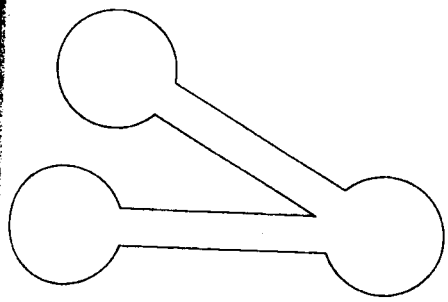
Mechanisms for a Reliable Timer-Based Protocol

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:

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

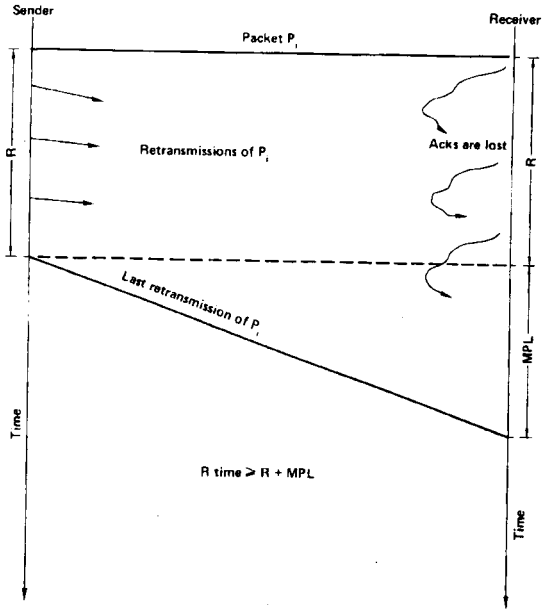


Fig. 1. Worst-case time for duplicate detection.

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