CHAPTER 3

PRACTICE SET

Questions

Q3-1. The answer is no. Host-to-host and process-to-process communication are needed because each computer in the Internet is designed to do multiple tasks: to run multiple application-layer programs.

Q3-3. Although any port number can be used for the client and server and they can be the same in this private communication, it is recommended to follow the division specified by ICANN:

- **a.** The client port number should be chosen from the dynamic range, 49,152 to 65,535.
- **b.** The server port number also should be chosen from the dynamic range, 49,152 to 65,535.
- **c.** It is advisable to choose different port numbers for the server and the client to be able to better debug the programs.

Q3-5. We check each protocol one by one:

- **a.** The protocol can be Stop-and-Wait with the receive window size of 1 and the send window size of 1.
- **b.** The protocol can also be Go-Back-*N* with the receive window size of 1 and the send window size of *n* packets.
- **c.** The protocol cannot be Selective-Repeat because the size of both windows should be the same (1), which means the protocol is Stop-and-Wait, not Selective-Repeat.

Q3-7. We describe the advantage and disadvantage of each first:

- **a.** The advantage of using the Go-Back-*N* protocol is that we can have a larger send window size. We can send more packets before waiting for their acknowledgment. The disadvantage of using this protocol is that the receive window size is only 1. The receiver cannot accept and store the out-of-order received packets; they will be discarded. Discarding of the out-of-order packets means resending these packets by the sender, resulting in congestion of the network and reducing the capacity of the pipe. So the advantage seen by a larger send window may disappear by filling the network with resent packets.

b. The advantage of using the Selective-Repeat protocol is that the receive window can be much larger than 1. This allows the receive window to store the out-of-order packets and avoids resending them to congest the network. On the other, the send window size of this protocol is half of the Go-Back-*N*, which means that we can send fewer packets before waiting for the acknowledgment.

c. If the bandwidth-delay product of the network is large, the reliability is good, and the delay is low, we should choose the Go-Back-*N* protocol to use more of the network capacity. On the other hand, if the bandwidth-delay product is small, or the network is not very reliable, or the network creates long delays, we need to use Selective-Repeat.

Q3-9. The networks need to be carefully designed to make the time between the two wraparounds as long as possible. For example, in a protocol that uses the sequence number field of size 3 ($m = 3$), every $2^m = 8$ packets have the same sequence number. If the previous packet with sequence number x (or its accidentally created duplicate) is still wandering in the network arrives at the destination, the receiver may confuse this with the expected new packet, also with sequence number x.

Q3-11. The transport-layer packets are encapsulated in the datagram at the network layer. The router through which the datagrams need to pass to reach their destination may be congested and drop the packets.

Q3-13. The rest of the packets ($2^m - 2$) are supposed to be in transit, filling the pipe. The size of the receive window is chosen to be 1 to accept only one packet, the one expected, and not out-of order packets. The receiver cannot be overwhelmed because it holds only one packet in its window. When the only packet in the window is consumed by the upper-layer protocol, the receive window slides to make it possible to receive the next packet in transit. If any packet in transit arrives before the window slides, it is discarded.

Q3-15. The protocol field of the datagram (see Figures 4.24 and 4.25 in Chapter 4) defines the transport-layer protocol that should receive the transport-layer packet. If the value is 06, the protocol is TCP; if the value is 17, the protocol is UDP.

Q3-17. UDP is preferred because each user datagram can be used for each chunk of data. However, a better solution is the new transport protocol, SCTP, discussed in Chapter 8.

Q3-19. The answer is positive. There is nothing in the UDP or TCP protocol that requires the use of the IP protocol. A UDP user datagram or a TCP segment can be encapsulated in an Ethernet frame. However, the protocol field of the Ethernet needs to define which protocol is directly used in this case.

Q3-21. There are two parties involved in a two-way communication. TCP allows each party to stop sending while the other party is still sending data. This means we need at least two FIN segments (or data segments in which the FIN bit is set). Since each FIN segment should be acknowledged, we normally need four segments for connection termination. Sometimes the second FIN segment can be combined with the ACK segment to reduce the number of segments to three.

Q3-23. The way the sequence number is determined depends on whether the segment is the first or not.

a. The sequence number of the first segment is the value of ISN, which is normally selected according to the RFC 793.

b. The sequence number of any segment except the first is determined by the sequence number of the previous segment plus the number of sequence numbers consumed by the previous segment. In other words, the sequence number of any segment is $y = x + n$, in which x is the sequence number of the previous segment and n is the count of sequence numbers consumed by the previous segment. If the previous segment does not consume any sequence numbers, $n = 0$.

Q3-25.

a. A SYN segment consumes one sequence number.

b. An ACK segment does not consume any sequence numbers.

c. A SYN + ACK segment consumes one sequence number because it is a SYN segment.

d. A data segment consumes n sequence numbers, where n is the number of bytes carried by the segment.

Q3-27. We cannot say. The size of the window field makes the size of the window 2^{16} bytes, but the sequence number is 2^{32}. The size of the window is definitely much smaller (actually 2^{16} times smaller) than half of the range of the sequence numbers. The requirement of each protocol, Go-Back-*N* and Selective-Repeat, is satisfied. However, as we mentioned in the text, TCP is closer to Selective-Repeat in other aspects.

Q3-29.

a. The maximum size of the TCP header is 60 bytes (20 bytes of header and a maximum 40 bytes of options).

b. The minimum size of the TCP header is 20 bytes.

Q3-31. A FIN segment closes the connection in only one direction. The party that issues a FIN segment cannot send data to the other party, but needs to acknowledge the data received from the other party. The other party can still send data and acknowledgments. To completely close the connection, two FIN segments are required, one in each direction. Of course, the FIN and ACK segments can always be combined.

Q3-33. For example, the SYN and the RST flags cannot be set in a segment. One of them requests the starting of a connection; the other requests that the connection be aborted. Another example is the combination of SYN and FIN.

Q3-35. This is done through acknowledgment and retransmission. If a packet is lost or corrupted, it will be resent. As an analogy, assume the postal service is unreliable. When we send a letter to a friend, we can ask for confirmation with a postcard. If we do not receive the postcard, we can resend the copy of the letter until we finally get the confirmation.

Q3-37. A connection is distinguished by a pair of socket addresses, one for each end. Although the socket addresses at Bob's site are the same in this case, the socket addresses at Alice's site are different. Each socket address at Alice's site has a different ephemeral port number.

Q3-39. The sender window in TCP is originally the same size as the receiver window, but as the congestion builds up in the network, it can become smaller than the receiver window. It can never become larger than the receiver window. The size of the sender window is the smaller of the window size either dictated by the congestion or determined by the receiver.

Q3-41. The answer depends on whether the segment is carrying data or not.

a. If the segment carries data, the sequence number defines the number of the first bytes in the segment.

b. If the segment carries no data (pure control segment), the sequence number identifies the segment itself.

Q3-43. The use of checksum in UDP is optional. If the sender does not use the checksum, it fills the checksum field with sixteen 0s. The use of the checksum in TCP, on the other hand, is mandatory. The sender should calculate the checksum; otherwise, the checksum calculation at the receiver fails and the segment is dropped.

Q3-45. The received segment is a duplicate. The TCP client needs to discard the segment and immediately send an ACK with acknowledgment number 2001. This reaction helps the server to update itself if the previous ACK with acknowledgment number 2001 is somehow lost (Rule 6 in ACK generation).

Q3-47. The server needs to store the new bytes and immediately send an ACK with acknowledgment number 2901. This reaction prevents the retransmission of the previous segment by the client (Rule 3 in ACK generation).

Q3-49. The server needs to store the new bytes, but send a segment with sequence number 4001 and acknowledgment number 8001 (piggybacking). This saves bandwidth because the data needs to go in the other direction, and it is better to acknowledge the received bytes in the same packet (Rule 1 in ACK generation).

Q3-51. The six rules we mentioned for ACK generation are related to flow and error control and are applicable during the data transmission phase, not during the connection establishment phase. In this case, a SYN needs to be acknowledged by a (SYN + ACK) segment if the server accepts the connection or by an RST segment otherwise.

Q3-53. The six rules we mentioned for ACK generation are related to flow and error control and are applicable during the data transmission phase, not the connection termination phase. In this case, the FIN segment should be acknowledged immediately if there is no data to send or acknowledged in the next data segment.

Problems

P3-1. The domain of IP addresses is universal. A device directly connected to the Internet needs a unique IP address (see exceptions in Chapter 4). The domain of port numbers is local; they can be repeated. Two computers running the HTTP server process use the same well-known port number (80); two computers running the HTTP client process can use the same ephemeral port number.

P3-3. The sequence number of any packet can be found using the following relation:

$$\textbf{seqNo} = \textbf{(starting segNo + packet number} - \textbf{1) mod } \mathbf{2}^{m}$$

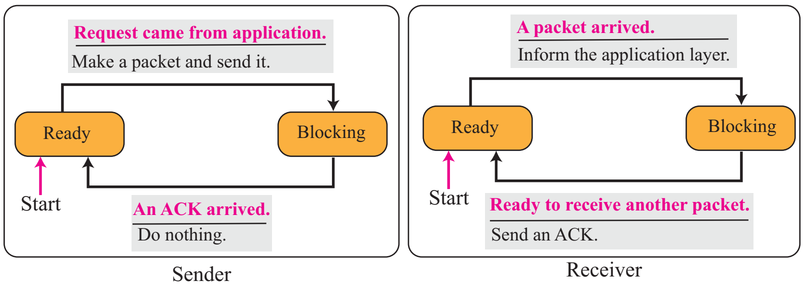
in which m is the number of bits used to define the sequence number. The sequence number in this case is

$$\textbf{seqNo} = (0 + 100 - 1) \textbf{ mod } 2^5 = 99 \textbf{ mod } 32 = 3$$

P3-5.

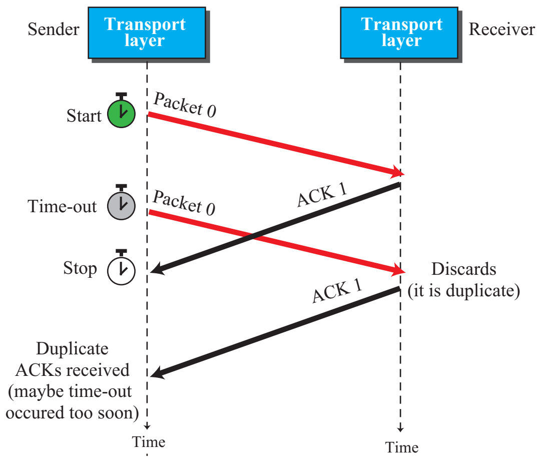
Stop-and-Wait:	**Max Send $W_{size} = 1$**	**Max Receive $W_{size} = 1$**
Go-Back-*N*:	**Max Send $W_{size} = 2^5 - 1 = 31$**	**Max Receive $W_{size} = 1$**
Selective-Repeat:	**Max Send $W_{size} = 2^5 / 2 = 16$**	**Max Send $W_{size} = 2^5 / 2 = 16$**

P3-7. The following shows the states and events. The sender needs two states: ready and blocking. The receiver also needs these two states. To send a packet, the sender should be in the ready state and receives data from the application layer. The sender cannot send a packet when it is in the blocking state. The receiver accepts a packet from the sender when it is in the ready state. It cannot accept a packet when it is in the blocking state. When the receiver becomes ready, it sends an ACK packet and moves to the ready state.



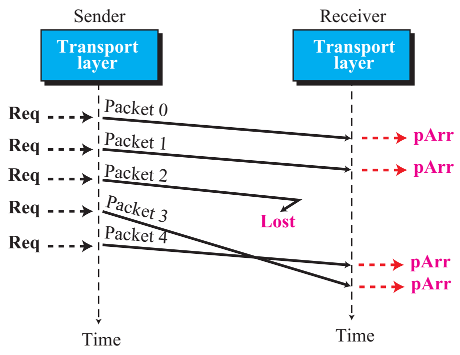
Request came from application.
Make a packet and send it.
Ready
Blocking
Start
An ACK arrived.
Do nothing.
Sender
A packet arrived.
Inform the application layer.
Ready
Blocking
Start
Ready to receive another packet.
Send an ACK.
Receiver

P3-9. The following shows the case. It happens when an ACK is delayed and the time-out occurs. The sender resends the packet that is already acknowledged by the receiver. The receiver discards the duplicate packet, but resends the previous ACK to inform the sender that there is a delay in the network.

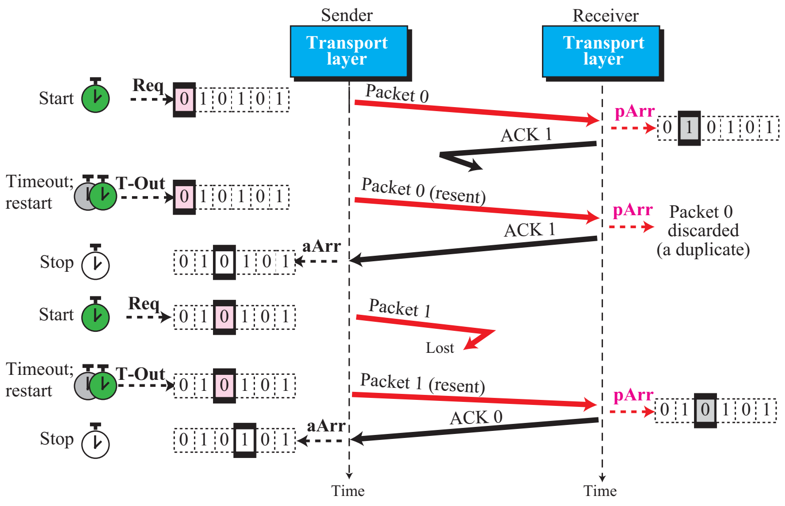


Receiving a duplicate ACK may alert the sender to increase the time-out to prevent resending the packets prematurely. Note that if the receiver ignores the duplicate packet and does not send the second ACK, the sender believes that the first packet is lost and the first ACK is actually acknowledging the packet that is resent. Duplicate ACKs give the clue about what is happening.

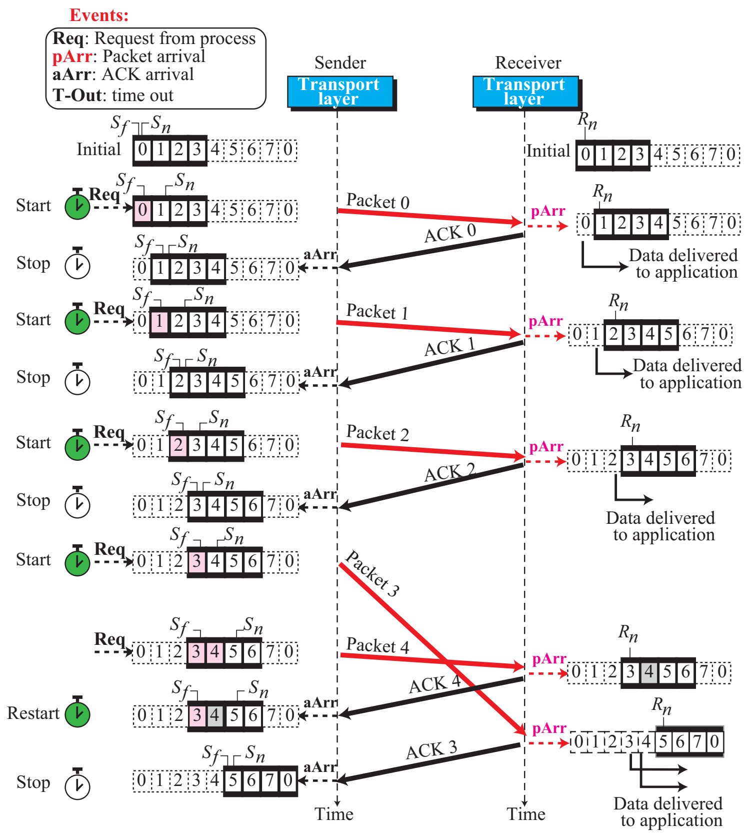
P3-11. The following shows the outline. Note that since the simple protocol provides no error control, if a packet is lost, the receiving process is responsible for finding a solution. The transport layer is not even aware that this has happened. The packets may also be delivered out of order to the receiving process. The responsibility again is on the receiving process to reorder the packets.



P3-13. See the following:



P3-15. The following shows the solutions:



P3-17. We assume each event is independent.

a. seqNo = 15.

b. Five packets with seqNos set to 10, 11, 12, 13, and 14 are to be resent.

c. $S_f = 13$ and $S_n = 15$.

d. The size of the window remains the same. Max $W_{size} = 64 - 1 = 63$.

e. $S_f = 18$ and $S_n = 21$. Next state = ready.

f. $R_n = 17$. Action: message is delivered and an ACK with ackNo = 17 is sent.

P3-19. In each case we first define the bandwidth-delay product (BDP) in bits and then find it in the number of packets:

a. BDP = 1 Mbps × 20 ms = 20,000 bits = 20 packets

b. BDP = 10 Mbps × 20 ms = 200,000 bits = 100 packets

c. BDP = 1 Gbps × 4 ms = 4,000,000 bits = 400 packets

P3-21. We first calculate the average round-trip time (RTT) and the number of packets in the pipe before finding the sizes of the windows, the value of *m*, and the time-out value.

a. Average RTT = 2 × (5,000 Km) / (2 × 10^8) = 50 ms.

b. The bandwidth-delay product = 1 Gbps × 50 ms = 50,000,000 bits.

c. The bandwidth-delay product = 50,000,000 bits / 50,000 bits = 1000 packets.

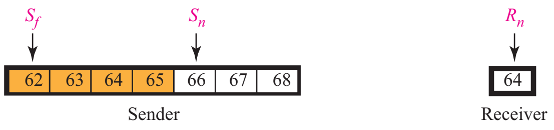
d. The maximum send window size should be 1000 to allow not more than 1000 packets in the pipe.

e. The maximum receive window size should also be 1000 packets.

f. We know that the (window size) ≤ (2^{m-1}) or 1000 ≤ (2^{m-1}). This means that we need to choose $(m - 1)$ to be at least 10 or $m = 11$. The sequence numbers are then 0 to 2047.

g. The timeout value should be at least the average RTT = 50 ms to avoid early retransmission of the packets and to prevent congestion.

P3-23. The following shows the situation.



a. If the receiver expects a packet with sequence number 64 and packets with sequence numbers 62 to 65 are already sent but not acknowledged, it means that two packets with sequence numbers 64 and 65 are in transit from the sender to the receiver.

b. If the sender expects the acknowledgment for packet 62, but the value of R_n = 64, it means that the ACK packets with acknowledgment numbers 62 and 63 are in transit from the receiver to the sender.

P3-25.

a. The minimum size is 8 bytes (header without payload).

b. Although the theoretical maximum size is 65,535 bytes, since a user datagram needs to be encapsulated in a single IP datagram (UDP is a connectionless protocol) and the maximum payload of an IP datagram is 65,515 bytes (see Figure 4.24 in Chapter 4), we should say the maximum size of a UDP datagram is only 65,515 bytes.

c. The minimum size of the application-layer payload is zero bytes.

- **d.** The maximum size of the application-layer payload is 65,507 bytes (65,515 – 8).

P3-27.

- **a.** The source port number is the first 16 bits or $(0045)_{16} = 69$.
- **b.** The destination port number is the second 16 bits $(DF00)_{16} = 57{,}088$.
- **c.** The total length of the datagram is the third 16 bits $(0058)_{16} = 88$ bytes.
- **d.** The length of the data is 88 – 8 = 80 bytes.
- **e.** The message is from a server with a small (well-known) port number to a client with a large (ephemeral) port number.
- **f.** The well-known port number 69 belongs to TFTP.
- **g.** The sender has not calculated the checksum for this packet because the value of the checksum is all zeros.

P3-29. The number $(0111)_2$ in decimal is 7. The total length of the header is then $(7 \times 4) = 28$. The base header is 20 bytes. The segment has 28 – 20 = 8 bytes of options.

P3-31. The following are eight out of 64 possible combinations that are normally used:

000000	$\rightarrow$	**A data segment with no acknowledgment**
110000	$\rightarrow$	**A data segment with urgent data and acknowledgment**
010000	$\rightarrow$	**An ACK segment with or without data**
000010	$\rightarrow$	**A SYN segment**
011000	$\rightarrow$	**A data segment with push data and acknowledgment**
000001	$\rightarrow$	**A FIN segment**
010010	$\rightarrow$	**An ACK + SYN segment**
000100	$\rightarrow$	**An RST segment**

P3-33. Even with three letters exchanged between Alice and Bob, there is no guarantee that both know where and when they should meet. However, more and more communication raises the probability that both parties know about the meeting. Experts believe that three communications between the two parties are adequate assurance that they can come to the meeting. Let us go through each event:

- **a.** Alice cannot go to the meeting because she is not sure that Bob has received the letter. The letter may have been lost and Bob knows nothing about the meeting. This is similar to sending a SYN segment from the client to the server. The client (Alice) sets the scenario.

b. Bob cannot go to the meeting because he does know if Alice has received his confirmation. This is similar to the SYN + ACK. The server (Bob) confirms Alice’s request.

c. Alice cannot go to the meeting with total assurance that Bob will be there because she does not know if Bob has received her letter and knows that she knows that the meeting is confirmed. This is similar to the last ACK. The client (Alice) confirms that she has received the confirmation from the server (Bob).

P3-35.

a. The sequence number in the SYN segment is 2171. The SYN segment consumes one sequence number; the next sequence number to be used is 2172.

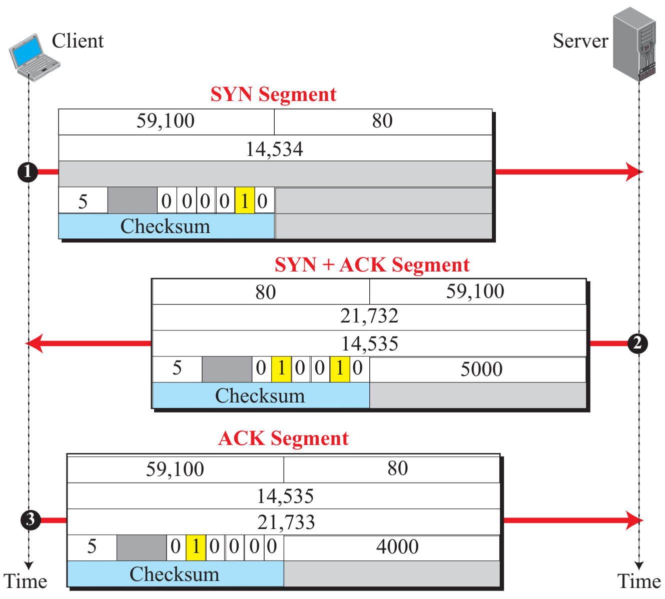
b. The sequence number in the data segment is 2172 (which represents the sequence number of the first byte). The bytes in the packets are numbered 2172 to 3171. Note that the client sends the data with the second packet (no separate ACK segment).

c. The sequence number in the FIN segment is 3172. Note that the FIN segment does consume a sequence number.

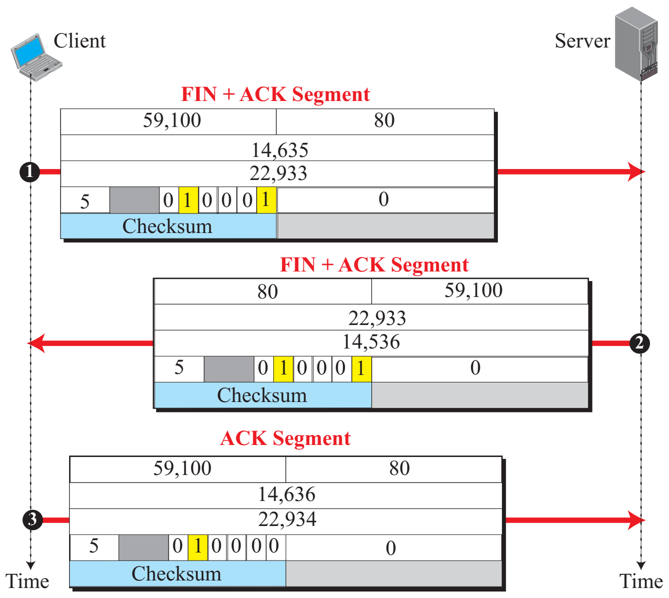
P3-37. The data section is only 16 bytes. The TCP header is 20 bytes. The efficiency is

$$(16) / (16 + 20) = 0.444 \rightarrow 44.4\%$$

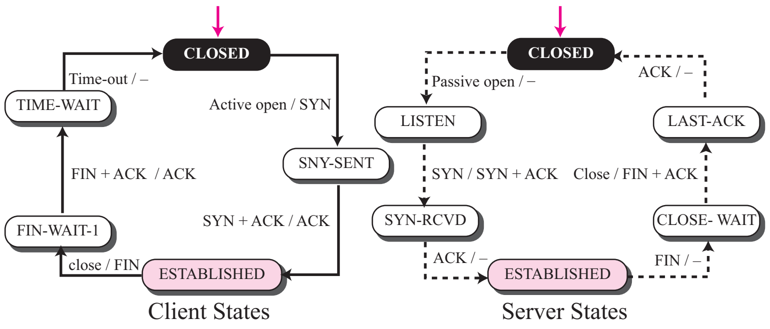
P3-39. The following shows the three segments exchanged during the connection establishment:



P3-41. The following shows the connection termination phase. We assume a three-handshake connection termination because the server has no more data to send.



P3-43. The following shows the new diagram.

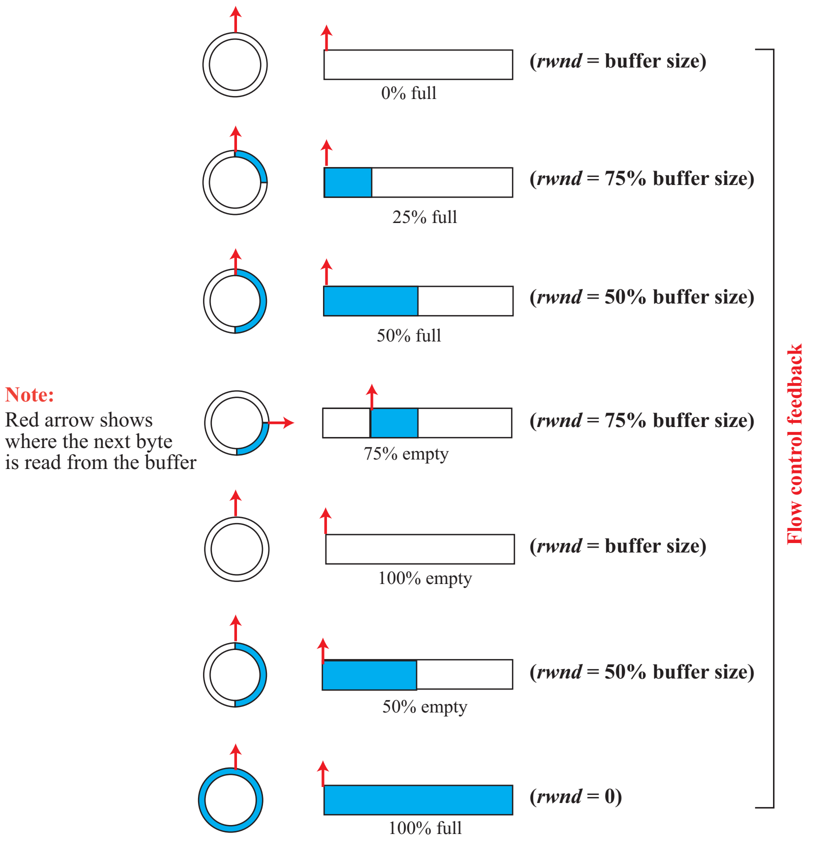


Note that the FIN-WAIT-2 state is not used in this case. Note also that there are changes in the server side that are not shown in Figure 3.51 (in the text) because this is somewhat of an implementation issue. In this case, the server sends FIN + ACK before going to the LAST-ACK state.

P3-45. Bob, the server, sends the response to Alice's IP address; the destination IP address is the source IP address in the request message. Since Alice has not requested this response, the response is dropped and lost. Eve can receive the response only if she can intercept the message.

P3-47. The probability of this mistake is very low because the initial sequence number (ISN) has a high probability of being unique. Assume Alice and Bob were using ISNs *x* and *y*, respectively in the previous connection, but *z* and *t* in this connection. The old ACK segment (third segment) has the acknowledgment number ($y + 1$); the new ACK segment should have the acknowledgment ($t + 1$). Bob's server immediately recognizes the problem and sends an RST segment to abort the connection. Alice then needs to start a new connection.

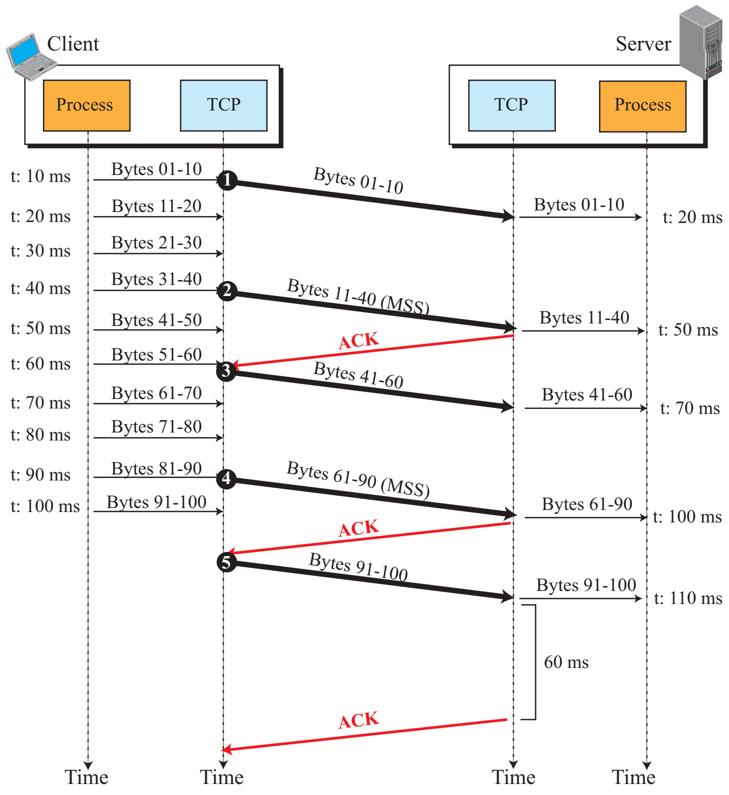
P3-49. The receiving TCP allocates a fixed-size buffer (the same size as the buffer allocated by the sending site).



(*rwnd* = buffer size)
0% full
(*rwnd* = 75% buffer size)
25% full
(*rwnd* = 50% buffer size)
50% full
Note:
Red arrow shows where the next byte is read from the buffer
(*rwnd* = 75% buffer size)
75% empty
(*rwnd* = buffer size)
100% empty
(*rwnd* = 50% buffer size)
50% empty
(*rwnd* = 0)
100% full
Flow control feedback

The application program at the receiver site pulls data from the buffer, which means there is no flow control from the receiving TCP toward the application program. Data received from the sending TCP are stored in the buffer until they are consumed by the application program. The part of the buffer that is still empty is advertised as the value of *rwnd* to the sending TCP (flow control). The following shows a simple example how the buffer status will change. We have shown both linear and circular representation of the buffer. The latter better shows the position of the data read by the application.

P3-51. The following shows the time line for each segment. Note that the situation is improved from the previous situation. Both of Nagle's rules are applied. Some segments are sent with the maximum segment size; others in response to an ACK. The improvement is because the receiver delays acknowledgments.

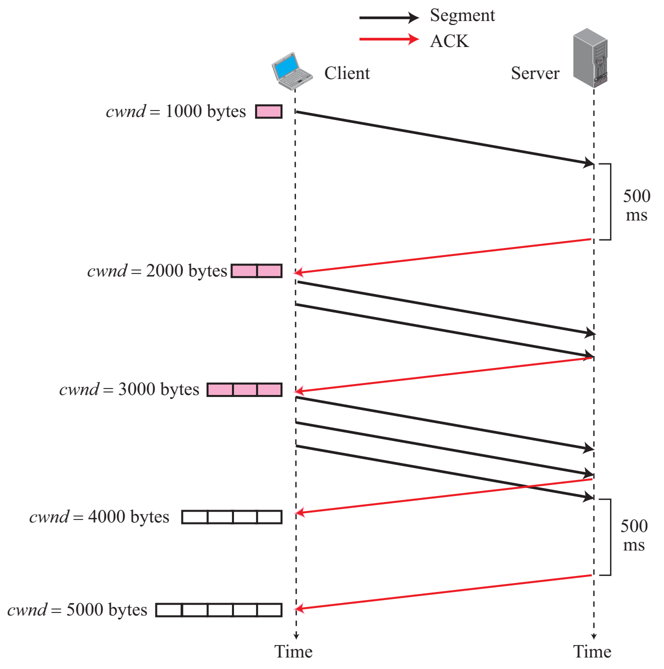


P3-53. We explain each case separately:

a. When a received segment has a sequence number greater than R_n, it means that the bytes are received out of order. TCP stores the bytes in the receive window, but it sends an ACK with the acknowledgment number equal to R_n to help *fast retransmission* of the missing bytes. This is a duplicate ACK because the receiver has already sent an ACK with the acknowledgment number equal to R_n. The issuing of duplicate ACKs is only a clue to the sender that some packets have arrived out of order. If three duplicate ACKs arrive, the sender deploys the *fast retransmission* and resends the packet with the sequence number defined by the acknowledgment.

b. When a duplicate segment arrives, the receiver TCP drops the packet and sends an ACK that defines the segment expected. This is also a duplicate ACK that gives a clue to the sender that its timer may have timed out prematurely. One might ask how the receiver could know whether the duplicate ACK is for an out-of-order segment or a duplicate segment. To trigger a fast retransmission, the sender needs to receive three duplicate ACKs (four ACKs with the same sequence number); a duplicate ACK per se does not trigger a fast retransmission.

P3-55. The data from the client process, 5400 bytes, can be divided into six chunks (five chunks of 980 bytes and one chunk of 500 bytes). After adding a header of 20 bytes, we have six segments (five segments of 1000 bytes and one segment of 520 bytes). The segments and the ACKs are created according to the rule we mentioned in the text. The size of the congestion window is increased by one MSS for each ACK received. If we follow the growth of the *cwnd*, we can see the pattern is exponential, but the base is decreased from 2 to 1.5 ($2^0 = 1$, $2^1 = 2$, $1.75^2 \approx 3$, $1.60^3 \approx 4$, and $1.5^4 \approx 5$).



P3-57. The following shows the events and the values. The units of windows and *ssthresh* are MSS. We use abbreviations for states such as slow start (SS), congestion avoidance (CA), and fast recovery (FR). The leftmost state shows the current state, the rightmost one shows the new state.

State	Event	*ssthresh*	*cwnd*	State
SS	ACK arrived	8	$5 + 1 = 6$	SS
SS	ACK arrived	8	$6 + 1 = 7$	SS
SS	ACK arrived	8	$7 + 1 = 8$	CA
CA	3 dup-ACKs	4	$4 + 3 = 7$	FR
FR	dup-ACK	4	$7 + 1/7 \approx 7.14$	FR
FR	dup-ACK	4	$7.14 + 1/(7.14) = 7.38$	FR
FR	ACK arrived	4	4	CA
CA	ACK arrived	4	$4 + 1/4 = 4.25$	CA
CA	Time-out	2.12	1	SS

P3-59. According to Karn's algorithm, we need to ignore the RTT for segment 1 in our calculation because it was timed-out and resent. Using only segment 2, the calculation is shown below:

$$\text{RTT}_M = 23 - 6 = 17 \text{ ms} \rightarrow \text{RTT}_S = (1 - 0.2)\ \text{RTT}_S + 0.2 \times \text{RTT}_M = 14.6 \text{ ms}$$