Timer-Based Mechanisms in Reliable Transport Protocol Connection Management

Richard W. Watson
University of California, Lawrence Livermore Laboratory, Livermore, California 94550, USA

There is a need for timer-based mechanisms (in addition to retransmission timers) to achieve reliable connection management in transport protocols designed to operate in a general network (datagram and internetwork) environment where packets can get lost, duplicated, or missequenced. This need is illustrated by discussing the timer mechanisms or assumptions (1) in the Department of Defense Transmission Control Protocol, initially designed using only a message exchange mechanism; and (2) in the Lawrence Livermore Laboratory Delta-t protocol, designed explicitly to be timer based. Some of the implementation and service implications of the approaches are discussed. The bounding of maximum packet lifetime and related parameters is important for achieving transport protocol reliability and a mechanism is outlined for enforcing such a bound.

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.

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 and from 1971–1976 with SRI International'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 valuable conversations on these issues with John Fletcher and Lansing Sloan at LLNL and with Jon Postel and others in the ARPA TCP community. The work performed here was under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

Computer Networks 5 (1981) 47–56

1. Introduction

Protocols at every level of a network architecture deal with two types of information, *control* and *data*. A protocol provides services with respect to the data. Control information is sent between and interpreted by the cooperating protocol modules to support the protocol's service. Mechanisms at any protocol level designed to deal with error control or *assurance* issues such as misaddressed, lost, damaged, duplicated, or missequenced information are based on

(1) identifying the data or control units being error controlled;
(2) reliably transmitting control or redundant information;
(3) reliably initializing, maintaining, and terminating state information at each end;
(4) assumptions made about the environment provided by the supporting lower level protocols and communication technology.

These mechanisms are often based on explicit or implicit (often unconscious) assumptions about bounds on certain time intervals. While the discussion in this paper focuses on transport level protocols, many of the issues discussed are also applicable to protocols at other levels.

Transport protocols designed for reliable transmission provide one or more error controlled channels between transport address pairs (*associations*). While state information at each end of an association is maintained for any purpose, a *connection* is said to exist. The state information is maintained in *connection records*. An important aspect of assurance is proper synchronization and evolution of state information in connection records in the face of arbitrary transmission delays, errors, and end-node crashes and deadstarts. This process is called connection management. In this paper connection management mechanisms used in two protocols, the Department of Defense Transmission Control Protocol (TCP) [5], and the Lawrence Livermore Laboratory (LLL) Delta-t protocol [3,8], are compared to illustrate the role timer-based mechanisms play in achieving a reliable transport protocol, to demonstrate the requirement for bounding factors affecting the life-

time of assurance identifiers, and highlight some of the implementation and usage implications of different connection management mechanisms. Both the TCP and Delta-t protocols were designed explicitly to operate in a general network (internetwork and datagram) environment where packets can be duplicated, lost, damaged, or missequenced. In the following discussion, although some background is provided, we assume that the reader is familiar with the basic issues of connection management and reliable transport protocol design and the fundamentals of the TCP and Delta-t assurance mechanisms [3, 4,7].

2. Background

There are three logical phases in connection management (explicit phase separation is not required): (1) initializing (*opening*) the connection records at each end, (2) evolving the state during ongoing data transfer, and (3) resetting or terminating (*closing*) state information when no further data requires transferring. During the reliable opening of a transport protocol assurance connection, the main problem is establishing initial error control identifiers, usually *Sequence Numbers* (SNs) or SNs qualified by connection identifiers, meeting the following conditions:

O1: If no connection exists and the receiver is willing to receive, then no packets from a previously closed connection should cause a connection to be initialized and duplicate data to be accepted.

O2: If a connection exists, then no packets from a previously closed connection should be acceptable within the current connection.

In a graceful close of a connection, each side must know that the other side has received any data sent. This implies two conditions:

C1: A receiving side must not close until it has received all of a sender's possible retransmissions and can unambiguously respond to them, and

C2: A sending side must not close until it has received an Acknowledgment (Ack) for all its transmitted data or allowed time for an Ack of its final retransmission to return before reporting a failure.

We believe a graceful close is important in an assurance protocol to limit those situations where ambiguity can exist as to whether data sent reached the other side. In particular we would like to limit ambiguity to end-node crash or network partition events. Ambiguity exists if a sender does not receive an Ack of some data sent. It will not know whether the data or Ack got lost. Depending on how it responds, lost or duplicate data may result. Ambiguity may thus require more expensive higher level error recovery mechanisms.

Most of the design decisions for a complete transport protocol are independent of the choice of a connection management mechanism, e.g., addressing, flow control, options, choice of basic transport data unit(s), etc. Section 7 discusses one area where the connection management mechanism may influence another protocol design decision. Many design decisions associated with assurance issues are also independent of the connection management mechanism. For example

- What is the meaning of an Ack? Does it mean that the data reached the destination protocol module or that the data were placed safely in the destination user's buffer?
- What is the unit and identification scheme used for error control?
- Should header, packet, or end-end checksums be used to protect against damaged packets?

Most importantly, we believe, all reliable transport protocls require explicit or clearly understood implicit bounds on *three* lifetime factors:

- *MPL,* the maximum time an identified data or control unit can exist within the routing network, which includes time in origin and destination protocol modules. At the transport level this is often called maximum packet lifetime (MPL). At other levels, it would be maximum frame, message, or other unit lifetime.
- *R,* the maximum time a sender requiring a positive acknowledgment will keep trying to retransmit a unit before ceasing retransmission.
- *A,* the maximum time a receiver will hold a unit before sending an Ack.

One can provide a mechanism to bound these factors explicitly or make "reasonable" assumptions about their bounds. These quantities can be bounded either individually [3] or as a sum [8]. The values(s) assumed in each data flow direction may be the same [3] or different [8]. In this paper we deal with these quantities as if bound separately.

In its first versions, TCP initiated a connection by having the first packet of a connection contain a syncronization flag (*syn*) indicating to the receiver that connection SNs would start with the one in this packet. This mechanism did not satisfy condition O1

and quickly led to difficulties with duplicates. An extended message exchange mechanism was introduced satisfying O1. This mechanism requires an end to receive an Ack of its connection opening SN before it will accept data. This exchange, illustrated in Section 4, is called a *three-way handshake* [4,7]. As TCP's designers came to more fully understand connection management hazards, time-based and other mechanisms were added. The mechanisms implicitly assume bounds on MPL, R, and A.

Delta-t, using the TCP experience, was designed from the beginning with connection management based solely on a timer mechanism. This mechanism explicitly recognizes the necessity to bound the quantities MPL, R, and A. Delta-t maintains connection records while any data unit or Ack (including duplicates) with a given identifier can be generated or exist on a given association.

In comparing protocol assurance mechanisms, one needs to ask two questions: (1) whether for an equivalent level of assurance one is more complicated or uses more resources than the other, or (2) for a given set of design decisions whether one is more complicated, uses more resources, or gives more assurance. One also needs to examine other service implications of each approach such as delay or throughput.

Our contention, illustrated with the two protocols discussed, is that because there is something fundamental at the transport level about the requirement to bound MPL, R, and A to achieve assurance, a transport protocol explicitly based around this notion should be simpler for a given set of equivalent design assumptions. The potential simplication in packet acceptance testing could result in higher throughput [2]. It also allows a reliable process-to-process, transaction-oriented (low delay) service to be efficiently supported. Delta-t is currently being implemented. Although as this is written we cannot refer to operational experience in comparing Delta-t with TCP, we feel it is important to present the following connection management issues because of the current level of national and international activity in the transport protocol design area.

3. Achieving Reliable Connection Management

Perfect error control is only possible if three assurance conditions are met:

A1: An identifier of an information unit used for assurance is never reused while one or more copies (duplicates) of that unit or its Ack exist.

A2: The assurance state or redundant information maintained at each end is never lost or damaged.

A3: The assurance control information transmitted between each end is itself perfectly error controlled.

Let us now consider how each protocol deals with these conditions and relate them to the opening and closing conditions of the previous section. Condition A1 implies opening condition O2 and that within a connection the size of the SN field, n, must satisfy an SN field-size inequality. Also implied, if A2 cannot be met, is that there must be care in restarting after a crash.

An example SN field size inequality is:

$$2^n > (2\text{MPL} + R + A)\, T,$$

where R belongs to the origin desiring an Ack, A belongs to the destination, and T is the maximum rate at which SNs can be generated at the origin. This is usually the maximum transmisssion rate, but if SNs can be skipped, such as by TCP's "*rubber EOLs*" [5] or Delta-t's window overrun mechanism [8], these must be considered. The assumption on which this inequality is based is that while possibly retransmitting one unit, units with new SNs are transmitted at the maximum rate. If a different retransmission/transmission strategy were used, a different function would result. This particular SN field-size inequality results because in the worst case an SN of a unit or its Ack can live for the following periods. A unit with a given SN could continue to be emitted from the sender for time R (because all but the last retransmitted instances were lost, for example). The last instance could take MPL to reach the destination. The destination could delay A before sending an Ack, and the Ack could take MPL to reach the origin. During this interval new SNs are assumed to be generated at the rate T.

This result illustrates that maximum transmission rate, reliable transmission, the transmission strategy, and the lifetime factors are intimately interrelated and must be carefully considered and be made explicit in choosing n. Too small an n could limit reliable throughput, particularly serious if used in a high bandwidth local or satellite network.

We now examine requirements for the two protocols to meet conditions O2. Because TCP does not use connection identifiers and can close a connection while duplicate packets can still exist within the net-

work, it requires a special mechanism to choose initial *sequence numbers* (ISNs) larger (modulo 2^n) than any used in recent connections. The accepted mechanism to do this, not requiring explicit per association state to be safely maintained across a crash [4], maps a monotonically increasing clock value into the ISN value. The clock must continue to operate across a crash. Because both the clock and SN fields can wrap around, there are critical relationships between the quantities (MPL, R, A), the rate at which SNs are being generated, the clock rate, and the SN and clock field size that can introduce hazards resulting from crashes due to SN reuse while units or Acks with that SN still exist. The TCP designers extensively analyzed these hazards [4,7]. After considering various dynamic timer oriented mechanisms, the TCP designers chose to omit them and instead chose the following simpler scheme. An origin waits an interval after a crash before opening a connection to allow time for all packets from any previous connection to die. Therefore, there will be no risk of a packet for the new connection having an SN used on a packet or Ack of a previous connection. For perfrect assurance this interval must satisfy the inequality

Sending Recovery Interval $> 2\text{MPL} + A$.

This inequality satisfies the worst case that results when the origin sent a packet just before it crashed with an SN that might be used in a new connection to be established after deadstart and then recovered instantly. The last packet before the crash takes MPL to reach the destination, the destination waits A to emit an Ack, and the Ack takes MPL to reach the origin. In the networks in which TCP operates, the interval MPL is only approximately bounded and can be very large (several hours), although the probability is quite low that packets will live beyond 30 to 60 sec. Therefore, the engineering decision was made to wait about 2 min after a crash. No Receiving Recovery Interval is required because TCP's three-way handshake and reset mechanisms handle recovery from half open connections [7].

Delta-t maintains state under time control to meet condition O2. In Delta-t any ISN can be used because if no active (timers >0) sending or receiving connection record exists for an association, it is guaranteed by the Delta-t timer intervals and rules (assuming MPL, R, and A are bounded) that no data or Ack packets from a previous connection exist [3]. Delta-t senders and receivers must wait an interval after a crash before initiating or accepting a conversaton. For senders it is:

Sending Recovery Interval $>3\text{MPL} + R + A$.

This interval ensures that all data or Ack packets from the previous connection have expired and that no receiver connection record exists (removing half open connections), so that any ISN will be acceptable. For receivers, the Receiving Recovery Interval is:

Receiving Recovery Interval $>2\text{MPL} + R + A$.

This interval guarantees that no unit with a given SN accepted before the crash will still exist. We include motivation for the TCP timer interval because we have not seen an explicit analysis in the TCP literature.

In summary, to meet A1 and thus O2, both protocols require satisfaction of an inequality on SN field size and must wait an interval after a crash. TCP requires an additional clock based mechanism for ISN selection.

If A2 is met, the state information in connection records would never be lost or damaged. This condition is difficult or impossible to satisfy. TCP, and Delta-t have the same issues and the same hazards here because of hardware errors, software bugs, or crashes with loss of memory. The implication of the latter was discussed above.

A3 requires that control information (e.g. that used for connection opening and closing) must itself be error controlled or not be affected by errors. TCP uses special packet header flags to indicate connection opening and closing packets, called *syn* and *fin* respectively. TCP protects its syn and fin flags against loss and duplication by placing them in the data SN space. In Delta-t there are no open or close flags corresponding to TCP's syn or fin flags. There is instead a *data-run flag* (DRF) used to signal the receiver that all previous SNs sent have been Acked. DRF does not require an Ack and is used to aid handling packet missequencing when a connection does not yet exist [3].

We now discuss how each protocol meets condition O1 (Section 2). TCP achieves a reliable connection open by having each side require the other to acknowledge its syn (ISN) before it will accept data. In effect, when an end receives a syn it asks the other if it is current or an old duplicate before it will establish a connection.

Delta-t achieves a reliable open by assuring that no

packets from a connection continue to exist before it discards state. In effect, Delta-t waits for the connection to be flushed before state is discarded (data transfer, however can be carried out at any time with no waiting interval).

We now consider connection closing. To achieve a smooghly running protocol and meet closing conditions C1 and C2, Delta-t requires receivers to maintain state for $2MPL + R + A$ from the time of their last Ack of a new SN and senders to maintain state for $3MPL + R + A$ from the time the last new SN was sent [3,8]. If new data are received or sent while state exists, Delta-t's receive and send timers and state are updated, automatically increasing the life of the connection.

To achieve a reliable close, TCP requires fins to be Acked, a *fin-ack*. In any protocol the last message cannot be critical, because it is not Acked. Because of the possibility of lost fin-acks, there is an ambiguity hazard that TCP defends against by requiring the side sending the last fin-ack to wait until all possible retransmissions of its partner's fin packet reach it before discarding its connection state. The assumption is that the fin-ack may also Ack data, otherwise no hazard exists. This holding of state is required to satisfy condition C1 for a graceful close. If the sender of the last fin-ack closed immediately after emitting the fin-ack (implicitly Acking data as well), and its fin-ack got lost, then the other side would timeout and retransmit the unAcked data and fin. The closed side on receiving the retransmitted information can only generate a *reset* packet indicating it was closed. The receiver of the reset would then not be able to distinguish this case where there was successful delivery of data, but a lost Ack, from the case of a crash or network failure with lost data. This ambiguity could cause unnecessary confusion possibly requiring entry to expensive higher level error recovery procedures. Therefore, the sender of the last fin-ack must wait an interval,

Final Close Interval $>MPL + R$,

inorder to assure it can respond to all its correspondents retransmitted fins, where R is that used by the sender of the last packet with a fin. This inequality results because, in the worst case, the final fin packet arrives at its destination instantaneously and Acks to it and all retransmissions but the last get lost, and the last retransmission takes MPL, to arrive. This TCP close-timer is similar to the Delta-t receive-timer.

We have not seen condition C2 discussed in the TCP literature, i.e., how long should the TCP module wait after emitting its last retransmission of a packet and without receiving an Ack before reporting failure to its user program? The answer is the timer interval

Giveup Interval $>2MPL + A$.

This condition results because the last emitted packet can take MPL to reach the destination, the destination can take A to emit the Ack, and the Ack can take MPL to reach the origin of the data packet. If a problem is reported before this Giveup Interval, the user process may create an unnecessary duplication or the user process may unnecessarily enter an involved error recovery procedure to deal with what appears as a possible partner crash or network failure. This interval was considered as one condition on Delta-t's send-timer value.

In summary, TCP uses three separate connection management mechanisms to achieve assurance: (1) exchange of connection management (syn, fin) messages and associated state transitions, (2) careful choice of an ISN, and (3) assurance timers with intervals containing assumptions about the bounds on MPL, R, and A. Delta-t uses a single timer mechanism with assumptions about bounds on MPL, R, and A.

In comparing TCP with Delta-t we see the following. TCP requires additional complexity to deal with special states to synchronize syn and fin and carefully choose an ISN. TCP also requires a reset mechanism, not required in Delta-t. These mechanisms require additional packet acceptance overhead during normal data transfer as well [2]. If senders are allowed to overrun a receiver's flow-control window, Delta-t requires an extra message, not needed in TCP, to resynchronize SNs [8].

The final question is, if explicit bounds on MPL, R, and A are not guaranteed and we rely on an engineering decision (mentioned earlier), then is one or the other protocol safer? Both rely on these being bounded in determining maximum SN generation rate, at crash recovery, and at connection close. TCP, in addition, relies on bounding these quantities in ISN selection. Delta-t, in addition, relies on bounding these factors for a reliable open. Because Delta-t relies on the bound to meet condition O1 and TCP does not, some extra hazard exists. The level of hazards depends on the percentage of packets that can live beyond the assumed bound. We believe that it would be best if all protocols bound MPL, R, and A. These quantities can be bound with support from the lower environment as described in Section 6.

4. Transaction Applications

We believe a useful design goal is to minimize the total number of messages and the delay needed to reliably send a single data message. This results in minimizing network traffic and increasing system performance. There are many important transactional applications, such as distributed operating systems, where a request may be sent, a reply may be received, and no further conversation is required [9]. It is important to minimize messages or packets at the transport level, because each packet at this level may require that many frames or blocks be transmitted by lower-level link or network-interconnection technologies. There is increased delay plus considerable network and host overhead in handling each packet, frame, or block. We want to minimize the number of such data units that all parties must handle. A datagram service can meet this need; however datagrams are not reliable, and many transaction-oriented applications could benefit from a reliable transaction-oriented transport mechanism.

To reliably send a single data packet, Delta-t only requires two packets to be exchanged, the data packet and its Ack. TCP requires a minimum three-message exchange procedure to send one data message, but there is a hazard inherent in this procedure not present in Delta-t. This hazard does not exist in TCP if it uses five message to send one data packet. The three-message procedure also requires a special case in the TCP rules and implementation, making writing the specification more complex.

The three TCP messages are, assuming node A sending to node B:

$M_1(A \text{ to } B)$ ISN_A, syn, data, fin

$M_2(B \text{ to } A)$ ISN_b, syn, fin, Ack (syn, data, fin)

$M_3(A \text{ to } B)$ Ack (syn, fin),

where the elements in parentheses are the elements being Acked. The TCP module at B cannot deliver the data in M_1 until it receives M_3, because it is not until that point that it knows M_1 was not an old duplicate from a previous connection. Holding the M_1 data until the arrival of M_3 is a special case not needed for succeeding data in a longer connection.

A hazard results because if M_3 or any retransmission of M_3 triggered by retransmissions of M_2 does not get through, the data in M_1 cannot be delivered. Yet M_1 was Acked, and A therefore was misinformed. Basically, this hazard arises because no protocol can support positive acknowledgment with retransmission of the last message. This hazard can be removed with a wait between connections or a five-message procedure [1]:

$M_1(A \text{ to } B)$ ISN_A, syn

$M_2(B \text{ to } A)$ ISN_B syn, Ack (syn)

$M_3(A \text{ to } B)$ data, fin, Ack (syn)

$M_4(B \text{ to } A)$ fin, Ack (data, fin)

$M_5(A \text{ to } B)$ Ack (fin)

There is another hazard in the three-message procedure if B crashes after it has sent M_2 and before it delivers the data. This hazard exists for any protocol if the design decision is that an Ack means delivery to the protocol module rather than to the user. Even in the latter case, if the user's buffer is in main memory at the time of a crash, it could be lost before it was acted on. Perfect error control is impossible at any level, but we believe our designs should reduce the probability of a problem. In either the three- or five-message exchange cases above, TCP requires a transaction delay three times that of Delta-t, because data in TCP can only be delivered after the exchange of three messages. Independent of the total number of messages necessary to reliably send a single data message, the delay penalty required in a message exchange-based connection management mechanism could place serious performance limitations on a transaction-oriented application such as a distributed operating system.

5. Required Connection Record Resources

It was suggested that Delta-t might use more connection-record resources than TCP because it must hold inactive connection recrods while they time out [7]. But TCP also needs to hold connection records, as discussed earlier, after a fin-ack and preferably after the last retransmission of a packet. Delta-t requires both ends of an association to hold connection records under timer control.

The connection records required by both protocols should be about the same size. For reasonable assumptions about numbers of connections per hour and values for MPL, R, and A (comparable for each protocol with the magnitude of A dependent on the assumption about the meaning of an Ack), the number of connection records held under timer

control waiting to close is quite small and about the same in each case and not significant.

The sum of the lifetime factors can be bound to an arbitrarily small value even if the underlying network has a large MPL (Section 6). The value chosen for this sum must reflect realistic times for R, A, and the transmission time across the network to avoid unnecessarily destroying packets because their lifetime ran out too soon. For example, for a large internetwork, and assuming that units are Acked on reaching the destination protocol module, binding their sum to be 60 sec might be reasonable. Then assuming a given host might have 600 connections per hour, there will be on average about 10 to 30 inactive connection records waiting to timeout for each protocol. In a local network transaction environment there will be considerably more connections per hour, but now the lifetime sum above will be much smaller. Therefore the connection record time product should be about the same.

One of Delta-t's design goals was to remove connection management as a user interface issue. That is, connection opening and closing primitives are not required at the interface as, we believe, they provide no essential user service. Their main function is to aid the underlying implementation manage connection record, buffer, or identifier resources. Delta-t's connection management mechanism allows an implementation to simply reclaim inactive connection record or buffer space when timeouts occur due to lags in the conversation. If there are lags in the conversation, TCP could also reclaim space, but the required fin and syn exchange overheads to close and start up again probably discourage this, particularly if connection opening and closing are visible at the interface. Therefore, TCP may actually use more connection record resources than Delta-t.

6. Bounding MPL, R, and A

The lifetime of a packet can be strictly controlled to any desired value by including a lifetime field in the packet header initialized by the origin and counted down by the intermediate and end nodes. For a TCP environment this field is already in the Internet Protocol header, but is, we understand, currently used for a hop count rather than actual lifetime [5]. As packets pass through each node, including the origin and destination protocol modules, they must count down the lifetime at least once, more if they hold the packet longer than one time unit (tick). Retransmissions can enter the network preaged if the sum (MPL + R + A) is bound. The packet is discarded when the lifetime field reaches zero before the packet is accepted or Acked. This mechanism requires nodes to know how long they hold a packet and how long a packet has spent on the previous link. Knowing the latter may be useful for routing, congestion avoidance, or other reasons as well as for assurance. Nodes are assumed to destroy all packets on recovery from a crash as part of the lifetime bounding process.

Nodes can easily compute the time they hold a packet, and computing a packet's time on the previous link is also simply accomplished [8]. If the link is a physical link without buffering, it is calculated easily based on transmission speed and packet size. If the link is a logical link, such as a network with an advertised guaranteed bound on MPL, this value can be used if small enough.

If the link is a link that can hold a packet for an indefinite time period, then the lower level protocols used on the link can be layered with a *Link Transit Time Protocol* (LTTP) that, as its only service, guarantees a conservative estimate of the logical link transit time. A LTTP can also be layered on a physical link level protocol (e.g., HDLC) if it does not report the above times or they cannot be safely estimated. The basic idea is that each link frame is time stamped by the sender so that the transit time can be computed by the receiver. Conceptually, each LTTP end of a full duplex link has a *link-send* and a *link-receive* timer (not to be confused with the timers needed in TCP or Delta-t). On link initialization each LTTP receiver module sends its link-receive time value to the other end's sender link module, which uses this value to initialize its link-send timer. (No other synchronization is required, although the clocks at each end are assumed to run at approximately the same rate.) The sender places its current link-send timer value in each LTTP logical link frame. The receiver subtracts this value from its link-receive timer value to estimate time on the link. This mechanism assures that the transit times are always overestimated, never underestimated, because of the frame delay in the original initialization exchange.

The initialization exchange above assumed that the logical link, while introducing frame delays, was error free. In practice this initialization must be accomplished in the face of lost, damaged, and duplicated frames and end node crashes. The exact details of an LTTP link-connection management procedure depend

on the assumptions that can be made about the link's error characteristics. Sloan presents an LTTP for a link in which frames cannot be missequenced and can be flushed [6]. Sloan estimates that with the drift specification common on most current line frequency clocks, reinitialization every 30 min is a conservatively safe interval [6]. If we assume the link transit time is measured to the nearest 1/30 sec, then a 16-bit LTTP time field is sufficient.

If the link cannot be flushed and all types of frame errors can occur, then a more careful initialization procedure is required. Initialization must take into account the possible existence of initialization and other frames from previous instances of link connections. To deal with this problem we need each instance of a link connection to have a unique identifier. In other words, link identifiers cannot be reused while any frames or their Acks from a previous link connection with the same identifier are still alive. An 8-bit field would provide over a 4-h interval for such a frame/Ack maximum lifetime assuming that link connections were not generated faster than once a minute. Normally we would expect to generate new link connections about every 30 min to correct for clock drift, and therefore this worst case of one link connection every minute would only happen in a long sequence of crashes and restarts. One would hardly expect such a sequence for the time interval assumed. Therefore, protection extends to tens of hours. For the identifiers to be unique across crashes, the previous identifier must be stored safely on some medium across a crash, such as a disk, or it could be generated from a clock that continues to function across a crash. Nodes would have to wait 1 min after each crash before LTTP link initialization with the above assumption. This unique link-connection identifier would be sent along with the time value in both LTTP link-connection management and data frames. Thus on the order of 24 bits of frame overhead are required, 8 bit for the identifier and 16 bits for the time value as mentioned above to achieve MPL bounding on an arbitrary network.

When a link is to be reinitialized during normal operation, reinitialization frames must include the current link-connection identifier as well as the new one, to be acceptable and distinguishable from old duplicates. Alternatively, the link could be first explicitly closed using a reliable message exchange before reinitialization. However, a graceful close is not required at this level, as any frames lost can be recovered at the next level. In effect, many of the features of the TCP connection-management procedures are needed to initialize an LTTP connection in the case of a link where no special error control assumptions are possible. There are, however, simplifications because data delivery assurance is not provided and there is not the same hazard at crash recovery time. The question reasonably asked is, why provide any such mechanism like the above at this level? The answer is that a number of gains are achieved:

- Only nodes attached to links requiring a LTTP need implement the mechanism. The amount of mechanism required depends on the link characteristics.
- LTTP connection management is simpler and more reliable than that at higher levels because the service provided is simpler.
- The frequency of link initialization is far lower than that for process-to-process transport connection initialization, therefore reducing network traffic and allowing the higher level assurance protocol to have a transaction orientation.
- The higher level protocol, required on all nodes, can be simplified and made more reliable.
- Small MPLs can be guaranteed at the higher level.
- Knowledge of link-transit time may be useful for routing, congestion avoidance, or other network control services as well as error control at higher levels.

7. Connection Management and the Out-of-Band Channel

There are a number of applications such as terminal handling where there is a need for the communication of control information not blockable by the flow-control mechanisms on the normal data channel [4]. In many such applications, the program handling the control information is logically a separate process from that handling the data and, in the general case, may reside on a separate host. Even if the control program is in the same process as that handling the data, a separate port number could be used. This suggests that the control channel be a separate association or connection. The control address could be known a priori, by convention, or be obtained at application start up, or when first needed.

Most transport protocol designs do not use a separate association for this *expedited data channel* and instead, like TCP, incorporate two channels for each connection. This is because the cost of support-

ing a simple separate out-of-band channel is quite low when compared to that for setting up, maintaining, and tearing down a separate control connection, either dynamically only when needed, or statically during the life of the data connection. This cost differential, we believe, results primarily because of the use of message exchange connection management mechanisms, such as TCP's. Again like the TCP syn, fin, reset, or ISN selection mechanism, this second channel is not difficult to support but does add one more set of conditions to check at packet acceptance time and adds specification and documentation complexity.

Because Delta-t's connection management mechanism efficiently supports the low-delay, transaction type exchanges needed by the expedited control application, it does not need to support a separate control channel for each association. When an application needs an expedited control channel, a separate association is used.

8. Summary

We showed that even a protocol (TCP) that started out to achieve transport-connection-management reliability strictly with message exchanges had to have timer-based and other mechanisms added to deal with the hazards introduced because packets or their Acks can live for periods determined by the MPL, R, and A factors. The result, for TCP we believe, is a protocol more complicated than Delta-t, which was designed from the beginning to explicitly deal with these issues. Delta-t without explicit lower level support bounding MPL, R, and A may be somewhat more hazardous than TCP at connection opening.

For normal data exchange, during multipacket connection conversations, Delta-t packet-acceptance should be simpler than TCP's, yielding less packet processing overhead. Both protocols require keeping connection records under timer control, but we do not think in practice this is a serious resource management issue for either of them.

We believe that a careful analysis of connection management mechanisms in other protocols will show that they are making implicit assumptions about bounds on the information unit lifetime factors, or they contain some of the hazards TCP and Delta-t have reduced or eliminated, or they will have possible performance or service limitations. An example is the *European Computer Manufacturers Association* Transport Protocol (ECMA) [11] which we believe, with the procedures in the version referenced, could not operate sefely in a general network environment [10]. An extension of the EMCA protocol, based in part of the timer considerations raised here, has been designed to operate reliably in a general network environment [10,13]. It supports two conversation modes, transaction and extended conversation (explicit connection opening and closing). No mode distinction is required in Delta-t. Because of the way it utilizes timers in its connection management, it can support a reliable two message transaction service. It logically treats each transaction as a separately identified connection and therefore will use more connection record resources than Delta-t (one connection record timing out per transaction). It optionally supports a three message exchange transaction or connection opening procedure for use in environments where MPL, R, are only approximately bounded. Transport protocols, such as ECMA, designed primarily for an X.25 environment assume these issues do not exist, when in fact even there it appears some of them may [12]. An examination of link-level protocols and network level protocols such as X.25 show that they also are making assumptions dealing with these issues. For example, most initialization mechanisms of link-level protocols assume (often not clearly stated) that the link itself will not minsequence or duplicate frames and that frames have a bounded known lifetime. Also, they assume that each end bounds A and R or T, and the SNs cannot be recycled until previous frames and Acks with a given SN no longer can exist or be emitted.

The X.25 protocol assumes that after a Reset, Restart, or Clear procedure, no packets from a previous connection or phase can exist within the network. In othe words, connection flushing is guaranteed. At the interface this is guaranteed by the X.25 link level mechanisms; across the network, DCE-to-DCE mechanisms and assumptions such as those described here must support this guarantee.

Because of the fundamental nature of MPL, R, and A time-interval-based connection-management hazards, we recommend that mechanisms be supported in transport and lower level services to explicitly bound them or their sum. We suggested how such a bounding mechanism could be implemented [8].

We briefly examined the need for a transaction-oriented reliable protocol and found that the three message TCP exchange procedure is more hazardous

than Delta-t's two message or TCP's five-message procedures. Delta-t seems most efficient for this type of use. The implications of good transaction characteristics for supporting an out-of-band control channel were discussed.

We showed that to achieve a given type and level of connection-management service at one level, tradeoffs exist between: (1) the number and frequency of overhead messages exchanged, (2) the amount of overhead control information in frame or packet headers, and (3) the state-information-time product of state information retained at each node. We saw that these tradeoffs can span architectural levels to achieve overall system connection management or other global optimization design goals.

References

[1] D. Belsnes, "Single-Message Communication" *IEEE Trans. On Comm.* COM-242, 1976, pp. 190-194.

[2] S.R. Bunch and J.D. Day, "Control Structure Overhead in TCP," *Proc. Trend and Appl: 1980 Computer Network Protocols,* IEEE Cat. No 80 CH 1529-7C, pp. 121–127.

[3] J.G. Fletcher and R.W. Watson, "Mechanisms for a Reliable Timer-Based Protocol," *Computer Networks,* Vol. 2, (4–5), Sept.-Oct. 1978, pp. 271–290.

[4] L. Garlick, R. Rom, and J. Postel, "Reliable Host to Host Protocols: Problems and Techniques", *Proc. 5th Data Comm. Sym. ACM/IEEE,* Sept. 1977.

[5] J.B. Postel, "DOD Standard Internet and Transmission Control Protocol Specification" IEN 128, 129 (Jan. 1980), available through Defense Advanced Research Project Agency, IPTO, Arlington, VA. (Also NTIS AD A 079 730, AD A 082 609, respectively.)

[6] L.J. Sloan, "Limiting the Lifetime of Packets in Computer Networks," *Computer Networks,* Vol. 3 (6), 1979, pp. 435–446.

[7] C.A. Sunshine and Y.K. Dalal, "Connection Management in Transport Protocols," *Computer Networks,* Vol. 2 (6), 1978, pp. 454–473.

[8] R.W. Watson, "Delta-t Protocol Specification," Lawrence Livermore National Laboratory, Feb. 1981.

[9] R.W. Watson and J.G. Fletcher, "An Architecture for Support of Network Operating System Services," *Computer Networks,* Vol. 4 (1), Feb. 1980, pp. 33–49.

[10] R.W. Watson, "Strawman Modifications to ECMA Transport Protocol," ANSI Doc. X3S33/X3T56/80–101, Sept. 25, 1980.

[11] European Computer Manufacturers Association, *Standard ECMA transport Protocol,* 4th Draft, ECMA/tc 24/80/43, May 1980.

[12] ANSI, "X.25 Characteristics of Concern to Transport Service–Working Paper," ANSI X3537-80-32R or X3S33/X3T56/80-32R, April 1980.

[13] Burruss, J. et al., "Specification of the Transport Protocol," NBS Draft Report ICST/HLNP-81-1, National Bureau of Standards, Feb. 1981.