National University of Computer and Emerging Sciences

Assignment 03

Chapter 03

CS-3001 Computer Networks – Spring 2026

Section: BSE-6B1 **Max Marks: 50**

Instructions:

- **Show all work.** Full credit is only given for deriving the correct final expressions and values.
- Assume 1 kbps = 10^3 bps, 1 Mbps = 10^6 bps, unless otherwise stated.
- **Answer all questions.** Partial credit may be awarded if your reasoning is clear, even if the final answer is incorrect.
- **Use neat and organized work.** Clearly label all steps, diagrams, and equations.
- **Units matter.** Always include appropriate units in your answers.
- **Academic Integrity:** Students are expected to submit their own original work. Plagiarism, copying from others, or sharing solutions is strictly prohibited and may result in zero marks and disciplinary action.
- **Submission Guidelines:** Submit your assignment by the deadline in the format specified by the instructor. Late submissions may be penalized unless prior permission is granted.

Question	**Max Marks**	**Obtained**
1	20	
2	10	
3	10	
4	10	
Total	**50**	

Question 1: Textbook Problems (20 Marks)

Part A: Review Questions (5 Marks)

R3. Consider a TCP connection between Host A and Host B. Suppose that the TCP segments traveling from Host A to Host B have source port number x and destination port number y. What are the source and destination port numbers for the segments traveling from Host B to Host A?

Solution:

For the segments traveling from Host B to Host A, the port numbers are swapped.

- **Source Port:** y
- **Destination Port:** x

Part B: Problems (5 + 5 + 5 = 15 Marks)

P3. UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (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. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum? With the 1s complement scheme, how does the receiver detect errors? Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

Solution:

Sum Calculation:

```
              01010011
            + 01100110
  --------------------
              10111001

              10111001
            + 01110100
  --------------------
1 00101101 (Carry occurred)

       Wrap the carry:
              00101101
                   + 1
  --------------------
              00101110
```

1s Complement (Checksum): Invert bits of `00101110` → `11010001`.

Why 1s Complement? Using the 1s complement allows the receiver to simply add the data and the checksum together. If there are no errors, the result should be all 1s (negative zero). If we just used the sum, the receiver would have to recompute the sum and compare it for equality. The 1s complement method is also endian-independent.

Error Detection: The receiver adds the three bytes and the checksum together. If the result is 11111111, no error is detected. If there is any 0, an error occurred.

1-bit Error: It will always be detected because it will flip a bit in the sum, preventing the result from being all 1s.

2-bit Error: It is possible for a 2-bit error to go undetected if one bit flips from 0 to 1 and another flips from 1 to 0 in the same column (bit position) but different words, compensating for each other in the sum.

P4. (a) Suppose you have the following 2 bytes: 01011100 and 01100101. What is the 1s complement of the sum of these 2 bytes?

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

(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.

Solution:

(a)

```
   01011100
 + 01100101
 ----------
   11000001
```

1s Complement: `00111110`

(b)

```
        11011010
      + 01100101
 ---------------
 1 00111111 (Carry)
             + 1
 ---------------
        01000000
```

1s Complement: `10111111`

(c) Example: Byte 1: Flip bit 0 (LSB) from 0 to 1 (add 1). Byte 2: Flip bit 0 (LSB) from 1 to 0 (subtract 1). New Byte 1: `01011101` New Byte 2: `01100100` Sum: `01011101 + 01100100 = 11000001`. The sum is identical, so the checksum remains `00111110`.

P40. Consider the given Figure. Assuming TCP Reno is the protocol experiencing the behavior shown above, 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.

(b) Identify the intervals of time when TCP congestion avoidance is operating.

(c) After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

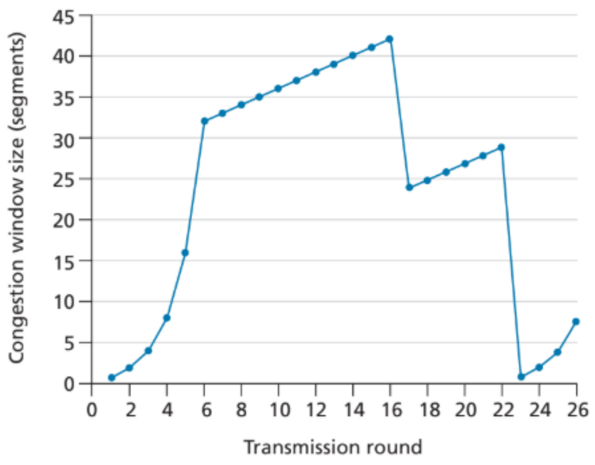
(d) After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

(e) What is the initial value of ssthresh at the first transmission round?

(f) What is the value of ssthresh at the 18th transmission round?

(g) What is the value of ssthresh at the 24th transmission round?

(h) During what transmission round is the 70th segment sent?



(i) Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?

(j) Suppose TCP Tahoe is used (instead of TCP Reno), and assume that triple duplicate ACKs are received at the 16th round. What are the ssthresh and the congestion window size at the 19th round?

(k) Again suppose TCP Tahoe is used, and there is a timeout event at 22nd round. How many packets have been sent out from 17th round till 22nd round, inclusive?

Solution:

(a) **Slow Start:** Rounds [1, 6] and [23, 26]. (Recognized by exponential growth).

(b) **Congestion Avoidance:** Rounds [6, 16] and [17, 22]. (Recognized by linear growth).

(c) **Triple Duplicate ACK:** The congestion window drops to half (from 42 to 21), indicating TCP Reno's Fast Recovery.

(d) **Timeout:** The congestion window drops to 1, indicating a timeout.

(e) **Initial ssthresh: 32.** This is the point where the graph switches from exponential to linear growth in the first cycle.

(f) **ssthresh at 18th: 21.** Loss occurred at round 16 when cwnd was 42. ssthresh is set to $42/2 = 21$.

(g) The threshold is set to half the value of the congestion window when packet loss is detected. When loss is detected during transmission round 22, the congestion windows size is 29. Hence the threshold is 14 (taking lower floor of 14.5) during the 24th transmission round.

(h) **Round 7.** Calculations: R1: 1 pkt (Total 1). R2: 2 (Total 3). R3: 4 (Total 7). R4: 8 (Total 15). R5: 16 (Total 31). R6: 32 (Total 63). R7: 33 packets (Total 96). The 70th packet is sent in Round 7.

(i) **ssthresh = 4, cwnd = 7.** At round 26, cwnd is 8. ssthresh becomes $8/2 = 4$. In Reno, cwnd becomes ssthresh + 3 (for the 3 dup ACKs) = 7.

(j) Threshold ssthresh is 21, and congestion window size is 1.

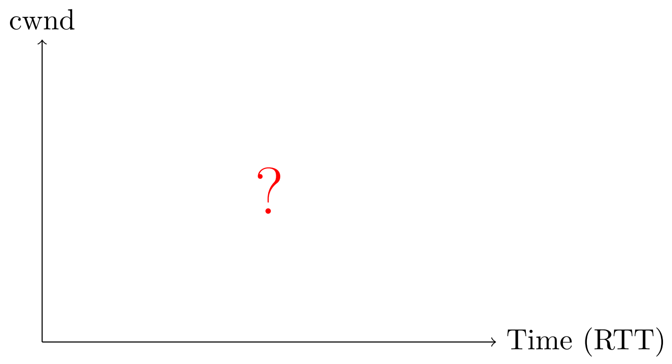
(k) **52 Packets.** Using Tahoe settings (ssthresh=21 after R16): R17: cwnd 1. R18: cwnd 2. R19: cwnd 4. R20: cwnd 8. R21: cwnd 16. R22: cwnd 21 (linear growth starts as it hits ssthresh). Total $= 1 + 2 + 4 + 8 + 16 + 21 = 52$.

Question 2: TCP Tahoe vs Reno (10 Marks)

Scenario: Consider the evolution of *cwnd* over time.

- $RTT = 1$ second. $MSS = 1$ unit.
- At $t = 0$, $cwnd = 1$, ssthresh=32. Slow Start begins.
- At $t = 4$, a Timeout occurs.
- The system enters Slow Start again.
- At the instance *cwnd* reaches 10 again, 3 Duplicate ACKs occur.

1. Calculate the value of `ssthresh` immediately after the 3-Dup-ACK event.
2. What is the value of *cwnd* 1 RTT after the 3-Dup-ACK event in Reno?



Solution:

1. **ssthresh = 5.** When 3 duplicate ACKs occur at $cwnd = 10$, the new threshold is set to half the current window size: $\lfloor 10/2 \rfloor = 5$.
2. **cwnd = 5.** Assuming TCP Reno: Upon 3 Dup ACKs, the system enters Fast Recovery. *cwnd* is set to $ssthresh + 3 = 8$. However, the question asks for the value **1 RTT after** the event. Assuming the retransmitted packet (Fast Retransmit) is acknowledged successfully within that RTT, the system exits Fast Recovery. Upon exiting Fast Recovery, *cwnd* is deflated back to *ssthresh*. Therefore, $cwnd = 5$.

Question 3: Selective Repeat Protocol (10 Marks)

Scenario: Consider a Selective Repeat (SR) protocol with the following parameters:

- **Window Size** $N = 4$.
- Sequence Numbers are integers (0, 1, 2, 3, 4, 5...). Assume no wrap-around for this problem.
- Sender and Receiver start with base = 0.

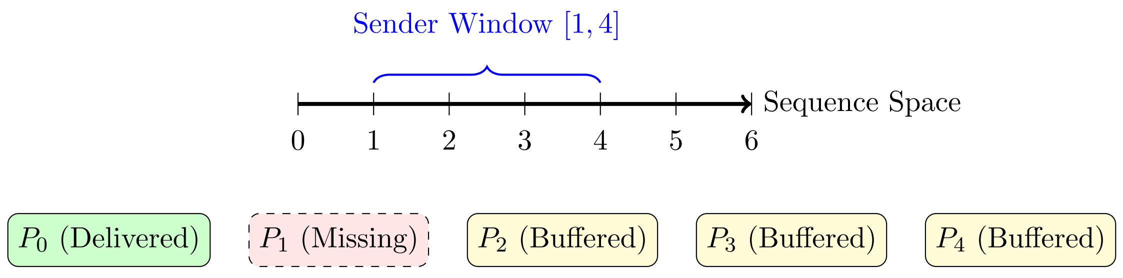
The following sequence of transmission and acknowledgement events occurs:

1. Sender sends packets P_0, P_1, P_2, P_3.

2. Receiver gets P_0, delivers it, sends ACK0.
3. P_1 is lost in the network.
4. Receiver gets P_2, buffers it, sends ACK2.
5. Receiver gets P_3, buffers it, sends ACK3.
6. ACK0 arrives at Sender. Sender advances window and sends P_4.
7. Receiver gets P_4, buffers it, sends ACK4.
8. ACK2 arrives at Sender.
9. Sender timeout for P_1 expires. Sender retransmits P_1.

Based on this specific timeline, answer the following with **Yes/No** and a brief logical explanation involving the window boundaries or buffer state.

1. Immediately after **Step 8** (ACK2 arrives at Sender), can the Sender transmit packet P_5? (Yes/No)
2. Immediately before the retransmitted P_1 arrives at the Receiver, has the packet P_4 been delivered to the Application Layer? (Yes/No)
3. Upon receipt of the retransmitted P_1, does the Receiver's window base jump to 5? (Yes/No)



Receiver State

Solution:

1. **No.** The Sender's window is defined by $[send\_base, send\_base + N - 1]$. Since ACK0 was received, the base moved to 1. The window is $[1, 2, 3, 4]$. Even though ACK2 is received, the base cannot move past 1 because ACK1 has not been received. Packet P_5 falls outside the window.
2. **No.** Selective Repeat receivers buffer out-of-order packets but can only deliver them to the application layer in sequential order. P_4 is buffered, but it is stuck behind the missing P_1.
3. **Yes.** The Receiver buffer currently holds P_2, P_3, P_4. When P_1 arrives, the sequence $1, 2, 3, 4$ becomes complete. The Receiver delivers all of them and slides the window base to $4 + 1 = 5$.

Question 4: Exponential Weighted Moving Average & TCP (10 Marks)

Scenario: TCP uses an Exponential Weighted Moving Average (EWMA) to calculate the `TimeoutInterval`. The updating formulas are:

$$EstimatedRTT_{new} = (1-\alpha)\cdot EstimatedRTT_{old} + \alpha \cdot SampleRTT$$
$$DevRTT_{new} = (1-\beta)\cdot DevRTT_{old} + \beta \cdot |SampleRTT - EstimatedRTT_{old}|$$
$$TimeoutInterval = EstimatedRTT_{new} + 4 \cdot DevRTT_{new}$$

Use standard values: $\alpha = 0.125$ and $\beta = 0.25$.

Suppose the current state of a TCP connection is:

$$EstimatedRTT = 200 \text{ ms}, \quad DevRTT = 10 \text{ ms}$$

The following events occur:

- **Segment 1:** Sent at $t = 0$, ACK received at $t = 280$ ms (Valid Sample).
- **Segment 2:** Sent at $t = 300$, ACK received at $t = 600$ ms (Valid Sample).
- **Segment 3:** Sent at $t = 700$. The timer expires. Segment 3 is retransmitted at $t = Timeout$. The ACK for the retransmission arrives at $t = Timeout + 400$ ms.

1. Calculate the $TimeoutInterval$ immediately after the receipt of the ACK for Segment 1.
2. Calculate the $TimeoutInterval$ immediately after the receipt of the ACK for Segment 2.
3. Determine the values of $EstimatedRTT$ and $DevRTT$ after the ACK for Segment 3 is processed. Explain your reasoning based on TCP rules for retransmissions.

Solution:

1. **After Segment 1 (SampleRTT = 280):**
   - $EstRTT = 0.875(200) + 0.125(280) = 175 + 35 = \mathbf{210\ ms}$.
   - $DevRTT = 0.75(10) + 0.25|280 - 200| = 7.5 + 20 = \mathbf{27.5\ ms}$.
   - $Timeout = 210 + 4(27.5) = 210 + 110 = \mathbf{320\ ms}$.
2. **After Segment 2 (SampleRTT = 300):**
   - $EstRTT = 0.875(210) + 0.125(300) = 183.75 + 37.5 = \mathbf{221.25\ ms}$.
   - $DevRTT = 0.75(27.5) + 0.25|300 - 210| = 20.625 + 22.5 = \mathbf{43.125\ ms}$.
   - $Timeout = 221.25 + 4(43.125) = 221.25 + 172.5 = \mathbf{393.75\ ms}$.
3. **Values Unchanged.** $EstRTT = 221.25$ ms, $DevRTT = 43.125$ ms. **Reasoning:** According to **Karn's Algorithm**, TCP does not update EstimatedRTT or DevRTT using samples from retransmitted segments because the sender cannot know if the ACK is for the original transmission or the retransmission.