Mustafa Alawadi

40217764

Theory Assignment 3

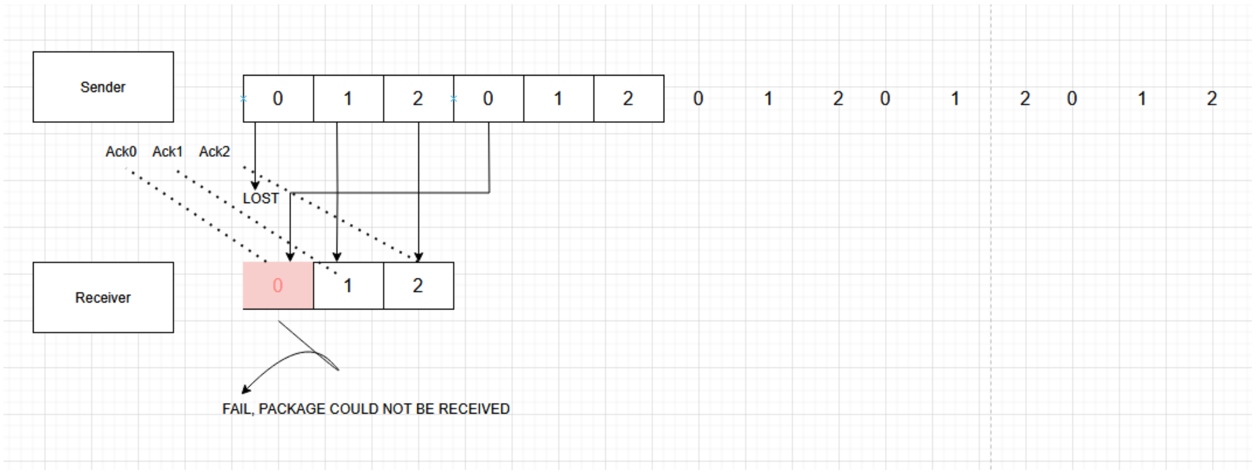
Dr Aiman Hanna

2025-03-16

1. **Showing that window size is bigger than 2^k-1 would fail**

   If we assume that K = 2 then Sender Window Size is = 6 > 2^2 - 1

   Receiver Window Size is = 3



In unrestricted control, When the window size is set to exactly 2^k-1, the sender can keep the pipeline full, closely matching the continuous transmission speed of the unrestricted protocol. This setup allows for nearly similar performance, but it may still be slower. This is because, in Go-Back-N, the receiver must send acknowledgments for each packet it receives, and any delays in these acknowledgments can slow down the sender. On the other hand, the unrestricted protocol doesn't require acknowledgments and allows the sender to keep sending packets without waiting, regardless of the receiver’s state.

2. In theory, UDP datagrams can delay TCP packets indefinitely because both use the same network layer and IP. Since UDP lacks congestion control, it can send an uninterrupted burst of data, consuming all available bandwidth and leaving little room for TCP packets. TCP, relying on congestion control, may experience significant delays or timeouts when the network is saturated by UDP traffic. To mitigate this while preserving UDP’s advantages, a simple solution would be to implement UDP-based congestion control. This would allow UDP to adjust its transmission rate during congestion, ensuring it doesn't entirely block TCP traffic.

3. **i. Error Detection Guarantee:**

   Yes, when an error is detected, it guarantees that an error has occurred. The challenge, however, is that while we are sure an error took place, the checksum does not tell us which chunk of data the error occurred in.

   ii. Using 1's Complement to Detect Errors:

   Yes, it's possible to use the 1's complement for error detection. Here's how it works with two data chunks, A and B:

   Adding the chunks:

   A + B = C (Here, two chunks are combined, and the sum is C).

Calculate the checksum:

Checksum = Not(C) (Take the 1's complement of the sum C).

Transmit the data:

C + Not(C) = All bits will be 1.

On the receiving side, the process works as follows:

Adding the 1's complements:

Not(A) + Not(B) = Not(C).

Calculate the checksum:

Checksum = Not(Not(C)) = C (Take the 1's complement again).

Verification:

C + Not(C) = All bits will be 1, confirming that no errors have occurred.

iii. 1-bit and 2-bit Error Detection:

1-bit error: A 1-bit error cannot go undetected. It will cause the checksum to fail, as it would produce a result that does not match the expected all-1s outcome.

2-bit error: A 2-bit error can sometimes go undetected, especially if the errors happen in a way that the checksum calculation cancels them out. For example:

Consider the following two chunks of data:

Chunk 1: 001111111

Chunk 2: 100000000

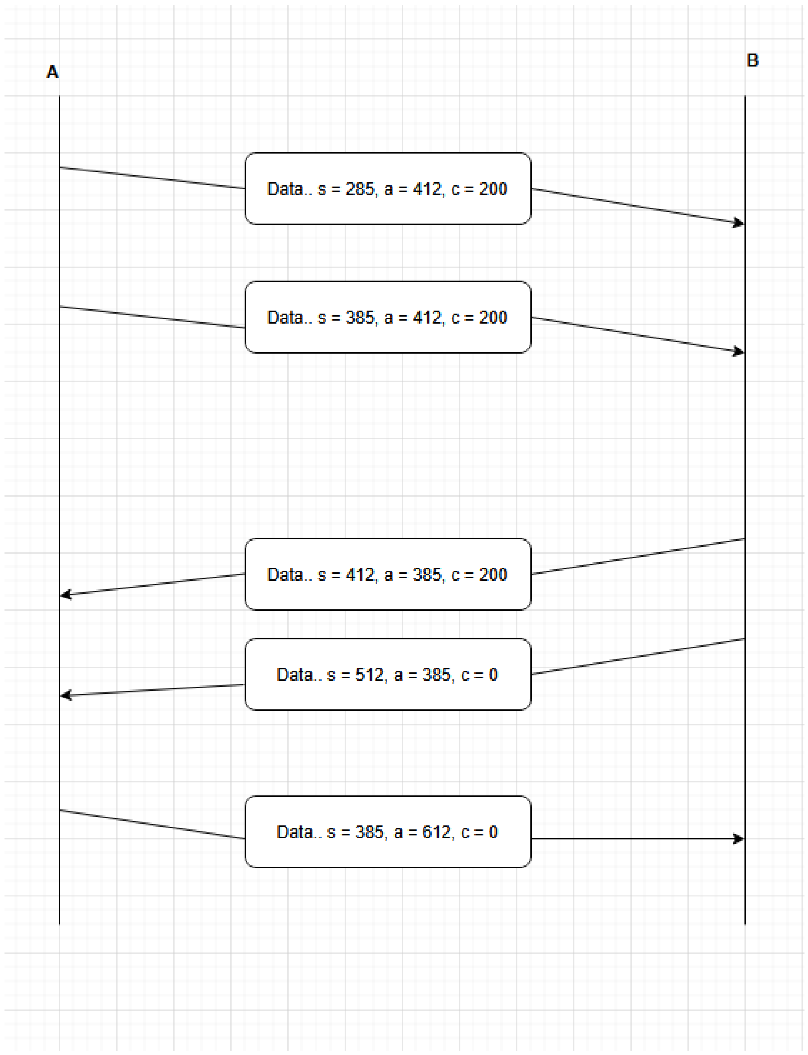
After adding them together:

001111111 + 100000000 = 101111111

Now, if the second bit in both chunks flips (two errors), the resulting data becomes:

001111101 + 100000010 = 101111111

4.


A
B
Data.. s = 285, a = 412, c = 200
Data.. s = 385, a = 412, c = 200
Data.. s = 412, a = 385, c = 200
Data.. s = 512, a = 385, c = 0
Data.. s = 385, a = 612, c = 0

5. **Sequence Number Calculation:**

Segment 1 starts with sequence number 524, and it carries 260 bytes. Therefore, this segment spans from byte 524 to byte 783.

Segment 2 begins with sequence number 784, covering the next 260 bytes. This segment spans from byte 784 to byte 1043.

Segment 3 starts with sequence number 1044, also covering 260 bytes. This segment spans from byte 1044 to byte 1303.

Segment 4 begins at sequence number 1304, covering the next 260 bytes. This segment spans from byte 1304 to byte 1563.

6. If the action of sending the acknowledgment message is removed, both the sender and receiver will end up in a deadlock, where they are both waiting for events that will never happen. Example:

   The sender sends packet 0 (pkt0) and moves into the "Wait for ACK0" state, where it waits for an acknowledgment (ACK0) from the receiver. The receiver, in the "Wait for 0 from below" state, receives a corrupted version of packet 0 (pkt0). Normally, the receiver would send an ACK0 back to the sender to acknowledge receipt, but without the action to send this acknowledgment, the receiver simply re-enters the same state, awaiting packet 0 again. As a result, the sender is stuck waiting for an ACK, which never arrives because the receiver doesn’t send it. Meanwhile, the receiver is stuck waiting for a data packet from the sender, which it has already received. This causes a deadlock where the sender is waiting for an acknowledgment that won’t come, and the receiver is waiting for data it has already received. In conclusion, removing the acknowledgment action leads to a deadlock, preventing the protocol from functioning properly.

7. We need $\log_2(N)$ because in TCP slow start, the congestion window grows exponentially with each round-trip time (RTT). Specifically, the window size doubles with each RTT:

   After 1 RTT, the window can send 2 segments (since it starts with 1 segment and doubles).

   After 2 RTTs, the window size grows to 4 segments.

   After 3 RTTs, the window size grows to 8 segments, and so on.

   The relationship can be formulated as:

   After k RTTs, the window size is 2^k segments.

8.

   a) The periods where TCP slow start is in effect occur between rounds 1 to 6 and again between rounds 23 to 26. During this time, the congestion window size grows exponentially as the sender increases its window size until the slow start threshold is reached.

   b) TCP enters the congestion avoidance phase during the periods from round 6 to 16 and from round 17 to 22. In this phase, the congestion window increases more gradually, following a linear growth pattern.

   c) After round 16, TCP Reno detects packet loss via triple duplicate ACKs. If the loss were instead detected by a timeout, the congestion window would drop to 1, and slow start would restart.

   d) By round 22, if packet loss is detected via a timeout, the congestion window size would reset to 1, indicating that slow start is being reinitiated.

e) At the start of the connection, it’s set to 32 segments. This is the value at which the transition from slow start to congestion avoidance occurs, marking the point where TCP shifts from exponential to linear growth of the congestion window.

f) When packet loss is detected, it’s set to half the size of the congestion window. For instance, in round 16, if the congestion window is 42, then threshold would be halved to 21 in round 18.

g) If packet loss occurs at round 22 with a congestion window of 29, the threshold would be halved to 14 by round 24. This halving mechanism helps control the flow of data and prevents congestion.

h) When detecting packet loss, the threshold is halved, and the congestion window is updated to the threshold plus 3 times the Maximum Segment Size (MSS). In this case, if threshold is set to 8, the new congestion window will be 7 (8 + 3 MSS).

i) In the 17th round, if TCP Tahoe is in use, the congestion window is reset to 1 after a packet loss or the receipt of triple duplicate ACKs. The threshold remains at 21 during this round.

j) The total number of packets sent between rounds 17 and 22 is as follows:

Round 17: 1 packet

Round 18: 2 packets

Round 19: 4 packets

Round 20: 8 packets

Round 21: 16 packets

Round 22: 21 packets

The total sum of packets sent is 52.