Experiment No:7

Aim: Write a program for congestion control using Leaky bucket algorithm.

Theory:

The congestion control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination.

In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back.

The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer.

Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time.

The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.

A]Leaky Bucket Algorithm

The leaky-bucket implementation is used to control the rate at which traffic is sent to the network. A leaky bucket provides a mechanism by which bursty traffic can be shaped to present a steady stream of traffic to the network, as opposed to traffic with erratic bursts of low-volume and high-volume flows.

Traffic analogy: An appropriate analogy for the leaky bucket is a scenario in which four lanes of automobile traffic converge into a single lane. A regulated admission interval into the single lane of traffic flow helps the traffic move. The benefit of this approach is that traffic flow into the major arteries (the network) is predictable and controlled. The major liability is that when the volume of traffic is vastly greater than the bucket size, in conjunction with the drainage-time interval, traffic backs up in the bucket beyond bucket capacity and is discarded.

Algorithm:

1. Start
2. Set the bucket size or the buffer size.
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)

6. Stop

The algorithm can be conceptually understood as follows:

- Arriving packets (network layer PDUs) are placed in a bucket with a hole in the bottom.
- The bucket can queue at most *b* bytes. If a packet arrives when the bucket is full, the packet is discarded.
- Packets drain through the hole in the bucket, into the network, at a constant rate of *r* bytes per second, thus smoothing traffic bursts.

The size *b* of the bucket is limited by the available memory of the system.
Sometimes the leaky bucket and token algorithms are lumped together under the same name.

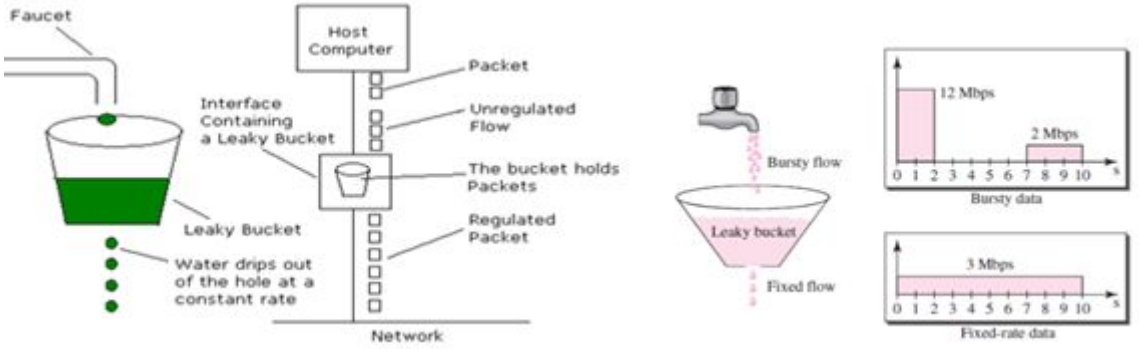


Figure 1: Leaky bucket

B]Token bucket Algorithm

Need of token bucket Algorithm:-

The leaky bucket algorithm enforces output pattern at the average rate, no matter how bursty the traffic is. So in order to deal with the bursty traffic we need a flexible algorithm so that the data is not lost. One such algorithm is token bucket algorithm.

Steps of this algorithm s:

1. In regular intervals tokens are thrown into the bucket. *f*
2. The bucket has a maximum capacity. *f*
3. If there is a ready packet, a token is removed from the bucket, and the packet is send.
4. If there is no token in the bucket, the packet cannot be send.

In figure (a) we see a bucket holding three tokens, with five packets waiting to be transmitted. For a packet to be transmitted, it must capture and destroy one token. In figure (b) We see that three of the five packets have gotten through, but the other two are stuck waiting for more tokens to be generated.

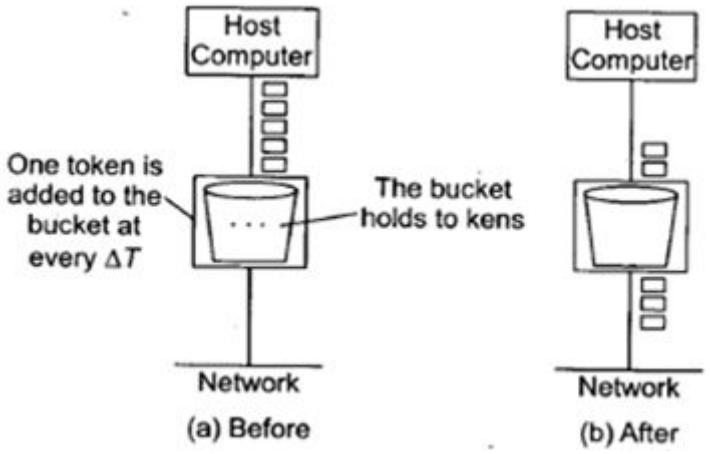


Figure 2: Token Bucket

Program:

```
import java.util.*;
class Leaky
{
  public static void main(String args[])
  {
        int b,c, i, clk, b_size, o_rate, p_sz_rm=0, p_sz, p_time, op,high=10000000,low=10,x;
    int packet_sz[]= new int[10];
        Random rad= new Random();
        Scanner sc = new Scanner(System.in);
        c= rad.nextInt(high-low)+low;
        c=(c%10)%4;
        if(c==0)
        {
              b=rad.nextInt(high-low)+low;
           b=(b%10)%6;
        }

        System.out.println(c+" TIMES BATMAN IS BETTER than any other hero  ");
        for(i=0;i<10;++i)
        {
              b=rad.nextInt(high-low)+low;
              b=(b%10)%6;
              if(b==0)
              {
            b=rad.nextInt(high-low)+low;
               b=(b%10)%6;
           }
             packet_sz[i] = b * 10;
        }

    for(i = 0; i<10; ++i)
        {
            System.out.println("\npacket "+i+" : "+packet_sz[i]+ " bytes\t");
        }
        System.out.println("\nEnter the Output rate:");
        o_rate = sc.nextInt();
        System.out.println("\nEnter the Bucket Size:");
        b_size = sc.nextInt();
        for(i = 0; i<10; ++i)
    {
         if( (packet_sz[i] + p_sz_rm) > b_size)
```

```
    {
        if(packet_sz[i] > b_size){
System.out.println("\n\nIncoming packetsize"+packet_sz[i]+"bytes is Greater than bucket capacity "+b_size+"
bytes PACKET REJECTED");
                        }
                        else
                        {
                                System.out.println("\n\nBucket apacity exceeded-PACKETS REJECTED!!");
                        }
                }
                else
                {
                        p_sz_rm += packet_sz[i];
                    System.out.println("\n\nIncoming Packet size:"+ packet_sz[i]);
                        System.out.println("\nBytes remaining to Transmit: "+ p_sz_rm);
                        p_time = c * 10;
                        System.out.println("\nTime left for transmission: "+p_time+" units" );
                        for(clk = 10; clk <= p_time; clk += 10)
                        {
                    try
                                {
                                  Thread.sleep(100);
                                }
            catch(InterruptedException ex)
             {
                    Thread.currentThread().interrupt();
             }
          if(p_sz_rm!=0)//checks value of p_sz_rm is greater than 0
                        {
                                        if(p_sz_rm <= o_rate)//checks value of output rate is greater than
p_sz_rm
                                        {
                                                op = p_sz_rm;
                                                p_sz_rm = 0;
                                        }
                                        else
                                        {
                                                op = o_rate;
                                                p_sz_rm -= o_rate;
                                                System.out.println("\nPacket of size "+op+" Transmitted");
                                       System.out.println("----Bytes Remaining to Transmit: "+ p_sz_rm);

                                        }
                                }
                                else
                                {
                                        x=p_time-clk;
                                        System.out.println("\nTime left for transmission:"+x+" units" );
                                        System.out.println("\nNo packets to transmit!!");
                            } }}
 }}}
```

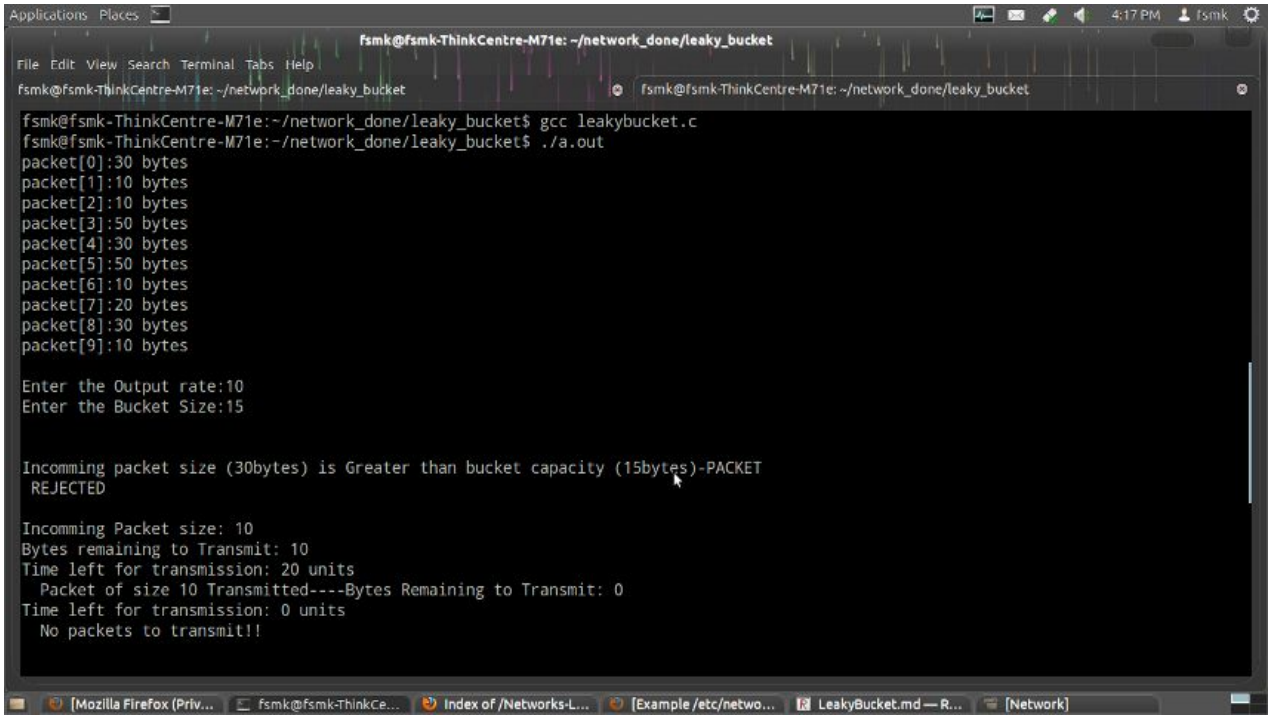
Alternate solution where we manually input each packet

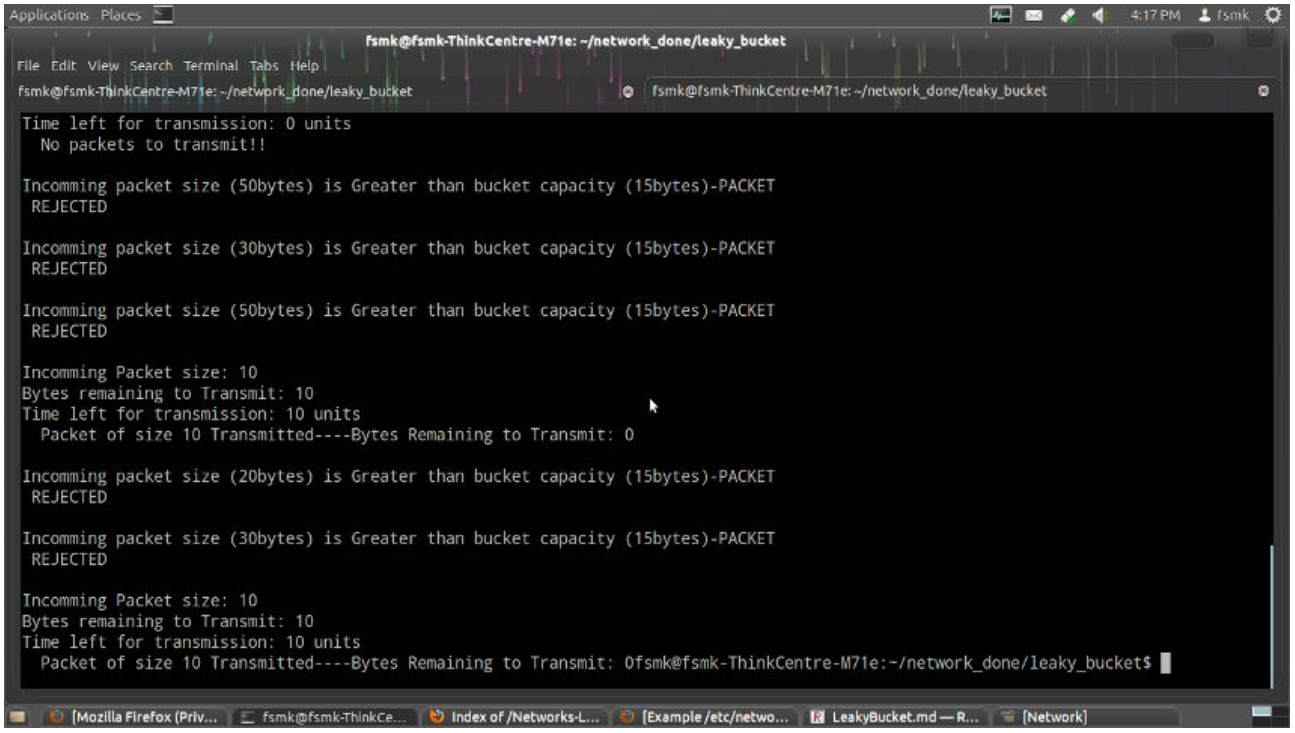
```c
#include<stdio.h>

int main(){
   int incoming, outgoing, buck_size, n, store = 0;
   printf("Enter bucket size, outgoing rate and no of inputs: ");
   scanf("%d %d %d", &buck_size, &outgoing, &n);

   while (n != 0) {
      printf("Enter the incoming packet size : ");
      scanf("%d", &incoming);
      printf("Incoming packet size %d\n", incoming);
      if (incoming <= (buck_size - store)){
         store += incoming;
         printf("Bucket buffer size %d out of %d\n", store, buck_size);
      } else {
         printf("Dropped %d no of packets\n", incoming - (buck_size - store));
         printf("Bucket buffer size %d out of %d\n", store, buck_size);
         store = buck_size;
      }
      store = store - outgoing;
      printf("After outgoing %d packets left out of %d in buffer\n", store, buck_size);
      n--;
   }
}
```

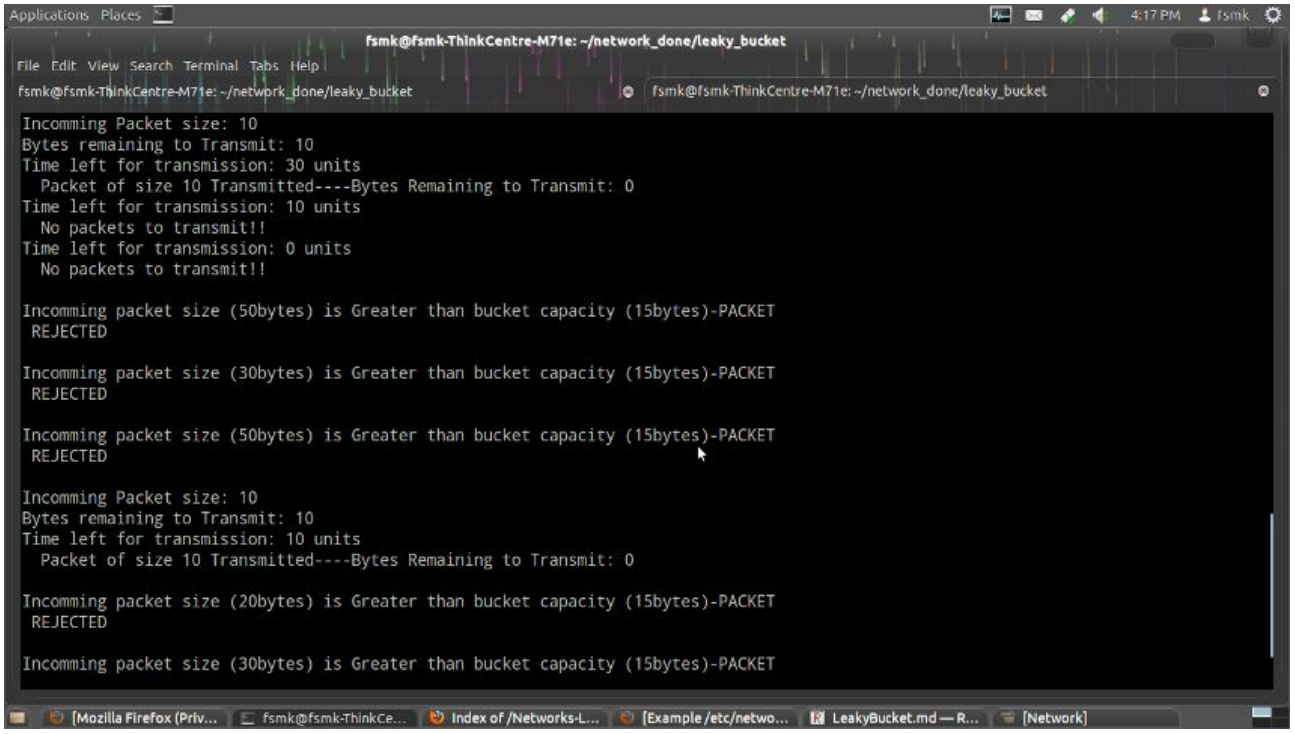
Output:

Commands for execution:Open a terminal.

Change directory to the file location.
Run gcc filename.c
If there are no errors, run ./a.out


Applications Places
4:17 PM fsmk
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
File Edit View Search Terminal Tabs Help
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
fsmk@fsmk-ThinkCentre-M71e:~/network_done/leaky_bucket$ gcc leakybucket.c
fsmk@fsmk-ThinkCentre-M71e:~/network_done/leaky_bucket$ ./a.out
packet[0]:30 bytes
packet[1]:10 bytes
packet[2]:10 bytes
packet[3]:50 bytes
packet[4]:30 bytes
packet[5]:50 bytes
packet[6]:10 bytes
packet[7]:20 bytes
packet[8]:30 bytes
packet[9]:10 bytes
Enter the Output rate:10
Enter the Bucket Size:15
Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
REJECTED
Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 20 units
Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0
Time left for transmission: 0 units
No packets to transmit!!
[Mozilla Firefox (Priv...
fsmk@fsmk-ThinkCe...
Index of /Networks-L...
[Example /etc/netwo...
LeakyBucket.md — R...
[Network]


Applications Places
4:17 PM fsmk
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
File Edit View Search Terminal Tabs Help
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
fsmk@fsmk-ThinkCentre-M71e: ~/network_done/leaky_bucket
Time left for transmission: 0 units
No packets to transmit!!
Incomming packet size (50bytes) is Greater than bucket capacity (15bytes)-PACKET
REJECTED
Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
REJECTED
Incomming packet size (50bytes) is Greater than bucket capacity (15bytes)-PACKET
REJECTED
Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0
Incomming packet size (20bytes) is Greater than bucket capacity (15bytes)-PACKET
REJECTED
Incomming packet size (30bytes) is Greater than bucket capacity (15bytes)-PACKET
REJECTED
Incomming Packet size: 10
Bytes remaining to Transmit: 10
Time left for transmission: 10 units
Packet of size 10 Transmitted----Bytes Remaining to Transmit: 0fsmk@fsmk-ThinkCentre-M71e:~/network_done/leaky_bucket$
[Mozilla Firefox (Priv...
fsmk@fsmk-ThinkCe...
Index of /Networks-L...
[Example /etc/netwo...
LeakyBucket.md — R...
[Network]



Conclusion: Hence we successfully studied congestion controlling using Leaky bucket algorithm.

Date:

Sign: **Grade:**