Problem Set 6

Title: Reliability (Go-Back-N Protocol, Sliding Window Protocol), TCP

Due: start of class, Friday, April 21st

All problems carry equal weight. To receive full credit, show all of your work. Answers MUST be typed in. Handwritten solutions will not be accepted. You can use Word, or any other software you would like to prepare your solution. Do not change the order of questions. Your solution must be converted to PDF before handing it in. Other file types (**.doc, .docx.,** *etc*) will not be graded. MacOS and Linux systems come already with PDF converters installed; there are various free software solutions to print PDFs available on the internet

Sequence Number Space

1. Consider the Go-Back-N protocol with a send window size of N and a sequence number range of 4096. Suppose that at time *t*, the next in-order packet that the receiver is expecting has a sequence number of *k*. Assume that, the medium may drop packets but does not reorder messages.

    a. What are the possible sets of sequence number inside the sender's window at time *t*? Justify your answer.

    b. What are all possible values of the ACK field in the message currently propagating back to the sender at time *t*? Justify your answer.

    c. With the Go-Back-N protocol, is it possible for the sender to receive an ACK for a packet that falls outside of its current window? Justify your answer with an example.

Sol

a. Here we have a window size of N. Suppose the receiver has received packet k-1, and has ACKed that and all other preceding packets. If all of these ACK's have been received by the sender, then the 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 senders window is of size N and begins somewhere in the range [k-N,k].

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 to k-1.

c. Yes. 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.

Sliding Window Protocols

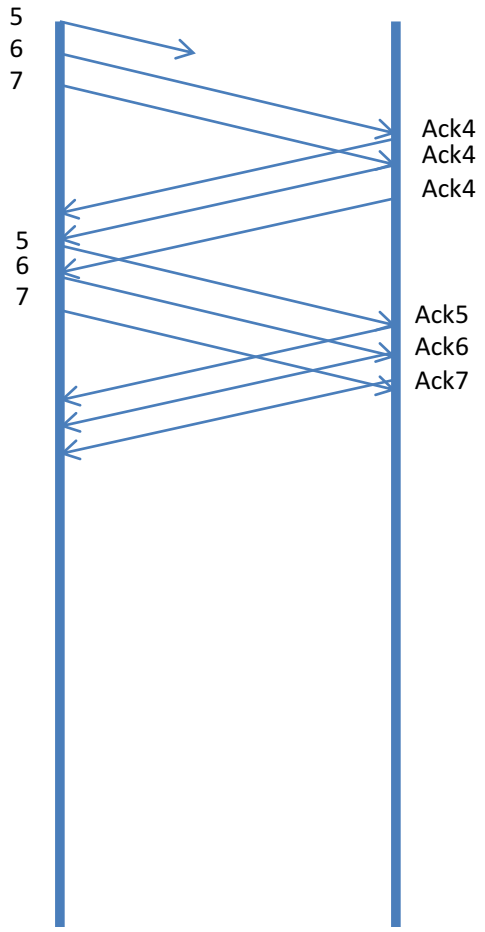
2. This question concerns the different trade-offs with sliding window protocols parameters.

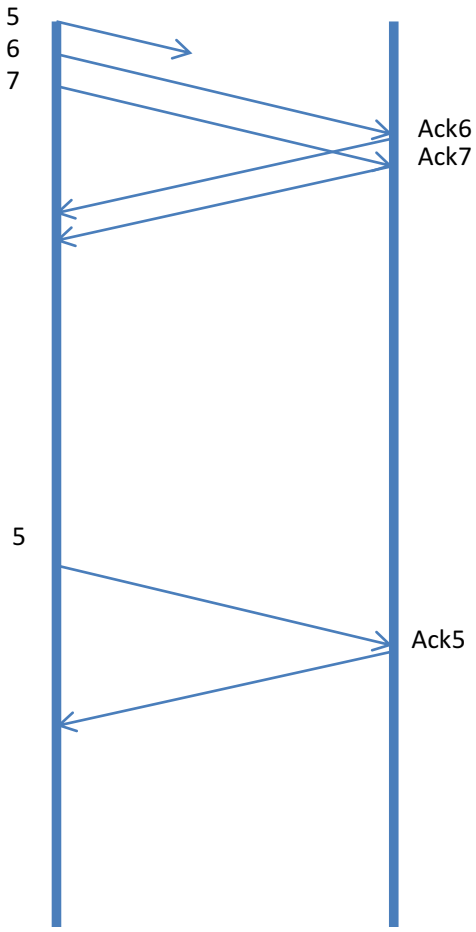
   a. Draw a timeline for the sliding window algorithm with SWS = RWS = 3 frames for the following two situations. Use a timeout interval of about 2 x RTT.
      i. Frame 5 is lost.
      ii. Frames 5-7 are lost.

   b. Consider the sliding window algorithm with SWS = RWS = 5 and NumSeqNumbers = =9. The Nth packet DATA[N] contains N mod 9 in its sequence number field. Show an example in which the algorithm becomes confused. No packets may arrive out of order. Note, this implies that NumSeqNumbers >= 10 is necessary and sufficient for correction operation of a sliding window protocol.

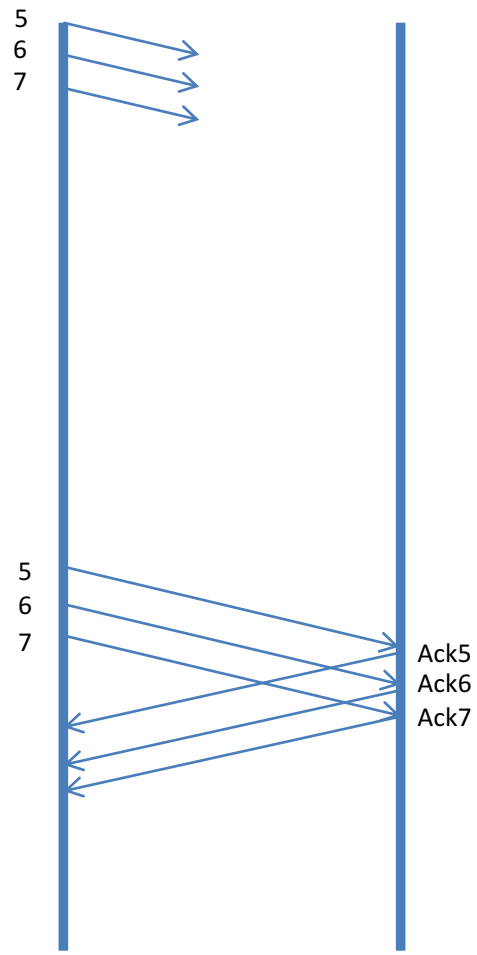
a.

i) Possible solutions are:





ii)



b. Sender transmits DATA[0 .. 4] first with sequence numbers 0 .. 4. Receiver receives all frames and sends a cumulative ACK which is lost. So, the sender retransmits DATA[0..4] with sequence numbers 0 .. 4. Now receiver is waiting for frames with sequence numbers 5, 6, 7, 8, 0. So, the receiver thinks it received frame 0 (DATA[9]) while it has received DATA[0]. So, the receiver makes an error.

3. You want to transfer a file from Champaign to Chicago. For this problem, assume:

   - The size of file is 120,000 bytes. It will be transferred in 1,500-byte data packets, for which 80 bytes are taken up with headers. The size of acknowledgement packets is 150 bytes including header. Every packet is acknowledged.
   - The communication is bidirectional and the bandwidth is 100 Mbps (megabits/sec) in each direction. Hosts can send and receive at the same time.
   - The propagation time from Champaign to Chicago is 6 ms, as is the propagation time in the opposite direction.
   - The time to process a packet is very small, so the receiver can send an acknowledgement as soon as it receives a data packet, but not before it receives the entire data packet.
   - Likewise, for sliding window, the sender can send packets back-to-back (to the degree that the window size permits).
   - Assume there is no packet loss, corruption or reordering.

   a. How long will it take to transfer the file using Stop-and-Wait?

   b. What is the corresponding throughput (total amount of user data transferred = 120,000 bytes over the total time required to transfer it).

   c. How long will it take to transfer the file using a sliding window protocol with SWS=RWS=20. What is the throughput?

   d. If SWS and RWS are set to the 2 times the bandwidth-delay product (bandwidth of the path times the roundtrip propagation time of the path), then how long does it take to transfer the file, and with what throughput?

   e. If SWS and RWS are set to three times the bandwidth-delay product, how long does the transfer take, with what throughput, and why?

   Sol: Each data packet holds 1,420 bytes of payload, so to transfer 120,000 bytes of payload will require 85 full-sized packets.

   For a full-sized packet, the time to send it from Champaign to Chicago is:
   T_data = 6ms + 1500 bytes x 8 / 100 Mbps = 6.12 ms

   Similarly, the time to send back a 150-byte acknowledgement is T_ack = 6.008 ms.
   The round-trip time (RTT) for a full-sized packet is therefore: RTT = T_data + T_ack = 6.12 ms + 6.008 ms = 12.128 ms

   a. To send the file with Stop-and-Wait, each full-sized round requires time RTT = T_data + T_ack. We include the final partial packet to this to get:

   *T_stop-and-wait = 85 packets * 12.128 ms = 1030.88 ms*

b. The throughput is the amount of data transferred divided by the time it took to send it, so:

Throughput = 120000 x 8 bits / 1.03088 s = 0.931243 Mbps

c. The first twenty packets will be sent back-to-back. The first ack comes back at a RTT which is 12.132 ms. The second ack is sent at T_2 = 1500 x 8 bits / 100 Mbps = 0.12ms. Therefore its ack is coming back at

T_2 + RTT = 0.12 ms + 12.132 ms = 12.252 ms.

After the 1st ack comes back, the sender starts to send another 20 packets back-to-back, the ith packet in this 20 packet is sent 1 RTT time away from the ack of the ith packet in the first 20 packets. E.g., packet #22 is sent at 0.12 ms + 12.252 ms = 12.372 ms.

*Since there are 84 full packets, we know that #84 packet is sent at T_4 + 4 x RTT = (0.12 * 3) + (4 x 12.132) ms = 48.888 ms.*

The partial packet is sent at 0.12ms + 48.888ms = 49.008ms. The RTT for the partial packet is (T_partial + T_ack) = 6.064 + 6.012 = 12.076. The sending finishes at:

49.008 + 12.076 = 61.084ms

*The ACKs for the last 5 packets will arrive the sender at 51.561ms + (1500 bytes x 8 / 100 Mbps) * 5 = 61.24 ms*

The throughput is 120000 x 8 bits / 61.084 ms = 15.7161 Mbps

d. The bandwidth-delay product of this path is

100 Mbps x 12 ms = 150 KB

A window size of 300 KB corresponds to sending the entire file in a single burst. We can double-check this as follows.

The total time to receive the last ACK from the receiver will take:

$$85 * 0.06 + 12.128\ ms = 17.228\ ms$$

The throughput is

120000 x 8 bits / 17.228ms = 55.723 Mbps

e. *Setting the window any higher won't change the throughput we can achieve, since the window computed using the bandwidth-delay product already allows us to send all our data in a single flight. More generally, using the bandwidth-delay product for the window allows us to completely "fill the pipe, namely to keep the forward transmission path continuously occupied. Any larger window can't enable us to go any faster, since we're already going as fast as the network can possibly let us go. (The larger window can lead to larger queues inside the network, since we give the network more load to process in a given amount of time. Ironically, this can lead to some of our packets being dropped due to exhausted queue space, which, as we'll see when we study congestion control, can actually cause the transfer to go much slower.)*

Delay-Bandwidth Product for Links in Series

4. Consider four nodes A, B, C, D connected in series. Node A is connected to node B via a 1.5Gbps link, 300km in length. Node B is connected to node C via 60 Mbps link, 35 km in length. Node C is connected to node D via a 85 Mbps 25 km in length. The links are fiber optic and full duplex. The rate of transmission errors on the links, the switching time at B, and the time to transmit an ACK are all negligible and can be ignored. A large file is to be sent from node A to node D, and there is no other traffic on the links. Packets are 1024B, including headers.

    a. Ignoring reliability and packet headers, what is the maximum throughput that can be achieved? Explain.

    b. What is the round-trip time from A to D?

    c. What is the round-trip bandwidth delay product for the path from A to D? (Specify the units you use.)

    d. Suppose an end-to-end sliding window protocol is used with SWS = RWS. What is the optimal value for SWS?

    e. What would happen if you use an SWS value many times larger than the value you suggested in part (d)?

Sol.

a. min{1.5 Gbps, 60 Mbps, 85 Mbps} = 60Mbps

b. Since the processing time and ACK transmit time are negligible, we have
*RTT = transmit time + 2 * propagation delay*
*= 8192/(1.5 x 10^9) + 8192/(60 x 10^6) + 8192/(85 x 10^6) + 2 * ((300 x 10^3)/(2 x 10^8) + (60 x 10^3)/(2 x 10^8))*
= 3.83837 ms
or
(using speed of light) = 2.63837 ms

*c. 60Mbps * 3.83837 ms = 230302.2 bits = 28787.775 B = 28.11 packets*
or
(using speed of light) = 19.324 packets

d. Since the round-trip pipe holds 28.11 packets, SWS = RWS = 29 would be optimal. So 5 bit sequence numbers suffice.
or
(using speed of light) SWS=RWS=20 with 5 bit

e. If SWS is very large it can cause the queue at B to be very large. If the buffer space at B is

limited, it could cause B to drop packets.

TCP RTT Estimation

5. One difficulty with the original TCP SRTT estimator is the choice of an initial value. In the absence of any special knowledge of network conditions, the typical approach is to pick an arbitrary value, such as 3 seconds, and hope this will converge quickly to an accurate value. If this estimate is too small, TCP will perform unnecessary retransmissions. If it is too large, TCP will wait a long time before retransmitting if the first segment is lost. Also, the convergence might be slow.
    a. Choose α = 0.7 and SRTT(0) = 1 seconds, and assume all measured RTT values = 0.5 second with no packet loss. What is SRTT(15)? Recall,

$$SRTT(k+1) = \alpha \cdot SRTT(k) + (1-\alpha) \cdot RTT(k+1).$$

    Describe your solution approach AND provide the numerical result (approximate to the 4th decimal place)

    b. Using the same values as in part a), what happens if we use α=0.5 or α=0.9? Provide a numerical result for SRTT(15) in both cases, then describe the effect of a larger or smaller α on the RTT estimation procedure.

    c. What is the retransmission ambiguity problem addressed by the Karn-Partridge algorithm? How does the algorithm avoid the ambiguity?

Sol:

a. Based on the formula and α = 0.7, we can calculate the nth SRTT value using:

$$SRTT(n) = \alpha^n * SRTT(0) + (1-\alpha) * RTT * \sum_{j=0}^{n-1} \alpha^j$$

So with the given initial conditions, we get $SRTT(15) = 0.5024$

b. Based on the formula and α = 0.5, we can get $SRTT(15) = 0.5000$, and similarly, if α = 0.9, $SRTT(15) = 0.62095$ From the calculation, we can conclude that A larger α puts more weight put on the initial value and so results in a slower convergence time to the real RTT.

c. An acknowledgement does not necessarily acknowledge the most recent transmission of a packet. Therefore, any ACKs that are received for a packet that was transmitted more than once are not used in calculating the estimated RTT.

TCP Slow Start

6. Although slow start with congestion avoidance is an effective technique for coping with congestion, it can result in long recovery times in high-speed networks.

    a. Assume a RTT delay of 50 ms and a link with an available bandwidth of 2 Gbps and a segment size of 1024 bytes. Determine the window size needed to keep the pipe full

and the time it will take to reach that window size after a timeout using the Jacobson Algorithm, assuming that the sender was transmitting at the full window before the timeout,

b. Repeat for a segment size of 20 Kbytes. (Assume 1K = 1024)

Sol:

a. The size of the window (in bits) is $50ms * 2Gbps = 100 * 10^6 bits$. *Each packet has 8192 bits, so the window size is:*

$$\left\lceil \frac{100 * 10^6 bits}{8192\ bits/packet} \right\rceil = 12208\ packets$$

After the timeout, our congestion threshold is half the full window size (6104 packets). TCP will use exponential growth until the congestion threshold is passed, at which point additive increase begins. Therefore, we can use exponential increase to a window size of $2^{12} = 4096$. *In the 13rd RTT, after receiving 2008 packet,(0.08 RTT) the window size becomes to 6104. So in the 13 RTT, we will transmit 6104 packets out. Now we do additive increase to get to the full window. This will take* $12208 - 6104 = 6104$ *RTTs using additive increase (one packet increase in the window per RTT). So, the total time to reach the full window is (13 + 6104)*RTT = 6117 *50ms = 305.85s.*

b. The window size (in bits) does not change, but we need to refresh the window size for packets of 20KB. Each packet is 163840 bits, so the window is:

$$\left\lceil \frac{100 * 10^6 bits}{163840\ bits/packet} \right\rceil = 611\ packets$$

*This makes the congestion threshold 305 packets, as what we described in a, there should be (9+306)*0.05 = 15.75 s*