CSE 3300/5299 Homework 2 (Due on 03/03/2024 Midnight)

Course No: 5299

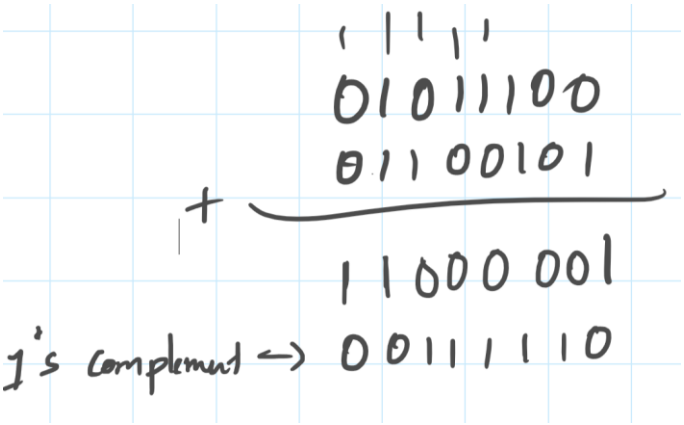
Name: Sai Kiran Belana

Question 1 (20 points)

a. Suppose you have the following 2 bytes: 01011100 and 01100101. What is the 1s complement of the sum of these 2 bytes? (5 pts)

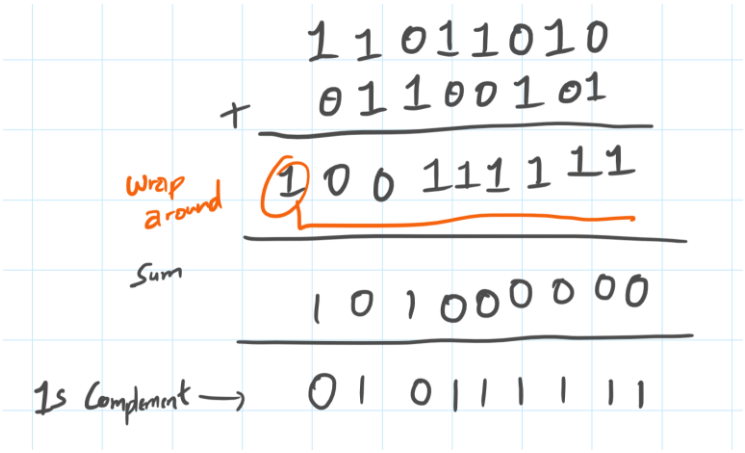
Answer: 1's complement of the sum of 01011100 and 01100101is **00111110.**

Explanation:



b. Suppose you have the following 2 bytes: 11011010 and 01100101. What is the 1s complement of the sum of these 2 bytes? (5 pts)

Answer: 1's complement of the sum of 11011010 and 01100101 is **10111111.**



c. For the bytes in part (a), give an example where one bit is flipped in each of the 2 bytes and yet the 1s complement doesn't change. (5 pts)

Answer: Given Byte1 = 010111000, Byte2 = 01100101

We can make sure that the 1s complement doesn't change by update the Least Significant bits to inverse.

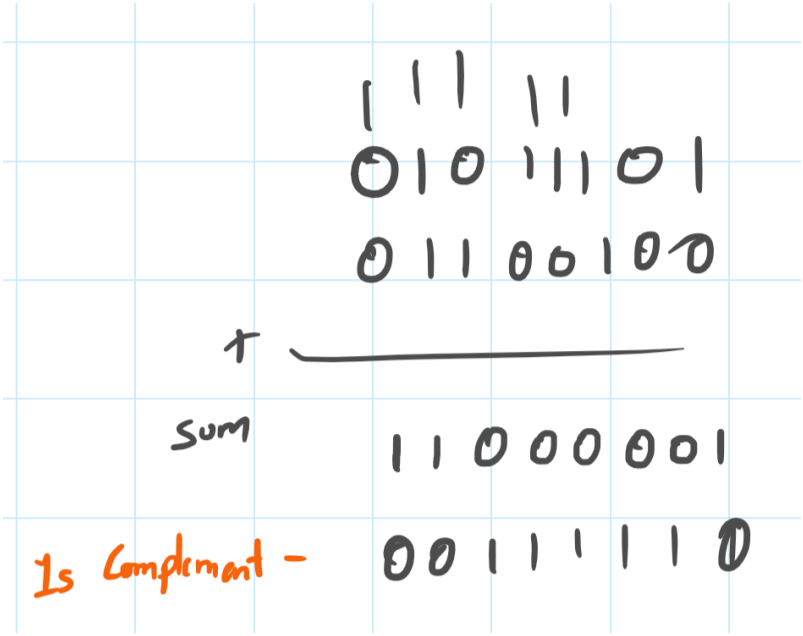
Meaning, let's update the LSB in Byte1 and Byte2 which would result as shown below.

Updates bytes:

Byte1 = 01011101

Byte2 = 01100100

Therefore, the final 1s complement doesn't change and result as **00111110.**



d. UDP and TCP use 1s complement for their checksums. Suppose you have the following bitstream: 010100110110011001110100. What is the checksum? (It is 1s complement of the sum of this bitstream. Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. (5 pts)

Answer: Considering 8-bit sums, we can solve this problem by the following steps.

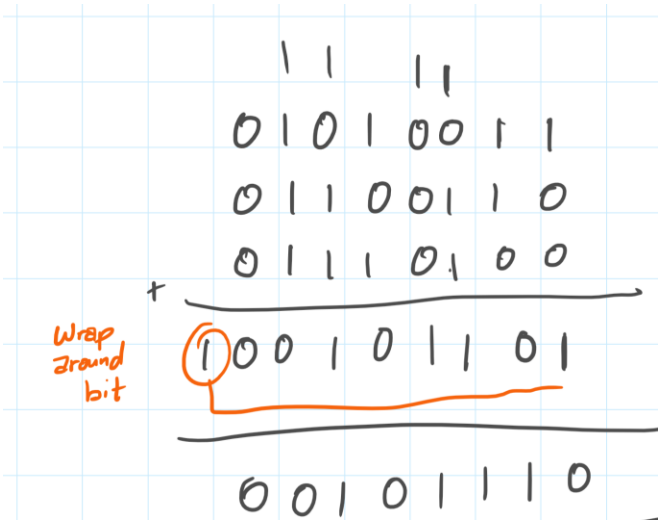
Step 1: Divide the bitstream into 8 segments.

Seg 1: 01010011

Seg 2: 01100110

Seg 3: 01110100

Step 2: Let's now sum up the segments.



Step 3: Calculate the 1s complement of the sum.

1s complement of 00101110 is **11010001.**

This method ensures error detection as part of the checksum process used in protocols like UDP and TCP, although they typically use 16-bit segments for their checksum calculations.

Question 2 (20 pts)

Two hosts X and Y are communicating over a TCP connection, all bytes up through byte 163 being sent by Host X and received by Host Y. Now assume that Host X sends three segments to Host B back-to-back. The first segment contains 40 bytes, the second segment contains 50 bytes, and the third segment contains 80 bytes.
In the first segment, the sequence number is 164, the source port number is 105, and the destination port number is 80. Host Y sends an acknowledgment whenever it receives a segment from Host X.

a. In the third segment sent from Host X to Y, what are the sequence number, source port number, and destination port number? (3 pts)

Answer: Here, the Source and destination numbers remain unchanged.
Source Port: 105, Destination Port: 80 and Sequence Number: 254.

Note: Sequence Number would be the first byte of byte stream for third segment.

Initially, sequence number starts at first segment as 164 and contains 40 bytes. So, the next byte to be sent will start at sequence number 204 (164 + 40). Seq # of 2nd segment is 204 and after adding 50 bytes, we get 254(204 + 50) which will be the Seq # for the third segment.

i.e., **254** (164 + 40 + 50)

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? (5 pts)

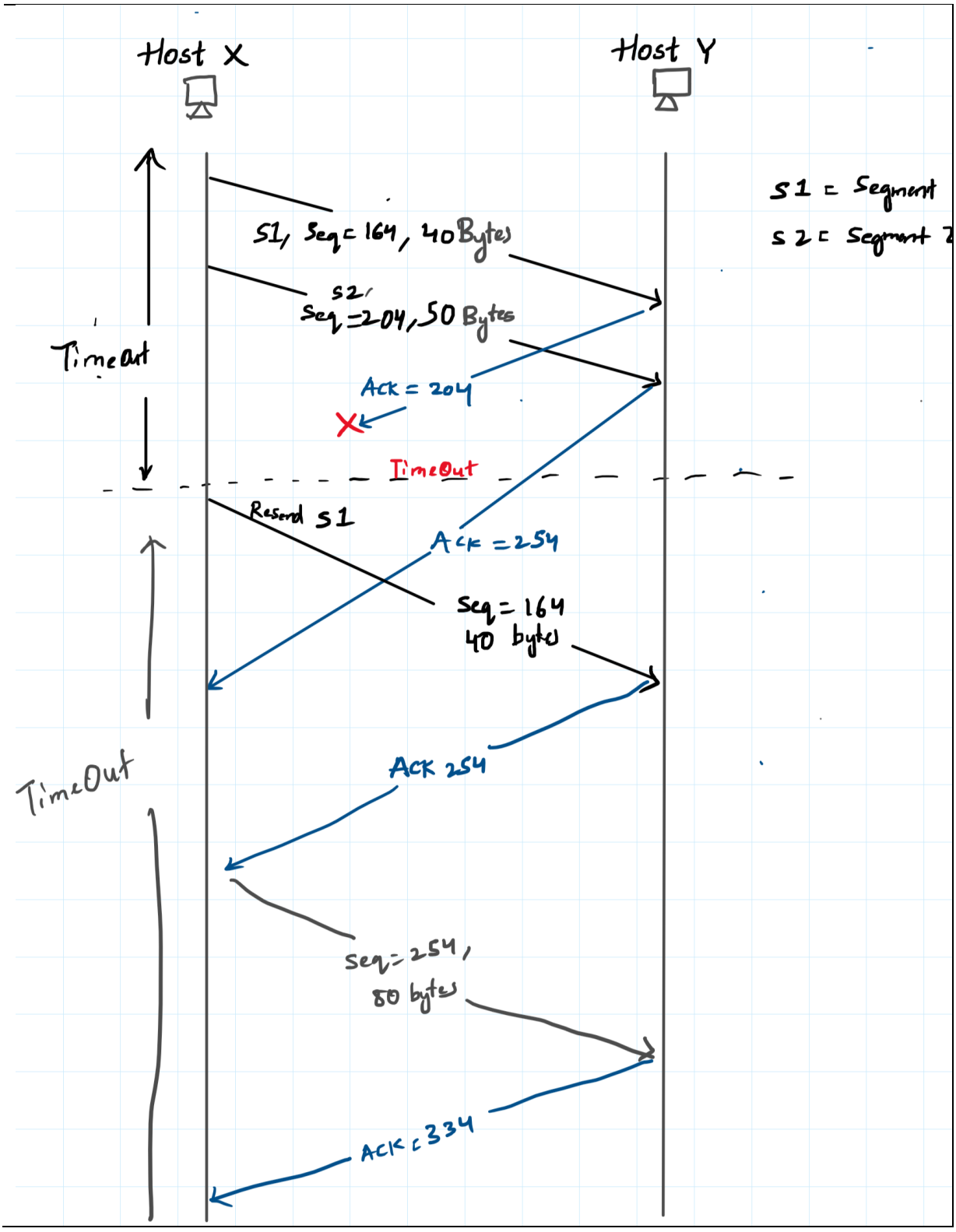
Answer: In TCP, Acknowledge Number (ACK) would be **204** (164 + 40), Source Port Number is **80** and Destination port Number is **105.**
Note: In the acknowledgement # sent from Host Y to Host X, the source and destination ports are reversed.

c. If the second segment arrives before the first segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number? (2 pts)

Answer: Acknowledge Number: **164**

d. Suppose the two segments sent by X arrive in order at Y. 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. (10 pts)

Answer:


Host X
Host Y
S1 = Segment
S2 = Segment
S1, Seq = 164, 40 Bytes
S2, Seq = 204, 50 Bytes
TimeOut
Ack = 204
TimeOut
Resend S1
Ack = 254
Seq = 164
40 bytes
TimeOut
Ack 254
Seq = 254,
80 bytes
Ack = 334

Question 3 (5 pts)

In design of the reliable data transfer protocol, what mechanism is used to handle the case that the receiver may receive a segment with errors?

Answer: In RDT 2.0, 2 mechanisms are implemented such as error detection and feedback control messages such as ACK and NAK from receiver to sender.

1. **Error Detection:**
   Both the sender and receiver use error detection codes, such as checksums to determine whether a segment has been changed during the transmission. If the calculated checksum at receiver mismatches with the send checksum by the sender, an error is detected.
2. **Acknowledgments (ACKs):**
   Receiver sends an ACK for correctly received segments, which indicates successful reception and the seq number of the next expected byte.
3. **Negative Acknowledgements (NACKs):**
   In case of errors, a NACK will be sent to request a retransmission, which prompts the sender to resend the data.

Checksums & ACK/NACK

Question 4 (5 pts)

In protocol rdt3.0 (unreliable channel can cause bit errors and packet loss), the ACK packets flowing from the receiver to the sender do not have sequence numbers (although they do have an ACK field that contains the sequence number of the packet they are acknowledging). Why is it that our ACK packets do not require sequence numbers?

Answer: In rdt3.0, ACK packets do not require their own sequence numbers because the protocol operates on a stop-and-wait basis, meaning only one packet is sent and acknowledged at a time. Here, only one packet is transmitted at a time, making the flow sequential. The ACK or NACK references the sequence number of the packet it acknowledges, making it clear which packet the sender needs to act upon net. This design simplifies the protocol and reduces the overhead, as each ACK directly corresponds to a single outstanding packet, eliminating the need for separate sequencing of acknowledgements.

Note: rdt3.0 uses a countdown timer for the packet sent.

Question 5 (10 pts)

Suppose that the five measured *SampleRTT* values 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 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 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. (beta = 0.25, alpha = 0.125)

Answer:

Given, SampleRTT values = 106, 120, 140, 90 and 115. alpha (α) =0.125, beta (β) = 0.25

a. Computing EstimatedRTT after each Sample RTT value is obtained.

Given, EstimatedRTT before first five samples obtained is 100ms. Let us consider

EstRTT0 = 100ms.

We know the formula for TCP round trip time and timeout:

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

ERTT 1 = (1-0.125) x ERTT0 + 0.125 x 106 = 87.5 + 13.2 = 100.75ms

ERTT2 = 0.875 x ERTT1 + 0.125 x 120 = 103.15625ms

ERTT3 = 0.875 X ERTT2 + 0.125 X 140 = 107.7617185ms

ERTT4 = 0.875 x ERTT3 + 0.125 x 90 = 105.54150390625ms

ERTT5 = 0.875 X ERTT4 + 0.125 X 115 = 106.7238159ms.

Therefore, after 5 SampleRTT values, the EstimatedRTT is 106.7238159ms

b. Computing DevRTT after each Sample RTT value is obtained.

Given, DevRTT before first five samples obtained is 5ms. Let us consider DevRTT0 = 5ms.

Note

We know the formula for DevRTT:

DevRTT = (1-β)*DevRTT +
β*|SampleRTT-EstimatedRTT|

DevRTT1 = (1 - 0.25) X DevRTT0 + 0.25 x |106 – ERTT1|

= (0.75) x 5 + 0.25 x 6

= 5.0625 ms

DevRTT2 = 0.75 x DevRTT1 + 0.25 X |120 -ERTT2| = 8.0078125 ms

DevRTT3 = 0.75 x DevRTT2 + 0.25 x |140 – ERTT3| = 14.0654296875 ms

DevRTT4 = 0.75 x DevRTT3 + 0.25 x |90 – ERTT4| = 14.4344482421875 ms

DevRTT5 = 0.75 x DevRTT4 + 0.25 x |115 – ERTT5| = 12.894882202148438 ms

Therefore, after 5 SampleRTT values, the DevRTT is 12.894882202148438 ms

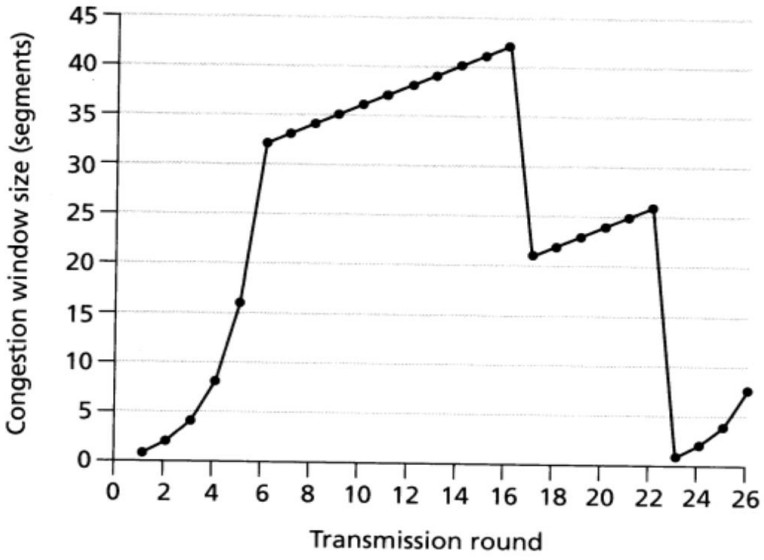
Question 6 (5 pts)

We saw that TCP waits until it has received three duplicate ACKs before performing a fast retransmit. Why do you think the TCP designers chose not to perform a fast retransmit after the first duplicate ACK for a segment is received?

Answer: The TCP designers chose not to perform a fast retransmit after the first duplicate ACL for a segment is received due to the following reasons for balancing efficiency and network stability:

1. Out-of-Order Packets: A single duplicate ACK may indicate out-of-order packets, not necessarily loss.
2. Network variability: Waiting for 3 duplicate ACKs helps in confirming it is packet loss rather than delay.
3. Congestion Avoidance:
   Avoids premature retransmission that could worsen network congestion.
4. Efficiency and Performance:
   Balances quick responses to loss with avoidance of unnecessary retransmissions

Question 7 (35 pts) Consider the figure below.



Assuming TCP Reno is the protocol experiencing the behavior shown, 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. (5 pts)
Answer: Here, TCP Reno Slow start intervals occur at the following times:
- [1,32] Congestion Window or [1,6] time intervals
- [1,7] CWND or [23,26] time intervals.

b. Identify the intervals of time when TCP congestion avoidance is operating. (5 pts)
Answer: Congestion avoidance is happening at intervals [6,16] and [17,22].
The `cwnd` grows linearly by one segment for each round, which indicates the congestion avoidance mechanism. Once the window recovers from the drop(packet-loss) and starts increasingly but at a slower, linear rate, congestion avoidance is at play. Each time the window recovers and grows without the steep, exponential increase seen during slow start, it indicates congestion avoidance.

c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout? (5 pts)
Answer: In TCP Reno, a triple duplicate is detected when there is a sudden reduction in the congestion window size but is not as drastic as it would be in the case of timeout. When a timeout occurs, the congestion windows size is typically reset to 1MSS.
In the given graph, the congestion window size reduced to a value that is likely around half of the previous cwnd.

d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout? (5 pts)
Answer: In TCP Reno, after 22nd transmission round, a timeout is detected since the congestion window size has a sharp decrease(reset) to 1 MSS.

e. What is the initial value of ssthresh at the first transmission round? (5 pts)
Answer: The initial value of ssthresh at the first transmission round is at `cwnd` **32** which afterwards grows linearly for congestion avoidance.

f. What is the value of ssthresh at the 20th transmission round? (5 pts)
Answer: The value of ssthresh at the 20th transmission round is at `cwnd` **21** since there is a triple duplicate detection took place. At 16th round, when the cwnd is at 42, a triple duplicate error os detected which leads to reducing the cwnd to half i.e., 42/2 = 21.

g. What is the value of ssthresh at the 24th transmission round? (5 pts)
Answer: At 24th transmission round, the ssthresh value is set to `cwnd` **1** since there is a timeout took place from 22 to 23 Transmission round and now we are entering to the new MSS

↳ (13) 26/2 = 13