# CHAPTER 24

## PRACTICE SET

### Questions

**Q24-1.** The server needs to store the bytes in the buffer and sends an ACK with sequence number 2401. This reaction helps the client to purge the corresponding segment from its queue (Rule 5 in ACK generation).

**Q24-2.** 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.

**Q24-3.** A connection identifier in this case needs to include the identifier for two end points. In this case, a unique identifier for each end is defined by a socket address. The connection identifier should therefore be a pair of socket addresses: the source socket address and the destination socket address.

**Q24-4.** 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, $n = 0$.

**Q24-5.** The acknowledgment identifies the sequence number of the next segment that is expected to arrive.

**Q24-6.** The following shows the pair of socket addresses; they are used in reverse order in each direction:

| | Source | Destination |
|---|---|---|
| From client to server | 122.45.12.7<br>51000 | 200.112.45.90<br>161 |
| From server to client | 200.112.45.90<br>161 | 122.45.12.7<br>51000 |

**Q24-7.** 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.

**Q24-8.**

**a.** The use of checksum in UDP is optional. If the sender does not use the checksum, it fills the checksum field with sixteen 0s.

**b.** 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.

**Q24-9.** The TCP server has received a segment whose sequence number is higher than expected. The TCP server should store the out-of-order segment and send an ACK with acknowledgment number 2001 to lead to the generation of a duplicate acknowledgment and possibly the fast retransmission of the missing segment (Rule 4 in ACK generation).

**Q24-10.** 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.

**Q24-11.** The protocol field of the datagram 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.

**Q24-12.** 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.

**Q24-13.** In this case, Bob has made a *passive open* connection when he publicly announced his telephone number or privately gave his telephone number to Alice. Alice makes an *active open* when she dials Bob's telephone number.

**Q24-14.**

**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.

**Q24-15.** A segment that should be unambiguously acknowledged needs to consume a sequence number; otherwise, the segment and the next one have the same sequence number and it is not clear to which segment the acknowledgment belongs.

**a.** A SYN segment needs to be acknowledged unambiguously because it opens the connection.

**b.** A SYN + ACK is actually a combination of two segments; the first embedded segment, which is a SYN, needs to be acknowledged.

**c.** A FIN segment needs to be acknowledged unambiguously because it signals closing of a connection.

**d.** An ACK segment that carries no data is never acknowledged (no ACK for an ACK). It, therefore, does not consume a sequence number.

**Q24-16.** 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.

**Q24-17.** Figures 24.22 and 24.23 define the FSMs for unidirectional communication. The first rule of ACK generation is for bidirectional communication. It is the interaction between the sender and receiver TCP at each end. The rule should be implemented in the software.

**Q24-18.** 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.

**Q24-19.**

| | |
|---|---|
| general header | 12 bytes |
| COOKIE-ECHO chunk | 204 bytes |
| DATA chunk | 36 bytes |
| **Total** | **252 bytes** |

**Q24-20.** UDP or TCP, as defined in the TCP/IP suite, are two protocols (specifications), not pieces of software. Their software implementations can be changed to accommodate the services required by different APIs.

**Q24-21.** 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 establishment phase. In this case, if the client accepts the connection, it needs to send an ACK segment immediately or a data segment that acknowledges the SYN + ACK segment. Otherwise, it needs to send an RST segment and abort the connection.

**Q24-22.** 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.

**Q24-23.** We find the sequence number of the second segment in each case:

**a.** The sequence number of the second segment is 101 + 0 = 101.

**b.** The sequence number of the second segment is 101 + 10 = 111.

**Q24-24.** 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.

**Q24-25.** A segment carrying an RST flag can close the communication in both directions immediately. The segment carrying an RST flag should not even be acknowledged.

**Q24-26.** 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.

**Q24-27.** The client needs to store the new bytes, but delay the acknowledgment until receiving the next segment or wait a period of 500 ms. This reduces the number of ACKs in the network. (Rule 2 in ACK generation).

**Q24-28.** 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.

**Q24-29.**

| | |
|---|---|
| general header | 12 bytes |
| DATA chunk #1 header | 16 bytes |
| DATA chunk #1 | 22 bytes |
| padding | 2 bytes |
| DATA chunk #2 header | 16 bytes |
| DATA chunk #2 | 22 bytes |
| padding | 16 bytes |
| **Total** | **92 bytes** |

**Q24-30.** 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).

**Q24-31.** A SYN segment opens the connection in only one direction. For a communication using TCP, two SYN segments are needed, one for each direction.

**Q24-32.** 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).

**Q24-33.** A TCP segment or a combination of two or more segments can perform one of the following tasks:

**a.** Establishing a connection

**b.** Terminating a connection

**c.** Transferring data

**d.** Acknowledging the receipt of a segment

**e.** Advertising the window size (*rwnd*)

**Q24-34.** 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).

**Q24-35.** The window size field of the TCP header is only 16 bits. A number with 16 bits can define a decimal number between 0 and 65,535.

**Q24-36.**

**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.

**Q24-37.** UDP is preferred because each user datagram can be used for each chunk of data.

**Q24-38.**

| | |
|---|---|
| general header | 12 bytes |
| SACK header | 16 bytes |
| out-of-order reporting $(4 \times 3)$ | 12 bytes |
| duplicate data reporting | 20 bytes |
| **Total** | **60 bytes** |

**Q24-39.** The server needs to immediately send a segment with the RST flag set to show that the connection cannot be established. As we see in Chapter 19, if the network layer knows about the availability, it can also send an ICMP packet.

**Q24-40.**

| | |
|---|---|
| general header | 12 bytes |
| COOKIE-ACK chunk | 4 bytes |
| DATA chunk | 36 bytes |
| **Total** | **52 bytes** |

**Q24-41.** The SYN segment cannot carry data. The SYN + ACK segment cannot carry data either, but this segment is actually the SYN segment with an additional ACK bit. Although some people think that the FIN segment may not carry data, it actually can. The client or the server can send the last bytes of data in a segment and set the FIN bit.

**Q24-42.** 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.

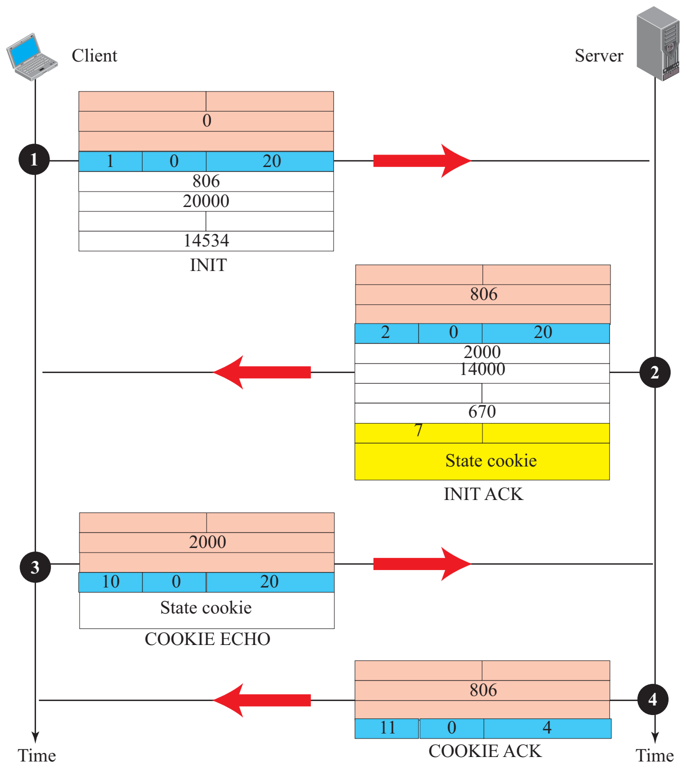
**Q24-43.** The SACK chunk with a cumTSN of 23 was delayed.

## Problems

**P24-1.** The data section is only 16 bytes. The TCP header is 20 bytes. The efficiency is

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

**P24-2.** See the following figure. We have filled the fields with available information. Each packet has the general header and the appropriate control chunk.



**P24-3.**

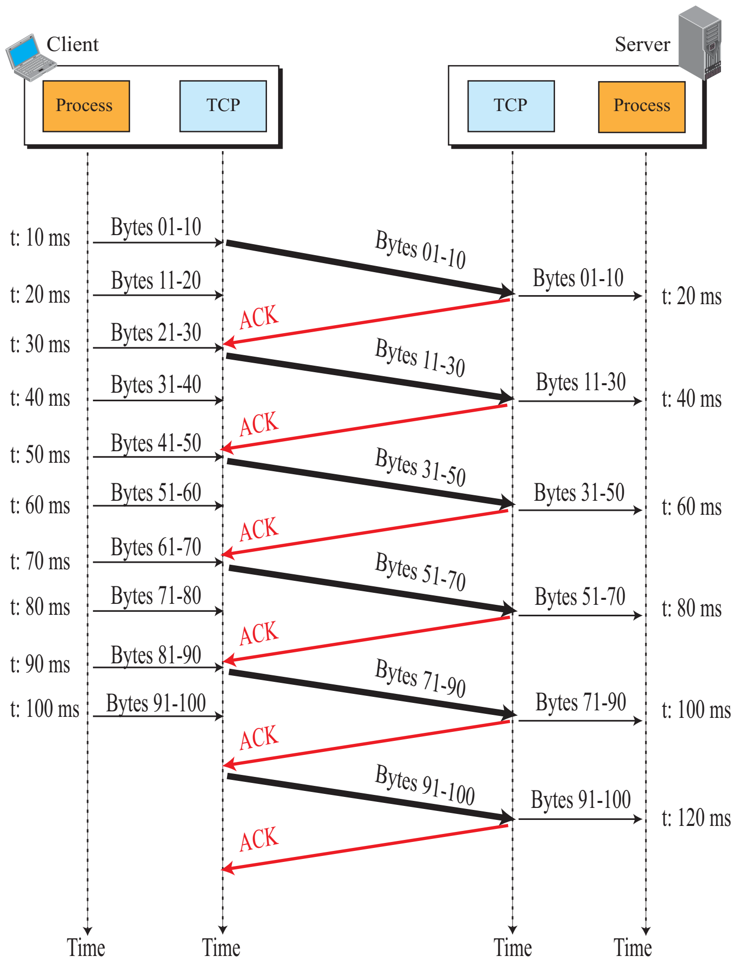
**a.** The minimum size is 8 bytes (header without payload).

**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 Chapter 19), 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).

**P24-4.** The following figure shows the time line for each segment. Note that a mild version of the silly window syndrome actually exists here. We have six segments of which two each carry 10 bytes of data and the other four each carry 20 bytes of data. No segment carries the MSS. The reason is that acknowledgments are not delayed by the receiver: each segment is acknowledged immediately.



**P24-5.** 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:

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

The following shows the calculation:

$$\text{RTT}_D = (1 - \beta) \times \text{RTT}_D + \beta \times |\text{RTT}_S - \text{RTT}_M|$$
$$\text{RTT}_D = (1 - 0.25) \times 7 + 0.25 \times |17 - 20| = 6 \text{ ms}$$

**P24-6.** We separately show which aspects are similar to Go-Back-*N* and which ones are similar to Selective-Repeat.

**a.** TCP is similar to the Go-Back-*N* protocol when we consider the cumulative acknowledgment. The cumulative acknowledgment used in TCP is similar to the Go-Back-*N* protocol. The acknowledgment field of a TCP segment can hold only one 32-bit number. This can be the cumulative acknowledgment of all bytes contained in the segment.

**b.** TCP is similar to the Selective-Repeat protocol in that the TCP receiver can store the out-of-order packets received; they are not discarded, but not delivered out of order to the application layer either. TCP is also similar to the Selective-Repeat protocol in that the size of the send and receive windows is the same if we ignore the effect of congestion and the congestion window.

**P24-7.** 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.

**P24-8.**

**a.** ordered, because the U flag is not set

**b.** B = **0**, E = **0** means this is a middle fragment

**c.** length is 21, so 3 bytes of padding are needed

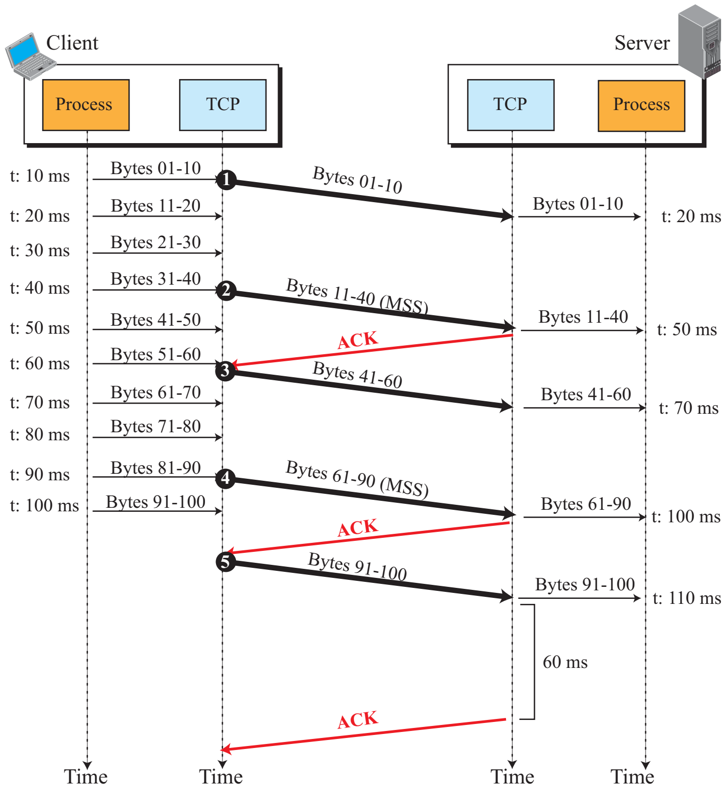
**d.** TSN = 5

**e.** SI = 3

**f.** SSN = 10

**g.** message is 48656C6C67

**P24-9.** The following figure 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.



**P24-10.** The probability of this mistake is very low. In TCP, an ISN (Initial Sequence Number) for a connection always has a high probability of being unique. When Alice sends the new SYN, its sequence number is different from the sequence number of the wandering SYN. Assume the sequence number of the old SYN is $x$ and the sequence number of the new SYN is $y$. The SYN + ACK segment sent by Bob in response to the old SYN has the acknowledgment number $(x + 1)$; Alice is expecting the acknowledgment number of $(y + 1)$ for the new SYN. Alice immediately sends an RST (reset) segment and aborts the connection. Alice then needs to start a new connection.

**P24-11.** 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.

**P24-12.** If Bob, the server, always uses a different ISN (Initial Sequence Number), Eve cannot create such a connection easily. Let us go through the three hand-shake events:

**a.** Eve sends a SYN segment using Alice's IP address as the source IP address.

**b.** Bob (the server) sends the SYN + ACK segment to Alice (not to Eve) because the destination address is Alice's address.

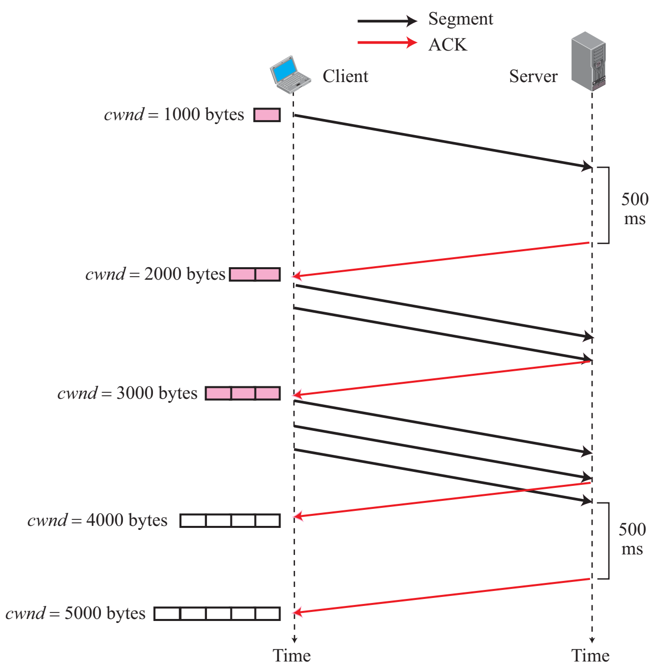
**c.** Since Alice has not initiated this connection, the third segment (ACK segment) will not be generated. If Bob has used a randomly generated sequence number (ISN), Eve cannot send the ACK segment because the acknowledgment number should be 1 plus the ISN used by Bob unless Eve intercepts the SYN + ACK packet and finds the ISN used by Bob, which is another issue. However, if Bob always uses the same ISN, then Eve can create the ACK message.

**P24-13.** 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.

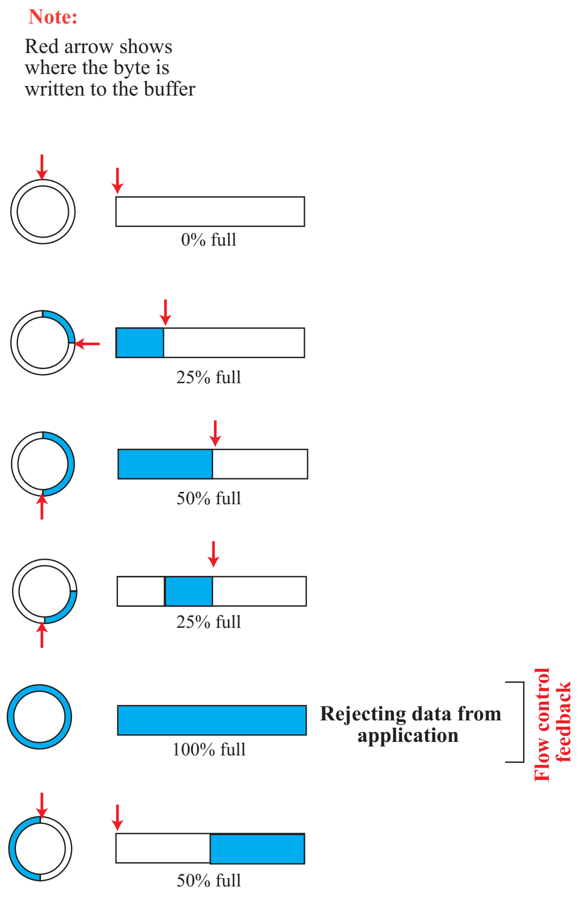
| State | Event | ssthresh | cwnd | State |
|---|---|---|---|---|
| SS | ACK arrived | 8 | $5 + 1 = 6$ | SS |
| SS | ACK arrived | 8 | $6 + 1 = 7$ | SS |
| SS | ACK arrived | 8 | $7 + 1 = 8$ | CA |
| CA | 3 dup-ACKs | 4 | $4 + 3 = 7$ | FR |
| FR | dup-ACK | 4 | $7 + 1/7 \approx 7.14$ | FR |
| FR | dup-ACK | 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 |

**P24-14.** A time-out event means no news. A three-duplicate-ACKs event means one item of bad news (the expected packet has not arrived) in the past followed by three items of good news recently (three other packet arrived out of order). The time-out event can mean a strong congestion; the three-duplicate ACKs can mean a slight congestion from which the network has recovered. 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 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.

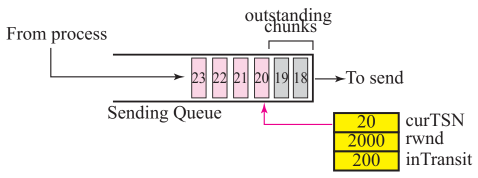
**P24-15.** The data from the client process, 5400 bytes, can be divided into six chunks (five chunks of 980 bytes and one chunk of 500 bytes). After adding a header of 20 bytes, we have six segments (five segments of 1000 bytes and one segment of 520 bytes). The segments and the ACKs are created according 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 $1.5^4 \approx 5$).



**P24-16.** The sending TCP allocates a fixed-size buffer, although the size of the sender window is changed continuously, controlled by the receiver window and the congestion window. The buffer is filled by the bytes written by the application and emptied by the bytes acknowledged by the sender. When the buffer is not full, TCP accepts data from the application; when the buffer is full, TCP rejects data from the application program. The following figure shows a simple example of 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 written by the application.


Note:
Red arrow shows where the byte is written to the buffer
0% full
25% full
50% full
25% full
100% full
Rejecting data from application
Flow control feedback
50% full

**P24-17.** See the following figure. Chunks 18 and 19 are sent but not acknowledged (200 bytes of data). 18 DATA chunks (1800 bytes) can be sent, but only 4 chunks are in the queue. Chunk 20 is the next chunk to be sent.



**P24-18.** Every second the counter is incremented by 64,000 × 2 = 128,000. The sequence number field is 32 bits long and can hold only $2^{32}$– 1. So it takes $(2^{32}–1)/(128,000)$ or 33,554 seconds (almost 9 hours and 19 minutes) to wrap around.

**P24-19.**

**a.** The source port number is the first 16 bits or $(0045)_{16}$ = 69.

**b.** The destination port number is the second 16 bits $(\text{DF00})_{16}$ = 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.

**P24-20.** The following table shows the comparison:

| Fields | UDP | TCP | Purpose |
|---|---|---|---|
| Source Port Number | √ | √ | Source port number |
| Destination Port Number | √ | √ | Destination port number |
| Checksum | √ | √ | For error detection |
| Total Length | √ | ▨ | Unnecessary even in UDP |
| Sequence Number | ▨ | √ | For flow and error control |
| Acknowledgment Number | ▨ | √ | For flow and error control |
| Header Length | ▨ | √ | To define variable header length |
| Control Bits | ▨ | √ | To define types of segments |
| Urgent Pointer | ▨ | √ | To define the end of urgent data |
| Options and Padding | ▨ | √ | To allow using different options |

Note that the only field that exists in UDP but is missing in TCP is the *total length* field. The designer of TCP did not feel that this field was needed because the size of the TCP segment can be determined from the size of the IP datagram that carries it.

**P24-21.**

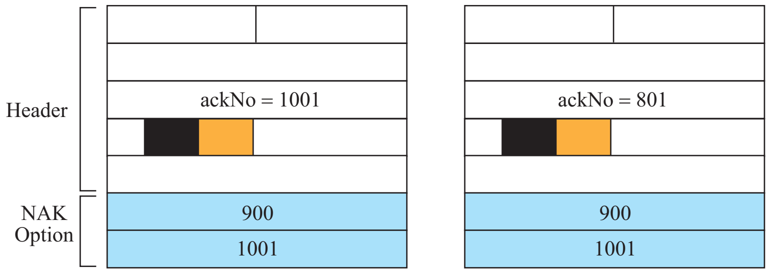
**a.** 0432 in hex is 1074 in decimal. The source port is 1074.

**b.** 0017 in hex is 23 in decimal. The destination port is 23.

**c.** The verification tag is 1.

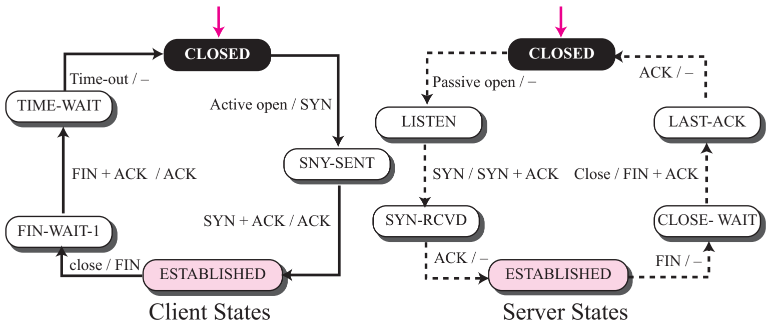
**d.** The checksum is 0.

**P24-22.** An 8-byte option can show a range of bytes. The first 4 bytes can show the beginning of the range; the second 4 bytes can show the end of the range. Combined with the acknowledgment number, they can show that the range is a duplicate or out of order. If the second sequence number in the option is the same as the acknowledgment number, the block is a duplicate block. If the second sequence number in the option is different from the acknowledgment number, the block is an out-of-order block. The following figure shows the two cases:



a. Duplicate bytes 900 to 1000 b. Out-of-order bytes 900 to 1000

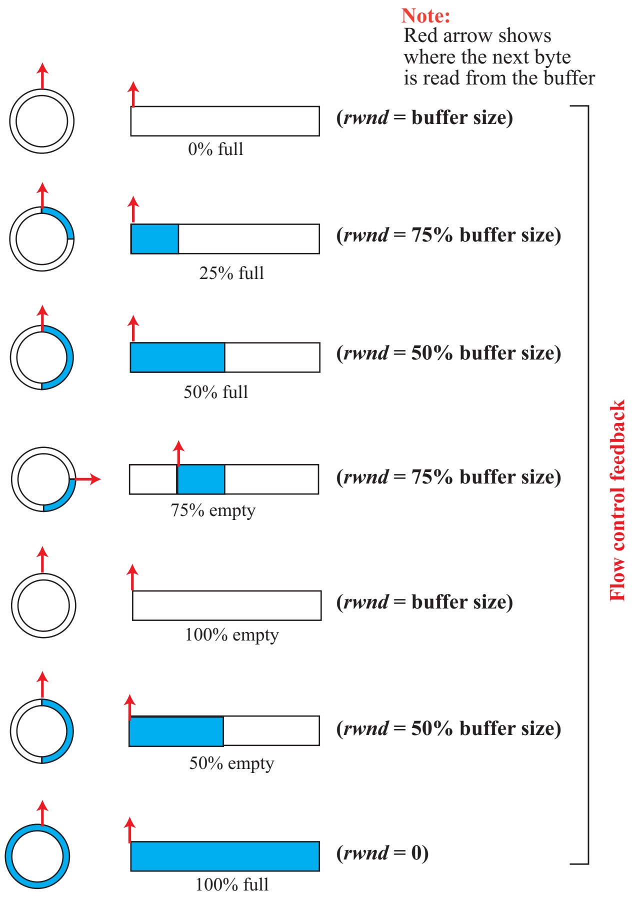
**P24-23.** The following figure shows the new diagram.



**P24-24.** The throughput is the average of window sizes divided by the round-trip time. In this case, the average window size is 45,000 bytes or 360,000 bits. This number of bits can be transmitted in one round-trip time.

throughput = 360,000 bits / 30 ms = 12,000,000 bps = 12 Mbps

**P24-25.** The receiving TCP allocates a fixed-size buffer (the same size as the buffer allocated by the sending site) as shown in the following figure.



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 figure 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.

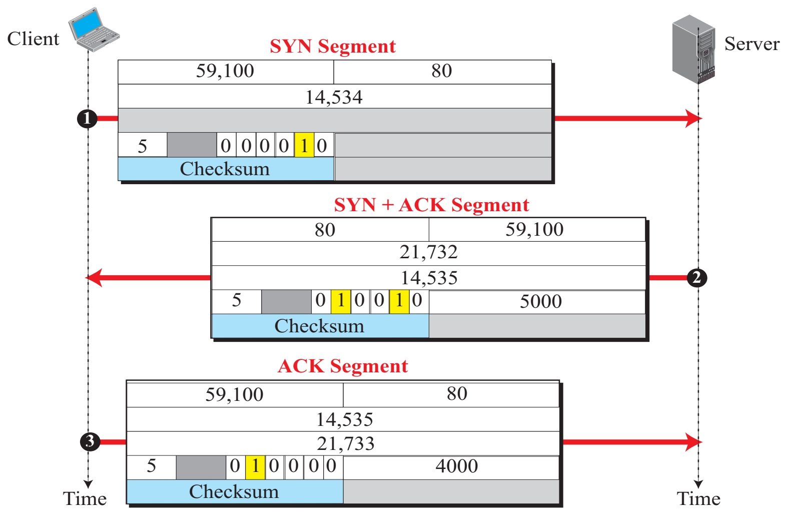
**P24-26.** The following shows the events and the values. The units of windows and *ssthresh* are MSS. We use abbreviations for states such as slow start (SS) and congestion avoidance (CA). The leftmost state shows the current state, the rightmost one shows the new state.

| State | Event | ssthresh | cwnd | State |
|---|---|---|---|---|
| SS | ACK arrived | 6 | 4 + 1 = 5 | SS |
| SS | ACK arrived | 6 | 5 + 1 = 6 | CA |
| CA | ACK arrived | 6 | 6 + 1/6 ≈ 6.17 | CA |
| CA | ACK arrived | 6 | 6.17 + 1/(6.17) ≈ 6.33 | CA |
| CA | Time-out | 3 | 1 | SS |
| SS | ACK arrived | 3 | 1 + 1 = 2 | SS |
| SS | ACK arrived | 3 | 2 + 1 = 3 | CA |
| CA | ACK arrived | 3 | 3 + 1/3 ≈ 3.33 | CA |

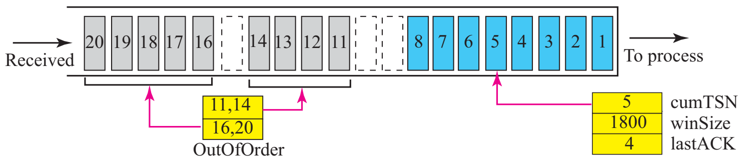
**P24-27.** The probability of this mistake is very low because the initial sequence number (ISN) has a high probability of being unique. Assume Alice and Bob were using ISNs *x* and *y*, respectively in the previous connection, but *z* and *t* in this connection. The old ACK segment (third segment) 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.

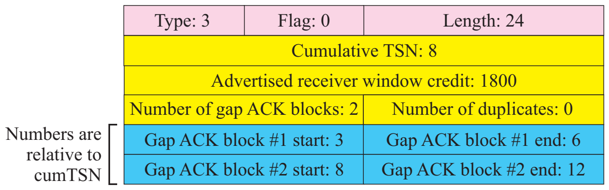
**P24-28.** The host sends a SHUTDOWN ACK and goes to the **SHUTDOWN-ACK-SENT** state.

**P24-29.** The following shows the three segments exchanged:



**P24-30.** See the following figure. Note that the value of cumTSN must be updated to 8.





| Type: 3 | Flag: 0 | Length: 24 |
|---|---|---|
| Cumulative TSN: 8 | | |
| Advertised receiver window credit: 1800 | | |
| Number of gap ACK blocks: 2 | Number of duplicates: 0 | |
| Gap ACK block #1 start: 3 | Gap ACK block #1 end: 6 | |
| Gap ACK block #2 start: 8 | Gap ACK block #2 end: 12 | |

Numbers are relative to cumTSN

The above figure also shows the contents of the SACK chunk after updating the cumTSN.

**P24-31.** The following are eight out of 64 possible combinations that are normally used:

| | | |
|---|---|---|
| 000000 | → | A data segment with no acknowledgment |
| 110000 | → | A data segment with urgent data and acknowledgment |
| 010000 | → | An ACK segment with or without data |
| 000010 | → | A SYN segment |
| 011000 | → | A data segment with push data and acknowledgment |
| 000001 | → | A FIN segment |
| 010010 | → | An ACK + SYN segment |
| 000100 | → | An RST segment |

**P24-32.**

**a.** The source port number is $(E293)_{16}$ or 58,003 in decimal.

**b.** The destination port number is $(0017)_{16}$ or 23 in decimal.

**c.** The sequence number is $(00000001)_{16}$ or 1 in decimal.

**d.** The acknowledgment number is $(00000000)_{16}$ or 0 in decimal.

**e.** The HLEN = 5. The header is $5 \times 4 = 20$ bytes long.

**f.** The combination of the reserved field and the control field is $(002)_{16}$ or $(000000000010)_2$. The rightmost 6 bits are 000010, which means only the SYN bit is set. This is the SYN segment used for connection establishment.

**g.** The window size field is $(07FF)_{16}$ or 2047 in decimal. The window size is 2047 bytes.

**P24-33.** 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.

**P24-34.** The data is 16 bytes; the total UDP user datagram size is $16 + 8 = 24$ bytes.

$$\text{Efficiency} = (16) / (24) = 0.666 = 66.6 \text{ percent.}$$

**P24-35.**

**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.

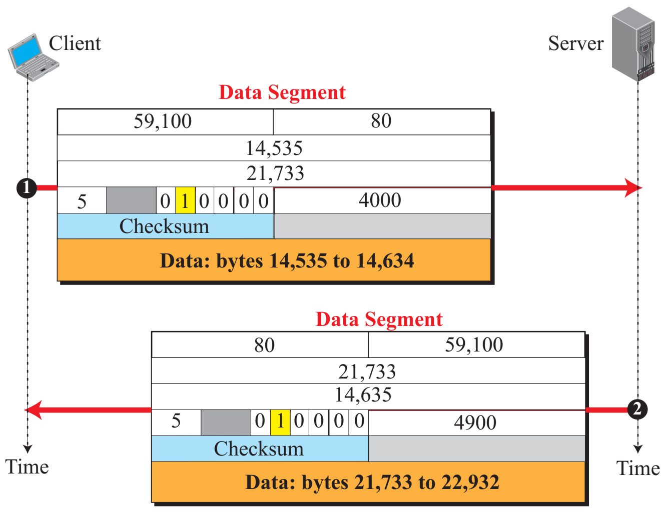
**b.** The sequence number in the data segment is 2172 (which represents the sequence number of the first byte). The bytes in the packets are numbered 2172 to 3171. Note that the client sends the data with the second packet (no separate ACK segment).

**c.** The sequence number in the FIN segment is 3172. Note that the FIN segment does consume a sequence number, but it needs a sequence number to be acknowledged.

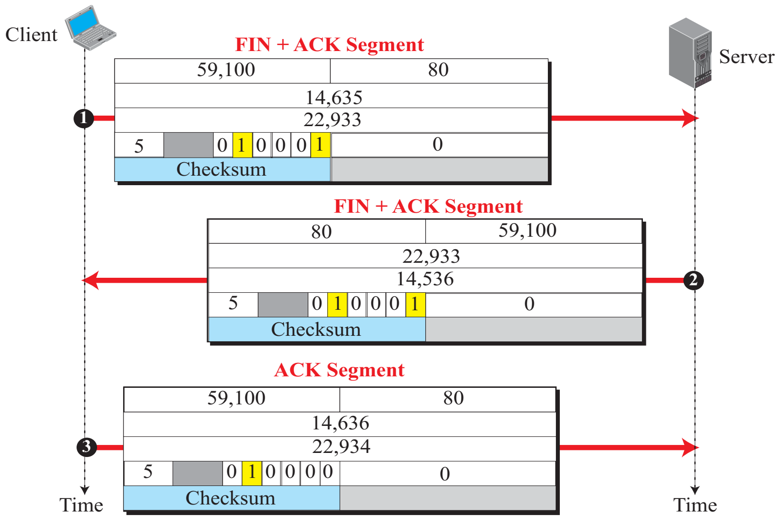
**P24-36.** The largest number in the sequence number field is $2^{32} - 1$. If we start at 7000, it takes $[(2^{32} - 1) - 7000] / 1{,}000{,}000 = 4295$ seconds or almost 72 minutes.

**P24-37.** It sends an INIT_ACK chunk.

**P24-38.** The following figure shows the segments exchanged during the data-transfer phase:



**P24-39.** The following figure shows the connection termination phase. We assume a three-handshake connection termination because the server has no more data to send.



**P24-40.** The window size is min(3000, 5000) = 3000. Since 2000 bytes is already sent, only 3000 – 2000 = 1000 more bytes can be sent.

**P24-41.** 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).

**P24-42.**

**a.** None of the control bits are set. The segment is part of a data transmission without a piggybacked acknowledgment.

**b.** The FIN bit is set. This is a FIN segment request to terminate the connection.

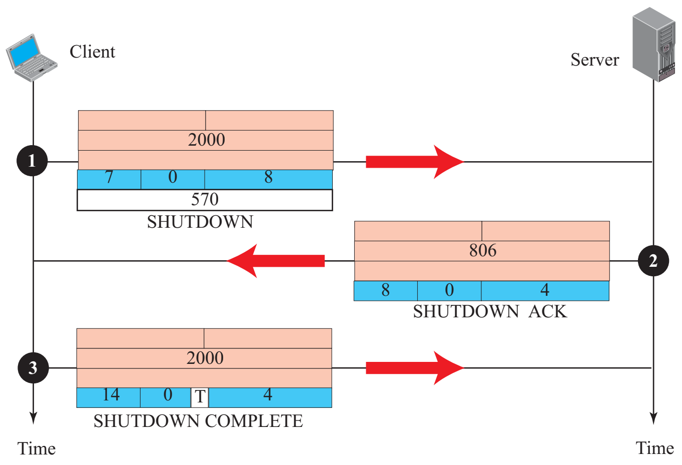
**c.** The ACK and the FIN bits are set. This is a FIN + ACK segment.

**d.** The RST bit is set. This is a reset segment (request for resetting).

**e.** The SYN bit is set. This is a SYN segment.

**f.** The ACK and the SYN bits are set. This is a SYN + ACK segment.

**P24-43.** See the following figure. We have filled the fields with available information. Each packet has the general header and the appropriate control chunk. Note that only the SHUTDOWN chunk has the cumTSN ACK, which acknowledges the receipt of the last packet.


Client
Server
1
2000
7
0
8
570
SHUTDOWN
2
806
8
0
4
SHUTDOWN ACK
3
2000
14
0
T
4
SHUTDOWN COMPLETE
Time
Time