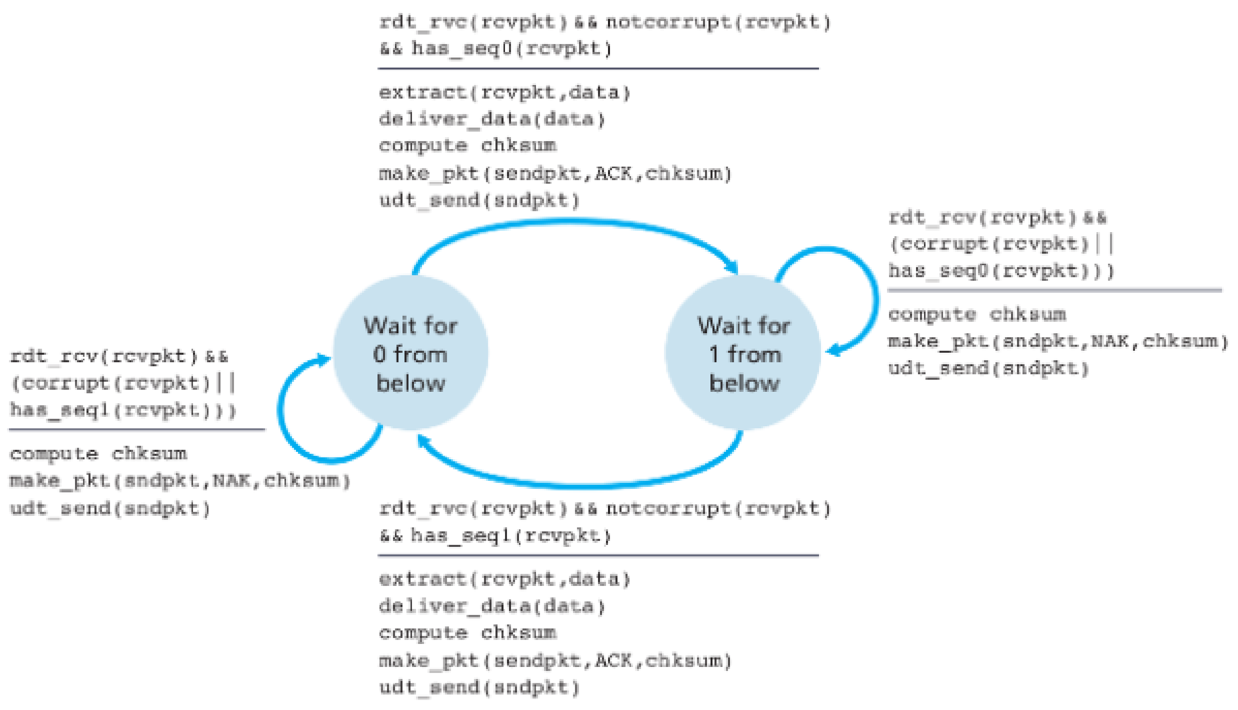


Figure 3.57 ♦ An incorrect receiver for protocol `rdt 2.1`

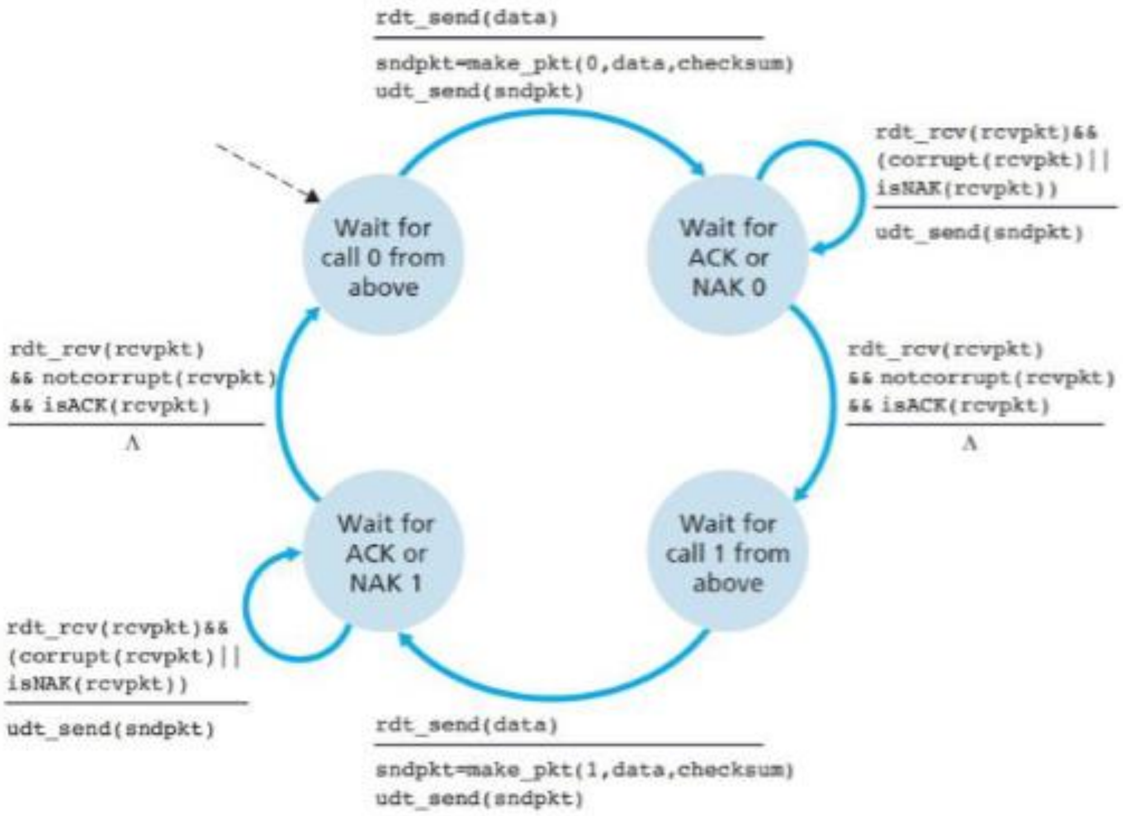


Figure 3.11 ♦ `rdt2.1` sender

P6. Consider our motivation for correcting protocol rdt2.1. Show that the receiver, shown in Figure 3.57, when operating with the sender shown in Figure 3.11, can lead the sender and receiver to enter into a deadlock state, where each is waiting for an event that will never occur.

Receiver would have to be waiting for 0 from below while sender is only sending packet 1. This could happen if while sender state is wait for call 1 from above and receiver state is wait for 1 from below, sender sends packet 1 and receiver sends a corrupted ACK, but moves on to state wait for 0 from below. The sender having received the corrupted ACK will transmit the packet 1 again, but receiver will send NAK since its waiting for packet 0. The process will then loop, causing deadlock

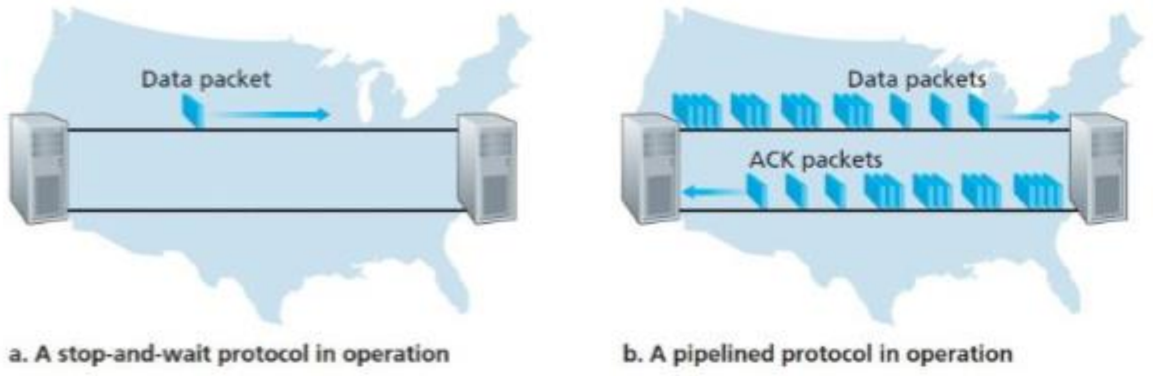

Data packet
Data packets
ACK packets
a. A stop-and-wait protocol in operation
b. A pipelined protocol in operation

Figure 3.17 ♦ Stop-and-wait versus pipelined protocol

P15. Consider the cross-country example shown in Figure 3.17. How big would the window size have to be for the channel utilization to be greater than 98 percent? Suppose that the size of a packet is 1,500 bytes, including both header fields and data. assume the bandwidth and the one-way propagation delay of the cross-country link is 100Mbps and 25ms, respectively.

1500 bytes = 12000 bits

U[sender] = N * [(L/R) / (RTT + L/R)] = 0.98

0.98 = N * [(12,000/100,000,000) / (25 + 12,000/100,000,000)

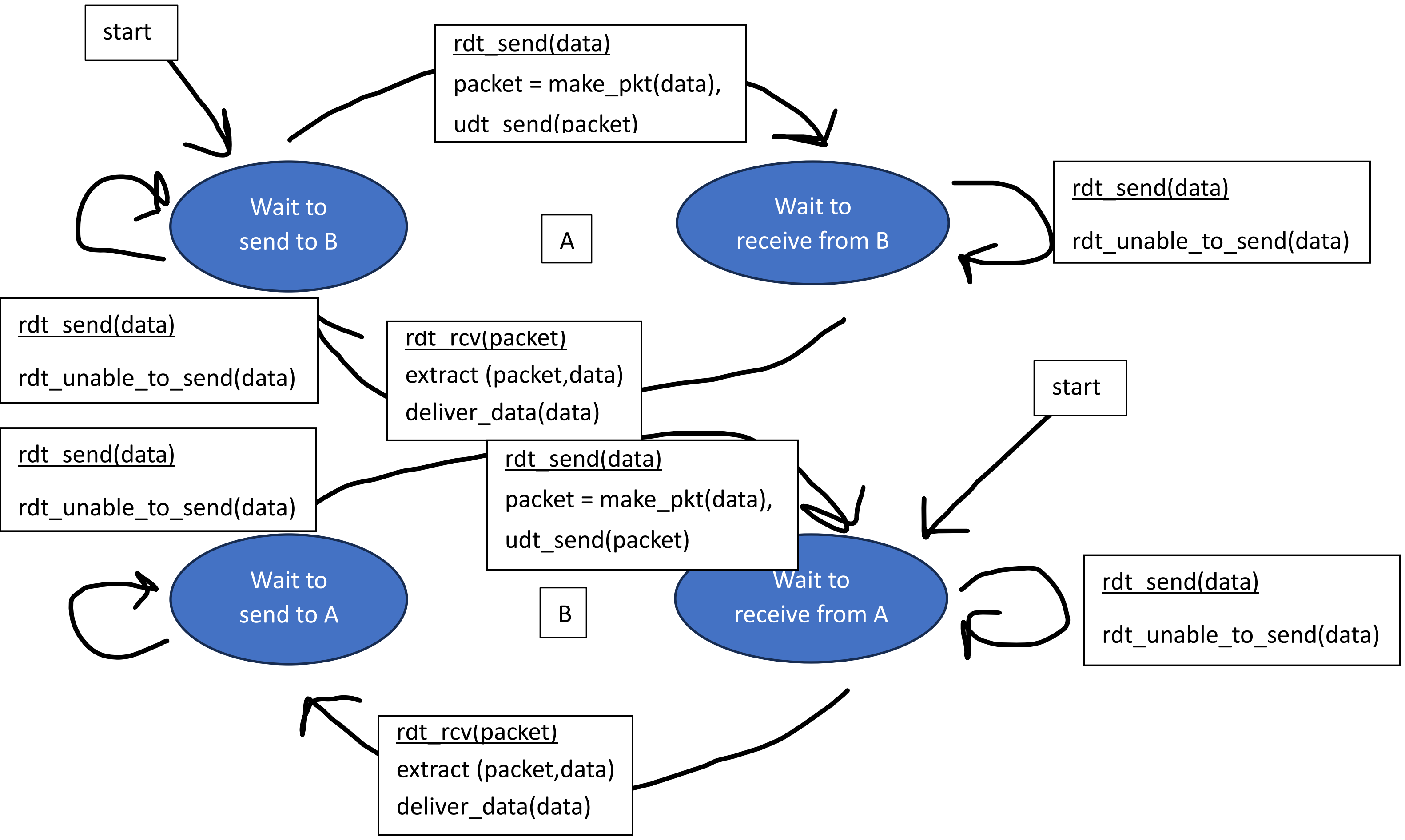
N = 205.15 packets in window

P17. Consider two network entities, A and B, which are connected by a perfect bi-directional channel (i.e., any message sent will be received correctly; the channel will not corrupt, lose, or re-order packets). A and B are to deliver data messages to each other in an alternating manner:

First, A must deliver a message to B, then B must deliver a message to A, then A must deliver a message to B and so on. If an entity is in a state where it should not attempt to deliver a message to the other side, and there is an event like rdt_send(data) call from above that attempts to pass data down for transmission to the other side, this call from above can simply be ignored with a call to rdt_unable_to_send(data), which informs the higher layer that it is currently not able to send data. [Note: This simplifying assumption is made so you don't have to worry about buffering data.]

Draw a FSM specification for this protocol (one FSM for A, and one FSM for B!). Note that you do not have to worry about a reliability mechanism here; the main point of this question is to create a FSM specification that reflects the synchronized behavior of the two entities. You should use the following events and actions that have the same meaning as protocol rdt1.0 in Figure 3.9:

rdt_send(data), packet = make_pkt(data), udt_send(packet), rdt_rcv(packet), extract (packet,data), deliver_data(data). Make sure your protocol reflects the strict alternation of sending between A and B. Also, make sure to indicate the initial states for A and B in your FSM descriptions.



P22. Consider the GBN protocol with a sender window size of 4 and a sequence
number range of 1,024. Suppose that at time t, the next in-order packet that the
receiver is expecting has a sequence number of k. Assume that the medium
does not reorder messages. Answer the following questions:

a. What are the possible sets of sequence numbers inside the sender's win-
dow at time t? Justify your answer.

Set 1:

Suppose the receiver has received packet k-1, and has ACKed that and all other preceding packets. If all of these ACK's have been received by sender, then sender's window is [k, k+N-1].

Set 2:

Suppose next that none of the ACKs have been received at the sender. In this second case, the sender's window contains k-1 and the N packets up to and including k-1. The sender's window then is [k-N,k-1].

So the senders window in the range [k-N,k].

b. What are all possible values of the ACK field in all possible messages cur-
rently propagating back to the sender at time t? Justify your answer.

If the receiver is waiting for packet k, then it has received and ACKed packets N-1 through k-1. If none of those N ACKs have been yet received by the sender, then ACK messages with values of [k-N, k-1] may still be propagating back. Because the sender has sent packets [k-N, k-1], then the sender has already received an ACK for k-N-1. Once the receiver has sent an ACK for k-N-1 it will never send an ACK that is less than k-N-1. So the range of incoming ACK values can range from k-N-1 to k-1.

P25. We have said that an application may choose UDP for a transport protocol
because UDP offers finer application control (than TCP) of what data is sent
in a segment and when.

a. Why does an application have more control of what data is sent in a segment?

In TCP, the application writes data to the connection send buffer and TCP will grab bytes but may put more or less than a single message in a segment. However, UDP encapsulates whatever the application gives it into a segment. So if the application gives UDP a message, the entire message will be the payload of the UDP segment. Therefore an application has more control of what data is sent in a segment with UDP.

b. Why does an application have more control on when the segment is sent?

Unlike UDP, TCP has flow control and congestion control, so there may be significant delay from the time when an application writes data to its send buffer until when the data is given to the network layer

P27. Host A and B are communicating over a TCP connection, and Host B has
already received from A all bytes up through byte 126. Suppose Host A then
sends two segments to Host B back-to-back. The first and second segments
contain 80 and 40 bytes of data, respectively. In the first segment, the
sequence number is 127, the source port number is 302, and the destination
port number is 80. Host B sends an acknowledgment whenever it receives a
segment from Host A.

a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

Seq_no 207, port 302, dest 80

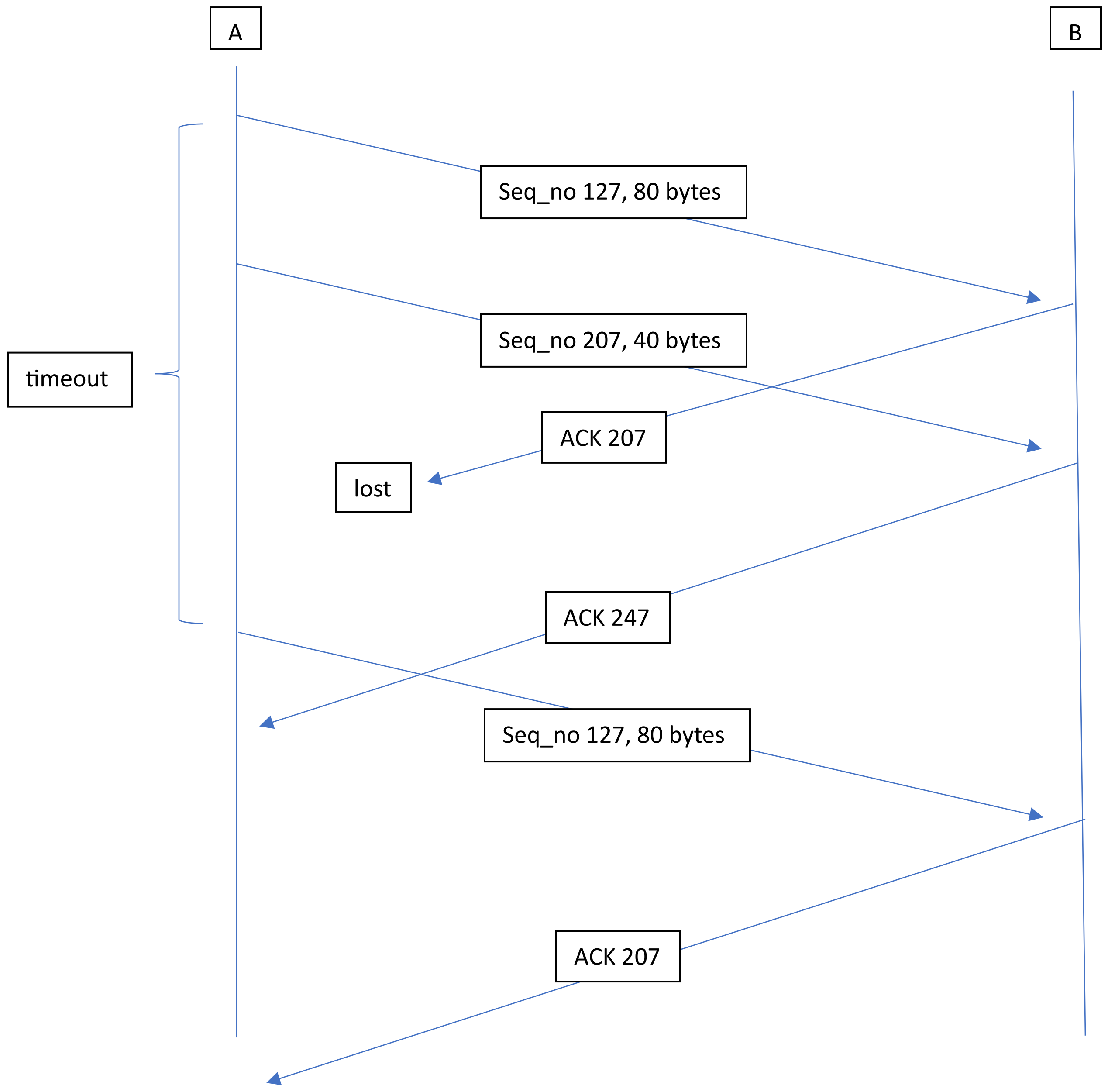
b. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number,
the source port number, and the destination port number?

ACK 207, port 80, dest 302

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

ACK 127

d. Suppose the two segments sent by A arrive in order at B. 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.



P37. Compare GBN, SR, and TCP (no delayed ACK). only consider the comparison between GBN and TCP protocols (do not consider SR protocol). Assume that the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments 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. In the end, all 5 data segments have been correctly received by Host B.

a. How many segments has Host A sent in total and how many ACKs has Host B sent in total? What are their sequence numbers? Answer this question for all three protocols.

GBN:

A Sent 123452345

Total 9 segments sent

B sent Ack 1111234

Total 8 ACKs sent

TCP:

A Sent 123452

Total 6 segments sent

B sent Ack 22226

Total 5 ACKs sent

b. If the timeout values for all three protocol are much longer than 5 RTT, then which protocol successfully delivers all five data segments in shortest time interval?

TCP since it has fast retransmit and only sends the lost segment(s) without wait

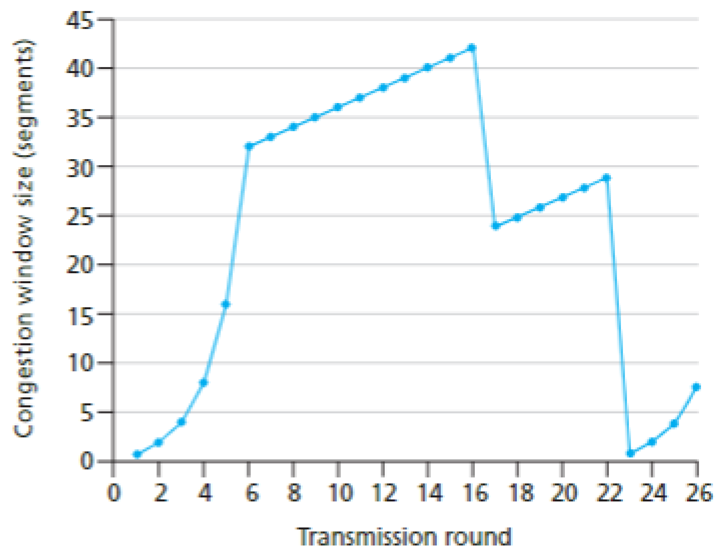


Figure 3.58 ♦ TCP window size as a function of time

P40. Consider Figure 3.58. 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.

1-6 and 23-26

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

6-23

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

Triple duplicate ACK

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

timeout

e. What is the initial value of `ssthresh` at the first transmission round?

32 kB

f. What is the value of `ssthresh` at the 18th transmission round?

30660 B

g. What is the value of `ssthresh` at the 24th transmission round?

21170 B

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

7

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`?

Window size = 4, sshthresh = 21170 B

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?

Sshthresh = 1, window size = 21

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?

52 packets

P44. Consider sending a large file from a host to another over a TCP connection that has no loss.

a. Suppose TCP uses AIMD for its congestion control without slow start. Assuming cwnd increases by 1 MSS every time a batch of ACKs is received and assuming approximately constant round-trip times, how long does it take for cwnd increase from 6 MSS to 12 MSS (assuming no loss events)?

1 RTTs to 7 MSS.

2 RTTs to 8 MSS.

3 RTTs to 9 MSS.

4 RTTs to 10 MSS.

5 RTTs to 11MSS.

6 RTTs to 12 MSS.

Total 6 RTTs

b. What is the average throughout (in terms of MSS and RTT) for this connection up through time = 6 RTT?

Avg = total segments / RTT = (6+7+8+9+10+11) / 6 = 8.5