I have read and understood the course academic integrity policy

1) **UDP and TCP checksum**
   a) Suppose you have the following two bytes: 00001001 and 01100101. What is the 1s complement of the sum of these two bytes? **(4 points)**
   b) Suppose you have the following two bytes: 01101001 and 11101101. What is the 1s complement of the sum of these two bytes? **(4 points)**
   c) For the bytes in part (a), give an example where one bit is flipped in each of the two bytes and yet the 1s complement of the sum doesn't change. **(4 points)**

a)

00001001+01100101=01101110

Comp = 10010001

b)

01101001+11101101=101010110 -> 01010111

Comp = 10101000

c)

00001101+01100001=01101110

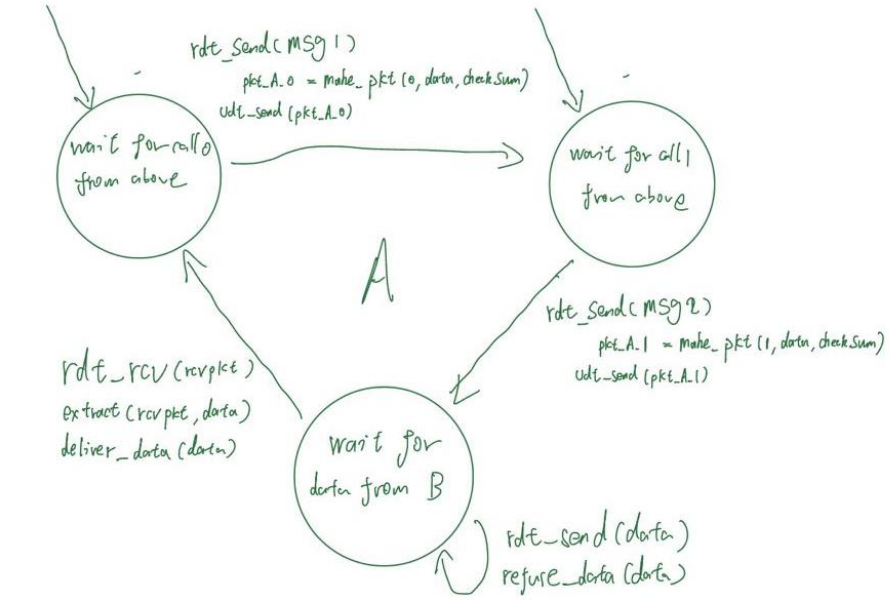
Comp = 10010001

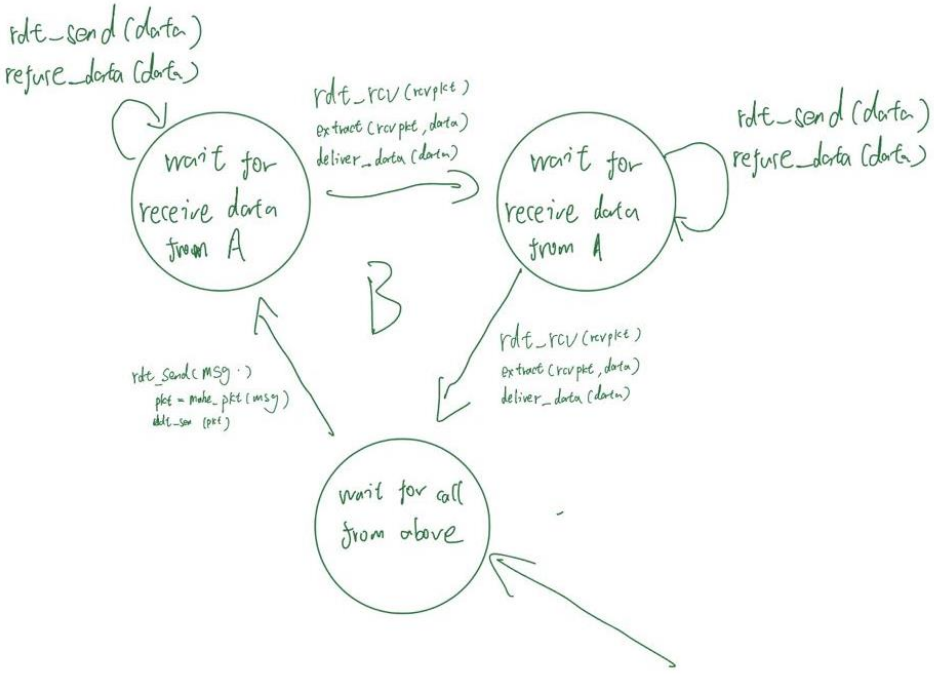
The answer is same when the last third bit flips together in each of two bytes.

2) **Simple synchronized message exchange protocol** – Consider two network entities, A and B, which are connected by a bi-directional channel that is perfect (that is, any message sent will be received correctly; the channel will not corrupt, lose, or reorder packets). A and B are to deliver data messages to each other in the following manner: A is to send *two* messages to B, then B is to send *one* message to A. Then the cycle repeats.

   Draw an FSM specification for this protocol (one FSM for A and one FSM for B). Don't worry about a reliability mechanism here; the main point is to create an FSM specification that reflects the synchronized behavior of the two entities. You should use the following events and actions, which have the sane meaning as in protocol rdt1.0, shown on page 206 of the textbook:
   `rdt_send(data), packet = make_pkt(data), udt_send(packet), rdt_rcv(packet), extract(packet, data), deliver_data(data), refuse_data(data).`

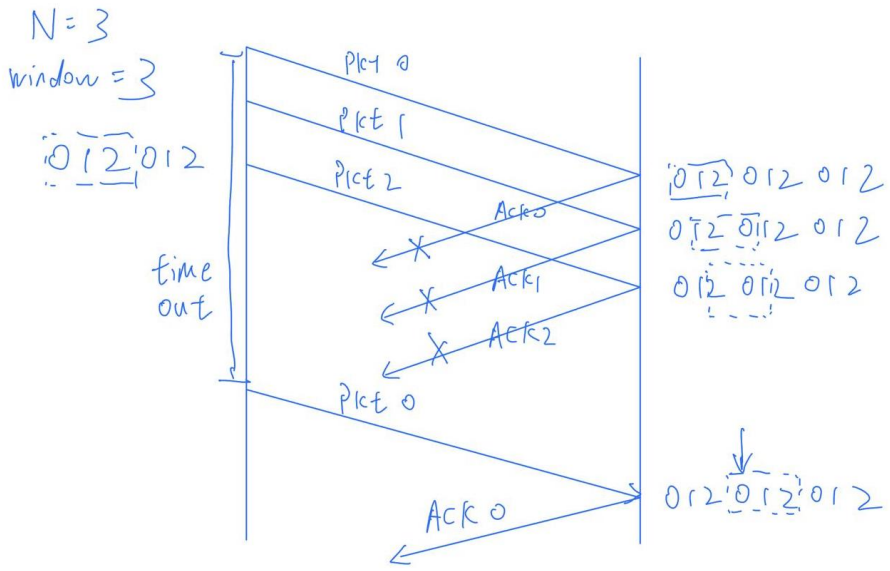
   Make sure your protocol reflects the strict alternation of sending between A and B. Also, be sure to indicate the initial states for A and B in your FSM description. (The key idea is to use states to track how many messages A has sent: one or two) **(12 points)**


rdt_send(msg1)
pkt_A_0 = make_pkt(0, data, checkSum)
udt_send(pkt_A_0)
wait for call 0 from above
wait for call 1 from above
A
rdt_send(msg2)
pkt_A_1 = make_pkt(1, data, checkSum)
udt_send(pkt_A_1)
rdt_rcv(rcvpkt)
extract(rcvpkt, data)
deliver_data(data)
wait for data from B
rdt_send(data)
refuse_data(data)


rdt_send(data)
refuse_data(data)
rdt_rcv(rcvpkt)
extract(rcvpkt, data)
deliver_data(data)
wait for receive data from A
wait for receive data from A
rdt_send(data)
refuse_data(data)
B
rdt_rcv(rcvpkt)
extract(rcvpkt, data)
deliver_data(data)
rdt_send(msg)
pkt = make_pkt(msg)
udt_send(pkt)
wait for call from above

3) **Can a window size be too large for a sequence number space?** – Consider the Go-Back-N protocol. Suppose the size of the sequence number space (number of unique sequence numbers) is N, and the window size is N. Show a scenario (give a timeline trace showing the sender, receiver, and the messages they exchange over time) where the Go-Back-N protocol will not work correctly in this case. What is the maximum window size for which the protocol will work correctly when the size of the sequence number space is N? **(16 points)**

Max size is N-1



In this case, when the window size is equal to N, then after losing 3 ACK, the resend packet 0, will be recognize as new packet by Host B, Host B can't tell the different between the New packet 0 and the resend packet 0, because the window size is equal to N.

But in this case, if the window size is N-1, when Host A resend packet 0, the sequence Host B expect to receive is 2, not zero, so it can tell packet is a resend packet.

4) **TCP sequence numbers** – Consider transferring an enormous file of L bytes from host A to host B. Assume an MSS of 536 bytes.

a) What is the maximum value of L such that TCP sequence numbers are not exhausted? **(5 points)**

b) For the value of L you obtained in (a), find how long it takes to transmit the file. Assume that a total of 66 bytes of transport, network, and link layer headers are added to each segment before the resulting packet is sent out over a 155 Mbps link. Ignore flow control and congestion control so A can pump out the segments back to back and continuously. **(5 points)**

a) Max_L = 2^32 ≈ 4.16 GB

b) NumOfPkt = L/(MSS-TotheaderLen) = (2^32)/(536) < 8012999

Time = (L+NumOfPkt*headerLen)/155Mbps

=(2^32*8b+8012999*66*8b)/(155*10^6bps)

= 248.97 s

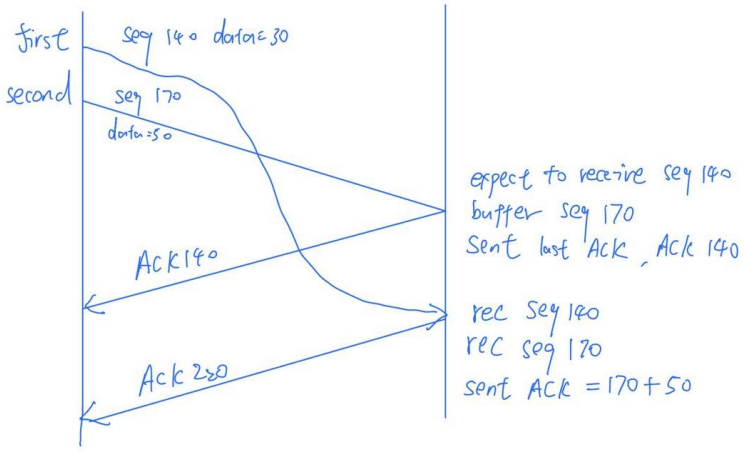
5) **TCP sequence numbers** – Host A and Host B are communicating over a TCP connection channel, and Host B has already received from host A all bytes up through byte 139. Suppose that A sends two segments to B back-to-back. The first and the second segments contain 30 and 50 bytes of data, respectively. In the first segment, the sequence number is 140, source port number is 543, and the destination port number is 80. Host B sends an ACK whenever it receives a segment from Host A.
   a) In the second segment sent from A to B, what are the sequence number, source port number, and destination port number? **(5 points)**
   b) If the first segment arrives before the second segment, in the ACK of the first arriving segment, what is the ACK number, the source port number, and the destination port number? **(5 points)**

a) Sequence number is 170
Source port is 543
Destination port is 80

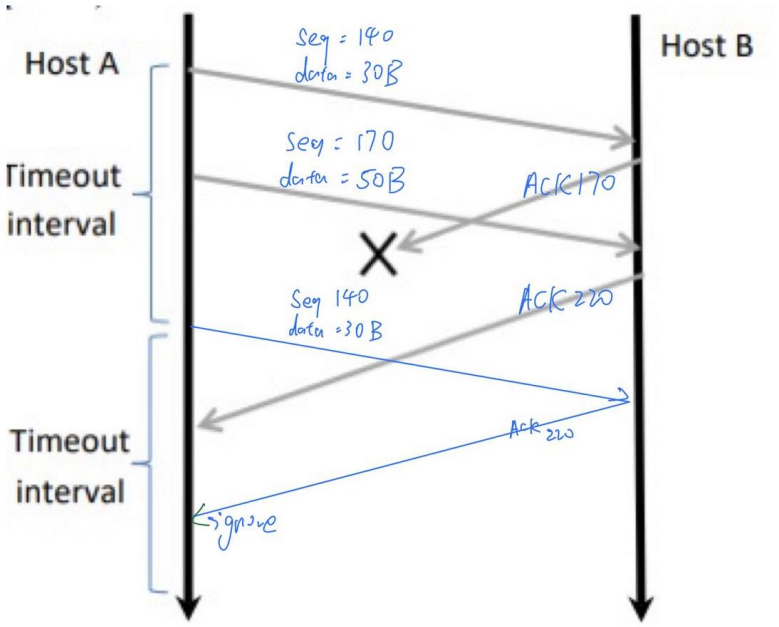
b) ACK number is 170
Source port is 80
Destination port is 543

   c) If the second segment arrives before the first segment, in the ACK of the first arriving segment, what is the ACK number? **(5 points)**
   d) Suppose the two segments sent by A arrive in order at B. The first ACK is lost and the second ACK arrives after the first timeout interval, as shown in the figure below. Complete the diagram, showing all other segments and ACKs sent. Assume there is no additional packet loss. For each segment you add to the diagram, provide the sequence number and number of bytes of data; for each ACK that you add, provide the ACK number. **(5 points)**

c) ACK number of first arriving packet is 140



d)



6) **Compare GBN, SR, TCP (no delayed ACK, no SACK)** – Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments (each containing only one byte) and their corresponding ACKs can be received (if not lost in the channel) by the receiving host (Host B) and the sending host (Host A), respectively. Suppose Host A sends 5 data segments to Host B, and the 2nd segment (sent from A) is lost – this is the only data segment or ACK loss. In the end, all 5 data segments have been correctly received by Host B. How many segments has Host A sent in total and how many ACKs has host B sent in total? What are data segment and ACK sequence numbers? Answer these two questions for all three protocols. **(30 points total, 10 points for each protocol)**

GBN: Total send 9 packets from A

Total send 8 ACK from B

Sequence number of data segment is 0,1,2,3,4,1,2,3,4

Sequence number of ACK is 0,0,0,0,1,2,3,4

SR: Total send 6 packets from A

Total send 5 ACK from B

Sequence number of data segment is 0,1,2,3,4,1

Sequence number of ACK is 0,2,3,4,1

TCP: Total send 6 packets from A

Total send 5 ACK from B

Sequence number of data segment is 0,1,2,3,4,1

Sequence number of ACK is 1,1,1,1,5