Homework Chapter 3

P15

Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data.

d = data transfer
L = packet length
R = transmission rate
W = window size
RTT = speed of light round trip propagation delay
d = L/R
Usender = (W * d) / (RTT + d)
Usender = (W * .008) / 30.008 = W * 0.00027 = .90
W = 3333.33

P16

Suppose an application uses rdt 3.0 as its transport layer protocol. As the stop-and-wait protocol has very low channel utilization (shown in the cross-country example), the designers of this application let the receiver keep sending back a number (more than two) of alternating ACK 0 and ACK 1 even if the corresponding data have not arrived at the receiver. Would this application design increase the channel utilization? Why? Are there any potential problems with this approach? Explain.

Yes. This actually causes the sender to send a number of pipelined data into the channel.
Yes. Here is one potential problem. If data segments are lost in the channel, then the sender of rdt 3.0 won’t re-send those segments, unless there are some additional mechanism in the application to recover from loss.

P22

Consider the GBN protocol with a sender window size of 4 and a sequence number range of 1,024. 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 does not reorder messages. Answer the following questions:
a. What are the possible sets of sequence numbers inside the sender's window at time t? Justify your answer.
b. What are all possible values of the ACK field in all possible messages currently propagating back to the sender at time t? Justify your answer.

- there are two scenarios. Scenario 1: receiver and received and ACKed all packets up to k-1. Senders window will contain [k, k+(4-1)]. If the ACKs were not received, then it resends those packets, making the window [k-4,k]
- since the receiver is waiting for packet sequence number k, it has already received all packets up to k-1. If the sender is sending [k-4, k-1], then sender has received ACK for k-(4-1). Also, receiver will not resend this ACK meaning current ACKs propagating is [k-5, k-1]

P23

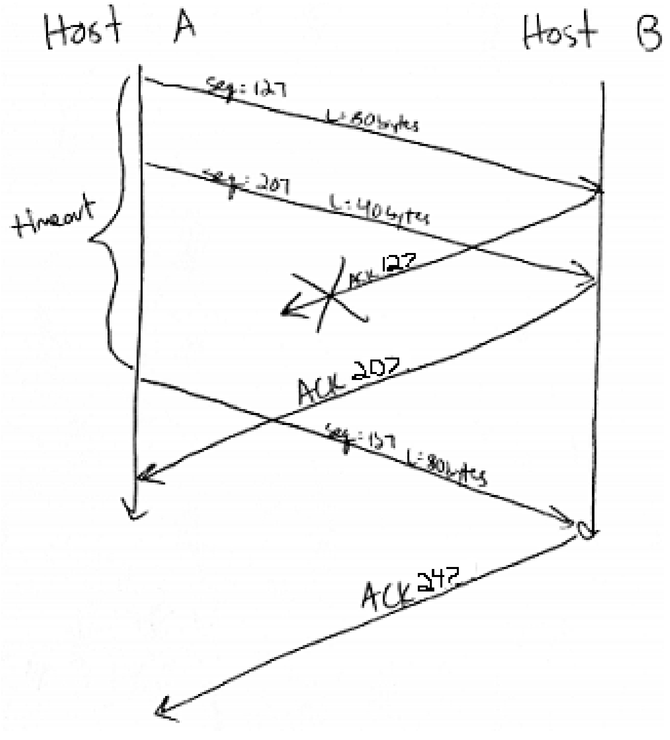
Consider the GBN and SR protocols. Suppose the sequence number space is of size k. What is the largest allowable sender window that will avoid the occurrence of problems such as that in Figure 3.27 for each of these protocols?

The sequence number space must be at least twice as large as the window size

P27

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.
a. In the second segment sent from Host A to B, what are the sequence num-

ber, source port number, and destination port number?
b. 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?
c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number?
d. Suppose the two segments sent by A arrive in order at B. The first acknowledgment is lost and the second acknowledgment arrives after the first timeout interval. Draw a timing diagram, showing these segments and all other segments and acknowledgments sent. (Assume there is no additional packet loss.) For each segment in your figure, provide the sequence number and the number of bytes of data; for each acknowledgment that you add, provide the acknowledgment number.

- sequence number = 207; src port number = 302; dest port number = 80;
- acknowledgement number = 207; src port number = 80; dest port number = 302;
- acknowledgement number = 127; //still waiting for bytes 127 and after
- 

P31

Suppose that the five measured SampleRTT values (see Section 3.5.3) are 106 ms, 120 ms, 140 ms, 90 ms, and 115 ms. Compute the EstimatedRTT after each of these SampleRTT values is obtained, using a value of α = 0.125 and assuming that the value of EstimatedRTT was 100 ms just before the first of these five samples were obtained. Compute also the DevRTT after each sample is obtained, assuming a value of β = 0.25 and assuming the value of DevRTT was 5 ms just before the first of these five samples was obtained. Last, compute the TCP TimeoutInterval after each of these samples is obtained.

1. After 100
    - EstimatedRTT = 0.875*100 + 0.125*106 = 100.75ms
    - DevRTT = 0.25*(106 - 100.75) + 0.75*5 = 5.06ms
    - TimeoutInterval = 100.75 + 4*5.06 = 120.99ms
2. After 106
    - EstimatedRTT = 0.875*106 + 0.125*120 = 103.15ms
    - DevRTT = 0.25*(120 - 103.15) + 0.75*5 = 8ms
    - TimeoutInterval = 103.15 + 4*8 = 135.15ms
3. After 120
    - EstimatedRTT = 0.875*120 + 0.125*140 = 107.76ms
    - DevRTT = 0.25*(140 - 107.76) + 0.75*5 = 14.06ms
    - TimeoutInterval = 107.76 + 4*14.06 = 164ms
4. After 140
    - EstimatedRTT = 0.875*140 + 0.125*90 = 105.54ms
    - DevRTT = 0.25*(90 - 105.54) + 0.75*5 = 14.42ms
    - TimeoutInterval = 105.54 + 4*14.42 = 163.22ms
5. After 90
    - EstimatedRTT = 0.875*90 + 0.125*115 = 106.71ms
    - DevRTT = 0.25*(115 - 106.71) + 0.75*5 = 12.88ms
    - TimeoutInterval = 106.71 + 4*12.88 = 158.23ms

P32

Consider the TCP procedure for estimating RTT. Suppose that = 0.1. Let SampleRTT 1 be the most recent sample RTT, let SampleRTT 2 be the next most recent sample RTT, and so on.

a. For a given TCP connection, suppose four acknowledgments have been

returned with corresponding sample RTTs: SampleRTT 4 , SampleRTT 3 , SampleRTT 2 , and SampleRTT 1 . Express EstimatedRTT in terms of the four sample RTTs.
b. Generalize your formula for n sample RTTs.
c. For the formula in part (b) let n approach infinity. Comment on why this averaging procedure is called an exponential moving average.

- EstimatedRTT(1) = SampleRTT
  EstimatedRTT(2) = α * SampleRTT(1) + (1-α)*SampleRTT(2)*
  *EstimatedRTT(3) = α * SampleRTT(1) + (1-α)(α*SampleRTT(2) + (1-α)*SampleRTT(3))*
  *EstimatedRTT(4) = α * SampleRTT(1) + (1-α)*EstimatedRTT(3)
- $$\alpha \sum_{i=0}^{n-1} (1-\alpha)^i \cdot \text{SampleRTT}_i + (1-\alpha)^n \cdot \text{EstimatedRTT}(n)$$
- $$\frac{\alpha}{1-\alpha} \sum_{i=1}^{\infty} (1-\alpha)^i \cdot \text{SampleRTT}_i = \frac{1}{9} \sum_{i=1}^{\infty} (0.9)^i \cdot \text{SampleRTT}_i$$

P40

Consider Figure 3.58. Assuming TCP Reno is the protocol experiencing the behavior shown above, answer the following questions. In all cases, you should provide a short discussion justifying your answer.
a. Identify the intervals of time when TCP slow start is operating.
b. Identify the intervals of time when TCP congestion avoidance is operating.
c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
e. What is the initial value of ssthresh at the first transmission round?
f. What is the value of ssthresh at the 18th transmission round?
g. What is the value of ssthresh at the 24th transmission round?
h. During what transmission round is the 70th segment sent?
i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?

j. Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size at the 19th round?
k. Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?

- 1-6, 23-26
- 6-23
- ACK
- timeout
- 32
- 21
- 13
- 7
- 4, 4
- 21, 4
- 1,2,4,8,16,21; total=52

P43

Host A is sending an enormous file to Host B over a TCP connection. Over this connection there is never any packet loss and the timers never expire. Denote the transmission rate of the link connecting Host A to the Internet by R bps. Suppose that the process in Host A is capable of sending data into its TCP socket at a rate S bps, where S = 10 · R. Further suppose that the TCP receive buffer is large enough to hold the entire file, and the send buffer can hold only one percent of the file. What would prevent the process in Host A from continuously passing data to its TCP socket at rate S bps? TCP flow control? TCP congestion control? Or something else? Elaborate.

Since the file being sent is smaller than the buffer on the receiver end, we need not worry about overflowing the buffer. There are no dropped packets, so it is not a congestion control issue. Since the send buffer is so small, it will fill up quickly. Once the buffer is full, it essentially can only send data at the transmission rate R, not at the TCP socket rate S.

P54

In our discussion of TCP congestion control in Section 3.7, we implicitly assumed that the TCP sender always had data to send. Consider now the case that the TCP sender sends a large amount of data and then goes idle (since it has no more data to send) at t 1 . TCP remains idle for a relatively long period of time and then wants to send more data at t 2 . What are the advantages and dis-advantages of having TCP use the cwnd and ssthresh values from t 1 when starting to send data at t 2 ? What alternative would you recommend? Why?

This method has an advantage because the sender sends all of it's data initially in a quick fashion. Then it proceeds with the second chunk of data slowly past its ssthresh. This is advantageous unless it receives too many duplicate ACKs. Then we see a set back in performance

P55

In this problem we investigate whether either UDP or TCP provides a degree of end-point authentication.
a. Consider a server that receives a request within a UDP packet and responds to that request within a UDP packet (for example, as done by a DNS server). If a client with IP address X spoofs its address with address Y, where will the server send its response?
b. Suppose a server receives a SYN with IP source address Y, and after responding with a SYNACK, receives an ACK with IP source address Y with the correct acknowledgment number. Assuming the server chooses a random initial sequence number and there is no "man-in-the-middle," can the server be certain that the client is indeed at Y (and not at some other address X that is spoofing Y)?

- the server will send it's response to address Y so the client at address X will not see any response. This is not a problem for establishing a connection because UDP does not require a three way handshake like TCP. Y will throw away this packet. This is common in DDoS
- the server will send it's response to address Y so the client at address X will not see any response. Client at Y cannot guess the 32 bit seq so they cannot complete the three way handshake. This is common in DDoS