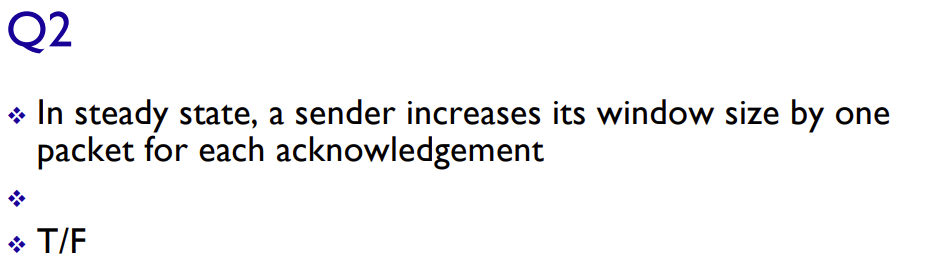


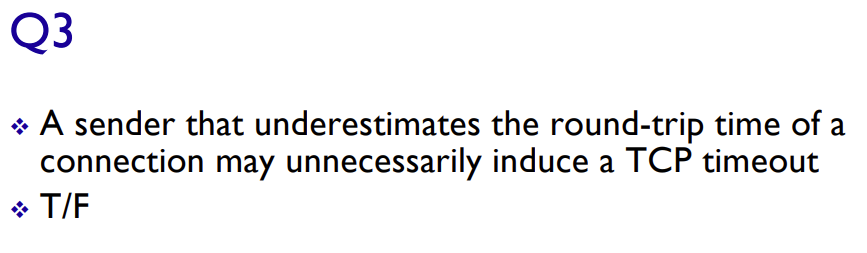
True.

The window size doubles exponentially until a packet loss is detected or a threshold (known as ssthresh, or slow start threshold) is reached, which transitions the process into the congestion avoidance phase.



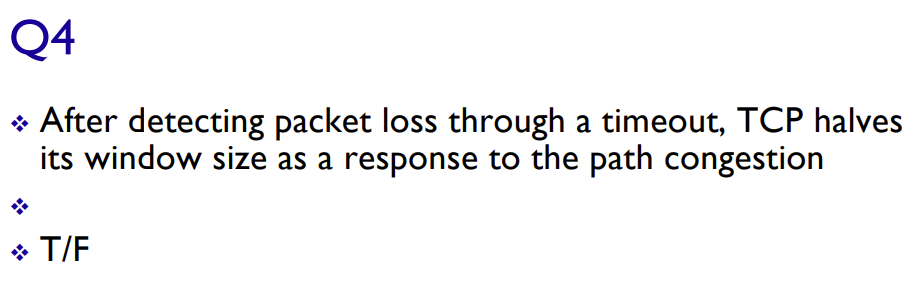
True.

In the steady state/congestion avoidance phase, the sender increases its congestion window slower than in slow start.

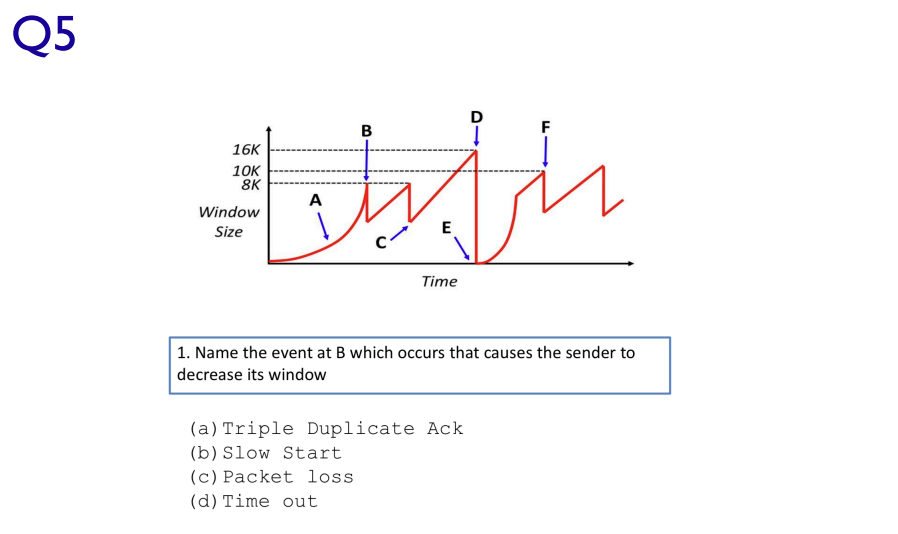


True.

If a sender underestimates the round-trip time of a connection, it could set the retransmission timeout too short. Making the sender trigger a timeout prematurely, because it assumes the packets were lost when in reality, the acknowledgments are simply delayed due to the underestimated RTT.

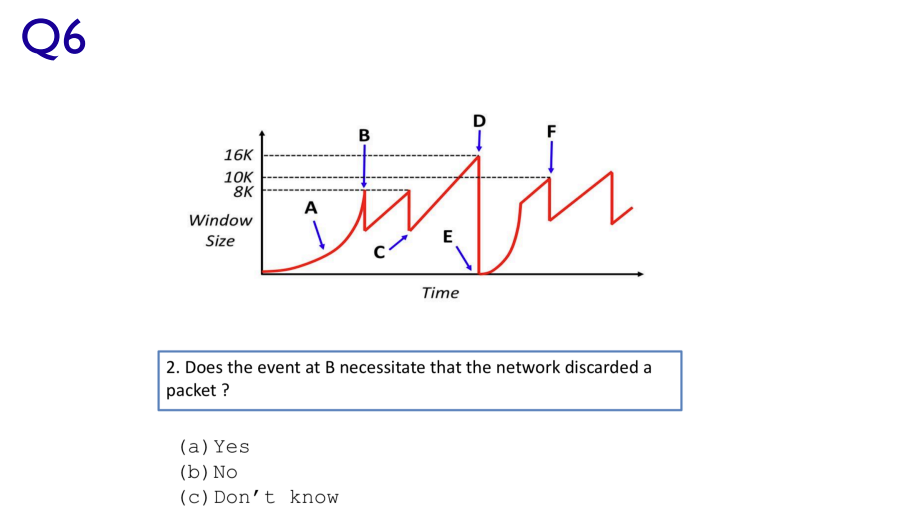
False.

Halving the window size is done when packet loss is detected through duplicate acknowledgements. If a packet loss detected through timeout occurs, the sender resets the congestion window to 1 segment size and enters the slow start phase again, it does this to probe the network capacity again.



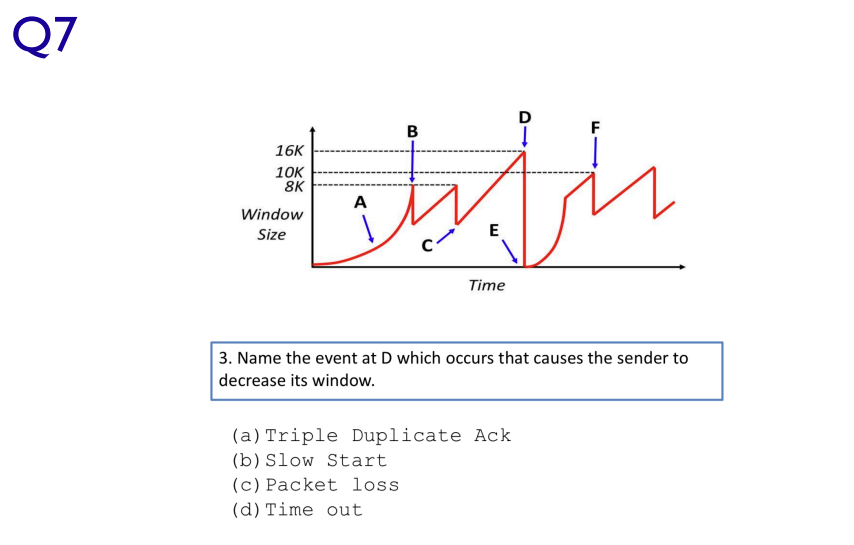
(a).

When a duplicate acknowledgement occurs, the window size is halved like in this case.



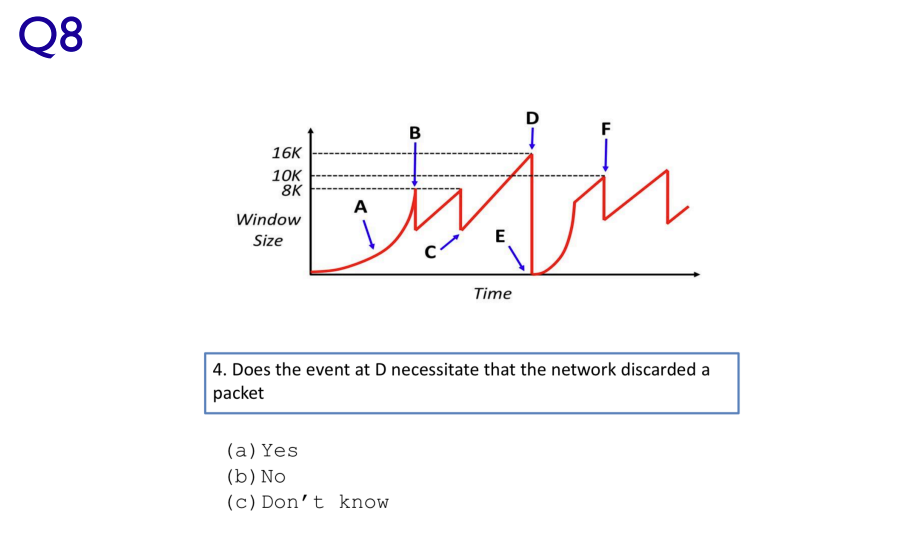
(b).

It is most likely that the event at B was caused by a triple duplicate acknowlegment, which would have discarded a packet and triggered a retransmission. However, it could have been caused by an out of order delivery of packets either, which would NOT have discarded the packet. So no, the event at B does not necessitate that the network discarded a packet.



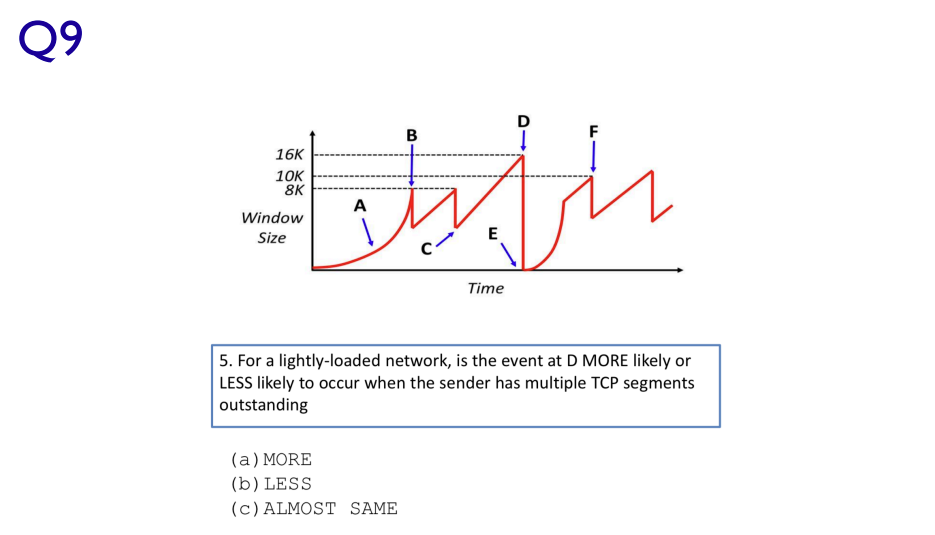
(d).

When a timeout occurs, the window size drops back to 1 and restarts the slow start phase which is what we see in the diagram.



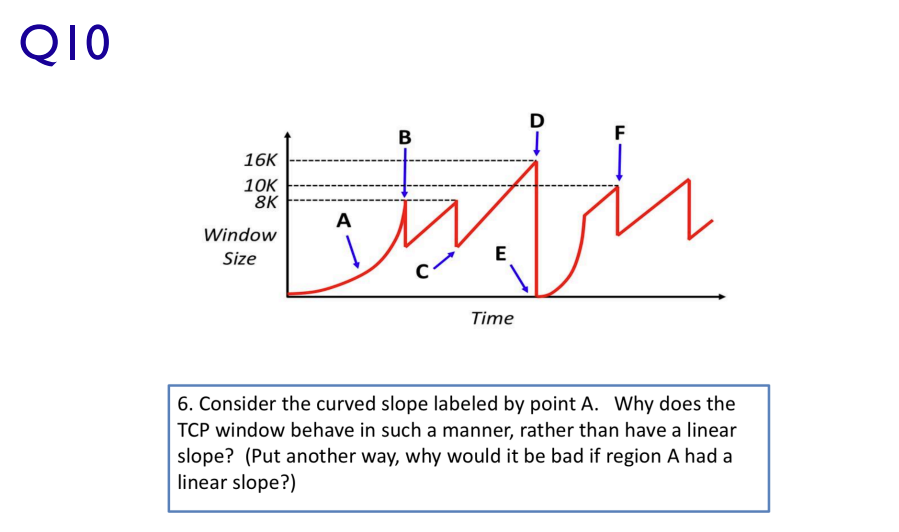
(a).

If a network timeout is triggered a packet must have been lost.



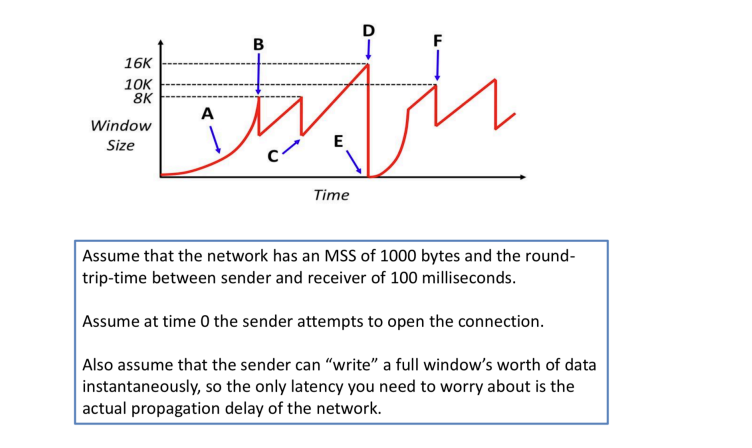
(b).

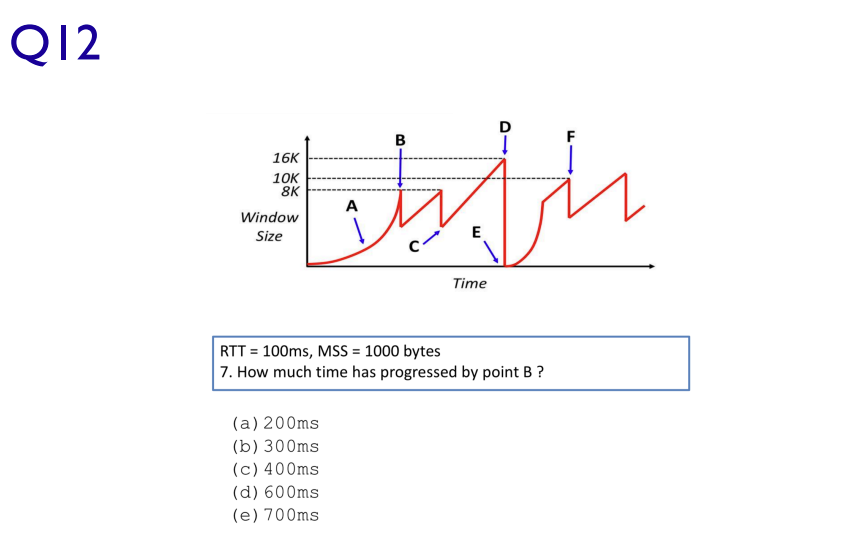
In lightly loaded networks, congestion is not as likely due to the network having enough capacity to handle the traffic. Since the event at D represents a congestion event, it wouldn't be as likely to occur in a lightly loaded network, even if the sender has multiple outstanding TCP segments. So, the network can handle more traffic without dropping packets.



(a).

It has exponential growth to find the maximum size that can be accepted by the destination as quickly as possible. If it used linear growth, it would be far too slow.

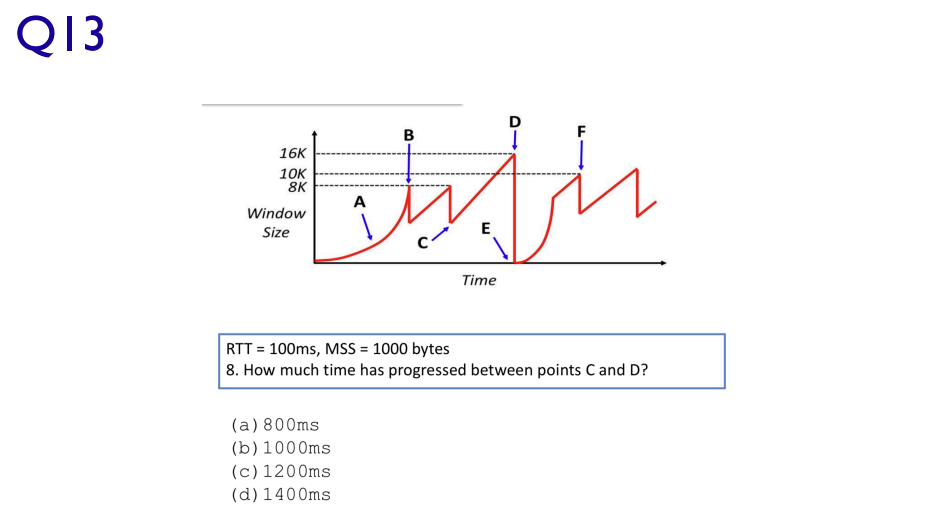


(b).

At the start the window size is 1MSS, after the first RTT, the window size would double to be 2MSS, second would be 4MSS and the third would be 8MSS.

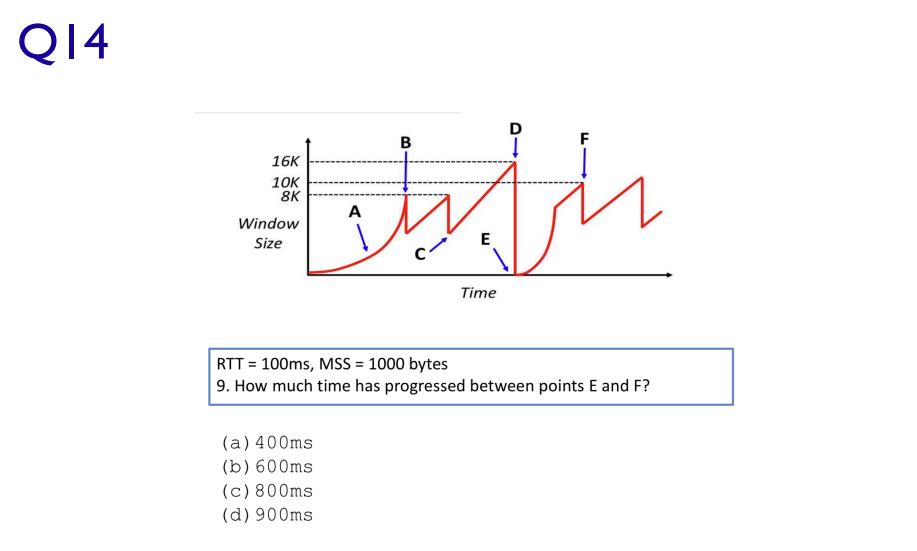
At the window size of 8MSS, 3RTT’s have passed. Each RTT is 100ms:

3 \* 100 ms = 300 ms

(b)

Point C has a window size of 6k, our target window size point D is 16k.

With an RTT of 100ms and MSS of 1000 bytes it will take 1000ms to go from 6k to 16k linearly.

Between (a) and (b)

1. 400ms
2. 600ms

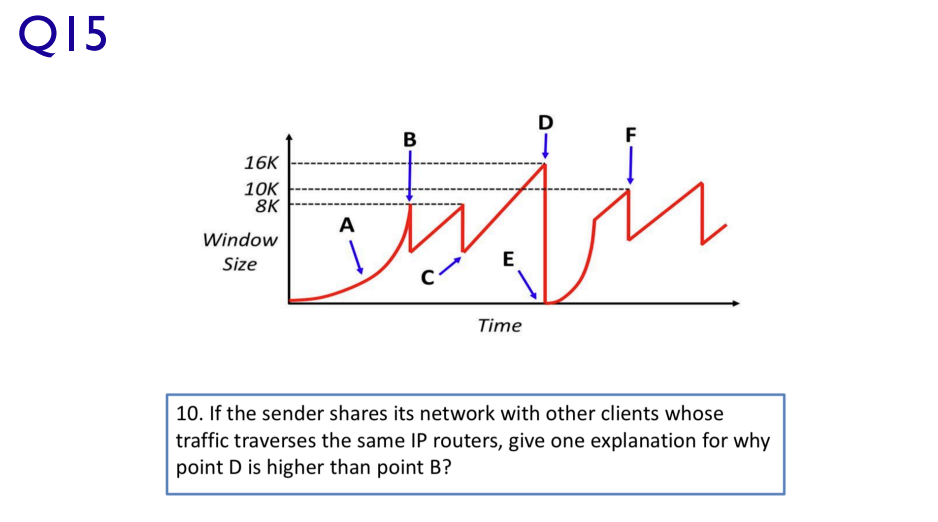
It will take point E, 3 RTT’s to reach 8000 bytes which is 300ms.

Then from 8000 bytes we convert to linear growth to get to point F.

With an RTT of 100ms and MSS of 1000 bytes it will take 200ms of linear growth to reach 10k.

300 + 200 = 500ms

Answer = 500ms

When the window size dropped at B it was likely because there was no remaining bandwidth to handle more data, so the window size dropped and began increasing again linearly.

Later we see at point D the window size to be double that of when it dropped at B, this is likely due to other devices on the same network sharing bandwidth at point B which decreases by the time we get to point D.