Name: Muhammad Mustafa Manga

Roll #: 17K3795 D

Assignment # 3 CN

P21

Because the A-to-B channel can lose request messages, A will need to timeout and retransmit its request messages (to be able to recover from loss). Because the channel delays are variable and unknown, it is possible that A will send duplicate requests (i.e., resend a request message that has already been received by B). To be able to detect duplicate request messages, the protocol will use sequence numbers. A 1-bit sequence number will suffice for a stop-and-wait type of request/response protocol. A (the requestor) has 4 states:

• **Wait for Request 0 from above**. Here the requestor is waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R0, to B, starts a timer and makes a transition to the “Wait for D0” state. When in the “Wait for Request 0 from above” state, A ignores anything it receives from B.

• **Wait for D0**. Here the requestor is waiting for a D0 data message from B. A timer is always running in this state. If the timer expires, A sends another R0 message, restarts the timer and remains in this state. If a D0 message is received from B, A stops the time and transits to the “Wait for Request 1 from above” state. If A receives a D1 data message while in this state, it is ignored.

• **Wait for Request 1 from above**. Here the requestor is again waiting for a call from above to request a unit of data. When it receives a request from above, it sends a request message, R1, to B, starts a timer and makes a transition to the “Wait for D1” state. When in the “Wait for Request 1 from above” state, A ignores anything it receives from B.

• **Wait for D1**. Here the requestor is waiting for a D1 data message from B. A timer is always running in this state. If the timer expires, A sends another R1 message, restarts the timer and remains in this state. If a D1 message is received from B, A stops the timer and transits to the “Wait for Request 0 from above” state. If A receives a D0 data message while in this state, it is ignored. The data supplier (B) has only two states:

• **Send D0**. In this state, B continues to respond to received R0 messages by sending D0, and then remaining in this state. If B receives a R1 message, then it knows its D0 message has been received correctly. It thus discards this D0 data (since it has been received at the other side) and then transits to the “Send D1” state, where it will use D1 to send the next requested piece of data.

• **Send D1**. In this state, B continues to respond to received R1 messages by sending D1, and then remaining in this state. If B receives a R1 message, then it knows its D1 message has been received correctly and thus transits to the "Send D1" state

P22: N = Window size = 4.

K = Packet in one window which sender sends [0,3] in first window.

If 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 sender, then sender's window is [k, k+N-1].

If sender not received any Ack from receiver. In this case, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window contains [k-N,k-1]. By these arguments, the senders window is of size 4 and begins somewhere in the range [k-N,k].

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 inflight ACK values can range from k-N-1 to k-1.

P23: When protocol buffers out-of-order packets, it does not require retransmission of packets. Thus, by excluding retransmission time it significantly improves the throughput.

P24:

a) Yes, this arrangement enables reliable delivery of messages because host B is just used as an intermediate layer to relay all messages received from host A to host C.
b) Host B cannot tell that host A has received certain message or not, but host C itself can identify whether host A has received certain packet correctly with the help of sequence number.

P25:

a) Consider sending an application message over a transport protocol. With TCP, the application writes data to the connection send buffer and TCP will grab bytes without necessarily putting a single message in the TCP segment; TCP may put more or less than a single message in a segment. UDP, on the other hand, encapsulates in a segment whatever the application gives it; so that, if the application gives UDP an application message, this message will be the payload of the UDP segment. Thus, with UDP, an application has more control of what data is sent in a segment.
b) With TCP, due to flow control and congestion control, there may be significant delay from the time when an application writes data to its send buffer until when the data is given to the network layer. UDP does not have delays due to flow control and congestion control.

P26: There are $2^{32} = 4{,}294{,}967{,}296$ possible sequence numbers.

a) The sequence number does not increment by one with each segment. Rather, it increments by the number of bytes of data sent. So the size of the MSS is irrelevant -- the maximum size file that can be sent from A to B is simply the number of bytes representable by $2^{32} \approx$ 4.19Gbytes.

b) The number of segments is $\lceil 2^{32} / 536 \rceil = 8{,}012{,}999$.

   66 bytes of header get added to each segment giving a total of 528,857,934 bytes of header. The total number of bytes transmitted is $2^{32} + 528{,}857{,}934 = 4.824\text{x}10^9$ bytes. Thus it would take 249 seconds to transmit the file over a 155~Mbps link.

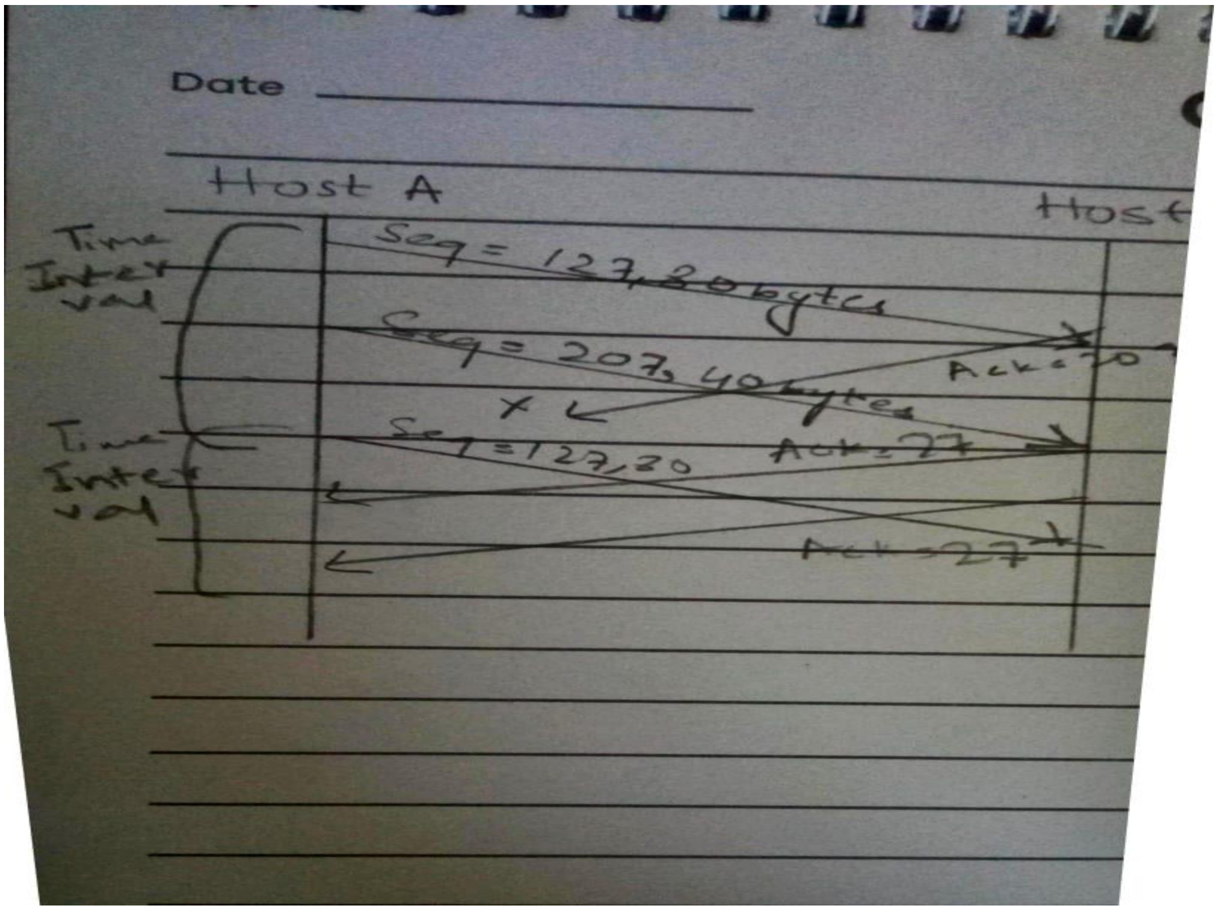
P27:

a) In the second segment from Host A to B, the sequence number is 207, source port number is 302 and destination port number is 80.

b) If the first segment arrives before the second, in the acknowledgement of the first arriving segment, the acknowledgement number is 207, the source port number is 80 and the destination port number is 302.

c) If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment, the acknowledgement number is 127, indicating that it is still waiting for bytes 127 and onwards.

d)



P28: Since the link capacity is only 100 Mbps, so host A's sending rate can be at most 100Mbps. Still, host A sends data into the receive buffer faster than Host B can remove data from the buffer. The receive buffer fills up at a rate of roughly 40Mbps. When the buffer is full, Host B signals to Host A to stop sending data by setting RcvWindow = 0. Host A then stops sending until it receives a TCP segment with RcvWindow > 0. Host A will thus repeatedly stop and start sending as a function of the RcvWindow values it receives from Host B. On average, the long-term rate at which Host A sends data to Host B as part of this connection is no more than 60Mbps.

P29:

a) The server uses special initial sequence number (that is obtained from the hash of source and destination IPs and ports) in order to defend itself against SYN FLOOD attack.
b) No, the attacker cannot create half-open or fully open connections by simply sending and ACK packet to the target. Half-open connections are not possible since a server using SYN cookies does not maintain connection variables and buffers for any connection before full connections are established. For establishing fully open connections, an attacker should know the special initial sequence number corresponding to the (spoofed) source IP address from the attacker. This sequence number requires the "secret" number that each server uses. Since the attacker does not know this secret number, she cannot guess the initial sequence number.
c) No, the sever can simply add in a time stamp in computing those initial sequence numbers and choose a time to live value for those sequence numbers, and discard expired initial sequence numbers even if the attacker replay them

P30:

a) If timeout values are fixed, then the senders may timeout prematurely. Thus, some packets are re-transmitted even they are not lost.
b) If timeout values are estimated (like what TCP does), then increasing the buffer size certainly helps to increase the throughput of that router. But there might be one potential problem. Queuing delay might be very large, similar to what is shown in Scenario 1.

P31:

DevRTT = (1- beta) * DevRTT + beta * | Sample RTT – Estimated RTT |

Estimated RTT = (1-alpha) * Estimated RTT + alpha * Sample RTT

Timeout Interval = Estimated RTT + 4 * DevRTT

After obtaining first Sample RTT 106ms:

DevRTT = 0.75*5 + 0.25 * | 106 - 100 | = 5.25ms

Estimated RTT = 0.875 * 100 + 0.125 * 106 = 100.75 ms

Timeout Interval = 100.75+4*5.25 = 121.75 ms

After obtaining 120ms:

DevRTT = 0.75*5.25 + 0.25 * | 120 – 100.75 | = 8.75 ms

Estimated RTT = 0.875 * 100.75 + 0.125 * 120 = 103.16 ms

Timeout Interval = 103.16+4*8.75 = 138.16 ms

After obtaining 140ms:

DevRTT = 0.75*8.75 + 0.25 * | 140 – 103.16 | = 15.77 ms

Estimated RTT = 0.875 * 103.16 + 0.125 * 140 = 107.76 ms

Timeout Interval = 107.76+4*15.77 = 170.84 ms

After obtaining 90ms:

DevRTT = 0.75*15.77 + 0.25 * | 90 – 107.76 | = 16.27 ms

Estimated RTT = 0.875 * 107.76 + 0.125 * 90 = 105.54 ms

Timeout Interval = 105.54+4*16.27 =170.62 ms

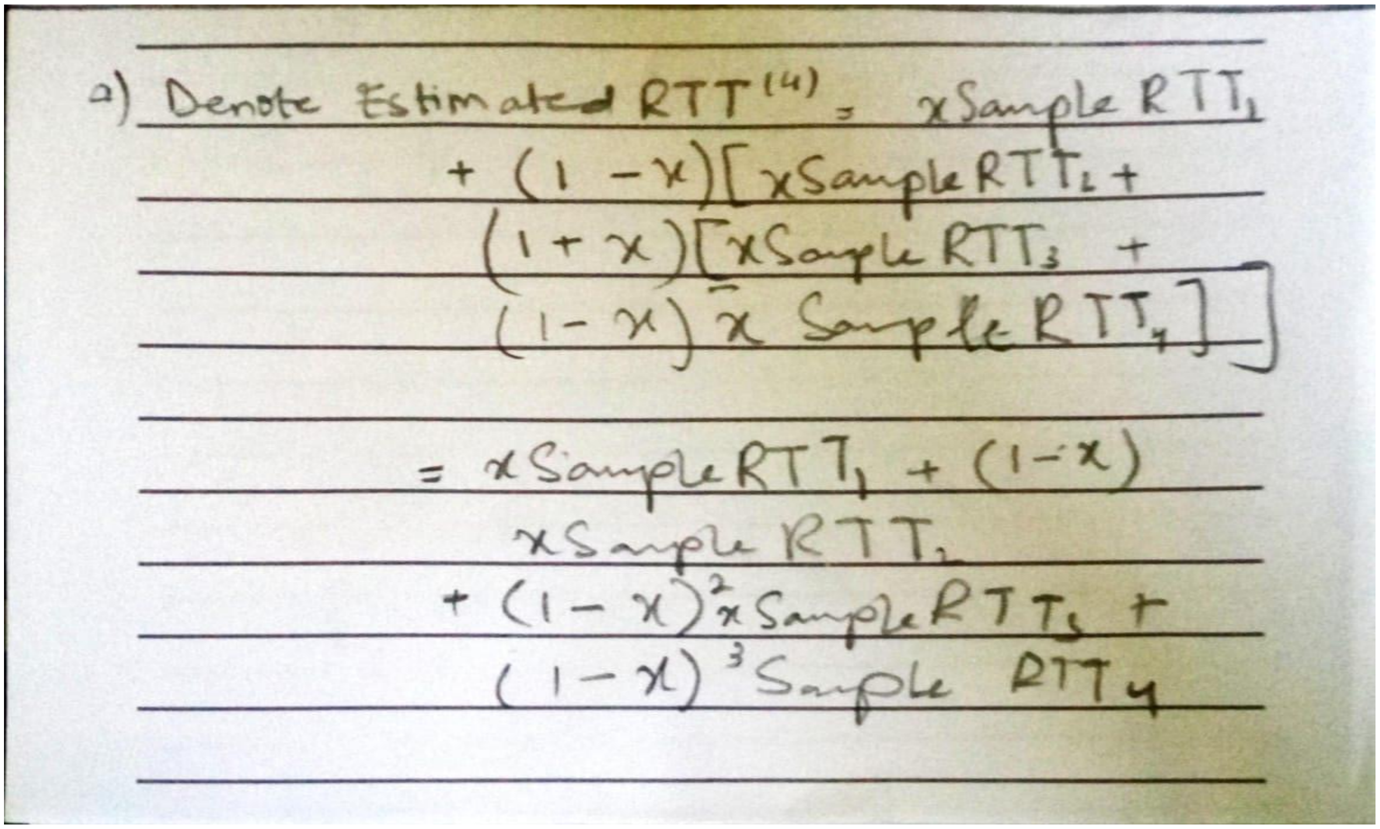
After obtaining 115ms:

DevRTT = 0.75*16.27 + 0.25 * | 115 – 105.54 | = 14.57 ms

Estimated RTT = 0.875 * 105.54 + 0.125 * 115 = 106.72 ms

Timeout Interval = 106.72+4*14.57 =165 ms

32:



a) Denote Estimated $RTT^{(4)} = x\,\text{Sample}RTT_1 + (1-x)[x\,\text{Sample}RTT_2 + (1+x)[x\,\text{Sample}RTT_3 + (1-x)\,x\,\text{Sample}RTT_4]]$

$= x\,\text{Sample}RTT_1 + (1-x)\,x\,\text{Sample}RTT_2 + (1-x)^2 x\,\text{Sample}RTT_3 + (1-x)^3\,\text{Sample}RTT_4$

(b)

Estimated RTT$^{(n)}$ = $x \sum_{j=1}^{n-1} (1-x)^{j-1}$ Sample RTT$_j$

$+ (1-x)^{n-1}$ Sample RTTn

(c) Esstimated RTT$^{(\infty)}$ = $\frac{x}{1-x} \sum_{j=1}^{\infty}$

$(1-x)^j$ Sample RTT$_j$

$= \frac{1}{9} \sum_{j=1}^{\infty} \cdot 9^j$ Sample RTT$_j$

The Weight give to past sample decay essentially.

33: Let's look at what could wrong if TCP measures SampleRTT for a retransmitted segment. Suppose the source sends packet P1, the timer for P1 expires, and the source then sends P2, a new copy of the same packet. Further suppose the source measures SampleRTT for P2 (the retransmitted packet). Finally suppose that shortly after transmitting P2 an acknowledgment for P1 arrives. The source will mistakenly take this acknowledgment as an acknowledgment for P2 and calculate an incorrect value of SampleRTT. Let's look at what could be wrong if TCP measures SampleRTT for a retransmitted segment. Suppose the source sends packet P1, the timer for P1 expires, and the source then sends P2, a new copy of the same packet. Further suppose the source measures SampleRTT for P2 (the retransmitted packet). Finally suppose that shortly after transmitting P2 an acknowledgment for P1 arrives. The source will mistakenly take this acknowledgment as an acknowledgment for P2 and calculate an incorrect value of SampleRTT .

34: At any given time t, **Send Base – 1** is the sequence number of the last byte that the sender knows has been received correctly, and in order, at the receiver. The actually last byte received (correctly and in order) at the receiver at time t may be greater if there are acknowledgements in the pipe.

Thus **Send Base – 1 ≤ Last Byte Rcvd**

35: When, at time t, the sender receives an acknowledgement with value y, the sender knows for sure that the receiver has received everything up through y-1. The actual last byte received (correctly and in order) at the receiver at time t may be greater.
If **y ≤ Send Base** or if there are other acknowledgements in the pipe. Thus **y-1 ≤ Last Byte Rcvd**.

36: Suppose packets n, n+1, and n+2 are sent, and that packet n is received and ACKed. If packets n+1 and n+2 are reordered along the end-to-end-path then the receipt of packet n+2 will generate a duplicate ack for n and would trigger a retransmission under a policy of waiting only for second duplicate ACK for retransmission. By waiting for a triple duplicate ACK, it must be the case that two packets after packet n are correctly received, while n+1 was not received. The designers of the triple duplicate ACK scheme probably felt that waiting for two packets (rather than 1) was the right tradeoff between triggering a quick retransmission when needed, but not retransmitting prematurely in the face of packet reordering.

37:

a) GoBackN: A sends 9 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later resent segments 2, 3, 4, and 5. B sends 8 ACKs. They are 4 ACKS with sequence number 1, and 4 ACKS with sequence numbers 2, 3, 4, and 5. Selective Repeat: A sends 6 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later resent segments 2. B sends 5 ACKs. They are 4 ACKS with sequence number 1, 3, 4, 5. And there is one ACK with sequence number 2. TCP: A sends 6 segments in total. They are initially sent segments 1, 2, 3, 4, 5 and later resent segments 2. B sends 5 ACKs. They are 4 ACKS with sequence number 2. There is one ACK with sequence numbers 6. Note that TCP always send an ACK with expected sequence number.
b) TCP. This is because TCP uses fast retransmit without waiting until time out.

38: Yes, the sending rate is always roughly cwnd/RTT.

39: If the arrival rate increases beyond R/2 in Figure 3.46(b), then the total arrival rate to the queue exceeds the queue's capacity, resulting in increasing loss as the arrival rate increases. When the arrival rate equals R/2, 1 out of every three packets that leaves the queue is a retransmission. With increased loss, even a larger fraction of the packets leaving the queue will be retransmissions. Given that the maximum departure rate from the queue for one of the sessions is R/2, and given that a third or more will be transmissions as the arrival rate increases, the throughput of successfully deliver data can't increase beyond λout. Following similar reasoning, if half of the packets leaving the queue are retransmissions, and the maximum rate of output packets per session is R/2, then the maximum value of λout is (R/2)/2 or R/4.

40:

a) TCP slow start is operating in the intervals [1, 6] and [23, 26]

b) TCP congestion avoidance is operating in the intervals [6, 16] and [17, 22]

c) After the 16th transmission round, packet loss is recognized by a triple duplicate ACK. If there was a timeout, the congestion window size would have dropped to 1.

d) After the 22nd transmission round, segment loss is detected due to timeout, and hence the congestion window size is set to 1.