Problem 1

Host A sends 5 data segments to Host B, the 2nd segment (sent from A) is lost. This loss is then detected by the protocol-specific means and the protocol invokes recovery actions. In the end, all 5 data segments are correctly received by Host B.

1. If A and B use Go-back-N for the data delivery, what is the total number of segments that Host A sent out (including retransmissions)? And what is the total number ACKs that Host B sent?
2. If A and B use TCP for the data delivery (no delayed ACKs), what is the total number of segments that Host A sent out (including retransmissions)? And what is the total number ACKs that Host B sent?
3. Assume that the retransmission timer is set to 10*RTT, which of the above two protocols (Go-back-N and TCP) finish the data delivery first?

Note: Answering questions 2 and 3 requires the knowledge about TCP fast retransmit, which is described on Lecture-7, slides 24-25, that will be discussed in next Monday lecture. If anyone wants to finish the homework this weekend, you probably can figure out how TCP fast retransmit works by taking a careful look at Lecture-7, slides 24-25.

1. Assuming we are using a Go-back-5 protocol recovery scheme, Host A sends out 9 segments: 1 2 3 4 5 2 3 4 5. Host B sends 8 ACKs: 1 1 1 1 2 3 4 5. In the first wave, A sends out 5 segments. Because the 2nd segment is lost, the ACKs for segments 3, 4, and 5 are all ACKs for segment 1, the last in-order segment. After the timeout associated with segment 2 expires, host A retransmits segments 2, 3, 4, and 5, and host B ACKs each received segment successfully.
2. Host A sends out 6 segments: 1 2 3 4 5 2. Host B sends 5 ACKs: 2 2 2 2 6. A sends out first 5 segments. The moment the 3rd duplicate ACK with expected sequence number 2 is received by A, A retransmits 2. B ACKs 2 to acknowledge the first segment followed by 3 more 2's to show the next expected sequence number. Then it successfully ACKs with 6 after receiving the retransmitted segment 2 because 6 is the next expected sequence number.
3. TCP will finish the data delivery first because the TCP fast retransmit resends the detected lost segment before the timer expires, whereas Go-back-N waits for the timer associated with segment 2 to timeout before resending all unACKed segments. TCP finishes in about $2RTT+$ delivery time for segments. Go-back-N finishes in about $11RTT+$ delivery time for segments. $2RTT < 11RTT$, so TCP finishes the data delivery first.

Problem 2

Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes up through byte 126. Suppose Host A then sends two segments to Host B back-to-back. The first and second segments contain 80 and 40 bytes of data, respectively. In the first segment, the sequence number is 127, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

1. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?
2. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?
3. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?

1. Sequence number: $127 + 80 = 207$
   Source port number: 302
   Destination port: 80
2. Acknowledgement number: 207
   Source port number: 80
   Destination port: 302
3. Acknowledgement number of first arriving segment if second segment arrives first: 127. This is Host B's way of telling Host A that it is still waiting for the packet with the sequence number of 127.

Problem 3

Follow the same problem setting in Page 37 of Slides lecture-06. Suppose packet size is 4000 bits, bandwidth is 2Mbps, and propagation delay is 15 msec. Ignore packet loss.

1. Suppose window size is 10, will the sender be kept busy? If yes, explain why. If not, What is the effective throughput?
2. What is the minimum window size to achieve full utilization? Then how many bits would be needed for the sequence number field?

1. The sender will not be kept busy. The first packet takes $t = \frac{L}{R} = \frac{4000}{2\times10^6} = 0.002s$ to transmit all of the its bits. For a window size of 10, that means it takes $10 \times \frac{L}{R} = 0.02s$ to finish sending all 10 packets. The $RTT = 2 \times d_p = 2 \times 15ms = 0.03s$. This means the sender won't send its 11th packet until $t = \frac{L}{R} + RTT = 0.002s + 0.03s = 0.032s$. Because the sender finishes sending 10 packets at $t = 0.02s$ and it won't send its 11th packet until $t = 0.032s$, the sender won't be kepy busy. Instead, it will be idle for $t = 0.032s - 0.02s = 0.012s$.

   The effective throughput can be calculated with $u_s \times R$, where u_s is the total utilization of the sender and R is the bandwidth of the network. $u_s = W \times \frac{\frac{L}{R}}{RTT+\frac{L}{R}}$, where W is the window size in packets and L is the packet size in bits.
   $u_s = 10 \times \frac{0.002}{0.03+0.002} = 0.625$
   Thus, the effective throughput is $u_s \times R = 0.625 \times 2Mbps = 1.25Mbps$

2. The minimum window size needs to be $\frac{\frac{L}{R}+RTT}{\frac{L}{R}} = \frac{0.002s+0.03s}{0.002s} = 16$ to achieve full utilization. To calculate the sequence bits needed, we can use the equation $windowsize <= \frac{(maxseqnum+1)}{2}$. Thus, $maxseqnum >= 2 \times windowsize - 1 = 2 \times 16 - 1 = 31$. Having 5 bits to represent the sequence numbers produces a maximum sequence number of $2^5 = 32 >= 31$. Thus, the minimum number of bits to represent the sequence numbers for a window size of 16 packets would be 5 bits.

Problem 4

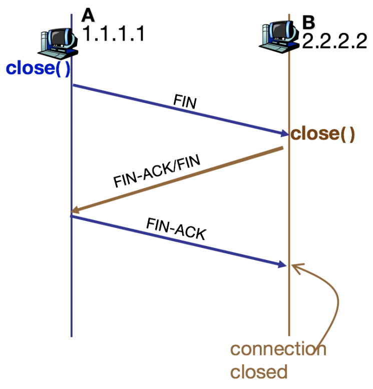
Consider a reliable data transfer protocol that uses only negative acknowledgments. Suppose the sender has a lot of data to send and the end-to-end connection experiences few losses. Would a NAK-only protocol be preferable to a protocol that uses ACKs? Why? Now suppose the sender sends data only infrequently. In this second case, would a NAK-only protocol be preferable to a protocol that uses ACKs? Why?

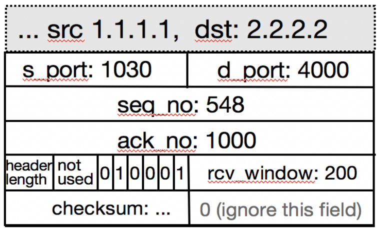
A NAK-only protocol would be preferable to a protocol that uses ACKs if a lot of data is send and the end-to-end connection experiences few losses. A lot of overhead is saved because there is no need to ACK all of the successfully received packets. For the few losses that do happen, the NAK-only protocol would be able to quickly communicate which packets did not successfully get sent because of how many packets are sent. The sender would thus only need to retransmit the few lost packets. In this scenario, all of the packets would eventually be sent successfully with minimal network overhead. If you imagine a protocol with ACKs, each successful packet sent would require an explicit ACK, which significantly increases the network traffic. The extra work in making sure packets are sent don't save much time, and thus a protocol with ACKs is not worth the overhead or complexity if it is only going to detect only a few losses.

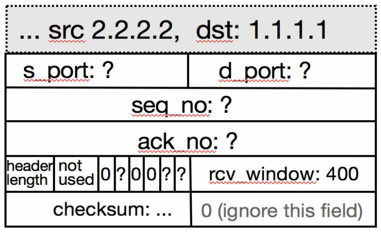
If the sender only sends data infrequently, we would prefer a protocol with ACKs, because of the guarantee of knowing when a packet is lost relatively quickly. Under a NAK-only protocol, we can only detect if packet p is lost once we receive packet $p + 1$. If packet $p - 1$ is received, there is a chance that packet $p + 1$ does not arrive until much later. That means that during this waiting period, we would not be able to tell if packet p was lost until we finally receive packet $p + 1$. On the other hand, if we have a protocol with ACKs, we know relatively soon if packet p was lost by detecting if we receive an ACK from the receiver.

Problem 5

A sends a TCP FIN message to B to close the TCP connection with B, the TCP header of A's FIN message is shown below. When B receives A's TCP FIN, it also decides to close the connection, so B sends a combined FIN and FIN-ACK message, whose TCP header is also shown below. Please fill in all the fields with a question mark in this TCP header.







`s_port:` 4000
`d_port:` 1030
`seq_no:` 1000
`ack_no:` 549
`control bits (6-bit):` 010001