Problem 2

Suppose the IP addresses of the hosts A, B, and C are a, b, c, respectively. (Note that a, b, c are distinct.)

To host A: Source port =80, source IP address = b, dest port = 26145, dest IP address = a

To host C, left process: Source port =80, source IP address = b, dest port = 7532, dest IP = c.   right: Source port =80, source IP address = b, dest port = 26145, dest IP =c

Problem 3

```
    0 1 0 1 0 0 1 1
+   0 1 1 0 0 1 1 0
-----------------------
    1 0 1 1 1 0 0 1
+   0 1 1 1 0 1 0 0            ---> wrap around if overflow
-----------------------
    0 0 1 0 1 1 1 0
```

One's complement = 1 1 0 1 0 0 0 1.
To detect errors, the receiver adds the four words (the three original words and the checksum). If the sum contains a zero, the receiver knows there has been an error. All one-bit errors will be detected, but two-bit errors can be undetected (e.g., if the last digit of the first word is converted to 0 and the last digit of the second word is converted to 1).

Problem 14

In a NAK only protocol, the loss of packet *x* is only detected by the receiver when packet *x+1* is received. That is, the receivers receives *x-1* and then *x+1,* only when *x+1 is* received does the receiver realize that *x* was missed. If there is a long delay between the transmission of x and the transmission of *x+1,* then it will be a long time until *x* can be recovered, under a NAK only protocol.

On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent – a significant reduction in feedback in the NAK-only case over the ACK-only case.

Problem 15

It takes 12 microseconds (or 0.012 milliseconds) to send a packet, as $1500*8/10^9$=12 microseconds. In order for the sender to be busy 98 percent of the time, we must have

$util = 0.98 = (0.012n)/30.012$

or n approximately 2451 packets.

Problem 22

a) Here we have a window size of N=4. Suppose the receiver has received packet k-1, and has ACKed that and all other preceding packets. If all of these ACKs have been received by sender, then sender's window is [k, k+N-1]. Suppose next that none of the ACKs have been received at the sender. In this second case, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window is thus [k-N, k-1]. By these arguments, the sender’s window is of size 4 and begins somewhere in the range [k-N, k]. It could be: {k-4, k-3, k-2, k-1}, {k-3, k-2, k-1, k}, {k-2, k-1, k, k+1}, {k-1, k, k+1, k+2} and {k, k+1, k+2, k+3}.

b) If the receiver is waiting for packet k, then it has received (and ACKed) packet k-1 and the N-1 packets before that. If none of those N ACKs have been yet received by the sender, then ACK messages with values of [k-N, k-1] may still be propagating back. Because the sender has sent packets [k-N, k-1], it must be the case that the sender has already received an ACK for k-N-1. Once the receiver has sent an ACK for k-N-1 it will never send an ACK that is less that k-N-1. Thus the range of in-flight ACK values can range from k-N-1 to k-1.

Problem 24

a) True. Suppose the sender has a window size of 3 and sends packets 1, 2, 3 at $t0$. At $t1$ $(t1 > t0)$ the receiver ACKS 1, 2, 3. At $t2$ $(t2 > t1)$ the sender times out and resends 1, 2, 3. At $t3$ the receiver receives the duplicates and re-acknowledges 1, 2, 3. At $t4$ the sender receives the ACKs that the receiver sent at $t1$ and advances its window to 4, 5, 6. At $t5$ the sender receives the ACKs 1, 2, 3 the receiver sent at $t2$. These ACKs are outside its window.

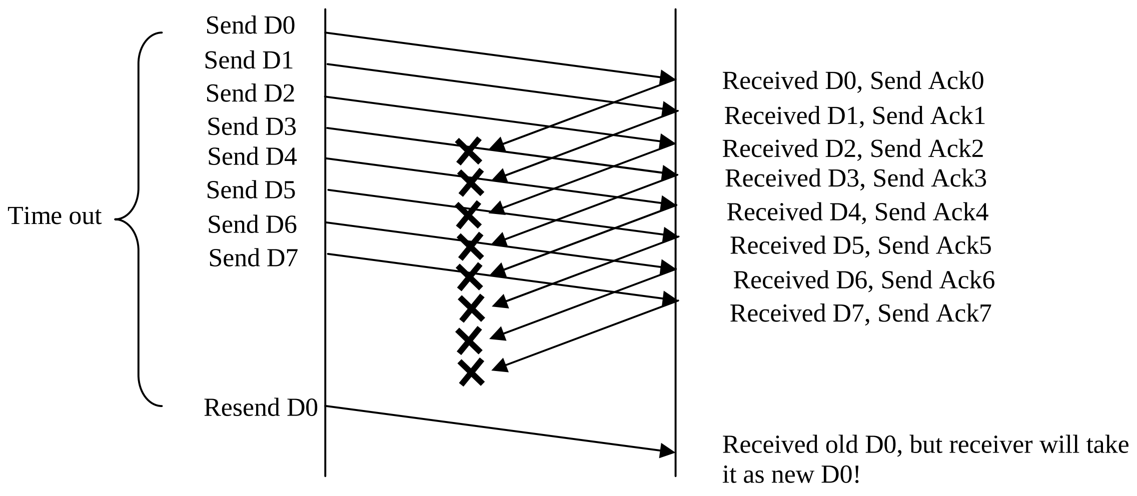
b) True. By essentially the same scenario as in (a).

c) True.

a) True. Note that with a window size of 1, SR, GBN, and the alternating bit protocol are functionally equivalent. The window size of 1 precludes the possibility of out-of-order packets (within the window). A cumulative ACK is just an ordinary ACK in this situation, since it can only refer to the single packet within the window.

Extra problem

The key is that your example should be able to show that the receiver is confused at some point about the coming packet since the sequence number is not enough to differentiate a new (old) packet from an old (new) one. Here is one example.


Send D0
Send D1
Send D2
Send D3
Send D4
Send D5
Send D6
Send D7
Time out
Resend D0
Received D0, Send Ack0
Received D1, Send Ack1
Received D2, Send Ack2
Received D3, Send Ack3
Received D4, Send Ack4
Received D5, Send Ack5
Received D6, Send Ack6
Received D7, Send Ack7
Received old D0, but receiver will take it as new D0!