IEN-74

Sequence Number Arithmetic

William W. Plummer

Bolt Beranek and Newman, Inc.
50 Moulton Street
Cambridge MA 02138

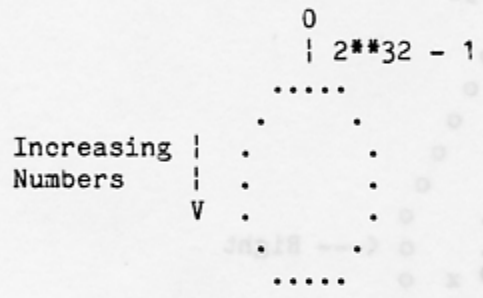
21 September 1978

1. Introduction

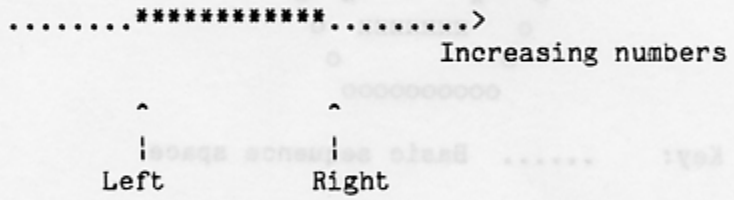
TCP deals with 32-bit numbers for sequencing and acknowledging data. A basic concept is that of a "window", a range of sequence numbers which begins at a "left" pointer and ends at a "right" pointer. Because a window may contain sequence number zero, deciding whether a given number is within a window is somewhat complicated. This paper describes some techniques for dealing with sequence numbers as they apply to TCP.

2. Representation

The space of 2**32 sequence numbers will be shown as a (rough) circle:



Segments of sequence numbers will be shown as:



Note that Left is the the first number within the segment. Right is not included. That is, the segment is semi-open. If Left = Right, the segment is considered to have zero length, not 2**32.

3. The CheckWindow Function

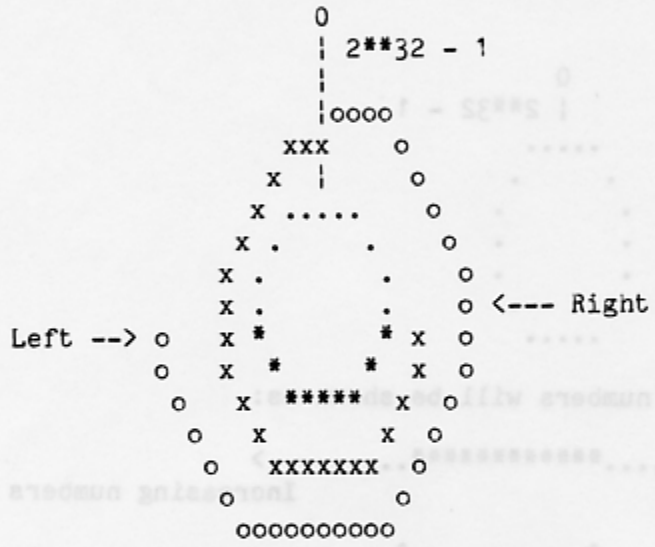
Because the sequence number space is actually a ring and arithmetic is done modulo 2**32, there is no concept of one sequence number being greater (or less) than another. The fundamental function for comparing sequence numbers is CheckWindow(Left, Sequence, Right). This function returns true if Sequence is contained in the semi-open segment between Left and Right.

For machines with word sizes greater than 32-bits or using unsigned arithmetic on 32-bit machines, the definition of CheckWindow is:

```
CheckWindow(L, S, R) := (L le R) =>
                            L le S and S lt R ,
                        not ( R le S and S lt L)
```

The second branch of the conditional is expressed in a way that it is the complement of the first branch when L and R are interchanged. Advantage is taken of this symmetry in the PDP-10 code which implements CheckWindow. Otherwise the second branch may be expressed as (R gt S) or (S ge L).

The first branch of the conditional is explained by the following diagram:



Key:

.....	Basic sequence space
******	Segment between Left and Right
xxxxxx	Sequence space where Sequence lt Right
oooooo	Sequence space where Sequence ge Left

The second branch of the conditional is the case where the segment crosses zero:

Key: ..... Sequence space

***** Segment between Left and Right

ooooo Sequence numbers ge Left

xxxxx Sequence numbers lt Right

A useful identity is: CheckWindow(L, S, R) = not CheckWindow(R, S, L). This says that either S is in the segment between L and R or it is not.

4. OverLap(L1, R1, L2, R2)

OverLap(L1, R1, L2, R2) is a predicate which tells if the two segments L1 to R1 and L2 to R2 have at least one point in common. If an overlap exists, then one segment must have its Left end within the other:

```
OverLap(L1, R1, L2, R2) :=

    CheckWindow(L1, L2, R1) or CheckWindow(L2, L1, R2)
```

Either L2 is within segment one or it is not. So either CheckWindow(L1, L2, R1) or not CheckWindow(L2, L1, R2) is true. In the first case there is an overlap even if it is just at the point L2. The second term can be rewritten:

not CheckWindow(L2, L1, R2) = CheckWindow(R2, L1, L2).

Since L2 is outside segment one, it is the position of R2 which determines whether an overlap exists. R2 can be either between L2 and L1 or it can be between L1 and L2. Thus, there are two subcases: either CheckWindow(L2, R2, L1) or CheckWindow(L1, R2, L2) must be true. In the first case there is no overlap and segment one does not contain R2. If the first case is not true then the second case must be since it is the complement of the first with the first and third arguments switched.

5. Include(L1, R1, L2, R2)

Include is true if segment one includes all of segment two. This is true only if the complement of segment one does not contain any of segment two.

Include(L1, R1, L2, R2) := not Overlap(R1, L1, L2, R2)

= CheckWindow(L1, L2, R1) and CheckWindow(R2, R1, L2)

The expansion states that L2 must lie in segment one and that the complement of segment two must contain the right end of segment one.

6. Uses Within a TCP

The functions CheckWindow, Overlap, and Include have many uses within the TCP. A few of these are described below. Some definitions are needed. A TCB contains a Send.Left cell, a Send.Window, and Send.Sequence having to do with packet generation. Send.Right does not exist explicitly but may be computed by adding Send.Left and Send.Window (mod 2**32).

The receive side of a connection has the cells Recv.Left and Recv.Window . Again, Recv.Right may be easily computed.

Each packet has its sequence number Pkt.Seq and an acknowledgement number Pkt.AckSeq. The number called Pkt.End may be computed by counting one for each control bit and data byte in the packet and adding this to Pkt.Seq mod 2**32. Note that Pkt.End is actually the sequence number following the packet. Currently only SYN and FIN occupy sequence space and these occur only at the start and end of a connection; otherwise, all sequence space is occupied by data only.

These variables define several segments. The send window between Send.Left and Send.Right, the receive window between Recv.Left and Recv.Right, and the packet segment between Pkt.Seq and Pkt.End. All of these segments are semi-open and are suitable for manipulation by the previously described functions such as CheckWindow.

The Retransmitter uses OverLap(Send.Left, Send.Right, Pkt.Seq, Pkt.End) to tell if a packet has anything transmittable in it. Note that Send.Right may lie within the segment between Send.Left and Send.Seq. This indicates that the window shrank due to Send.Right having moved "backwards". In this case data between Send.Right and Send.Seq is (temporarily) not retransmittable.

The InputProcessor makes heavy use of all of the functions. The basic acceptance test for packets arriving on an ESTABLISHED connection is OverLap(Recv.Left, Recv.Right, Pkt.Seq, Pkt.End). If this is not true, the packet is assumed to be either from the future or a duplicate from the past.

Processing the Acknowledgement field of a packet involves a scan of the retransmission queue to see which packets may be deleted. For each packet on the queue CheckWindow(Send.Left, Pkt.End-1, Acknowledgement) is true if the packet has been acknowledged. Pkt.End-1 is the sequence number of the last byte in the packet. Note that any packet on the retransmission queue must occupy at least one sequence number and therefore no special case checks must be made worry about Pkt.End-1 being less than Pkt.Seq .

TCP11 sends each newly typed character in a separate packet. Retransmissions carry all unacknowledged data. TCP for the PDP-10/20 tries to minimize internal storage requirements by saving a retransmitted packet and releasing the storage for the original transmissions. Given a incoming packet InPkt, the following test is performed against each packet queued for action by the buffer reassembler: Include(InPkt.Seq, InPkt.End, QdPkt.Seq, QdPkt.End) means that the incoming packet contains at least as much as the already queued packet and that the latter may be released.

7. Sample Code

The following routines have been excerpted from the hand coded TCP for TENEX and TOPS20. They have been included here to provide an indication of complexity. Note that the PDP-10 has a 36-bit word size and thus 32-bit numbers are always positive. Operations such as CAM which are signed compares on 36-bit quantites are unsigned operations on 32-bit numbers as required.

```
; CheckWindow(Left, Sequence, Right)

; Test "Sequence" to see if it is between "Left" (inclusive) and "Right"
; (not incl.).  Sequence numbers are  modulo MAXSEQ and are always
; positive when represented in a 36-bit word.

;T1/    Left
;T2/    Sequence
;T3/    Right
;
;       CALL CHKWND
;Ret+1: always.  T1 non-zero if Sequence is in the window

CHKWND::TEMP <VAL,SEQ,RIGHT,LEFT> ; Give names to T1, T2, T3, T4
        MOVEM T1,LEFT             ; Make T1 available for value
        SETO VAL,                 ; Init value to TRUE
        CAMG LEFT,RIGHT           ; Crosses 0?
         TDZA VAL,VAL             ; No.  Set VAL to FALSE.
         EXCH LEFT,RIGHT          ; Yes.  Reverse Left and Right.
        CAMGE SEQ,RIGHT
        CAMGE SEQ,LEFT
        CAIA
         SETCA VAL,               ; Complement VAL.
        RESTORE
        RET
```

By way of comparison, the BCPL expression for this compiles into approximately 40 instructions on the PDP-10. This expression is:

```
let CheckWindow(L, S, R) := (L le R) => (L le S < R), (R le S < L)
```

```
; Test to see if two sequence number segments have one or more common
; points.  The two segments are semi-open on the right, similar
; to CHKWND.

;T1/    Left1
;T2/    Right1
;T3/    Left2
;T4/    Right2
;
;       CALL OVRLAP
;Ret+1  always, T1 non-zero if overlap exists

OVRLAP::LOCAL <LEFT1,LEFT2,RIGHT2> ; Define some local ACs.
        MOVEM T1,LEFT1
        DMOVEM T3,LEFT2          ; T3,T4 to LEFT2,RIGHT2
        EXCH T2,T3
        CALL CHKWND
        JUMPN T1,OVRLAX
        MOVE T1,LEFT2
        MOVE T2,LEFT1
        MOVE T3,RIGHT2
        CALL CHKWND
OVRLAX: RESTORE
        RET
```