Selecting Sequence Numbers

Raymond S. Tomlinson

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

Introduction

A characteristic of almost all communication protocols is the use of unique numbers to identify individual pieces of data. These identifiers permit error control through acknowledgement and retransmission techniques. Usually successive pieces of data are identified with sequential numbers and the identifiers are thus called sequence numbers.

This paper discusses techniques for selecting and synchronizing sequence numbers such that no errors will occur if certain network characteristics can be bounded and if adequate data error detection measures are taken. The discussion specifically focuses on the protocol described by Cerf and Kahn [1], but the ideas are applicable to other similar protocols.

The Problem

One of the problems with the protocol described by Cerf and Kahn which was brought out by our experiments at BBN is "How do you identify duplicate packets from a previous use of a particular connection?". What is necessary is a means for either the receiver of the late packet to identify the packet as late or a means for the originator of the packet to tell the receiver that the packet was late.

The protocol as described provides a fine mechanism for identifying late packets while data are actively flowing between two ports. It would also work fine if hosts never went down and had large amounts of storage. But hosts do go down and have to be restarted and hosts don't have an inexhaustible supply of storage.

The Solution

The essence of the solution is that sequence numbers must be chosen such that a particular sequence number never refers to more than one byte at any one time and the valid range of sequence numbers must be positively synchronized whenever a connection is used. The former requires careful attention to the method used for selecting initial sequence numbers. The latter requires a more involved handshake than that provided by the protocol.

---

(1) Cerf, V. G. and Kahn, R. E., "A Protocol for Packet Network Intercommunication," IEEE Transactions on Communications, Vol. COM-22 #5, May 1974, pp. 637-648.

Positive Synchronization

Achieving positive synchronization requires a three way handshake for SYN and a two-way handshake for REL. This is necessary because the passive side of the connection must have positive assurance from the active side that the packet received is current. Simply receiving a packet does not provide this assurance. Assuming one end as the initiator and the other end as responder, the normal procedure for synchronizing sequence numbers is:

1) Initiator sends a SYN with a unique sequence number (one that cannot be outstanding in the net).

2) Responder receives this packet but does not process the data (if any) because it does not know whether or not the packet is a late duplicate.

3) Responder returns a packet which ACKs the initiator’s number and SYNs the responder’s unique sequence number.

4) The initiator receives the responder’s SYN and believes it because it ACKs an appropriate sequence number and SYNs a previously unsynchronized (in that direction) connection. The initiator now knows that the responder is willing to go ahead and knows where the responder is going to start but the responder does not yet know if the initiator was really trying to synchronize or if the packet received was a late duplicate.

5) The initiator sends back a packet which ACKs the responder’s sequence number.

6) The responder receives this packet and believes it because it ACKs an appropriate sequence number and has a sequence number of its own which is in the appropriate range. The original packet and this one may now be processed further, data delivered, RELs processed etc.

The handshake for REL needs to be only two-way because valid sequence numbers have been established and may be used to acknowledge the REL. Since the need to send a REL may occur at times when there are no data bytes to transmit, a dummy byte which is not delivered to the user process is sent to provide something to be acknowledged.

A consequence of this need for positive synchronization is that any data in the initial packet with the SYN may not be delivered to the user process until the validity of the packet is verified. It is useful to include data in the initial packet, however, since that data may be acknowledged in the first response packet eliminating the need for a subsequent packet for acknowledging the data. A minimum exchange for data flow in one direction is four packets. This is illustrated by example below.

The fourth packet is required to inform the initiator that it may forget the connection information.

It is also necessary to provide a mechanism to negatively acknowledge responses to spurious packets. A REJ bit or command should be provided for this purpose.

Selecting Unique Sequence Numbers

The assumption that networks may generate duplicate packets with possibly long lifetimes renders the task of providing for unique sequence number non-trivial. The period of time over which a particular sequence number may refer to a particular byte requires knowing an upper bound on the packet lifetime in the network. This bound must be moderately tight because, as this bound becomes looser, either packet overhead must increase due to the need for more bits of sequence number or transmission rate must be restricted so that sequence number are not reused before packets die out in the network. The period during which a particular sequence number refers to a particular byte is then some small multiple of this packet lifetime (depending on how many reverberations the protocol will support). I will use T as a parameter of the protocol design which designates this small multiple of the packet lifetime.

The size of the window plays a part in determining whether a late packet might be confused with current packets. A large window increases the maximum sequence number the receiver will accept as valid. The maximum window size is a design parameter of the protocol and must be specified before the protocol is complete. For our purposes, the window size can be conveniently thought of as extending the apparent packet lifetime (T) and will not be discussed further here.

Also subsumed into the parameter T is the maximum host service interruption time. If a host stops executing for a period of time and then resumes with no loss, any packets held in that host during that period, whether they are data packets or acknowledgement packets, are effectively delayed by the duration of the service interruption. This delay is indistinguishable from any delay caused by the network itself.

Another parameter of the protocol design is the maximum data rate. The maximum data rate design parameter may be less than or greater than the data rate achievable by the network and associated hardware. The design parameter will probably ultimately be less than the hardware/network limitation as technology improves. If this is the case, software control will have to be used to limit the actual data rate.

Given the maximum data rate (R) and the maximum packet lifetime (T) and the maximum sequence number (M), we can define the minimum sequence number cycle time (C) as

$$C = M/R$$

and state that this must be larger than the maximum packet lifetime

$$C > T$$

This inequality must be guaranteed otherwise packets with the same sequence numbers, but from an earlier cycle, may still exist in the network.

The selection of values for some of these parameters is arbitrary. One could select a field width for sequence numbers thus determining M and compute R from that. T is determined by the network characteristics. The amount by which C exceeds T must also be selected. The amount by which C exceeds T determines how frequently sequence numbers must be resynchronized when activity is low (see below). Therefore, C should be substantially greater than T.

Since the choice of sequence numbers is directly under control of the sender, it is best to place the responsibility for selecting unique sequence numbers on the sender. The receiver then accepts a packet on the basis of whether the sequence number falls within the current window or not. If the receiver has no current window then the handshake described above is used to establish one.

To prevent reusing a particular sequence number too soon it is necessary for the sender to have knowledge about when that sequence number was last used. Figure 1 illustrates the situation when complete knowledge of this sort is available. The curve of actual sequence numbers used as a function of time permits a region of forbidden sequence number vs. time points to be defined. The curve of actual sequence numbers is not permitted to reenter this region. Complete knowledge of this kind requires a prohibitive amount of storage.

Another possibility is to retain a few numbers which permit the sequence number curve to be bounded. This scheme and the previous one require a memory which persists at least as long as T even in the event of system crashes or other memory destroying events.

The method I will describe uses a time-of-day clock to govern the selection of sequence numbers. The choice of a time-of-day clock for this purpose is natural since time is the factor which determines whether duplicates might still exist. If the time-of-day clock has its own natural period such as seconds since midnight, then it is necessary to choose the sequence cycle time (C) such that the clock period is an integral multiple of C. There are several mappings of clock to sequence number. The mapping which maintains maximum resolution is given by the expression

(V mod C)*M/C

where V is the clock value, C is the sequence cycle time in units of V, and M is the maximum sequence number. The sequence number equivalent given by this formula is called ISN (initial sequence number) in the following discussion. ISN is used for the initial sequence number for establishing or re-establishing a connection in the absence of other information.

A plot of ISN as a function of time is shown in figure 2. The step size depends on the resolution of the time-of-day clock and the range of sequence numbers required. The step size is exaggerated in the figure. Also on figure 2 are drawn another staircase preceeding the ISN curve by one step, a staircase delayed from that by C-T, and a horizontal line labelled "last used seq no.". These lines delimit a region of allowed sequence numbers. If packets are never given sequence numbers outside of this region, there will never be any problem with confusing late arriving packets with current packets. This is because the sequence number of a late arriving packet will be outside the allowed region either because the current sequence numbers have gone beyond it thus raising the lower boundary or because it is impossible for enough time to have passed to place it in the allowed region of the next cycle. Remember that the next cycle will not occur until an amount of time greater than T has passed and, by definition, the packet cannot persist for that length of time.

The sequence number constraints may be summarized as follows:

a. The current sequence number must not "get ahead" of the ISN clock by more than one step because packets with those sequence numbers from the previous sequence number cycle may be left in the network.

b. The current sequence number must not "fall behind" the ISN clock by more than C-T-<one clock step> because duplicates of packets generated with such sequence numbers may appear later and be confused with the (then current) ISN.

c. Potential sequence numbers must increase monotonically even when the connection is inactive because there will almost certainly be a conflict if there are any packets remaining in the network.

Referring to figure 2 again, note that one of the bounds of the allowed region is the largest (last) used sequence number. This would seem to require that that number be remembered at least as long as its reuse might cause confusion (T). This would require excessive amounts of storage in all except a few special cases. It is necessary to remember it for a short time because it may exceed the ISN curve. This could be avoided by moving the ISN curve to the extreme left edge of the allowed region. Doing this, however, makes it immediately impossible to use the sequence number selected from the ISN curve for communication until the next tick of the clock. This is, in fact, the reason why the ISN curve in figure 2 is shown displaced from the left edge of the allowed region.

The solution to this possible dilemma is to remember the last used sequence number if it is greater than ISN until the next clock tick. At this point the ISN curve will necessarily exceed the last used sequence number since, when the connection was active, the sequence numbers would not have been permitted to exceed the ISN curve by more than one tick since doing so would have placed the sequence numbers outside of the allowed region. In the event of a memory destroying event, such as a system restart, it is necessary to wait for one tick of the ISN because that insures that an initial sequence number selection will be greater than the sequence numbers in use just prior to the memory loss. The rules for selecting initial sequence numbers are as follows:

1. Remember the last used sequence number at least until the next tick of ISN.

2. Inhibit initial sequence number selection until at least one ISN tick has occurred following a memory loss (system restart).

3. If the last used sequence number is known, use it (incremented by 1) for a new initial sequence number.

4. If no last used sequence number is being remembered and rule 2 is satisfied, use ISN for a new initial sequence number.

Once an initial sequence number is selected, packets are given sequence numbers as specified by the protocol. Sequence numbers so selected must not equal or exceed ISN plus one step.

If data flows at less than maximum rate for a long enough time, the current sequence number would cross into the forbidden region on the right. Packets emitted with these sequence numbers and delayed by time T would conflict with a later value of ISN. This must be prevented by resynchronizing the sequence numbers to a larger value. In order to resynchronize sequence numbers, the current sequence numbers must be released and then a new sequence (using sequence number ISN) established.

Examples of Connection Synchronization

The dialogs below illustrate the interchange of letters that occurs in various situations. Each line of the dialog consists of a packet label in parentheses followed by the activity at process A where "<--" signifies the packet being received by A, "-->" signifies the packet being transmitted by A and "..." signifies that A is unaware of the packet at that time. Next appears a description of the packet in the form <SEQ n><control><data>. Next appears the activity at process B where "-->" signifies the packet is received by B. "<--" signifies the packet is transmitted by B. "..." signifies that B is unaware of the packet. The label is a handle on the packet for illustration purposes. When a packet is duplicated or retransmitted it will be given the same label. When the delay between transmission and reception of a packet is unimportant, the packet will be shown to be transmitted and received on the same line.

Unidirectional Connection with Sender Initiating

```
Lbl  A    Packet                                       B

(1)  -->  <SEQ O><SYN><>                               -->
          A synchronizes to O.

(2)  <--  <SEQ O><SYN, ACK O><>                        <--
          B synchronizes to O and acknowledges
          0.

(3)  -->  <SEQ O><ACK 0><10 Data Bytes>                -->
          A acknowledges B’s O and sends some
          data.

(4)  <--  <SEQ O><ACK 10><>                            <--
          B acknowledges A’s data.

(5)  -->  <SEQ 10><ACK 0><6 Data Bytes>                -->

(6)  <--  <SEQ O><ACK 16><>                            <--

(7)  -->  <SEQ 16><ACK 0, REL><20 Data Bytes>          -->
          A sends last data bytes and REL.

(8)  <--  <SEQ O><ACK 36, REL, CTL><Dummy Byte>        <--
          B acknowledges all A’s data (and the
          REL) and sends its own REL.

(9)  -->  <SEQ 36><ACK 1><>                            -->
          A acknowledges B’s REL.
```

Unidirectional with Receiver Initiating

```
(1)  -->  <SEQ O><SYN><>                               -->
          A synchronizes to 0.

(2)  <--  <SEQ O><SYN, ACK 0><14 Data Bytes>           <--
          B synchronizes to 0, acknowledges A,
```

```
                 and sends data.

     (3)  -->    <SEQ O><ACK 14><>                       -->
                 A acknowledges B’s data.

     (4)  <--    <SEQ 14><ACK 0><10 Data Bytes>          <--

     (5)  -->    <SEQ O><ACK 24><>                       -->

     (6)  <--    <SEQ 24><ACK 0, REL><5 Data Bytes>      <--

     (7)  -->    <SEQ O><ACK 29, CTL, REL><Dummy Byte>   -->

     (8)  <--    <SEQ 29><ACK 1><>                       <--
```

Minimal Exchange Sender Initiating

```
     (1)  -->    <SEQ O><SYN><21 Data Bytes>             -->
                 A synchronizes to O and sends data.

     (2)  <--    <SEQ O><SYN, ACK 21><>                  <--
                 B acknowledges A’s data and SYN’s its
                 own sequence number. Data is not yet
                 delivered to user process.

     (3)  -->    <SEQ 21><REL, ACK O><>                  -->
                 A acknowledges B’s sequence number and
                 REL’s the connection. B delivers data
                 to user process.

     (4)  -->    <SEQ 22><ACK 1>                         -->
                 A acknowledges B’s REL and drops the
                 connection. Note that if this final
                 ACK gets lost, B will resend its REL
                 and A will respond with an error
                 reply. B must interpret this as
                 equivalent to the ACK.
```

Late Duplicate SYN

```
     (1)  ...    <SEQ x><SYN><23 Data Bytes>             -->
                 B receives a late duplicate packet.

     (2)  <--    <SEQ O><SYN, ACK x+23><>                <--
                 B responds synchronizing its own
                 sequence numbers. A is not aware of
                 the connection.

     (3)  -->    <SEQ x+23><REJ, ACK O><>                -->
```

```
          A rejects the response. B throws away
          the data and resets the connection.


Late Duplicate SYN Plus Late Response

     (1)  ...  <SEQ x><SYN><22 Data Bytes>             -->
               B receives the late duplicate packet
          containing SYN.

     (2)  <--  <SEQ O><SYN, ACK x+22><>                <--
               B responds.

     (3)  ...  <SEQ x+22><ACK y><31 Data Bytes>        -->
               B receives a duplicate of the original
               response to the response to packet
               (1). Note that it ACK's y which is
               outside B's current sequence numbers.
               The packet is rejected since A's
               sequence numbers have not been
               verified and the ACK is out of range.

     (4)  -->  <SEQ x+22><REJ, ACK O><>                -->
               A rejects the response as in the
               previous example.


Simultaneous Initiation

     (1)  -->  <SEQ O><SYN><15 Data Bytes>             ...
               A synchronizes to 0.

     (2)  ...  <SEQ O><SYN><>                          <--
               Meanwhile, B synchronizes too.

     (1)  ...  <SEQ O><SYN><15 Data Bytes>             -->
               A's packet arrives.

     (3)  ...  <SEQ O><SYN, ACK O><>                   <--
               B acknowledges A's sequence number and
          resynchronizes its own since it has not
               yet received verification from A.

     (2)  <--  <SEQ O><SYN><>                          ...
               B's original packet arrives at A.

     (4)  -->  <SEQ O><SYN, ACK 0><15 Data Bytes>      -->
               A sends its first packet again but this
               time acknowledges B's sequence number.
               B is now completely synchronized.

     (3)  <--  <SEQ O><SYN, ACK O><>                   ...
```

B’s first response arrives at A. A is now completely synchronized. This continues as in the above cases.

Conclusion

The methods described above for managing use of sequence numbers on a connection assure that each byte of information is uniquely identified in spite of packet duplication and long delay in the network, and in spite of service disruptions or complete memory loss by either or both communicating hosts. It is our belief that any lesser schemes will lead to unreliable network performance and are, therefore, unsatisfactory.