HW4 Answers

1-

a)True, In particular, when the window size and bandwidth-delay product are both large, many packets can be in the pipeline. A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily.
b) False, it waits for the packets which have not been received(including packet n-1), up to packet n and then delivers them to upper layer.

c)False, *in* this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.

D) False, The TCP "connection" is not an end-to-end TDM or FDM circuit as in a circuit switched network. Nor is it a virtual circuit, as the connection state resides entirely in the two end systems. Because the TCP protocol runs only in the end systems and not in the intermediate network elements (routers and link-layer switches), the intermediate network elements do not maintain TCP connection state.

E) False, a TCP connection is also always **point-to-point**, that is, between a single sender and a single receiver. So-called "multicasting" —the transfer of data from one sender to many receivers in a single send operation—is not possible with TCP. With TCP, two hosts are company and three are a crowd!

F) True, In the TCP 3-way handshaking, The first two segments carry no payload, that is, no application-layer data; the third of these segments may carry a payload.

G) This statement is false because of the following reasons:
1. the MSS is the maximum amount of application-layer data in the segment, not the maximum size of the TCP segment including headers.
2. The MSS is typically set by first determining the length of the largest link-layer frame.

H)False, since it is a 4-bit number specifying in 32-bit words, it's maximum value would be equal to 60 bytes.

2-

a)

Recall that the receiver must deliver data in order to the upper layer. Suppose now that packet *n* is expected, but packet *n* + 1 arrives. Because data must be delivered in order, the receiver *could* buffer (save) packet *n* + 1 and then deliver this packet to the upper layer after it had later received and delivered packet *n*. However, if packet *n* is lost, both it and packet *n* + 1 will eventually be retransmitted as a result of the GBN retransmission rule at the sender. Thus, the receiver can simply discard packet *n* + 1. The advantage of this approach is the simplicity of receiver buffering—the receiver need not buffer *any* out-of-order packets. Thus, while the sender must maintain the upper and lower bounds of its window and the position of nextseqnum within this window, the only piece of information the receiver need maintain is the sequence number of the next in-order packet.
The disadvantage of throwing away a correctly received packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.

b) One manifestation of packet reordering is that old copies of a packet with a sequence or acknowledgment number of *x* can appear, even though neither the sender's nor the receiver's window contains *x*. With packet reordering, the channel can be thought of as essentially buffering packets and Spontaneously emitting these packets at *any* point in the future. Because sequence numbers may be reused, some care must be taken to guard against such duplicate packets. The approach taken in practice is to ensure that a sequence number is not reused until the sender is "sure" that any previously sent packets with sequence number *x* are no longer in the network. This is done by assuming that a packet cannot "live" in the network for longer than some fixed maximum amount of time. A maximum packet lifetime of approximately three minutes is assumed in the TCP extensions.

c) In practice, both sides of a TCP connection randomly choose an initial sequence number. This is done to minimize the possibility that a segment that is still present in the network from an earlier, already-terminated connection between two hosts is mistaken for a valid segment in a later connection between these same two hosts.

d) TCP provides a flow-control service to its applications to eliminate the possibility of the sender overflowing the receiver's buffer but in congestion control mechanism TCP sender can be throttled due to congestion within the IP network.

3.

This situation could happen if the number of errors in 16 bit numbers which XOR together is even on each column.

Assume the following example in which an error cause one of the bits to be altered:

Case 1:no error has occurred

1000000000000000

0001000000000000

0100000000000000

1101000000000000

Case2:an error causes the last bit of the first 16bit number to be changed to1:

1000000000000001

0001000000000000

0100000000000000

1101000000000001

In this situation if the last bit of one of two other 16 bit number is changed to 1 no error could be detected using this approach.

1000000000000001

0001000000000001

0100000000000000

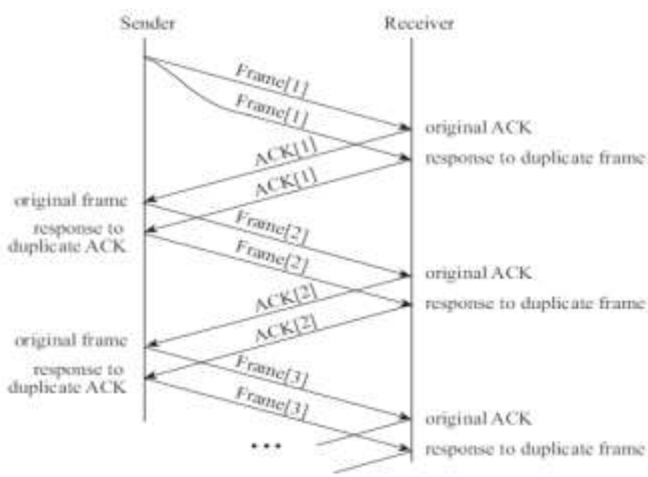
1101000000000001

According to the above example the probability of not being an specific error at the receiver is equal to:

$$\sum_{k=0}^{\frac{n}{4}+1} \binom{\frac{n}{2}+4}{2k+1} p^{2k}(1-p)^{\frac{n}{2}+4-2k}$$

4.

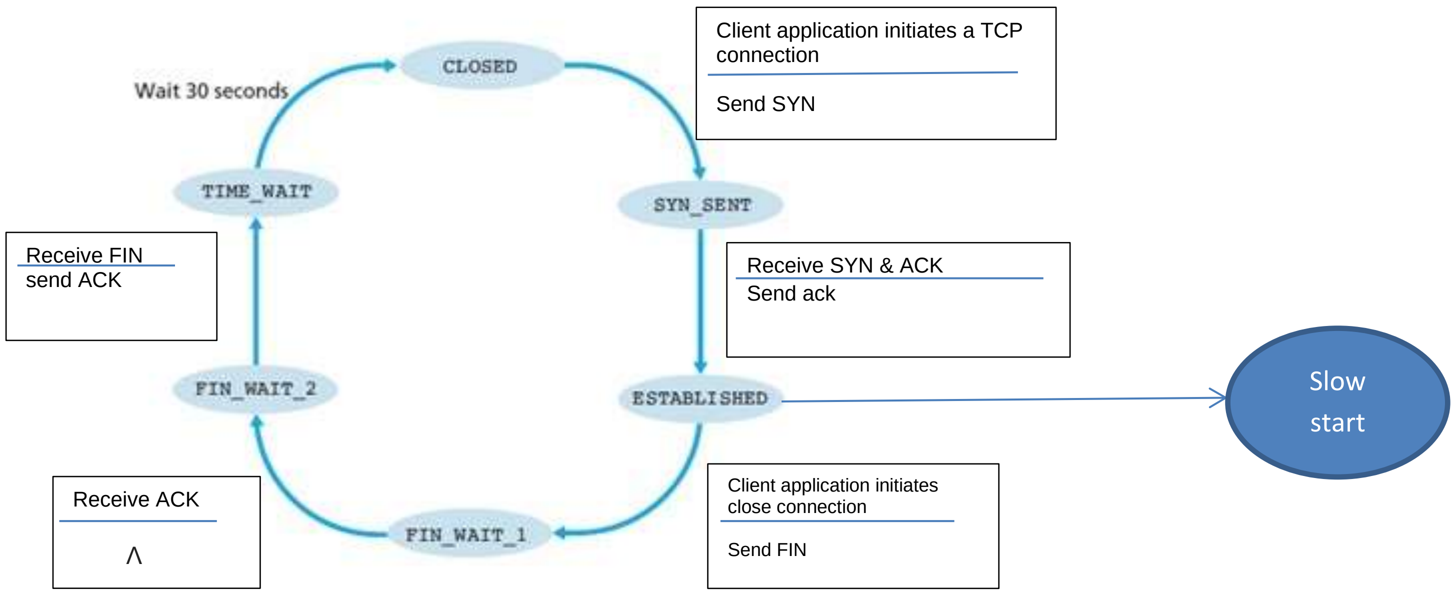
(a) The duplications below continue until the end of the transmission.



b)To trigger the sorcerer's apprentice phenomenon, a duplicate data frame must cross somewhere in the network with the previous ACK for that frame. If both sender and receiver adopt a resend-on-timeout strategy, *with the same timeout interval*, and an ACK is lost, then both sender and receiver will indeed retransmit at about the same time. Whether these retransmissions are synchronized enough that they cross in the network depends on other factors; it helps to have some modest latency delay or else slow hosts. With the right conditions, however, the sorcerer's apprentice phenomenon can be reliably reproduced.

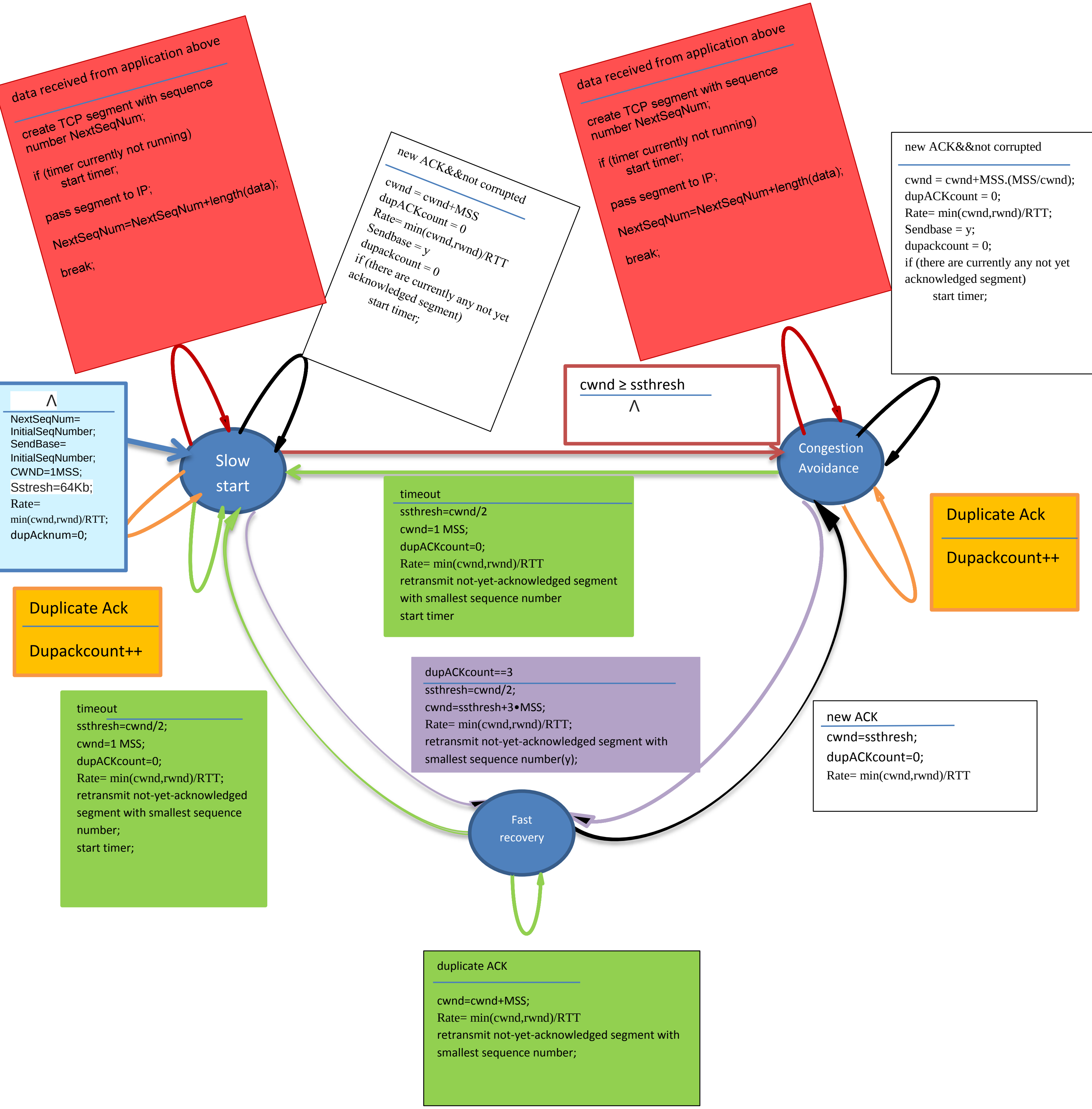
5.sender side of TCP:

Connection management:

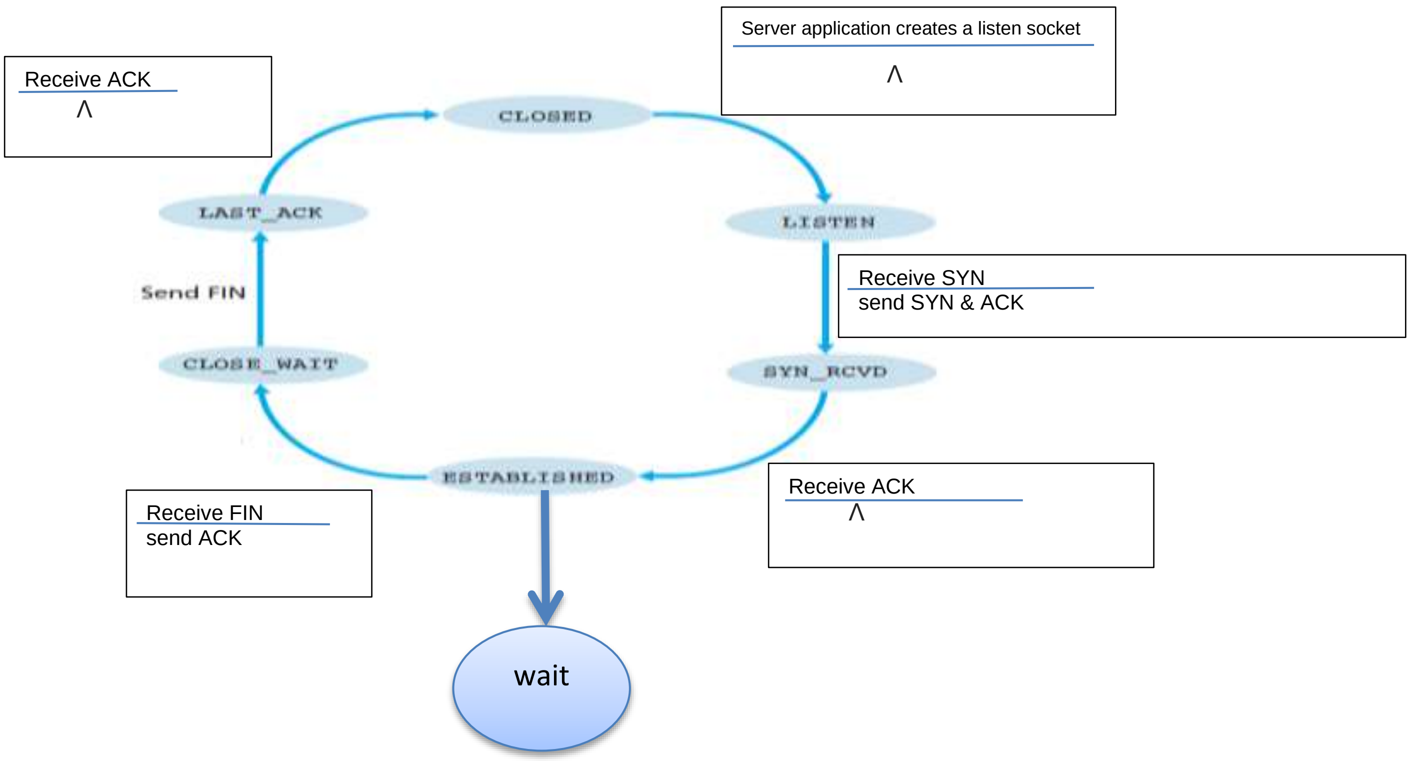


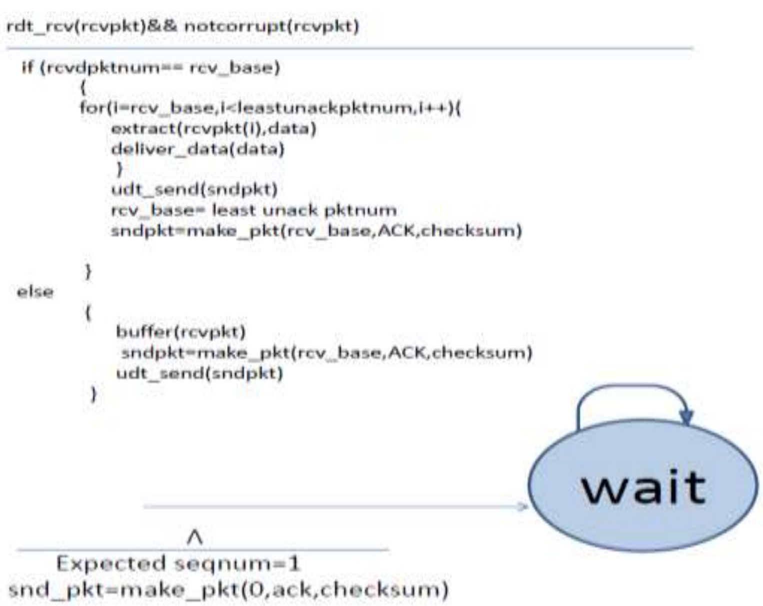
5. sender side: Notations:

Ack field value: y dupackcount: used to keep track of the number of duplicate acks

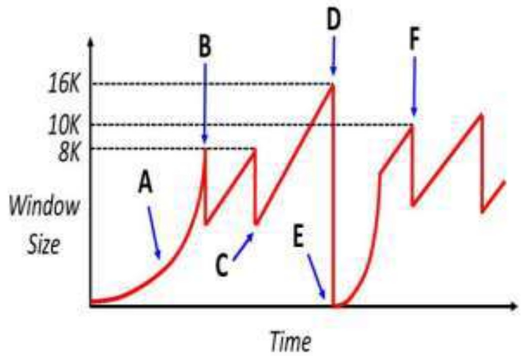


Receiver side:





6.



a.

A: Slow start

B: Triple duplicate Ack

C: arrival of a new ack

D:time out

E:slow start

F:Triple duplicate ack

b.

Because if it had a linear slope it would take longer time for TCP to reach high rates, which is a bad effect specially when the network traffic load is low and we want to use the network capacity in a more optimal way.

c.

C1. 500ms

C2.600ms

7.

a) The key difference between C1 and C2 is that C1's RTT is only half of that of C2. Thus C1 adjusts its window size after 50 msec , but C2 adjusts its window size after 100 msec. Assume that whenever a loss event happens, C1 receives it after 50msec and C2 receives it after 100msec. We further have the following simplified model of TCP. After each RTT, a connection determines if it should increase window size or not. For C1, we compute the average total sending rate in the link in the previous 50 msec. If that rate exceeds the link capacity, then we assume that C1 detects loss and reduces its window size. But for C2, we compute the average total sending rate in the link in the previous 100msec. If that rate exceeds the link capacity, then we assume that C2 detects loss and reduces its window size. Note that it is possible that the average sending rate in last 50msec is higher than the link capacity, but the average sending rate in last 100msec is smaller than or equal to the link capacity, then in this case, we assume that C1 will experience loss event but C2 will not.

The following table describes the evolution of window sizes and sending rates based on the above assumptions.

	C1		C2	
Time (msec)	Window Size	Average data sending rate (Window/0.05)	Window Size	Average data sending rate (window/0.1)
0	10	200	10	100
50	5	100	10	100
100	2	40	5	50
150	1	20	5	50
200	1	20	2	20
250	1	20	2	20
300	1	20	1	10
350	2	40	1	10
400	1	20	1	10
450	2	40	1	10
500	1	20	1	10
550	2	40	1	10
600	1	20	1	10
650	2	40	1	10
700	1	20	1	10
750	2	40	1	10
800	1	20	1	10
850	2	40	1	10
900	1	20	1	10
950	2	40	1	10
1000	1	20	1	10

b)

No. In the long run, C1's bandwidth share is roughly twice as that of C2's, because C1 has shorter RTT, only half of that of C2, so C1 can adjust its window size twice as fast as C2.

8.

a.

Since there were no acknowledgement returned before RTT_4 we could assume that estimated $RTT_4=RTT_4$

By taking this into consideration we will have:

Estimated $RTT_3 = (1-\alpha)$ $sampleRTT_4 + \alpha sampleRTT_3$

Estimated $RTT_2 = (1-\alpha)((1-\alpha)$ $sampleRTT_4 + \alpha sampleRTT_3) + \alpha sampleRTT_2$

Estimated $RTT_1 = (1-\alpha)((1-\alpha)((1-\alpha)$ $sampleRTT_4 + \alpha sampleRTT_3) + \alpha sampleRTT_2) + \alpha sampleRTT_1$

Estimated $RTT_1 = \alpha sampleRTT_1 + \alpha(1-\alpha) sample\ RTT_2 + \alpha(1-\alpha)^2\ sampleRTT_3 + (1-\alpha)^3\ sampleRTT_4$

For $\alpha=0.1$ we have:

Estimated $RTT_1 = 0.1 sampleRTT_1 + 0.09 sample\ RTT_2 + 0.081\ sampleRTT_3 + 0.729\ sampleRTT_4$

b.

By generalizing the resulted formula from the previous part we can obtain the following formula:

$RTT_1 = \alpha \sum_{i=1}^{n-1} (1-\alpha)^{i-1}$ sample $RTT_i + (1-\alpha)^n\ RTT_n$

For $\alpha=0.1$ we have:

$RTT_1 = 0.1 \sum_{i=1}^{n-1} (0.9)^{i-1}$ sample $RTT_i + (1-\alpha)^n\ RTT_n$

c.

Estimated $RTT_1 = \alpha \sum_{i=1}^{\infty} (1-\alpha)^{i-1}$ sample $RTT_i + (1-\alpha)^{\infty}$ sample RTT_n

Since $0 <= (1-\alpha) <= 1$

Estimated $RTT_1 = \alpha \sum_{i=1}^{\infty} (1-\alpha)^{i-1}$ sample RTT_i

Estimated $RTT_1 = \frac{\alpha}{1-\alpha} \sum_{i=1}^{\infty} (1-\alpha)^{i}$ sample RTT_i

According to the above formula obviously this procedure is an exponential moving average, because the effect of the past samples decays exponentially.

For $\alpha=0.1$ we would have:

Estimated $RTT_1 = \frac{1}{9} \sum_{i=1}^{\infty} (0.9)^{i}$ sample RTT_i

9.

A

Acknowledgment number=561;

Receive window=1024-500=524;

B.i)

700+560=1260➔so the smallest number that bob will not accept is 1261

B.ii)

Can send min (congestion window, flow window)

Her congestion window is 536.

700+536=1263.

So greatest byte = min (1236, 1260) = 1236

B.iii)

Can fill up network buffer.

Local buffer is 512 bytes.

Buffer currently has 200 bytes (900 not ACKed – 700 ACKed), as Alice can't delete bytes between 700 and 900 because it hasn't been ACKed (and TCP needs to keep around to possibly retransmit under a loss)

So app can write 512 – 200 = 312 bytes more

10.

A. 500ms

1 RTT for setup, then transitions 1 -> 2 -> 4 -> 8 -> 16 (4RTT)

= 5 RTT = 500ms

B.

After timeout, drops to 1 MSS, then does fast retransmit to 1/2 previous cwnd:

1-> 2 -> 4 -> 8

Then additive increase

8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14

= 9 RTT = 900ms

C.

Just drops by 1/2 cwnd+3.MSS to 10MSS

Then does additive increase :

10 -> 11 -> 12

= 2 RTT = 200 ms