PART A PROGRAMS

Program 1

Implement three nodes points-to point network with duplex links between them for different topologies. 1set the queue size, vary the bandwidth, and find the number of packet dropped for various iterations.

Step1: open text editor, type the below program and save with extension .tcl (p1.tcl)

```
set ns [new Simulator]
set nf [open prog1.nam w]
$ns namtrace-all $nf
Set nd [open prog1.tr w]
$ns trace-all $nd

proc finish {}{
global ns nf nd
$ns flush-trace
close $nf
close $nd
exec nam prog1.nam &
exit 0
}

set n0 [$ns node]
set n1[$ns node]
set n2[$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms  DropTail
$ns duplex-link $n1 $n2 512Mb 10ms  DropTail
$ns queue-limit $n1 $n2 10
```

```
Set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]

$cbr0 set packetSize_500
$cbr0 set interval_0.005
$cbr0 attach-agent $udp0

set sink [new Agent/Null]
$ns attach-agent $n2 $sink
$ns connect $udp0 $sink

$ns at 0.2 “$cbr0 start”
$ns at 4.5 “$cbr0 stop”
$ns at 5.0 “finish”
$ns run
```

Step 2: Open text editor,type the below program and save with extention.awk(prog1.awk)

```
BEGIN {
 dcount=0;
  rcount=0;
  }
  {
   event = $1;
if(event==”d”)
```

```
{
dcount++;
}
if(event==”r”)
{
rcount++;
}
}
END {
printf(“The no.of packets dropped: %d\n “,dcount);
 printf(“The no.of packets received: %d\n “,rcount);
}
```

Step3: Run the simulation program **ns prog1.tcl**

Step 4: Now press the play button in the simulation window and the simulation will begin.

Step 5:after simulations is completed run **awk file** to see output,

awk –f prog.awk prog.tr

Step6: To see the trace file contents open the file as,

vi prog.tr

Program 2

Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.

Step1: Open text editor, type the below program and save with extention .tcl (**prog6.tcl**)

```
set ns [new Simulator]

set tf [open prog6.tr w]


$ns trace-all $tf

set topo [new Topography]

$topo load_flatgrid 1000 1000
set nf [open prog6.nam w]

$ns namtrace-all-wireless $nf 1000 1000


set val(chan) Channel/WirelessChannel ;
set val(prop) Propagation/TwoRayGround ;
set val(netif) Phy/WirelessPhy ;

set val(mac) Mac/802_11 ;

set val(ifq) Queue/DropTail/PriQueue ;
set val(ll) LL ;

set val(ant) Antenna/OmniAntenna ;
set val(ifqlen) 50 ;


set val(nn) 2 ;

set val(rp) AODV ;
set val(x) 500 ;

set val(y) 400 ;

set val(stop) 10.0 ;
```

```
$ns node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON

create-god 3

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"

$n0 set X_ 50
```

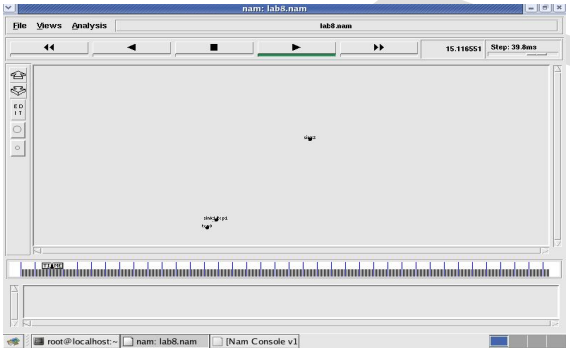
```
$n0 set Y_ 50

$n0 set Z_ 0

$n1 set X_ 100

$n1 set Y_ 100

$n1 set Z_ 0

$n2 set X 600

$n2 set Y 600

$n2 set Z  0

$ns at 0.1 "$n0 setdest 50 50 15"

$ns at 0.1 "$n1 setdest 100 100 25"

$ns at 0.1 "$n2 setdest 600 600 25"


set tcp0 [new Agent/TCP]

$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]

$ftp0 attach-agent $tcp0

set sink1 [new Agent/TCPSink]

$ns attach-agent $n1 $sink1

$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]

$ns attach-agent $n1 $tcp1

set ftp1 [new Application/FTP]

$ftp1 attach-agent $tcp1

set sink2 [new Agent/TCPSink]

$ns attach-agent $n2 $sink2
```

```
$ns connect $tcp1 $sink2


$ns at 5 "$ftp0 start"

$ns at 5 "$ftp1 start"

$ns at 100 "$n1 setdest 550 550 15"

$ns at 190 "$n1 setdest 70 70 15"


proc finish { } {

        global ns nf tf

        $ns flush-trace

        exec nam prog6.nam  &
        close $tf

        exit 0

}

$ns at 250 "finish"

$ns run
```

Step2: Open text editor, type the below program and save with extention .awk (**prog6.awk**)

```
BEGIN{

        count1=0
        count2=0
        pack1=0



                p
        ack2=0
        time1=0
        time2=0

}
```

```
{       if($1=="r"&& $3=="_1_" && $4=="AGT")

        {       count1++
                pack1=pack1+$8
                time1=$2        }

        if($1=="r" && $3=="_2_" && $4=="AGT")

        {       count2++
                pack2=pack2+$8
                time2=$2        }

}
END{

printf("The Throughput from n0 to n1: %f Mbps \n", ((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps", ((count2*pack2*8)/(time2*1000000)));

}
```



Program 3

Implement transmission of ping messages/trace route over a network topology consisting of 6nodes and find the number of packets dropped due to congestion.

Step1: Open text editor, type the below program and save with extention .tcl

(**prog3.tcl**)set ns [new Simulator]

```
set nf [open prog3.nam w]

$ns namtrace-all $nf

set nd [open prog3.tr w]

$ns trace-all $nd


proc finish {}
{global ns nf
nd

$ns flush-trace
close  $nf
close $nd

exec nam prog4.nam &
exit 0

}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]


$ns duplex-link $n1 $n0 1Mb 10ms DropTail
```

```
$ns duplex-link $n2 $n0 1Mb 10ms DropTail

$ns duplex-link $n3 $n0 1Mb 10ms DropTail

$ns duplex-link $n4 $n0 1Mb 10ms DropTail

$ns duplex-link $n5 $n0 1Mb 10ms DropTail

$ns duplex-link $n6 $n0 1Mb 10ms DropTail


Agent/Ping instproc recv {from rtt} {

$self instvar node_

puts "node [$node_ id] recieved ping answer from \

$from with round-trip-time $rtt ms."


        set     p1      [new
Agent/Ping] set p2 [new
Agent/Ping] set p3 [new
Agent/Ping] set p4 [new
Agent/Ping] set p5 [new
Agent/Ping] set p6 [new
Agent/Ping]


$ns attach-agent $n1 $p1

$ns attach-agent $n2 $p2

$ns attach-agent $n3 $p3

$ns attach-agent $n4 $p4

$ns attach-agent $n5 $p5

$ns attach-agent $n6 $p6


$ns queue-limit $n0 $n4 3
```

```
$ns queue-limit $n0 $n5 2

$ns queue-limit $n0 $n6 2


$ns connect $p1 $p4

$ns connect $p2 $p5

$ns connect $p3 $p6


$ns at 0.2 "$p1 send"

$ns at 0.4 "$p2 send"

$ns at 0.6 "$p3 send"

$ns at 1.0 "$p4 send"

$ns at 1.2 "$p5 send"

$ns at 1.4 "$p6 send"

$ns at 2.0 "finish"

$ns run
```

Step2: Open text editor, type the below program and save with extention .awk (**prog3.awk**)BEGIN {

```
count=0;

}

{

event=$1;
if(event=="d")

{

count++;


        }
```

```
}
END {
printf("No of packets dropped : %d\n",count);
}
```

Step3: Run the simulation program

[root@localhost~]# ns prog3.tcl

Step 4: Now press the play button in the simulation window and the simulation will begins.

Step 5: After simulation is completed run awk file to see the output ,

[root@localhost~]# awk –f prog3.awk prog3.tr

Step 6: To see the trace file contents open the file as ,

[root@localhost~]# vi prog3.tr

Program 4:

Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

Step1: Open text editor, type the below program and save with extention .tcl (**prog5.tcl**)

```
set ns [new Simulator]

set nf [open prog5.nam w]

$ns namtrace-all $nf

set nd [open prog5.tr w]

$ns trace-all $nd


$ns color 1 Blue

$ns color 2 Red


proc finish { }
{global ns nf nd

$ns flush-trace
close  $nf
close $nd

exec nam prog5.nam &
exit 0

}


set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

```
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]


$n7 shape box

$n7 color Blue

$n8 shape hexagon

$n8 color Red
```

```
$ns duplex-link $n1 $n0 2Mb 10ms DropTail
$ns duplex-link $n2 $n0 2Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 20ms DropTail

$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 40ms LL Queue/DropTail Mac/802_3

$ns duplex-link-op $n1 $n0 orient right-down
$ns duplex-link-op $n2 $n0 orient right-up
$ns duplex-link-op $n0 $n3 orient right

$ns queue-limit $n0 $n3 20

set tcp1 [new Agent/TCP/Vegas]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n7 $sink1
$ns connect $tcp1 $sink1
$tcp1 set class_ 1
$tcp1 set packetsize_ 55

set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
```

```
set tfile [open cwnd.tr w]
$tcp1 attach $tfile
$tcp1 trace cwnd_

set tcp2 [new Agent/TCP/Reno]
$ns attach-agent $n2 $tcp2
set sink2 [new Agent/TCPSink]
$ns attach-agent $n8 $sink2
$ns connect $tcp2 $sink2
$tcp2 set class_ 2
$tcp2 set packetSize_ 55

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2

set tfile2 [open cwnd2.tr w]
```

```
$tcp2 attach $tfile2

$tcp2 trace cwnd_

$ns at 0.5 "$ftp1 start"

$ns at 1.0 "$ftp2 start"

$ns at 5.0 "$ftp2 stop"

$ns at 5.0 "$ftp1 stop"


$ns at 5.5 "finish"

$ns run
```

Step2: Open text editor, type the below program and save with extention .awk (**prog5.awk**)

```
BEGIN {

}

{

if($6=="cwnd_")
{ printf("%f\t%f\n",$1,$
7);

}

}

END {

}
```

Step3: Run the simulation program

[root@localhost~]# ns prog5.tcl

(Here **"ns"** indicates network simulator. We get the topology shown in the snapshot.)

Step 4: Now press the play button in the simulation window and the simulation will begins.

Step 5: After simulation is completed run awk file and generate the graph , [root@localhost~]

awk –f prog5.awk cwnd.tr > a1 [root@localhost~]

awk –f prog5.awk cwnd2.tr > a2 [root@localhost~]

#xgraph a1 a2

Step 6: To see the trace file contents open the file as ,

[root@localhost~]# vi prog5.tr

PART B PROGRAMS

Experiment No 1
CRC

Write a program for error detecting code using CRC-CCITT (16 bits).

Theory

CRC(Cyclic Redundancy Check) is an error detecting technique used in digital networks and storage devices to detect the accidental changes to raw data. It cannot be used for correcting errors.

If an error is detected in the received message, a 'Negative acknowledgement' is sent to the sender. The sender and the receiver agree upon a fixed polynomial called generator polynomial. The standard agreed generator polynomial is $x^{16}+x^{12}+x^5+x^0$ (any polynomial can be considered, of degree 16).

The CRC does error checking via polynomial division. The generated polynomial g(x) = $x^{16}+x^{12}+x^5+x^0$

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1

→ 17 bits.

So the g(x) value is 10001000000100001

Algorithm:

1. Given a bit string (message to be sent), append 16 0^S to the end of it (the number of 0^s is the same as the degree of the generator polynomial) let this string + 0^S be called as modified string B
2. Divide B by agreed on polynomial g(x) and determine the remainder R(x). The 16-bit remainder received is called as checksum.
3. The message string is appended with checksum and sent to the receiver.
4. At the receiver side, the received message is divided by generator polynomial g(x).
5. If the remainder is 0, the receiver concludes that there is no error occurred otherwise, the receiver concludes an error occurred and requires a retransmission.

```
import java.io.*;
class crc_gen

{

   public static void main(String args[]) throws IOException

   {

      BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
      int[] data;

      int[] div;
      int[] divisor;
      int[] rem;

      int[] crc;

      int data_bits, divisor_bits, tot_length;



      System.out.println("Enter number of data bits : ");
      data_bits=Integer.parseInt(br.readLine());
      data=new int[data_bits];



      System.out.println("Enter data bits : ");
      for(int i=0; i<data_bits; i++)

         data[i]=Integer.parseInt(br.readLine());



      divisor_bits = 17;

      divisor = new int[]{1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1};


      tot_length=data_bits+divisor_bits-1;

      div=new int[tot_length];
```

```
    rem=new int[tot_length];
    crc=new int[tot_length];
/*----------CRC GENERATION----------*/
    for(int i=0;i<data.length;i++)
        div[i]=data[i];

    System.out.print("Dividend (after appending 0's) are : ");
    for(int i=0; i< div.length; i++)
        System.out.print(div[i]);
    System.out.println();

    for(int j=0; j<div.length;
         j++){rem[j] = div[j];

    }

    rem=divide(divisor, rem);

    for(int i=0;i<div.length;i++)          //append dividend and ramainder
    {
        crc[i]=(div[i]^rem[i]);
    }

    System.out.println();
    System.out.println("CRC code : ");
    for(int i=0;i<crc.length;i++)
        System.out.print(crc[i]);
```

```
/*----------ERROR DETECTION----------*/
    System.out.println();

    System.out.println("Enter CRC code of "+tot_length+" bits : ");
    for(int i=0; i<crc.length; i++)

        crc[i]=Integer.parseInt(br.readLine());



    for(int j=0; j<crc.length;
         j++){rem[j] = crc[j];

    }


    rem=divide(divisor, rem);

    for(int i=0; i< rem.length; i++)

    {

        if(rem[i]!=0)

        {

            System.out.println("Error");
            break;

        }

        if(i==rem.length-1)
            System.out.println("No Error");

    }


    System.out.println("THANK YOU.....)");

}

  static int[] divide(int divisor[], int rem[])
```

```
{
    int cur=0;
    while(true)
    {
        for(int i=0;i<divisor.length;i++)
            rem[cur+i]=(rem[cur+i]^divisor[i]);


        while(rem[cur]==0 && cur!=rem.length-1)
            cur++;
        if((rem.length-cur)<divisor.length)
            break;
    }
    return rem;
}
```

OUTPUT

RUN1:

```
C:\Users\abhijith\Desktop>javac crc_gen.java

C:\Users\abhijith\Desktop>java crc_gen
Enter number of data bits :
3
Enter data bits :
1 1 0
Dividend (after appending 0's) are : 1100000000000000000

CRC code :
11001100000011000110
Enter CRC code of 19 bits :
1 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 0
No Error
THANK YOU.... :)
```

Bellman-Ford algorithm

Problem Statement

Write a program to find the shortest path between vertices using bellman-ford algorithm.

Theory

Routing algorithm is a part of network layer software which is responsible for deciding which output line an incoming packet should be transmitted on. If the subnet uses datagram internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the subnet uses virtual circuits (connection Oriented), routing decisions are made only when a new established route is being set up.

Routing algorithms can be grouped into two major classes: adaptive and nonadaptive. Nonadaptive algorithms do not base their routing decisions on measurement or estimates of current traffic and topology. Instead, the choice of route to use to get from I to J (for all I and J) is compute in advance, offline, and downloaded to the routers when the network ids booted. This procedure is sometime called static routing.

Adaptive algorithms, in contrast, change their routing decisions to reflect changes in the topology, and usually the traffic as well. Adaptive algorithms differ in where they get information (e.g., locally, from adjacent routers, or from all routers), when they change the routes (e.g., every ΔT sec, when the load changes, or when the topology changes), and what metric is used for optimization (e.g., distance, number of hops, or estimated transit time).

Two algorithms in particular, distance vector routing and link state routing are the most popular. Distance vector routing algorithms operate by having each router maintain a table (i.e., vector) giving the best known distance to each destination and which line to get there. These tables are updated by exchanging information with the neighbors.

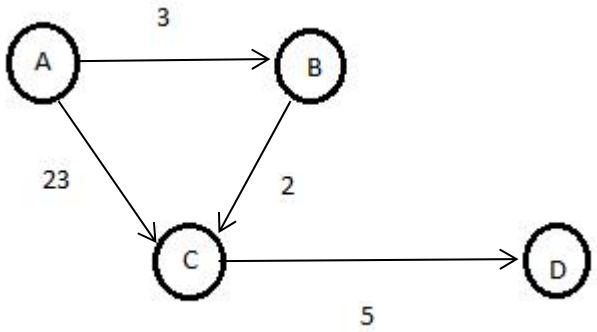
The distance vector routing algorithm uses Bellman-Ford routing algorithm and Ford-Fulkerson algorithm. In distance vector routing, each router maintains a routing table that contains two parts: the preferred out going line to use for that destination, and an estimate of the time or distance to that destination. The metric used might be number of hops, time delay in milliseconds, total number of packets queued along the path, or something similar.

Program

```
import java.io.*;

import java.util.Scanner;
class dist_vec

{

public static void main(String args[])

{

        int dmat[][];

        int dist[][];

        int via[][];

        int n=0,i=0,j=0,k=0,count=0;

        Scanner in = new Scanner(System.in);



        System.out.println("enter the number of
        nodes\n");n = in.nextInt();



        dmat = new int[n][n];
        dist = new int[n][n];
        via = new int[n][n];

        System.out.println("enter the cost
        matrix\n");for(i=0;i<n;i++)

        for(j=0;j<n;j++)
```

```
        {
                dmat[i][j] = in.nextInt();
                dmat[i][i]=0;
                dist[i][j]=dmat[i][j];
                via[i][j]=j;
        }
        do
        {
                count=0;
                for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                for(k=0;k<n;k++)
                if(dist[i][j]>dmat[i][k]+dist[k][j])
                {
                        dist[i][j]=dist[i][k]+dist[k][j];
                        via[i][j]=k;
                        count++;
                }
        }while(count!=0);
        for(i=0;i<n;i++)
        {
                System.out.println("state value for router"+i+" is");
                for(j=0;j<n;j++)
                {
                        System.out.println("To "+j+" -Via "+via[i][j]+" distance is "+dist[i][j]);
                }
        }
}
}
```

OUT PUT




```
enter the number of nodes

4
enter the cost matrix

0 3 23 999
999 0 2 999
999 999 999 5
999 999 999 999
state value for router0 is
To 0 -Via 0 distance is 0
To 1 -Via 1 distance is 3
To 2 -Via 1 distance is 5
To 3 -Via 2 distance is 10
state value for router1 is
To 0 -Via 0 distance is 999
To 1 -Via 1 distance is 0
To 2 -Via 2 distance is 2
To 3 -Via 2 distance is 7
state value for router2 is
To 0 -Via 0 distance is 999
To 1 -Via 1 distance is 999
To 2 -Via 2 distance is 0
To 3 -Via 3 distance is 5
state value for router3 is
To 0 -Via 0 distance is 999
To 1 -Via 1 distance is 999
To 2 -Via 2 distance is 999
To 3 -Via 3 distance is 0
```

Leaky Bucket

Problem Statement

Write a program for congestion control using leaky bucket algorithm.

Theory

The congesting 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 occurin 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.

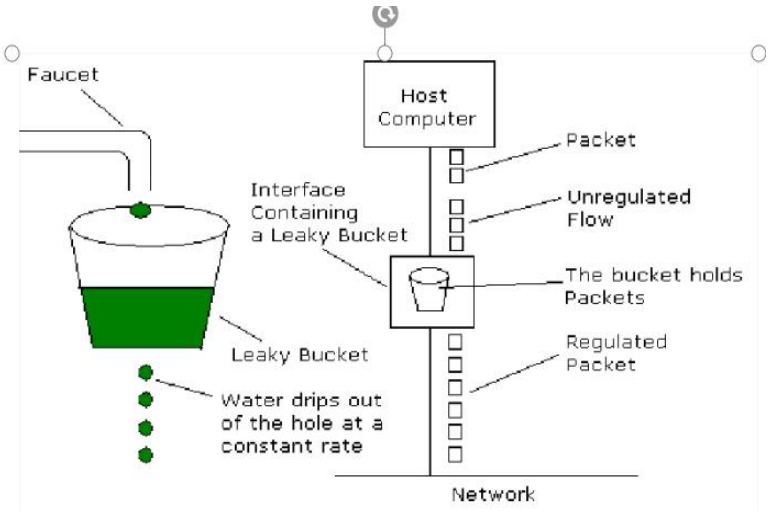
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn 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 thesystem) can handle. For subnets that use virtual circuits internally, these methods can be used at the networklayer.

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, thepacket 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 thehost 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 unevenflow of packet from the user process inside the host into an even flow of packet onto the network, smoothingout bursts and greatly reducing the chances of congestion.



Program:

```
import java.lang.*;
import java.util.Random;
import java.io.*;

import java.util.Scanner;
class leaky_bucket

{

public static void main(String args[])

{

        int drop=0,mini,nsec,p_remain=0;
        int o_rate,b_size,i,packet[];

        packet = new int[100];

        Scanner in = new Scanner(System.in);

        System.out.println("Enter bucket size:");
        b_size = in.nextInt();

        System.out.println("Enter the output rate:");
        o_rate = in.nextInt();

        System.out.println("Enter the number of seconds you want to simulate:");
        nsec = in.nextInt();

        Random rand = new Random();
        for(i=0;i<nsec;i++)

                packet[i]=((rand.nextInt(9)+1)*10);
```

```
System.out.println("Seconds|packets received|packets sent|packets left|packets dropped");
System.out.println("-------------------------------------------------------------------------------------------");
for(i=0;i<nsec;i++)


{

        p_remain+=packet[i];
        if(p_remain>b_size)

        {

                drop=p_remain-b_size;
                p_remain=b_size;
                System.out.print(i+1+"                  ");

                System.out.print(packet[i]+" ");
                mini=Math.min(p_remain,o_rate);
                System.out.print(mini+"                 ");
                p_remain=p_remain-mini;
                System.out.print(p_remain+"             ");

                System.out.print(drop+"                 ");
                System.out.println();

                drop=0;

        }

}

while(p_remain!=0)

{

        if(p_remain>b_size)

        {

                drop=p_remain-b_size;
                p_remain=b_size;

        }

        mini=Math.min(p_remain,o_rate);
```

```
                System.out.print("              "+p_remain+"          "+mini);
                p_remain=p_remain-mini;

                System.out.println(p_remain+"                 "+drop);
                drop=0;

        }
}

}
```

Output:

```
C:\Users\abhijith\Desktop>javac leaky_bucket.java

C:\Users\abhijith\Desktop>java leaky_bucket
Enter bucket size:
10
Enter the output rate:
4
Enter the number of seconds you want to simulate:
10
Seconds|packets received|packets sent|packets left|packets dropped
-------------------------------------------------------------------------
1               90              4               6               80
2               20              4               6               16
3               50              4               6               46
4               90              4               6               86
5               70              4               6               66
6               40              4               6               36
7               70              4               6               66
8               90              4               6               86
9               10              4               6               6
10              70              4               6               66
                6               42              0
                2               20              0
```