CHAPTER 30

PRACTICE SET

Questions

Q30-1. We cannot define the absolute order, but in normal situation we will rank them as follows:

1. SNMP 2. HYYP 3. SMTP 4. VoIP

Q30-2. We cannot define the absolute order, but in normal situation we will rank them as follows:

1. VoIP 2. SNMP 3. HTTP 4. SMTP

Q30-3. We cannot define the absolute order, but in normal situation we will rank them as follows:

1. VoIP 2. SNMP 3. HTTP 4. SMTP

Q30-4. We cannot define the absolute order, but in normal situation we will rank the as follows:

1. VoIP 2. HTTP 3. SMTP 4. SNMP

Q30-5. Our choice is VoIP. The goal is to make VoIP similar to the telephone conversation we have over the phone using circuit switching.

Q30-6. Our choice is SMTP. When we send e-mails, we expect that the whole contents, including multimedia, reaches the receiver. However, the e-mail packets can be sent whenever bandwidth available.

Q30-7. None of these applications actually fit in this flow class. It is mostly designed for interactive multimedia applications.

Q30-8. All of these applications can use this flow class. As a matter of fact, the current Internet uses this class for all applications.

Q30-9. FIFO queueing and priority queueing are techniques used for scheduling.

Q30-10. Token and leaky buckets are used for traffic shaping.

Q30-11. IntServ uses two flow specifications: Rspec and Tspec.

a. Rspec (Resource specification) defines the resources that the flow needs to reserve before starting the communication.

b. Tspec (Traffic specification) defines the traffic specification.

Q30-12.

a. The *guaranteed-service* class guarantees the minimum end-to-end delay.

b. The *controlled-load* service class guarantees that the no part of the communication is dropped due to congestion and network overload.

Q30-13. IntServ is called *destination-base* service because the destination host needs to define the type of service required to start the communication.

Q30-14. DiffServ is called *source-base* service because the source host needs to define the type of service required to start the communication.

Q30-15. IntServ is designed for multicast communication. However, unicast is a special case of multicast in which there is only one member in the group.

Q30-16. In DiffServ, the source host defines the type of service. Both unicast and multicast communication have only one source.

Q30-17. In IntServ, the resources needs to be reserved before the communication starts. We need to find the resources needed at each router. We need to send Path and collect Resv packets to find the resources the routers need for communication flow.

Q30-18. DiffServ defines three per-hop behaviors: DE, EF, and AF

a. DE (default) behavior is the best-effort delivery. The hop tries to do its best, but it many not be possible.

b. EF (expected forwarding) behavior is similar to virtual connection. It provides low loss, low delay, and ensured bandwidth.

c. AF (assured forwarding) behavior delivers the packet with high assurance as long as the class traffic does not exceed the node profile.

Q30-19. The traffic conditioner has four components:

a. Meter

b. Marker

c. Shaper

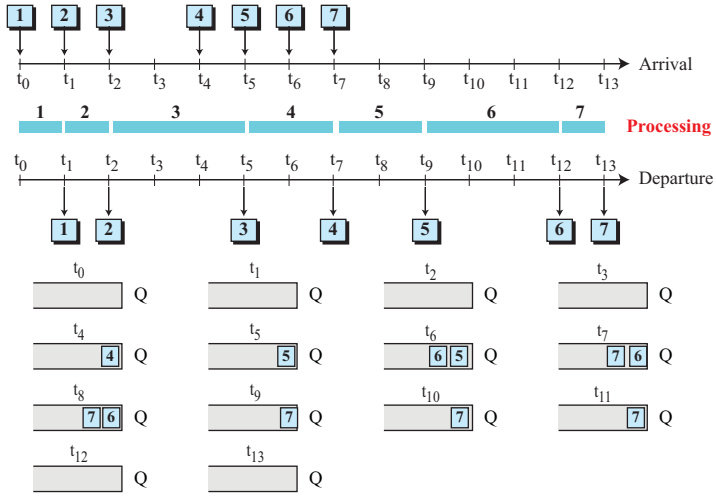
d. Dropper

Q30-20. The flow label in IPv6 is more appropriate for DiffServ quality service. It is actually designed for this purpose.

Problems

P30-1. We answer each question separately:

a. The following shows the time lines and the contents of the queue:



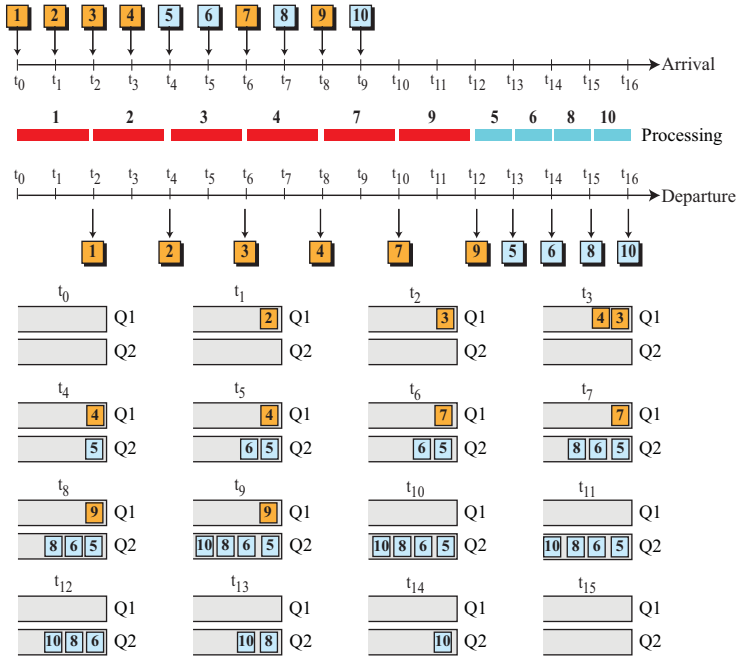
b. The following shows the calculation of the time spent in the router and the relative delay for each packet.

Packets:	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Arrival time	t_0	t_1	t_2	t_4	t_5	t_6	t_7
Departure time	t_1	t_2	t_5	t_7	t_9	t_{12}	t_{13}
Time spent in the router	1	1	3	3	4	6	6
Relative departure delay	–	1	3	2	2	3	1

c. Since the relative departure delay between each packet and the previous one is not the same, there is jitter even if all packets have the same propagation delay.

P30-2. We answer each question below:

a. The following figure 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.

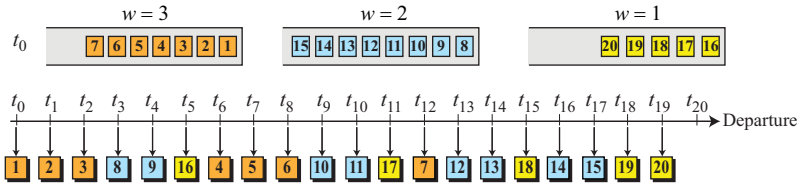
Packets:	**1**	**2**	**3**	**4**	**7**	**9**
Arrival time:	t_0	t_1	t_2	t_3	t_6	t_8
Departure time:	t_2	t_4	t_6	t_8	t_{10}	t_{12}
Time spent in the router	2	3	4	5	4	4
Relative delay	–	2	2	2	2	2

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.

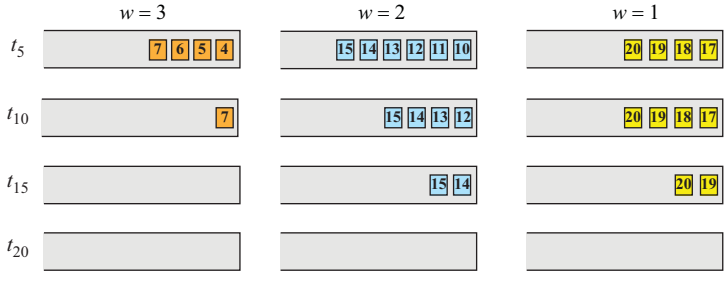
Packets:	**5**	**6**	**8**	**10**
Arrival time:	t_4	t_5	t_7	t_9
Departure time:	t_{13}	t_{14}	t_{15}	t_{16}
Time spent in the router	9	9	8	7
Relative delay	–	1	1	1

P30-3. We answer each question separately:

a. The following shows the departure time line.



b. The following shows the contents of queues at t_0, t_5, t_{10}, t_{15}, and t_{20}.



c. We can calculate the relative delay between packets for the class $w = 3$ as shown below. Since departure delays are different, the packets introduce jitter even if they have the same propagation time.

Packets	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Departure time	t_0	t_1	t_2	t_6	t_7	t_8	t_{12}
Delay with respect to previous packet	–	1	1	4	1	1	4

d. We can calculate the relative delay between packets for the class $w = 2$ as shown below. Since departure delays are different, the packets introduce jitter even if they have the same propagation time.

Packets	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Departure time	t_3	t_4	t_9	t_{10}	t_{13}	t_{14}	t_{16}	t_{17}
Delay with respect to previous packet	–	1	5	1	3	1	2	1

e. We can calculate the relative delay between packets for the class $w = 1$ as shown below. Since departure delays are different, the packets introduce jitter even if they have the same propagation time.

Packets	16	17	18	19	20
Departure time	t_5	t_{11}	t_{15}	t_{18}	t_{19}
Delay with respect to previous packet	–	6	4	3	1

P30-4. 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

P30-5.

a. The input during the first minute is (100 gallons/minute) × (12/60 minute) or 20 gallons.

b. The output during the first minutes is (5 gallons/minute) × (1 minute) or 5 gallons.

c. This means that after the first minute, 15 gallons of liquid is left in the bucket. The problem does not mention the input rate after the first minute. If there is no input flow, the bucket would be empty after four minutes.

P30-6. 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.

P30-7. We set the value of n = 1000 bits and the time tick to one second. In each tick of time, only 2 packets will be transmitted, as shown below.

a. The value of the counter is set to 1000; the size of the first packet (400 bits) is less than the value of the counter. The first packet is transmitted. The value of the counter is now decremented to 600.

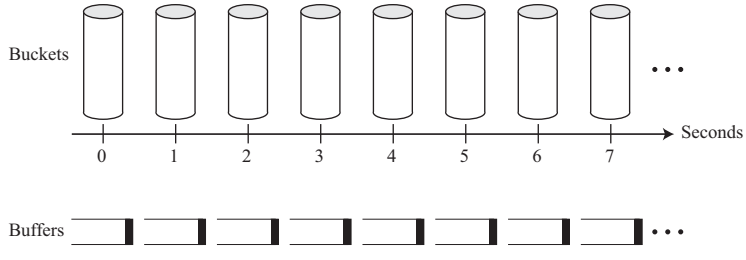
b. The value of the counter is now 600; the size of the second packet (400 bits) is less than the value of the counter, so it is transmitted. The value of the counter is now decremented to 200.

c. The value of the counter is now 200; the size of the third packet (400 bits) is now larger than the value of the counter, so it cannot be transmitted. It needs to wait for the next tick of time (next second).

d. The process repeats for every second. In each tick of time only two packets can be sent. This means 800 bits per second or the rate of 0.8 Kbps. We have reduced the rate from 4 Kbps to 0.8 Kbps.

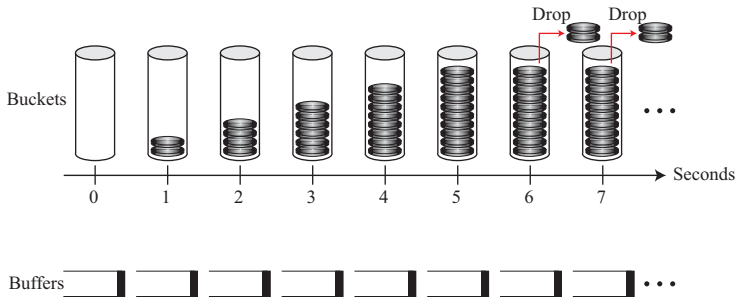
P30-8. We discuss each case separately.

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.

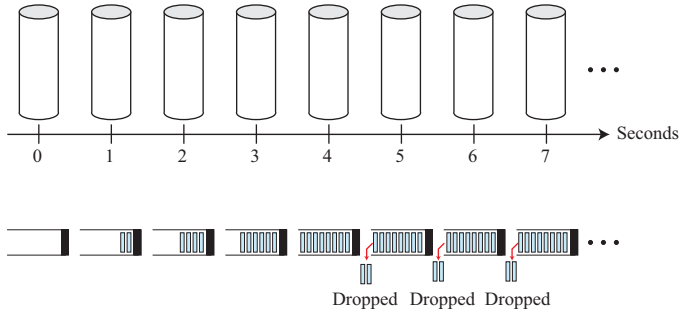


b. The second case is an example in which the sender does not use the capacity of the switch. The sender sends at a lower rate than the rate the switch can handle. At the end of each second, the bucket has some tokens that have not been used. After five seconds, the bucket is full and needs to drop some 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.

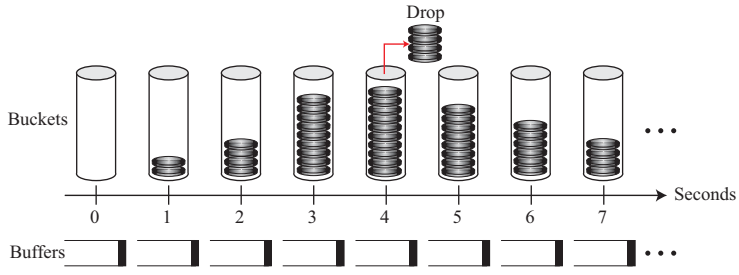


c. The third case is an example in which the sender does not follow the capacity of the switch; its rate is more than it should be. The sender sends at a rate higher than the rate the switch can handle. At the end of each second, the bucket has consumed all of its tokens, which means some packets need 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.



P30-9. The following shows that the sender collects credits (tokens) that can be used during the last three seconds. However, the sender loses some of the tokens

when the bucket becomes full because it uses very little during the first four seconds and starts using the credits a little bit late.



P30-10. 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.

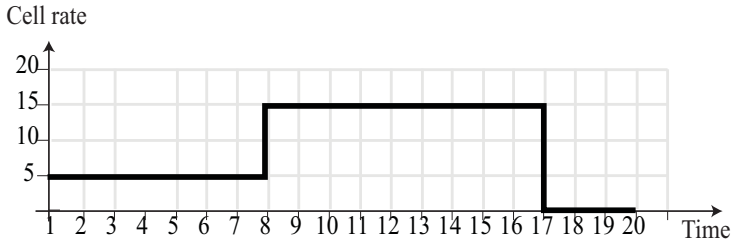
P30-11. To better understand the behavior of the leaky bucket in this problem, we first create a table in each case to show the movement of cells in the system. The flow diagram can follow the table. Although adding, removing, and dropping cells are done for each individual cell, we assume that the cells are first added to the queue, the ones that can be transmitted are removed from the queue, and

at the end of each second, the extra cells are dropped (discarded) if the queue is full. Now we create the table and the flow diagrams for each case separately.

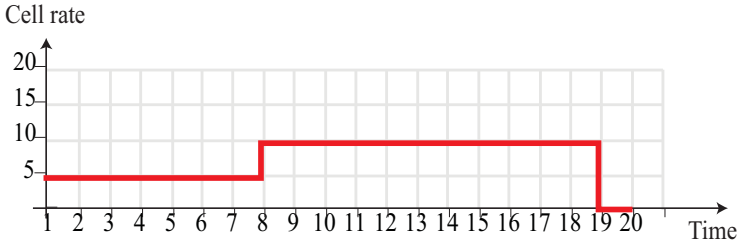
a. The following shows the table for the first case. The table shows that there are occasions when the queue is full and some cells are dropped (t_{12} to t_{16}).

(t_i)	**Previous number of cells in queue**	Number of cells arrived	**Total number of cells in queue**	**Number of cells sent**	**Number of cells left in queue**	**Number of cells dropped**
01	0	5	5	5	0	0
02	0	5	5	5	0	0
03	0	5	5	5	0	0
04	0	5	5	5	0	0
05	0	5	5	5	0	0
06	0	5	5	5	0	0
07	0	5	5	5	0	0
08	0	15	15	10	5	0
09	5	15	20	10	10	0
10	10	15	25	10	15	0
11	15	15	30	10	20	0
12	20	15	35	10	**20**	5
13	20	15	35	10	**20**	5
14	20	15	35	10	**20**	5
15	20	15	35	10	**20**	5
16	20	15	35	10	**20**	5
17	20	0	20	10	10	0
18	10	0	10	10	0	0
19	0	0	0	0	0	0
20	0	0	0	0	0	0

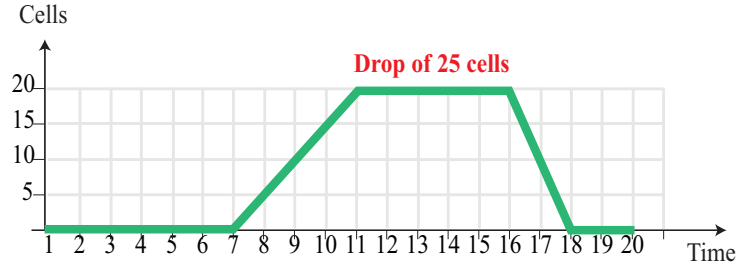
The following shows the flow of cells graphically. The first seven seconds, the ISP rate follows the rate of the customer, but the ISP rate is reduced (to the maximum 10 cells per second. The customer sends 170 cells, but the ISP sends only 145 cells, which means that 25 cells are lost. The third figure shows the number of cells left in the queue in each second. Since some cells are queues and some not, this means that there is uneven delay between the cells (jitter).



Cells arrived at ISP (170)



Cells cent by ISP (145)

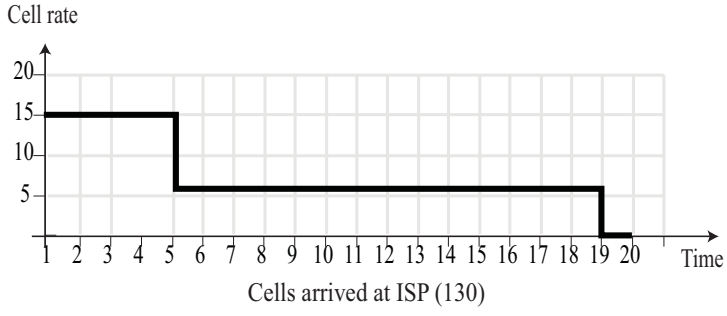


Cells in the queue

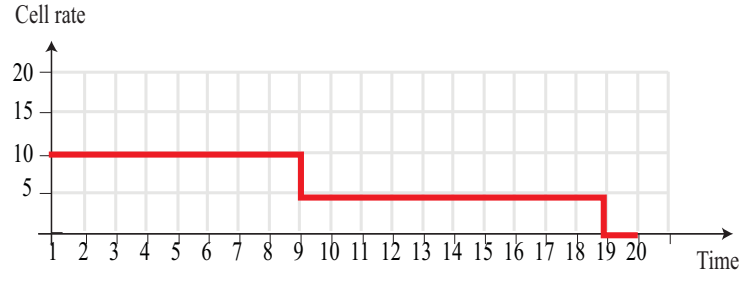
b. The following shows the table for the second case. The table shows that there is no cell lost. Although the customer sends more number of cells in each second than its maximum rate, this only occurs only for four seconds and does not overflow the queue.

(t_i)	**Previ-ous number of cells in queue**	Number of cells arrived	**Total number of cells in queue**	**Number of cells sent**	**Number of cells left in queue**	**Number of cells dropped**
01	0	15	15	10	5	0
02	5	15	20	10	10	0
03	10	15	25	10	15	0
04	15	15	30	10	20	0
05	20	5	25	10	15	0
06	15	5	20	10	10	0
07	10	5	15	10	5	0
08	5	5	10	10	0	0
09	0	5	5	5	0	0
10	0	5	5	5	0	0
11	0	5	5	5	0	0
12	0	5	5	5	0	0
13	0	5	5	5	0	0
14	0	5	5	5	0	0
15	0	5	5	5	0	0
16	0	5	5	5	0	0
17	0	5	5	5	0	0
18	0	5	5	5	0	0
19	0	0	0	0	0	0
20	0	0	0	0	0	0

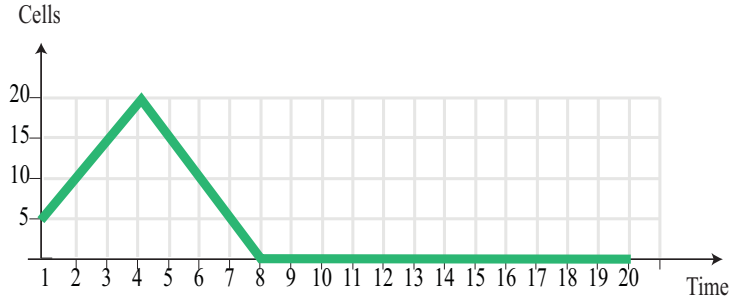
The following shows the flow of cells graphically. The customer sends 130 cells, the ISP also sends 130 cells. There is no cell loss, but there is a delay because some of the cells need to be queued during the first four seconds. The queueing delays all cells after the t_5 (jitter). The queue first becomes full, but it gradually becomes empty.



Cells arrived at ISP (130)



Cells cent by ISP (130)

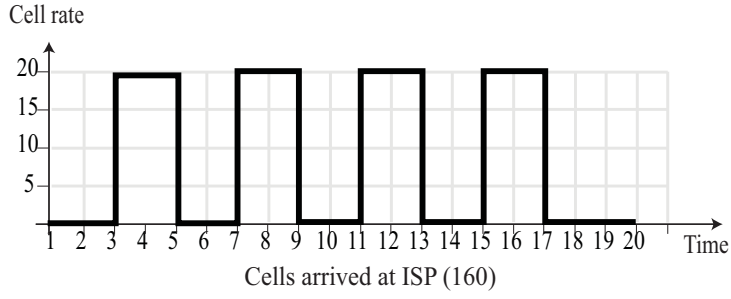


Cells in the queue

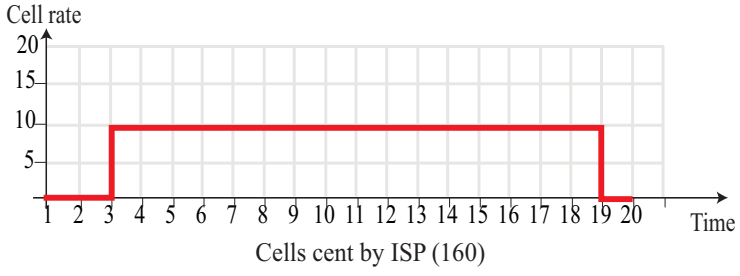
c. The table for the third customer is shown below. The following shows the table for the third case. This is another case in which there is no cell loss.

(t_i)	**Previ-ous number of cells in queue**	Number of cells arrived	**Total number of cells in queue**	**Number of cells sent**	**Number of cells left in queue**	**Number of cells dropped**
01	0	0	0	0	0	0
02	0	0	0	0	0	0
03	0	20	20	10	10	0
04	10	20	30	10	20	0
05	20	0	20	10	10	0
06	10	0	10	10	0	0
07	0	20	20	10	10	0
08	10	20	30	10	20	0
09	20	0	20	10	10	0
10	10	0	10	10	0	0
11	0	20	20	10	10	0
12	10	20	30	10	20	0
13	20	0	20	10	10	0
14	10	0	10	10	0	0
15	0	20	20	10	10	0
16	10	20	30	10	20	0
17	20	0	20	10	10	0
18	10	0	10	10	0	0
19	0	0	0	0	0	0
20	0	0	0	0	0	0

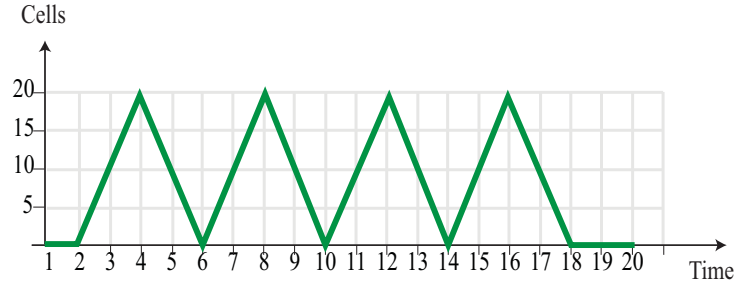
The following shows the flow of cells graphically. This is a very interesting case. Although, some cells are queues, but the cells are departed from the ISP with even delays. If we think that the cells are created at the ISP, we can say that, from t_3 to t_{18}, ISP sends 10 cells per second.



Cells arrived at ISP (160)



Cells cent by ISP (160)



Cells in the queue

P30-12. 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

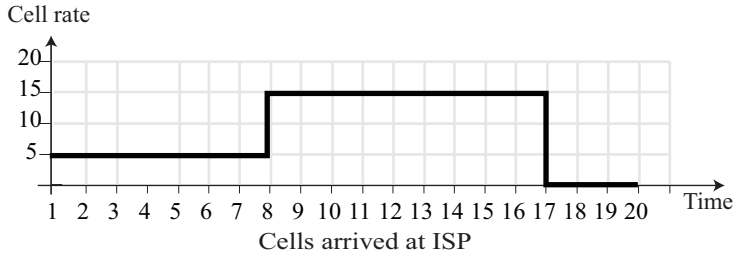
added to the queue, the ones that can be transmitted are removed from the queue, and at the end of each second, we show the number of cells left in the queue. We do the same for the tokens, we first assume that the tokens are added at the beginning of the second to the bucket, some are consumed during 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)

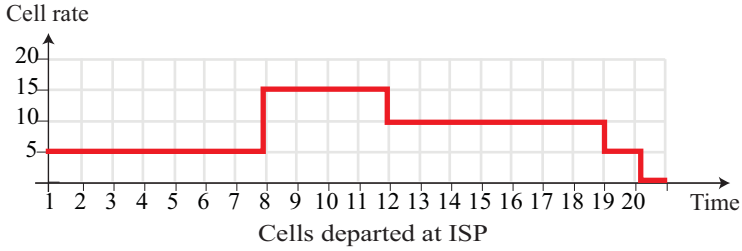
a. The table for the first customer is shown below. We don't have cell loss, but we need to discard some tokens when the bucket is full (t_5 to t_7). Discarding tokens means that the customer has not used its allocated rate.

Time	Number of cells					Number of Tokens				
(t_i)	in queue before	arrived	in queue after	sent	left in queue	in queue	added to bucket	in bucket after	used 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

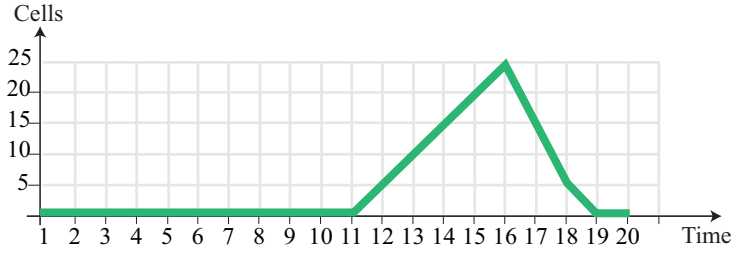
The following shows the movement of cells and tokens graphically. The figure definitely shows that there are uneven delays for the cells after t_8. Some cells are stored in the queue for further transmission, but there is no cell loss because the queue size if very large.



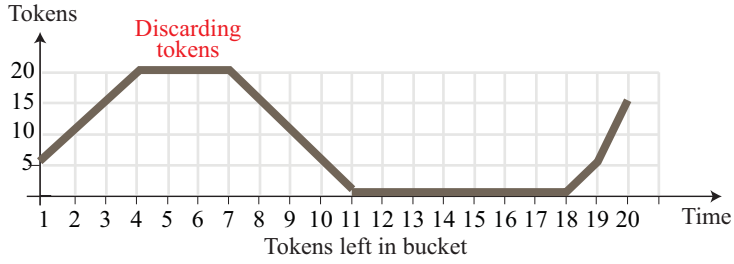
Cells arrived at ISP



Cells departed at ISP



Cells in the queue

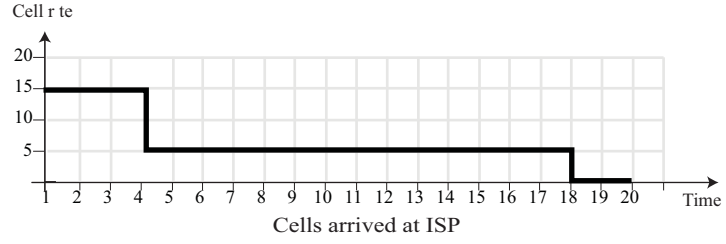


Tokens left in bucket

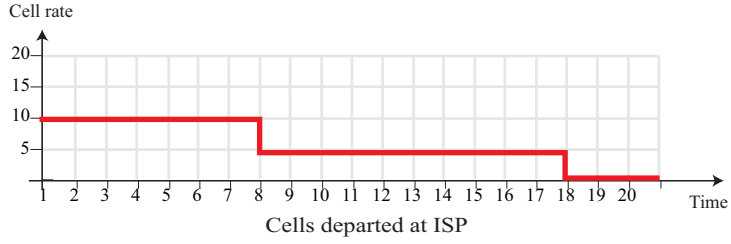
b. The table for the second customer is shown below. We don't have cell loss, but we need to discard some tokens when the bucket is full (t_{13} to t_{20}). Discarding of token means that the customer has not used its allocated rate.

Time	Number of cells					Number of Tokens				
(t_i)	**in queue before**	arrived at ISP	**in queue after adding**	**sent by ISP**	**left in queue**	**in bucket**	**added to bucket**	**in bucket after**	**used 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**

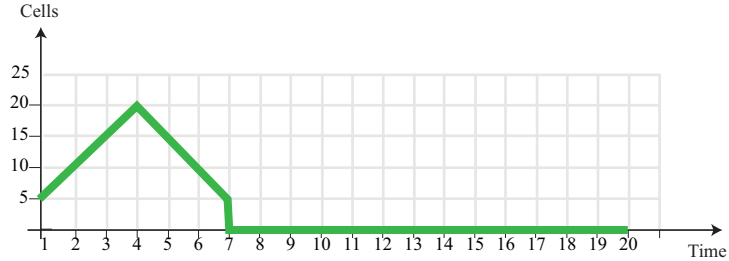
The following shows the movement of cells and tokens graphically. The figure definitely shows that there are uneven delays for the cells. Some cells are stored in the queue for further transmission, but there is no cell loss because the queue size is very large.



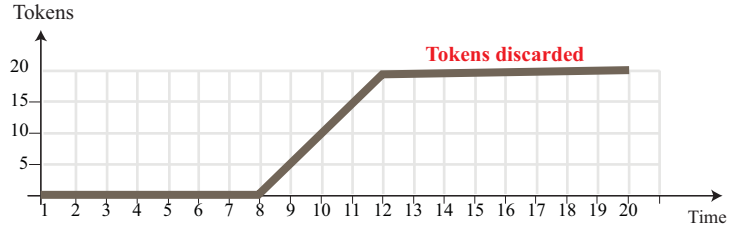
Cells arrived at ISP



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)	in queue before	arrived at ISP	in queue after adding	sent by ISP	left in queue	in bucket before	added to bucket	in bucket after	used 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.