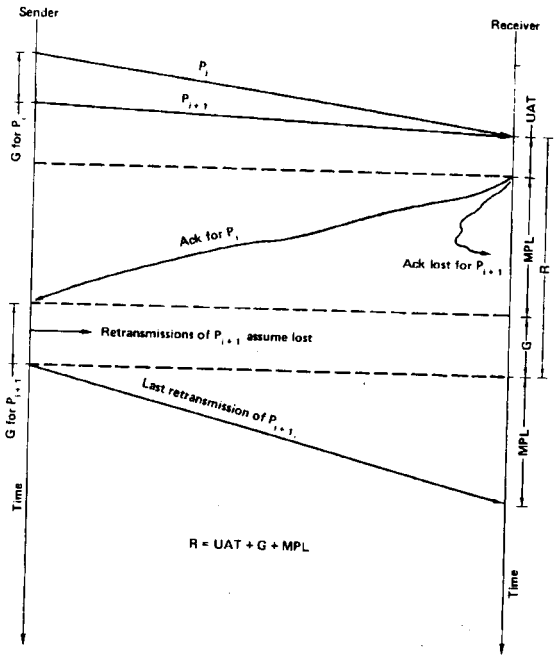


Fig. 3. Determining the value of R.

possible pairs of partners would seem to be an unnecessary burden. Therefore, we define

$$\Delta t = \text{MPL} + \text{UAT} + G$$

so that each node only needs to know a single value, Δt, for conversations with any partner (both ends use the same Δt), and define:

• *Rule 8:*

$$R_{\text{time}} = 2\Delta t, \qquad S_{\text{time}} = 3\Delta t,$$

as the appropriate timer values to be used. While both these values are larger than necessary, they are only slightly so, since MPL is usually the major component of Δt. The reader can see that these values satisfy the R_{time} and S_{time} inequalities. If a receiver uses a value of R_{time} larger than above, then the value of S_{time} must be correspondingly increased. The actual values used are negotiable between any sender/receiver pair.

The rule for closing is:

• *Rule 9:* A side may close when its receive and send timers both equal zero.

Because the send and receive sides of a connection end are logically separate, they can be independently opened and closed, if desired.

This completes the discussion of the considerations for a simplex connection. Extension to a full-duplex connection is straightforward as send and receive rules operate independently. Each end has both a send and a receive timer. The rules for use of DSN's and the DRF remain the same.

We have now finished the development and justification of the Δt mechanism as represented in the timer values and rules above. The simple rules above yield a very easily implemented and efficient protocol as shown in the Appendix.

6. Choice of an initial sequence number and crash recovery

The problem of choosing an initial sequence number is not unique to the Δt algorithm. It must also be dealt with in the three-way-handshake approach [3]. The problem is the following.

When an initial sequence number is chosen for a new connection, it must be such that no old duplicates from previous connections could be accepted within the new connection. On the assumption that there has been no crash with loss of memory, the Δt algorithm handles this problem by using the next contiguous sequence number while the send timer is non-zero. Once the send timer has gone to zero, by the basic assumption of the algorithm that Δt can be bounded, there are no old duplicates from previous connections to worry about. Thus, any initial DSN (such as zero) is acceptable.

A similar approach of remembering the last sequence number of previous connections for a time equivalent to Δt has been suggested for use with the three-way-handshake mechanism [13].

The problem of initial sequence number choice still exists on recovery from a crash with loss of memory. In this case the system does not retain the last DSN used or the value of the send and receive timers. We feel that for any reasonable networks or chain of networks Δt will range from a few milliseconds to around 30 seconds and that the correct solution on recovery from a crash is not to send or receive from a given node until waiting one S_{time} interval. (There is a proposal from IFIP to CCITT that the maximum packet lifetime through any chain of networks be less than 30 seconds [5]. The recommendation has not yet been accepted.)

Another case of failure is possible, namely that there is no loss of memory during a crash, but timers were not being decremented for some period. If the

timers were represented as time of day, then recovery is possible. If the timers were relative, then we recommend closing all connections and waiting a S_{time} period for each connection before resuming communication.

7. Bound on Δt

There are three factors that require that Δt not be overly large.

- *Data sequence number space:* The field size chosen for the data-sequence-number is of finite length and effectively wraps around modulo the largest integer representable plus one. Therefore,

$$2\,\Delta t = R_{time} < (\text{Max DSN/Max Element Transmission rate})$$

- *Crash recovery time:* Because we want to assure that old packets have died out before sending and receiving after a crash, we must wait a $3\Delta t$ time. This time should be a reasonable number.
- *Connection record storage limitations:* This issue is discussed in more detail below but basically the average number of connections expected per the S_{time} and R_{rtime} intervals should be low.

8. Comparison of the three-way-handshake and Δt approaches

Let us examine the three-way-handshake and Δt approaches in the areas of efficiency and reliability.

8.1. Efficiency

There are two aspects of efficiency, one involved with the actual throughput, which includes message and packet header overhead, and the second concerning implementation issues such as code and state information space required.

The header overhead in the two approaches is essentially identical, plus or minus a few control flags, as each could use the same size address, DSN, ASN, window, and length fields. The throughput is primarily determined by the interaction of the chosen flow-control and buffering strategy, the retransmission strategy, and implementation decisions. We have no reason to assume any significant differences in these areas between the two approaches.

The main area of difference between the approaches is in the overhead of opening and closing connections. It is smaller in the Δt approach, which makes it attractive for use with single message type applications. This is probably the main attraction of the Δt approach. For longer-lived connections, such as for interactive terminal sessions or transmissions of large files, the extra overhead of the three-way approach would not seem very significant.

The rules for the Δt algorithm are quite simple, as are those for use with the three-way handshake, and we have no reason to suppose one or the other to require more code space in its implementation.

The Δt approach does require state information (connection records) to be maintained for possibly a period equal to Δt longer than might be necessary for the three-way handshake, although the three-way handshake does have to maintain them for the time necessary to exchange several close messages. In a high-speed local network environment Δt will be quite small, probably less than 5 seconds and likely less than 1 second. In a geographically distributed network or chain of networks, we believe it can be kept around 1 minute or less. So that even assuming several thousand connections per hour for a local network host or several hundred per hour for a geographically distributed network host, we find that the number of expected unused connections within a R_{time} or S_{time} period waiting to close is quite low, on the order of 2 to 30. If this number seems large or should be higher occasionally, infrequently used connection records could be moved to secondary storage. Connection-record space requirements do not seem to be an area of crucial difference between approaches.

The Δt algorithm depends on knowing the value for Δt to use between each pair of communicating nodes. Each end must use the same value or a close approximation thereto. The values used can be checked in an exchange of control state information if desired. In fact, Δt must only be larger than a minimum value between any pair of nodes and so we would expect that it would be chosen to be the same as the worst case for all nodes in a given cluster or area. The three-way-handshake approach using a positive acknowledgment retransmission mechanism must also have built in assumptions or tables indicating the retransmission interval to be used between nodes. It might even need a Δt-type interval to prevent, on crash recovery, the problem of reuse of an old sequence number, as mentioned above. We feel the two approaches are about equivalent in the

amount of long-te
for timeout values

8.2. Reliability

The crucial diff
in dealing with t
connections (they
assurance problem
MPL, and UAT ca
mechanisms to a
earlier. Therefore,
Actually getting
implement such b
cal problem in th
dard. Therefore,
applicable in a loc
we did point out
has some implici
on MPL built i
difference is in v
of hosts at either e
Δt approach crasl
to detect duplicat
problem was deal
resuming commu
ular partner.

Waiting after a
the resyncronizat
three-way handsha

There is one
approach not fo
approach, namely
the same value fo
ly. Let us conside
clock runs too s
problem introduc
required to maint
necessary. If the
receiver's timer
problems are int
be affected due to
by the receiver pc
ments with the I
arrive before the
practice the time
siderably for mucl

The one possib
runs too fast so t
DRF on). In prac
off by a considera

amount of long-term information that has to be kept for timeout values.

8.2. Reliability

The crucial difference between the two approaches in dealing with the detection of duplicates between connections (they are the same in handling the other assurance problems) is the explicit assumption that G, MPL, and UAT can be bounded. There are a variety of mechanisms to accomplish this and one was given earlier. Therefore, we see no technical problems here. Actually getting existing networks and hosts to implement such bound enforcement may be a political problem in the absence of an international standard. Therefore, we see the Δt approach as most applicable in a local network environment. However, we did point out earlier that the three-way approach has some implicit assumptions about the bound on MPL built into it. Another area of possible difference is in vulnerability to problems if crashes of hosts at either end should occur. If a host using the Δt approach crashes losing memory, then its ability to detect duplicates fails for a period of R_{time}. This problem was dealt with by waiting for R_{time} before resuming communication reception from a particular partner.

Waiting after a crash for MPL would also eliminate the resyncronization problem documented in the three-way handshake [6,9,13,23].

There is one more requirement with the Δt approach not found using a three-way-handshake approach, namely, that each end of a connection use the same value for Δt and that the clocks run correctly. Let us consider the different cases. If the sender's clock runs too slowly then there is no reliability problem introduced, although the sender will be required to maintain connection records longer than necessary. If the sender's timer runs too fast or the receiver's timer runs too slowly, no reliability problems are introduced, although efficiency could be affected due to unnecessary retransmissions caused by the receiver possibly rejecting some packets or elements with the DRF on and new initial DSNs that arrive before the receiver's timer has run out. In practice the timer rates would have to be off considerably for much effect to take place.

The one possibly serious case is if the receiver timer runs too fast so that it cannot detect duplicates (with DRF on). In practice the timer rate would have to be off by a considerable amount to cause significant problems. We can, however, only conclude that timers on systems using the Δt protocol must be periodically checked to avoid this possibility.

Is one or the other approach more susceptible to lost or duplicated information due to uncertainty in either partner as to whether or not a packet reached its destination? On the assumption of waiting for $3\Delta t$ on recovery for the Δt algorithm, we see no difference between the approaches, since similar scenarios can be generated for either that would lead to lost or duplicate information if one or the other host crashed at a particular instant [2].

9. Conclusion

A timer-based approach to the development of a simple efficient protocol for dealing with omission, duplicate, and transposition detection problems was presented. A number of necessary conditions were discussed for its successful operation and rules presented that guaranteed satisfaction of these conditions. That these rules satisfied the required conditions was demonstrated with informal arguments. Future work would include implementation and experimentation. Additional analytic work leading to the development of both necessary and sufficient conditions for reliable communication and a formal proof that the rules of the Δt protocol met these conditions would also be useful.

The important assumptions requiring the bounding of certain times were discussed, particularly that of the maximum packet lifetime within the routing network. We argued that these times could be bounded and were small relative to the average number of connections per hour that are established, the amount of time it is reasonable to wait on crash recovery before sending or receiving packets, and the "wrap around" time for sequence numbers.

The Δt protocol mechanism that has resulted is simple and efficient for a wide range of applications including both single data message and datastream traffic. Because of the low message overhead for reliable single-message transmission, it seems very attractive in situations in which single messages are numerous, particularly in high-speed local networks.

Another current direction of our work is to consider the development of a hybrid protocol providing the advantage of both the Δt and threeway-handshake approaches with little increase in implementation complexity. Such a protocol would work with

maximum efficiency (minimum message overhead) in a local network environment when MPL can be bounded, and could also operate in an internetting environment involving vendor, commercial, or other geographically distributed or local networks where an MPL bound may not be guaranteed.

APPENDIX
Algorithms for a timer-based protocol

The protocol discussed in this paper, like any end-to-end protocol, is really defined by the intended interpretation of the fields in packet headings. Each of the partner ends of a connection may carry out any algorithms that are consistent with that interpretation. The set of protocol algorithms that follows is, therefore, only typical and necessarily involves some assumptions about matters that are external to the protocol and not required by it. An attempt has been made to keep such assumptions to a minimum. The algorithms do not constitute a complete end-to-end protocol but cover only matters relating to the prevention of the omission, duplication, and transposition of data elements.

The protocol algorithms manipulate information found in three places: in packet headings, in an interface to higher-level (user) algorithms that generate data to be sent and interpret data received, and in the connection record that is the algorithms' memory. In each of the three places, the information falls into two independent categories: that related to data sent and that related to data received. This information is now summarized. All items listed are logical concepts and should not be construed to imply an implementation.

Packet heading

Packet heading fields relevant to data moving in the same direction as the packets are as follows:

The *data sequence number* (DSN) is the sequence number of the first data element in the packet. If there is no data in the packet, no particular value for the DSN is required for the purposes of the present algorithms.

The *data run flag* (DRF) is a boolean indicator which, if on, signifies that the sender of the packet has received acknowledgment of all data elements preceding the one defined by the DSN. That is, the DSN is the same as the largest sequence number acknowledged to the sender or is the initial sequence number of a new connection (in which case no elements precede the one defined by the DSN). Only if this flag is on may the packet begin a new run of consecutively numbered elements.

The *length* (LEN) is the number of data elements in the packet. The elements in the packet have, in order, sequence numbers s in the range $DSN \leqslant s < DSN + LEN$.

Packet heading fields relevant to data moving in the direction opposite to the packet, that is, to data that has been received by the sender of the packet, are as follows:

The *acknowledged sequence number* (ASN) is the sequence number of the data element that the sender of the packet expects to receive next. The receipt of all preceding elements is acknowledged.

The *acknowledged run flag* (ARF) is a boolean indicator which, if on, signifies that the ASN in meaningless, that no elements are being acknowledged, and that the sender of the packet expects DRF to be on in the next packet it receives.

The *window* (WIN) is the number of data elements beyond those acknowledged by the ASN that the sender of the packet is prepared to receive. While this value is primarily of interest for flow control purposes, it is necessary to the algorithms discussed here for defining a valid range for the sequence numbers s of data elements to be accepted by the sender of the packet, namely $ASN \leqslant s < ASN + WIN$. (Since sequence numbers use modulo arithmetic, all inequalities must provide bounds from both sides.)

User interface

The interface to those user algorithms that send data must include the following:

The *output source* generates, in order, the data elements to be sent.

It is assumed here that the protocol algorithms are prepared to buffer only what they believe the partner will accept (as indicated by received values of WIN). Therefore, flow control is required at the user interface. This could be implemented in many ways, such as by a semaphore, by direct control by the protocol algorithms over the user algorithms' execution, or by a pair of pointers into a circular buffer. No matter how implemented, there is from a logical standpoint an *output window size*, a count of the number of additional elements that can be generated by the output source. This count is decremented as elements are generated and incremented by the protocol algorithms as buffer space becomes available. The algorithms given here adhere literally to the window defined by the partner, even through it might be reasonable to overrun it somewhat because the WIN values received are usually a little out of date. Also, the defined window size is never reduced because of a WIN value received from the partner, since this is viewed as an illegitimate renege on a promise.

A boolean *output error indicator* is turned on by the protocol algorithms to indicate that not all data generated by the output source was sent and acknowledged.

The interface to those user algorithms that receive data must include the following:

The *input sink* consumes in order (as determined by sequence number) the data elements received.

It is assumed here that the protocol algorithms are not prepared to buffer elements that fall outside an input window determined by flow control through the user interface. As with output this can be implemented in many ways. However implemented, there is logically an *input window size*, a count of the number additional elements that can be consumed by the input sink. This count is decremented as elements are consumed and incremented by the user algorithms as buffer space becomes available. A count of zero is the indication that the user algorithms are unwilling to receive; further indications in this regard (such as whether the unwillingness is permanent) are viewed as being outside the scope of the present algorithms.

A boolean *input error indicator* is turned on by the protocol algorithms to indicate that there is evidence that not all data sent by the partner was received. However, a failure of this nature may occur without leaving suitable evidence.

Connection record

The algorithms for sending data require the following to be remembered in the connection record:

The *output list* is the set of data elements that have been generated by the output source, sent zero or more times to the partner, but not yet acknowledged. With each element is associated its assigned sequence number, a count of the number of times that it has been transmitted, and a retry timer that counts down the interval until the next retransmission. (The scheme used here for giving up on data transmission is the one in which each element is sent a maximum number of times, rather than retransmissions being limited by an explicitly defined give-up interval.) Explicitly associating a sequence number, a retry count, and a retry timer with each element in the list is likely to be inefficient, and information compression techniques (such as treating many consecutively numbered elements as a unit) should probably be used.

The *output window left edge* is the lowest sequence number of an element in the output list. That element is the oldest unacknowledged element.

The *next output sequence number* is the sequence number to be assigned to the next element generated by the output source. The sequence numbers s in the output list are exactly those in the range,

output window left edge $\leq s <$ next output sequence number.

The output window right edge equals next output sequence number plus output window size.

The *send timer* (S_{timer}) counts down the interval until the sender portion of the connection record should be reinitialized.

The algorithms for receiving data require the following to be remembered in the connection record:

The *input list* is the set of data elements that have been received but not as yet delivered to the input sink. Unlike the sequence numbers in the output list, those in the input list do not necessarily form a consecutive run; there may be gaps because of transpositions of the received data. In fact, the possibility of gaps is the only reason for having an input list, since elements that arrive in sequence can be delivered to the input sink immediately. An implementation could eliminate the need for an input list by ignoring elements received out of sequence and relying on retransmissions by the partner to provide them again; this considerably simpler *strictly sequential option* is presented here as an alternative. With each element in the input list is associated its sequence number. As with the output list, information compression seems to be called for in an efficient implementation. Input list (and output list) elements could be buffered directly in space supplied by the user algorithms; however, in certain cases when transmitted data becomes transposed soon after the initialization of the connection, it may not be possible to avoid relocating elements after they have been placed into the buffer.

The *input window left edge* is the sequence number of the element expected to be received next; all preceding elements have been acknowledged to the partner. All sequence numbers s in the input list are in the input window defined by

input window left edge $\leq s <$ input window right edge,

which equals input window left edge plus input window size.

The boolean *run indicator* is needed for treating the possibility that the first packet sent on a new connection gets transposed and is received after a subsequent packet. It therefore is not needed if the strictly sequential option is used. If on, it signifies that no packet with DRF = on has been received since the receive connection has initialized.

The boolean *respond indicator*, if on, signifies that a packet should be sent to the partner for the purpose of conveying information about data received.

The *receive timer* (R_{timer}) counts down the interval until the receiver portion of the connection record should be reinitialized.

Algorithms

The algorithms are given in a dialect of colloquial Algol. The meanings of the reserved (italicized) words are conventional. The scope of a construct is indicated solely by indentation.

Figs. A1 and A2 outline the protocol and list structure.

Generate connection record

```
procedure for connection record generation:
    comment This procedure is called implicitly whenever
        reference is made to the connection record and it
        does not exist.
    if no connection record exists,
    then
        Allocate space for the record
        output list ← empty.
        output window left edge ← anything.
        next output sequence number ← output win-
            dow left edge.
        send timer ← 0.
        input list ← empty.
        input window left edge ← anything.
        run indicator ← on.
        respond indicator ← off.
        receive timer ← 0.
        comment It is assumed that the user interface,
            including the window sizes, already exists.
end
```

Abandon connection record

```
procedure for connection record abandonment:
    comment This procedure should be called only when
        there is need for the space occupied by the connec-
        tion record.
```

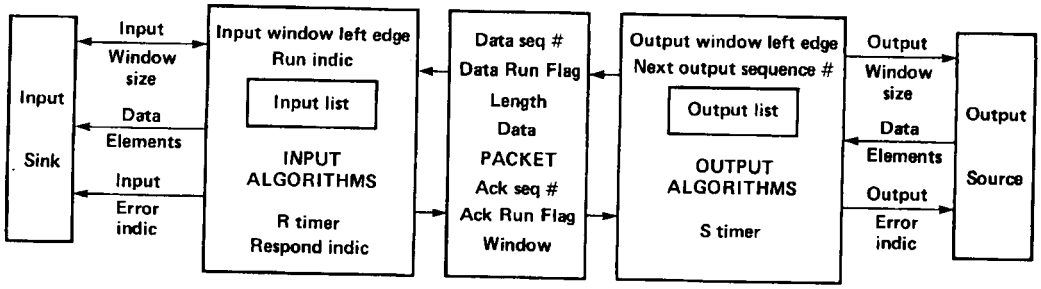


Fig. A1. Overview block diagram of the simplex Δt protocol.

if send timer = 0 *and* receiver timer = 0 *and*
respond indicator = off,
then Deallocate space for the record.
end

Initialize sender part of connection record

procedure for sender connection record initialization:
comment This procedure is called when the send timer becomes zero.
if send timer = 0,
then
if output list ≠ empty,
then
output error indicator ← on.
output list ← empty.
respond indicator ← on. *comment* The response may include an error indication, which is regarded as outside the scope of these algorithms.

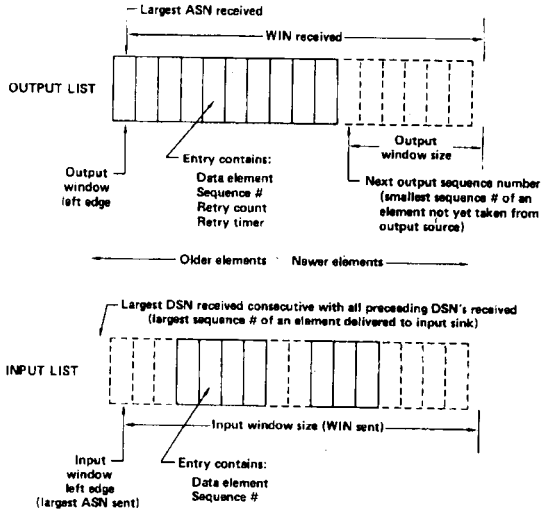


Fig. A2. Output and input lists and associated information.

output window left edge ← next output sequence number.
end

Initialize receiver part of connection record

procedure for receiver connection record initialization
comment This procedure is called when the receive timer becomes zero. It is not needed if the strictly sequential option is used.
if receive timer = 0,
then
if input list ≠ empty,
then
input error indicator ← on.
input list ← empty.
respond indicator ← on. *comment* The response may include an error indication.
run indicator ← on.
end

Getting data through the user interface

procedure for taking elements from output source:
comment This procedure is called when an element is available from the user program.
if an element is present at output source
and output window size > 0,
then
Remove the element from the output source and place it into the output list.
its sequence number ← next output sequence number.
its retry count ← 0.
its retry timer ← 0.
next output sequence number ← next output sequence number + 1.
output window size ← output window size - 1.
end

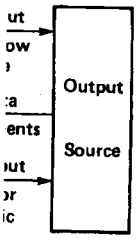
Generate a packet

procedure for generating a packet:
comment This procedure is called and the packet sent,

perhaps after a
should not ex
defining Δt), v
ready to be se
addition to the
the respond in
window size is
if no retry coun
least one retr
means that th
then
DSN ← lea
ready-t
LEN ← the
and the
ments
beginni
Copy into
defined
sequen
+ LEN)
retry cour
elemen
retry time
elemen
send timer
else
DSN ← ne
LEN ← 0.
if DSN = output
then DRF ← c
else DRF ← of
ASN ← input win
WIN ← input win
if receive timer =
then ARF ← c
else ARF ← of
respond indicator
end

Receiving a packe

procedure for treatin
comment This p
received.
comment The f
acknowledged
if ARF = off *an*
output list, th
edge < ASN ≤
then
Discard fi
elemen
satisfyi
output wi
if ARF = on *or*
then output
window si
left edge -
comment The 1



perhaps after a brief delay (which in certain instances should not exceed the unacknowledged time used in defining Δt), whenever an element in the output list is ready to be sent and/or the respond indicator is on. In addition to the occasions cited in the other algorithms, the respond indicator is turned on whenever the input window size is increased by the user algorithms.

if no retry count in the output list = maximum and at least one retry timer in the output list = 0 (which means that the associated element is ready-to-be-sent),
then
DSN ← least sequence number of an element ready-to-be-sent.
LEN ← the smaller of the maximum packet size and the number of consecutively numbered elements in the output list ready-to-be-sent, beginning with the element numbered by DSN.
Copy into the packet in order all the elements defined by DSN and LEN (that is, with sequence numbers s satisfying DSN ⩽ s < DSN + LEN), all of which are ready-to-be-sent.
retry count ← retry count + 1, for each of those elements.
retry timer ← retry interval, for each of those elements.
send timer ← 3 Δt.
else
DSN ← next output sequence number.
LEN ← 0.
if DSN = output window left edge,
then DRF ← on.
else DRF ← off.
ASN ← input window left edge.
WIN ← input window size.
if receive timer = 0,
then ARF ← on.
else ARF ← off.
respond indicator ← off.
end

Receiving a packet

procedure for treating a received packet:
comment This procedure is called whenever a packet is received.
comment The first part of the procedure treats data acknowledged by the packet.
if ARF = off *and* ASN acknowledges an element in the output list, that is, satifies output window left edge < ASN ⩽ next output sequence number,
then
Discard from the output list all acknowledged elements (those with sequence numbers s satisfying output window left edge ⩽ s < ASN).
output window left edge ← ASN.
if ARF = on *or* output window left edge = ASN,
then output window size ← the larger of output window size and the sum WIN + output window left edge - next output sequence number.
comment The last part of the procedure treats data included in the packet. It has one of two forms, depending on whether the strictly sequential option is used.
comment The simpler strictly sequential option is presented first.
if LEN > 0,
then
if receive timer = 0,
then if DRF = on,
then input window left edge ← DSN.
else input window left edge ← DSN + LEN (thereby assuring that the next test will fail).
if input window size > 0 *and* the sequence number of some element in the packet (equal to its position in the packet + DSN) = input window left edge, that is, DSN ⩽ input window left edge < DSN + LEN,
then
Place into the input sink in order all $n > 0$ elements from the packet that lie in the input window that is, have sequence numbers s satisfying input window left edge ⩽ s < the smaller of input window left edge + input window size and DSN + LEN.
input window left edge ← input window left edge + n.
input window size ← input window size - n.
receive timer ← 2 Δt.
respond indicator ← on.
comment The preceding statement used with the strictly sequential option is replaced by the following if the more complex option, which does not ignore elements arriving out of sequence, is used.
if LEN > 0,
then
if run indicator = on (which implies that no data with DRF = on has been received since the last initialization of the receiver connection, which emptied the input list and turned on the run indicator when the receiver timer went to zero).
then
if receive timer = 0 (which means that no data at all has been received since the last initialization of the receiver connection),
then
if input window size > 0,
then receive timer ← 2Δt.
input window left edge ← DSN.
else if DSN does not lie in the input window defined by input window left edge ⩽ DSN < input window left edge + input window size (which means that the arriving data has been transposed with all the data that has already arrived since the last initialization of the receiver connection),
then input window left edge ← DSN.
if DRF = on *and* input window size > 0,
then run indicator ← off.

Place into the input list in order all the elements in the packet (equal in number to LEN), each associated with its sequence number (equal to its position in the packet + DSN), discarding any duplicates already in the input list.

if any sequence numbers s in the resulting input list do not lie in the input window defined by input window left edge ⩽ s < input window left edge + input window size.

then

Discard from the input list all elements associated with such sequence numbers.
respond indicator ← on.

if run indicator = off and there are n > 0 consecutive sequence numbers in the input list beginning with the input window left edge,

then

Remove from the input list in order the associated n elements and place them into the input sink.
input window left edge ← input window left edge + n.
input window size ← input window size - n.
receive timer ← 2Δt.
respond indicator ← on.

end comment end of algorithms

References

[1] E. Akkoyunlu, A. Bernstein, and R. Schantz, "Interprocess Communication Facilities for Network Operating Systems," Computer 7, 6 (1974).

[2] D. Belsnes, "Single-Message Communication," IEEE Transactions on Communications COM-24, No. 2 (1976).

[3] J. Burchfiel et al., "Proposed Revisions to the TCP, "INWG Protocol Note No. 44, September 1976.

[4] V. Cerf and R. Kahn, "A Protocol for Packet Networks Intercommunication," IEEE Transactions on Communication COM-20, No. 5 (1974).

[5] V. Cerf, A. McKenzie, et al., "Proposal for an International End to End Protocol," Computer Communication Review 6, 63-89 (1976).

[6] V. Cerf "Specification of Internet Transmission Control Program," TCP Version 2, March 1977.

[7] D. Clark, "Comparison of TCP and DSP," MIT Laboratory for Computer Science, Local Network Note No. 7, April 1977.

[8] S. Crocker and J. Postel, private communication, Reliable Transfer of Control Information, August 1977.

[9] Y. Dalal, "More on Selecting Sequence Numbers," INWG Protocol Note 4, October 1974. Also in Proceedings ACM SIGCOMM/SIGOPS Interprocess Communications Workshop, March 1975.

[10] J.E. Donnelley, "A Distributed Capability Computing System," Proceedings Third International Conference on Computer Communication, August 1976.

[11] J. Fletcher, Lawrence Livermore Laboratory, private communication (1977.)

[12] J. Fletcher, "The Octopus Computer Network," Datamation, April 1973.

[13] L. Garlick, R. Rom, and J. Postel, "Reliable Host to Host Protocols: Problems and Techniques," Proceedings 5th Data Communications Symposium, IEEE/ACM, September 1977.

[14] A.A. McKenzie, "A Host/Host Protocol," ARPA Network Working Group RFC 714, NIC 35144, April 1976.

[15] R. Metcalfe, "Packet Communication," MIT MAC TR-114, NTIS AD-731430, Ph.D. Thesis, May 1973.

[16] Network Systems Corporation, "System Description Series A Network Adapters," Pub. No. A01-0000-01, October 1976.

[17] P. Johnson et al., "MSG: The Interprocess Communication Facility for the National Software Works," Computer Associates CADD-7601-2611 (1976).

[18] L. Pouzin, "Flow Control in Data Networks - Methods and Tools," Proceedings Third International Conference on Computer Communication, Toronto, August 1976.

[19] D. Reed, "Protocols for the LCS Network," MIT Laboratory for Computer Science, Local Network Note No. 3, November 1976.

[20] D. Reed, "The Initial Connection Mechanism in DSP," MIT Laboratory for Computer Science, Local Network Note 10, September 1977.

[21] C. Sunshine, "Interconnection of Computer Networks," Computer Networks 1, (1977) 175.

[22] C. Sunshine, "Factors in Interprocess Communication Protocol Efficiency in Computer Networks," AFIPS Conference Proceedings 45, 571-576 (1976).

[23] R. Tomlinson, "Selecting Sequence Numbers," INWG Protocol Note No. 2, September 1974. Also in Proceedings SIG COMM/SIGOPS Interprocess Communications Workshop, March 1975.

[24] D. Walden, "A System for Interprocess Communication in a Resource Sharing Network," Comm. Assoc. Comput. Mach. 15 221-330 (1972).

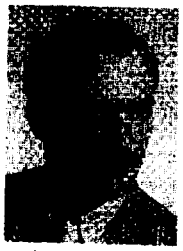
[25] J. White, "A High-Level Framework for Network-Based Resource Sharing," AFIPS Conference Proceedings 45, 561-570 (1976).

A Virtu the "C

Günter Schulz
GMD-IFV, Darmst
and
Joachim Börge
WRB-BERNET, Be

The ideas prese within the PIX wo "Virtual Terminal the PIX virtual between several ne area in the FRG, s Technology. The interprocess comm used to fit the req ing class of termin whereas the protoc defining the appr The complexity of depends upon the example, a protocol mode terminal is gi

Keywords: Com ted term tion



projects for the c switching networks.