IEN # 18
Supercedes: None
Replaces: None

Section: 2.4.4.3

J.D. Burchfiel
W.W. Plummer
R.S. Tomlinson
BBN
26 October 1976

IEN # 18

Proposed Revisions to the TCP

Jerry D. Burchfiel
William Plummer
Raymond S. Tomlinson

Bolt, Beranek and Newman, Inc.
50 Moulton Street
Cambridge, Mass. 02173

10/26/1976

By way of example, consider the following "worst" case scenario:

(1) Assume the Sender sees a 0 window and is blocked with 10 1-character letters waiting to be sent.

(2) The receiver does one 10-character RECV and sends this 10 character window to the Sender.

(3) The sender sees the window open and sends all 10 letters each with EOL. Note that the window has not been violated.

(4) The Receiver receives the first of the ten letters and returns the one RECV buffer to his user because the EOL forced completion of the buffer. An ACK is now returned to the Sender which specifies a window of 0 bytes.

(5) Since the network is slow, the Sender continues to retransmit the 10 letters. Note that the initial retransmission rate is high in most implementations.

(6) The user on the Receive side does another RECV. Since his TCP is being flooded with packets from the sender, he stands only a 1/10 probability of getting the "right" next packet. After some amount of processing in order to determine that the current packet is wrong, he will try again.

(7) Eventually the sender receives the packet saying that the window has closed down to zero, and ceases retransmitting. At this point one packet will have been ACKd and deleted from the retransmission queue.

(8) Somewhat later an ACK packet will arrive which specifies another 10-character window. This is generated when the second RECV was done. This will allow the sender to generate one more 1-character letter, replacing the one which was acknowledged.

Experiments with TCP have shown it to have marginal performance and also be lacking certain functions. Under certain source-destination letter size to buffer size mismatch conditions, real-time response has been poor and excessive network traffic generated. The CPU load caused by running TCP is much higher than with simpler, e.g. NCP, protocols. Suggestions are made in the following which will improve these deficiencies. Since several of the improvements will require new packet formats and large scale software changes to existing TCP's, we suggest that they all be done in one large change, yielding "Protocol Version 3".

1. New Sequence Number Scheme

Because the current protocol permits using sequence number space without using (user) buffer space by sending control information (INTERRUPT, DSN, etc.) a receiving TCP cannot do partial, out of order reassembly of the incoming data; instead, it must store the packets and pass them to the user in order in case there might be control information in one of the packets. This results in complicated and time consuming computations in order to do the reassembly process and a high load on the free storage area if any reasonable size window is being used.

To correct this we will use a 24-bit sequence number space for data bytes and a 8-bit space for control information. Since control information is to be interpreted as occurring "between" data bytes so that overall sequencing remains correct, the control sequence may be thought of as a fraction of the data sequence. Many of the current routines which deal with sequence numbers in the current TCP will still work if the 24-bit data sequence is handled as the most significant bits of an overall 32-bit sequence number.

The advantage in this arrangement is that a receiving TCP can infer the address of each data byte by looking at the data sequence number at the time of reception. This allows placing the data in the user's buffer at this time; no actual reassembly is needed even though packets may arrive out of order.

For this scheme (and previous versions!) to function correctly, we require that an INTERRUPT not be sent if any other control is outstanding, especially a previous INTERRUPT. As with the previous protocols, nothing may be sent after the end of a sequence (i.e., after a DSN or FIN) without first re-establishing a valid sequence.

Example: Assume 6 packets are sent: (1) 3 bytes of data, (2) 2 control bytes, (3) 2 controls, (4) 2 data, (5) 2 leading controls plus 3 data, and (6) 1 data byte. If the current data sequence is 100 and the control sequence is 0, the packets will have the following sequence numbers:

```
100[3],0[0]        ;3 of data at sequence 100. No control.
103[0],0[2]        ;2 controls, sequenced after previous packet.
103[0],2[2]        ;2 more controls, after previous ones.
103[2],4[0]        ;More data.  Note control sequence.
105[3],0[2]        ;Data only.  Note control sequence reset.
108[1],0[0]        ;The final data.
```

Thus, in a packet containing both controls and data the controls are sequenced before the first data byte.

A problem arises when the 8-bit control sequence field overflows. To simply let the carry go into the data field almost works but destroys the simple relation between data sequence number and user byte address. The receiving TCP is made aware of the change with DSN. Thus, if a control byte is to be sent and the control sequence is at 377 (octal) a DSN is sent, followed by a SYN at ISN,0 where ISN is the "initial sequence number" which is geared to time. Of course, DSN will continue to be used as it was in the old protocol in addition.

2. Windowing

During testing of TCPs using server and user programs written by different individuals, it was noted that transmission became very inefficient with many retransmissions required to get even a few letters across a connection. The problem was eventually identified as having to do with three factors:

☆ A mismatch in the size of the RECV buffers (large) and the size of the letters sent (small). This is the major problem area.

☆ Long delays in the network and/or slow processing in the TCPs, causing window information to be stale.

☆ The design choice made in the receiving TCP to discard all packets on a connection except the one specifically required. This is a valid technique; retransmissions by the sender should eventually supply the needed packet.

> Note: Should the user on the Send side decide to switch to 10-character buffers the performance improvement will not be immediate. For some time afterwards the TCP will chop the bigger buffers into 1-character packets, because the receiving TCP is acknowledging only single characters. Thus, the "bad" performance will continue until all of the single character letters have been acknowledged by the receiver. Until the improvement actually happens, both ends of the connection will be generating 10 times as many packets (in the example) as needed, and using 10 times as much computing power.

In this example, the culprit was the EOL which went along with the first letter. It was this which caused the window to decrease to 0. In effect it claimed the 9 unfilled bytes in the RECV buffer. But the Sender had no way of knowing this -- he has no information about how big the buffers are on the RECV side.

In general the situation arises because EOL (end-of-letter signal from a sender to a receiver) will claim an unknown amount of buffer space. The existing protocol has no way to cause the sender to decrease his available window by the number of bytes implicitly sent by the EOL which caused a RECV buffer to complete. In a sense it has been filled with non-existent bytes which do get removed from the window because the buffer gets (by definition of EOL) returned to the user.

To cure this we extend the protocol by defining contents of the window field of a packet as being the size of the user's RECV buffers if (1) the sending TCP does not already know this for the connection in question, and (2), the number in the window field is not zero. Note: Receive buffer size remains constant for the life of the connection. To insure reliable transmission of the buffer size information, the ARQ control bit should be on in the packet which carries it. See Section 7.

The buffer size information is used by the packetizer process in a sending TCP to compute the actual amount of window space that can be used. Each actual data byte sent decreases the available window by one. Sending an EOL decreases the available window (advances the packetizing sequence number) by all of the bytes remaining in the current buffer (which may be zero). Sending an INTERRUPT will also cause completion of the receiver's current buffer. Packet ends need not correspond with buffer ends.

3. Checksum and Fragmentation

For the near future the original fragmenting concepts will be retained. Specifically, only the last fragment (marked with EOS) will carry a checksum. The checksum found in fragments with EOS on will be that for the entire segment, as copied from the fragmented parent. This is an end-to-end check and has the advantage of requiring little computation by gateways and makes it possible to change the current checksum function without changing all of the gateways. Destination TCP's will not be burdened by having to checksum the fragments.

With the exception of the BOS and EOS bits, the DataLength, and the SequenceNumber, the headers of all fragments can be the same. This is a consequence of the fact that none of the information in the packet (DSN, ACK, etc.) can be acted on until the whole segment has been reassembled. It is not until then that the segment checksum can be verified. After an entire segment has been reassembled, all of the information in the segment will be processed at one time and an ACK issued. The fact that all fragment headers may be nearly the same simplifies the gateway code since it does not have to be very selective in constructing the fragment headers. No problem arises because two fragments have (say) two INIs on because the header was copied from the parent.

There are a few comments which bear on segment reassembly. First, packets (fragments) which are outside of the window are rejected on the basis of their apparent, unchecked SequenceNumber or ACKSequence (if not synchronized). If the packet has EOS on, the entire header is saved, at least conceptually, for later checksum verification. Since the ACKSequence and Window of a packet may change from one retransmission to the next, care must be exercised so that these fields and the checksum used for validation are taken from the same packet header, i.e. the one marked with EOS. The checksum function is computed for the data portion of the packet and "added" into what will be the checksum of the entire segment. Since the SequenceNumber tells where in the user's buffer this data belongs, the data may be transferred there directly.

As fragments are received and the data moved to the user's buffer(s), the TCP must keep a table which describes the portion(s) of the segment which have been filled. Fragments received from different gateways may overlap so not every packet processed will contribute to filling the missing data by an amount equal to the DataLength of the fragment. The TCP must be able to know the data length of the original segment for use in the header checksum computation however.

When a set of fragments big enough to cover the entire segment have been processed, the TCP will know the value of the checksum function over the data portion(s) and will have enough information to reconstruct the header associated with the entire segment. The checksum of the header is generated and combined with that for the data. If it is zero, the segment is valid and all of the control information including ACK if present may be acted on. If the segment checksum turns out bad, the entire segment is forgotten and the TCP waits for retransmissions.

4. Half-Open Connections

A solution to the problem of dealing with half-open connections has been found. Because of this, there is no longer a need for the "RESET(connection)" control command, and TCP code has been made simpler.

Basically, the change is to the interpretation of the EFP+7 "no such TCB" error message. Typically, this error will be generated by a TCP which has crashed and restarted upon receipt of a packet for a connection which existed before the crash. The ACKSequence of the error packet will acknowledge the entire packet which provoked the error reply and its sequence number will be taken from the ACKSequence of the provoker, which guarantees that the error packet will be acceptable for processing by its receiver. A TCP receiving an acceptable EFP+7 error packet should delete the TCB associated with the connection and inform the user owning the connection of the fact.

It is possible for delayed copies of packets from a previous incarnation of a connection to arrive at a TCP which has already closed the connection. These will also elicit EFP+7 errors. Should a delayed copy of one of the error packets arrive at the other end, which has by then opened a new incarnation of the connection, it will not be acceptable for processing because (depending on the state of this end) its Sequence number or ACKSequence will be out of bounds due to the properties of the Sequence number selection algorithm.

5. ABORT User Call

A new user call "ABORT(connection)" has been implemented. This simply deletes the local copy of the TCB after sending an EEP#7 error to the remote end. Note that sending this error packet is a courtesy and is not required. The other end will find out that the connection has gone away if it attempts to send a packet on it -- see the description above.

ABORT is intended to be used by a user when he has (for instance) given up on trying to CLOSE a connection. This can occur if the remote end has become very slow at processing data sent to it, and never makes it to the point of handling the FIN (which is sequenced).

6. Optional Information in Header

N.B. The discussion which follows pertains not only to TCP packets but also to InterNet packets.

Timestamp(s) and security information are examples of information which might appear in some packets but not others. Thus, it would be wasteful to preallocate fields in the header of all packets to hold this data. Instead, we will permit using the area between the end of the actual header and the beginning of the data. The presence of this optional information is discovered by noticing that the HeaderLength of the packet in question contains a number larger than the minimum header length required by the protocol.

Within the options area will be found any number of repetitions of the pattern "1 byte of count, 1 byte of kind, and N bytes of actual option information". This permits an arbitrary amount of option information to be added to any InterNet or TCP packet. In particular, multiple timestamps may be added.

One special code will be specified here: 0 in the count and kind bytes of any option will be understood to mean "this is a null option which occupies all of the space remaining in the options area". This assignment permits padding the header with bytes containing zeros as is now done in the TENEX TCP in order to align the beginning of the data area with a word boundary inside TENEX.

The fine-structure of options is not specified here. One rule to be followed in designing options is that they must be a multiple of 16 bits in length to avoid complicating programming in minicomputers. Multiples of 32 bits are favored even more due to limitations of some common higher-level languages.

The reader should not confuse the InterNet Format field indicating "Secure TCP" or not, and the (possible) existence of options of the kind "security information". The Format field is purely a specification to gateways of what fragmentation algorithm is to be used, independent of what is in the packet.

7. Acknowledge Request (ARQ) Control Bit

It frequently occurs that information such as window size or buffer size must be passed reliably across a connection. Normally these are not acknowledged, but in some cases the sender must know that the information has been heard by the receiver. Thus the ARQ bit has been added to the Control Flags field of TCP packets. This occupies one slot in the control sequence space and must therefore be acknowledged by the receiver. The exact position of ARQ in the packet is shown in section 11. ARQ is to be processed after INT (which is after SYN), but before the packet data or "trailing controls".

8. Delete FSH (Flush) Control Bit

FSH was originally intended to be used to "clear the pipe" so that a FIN could be forced through to the remote TCP. This was needed to guarantee prompt closing of connections. With the above discussion of half-open connections and the ABORT call, there is no longer a need for FSH.

9. Delete REJect

No use is seen for the REJect primitive in the protocol. It may safely be deleted.

10. Delete the T (Trace Control Bit)

The T bit of the TCP header control flags was previously used to control timestamping. Since timestamping is an InterNet function which can apply to non-TCP packets, it cannot be controlled by a TCP header bit. The definition of the T bit has been deleted.

11. Version 3 Packet Format.

The revised packet format is shown below. It is nearly the same as the Version 2 format. The major differences are in the deletion of several control bits, and the space provided for options.

```
InterNet Header (11.5 bytes)
        Byte    0:      Destination Network
        Bytes 1-3:      Destination TCP (host)
        Byte    4:      Source Network
        Bytes 5-7:      Source TCP (host)
        Bytes 8-9:      Data Length (in bytes)
        Byte   10:      Header Length (in bytes)
        Byte   11:      13 Hex
            Bits 0-3:   Format=1 for TCP. (2 for Secure TCP, etc.)
TCP Header
            Bits 4-7:   TCP Protocol Version = 3
        Bytes 12-15:    Sequence Number
            Bits  0-23: Data Sequence
            Bits 24-31: Control Sequence
        Bytes 16-17:    Window for data
        Bytes 18-19:    Control Flags
            Bit   0:    SYN
            Bit   1:    ACK
            Bit   2:    FIN
            Bit   3:    DSN
            Bit   4:    EOS
            Bit   5:    EOL
            Bit   6:    INT
            Bit   7:    (unused)

            Bit      8: BOS
            Bit      9: (unused)
            Bit     10: ARQ
            Bits 11-12: (unused)
            Bits 13-15: Control Dispatch (1 for ERROR, 2 for SPECIAL)
        Byte     20:    Control Data (for ERRORs, etc.)
        Bytes 21-23:    Destination Port (socket)
        Byte     24:    Label (for debugging)
        Bytes 25-27:    Source Port (socket)
        Bytes 28-31:    ACK Sequence
        Bytes 32-33:    Checksum

Optional Data begins here if HeaderLength > 34.
        Bytes 34-(34+2+L0-1):               Option 0
            Byte  0:            L0 (length of Option 0 in bytes)
            Byte  1:            K0 (kind of Option 0)
            Bytes 2-(2+L0):     Data associated with Option 0
        Bytes (34+2+L0) - (34+2+L0+2+L1-1):       Option 1
                  .
                  .
                  .

Actual TCP Packet Data begins here
        Bytes (HeaderLength) - (HeaderLength+DataLength-1):      Data
```