Issues in Reliable Host-to-Host Protocols

ABSTRACT

Fully reliable network host-to-host protocols have recently gained significant attention, primarily due to more stringent security requirements of network users. This paper will discuss issues related to one such protocol, which is supported by the Transmission Control Program (TCP). The protocol, first introduced in 1974, features end-to-end positive acknowledgement, retransmission, internetwork addressing capabilities, and ordered delivery.

The issues of interest in this paper are protocol correctness and completeness, protocol efficiency, and complexity of implementation. The discussion will suggest alterations and extensions to TCP.

Flow control heuristics using TCP's windowing techniques are explored. Flow control information is augmented to allow fair apportionment of bandwidth, better bandwidth utilization through optimistic credits, flow control credits matched to the type of traffic, and increased performance for high precedence connections.

An alternative for selecting the startup sequence number of a connection is presented. It is suggested that the resynchronization method for sequence number space management should be abandoned because it is overly complicated and can actually fail when the data stream is stopped by flow control.

The need for the separation of data and control channels is motivated, introducing the notion of a reliable subchannel.

The findings are presented both to further the understanding of reliable protocols and to encourage intelligent implementations of TCP.
ABSTRACT

Section: 2.4.4.1

July 1977

L. Garlick / SRI-ARC
R. Rom / SRI-ARC
J. Postel /SRI-ARC
15 March 1977

Issues in Reliable Host-to-Host Protocols

The problem of reliably transmitting information between two computers is

LAWRENCE L. GARLICK

Raphael Rom

Jonathan B. Postel

Department of Computer Science
California Institute of Technology
Pasadena, California 91125

An Interconnection for Selecting the Shortest Distance

A connection is proposed. It is described with the

by the use of an address of a reliable connection:

Platform-independent design for reliability. It is aimed to

by selecting another reliable connection.

March 15, 1977

Augmentation Research Center
Stanford Research Institute
Menlo Park, California 94025

(415) 326-6200
Issues in Reliable Host-to-Host Protocols

INTRODUCTION

Due to numerous advances in computer communications, there has been a tremendous growth in computer networking. This has led to the need for numerous advances in distributed computing protocols. Typical of these advances are the packet switching network protocols developed for the ARPA network. The need for a protocol that supports distributed process-to-process communication was realized early by ARPA network designers and the ARPA host-to-host protocol (AHHP) became the reference point for such process-to-process protocols.

The AHHP has been very successful in providing a basis for abundant research in distributed computing and in providing a prototype for process-to-process protocols. As experience with networking has grown, new applications, new topologies, new network access methods, and new higher level protocols have emerged. The AHHP has not been entirely suited for the new requirements that have resulted from this experience.

End-to-end reliability is an example of a new requirement needed by host-to-host protocols. It has been a concern for builders of both secure applications and higher level protocols. There are two important motivations for stringent reliability requirements. First, security measures, such as encryption, are often applied at the host-to-host level or lower. Second, higher level protocols, such as the ARPA TELNET protocol, should not be required to handle transmission error checking. The AHHP does not provide host-to-host acknowledgement; it relies upon subnet and host-to-subnet protocols to deliver messages reliably.

While the performance of the AHHP has been almost error free, it has been known to lose messages; thus it cannot be considered a fully reliable protocol.

Other deficiencies in AHHP include addressing constraints, weak error recovery, simplex connections, and large overhead for passing flow control information.

TCP, which, throughout this paper will be an abbreviation for both the Transmission Control Program and the protocol it supports, corrects the deficiencies of AHHP. TCP was
initially designed to be a reliable internetwork host-to-host protocol [Reference 1], as well as a solution to many of the problems of the AMP. When the special internetwork addressing considerations are ignored (as they shall be in this paper), it represents a significant advancement in host-to-host protocols. Among its reliability features are positive acknowledgement, retransmission, and sequencing of data and controls. It guarantees the error free delivery of each message for which it claims successful delivery. Other improvements include duplex connections and the ability to use a network address (socket) in several connections.

The paper is organized around three issues—a discussion of flow control techniques for TCP, alternate strategies for the management of connection sequence number space, and the need for a control subchannel for each TCP connection. To provide further context for the discussion, a brief summary of interesting TCP features is presented. It is assumed that the reader is somewhat familiar with the AMP and has been exposed to the early literature on TCP-like protocols [References 1, 2, 6]. A glossary of abbreviations and terms, and appendices that magnify a few of the more involved issues can be found at the end of the paper.

**TCP: A RELIABLE TRANSMISSION PROTOCOL**

**Network Characteristics**

TCP does not depend on the transmission medium for its reliability, i.e., it is assumed that the subnetwork may be unreliable. The subnet need not ensure the orderly or errorless delivery of subnet packets, or account for lost packets. TCP functions correctly in the face of large packet lifetimes, and the opening and closing of connections in quick succession.

**Connections**

Logical connections are established for process-to-process (user-to-user) communication. TCP connections are full-duplex channels established between source and destination sockets (network-wide process names). A socket may be a party to more than one connection, but only one connection can exist between any pair of sockets.
TCP provides the means by which a connection between the processes is established, controlled during the transfer of data, and terminated at the completion of the session. Connection management requires the exchange of controls between TCP's. There are controls for connection synchronization, out-of-band signalling (interrupt), data flushing, resynchronization, and connection closing. As described below, controls accompany data whenever possible to avoid the overhead of separate control packets.

**Packaging and Headers**

TCP packages user letters (messages) into packets suitable for transmission over a subnetwork. Each letter or partial letter is prefixed by a TCP header, which includes fields for addressing, sequencing, acknowledgements, flow control, controls, and error checking. The header is optionally followed by a block of data. The smallest unit of data transfer and the unit of sequencing is the 8-bit byte (octet).

**Sequencing**

Sequence numbers are used as acknowledgement identifiers and as an ordering mechanism. They are assigned to each octet of data and to those controls that need synchronization with the data stream. Only one sequence number is sent with each TCP header; it represents the sequence number assigned to the first control or data in the packet. This means that data and control sequence numbers come from the same name space. The packet length is used to determine the highest sequence number consumed by the packet.

Reuse of sequence numbers is allowed only for duplicate retransmissions. The sequence number space is managed by a cooperatively by the sender and the receiver, as will be discussed later.

**Acknowledgement and Retransmission**

A TCP acknowledgement represents the successful delivery of some number of octets to the receiving process's buffer or to the remote TCP (controls). It is sent to the transmitting TCP in the acknowledgement field of a subsequent TCP header. The sequence number placed in this field is the highest sequence number acknowledged by the
receiver and implies acknowledgement of all previous octets. If packets arrive out of order, an acknowledgement cannot be sent for octets with sequence numbers higher than the missing octets, since that would implicitly acknowledge the missing data.

Packets can be retransmitted at will until they are acknowledged; however, bandwidth may be underutilized if improper retransmission policies are followed. Duplicates naturally arise from retransmissions that occur prior to the receipt of an acknowledgment and are detected and handled as described below.

Synchronization and Resynchronization

TCP is expected to run in a network with relatively long packet lifetimes and relatively short times between the closing and opening of a connection. Therefore, several problems must be solved concerning detection of old duplicate packets, that is, packets that have sequence numbers assigned by old instances of a connection between the same sockets. These problems are how to select startup sequence numbers, how to reliably exchange new sequence numbers, and how to determine when resynchronization of sequence numbers is necessary.

The exchange of sequence numbers at synchronization or resynchronization time is accomplished using a "three-way handshake" method [References 2, 4, 5]. This method provides positive acknowledgement of the exchanged sequence numbers and is sufficient to handle the problem of simultaneous connection establishment attempts.

A solution to the other two problems has been an Initial Sequence Number curve [References 4, 5, 61, that is used by the sender as a mechanism for 1) selecting the first sequence number for a connection and 2) detecting when the consumption of sequence numbers is not progressing in a manner that will guarantee that old duplicates can be reliably identified by the receiving TCP.

The management of the sequence number space will be discussed in section 4.
Flow Control

Flow control is exerted by the receiver by issuing credits, which represent the receiving process's willingness to buffer data. Credits are passed in the TCP header in the window size field. The window size is added to the last acknowledged sequence number (the window's left edge) to give the highest allowable sequence number that may be sent (the window's right edge). Flow control is discussed in further detail in section 3.

Packet Acceptance Checking

The receiving TCP is responsible for the detection of packets with improper sequence numbers. These may have sequence numbers that are either old duplicates (from previous connections) or illegal because they are not within an acceptable flow control range.

To determine the action to be taken for a newly received packet, acceptability ranges are defined. The following three ranges are mutually exclusive and collectively exhaustive of the sequence number space (see Figure 1):

- Acknowledge-deliver range (ADR)

  The packet has arrived in-order and does not exceed the receiving process's buffer space. Data will be placed in the buffer and an acknowledgement will be generated to indicate successful delivery.

- Acknowledge-only range (AOR)

  A duplicate packet has arrived, as a result of retransmission. It will be acknowledged, but not delivered, since delivery has already occurred.

- Discard range (DR)

  An illegal packet has arrived. It may be an old duplicate or a packet that cannot be delivered due to flow control.

Appendix A provides more details of the packet acceptance policy.
Issues in Reliable Host-to-Host Protocols

Garlick, Rom, & Postel
FLOW CONTROL TECHNIQUES

Flow control is basically a mechanism to prevent the receiving process's buffers from overflowing. A good flow control scheme must handle a whole spectrum of problems that result from performing this basic duty. This section first discusses general flow control goals and methods, and then specific techniques for use with TCP that could significantly improve protocol performance. Where suggestions occur, they represent an enhancement to the flow control scheme used in the initial versions of TCP.

The goals of an ambitious flow control scheme include the following:

Receiver's Allocation

Any flow control strategy should consider the buffer space offered by a receiving user, since this represents a depository for incoming messages and relieves the TCP of resource allocation problems.

Congestion Prevention

The flow control strategy should prevent queuing of messages in the protocol module (TCP), so that TCP resources can be used to handle those messages that have a high probability of being delivered immediately. Congestion in the subnet can be caused by a retransmission protocol like TCP, since each unacknowledged packet is retransmitted. The flow control scheme should make it easy to slow or stop retransmission from the sender.

Deadlock Prevention

When congestion does occur, resources must be available to handle traffic-clearing messages. Controls and flow control information must be delivered and interpreted even when data is queued.

Fair Apportionment Of Bandwidth

In a virtual connection environment, it is important to be able to fairly allocate the available bandwidth to...
users, based on a variety of criteria. One criterion may be precedence of the user or the connection. Another may be the mode of traffic, e.g., interactive traffic may get preference over bulk traffic.

Bandwidth Utilization For Various Modes Of Transmission

A network will usually serve several types of user communities and thus should be able to adapt the flow control strategy to the needs of the user. For example, transmission patterns for interactive users and bulk transfer users are quite different. Those differences should be reflected in the flow control strategies.

Interplay With Subnet Flow Control

Often the interfaces between modules representing levels of protocol can cause flow control problems [Reference 8]. For instance, the subnet flow control of the ARPANET is adversely affected whenever a host does not readily accept incoming data from the packet switch (IMP). TCP is especially flexible in this regard, because it can absorb congested traffic from the subnet and discard it if necessary.

Exchanging Flow Control Information

A windowing scheme to convey flow control information has been used for many different types of protocols. It is an efficient technique that is useful whenever positive acknowledgement and retransmission are used for reliable transmission. Flow control information is passed in the header of a packet as a window size. It is used in conjunction with the acknowledgement sequence number (the window's left edge) to determine the highest sequence number that can be transmitted with some assurance that it will be acknowledged without retransmission. The acknowledge sequence number plus the window size gives the right edge of the flow control window.

A nonzero window size gives permission to send a message of a certain length. It is an "oversend" to send messages with sequence numbers that exceed the window right edge. In TCP, oversends will occur occasionally, since the flow control information is always slightly out of date and it is possible to withdraw flow control credits. Occasional oversends are not a problem, because the receiver can al-
ways discard incoming data without sending acknowledgements.

Determining the Window Size

The TCP acknowledgment and retransmission scheme allows flexibility in determining the correct flow control window size. The window size should indicate the willingness of the receiving process to provide buffer space. The window size could represent exactly the available buffer space that the user has offered for letter receiving (the conservative strategy), or it could reflect some expected buffer space, based on previous allocations (the optimistic strategy).

Conservative Guaranteed Allocation

The conservative approach to window size setting gives the receiving process almost full control over the flow control mechanism. By assuring the sender that there will be space for a particular number of octets, the policy reduces discards thus reducing the number of retransmissions. (Some messages may still be discarded if they arrive out of order and sufficient reassembly space is not available.)

There are some disadvantages to the conservative strategy of window size setting. Flow control information is always slightly out of date when it is finally received. The receiving process could have drastically increased or decreased its allocation, making the information useless. Unless a process provides for double buffering, the window very likely will go from a fixed size (whatever the user's buffer is) to zero, each time a message is passed on to the receiving process. Depending on the scheduling algorithm in the host, this could result in windows of size zero, totally inhibiting message flow. Before messages can flow again, a packet with flow control information must arrive at the source. Thus, a round trip delay is experienced between messages and there is an increase of dataless packets in the network.

Another related problem is that large single buffers may be used to receive small letters. If a window of say size \( k \) is advertised and a packet of size \( < k \) arrives that includes the end of a letter, then the destination
buffer is returned to the receiving process. The previous flow control credit, which was large, is withdrawn and the window becomes zero. In the interim, the sender may have sent several small letters, thinking the receiver has the buffers to accept them. The receiving TCP, knowing that the receiving process has no available buffer space, will advertise a zero window. By the time the window information arrives at the sending TCP, it likely will be an inaccurate report and cause further delays.

Optimistic Credits

The alternative to the conservative approach is to send flow control information that is a good estimate of the expected receiver's available space [References 3, 7]. Thus, the window size should be a function of previous window sizes as well as the current available space. The window size should be an average, weighted very heavily toward the current time, so that a process that is truly rejecting data will soon reflect a very small window.

This method could even be mixed with heuristics to force the window to zero after a fixed period without receiving.

Optimistic allocation can do much to help solve the problem of drastic window size changes experienced with the conservative scheme. In granting permission to transmit messages before the user has allocated buffer space, it fills the pipe and allows a smoother flow. It is still reliable, because any message can be discarded in the receiver since it will be retransmitted later.

The disadvantages of the method are its instability when faced with very irregular receiving patterns. A poorly behaving receiver can still sabotage this policy, but not as easily as with conservative technique. As will be shown below, an optimistic strategy may be quite dynamic with respect to recent receiving patterns, connection precedence, and the fair sharing of the available bandwidth.

It may be possible to determine the semantics associated with the window size by exchanging transmission mode or topological information. When a connection is opened,
the transmission mode (e.g., interactive, bulk) and the topology (e.g., satellite link) could be exchanged. This would be used to determine the weighting of previous window sizes in calculating the current window.

To demonstrate the idea of an optimistic flow control policy, a method for setting the receive window size is given in Appendix B.

Zero Flow Control Windows

It may be necessary to stop the flow on a TCP connection, i.e., stop all new transmissions and unnecessary retransmissions. This is required when there are no user receive buffers into which data can be placed. A zero receive window indicates an unwillingness to receive data. This reluctance is conveyed to the remote TCP by sending a packet with zero in the window size field.

When interpreting packets, each TCP must read window sizes on all packets, even those that acknowledge old duplicates. This is necessary for setting the window to zero when there is no data to carry the flow control information.

TCP must perform special functions with regard to sending packets into a zero window. If no data is being sent on the connection, a zero window is of no concern to the sending TCP. If there is data to be sent, it must be queued. If necessary, new data from the sending process must be rejected. The creation of new packets must be suspended entirely, and retransmission must be suspended, except for flushing controls, synchronizing controls, and the window opening control mentioned below.

Opening a window of size zero also presents some special problems [Reference 6]. Since a window size can accompany each packet, it seems that the normal data packet and acknowledgement transmissions should be sufficient to vary the size of the windows. However, when the remote TCP is showing a zero receive window, it is difficult to send a window change reliably. A data packet cannot be sent because the closed window indicates that only controls should be retransmitted; moreover, there may be no data to send. If ACKs are used and they arrive out of order, it may be impossible to tell if the window is opening or closing.
The problem of opening a window of size zero is solved by using a pair of controls, one sent by the local TCP that is making its window size nonzero (WOPEN) and one that is sent by the foreign TCP to acknowledge the opening (WACK). These are special controls that must be handled immediately, without regard for flow control restrictions. If controls can be blocked by data, as in the present TCP, then the WOPEN must be tagged with, but must not consume a sequence number.

SEQUENCE NUMBER SPACE MANAGEMENT

The second area of the current TCP protocol that needs attention is that of the reliable handling of the sequence number space. In a packet-switching network with alternative routing schemes, a packet can have a relatively long lifetime, especially if the topology of the network includes satellite links. Due to misrouting, a packet can arrive at its destination minutes or even hours late, depending on the topology. A reliable protocol must be able to determine if such a packet is deliverable, acknowledgeable, or if it must be discarded without acknowledgement. If during the packet's transit time the connection is closed or broken due to a crash with loss of memory, then the packet is no longer valid. If the connection is reestablished, using the same source and destination addresses, then the arrival of the old packet can cause confusion in the receiving TCP. A reliable mechanism must exist to guarantee that the receiving TCP can distinguish packets of the current connection from packets of an old connection.

Resynchronization, suggested by Tomlinson [Reference 4,5], is one such mechanism. Resynchronization is used in this paper to denote the mechanism itself, rather than the stage of the mechanism when the actual resetting of the sequence numbers is done. The scheme is based on selecting initial sequence numbers (ISN's) from a curve in the sequence-number/time plane. When a new connection is opened, its first sequence number is taken from the ISN curve. If the consumption of sequence numbers is satisfactory, i.e., similar in slope to the ISN curve, resynchronization of sequence numbers need not occur. However, if the rate of consumption is too slow, resynchronization may be required to avoid colliding with the ISN curve. The ISN curve has a parallel boundary (defining a "forbidden zone") that indicates that
no new sequence numbers may be assigned and that
resynchronization must take place immediately. If this is
not done and if a crash occurs, sequence numbers assigned in
the forbidden zone could conflict with the ISN chosen for
the new connection. See Appendix C, and References 4, 5, 6
for further details of the resynchronization mechanism.

A few of the problems related to implementing
resynchronization are discussed below.

Understanding and Documenting the Problem

Even though the resynchronization method is a workable
one, it is not at all straightforward. It takes
numerous pages and illustrations just to document the
concept (Reference 4, 5, 6). As has been pointed out in
the past by weathered ARPANET protocol implementers, a
protocol must be reasonably easy to understand and easy
to document. After all, if the network is
heterogeneous, it will be implemented on numerous kinds
of hardware by system programmers with various degrees
of skill.

Testing for the Need to Resynchronize

The protocol requires that if a connection is broken due
to a system crash, the sequence number chosen at startup
must be one that cannot be confused with any sequence
number still in the network for the old instance of that
connection. To satisfy this requirement, periodic
runtime checking must be done to determine if the se-
quence number consumption rate is satisfactory, i.e., if
it is approaching the forbidden zone. This check must
be done at fixed time intervals, not just when sequence
numbers are being assigned. The check may result in the
need to resynchronize even (and especially) if the
connection is idle.

Resynchronization and Flow Control

The need to resynchronize may occur at any time, and the
resynchronization must proceed in a timely manner if
normal activity is to continue. However, since
resynchronization means changing from the old sequence
numbers to new sequence numbers and since the
resynchronization control must be acknowledged (marked
with an "old" sequence number), all data marked with the
"old" numbers must be acknowledged before the resynchronization control is acknowledged. If data is not being accepted because the user is not receiving, then resynchronization cannot proceed. If resynchronization cannot proceed, then neither new controls nor new data may be sent.

The Loss of a Truly Out-Of-Band Signal

Due to the flow control problem mentioned above, all controls can be blocked during a resynchronization process. This includes the interrupt, which is supposed to be an out-of-band signal. Losing the out-of-band capability, even in rare instances, is an unfortunate deficiency. Higher-level protocols that rely on an out-of-band signal could be severely crippled by the inability to interrupt a "runaway" process. In fact, it is the runaway process, by not accepting data, that will soon force resynchronization and will not be interruptable.

Extra Connection States and Controls

When a state diagram is used to represent a TCP connection, 49% of the connection states are a result of the resynchronization mechanism (Reference 6). These seven extra states allow for simultaneous resynchronization attempts and resynchronization attempts during connection closing (with no data loss).

One extra control is required to support resynchronization. It is believed that more would be required for satisfactory solutions to the problems of resynchronizing a connection that is blocked by data flow control and for support of a true out-of-band signal.

Decentralized Code

Code to support resynchronization would be scattered throughout many modules of the protocol implementation. There must be a watchdog for detecting the forbidden zone. There would be heuristics strewn throughout the control sending and parsing modules. Also, to solve the flow control and interrupt problems mentioned above, special provisions must be made for either flushing data or saving old sequence numbers.
An Alternative to Resynchronization

An alternative to resynchronization is a strategy that uniquely names each instance of a connection. The name (or incarnation number) is passed in each packet and is used by the receiver to filter out packets from old connections. The incarnation number is generated from clock time; thus, like the resynchronization method, no crash-proof memory is required.

Each time a TCP comes up, it determines its incarnation number from a clock. The appropriate clock resolution and wraparound period is a factor of the maximum packet lifetime for the network or interconnected network. Let us assume that the clock has a resolution of one minute and a wraparound period of 256 minutes. The resulting incarnation number is 8 bits long, and is used to assure the receiver that any message received with this incarnation number is from the active connection and not an old one. The uniqueness of the incarnation number allows the resetting of the sequence number space to zero at initialization of each new path (first connection between two users).

When a connection is closed, a TCP must save the last sequence number used. It must retain the number for time NPL (maximum packet lifetime). Saving the sequence number and the time of a closed connection solves the problem of the repeated opening and closing of the same connection (source and destination). It does not solve the problems created by TCP or host computer crashes.

When connection establishment is requested, the list of old connections must be searched by (source, destination). If a match is found, the sequence number plus one is the first sequence number used when the connection is opened. If there is no match, then numbering can start at zero.

Management of the old connection list entails removal of outdated items. This can be handled for the most part, during normal searching. When list storage becomes scarce, a simple garbage collection routine can be invoked.

There are two problems with the method using incarnation numbers. First, there is some concern about the size of the old connection list. It would not be surprising to see 1000 connections per hour for an average host. The
fact that TCP allows a socket to be party to many connections will lead to fewer source and destination pairs; thus, many connections will be reused. (This is in contrast to the ARPANET network, where restrictions in socket usage result in contact connections being used to spawn direct, dynamically named service connections.) Another factor that alleviates concern about the space required for the old connection list is the recent progress in inexpensive memories.

The second problem is how to keep the incarnation number small enough to be sent in each header and still keep the clock cycle (name space) large enough to ensure uniqueness. It is felt that an incarnation number field greater than 8 bits is excessive header overhead. To accommodate this, the resolution of the clock is constrained, which leads to the following restriction applied at host startup time. When a host comes up after a crash, it must delay at least MPL / 2^N8 before any connections are opened, so that a unique TCP incarnation number is always chosen. A startup delay of one minute is probably sufficient for the internetting case since it implies a maximum packet lifetime (MPL) of 256 minutes.

THE NEED FOR A CONTROL SUBCHANNEL

In earlier versions of TCP, data, controls, and out-of-band signals (also a control) are all multiplexed onto one logical channel. This means that one set of sequence numbers is used for their orderly and reliable delivery.

One advantage of a single logical channel is the savings in the TCP header. Protocol overhead is a serious matter, since it is suffered with each message. Let us assume that it is desirable to allow piggybacking of activity from each channel. Since each logical channel requires header fields for both a sequence number and an acknowledgement number, header sizes increase by twice the sequence number field size as each new channel is added.

A second advantage to one logical channel is the ability to synchronize the control stream with the data stream. Synchronization of the control and data streams is useful for handling interrupts and connection closing (without data loss). However, synchronization of streams can result in
unwanted interdependencies, since the acknowledgement of a control may require the acknowledgement of preceding data. 7c

Two disadvantages of the single sequence number space scheme have been discovered recently: reassembly of data mixed with controls is costly when packets arrive out of order, and a true out-of-band signal is not being provided. The first problem is an efficiency matter that has plagued early implementers [Reference 9]. User buffer space cannot be used for the reassembly of out of order packets because there is no way to know if the unarrived packets contain only data or if controls are intermixed with the data. 7d

The essence of the second problem is that the acknowledgement scheme requires that acknowledgement of a sequence number is implicit acknowledgement of all preceding sequence numbers. Since interrupts must be acknowledged for reliability, the transmission of an interrupt can be blocked by data flow control in the receiver. This was noticed by Corf initially (Reference 2) and an attempt was made to rectify the matter by giving the interrupt extra semantics-- that it always flushes unacknowledged data. This solution is probably sufficient unless resynchronization methods are used for sequence number selection. 7e

As mentioned earlier, when the resynchronization method is used, there is no clean solution to the problem of achieving both synchronization with the data stream and independence of data flow control. This is due to the fact that the resynchronizing control can be blocked by data flow control but cannot be flushed. 7f

A compromise solution when using resynchronization is to separate controls and interrupts from the data channel, making a control subchannel. The control sequence number is the composite of the data channel sequence number (DCSN) and the subchannel sequence number (SCSN). This serves the dual purpose of synchronizing the two streams and using the resynchronization mechanism of the data channel for all subchannels. A subchannel allows reliable transmission even when the data channel is inactive, without flushing data. 7g

From the SCSN, the number of control fields, and the last SCSN received, the receiver can determine if subchannel traffic is coming in order and thus, whether it can be acknowledged. 7h
The field size holding the SCSN determines the wraparound point in the SCSN space. The SCSN space is initialized to zero when the DCSN is synchronized. It IS NOT reset with each DCSN change.

There is no flow control information passed for the subchannel. Discarding controls (without acknowledgement) is the flow control mechanism. Since the sequence number space is small compared to that needed to prevent wraparound in the worst case, the TCP must keep track of the DCSN to which the first SCSN was assigned. If wraparound of the SCSN space occurs, in the rare event that many controls are sent while the data channel is blocked, then the control channel becomes blocked. This is very unlikely because a long series of controls will probably contain a string of interrupts, and successfully delivered interrupts will usually cause the receiving process to unblock the data channel.

Acceptability Test for Subchannel Traffic

The acceptability test of items on the subchannel is a composite test of both sequence numbers. First the DCSN is checked to see if it would be acknowledged if it were an octet received on the data channel. Only if it would have been discarded will the item on the subchannel be discarded. Having passed the DCSN test, the SCSN is checked to see if the item is deliverable and acknowledgeable with respect to the SCSN sequence number space. The SCSN test is less involved than the DCSN test because there is no flow control range. To be believable, the SCSN must fall in the range of SCSN's sent and SCSN's for which acknowledgements have been received. This is a check for everything except the existence of old duplicates from old instances of the connection, which is made by checking the DCSN.
A Scenario Using a Control Subchannel

Let us examine a short scenario between TCP A and TCP B. The scenario assumes connections have been established and transmission has proceeded normally. Only those header fields that relate to data and control channels will be shown.

Note that the control length can be determined by the receiver from other fields in the header. The following shorthand will be used in the scenario:

- **DSN** - data sequence number
- **DL** - length of data in octets
- **DACK** - acknowledgement for all preceding data octets
- **CSN** - control sequence number
- **CACK** - acknowledgement for all preceding controls

---

Garlick, Rom, & Postel
#1 from TCP A

\[ \begin{array}{c}
! \text{DSN} ! \text{DL} ! \text{DACK} ! \text{CSN} ! \text{CACK} ! \text{===>} \\
! 100 ! 2 ! 200 ! 5 ! 25 ! \text{===>}
\end{array} \]

Sends 2 data octets (100 & 101),
acks data through 200;
sends 1 control (5), acks controls through 25.

#2 from TCP A

\[ \begin{array}{c}
! \text{DSN} ! \text{DL} ! \text{DACK} ! \text{CSN} ! \text{CACK} ! \text{===>} \\
! 102 ! 3 ! 200 ! 5 ! 25 ! \text{===>}
\end{array} \]

Sends 3 data octets (102-104),
acks data through 200;
sends no controls,
acks controls through 25.

#3 from TCP A

\[ \begin{array}{c}
! \text{DSN} ! \text{DL} ! \text{DACK} ! \text{CSN} ! \text{CACK} ! \text{===>} \\
! 105 ! 3 ! 201 ! 6 ! 25 ! \text{===>}
\end{array} \]

Sends 3 data octets (105-107),
acks data through 201;
sends 1 control (6),
acks controls through 25.

#4 from TCP B

\[ \begin{array}{c}
<== ! \text{DSN} ! \text{DL} ! \text{DACK} ! \text{CSN} ! \text{CACK} ! \\
<== ! 202 ! 1 ! 101 ! 26 ! 6 !
\end{array} \]

Having received #1, #3, but not #2,
sends 1 data octets (202),
acks data through 101;
sends 1 control (26),
acks controls through 6.

The main things to notice from this scenario are that data and controls are still piggybacked, as in the current version of TCP, and that there is a degree of independence between the two channels. As the scenario shows, TCP B can acknowledge controls that have arrived in order even though
it has not received data in order. Moreover, TCP B is able to use the latest data sequence number to test the acceptability of the latest control sequence numbers.

SUMMARY

Several suggestions have been presented here for the improvement of TCP. The suggestions relate to improved efficiency, simplification of implementation, and protocol functionality. The motivation for the suggestions is more than to improve a specific protocol. It is also to focus attention on a set of issues that are common to all reliable host-to-host protocols.

Flow control ideas have been discussed, with attention to implementation ideas that satisfy fairly ambitious goals. Window management techniques have been suggested that could improve efficiency. A window setting method was presented that features optimistic credits that are a function of past credits, congestion, and available buffer space.

An alternative to the resynchronization method of sequence number space management has been given. The suggested method is based on passing TCP incarnation numbers and keeping an old connection list. The method is simple to implement, requires no nonvolatile memory, and still guarantees reliable detection of illegal packets.

Finally, the need for the separation of data and control channels was motivated. The solution, a reliable subchannel, is achievable with no separate sequence number space maintenance.

It is hoped that each of these suggestions will be implemented in future versions of TCP. There are interdependencies involved; that is, some of the stated problems become less severe when others are solved. For example, if resynchronization is abandoned, then the argument for separate channels is motivated only by the need for the efficient reassembly of out of order packets.

Of all the suggestions, the most important is that concerning a new approach to sequence number space management. However, if resynchronization methods are retained, then a
subchannel for controls is a must. Otherwise, a truly out-of-band signal is lost.

The discussion of flow control indicated areas that should gain attention as more experience with TCP is gained. This should be an area for significant measurement, under many different transmission modes.
REFERENCES


GLOSSARY

AHHP: ARPANET host-to-host protocol.

control: commands passed between TCP's that are used to co-
ordinate connection management.

DCSN: data channel sequence number.

host: a computer that is connected to the network and that
executes programs on behalf of its users. A host may pro-
vide services to other computers on the network.

ISN: Initial sequence number; the first sequence number
used when a connection is synchronized or renumbered.

MPL: maximum packet lifetime.

octet: eight bits.

SCSN: subchannel sequence number; control channel sequence
number.

socket: an entity defining one end of a TCP connection; the
inter-network-wide name of a process port.

subnetwork: the network of computers that provides a com-
munication medium for network hosts. The nodes of a
subnetwork may function as host interface points as well as
store and forward computers.

TCP: Transmission Control Program and the protocol it
implements.

window: a dynamic range in the sequence number space used
in flow control management.
APPENDIX A: PACKET ACCEPTANCE

This appendix provides details of the TCP packet acceptance testing scheme. It should clarify the possible actions the receiving TCP may take when it receives an arbitrary packet. Remember, the receiver is responsible for the detection of packets with improper sequence numbers from either old connections or ill-behaving TCP’s. For notation, let

ADR = acknowledge and deliver range
AOR = acknowledge only range
DR = discard range
S = size of sequence number space (number per octet)
x = sequence number to be tested
FCLE = flow control left window edge
ADRE = (FCLE + ADR) mod S = Ack-deliver right edge (Discard left edge - 1)
AOLE = (FCLE - AOR) mod S = Ack-only left edge (Discard right edge + 1)
TSE = time since connection establishment (in sec)
MPL = maximum packet lifetime (in sec)
TB = TCP bandwidth (in octets/sec)

For any sequence number, x, and packet text length, l, if

(AOLE <= x <= ADRE) mod S and
(AOLE <= x + l - 1 <= ADRE) mod S

then the packet should be acknowledged.

If x and l satisfy

(FCLE <= x <= ADRE) mod S and
(FCLE <= x + l - 1 <= ADRE) mod S
then \( x \) can also be delivered to the user; however, ordered
delivery requires that \( x = FCLE \).

A packet is not in a range only if all of it lies outside a
range. When a packet falls in more than one range, prece-
dence is AOR, then ACR, then CR. When a packet falls in the
AOR then an ACK should be sent, even if a packet has to be
created. The ACK will specify the current left window edge.
This assures acknowledgment of all duplicates.

ADRE is exactly the maximum sequence number ever
"advertised" through the flow control window, plus one.
This allows for controls to be accepted even though
permission for them may never have been explicitly given.
Of course, each time a control with a sequence number equal
to the ADRE is sent, the ADRE must be incremented by one.

AOR is set so that old duplicates (from previous
incarnations of the connection) can be detected and dis-
carded. Thus

\[ AOR = \text{Min}(TSE, MPL) \times TB. \]
Each time a packet is sent for this connection, the local TCP sets the receive window and the packet header window size field so that

\[ W = (AVWT \times \text{Avail/Tot}) \times WLT + (OLDWT \times W) \quad (\text{for nonzero} \quad \text{Tot}) \]

and

\[ W = OLDWT \times W \quad (\text{for Tot} = 0). \]

It is important to note that a user's receive buffer is returned when an End-of-Letter is received. Thus, a small letter sent to a large buffer can cause the Avail and Tot to vary abruptly, even though there may be a smooth flow of letters.

This window size setting scheme meets the goals mentioned in section 3 in the following ways:

WLT is dependent upon the number of the connections, thereby administering fairness among connections. It also considers the level of congestion in the receiving TCP, assuming some measure of resource availability can be provided.

The window size will never exceed the bandwidth allocated to the connection. The algorithm may sometimes give credit to a "well behaving" process by setting his window to greater than the actual buffer available. This window will be reduced if the process does not supply new receive buffers promptly.

The current window size is dependent upon previous window sizes and upon the rate at which the process makes letter space available. If a process fails to make such space available, its receive window will be reduced by OLDWT every time a packet is sent. (The TCP may also apply a threshold mechanism by which a window is set to zero when it is reduced below the threshold.)

The algorithm can be modified slightly to support high throughput for high precedence connections. Parameter WLT can be made dependent on some criterion for the high priority traffic. Categories of priority can be used with some guaranteed service (part of the bandwidth) given the highest priority categories.
APPENDIX C: RESYNCHRONIZATION DETAILS

In Figure 2, we show the history of sequence numbers used by a particular connection. The lines labeled "ISN" represent the maximum permitted rate at which sequence numbers can be used, however, this may be different than the maximum throughput rate for the TCP.

Suppose that the TCP supporting the connection fails at "C" and must be restarted. Assume, also, that the sequence number selected to restart is drawn from the value of ISN at the time event "C" occurred. The shaded area between "C" and "B" represents the maximum expected time that packets emitted at "C" can stay in the net. Clearly, the ISN line intersects this shaded area, indicating that, after the restart, it is possible that packets emitted at "C" may become undistinguishable from those potentially emitted along the ISN curve. To correct this flaw, the sequence number currently to be used on the connection must be resynchronized before running into the forbidden zone to the left of the ISN line.

Testing for the need to resynchronize

As packets are produced and sequence numbers assigned to them, the TCP must check for two possible conditions which indicate that resynchronization is needed. The first is that sequence numbers are being used up so fast that they advance faster than ISN. The other is that they advance so slowly that ISN "catches up with them."

The basic method of selecting an initial sequence number is to delay for an arbitrary period labelled a "clock tick" or STEP and then select the new ISN.

In Figure 2, three sequence number histories are traced, ending in points "A", "B", and "C". In the trace labelled "A," sequence numbers are used at such a rate that point "A" lies beyond ISN plus one STEP. If the connection were to fail and be restarted at "A," the new ISN would be just below point "A" and would introduce potential unwanted duplicates.

This situation can be detected before transmission of the packet. Let $L$ be the length of the data in octets. Let $SEQ$ represent the proposed sequence number of the packet, and $SEQ+L-1$ be the sequence number implicitly associated
with the last octet of packet data. Also, let SNPL be the sequence numbers consumed at maximum TCP throughput during a maximum packet lifetime. If ISN+STEP (at the moment that SEQ is to be assigned) lies in the range [SEQ, SEQ+L-1], then the type "A" ISN failure is about to occur. The solution is to send only as much text as is allowed (which does not result in the failure) and WAIT for the clock to tick again.

The situation in curve "B" is quite different. In this case, the connection is using numbers so slowly that the forbidden zone preceding the ISN curve has advanced and run into the connection sequence number curve. There are two solutions. One is to wait for the packet lifetime plus one clock step to expire (in which case the sequence history will pop out of the forbidden zone again). The other is to actively resynchronize the connection. The test for the type "B" situation is whether sequence number SEQ lies in the range [ISN, ISN+SNPL+STEP].

Note that all tests for inclusion must be modulo S, the size of the sequence number space, to account for the wrap around of sequence numbers.

Curve "C" in Figure 2 shows a sequence number trace which tends, on the average, to lie within legal values at all times.
APPENDIX B: WINDOW SIZE SETTING

To demonstrate the idea of an optimistic policy for window size setting, a method for setting the receive window size is given (Reference 6). The scheme satisfies the flow control goals discussed earlier. Several parameters have been vaguely unspecified since they can be determined only after considerable testing and measurement of a specific TCP implementation.

First, some notation:

B - Total bandwidth of the TCP, given unlimited user resources

N - The number of connections in the TCP

CONGEST - A congestion factor which reflects available TCP resources (CONGEST < 1)

WLT - The long term window

W - The current window

AWWT - Weighting coefficient for available buffer space

OLDWT - Weighting coefficient for old window (OLDWT = 1-AWWT)

Tot - Total user buffer space

Avail - The unfilled part of Tot

The long term window might look like:

WLT = B/N & CONGEST.

The algorithm used to update the current window is the following. Upon the processing of a user's receive request (buffer offering), the local receive window is set so that:

W = MINIMUM(WLT, Tot).

Garlick, Rom, & Postel