Jason Lau
204995126

Problem 1

1. A, B
2. C, D
3. B, C
4. D
5. D
6. C (TA said large enough to achieve best efficiency)
7. B
8. D
9. A
10. D

Problem 2

Jason Lar
204995126

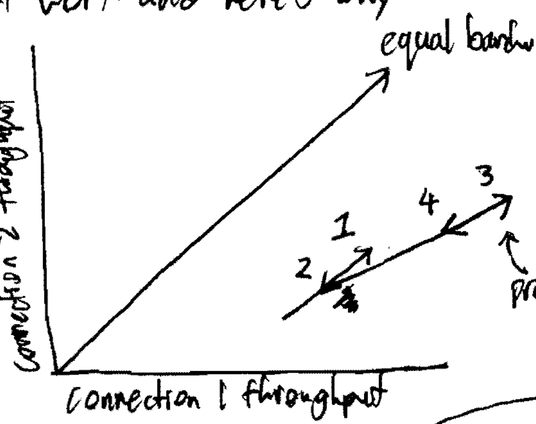
1. For the first question, no the receiver cannot be absolutely sure that bit errors have occurred with the data payload, because the checksum also includes the UDP header in its computation, so it may be that the payload is fine, but the UDP segment header ~~[illegible]~~ was corrupted.

For the second question, no the receiver cannot be absolutely sure that no bit errors have occurred. This is due to the way that the checksum ~~[illegible]~~ is calculated, where you may still compute the "correct" checksum despite having flipped bits. For this reason, the checksum's error detection isn't very high.

2. ~~No, this would entail piggybacking the FIN from the receiver to the sender on the ACK from the receiver to the sender for the first FIN from sender to receiver. However, this piggybacking may not work in any arbitrary TCP connection setting, since suppose the sender's receiver buffer~~

Yes, you can piggyback the FIN from the receiver to the sender to close the reverse TCP connection on the ACK from the receiver to the sender. This would reduce the messages to 3, so it would be 3-way handshake

3. We learned AIMD is the only way to assure fairness, so MIAD will not work and here's why



We increase proportionate to bandwidth share, but decrease by constant; this is not fair.

pretend both connections' throughput was doubled.

while we decrease both connections' share by only a constant (not proportionate)

In essence, the connection with greater bandwidth share will have its throughput aggressively increased, thus diverging from equal bandwidth share with connection 1, so MIAD does not work. AIMD works because, the connection with greater share is aggressively decreased as it's proportionate to bandwidth share (multiplicative decrease) so converges to equality.

Problem 2 Cont.

4. One scenario where timeout is triggered instead of FR/FR would be if the duplicate ~~packets~~ ACKs for triggering FR/FR are lost, so the sender does not receive the three ~~pkt~~ ACKs required to trigger FR/FR, so it times out instead (assuming no ACKs arrived prior to move the window forward). Another scenario is if the sender window is of size ≤ 3 and ~~one of the~~ the very first packet in the window is lost, ~~so~~ but the next two arrive at the sender, however, only two duplicate ACKs arrive, so we cannot trigger FR/FR.

5. The number of bits needed for the sequence number field is 5. This is because GBN requires $N+1$ sequence numbers, which in this case is $16+1=17$. Since 17 sequence numbers are needed and $2^4 = 16$ representable numbers, we need $2^5 = 32$, so 5 bits.

Problem ~~#~~ 3

Jason Lai
204995126

1. The sender reacts by retransmitting the last packet that was sent, since the sender cannot distinguish if it is a duplicate ACK or ACK for the last packet.

2. The sender would send the next segment that is usable but not sent in its window, since the arrival of the ~~segmen~~ ACK with number 900B would validate that all segments before it (including 800B) were received successfully by how cumulative ACK works.

3. The sender will resend the entire window of ~~packets~~ unacked packets to the receiver, since the timeout indicates a packet dropped, so as in Go-Back-N, the sender must send all packets in its window starting with the oldest unacked packet, as we assume receiver ~~at bu~~ has buffer size 1, so it did not buffer out-of-order packets.

4. No, it will not affect correctness, since eventually ACKs for every packet will arrive, so the sender will manage to send all of its data to the receiver. The only issue is that we ~~may~~ may effectively double the network traffic due to wasted retransmissions, which can lead to congestion, and cause packet loss, ~~which will result in~~ however reliable data transfer will still work, albeit slowly.

Problem 4

Jason Cai
204995126

1. ~~Source port~~: Source IP : 2.2.2.2
   Source Port: 80
   Dest IP: 1.1.1.1
   Dest Port: 9157

2. One socket is running at Client C, while there are two sockets running at server B.

3. a) The procedure performed over the first three messages is the three-way handshake for TCP setup.

   Fourth: seq: 79
   Ack: 45

   b) First Message: Sequence number : 41

   Second Message: Sequence number : 78
   ACK number: 42

   Fifth Message: Sequence number: 45
   Ack number: 79

   c) Source port #: 80
   Dest Port #: 9157
   Sequence number: 79
   Acknowledge number : 45
   ~~Header length: 20 bytes~~

   URG : 0
   ACK : 1
   PSH : 0
   RST : 0
   SYN : 0
   FIN : 0

Problem 5

Jason Lai
204995126

1. $$\text{EstimatedRTT} = \overset{1-\frac{7}{8}}{\left(\tfrac{1}{8}\right)} 80 + \tfrac{7}{8}(160) = 10 + 140 = 150$$

   $$\text{DevRTT} = \overset{1-\frac{3}{4}}{\left(\tfrac{1}{4}\right)} 40 + \tfrac{3}{4}\left|160 - 150\right| = 17.5$$

   $$\text{RTO} = 150 + 4(17.5) = 220\text{ ms}$$

   Segment 1: 220 ms

   $$\left.\begin{array}{l}\text{Segment 2: 220 ms}\\ \text{Segment 3: 220 ms}\end{array}\right\}$$ The same because we ignore the RTTs of retransmitted packets.

2. The RTO for the 4th Segment is still 220 ms, since we ignore the 4th segment's RTT when its ACK arrives, just like for segments 2 and 3, due to ambiguity, which is resolved by Karn's Algorithm (ignore RTTs of retransmitted segments.

Problem 6

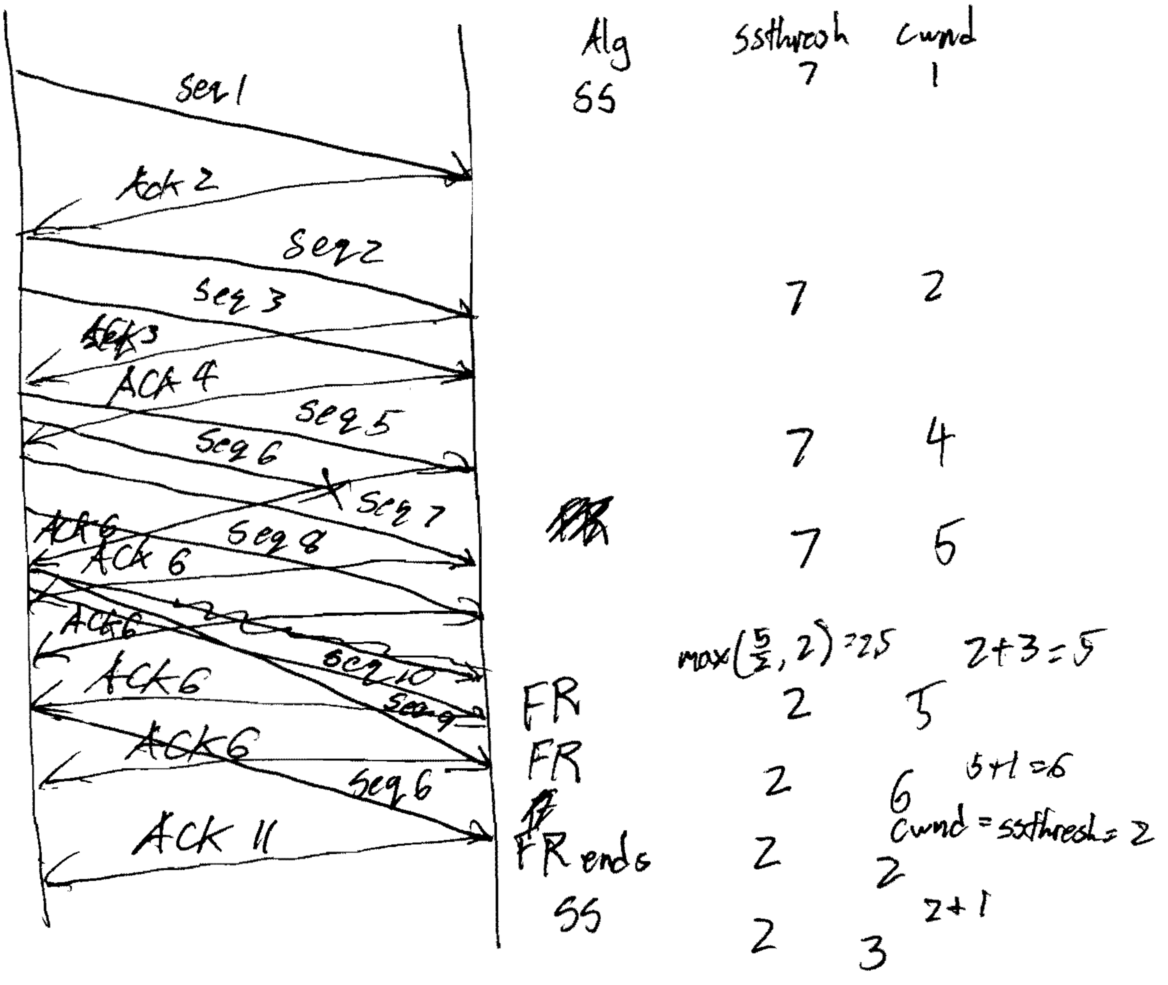
Jason Lai
204995126

1. a) TCP congestion control updates its cwnd by setting it equal to 1 MSS.

b) ~~[illegible]~~ TCP will update its ssthresh by setting it equal to 4.
This is because ssthresh = max(cwnd/2, 2 MSS) = $max(\frac{9}{2}, 2) = 4.5 \approx 4$
upon fr̶e̶a̶t̶.

c) No, because the sender window is now equal to 1, since cwnd is 1 MSS and rwnd is something larger than that. So because the sender window is now 1, ~~[illegible]~~ the sender can only have one sent, unacked packet, which is the retransmitted packet only.

2.



Problem ~~Th~~ 6 Cont.

Jason Lai
204995126

2. a) It should use slow start. Cwnd = 5 and ssthresh = 7 after the algorithm

b) Fast retransmit / Fast Recovery since the ACK will be the third duplicate for segment 6. cwnd = 5 and ssthresh = 2

c) The sender uses Fast Recovery, since the ACK will be a fourth one expecting segment 6. cwnd = 6 and ssthresh = 2

d) The ACK number is 11, since all packets up to 10 were received and buffered. Fast recovery ends since we got a new ACK, so we do slow start. Cwnd = 3 and ssthresh = 2