CHAPTER 24

PRACTICE SET

Questions

Q24-1. 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.

Q24-3. UDP is preferred because each user datagram can be used for each chunk of data.

Q24-5. 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.

Q24-7. 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.

Q24-9. 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.

Q24-11. 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.

Q24-13. 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.

Q24-15. A SYN segment opens the connection in only one direction. For a communication using TCP, two SYN segments are needed, one for each direction.

Q24-17. 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.

Q24-19. 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.

Q24-21. 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.

Q24-23. 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.

Q24-25. 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*)

Q24-27. The acknowledgment identifies the sequence number of the next segment that is expected to arrive.

Q24-29. 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).

Q24-31. 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).

Q24-33. 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).

Q24-35. 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.

Q24-37. 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.

Q24-39.

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**

Q24-41.

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

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

Problems

P24-1.

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).

P24-3.

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.

P24-5. 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.

P24-7. 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

P24-9. 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).

P24-11.

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.

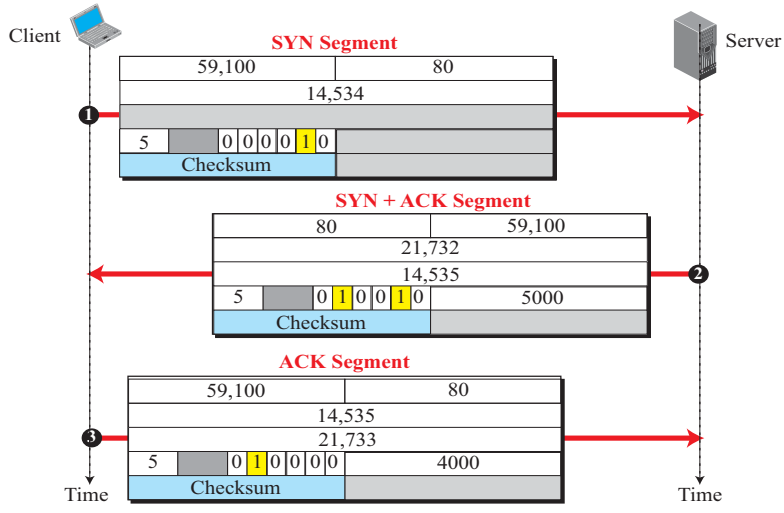
b. The sequence number in the data segment is 2172 (which represents the sequence number of the first byte). The bytes in the packets are numbered 2172 to 3171. Note that the client sends the data with the second packet (no separate ACK segment).

a. 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.

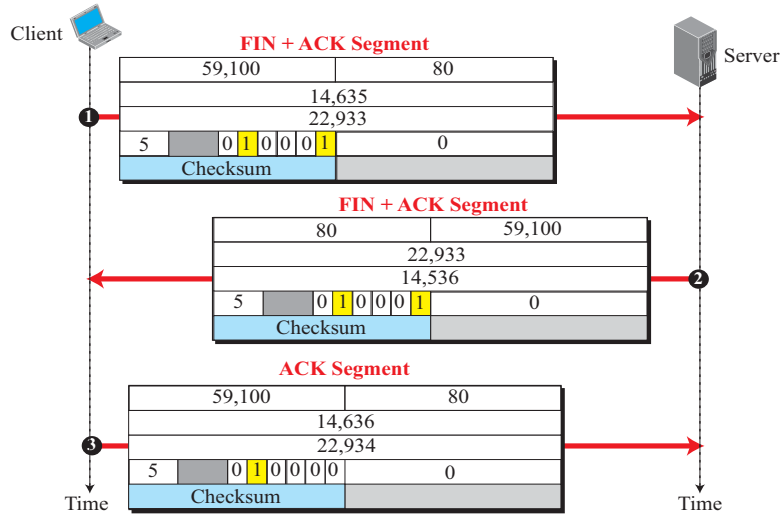
P24-13. The data section is only 16 bytes. The TCP header is 20 bytes. The efficiency is

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

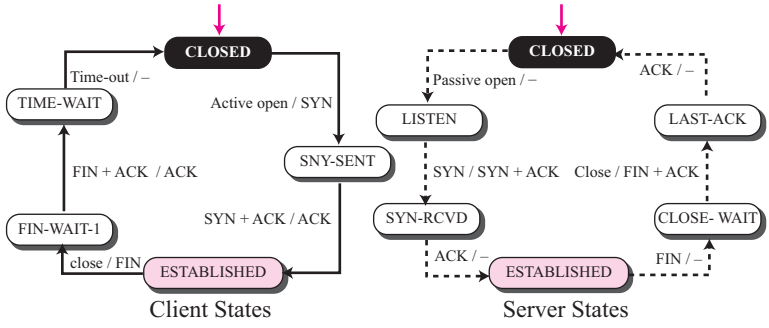
P24-15. The following shows the three segments exchanged:



P24-17. The following figure shows the connection termination phase. We assume a three-handshake connection termination because the server has no more data to send.



P24-19. The following figure shows the new diagram.



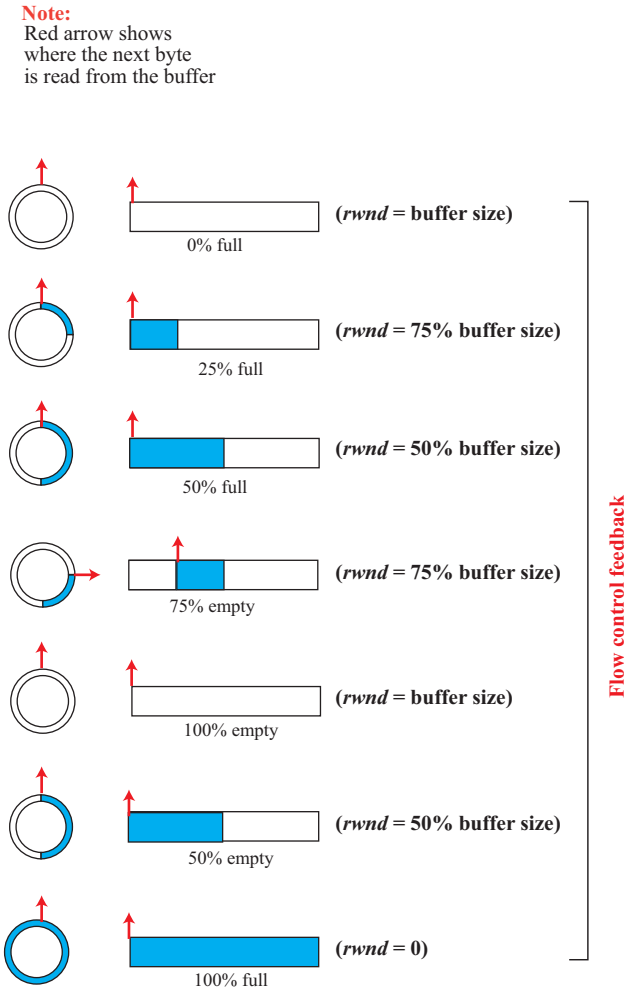
Note that the FIN-WAIT-2 state is not used in this case. Note also that there are changes in the server side that are not shown in the text because this is somewhat of an implementation issue. In this case, the server sends FIN + ACK before going to the LAST-ACK state.

P24-21. 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.

P24-23. 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.

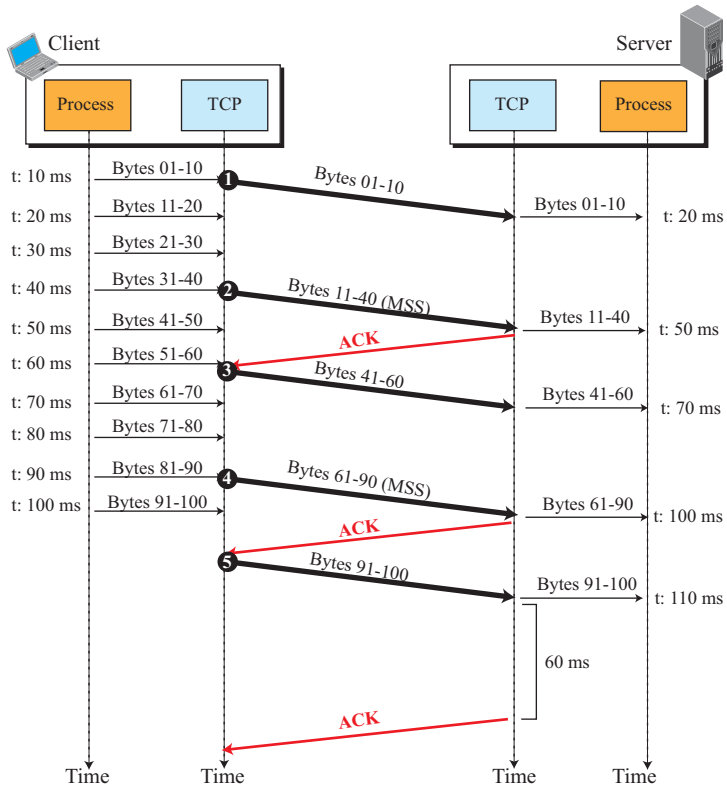
P24-25. The receiving TCP allocates a fixed-size buffer (the same size as the buffer allocated by the sending site). The application program at the receiver site pulls data from the buffer, which means there is no flow control from the receiving TCP toward the application program. Data received from 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.



P24-27. 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.



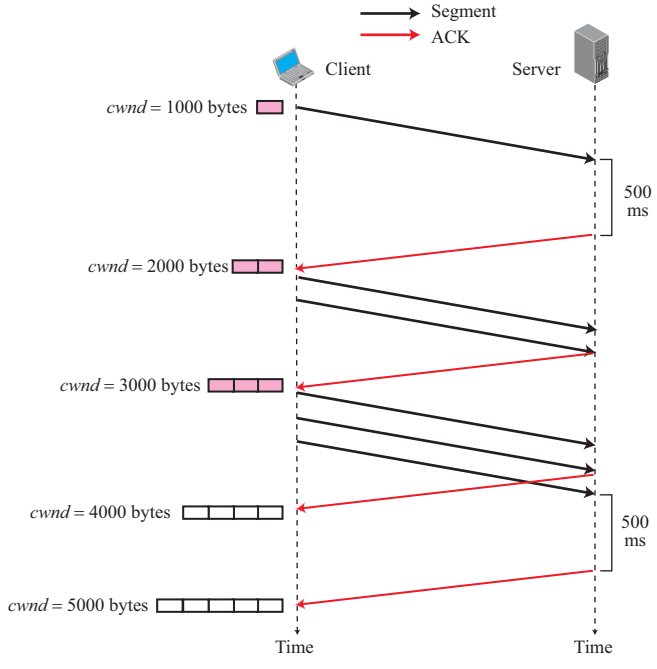
P24-29. We explain each case separately:

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 dupli-

cate 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.

P24-31. 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 $1.5^4 \approx 5$).



P24-33. 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

P24-35. 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:

$$RTT_M = 23 - 6 = 17 \text{ ms}$$
$$RTT_S = (1 - 0.2)\, RTT_S + 0.2 \times RTT_M = 14.6 \text{ ms}$$

The following shows the calculation:

$$RTT_D = (1 - \beta) \times RTT_D + \beta \times |RTT_S - RTT_M|$$
$$RTT_D = (1 - 0.25) \times 7 + 0.25 \times |17 - 20| = 6 \text{ ms}$$

P24-37. It sends an INIT_ACK chunk.

P24-39.

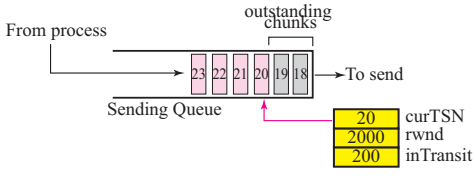
a. 0432 in hex is 1074 in decimal. The source port is 1074.

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

c. The verification tag is 1.

d. The checksum is 0.

P24-41. 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.



P24-43. 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.