CHAPTER 8

PRACTICE SET

Questions

Q8-1. In dictionary coding, the number of iterations in the loop is one less than the number of characters in the message. In this case, the number of iterations is 59.

Q8-3. In a Huffman tree, the number of leaves is the same as the number of symbols. Therefore, we have 20 leaves in the tree.

Q8-5. The encoding table can be the following:

A $\rightarrow$ **00** **B** $\rightarrow$ **01** **C** $\rightarrow$ **10** **D** $\rightarrow$ **11**

Huffman coding cannot help here because we could have encoded each character as two bits. Huffman coding will help if the occurrence probabilities of the characters in the messages are different and we can send fewer bits for characters with high probability and more bits for characters with low probability.

Q8-7. ADM is a variation of DM in which the value of Δ may change from one step to another.

Q8-9. The answer is given for each case for both PCM and DM.

a. PCM: 4 (because $2^4 = 16$) and DM: 1

b. PCM: 5 (because $2^5 = 32$) and DM: 1

c. PCM: 6 (because $2^6 = 64$) and DM: 1

Q8-11. In DM, only the immediate previous y_n value is used in the calculation of e_n. In DPCM, weighted averages of several previous y_n values are used in the calculation of e_n.

Q8-13. The answer is no. The value of the T matrix entries, for each value of N, is fixed. The receiver either has the matrix or can generate it.

Q8-15. The value of Q(m, n) becomes larger and larger when we move toward the higher frequencies in the matrix, from M(0, 0) to M(N–1, N–1) diagonally. This is done to reduce the effect of higher frequencies. Human eyes cannot recognize the higher frequencies when their amplitude (strength) is outside of an upper and a lower threshold. The values in the Q(m, n) matrix are selected based on this property. Dividing the amplitude of higher frequencies by a larger number makes these amplitudes smaller and smaller, finally moving them toward zero values and elimination (better compression).

Q8-17. The whole idea is to change the real numbers (values with fractions) to integer values to reduce the number of transmitted bits. The real value 14.125 or $(1110.001)_2$ needs seven bits to send; the integer value 14 or $(1110)_2$ needs only four bits to send. The number of bits to send tremendously increases if we want to use a high level of precision.

Q8-19. The answer is no. Multimedia data encoded in JPEG cannot be recovered using GIF and vice versa. If two entities want to communicate, they must use the same encoding and decoding scheme.

Q8-21. In the second approach, the whole multimedia file is downloaded using HTTP as the application-layer protocol. In the third approach, HTTP is only responsible for downloading the metafile; the actual multimedia file is normally downloaded using a specific application program that may use the service of UDP/RTP.

Q8-23. In live audio/video, the multimedia data is being sent on the Internet no matter whether we use it or not. In real-time interactive audio/video, two parties start exchanging multimedia when they both agree to do so.

Q8-25. The answer is yes. The protocol defines a server as a process running at the application level. The two servers can be on the same machine or different machines.

Q8-27. The answer is yes because the distance between each codeword and the others is three or more. According to the Hamming distance principle, the code can correct one single-bit error ($d_{min} = 2t + 1$).

Q8-29. In this scheme each packet is independent. The scheme still works, but more packets lose their high-level resolution.

Q8-31. One of the main tasks of a transport-layer protocol, such as UDP or TCP, is to provide many-to-many end-to-end communication between application-layer processes. In UDP and TCP, this task is done using source and destination socket addresses. RTP lacks such a capability because it does not define socket addresses. RTP assumes that the underlying protocol, UDP, provides the multiplexing and demultiplexing; RTP is assigned other tasks specific to multimedia applications.

Q8-33. A real-time interactive multimedia application needs a few services that are not provided by UDP, such as timestamping, sequence numbers, synchronization of audio and video, and defining the payload type. If a new application can provide these services and somehow create a session with the receiver, it can use the service of UDP and ignore the services provided by RTP. The RTP protocol has been devised to remove the burden of creating these services for each the application-layer protocol.

Q8-35. Encoding and decoding of the multimedia data is done by the application program. RTP only carries information in the payload that defines the type of encoding performed on the payload. The receiver needs to use the same decoding to obtain the original data.

Q8-37. The two applications at the two ends, the sender and the receiver, need to first create a session using other protocols such as SIP before using the service of RTP. During the session establishment, the two ends agree on two port numbers that need to be used for connection. As long as the session is not torn down, the UDP packets carrying the same port numbers belong to the same session.

Q8-39. RTP plus UDP provides about the same services as TCP without the latter's retransmission policy. However, the combination of RTP plus UDP is more suitable for multimedia transmission because it can provide boundaries of chunks, encoding type, and timestamping. In addition, UDP does not resend the lost packets, so it is appropriate for real-time multimedia applications.

Q8-41. The answer is negative. SIP is an application-layer protocol that is designed to directly use the services of UDP, TCP, or even STCP. It listens to the registered port 5060.

Q8-43. SIP is a signaling protocol that initiates, terminates, and modifies a session between two parties. It doesn't care about the actual data exchange. Audio, video, or other multimedia data can be transmitted. One drawback could be the low level of efficiency.

Q8-45. The sender can handle the situation more easily in a unicast session because there is only one receiver. In a multicast situation, the feedback from multiple receivers can be different and the sender needs to use a strategy to analyze the feedback and respond to the situation.

Q8-47. SIP can provide all of these services:

a. The INVITE message in SIP provides several pieces of information about the caller. The SIP interface software can be set to display all or some of these pieces before the callee can answer the call.

b. The SIP interface software can be set to show all pieces of information in a new call when the user is busy handling the current call.

c. SIP is designed to provide multiparty and multicasting service.

Q8-49. SIP just sets up and terminates a session between two parties. H.323 is a protocol suite that covers everything pertaining to the session. SIP is more flexible in its address formats. H.323 setup is more complicated than that of SIP.

Q8-51. In this case, we have two queues, but we also have two separate servers. This is not an example of priority queuing; it is an example of FIFO queuing with two independent queues.

Q8-53. This is an example of priority queuing. The business customers have priority over the regular customers.

Problems

P8-1. We replace each run of the same character with a count and the symbol as shown below.

3A6C1B4C5D4A3B

The compression rate is 26/14 or almost 1.86.

P8-3. Let us show how the dictionary for each case is made using the knowledge about the previous case.

a. Dictionary is 0 → A; the code is 0.

b. Dictionary is 0 → A and 1 → AA; the code is 00.

c. Dictionary is 0 → A and 1 → AA; the code is 01.

d. Dictionary is 0 → A, 1 → AA, and 2 → AAA; the code is 010.

e. Dictionary is 0 → A, 1 → AA, and 2 → AAA; the code is 011.

f. Dictionary is 0 → A, 1 → AA, and 2 → AAA; the code is 012.

P8-5. We follow the procedure in Table 8.2. The loop is shown inside the frame with a thick border. We need to initialize the dictionary before reading the code. The message is "**AACCCBCCDDAB**".

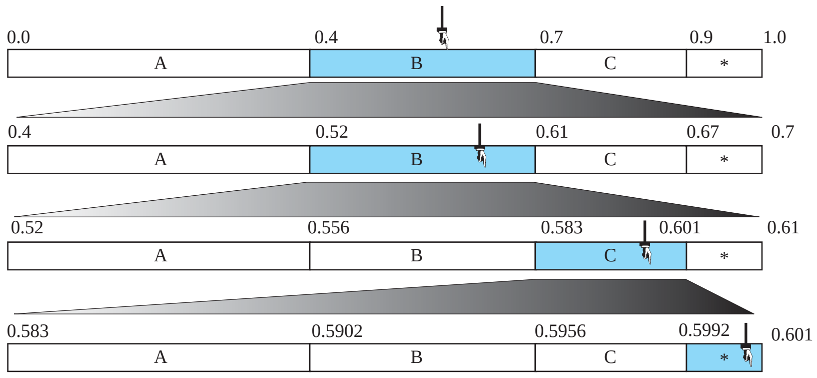
	PreC	S	C	S + first char	Dictionary		Mes-sage
					0	A	
					1	B	
					2	C	
					3	D	
			0				A
if	0	A	0	AA	4	AA	A
if	0	A	2	AC	5	AC	C
else	2	C	6	CC	6	CC	CC
if	6	CC	1	CCB	7	CCB	B
if	1	B	6	BC	8	BC	CC
if	6	CC	3	CCD	9	CCD	D
if	3	D	3	DD	10	DD	D
if	3	D	0	DA	11	DA	A
if	0	A	1	AB	12	AB	B

P8-7. We need to read the code, bit by bit, until we get a sequence of bits that can be decoded using the table:

0	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**0**
A	**A**	–	–	**C**	–	–	**C**	**A**	–	–	**D**	–	**B**	–	–	**D**	–	–	**D**	**A**	–	**B**

The decoded message is "AACCADBDDAB".

P8-9. The code 100110011 is interpreted as $(0.100110011)_2$ or (0.599609375) in decimal. We start with the half-open interval [0, 1) to see where this number fits. The following shows how we narrow the intervals to find the location of the received code. The result is "BBC*"; the actual message is "BBC".



P8-11. We first calculate the value of q_n from the code. If the code is bit 1, the value of q_n is 1; otherwise, it is –1. The value of the y_n can then be calculated as $y_n = y_{n-1} + q_n \times \Delta$. Note that the value of y_0 is not shown in the table, but is given in the problem as $y_0 = 8$.

n	1	2	3	4	5	6	7	8	9	10	11
C_n	1	0	0	1	0	1	1	0	0	1	1
q_n	1	–1	–1	1	–1	1	1	–1	–1	1	1
y_n	14	8	2	8	2	8	14	8	2	8	14

P8-13. We first calculate the value of q_n from the code. If the code is bit 1, the value of q_n is 1; otherwise, it is –1. The value of M_n can then be calculated depending on the change in q_n. We can then calculate the value of Δ_n as defined in the problem. The value of y_n can then be calculated using $y_n = y_{n-1} + q_n \times \Delta_n$. Note that the value of y_0 is not shown in the table, but is given in the problem as $y_0 = 20$.

n	1	2	3	4	5	6	7	8	9	10	11
C_n	1	1	1	0	0	1	1	0	0	1	1
q_n	1	1	1	–1	–1	1	1	–1	–1	1	1
M_n	1	1.5	2.25	1.13	1.70	0.85	1.28	0.64	0.96	0.48	0.72
Δ_n	4	6	13.5	15.3	26	22.1	28.3	18.1	17.4	8.4	6
y_n	24	30	43.5	28.2	2.2	24.3	52.6	34.5	17.1	25.5	31.5

P8-15. The answer is yes. We have $T(m, n) = C(m, n)\cos(x)$. The value of $\cos(x)$ is always between -1 and 1. We only need to prove that $C(m, n) \leq 1$, which can be done using two cases:

a. If $m = 0 \rightarrow C(m, n) = (1/N)^{1/2} \leq 1$ because $N \geq 1$.

b. If $m > 0 \rightarrow C(m, n) = (2/N)^{1/2} \leq 1$ because $N \geq 2$.

P8-17. The following shows the four matrices. The first one ($N = 1$) is actually a number (scalar value). Note that we have rounded the values to two decimal digits.

$$\begin{bmatrix} 1.00 \end{bmatrix}$$

$N = 1$

$$\begin{bmatrix} 0.71 & 0.71 \\ 0.71 & -0.71 \end{bmatrix}$$

$N = 2$

$$\begin{bmatrix} 0.50 & 0.50 & 0.50 & 0.50 \\ 0.65 & 0.27 & -0.27 & -0.65 \\ 0.50 & -0.50 & -0.50 & 0.50 \\ 0.27 & -0.65 & 0.65 & -0.27 \end{bmatrix}$$

$N = 4$

$$\begin{bmatrix} 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 & 0.35 \\ 0.49 & 0.42 & 0.28 & 0.10 & -0.10 & -0.28 & -0.42 & -0.49 \\ 0.46 & 0.19 & -0.19 & -0.46 & -0.46 & -0.19 & 0.19 & 0.46 \\ 0.42 & -0.10 & -0.49 & -0.28 & 0.28 & 0.49 & 0.10 & -0.42 \\ 0.35 & -0.35 & -0.35 & 0.35 & 0.35 & -0.35 & -0.35 & 0.35 \\ 0.28 & -0.49 & 0.10 & 0.42 & -0.42 & -0.10 & 0.49 & -0.28 \\ 0.19 & -0.46 & 0.46 & -0.19 & -0.19 & 0.46 & -0.46 & 0.19 \\ 0.10 & -0.28 & 0.42 & -0.49 & 0.49 & -0.42 & 0.28 & -0.10 \end{bmatrix}$$

$N = 8$

P8-19. The palette should have the combination of $2 \times 2 \times 2 = 8$ rows, as shown below with the bit index.

	Red	Blue	Green	
000	00000000	00000000	00000000	R = 0, B = 0, G = 0
001	00000000	00000000	00000100	R = 0, B = 0, G = 4
010	00000000	00000101	00000000	R = 0, B = 5, G = 0
011	00000000	00000101	00000100	R = 0, B = 5, G = 4
100	00000111	00000000	00000000	R = 7, B = 0, G = 0
101	00000111	00000000	00000100	R = 7, B = 0, G = 4
110	00000111	00000101	00000000	R = 7, B = 5, G = 0
111	00000111	00000101	00000100	R = 7, B = 5, G = 4

a. For each color in the palette we can send three bits (the index of the row).

b. The bit index for the red color is 100, for the blue color is 010, for the green color is 001, for the black color is 000, for the white color is 111, and for the magenta is 110.

P8-21. The receiver misses samples 21, 23, 25, 27, 29, 31, 33, 35, 37, and 39. However, the even-numbered samples are received and played. There may be some glitches in the audio, but that passes immediately.

P8-23. The redundant bits in this case need to find $(n + 1)$ different states because the corruption can be in any of the n bits or in no bits (no corruption). A set of r bits can define 2^r states. This means that we need to have the following relationship: $2^r \geq n + 1$. We need to solve the equation for each value of k using trial and error to find the minimum value of r.

a. If $k = 1$, then $r = 2$ and $n = 3$ because $(2^2 \geq 3 + 1)$, which means $C(3, 1)$.

b. If $k = 2$, then $r = 3$ and $n = 5$ because $(2^3 \geq 5 + 1)$, which means $C(5, 1)$.

c. If $k = 5$, then $r = 4$ and $n = 9$ because $(2^4 \geq 9 + 1)$, which means $C(9, 5)$.

d. If $k = 50$, then $r = 6$ and $n = 56$ because $(2^6 \geq 56 + 1)$, which means $C(56, 50)$.

e. If $k = 1000$, then $r = 10$ and $n = 1010$ because $2^{10} \geq 1010 + 1$, which means $C(1010, 1000)$.

P8-25. If we need to correct m bits in an n bit codeword, we need to think about the combination of n objects taking no object at a time or $Com(n, 0)$, which means the state of no error, the combination of n objects taking one object at a time or $Com(n, 1)$, which means the state of one-bit error, the combination of n objects taking two objects at a time or $Com(n, 2)$, which means the state of two-bit error, and so on. We can have the following relationship between the value of r (number of redundant bits) and the value of m (the number of errors) we need to correct.

$$2^r \geq Com(n, m) + Com(n, m-1) + \ldots + Com(n, 1) + Com(n, 0)$$

P8-27. The answer follows:

a. At time 00:00:17

first packet:	**10 units arrived, 9 units played, 1 unit in buffer**
second packet:	**2 units arrived, none played, 3 units in buffer**

b. At time 00:00:20

first packet:	**none left**
second packet:	**5 units arrived, 2 units played, 3 units in buffer**

c. At time 00:00:25

first packet:	**none left**
second packet:	**10 units arrived, 7 units played, 3 units in buffer**

d. At time 00:00:30

first packet:	**none left**
second packet:	**none left**
third packet:	**3 units arrived, 2 units played, 1 unit in buffer**

P8-29. The first eight hexadecimal digits are in binary:

1000 0110 0000 0011 0010 0001 0011 0010

a. The first two bits define the version, which means the version is 2.

b. The third bit is the P bit, which means there is no padding.

c. The fourth bit is the X bit, which means there is no extension header.

d. The next four bits define the number of contributors, which is 6 in this case.

e. The seven bits after the color bit (M) (with the value 3 in decimal) define the payload type, which means GSM audio.

f. Since we have six contributors, the total size of the packet header is 12 bytes (first part of the header) plus 6×4 bytes for CSRCs, for a total of 36 bytes.

P8-31. We can mention several reasons.

a. Live or real-time streaming of multimedia requires that the data be sent in chunks. A chunk of data is a sequence of bits that have been encoded together and need to be decoded together; each bit may depend on the rest of the bits. Although TCP collects eight bits in a byte and transfers each byte as an atomic unit, the size of the byte is not enough to be a chunk of multimedia data (even a single pixel in an image may need more than eight bits). A chunk cannot be associated with several bytes because TCP may split bytes, making a chunk into different segments, which means a chunk of multimedia data may be in one segment and the other parts in other segments. What is needed in multimedia is that a chunk of data be carried in a single packet. Although TCP may use push and urgent flags to artificially create boundaries for each segment, it is not guaranteed that a TCP implementation will honor these flags all of the time. On the other hand, a UDP user datagram is a packet with clear-cut boundaries. A UDP packet, when combined with RTP/RCTP packets, can carry a single chunk of multimedia payload, and the size of the UDP datagram is large enough to hold any chunk of multimedia data as defined so far.

b. TCP is not an appropriate protocol for multicasting and multi-party communication because it is a connection-oriented protocol in which a connection involves only two parties. UDP, on the other hand, is a connection-less protocol. Multicasting and multi-party communication can be achieved using UDP. Using the same multicast address, a UDP sender can send multicast packets to many receivers; several UDP senders can use the same multicast addresses to create multi-party communication.

c. TCP delivers data bytes to the application program in order. This means that if a segment is missing, the out-of-order bytes need to wait. This is not appropriate for live or real-time interactive multimedia streaming, as discussed in the text. This problem does not exist in UDP. Each UDP packet is independent, and the chunk of data carried in a UDP datagram can be delivered to the application program as it arrives.

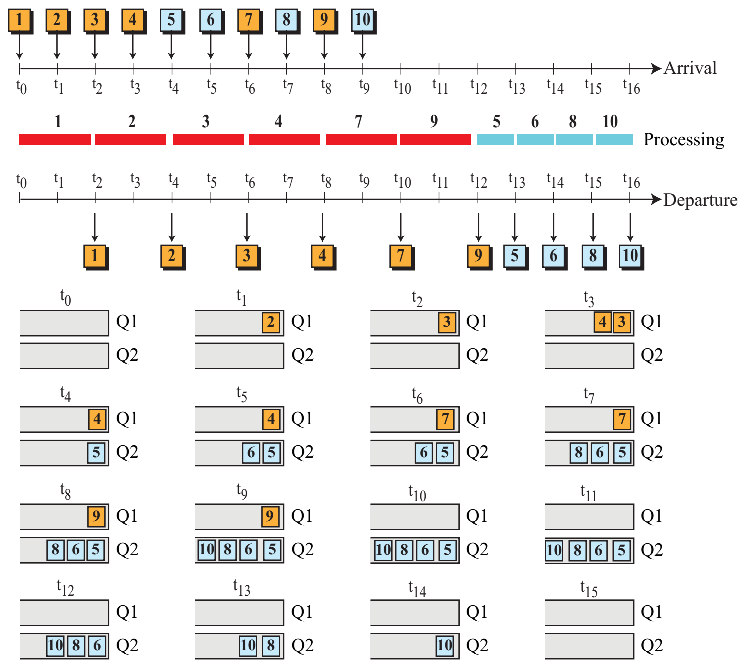
d. Since multimedia chunks can use different encoding methods, we need a field to define the type of payload. A TCP segment lacks this field. A UDP datagram can allow encapsulation of the RTP packet, which includes this field.

e. For a live or real-time multimedia streaming, there is a need for synchronization using timestamps that define the temporal relationship between each chunk and the other chunks. None of the three protocols, TCP, UDP, or SCTP, provide timestamping. UDP, however, allows the encapsulation of the RTP packets, which includes timestamping. Since an SCTP packet can carry many data chunks in a single packet, timestamping may not be needed if the stream is short.

f. Congestion control in multimedia communication is a big issue because of the volume of data exchanged. Although TCP has a congestion control mechanism, it is not suitable for multimedia communication, because it is based on segments, not chunks. Congestion control in UDP/RTP is more convenient for multimedia communication.

P8-33. We answer each question below:

a. The following shows the time lines and the contents of the priority queue (Q1) and non-priority queue (Q2).



b. For the packets in the priority class, we find the time spent in the router for each class, which is the departure time minus the arrival time. We also find the relative departure delay, as shown below. Since the relative delays are the same for each packet, the router does not create jitter in this class.

Packets:	**1**	**2**	**3**	**4**	**7**	**9**
Arrival time:	t_0	t_1	t_2	t_3	t_6	t_8
Departure time:	t_2	t_4	t_6	t_8	t_{10}	t_{12}
Time spent in the router	**2**	**3**	**4**	**5**	**4**	**4**
Relative delay	–	**2**	**2**	**2**	**2**	**2**

c. For the packets in the non-priority class, we do the same, as shown below. We notice that there is no jitter for the packets in this class because the priority packets blocked these packets and they all departed at the end.

Packets:	**5**	**6**	**8**	**10**
Arrival time:	t_4	t_5	t_7	t_9
Departure time:	t_{13}	t_{14}	t_{15}	t_{16}
Time spent in the router	**9**	**9**	**8**	**7**
Relative delay	–	**1**	**1**	**1**

P8-35. We show the list of packets transmitted in each situation:

a. In the first situation, four packets are sent from the top queue, two packets from the middle queue, and one packet from the bottom.

AAAABBCAAAABBCAAAABBCAAAABBCAAAABBCAAAABBC…

b. In the second situation, the same pattern occurs as in part a, but the third queue has no packet to send. The process stops when there is no packet to send.

AAAABBAAAABBAA

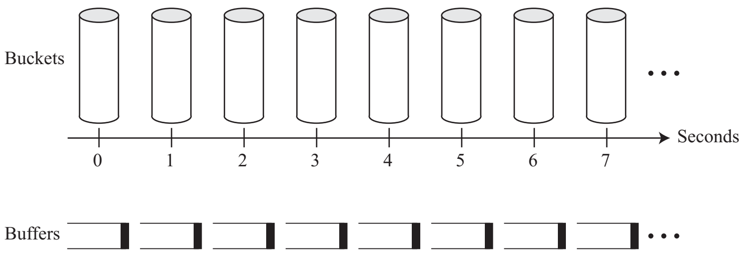
c. In the third situation, the same pattern occurs as in part a, but the first queue has no packet to send. The process stops when there is no packet to send. When there is no packet in the second queue, all packets from C are transmitted one after another.

BBCBBCBCCCCCCCC

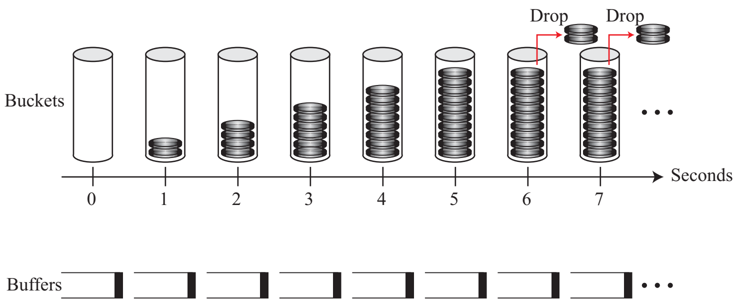
P8-37. The router can implement the leaky bucket algorithm with n = 1 packet in each clock tick and set the clock tick to 1/2 second. In each clock tick, the router sends out one packet. This can be implemented using a FIFO queue that stores the packets and sends them out one each half second. The problem with this approach is that, if the packets continue arriving, a time comes when the queue is full and any new packet would be dropped. The problem occurs because the rate of input is more than the rate of output.

P8-39. We discuss each case separately.

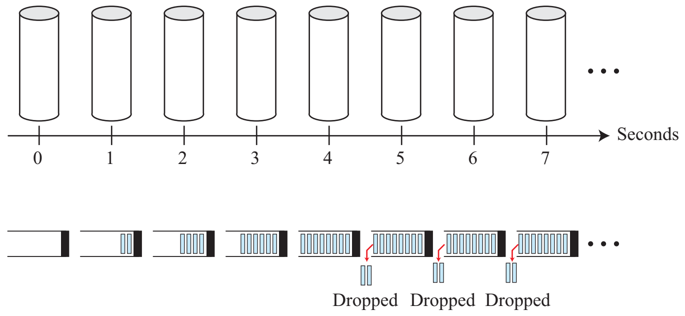
a. The first case is an example of a perfect situation. The sender has matched its transmitting rate with the rate the switch can handle the flow. At the end of each second, both the bucket and the queue are empty, as shown in the following figure. Five packets have arrived each second and five packets have departed each second. No packets have been dropped. Five tokens are added to the bucket in each second and five tokens are removed in each second. The bucket has never become full and has never dropped tokens. Each packet departed with a delay which is equal to the processing time, but there is no jitter (in case the packets belong to the same application); the delay is the same for each packet.



b. The second case is an example in which the sender does not use the capacity of the switch. The sender sends at a lower rate than the rate the switch can handle. At the end of each second, the bucket has some tokens that have not been used. After five seconds, the bucket is full and needs to drop some new tokens added to it. At the end of each second the queue is empty. Each packet departed with a delay which is equal to the processing time, but there is no jitter (in case the packets belong to the same application); the delay is the same for each packet.



c. The third case is an example in which the sender does not follow the capacity of the switch; its rate is more than it should be. The sender sends at a rate higher than the rate the switch can handle. At the end of each second, the bucket has consumed all of its tokens, which means some packets need to wait in the queue until new tokens are injected into the bucket. After four seconds, the queue is full and two packets need to be dropped at the end of each following second. Some packets are lost. The packets that are not lost encounter unequal delays because some need to remain in the queue longer than the others. As the time goes on, each packet encounters more delay. There are packet loss. There is definitely jitter if the packets belong to the same application.



P8-41. We show the frames sent during each second.

a. First tick: *n* is set to 8000.
After sending frame 1, $n = 8000 - 4000 = 4000$.
After sending frame 2, $n = 4000 - 4000 = 0$.
No more frames can be sent ($n <$ size of the next frame).

b. Second tick: *n* is set to 8000.
After sending frame 3, $n = 8000 - 4000 = 4000$.
After sending frame 4, $n = 4000 - 4000 = 0$.
No more frames can be sent ($n <$ size of the next frame).

c. Third tick: *n* is set to 8000.
After sending frame 5, $n = 8000 - 3200 = 4800$.
After sending frame 6, $n = 4800 - 3200 = 1600$.
No more frames can be sent ($n <$ size of the next frame).

d. Fourth tick: *n* is set to 8000.
After sending frame 7, $n = 8000 - 3200 = 4800$.
After sending frame 8, $n = 4800 - 400 = 4400$.
After sending frame 9, $n = 4400 - 400 = 4000$.
After sending frame 10, $n = 4000 - 2000 = 2000$.
After sending frame 11, $n = 2000 - 2000 = 0$.
No more frames can be sent ($n <$ size of the next frame).

e. Fifth tick: *n* is set to 8000.
After sending frame 12, $n = 8000 - 2000 = 6000$.
There are no more frames to send.

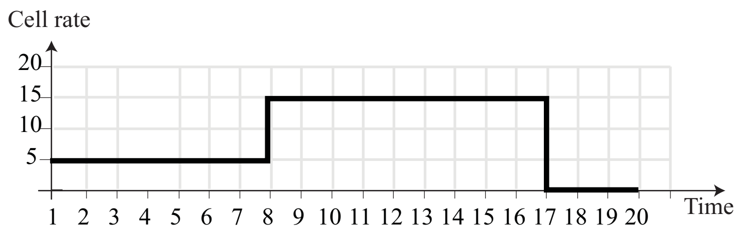
P8-43. To better understand the behavior of the token bucket in this problem, we first create a table in each case to show the movement of cells and tokens in the system. The flow diagram can follow the table. Although adding and removing cells are done for each individual cell, we assume that the cells are first added to the queue, the ones that can be transmitted are removed from the queue, and at the end of each second, we show the number of cells left in the queue. We do the same for the tokens, we first assume that the tokens are added at the beginning of the second to the bucket, some are consumed during each second, and some are discarded if the bucket is full. We need to say that the number of cells transmitted in each second is determined by the following rule.

transmitted cells = ***min*** **(number of cells in queue, number of available tokens)**

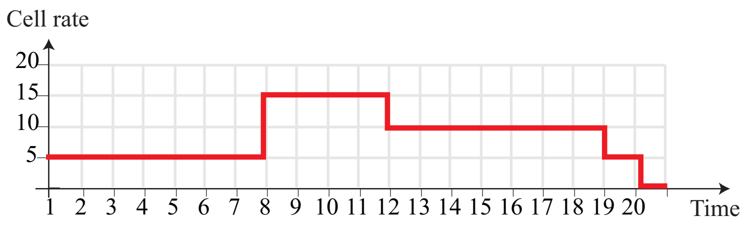
a. The table for the first customer is shown below. We don't have cell loss, but we need to discard some tokens when the bucket is full (t_5 to t_7). Discarding tokens means that the customer has not used its allocated rate.

Time	Number of cells					Number of Tokens				
(t_i)	previ-ously in queue	arr-ived at ISP	in queue after adding	sent by ISP	left in queue	previ-ously in bucket	added to bucket	in bucket after	con-sumed for cells	left in bucket
01	0	5	5	5	0	0	10	10	5	5
02	0	5	5	5	0	5	10	15	5	10
03	0	5	5	5	0	10	10	20	5	15
04	0	5	5	5	0	15	10	25	5	20
05	0	5	5	5	0	20	10	30	5	20
06	0	5	5	5	0	20	10	30	5	20
07	0	5	5	5	0	20	10	30	5	20
08	0	15	15	15	0	20	10	30	15	15
09	0	15	15	15	0	15	10	25	15	10
10	0	15	15	15	0	10	10	20	15	5
11	0	15	15	15	0	5	10	15	15	0
12	0	15	15	10	5	0	10	10	10	0
13	5	15	20	10	10	0	10	10	10	0
14	10	15	25	10	15	0	10	10	10	0
15	15	15	30	10	20	0	10	10	10	0
16	20	15	35	10	25	0	10	10	10	0
17	25	0	25	10	15	0	10	10	10	0
18	15	0	15	10	5	0	10	10	10	0
19	5	0	5	5	0	0	10	10	5	5
20	0	0	0	0	0	5	10	15	0	15

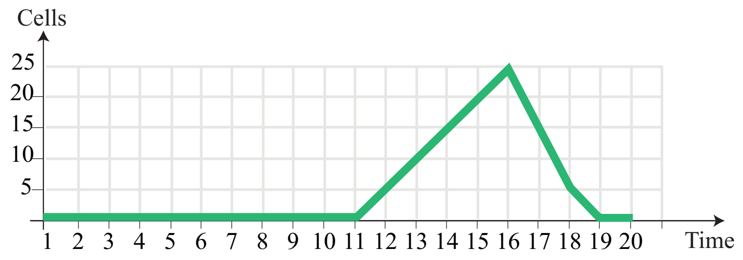
The following shows the movement of cells and tokens graphically. The figure definitely shows that there are uneven delays for the cells after t_8. Some cells are stored in the queue for further transmission, but there is no cell loss because the queue size if very large.



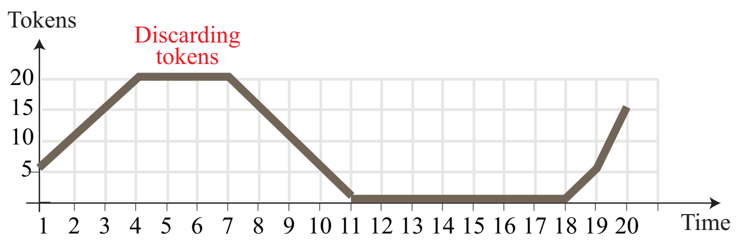
Cells arrived at ISP



Cells departed at ISP



Cells in the queue

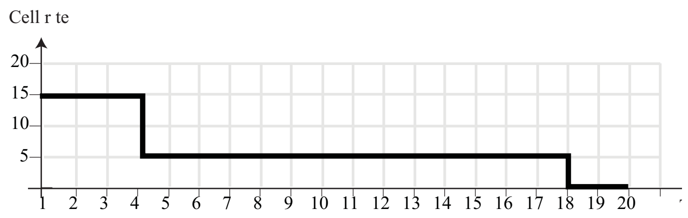


Tokens left in bucket

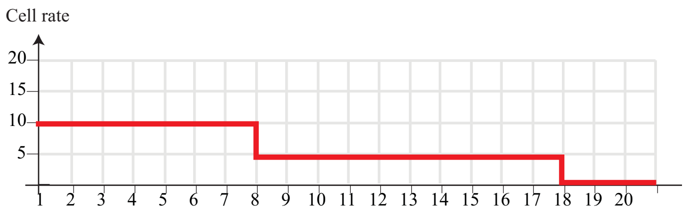
b. The table for the second customer is shown below. We don't have cell loss, but we need to discard some tokens when the bucket is full (t_{13} to t_{20}). Discarding of token means that the customer has not used its allocated rate.

Time	Number of cells					Number of Tokens				
(t_i)	previously in queue	arrived at ISP	in queue after adding	sent by ISP	left in queue	previously in bucket	added to bucket	in bucket after adding	consumed for cells	left in bucket
01	0	15	15	10	5	0	10	10	10	0
02	5	15	20	10	10	0	10	10	10	0
03	10	15	25	10	15	0	10	10	10	0
04	15	15	30	10	20	0	10	10	10	0
05	20	5	25	10	15	0	10	10	10	0
06	15	5	20	10	10	0	10	10	10	0
07	10	5	15	10	5	0	10	10	10	0
08	5	5	10	10	0	0	10	10	10	0
09	0	5	5	5	0	0	10	10	5	5
10	0	5	5	5	0	5	10	15	5	10
11	0	5	5	5	0	10	10	20	5	15
12	0	5	5	5	0	15	10	25	5	20
13	0	5	5	5	0	20	10	30	5	20
14	0	5	5	5	0	20	10	30	5	20
15	0	5	5	5	0	20	10	30	5	20
16	0	5	5	5	0	20	10	30	5	20
17	0	5	5	5	0	20	10	30	5	20
18	0	5	5	5	0	20	10	30	5	20
19	0	0	0	0	0	20	10	30	0	20
20	0	0	0	0	0	20	10	30	0	20

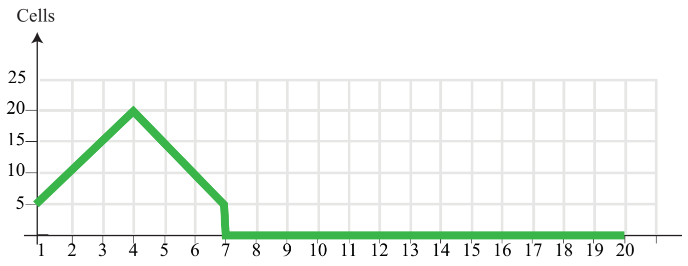
The following shows the movement of cells and tokens graphically. The figure definitely shows that there are uneven delays for the cells. Some cells are stored in the queue for further transmission, but there is no cell loss because the queue size is very large.



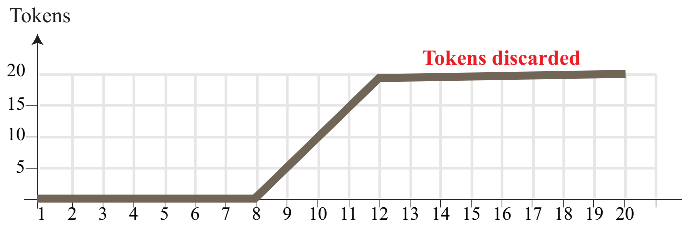
Cells arrived at ISP



Cells departed at ISP



Cells in the queue

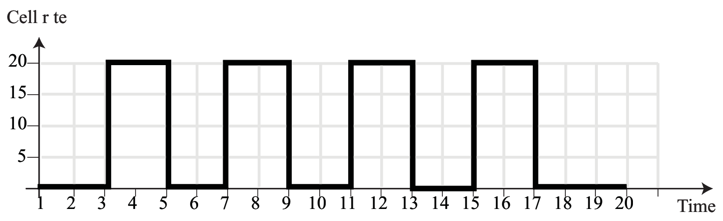


Tokens left in bucket

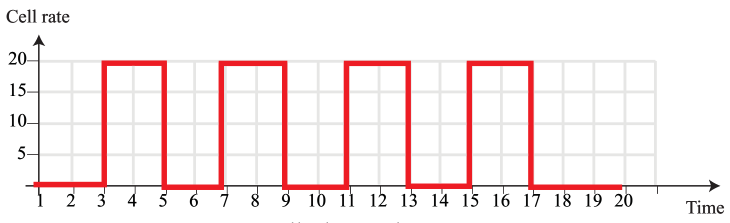
c. The table for the third customer is shown below. We don’t have cell loss; no tokens are discarded. This means that customer has used its full allocated rate. It does not send cells for two seconds, but it sends two times its rate for the next two seconds.

Time	Number of cells					Number of Tokens				
(t_i)	previ-ously in queue	arrived at ISP	in queue after adding	sent by ISP	left in queue	previ-ously in bucket	added to bucket	in bucket after add-ing	con-sumed for cells	left in bucket
01	0	0	0	0	0	0	10	10	0	10
02	0	0	0	0	0	10	10	20	0	20
03	0	20	20	20	0	20	10	30	20	10
04	0	20	20	20	0	10	10	20	20	0
05	0	0	0	0	0	0	10	10	0	10
06	0	0	0	0	0	10	10	20	0	20
07	0	20	20	20	0	20	10	30	20	10
08	0	20	20	20	0	10	10	20	20	0
09	0	0	0	0	0	0	10	10	0	10
10	0	0	0	0	0	10	10	20	0	20
11	0	20	20	20	0	20	10	30	20	10
12	0	20	20	20	0	10	10	20	20	0
13	0	0	0	0	0	0	10	10	0	10
14	0	0	0	0	0	10	10	20	0	20
15	0	20	20	20	0	20	10	30	20	10
16	0	20	20	20	0	10	10	20	20	0
17	0	0	0	0	0	0	10	20	0	10
18	0	0	0	0	0	10	10	20	0	20
19	0	0	0	0	0	20	10	30	0	20
20	0	0	0	0	0	20	10	30	0	20

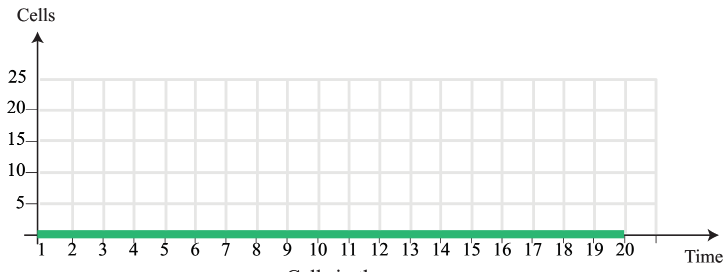
The following shows the movement of cells and tokens graphically. This is a situation in which there is no jitter (comparing the customer and ISP cell-rate transmission. No tokens are discarded until the 18*th* second.



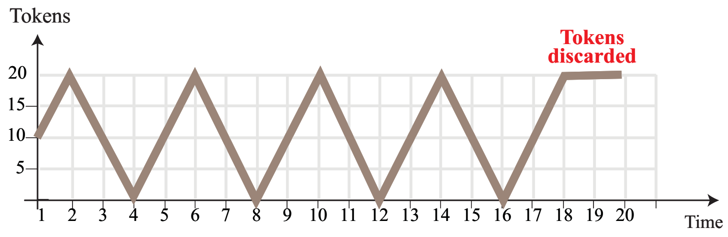
Cells arrived at ISP



Cells departed at ISP



Cells in the queue



Tokens left in bucket