

RUBRIC SHEET**DAILY CONDUCTION (Max: 15 Marks)**

Sl. No	Experiment Name	Write-Up & Implementation 3 Marks	Analysis & Execution 4 Marks	Results & Tabulation 3 Marks	Record 5 Marks	Total 15 Marks
1	Write a program for error detecting code using CRC-CCITT (16- bits).					
2	Develop a program to implement a sliding window protocol in the data link layer.					
3	Develop a program to find the shortest path between vertices using the Bellman-Ford and path vector routing algorithm.					
4	Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.					
5	Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth, and find the number of packets dropped.					
6	Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.					
7	Develop a program for a simple RSA algorithm to encrypt and decrypt the data.					
8	Develop a program for congestion control using a leaky bucket algorithm					
9	Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.					
10	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.					

Program 2

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

Program:

```
set ns [new Simulator]
set tf [open lab2.tr w]
$ns trace-all $tf

set nf [open lab2.nam w]
$ns namtrace-all $nf

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]

$n0 label "Ping0"
$n4 label "Ping4"
$n5 label "Ping5"
$n6 label "Ping6"
$n2 label "Router"

$ns color 1 "red"
$ns color 2 "green"

$ns duplex-link $n0 $n2 100Mb 300ms DropTail
$ns duplex-link $n1 $n2 1Mb 300ms DropTail
$ns duplex-link $n3 $n2 1Mb 300ms DropTail
$ns duplex-link $n5 $n2 100Mb 300ms DropTail
$ns duplex-link $n2 $n4 1Mb 300ms DropTail
$ns duplex-link $n2 $n6 1Mb 300ms DropTail

$ns queue-limit $n0 $n2 5
$ns queue-limit $n2 $n4 3
$ns queue-limit $n2 $n6 2
$ns queue-limit $n5 $n2 5
```

#The below code is used to connect between the ping agents to the node n0, n4 , n5 and n6.

```
set ping0 [new Agent/Ping]
$ns attach-agent $n0 $ping0

set ping4 [new Agent/Ping]
$ns attach-agent $n4 $ping4
set ping5 [new Agent/Ping]
$ns attach-agent $n5 $ping5

set ping6 [new Agent/Ping]
$ns attach-agent $n6 $ping6

$ping0 set packetSize_ 50000
$ping0 set interval_ 0.0001
$ping5 set packetSize_ 60000
$ping5 set interval_ 0.00001

$ping0 set class_ 1
$ping5 set class_ 2

$ns connect $ping0 $ping4
$ns connect $ping5 $ping6
```

#Define a 'recv' function for the class 'Agent/Ping'

#The below function is executed when the ping agent receives a reply from the #destination

```
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts " The node [$node_ id] received an reply from $from with round trip time of $rtt"
}
proc finish {} {
global ns nf tf
exec nam lab4.nam &
exec echo "The total number of packet dropped due to congestion is:" &
exec grep -c "^d" lab4.tr &
$ns flush-trace
close $nf
close $tf
exit 0
}
```

#Schedule events

\$ns at 0.1 "\$ping0 send"
\$ns at 0.2 "\$ping0 send"
\$ns at 0.3 "\$ping0 send"
\$ns at 0.4 "\$ping0 send"
\$ns at 0.5 "\$ping0 send"
\$ns at 0.6 "\$ping0 send"
\$ns at 0.7 "\$ping0 send"
\$ns at 0.8 "\$ping0 send"
\$ns at 0.9 "\$ping0 send"
\$ns at 1.0 "\$ping0 send"
\$ns at 1.1 "\$ping0 send"
\$ns at 1.2 "\$ping0 send"
\$ns at 1.3 "\$ping0 send"
\$ns at 1.4 "\$ping0 send"
\$ns at 1.5 "\$ping0 send"
\$ns at 1.6 "\$ping0 send"
\$ns at 1.7 "\$ping0 send"
\$ns at 1.8 "\$ping0 send"
\$ns at 0.1 "\$ping5 send"
\$ns at 0.2 "\$ping5 send"
\$ns at 0.3 "\$ping5 send"
\$ns at 0.4 "\$ping5 send"
\$ns at 0.5 "\$ping5 send"
\$ns at 0.6 "\$ping5 send"
\$ns at 0.7 "\$ping5 send"
\$ns at 0.8 "\$ping5 send"
\$ns at 0.9 "\$ping5 send"
\$ns at 1.0 "\$ping5 send"
\$ns at 1.1 "\$ping5 send"
\$ns at 1.2 "\$ping5 send"
\$ns at 1.3 "\$ping5 send"
\$ns at 1.4 "\$ping5 send"
\$ns at 1.5 "\$ping5 send"
\$ns at 1.6 "\$ping5 send"
\$ns at 1.7 "\$ping5 send"
\$ns at 1.8 "\$ping5 send"
\$ns at 5.0 "finish"
\$ns run

OUTPUT:

Program 3

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

Experiment Specific Instructions

1. To analyze the given problem you have to write a Tcl script and simulate with ns2
2. Begin by specifying the trace files and the nam files to be created
3. Define a finish procedure
4. Determine and create the nodes that will be used to create the topology. Here in our experiment we are selecting 6 nodes namely 0, 1, 2, 3, 4, 5
5. Create the links to connect the nodes
6. Set up the LAN by specifying the nodes, and assign values for bandwidth, delay, queue type and channel to it
7. Optionally you can position and orient the nodes and links to view a nice video output with Nam
8. Set up the TCP and/or UDP connection(s) and the FTP/CBR (or any other application) that will run over it
9. Schedule the different events like simulation start and stop, data transmission start and stop
10. Call the finish procedure and mention the time at what time your simulation will end
11. Execute the script with ns

Steps for execution

- Open gedit editor and type program. Program name should have the extension “.tcl”
 - `[root@localhost ~]# gedit lab3.tcl / vi lab3.tcl`
- Save the program.
- Run the simulation program
 - `[root@localhost ~]# ns lab6.tcl`
- To see the trace file contents open the file as,
 - `[root@localhost ~]# gedit lab6.tr / vi lab6.tr`
- To see the graph contents open the file as,
 - `[root@localhost ~]# xgraph congestion1.xg`
 - `[root@localhost ~]# xgraph congestion2.xg`

Program**#set ns Simulator**

```
set ns [new Simulator]
```

#define color for data flow

```
$ns color 1 Blue
```

```
$ns color 2 Red
```

#open trace file

```
set tracefile1 [open lab6.tr w]
```

```
set winfile [open winfile w]
```

```
$ns trace-all $tracefile1
```

#open namtrace file

```
set namfile [open lab6.nam w]
```

```
$ns namtrace-all $namfile
```

#define finish procedure

```
proc finish { } {  
    global ns tracefile1 namfile  
    $ns flush-trace  
    close $tracefile1  
    close $namfile  
    exec nam lab6.nam &  
    exit 0  
}
```

#create 6 nodes

```
set n0 [$ns node]
```

```
set n1 [$ns node]
```

```
set n2 [$ns node]
```

```
set n3 [$ns node]
```

```
set n4 [$ns node]
```

```
set n5 [$ns node]
```

```
$n1 shape box
```

#create link between nodes

```
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
```

```
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
```

```
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
```

```
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail
```

```
set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/802_3]
```

```
#give node position
```

```
$ns duplex-link-op $n0 $n2 orient right-down
```

```
$ns duplex-link-op $n1 $n2 orient right-up
```

```
$ns simplex-link-op $n3 $n2 orient left
```

```
$ns simplex-link-op $n2 $n3 orient right
```

```
#set queue size of link(n2-n3)
```

```
$ns queue-limit $n2 $n3 20
```

```
#setup tcp connection
```

```
set tcp [new Agent/TCP]
```

```
$ns attach-agent $n0 $tcp
```

```
set sink [new Agent/TCPSink]
```

```
$ns attach-agent $n4 $sink
```

```
$ns connect $tcp $sink
```

```
$tcp set fid_ 1
```

```
$tcp set packetSize_ 552
```

```
#set ftp over tcp connection
```

```
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
```

```
#setup a TCP1 connection
```

```
set tcp1 [new Agent/TCP]
```

```
$ns attach-agent $n1 $tcp1
```

```
set sink1 [new Agent/TCPSink]
```

```
$ns attach-agent $n5 $sink1
```

```
$ns connect $tcp1 $sink1
```

```
$tcp1 set fid_ 2
```

```
$tcp1 set packetSize_ 552
```

```
set telnet0 [new Application/Telnet]
```

```
$telnet0 attach-agent $tcp1
```

```
#title congestion window1
```

```
set outfile1 [open congestion1.xg w]
```

```
puts $outfile1 "TitleText: Congestion Window-- Source _tcp"
```

```
puts $outfile1 "xUnitText: Simulation Time(Secs)"
```

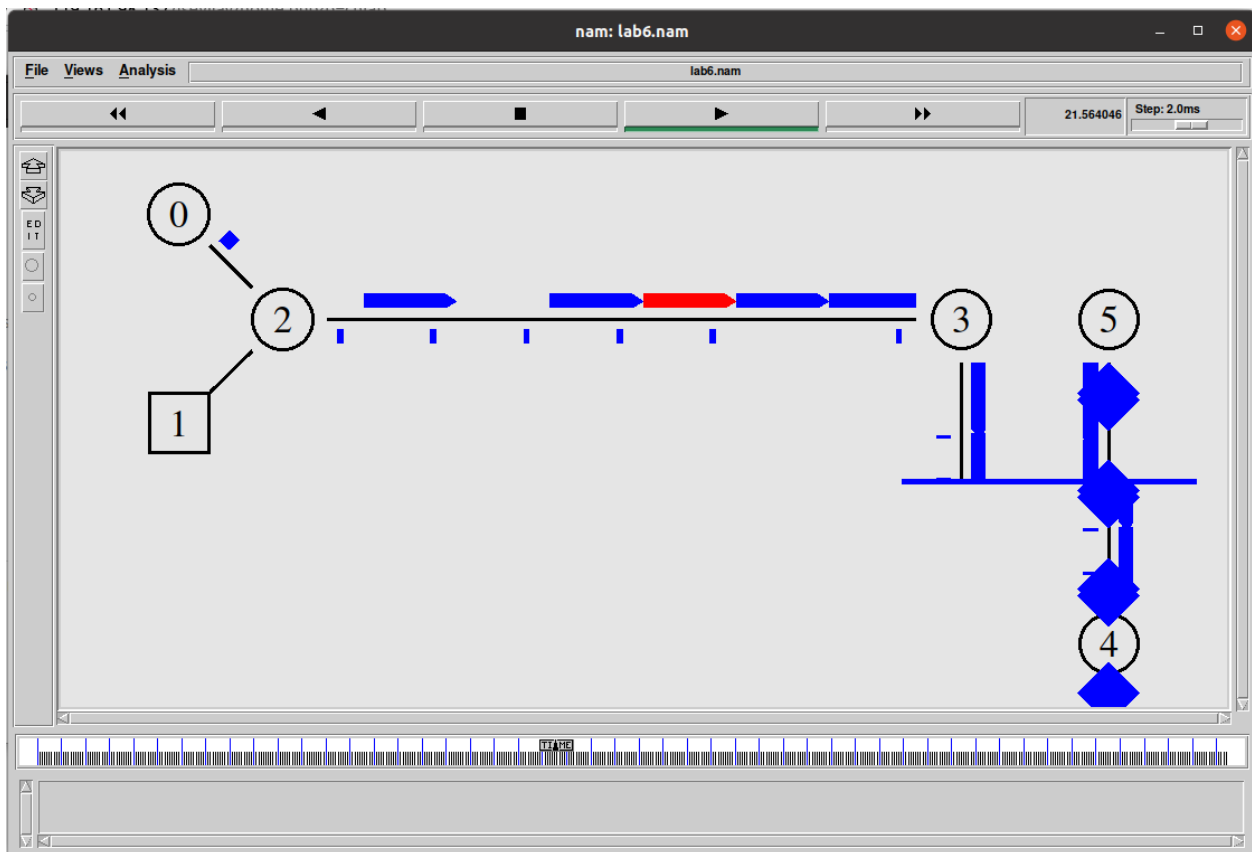
```
puts $outfile1 "yUnitText: Congestion WindowSize"
```


#title congestion window2

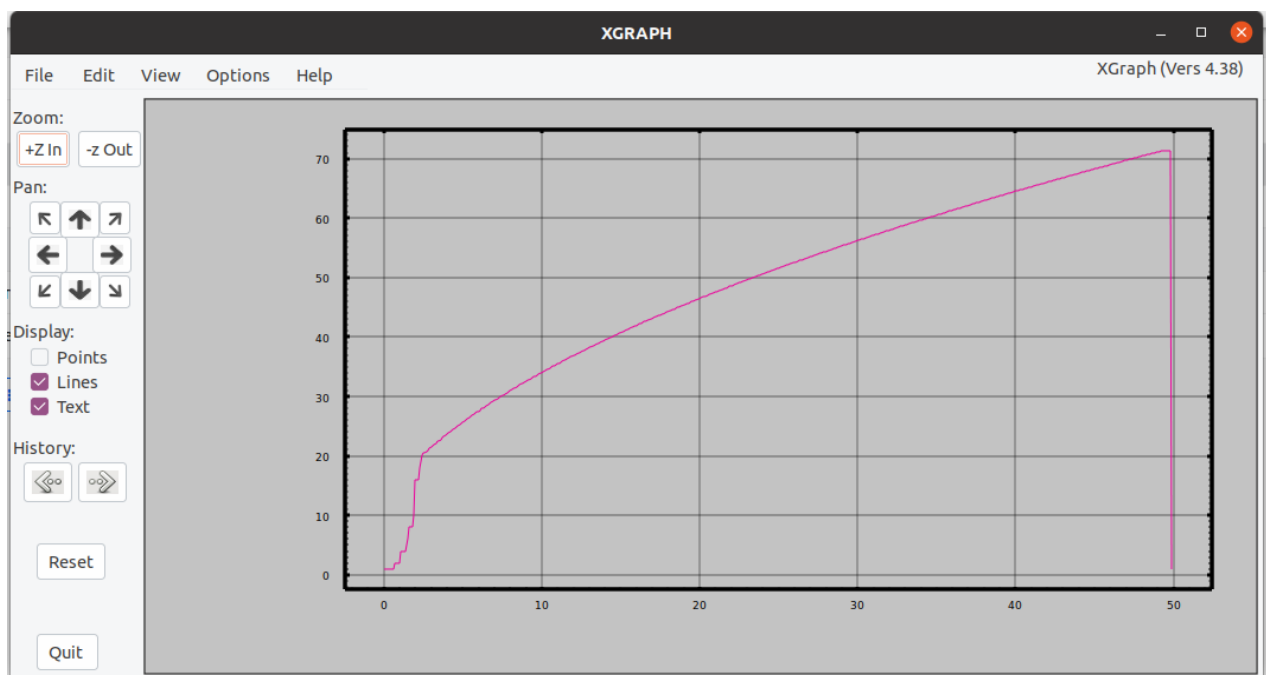
```
set outfile2 [open congestion2.xg w]
puts $outfile2 "TitleText: Congestion Window-- Source _tcp1"
puts $outfile2 "xUnitText: Simulation Time(Secs)"
puts $outfile2 "yUnitText: Congestion WindowSize"

proc plotWindow {tcpSource outfile} {
    global ns
    set time 0.1
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
    puts $outfile "$now $cwnd"
    $ns at [expr $now+$time] "plotWindow $tcpSource $outfile"
}

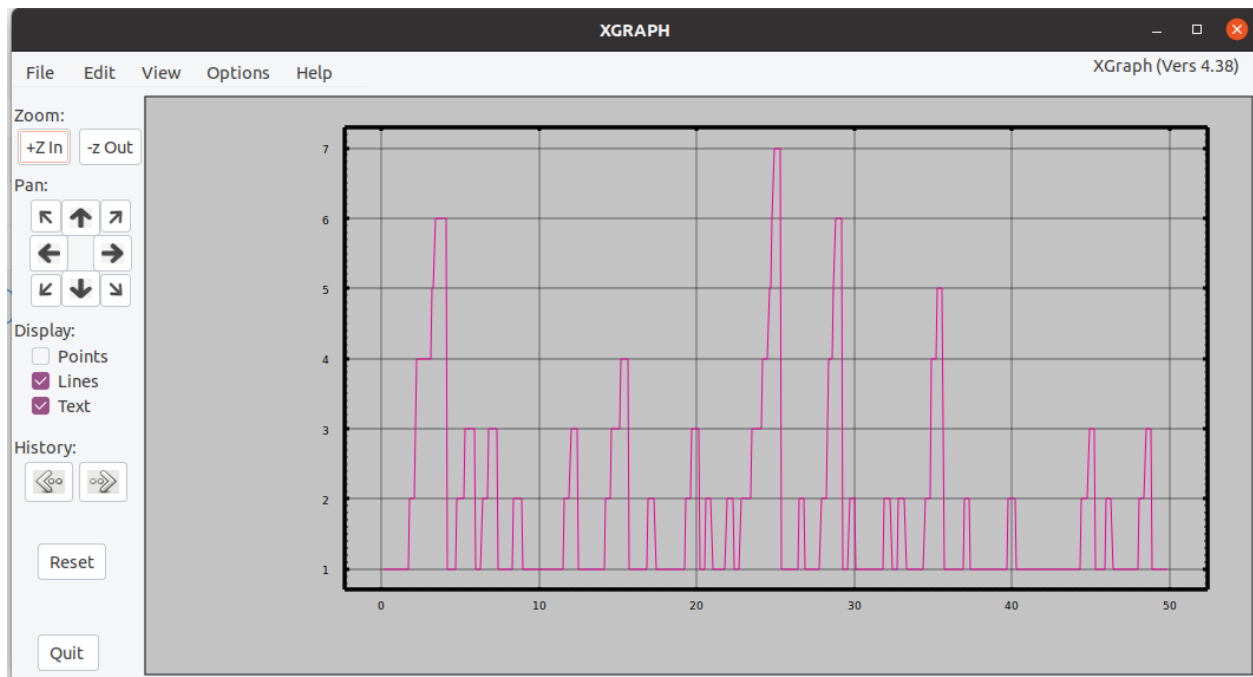
$ns at 0.1 "plotWindow $tcp $winfile"
$ns at 0.0 "plotWindow $tcp $outfile1"
$ns at 0.1 "plotWindow $tcp1 $outfile2"
$ns at 0.3 "$ftp start"
$ns at 0.5 "$telnet0 start"
$ns at 49.0 "$ftp stop"
$ns at 49.1 "$telnet0 stop"
$ns at 50.0 "finish"
$ns run
```

Topology:**Congestion graph**

```
ise@Ubuntu20:~/cnlab$ xgraph congestion1.xg
```



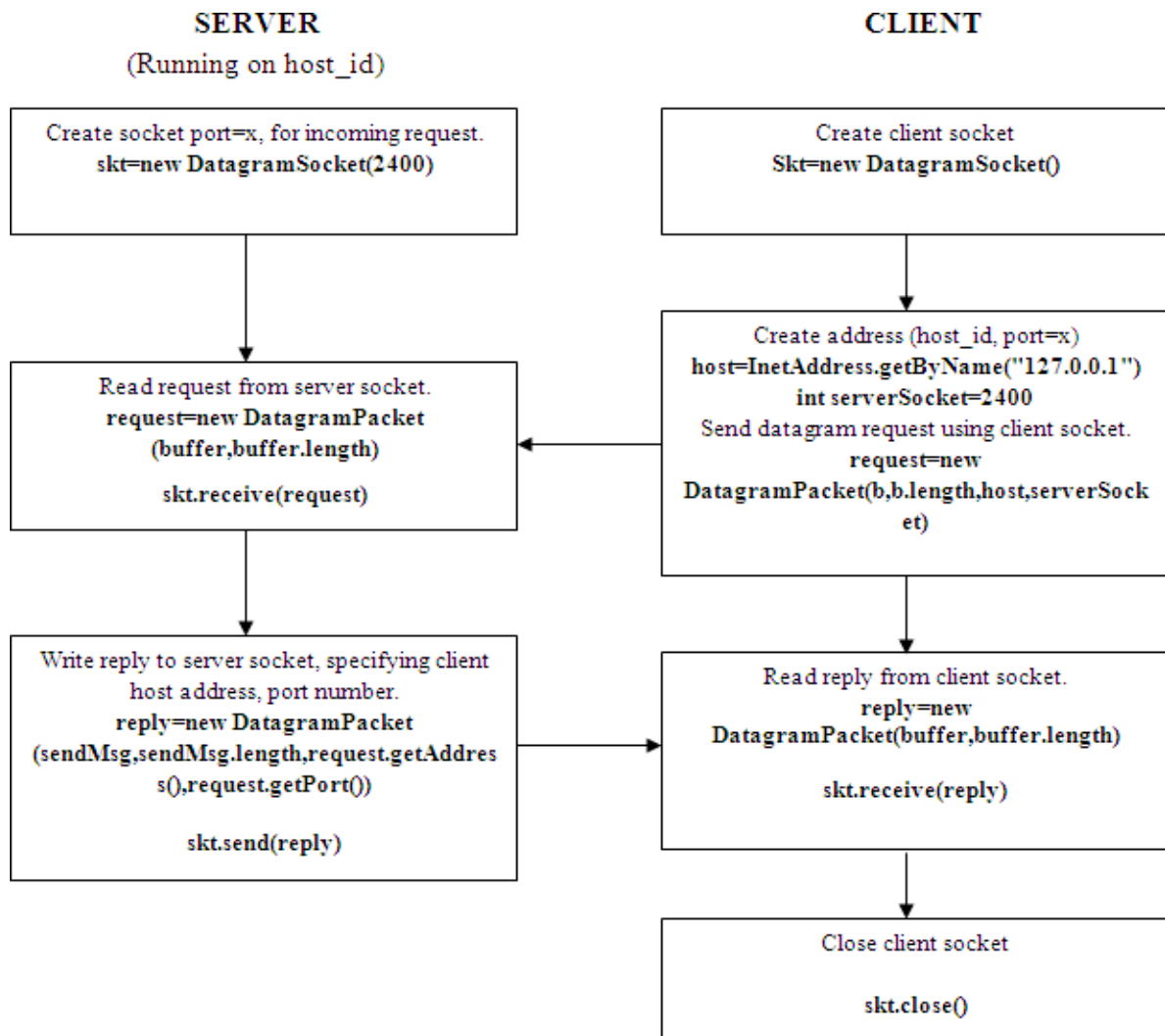
```
ise@Ubuntu20:~/cnlab$ xgraph congestion2.xg
```



lab6.tr ~/cnlab									
1	+	0.3	0	2	tcp	40	-----	1	0.0 4.0 0 0
2	-	0.3	0	2	tcp	40	-----	1	0.0 4.0 0 0
3	r	0.31016	0	2	tcp	40	-----	1	0.0 4.0 0 0
4	+	0.31016	2	3	tcp	40	-----	1	0.0 4.0 0 0
5	-	0.31016	2	3	tcp	40	-----	1	0.0 4.0 0 0
6	r	0.411227	2	3	tcp	40	-----	1	0.0 4.0 0 0
7	h	0.411227	3	6	tcp	40	-----	1	0.0 4.0 0 0
8	+	0.451227	3	6	tcp	40	-----	1	0.0 4.0 0 0
9	-	0.451227	3	6	tcp	40	-----	1	0.0 4.0 0 0
10	d	0.451231	5	6	tcp	40	-----	1	0.0 4.0 0 0
11	r	0.491231	6	4	tcp	40	-----	1	0.0 4.0 0 0
12	h	0.491231	4	6	ack	40	-----	1	4.0 0.0 0 1
13	+	0.531231	4	6	ack	40	-----	1	4.0 0.0 0 1
14	-	0.531231	4	6	ack	40	-----	1	4.0 0.0 0 1
15	d	0.531235	5	6	ack	40	-----	1	4.0 0.0 0 1
16	r	0.571235	6	3	ack	40	-----	1	4.0 0.0 0 1
17	+	0.571235	3	2	ack	40	-----	1	4.0 0.0 0 1
18	-	0.571235	3	2	ack	40	-----	1	4.0 0.0 0 1
19	r	0.672301	3	2	ack	40	-----	1	4.0 0.0 0 1
20	+	0.672301	2	0	ack	40	-----	1	4.0 0.0 0 1
21	-	0.672301	2	0	ack	40	-----	1	4.0 0.0 0 1
22	r	0.682461	2	0	ack	40	-----	1	4.0 0.0 0 1
23	+	0.682461	0	2	tcp	592	-----	1	0.0 4.0 1 2
24	-	0.682461	0	2	tcp	592	-----	1	0.0 4.0 1 2
25	+	0.682461	0	2	tcp	592	-----	1	0.0 4.0 2 3
26	-	0.684829	0	2	tcp	592	-----	1	0.0 4.0 2 3
27	r	0.694829	0	2	tcp	592	-----	1	0.0 4.0 1 2
28	+	0.694829	2	3	tcp	592	-----	1	0.0 4.0 1 2
29	-	0.694829	2	3	tcp	592	-----	1	0.0 4.0 1 2
30	r	0.697197	0	2	tcp	592	-----	1	0.0 4.0 2 3
31	+	0.697197	2	3	tcp	592	-----	1	0.0 4.0 2 3
32	-	0.710616	2	3	tcp	592	-----	1	0.0 4.0 2 3
33	r	0.810616	2	3	tcp	592	-----	1	0.0 4.0 1 2
34	h	0.810616	3	6	tcp	592	-----	1	0.0 4.0 1 2
35	r	0.826403	2	3	tcp	592	-----	1	0.0 4.0 2 3
36	h	0.826403	2	6	tcp	592	-----	1	0.0 4.0 2 3

Program 8

Develop a program on a datagram socket for client/server to display the messages on client side, typed at the server side.



UDP client/server communication flow.

Methods and description

- **DatagramSocket(int port)** throws **SocketEeption**: it creates a datagram socket and binds it with the given Port Number.
- **DatagramPacket(byte[] buffer, int length)**: it creates a datagram packet. This constructor is used to receive the packets.
- **DatagramPacket(byte[] buffer, int length, InetAddress address, int port)**: it creates a datagram packet. This constructor is used to send the packets.

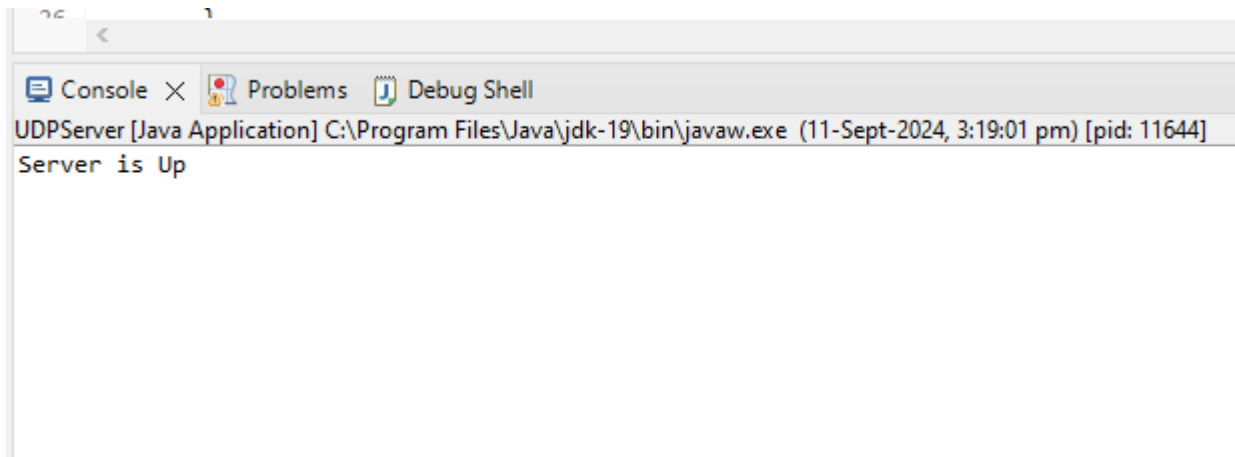
Program:**Client Side:**

```
package pgm8;
import java.io.*;
import java.net.*;
import java.net.InetAddress;
class UDPClient
{
    public static void main(String[] args) throws Exception
    {
        BufferedReader inFromUser=new BufferedReader(new
InputStreamReader(System.in));
        DatagramSocket clientSocket=new DatagramSocket();
        InetAddress
IPAddress=InetAddress.getByName("localhost");
        byte[] sendData=new byte[1024];
        byte[] receiveData=new byte[1024];
        System.out.println("Enter the sting to be converted in
to Upper case");
        String sentence=inFromUser.readLine();
        sendData=sentence.getBytes();
        DatagramPacket sendPacket=new

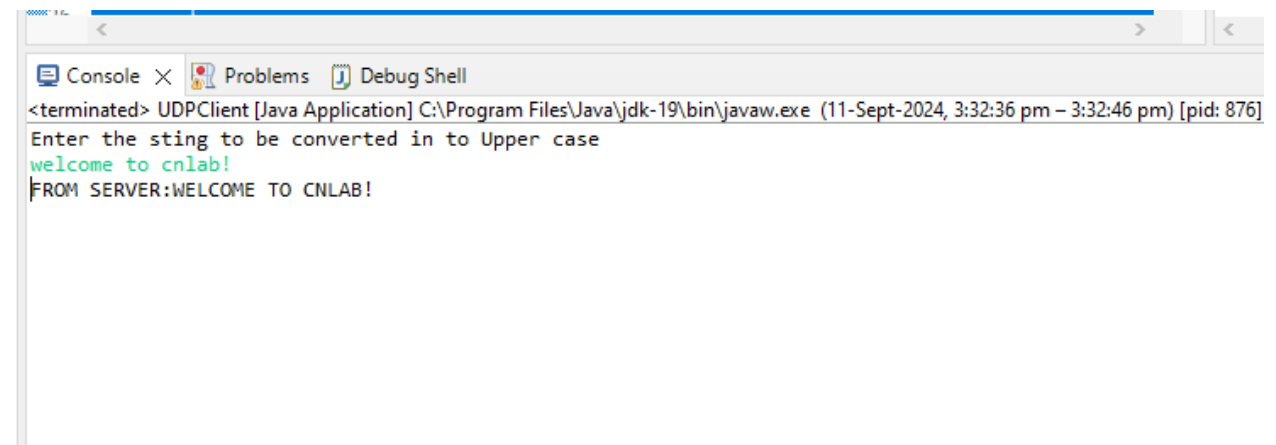
        DatagramPacket(sendData,sendData.length,IPAddress,9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket=new
DatagramPacket(receiveData,receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence=new
String(receivePacket.getData());
        System.out.println("FROM SERVER:"+modifiedSentence);
        clientSocket.close();
    }
}
```

Server Side:

```
package pgm8;
import java.net.*;
import java.net.InetAddress;
class UDPServer
{
    public static void main(String args[]) throws Exception
    {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData=new byte[1024];
        byte[] sendData=new byte[1024];
        while(true)
        {
            System.out.println("Server is Up");
            DatagramPacket receivePacket=new
DatagramPacket(receiveData,receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence=new
String(receivePacket.getData());
            System.out.println("RECEIVED:"+sentence);
            InetAddress IPAddress=receivePacket.getAddress();
            int port=receivePacket.getPort();
            String capitalizedSentence=sentence.toUpperCase();
            sendData=capitalizedSentence.getBytes();
            DatagramPacket sendPacket=new
DatagramPacket(sendData,sendData.length,IPAddress,port);
            serverSocket.send(sendPacket);
        }
    }
}
```

OUTPUT:

```
UDPServer [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (11-Sept-2024, 3:19:01 pm) [pid: 11644]
Server is Up
```



```
<terminated> UDPClient [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (11-Sept-2024, 3:32:36 pm - 3:32:46 pm) [pid: 876]
Enter the sting to be converted in to Upper case
welcome to cnlab!
FROM SERVER:WELCOME TO CNLAB!
```

Program 9

Develop a program for a simple RSA algorithm to encrypt and decrypt the data.

Cryptography has a long and colorful history. The message to be encrypted, known as the plaintext, are transformed by a function that is parameterized by a key. The output of the encryption process, known as the ciphertext, is then transmitted, often by messenger or radio. The enemy, or intruder, hears and accurately copies down the complete ciphertext. However, unlike the intended recipient, he does not know the decryption key and so cannot decrypt the ciphertext easily. The art of breaking ciphers is called cryptanalysis the art of devising ciphers (cryptography) and breaking them (cryptanalysis) is collectively known as cryptology.

There are several ways of classifying cryptographic algorithms. They are generally categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms are as follows:

1. Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption. It is also known as symmetric cryptography.
2. Public Key Cryptography (PKC): Uses one key for encryption and another for decryption. It is also known as asymmetric cryptography.
3. Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information

Public-key cryptography has been said to be the most significant new development in cryptography. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it does not matter which key is applied first, but that both keys are required for the process to work. Because pair of keys is required, this approach is also called asymmetric cryptography.

In PKC, one of the keys is designated the public key and may be advertised as widely as the owner wants. The other key is designated the private key and is never revealed to another party. It is straight forward to send messages under this scheme.

The RSA algorithm is named after Ron Rivest, Adi Shamir and Len Adleman, who invented it in 1977. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers.

Algorithm

1. Generate two large random primes, P and Q , of approximately equal size.
2. Compute $N = P \times Q$
3. Compute $Z = (P-1) \times (Q-1)$.
4. Choose an integer E , $1 < E < Z$, such that $\text{GCD}(E, Z) = 1$
5. Compute the secret exponent D , $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$
6. The public key is (N, E) and the private key is (N, D) .

Note: The values of P , Q , and Z should also be kept secret.

The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. Select primes $P=11$, $Q=3$.
2. $N = P \times Q = 11 \times 3 = 33$
 $Z = (P-1) \times (Q-1) = 10 \times 2 = 20$
3. Lets choose $E=3$
Check $\text{GCD}(E, P-1) = \text{GCD}(3, 10) = 1$ (i.e. 3 and 10 have no common factors except 1),
and check $\text{GCD}(E, Q-1) = \text{GCD}(3, 2) = 1$
therefore $\text{GCD}(E, Z) = \text{GCD}(3, 20) = 1$

Program:

```
package pgm9;

import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;

public class RSA
{
    private BigInteger p,q,N,phi,e,d;
    private int        bitlength = 48;
    private Random      r;

    public RSA()
    {
        r = new Random();
        p = BigInteger.probablePrime(bitlength, r);
        q = BigInteger.probablePrime(bitlength, r);
        System.out.println("Prime number p is"+p);
        System.out.println("prime number q is"+q);
        N = p.multiply(q);
        phi =
p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitlength / 2, r);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 &&
e.compareTo(phi) < 0)
        {
            e.add(BigInteger.ONE);
        }
        System.out.println("Public key is"+e);
        d = e.modInverse(phi);
        System.out.println("Private key is"+d);
    }

    public RSA(BigInteger e, BigInteger d, BigInteger N)
```

```
{
    this.e = e;
    this.d = d;
    this.N = N;
}

public static void main(String[] args) throws IOException
{
    RSA rsa = new RSA();
    DataInputStream in = new DataInputStream(System.in);
    String teststring;
    System.out.println("Enter the plain text:");
    teststring = in.readLine();
    System.out.println("Encrypting String: " + teststring);
    System.out.println("String in Bytes: " +
bytesToString(teststring.getBytes()));
    // encrypt
    byte[] encrypted = rsa.encrypt(teststring.getBytes());
    // decrypt
    byte[] decrypted = rsa.decrypt(encrypted);
    System.out.println("Decrypting Bytes: " +
bytesToString(decrypted));
    System.out.println("Decrypted String: " + new
String(decrypted));
}

private static String bytesToString(byte[] encrypted)
{
    String test = "";
    for (byte b : encrypted)
    {
        test += Byte.toString(b);
    }
    return test;
}
```

```
// Encrypting message
public byte[] encrypt(byte[] message)
{

    return (new BigInteger(message)).modPow(e, N).toByteArray();
}

// Decrypting message
public byte[] decrypt(byte[] message)
{
    return (new BigInteger(message)).modPow(d, N).toByteArray();
}
}
```

OUTPUT:

```
Prime number p is253073860231303
prime number q is258382793336351
Public key is14869111
Private key is28229870077256696875375113691
Enter the plain text:
```

cnlab

```
Encrypting String: cnlab
String in Bytes: 991101089798
Decrypting Bytes: 991101089798
Decrypted String: cnlab
```

```
Prime number p is59580106145912050719652700059
Prime number q is44006362862628452357432506297
Public key is183279081972809
Private key
is7351277287075139659808324247455467514791618395559058072
57
```

```
Enter the plain text:
```

```
RSA Algorithm
```

```
Encrypting String: RSA Algorithm
String in Bytes: 8283653265108103111114105116104101109
Decrypting Bytes: 8283653265108103111114105116104101109
Decrypted String: RSA Algorithm
```

Program 10

Develop a program for congestion control using a 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 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.

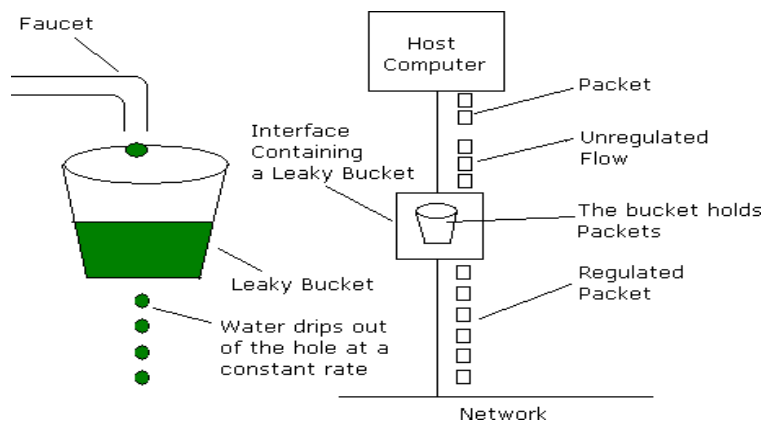
In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn 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 simulated 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.



Program:

```
package cnlab;

import java.util.Scanner;

public class Leakybucket
{
    public static void main(String [] args)
    {
        int i;
        int a[]=new int[20];
        int buck_rem=0, buck_cap=4, rate=3, sent, recv;
        Scanner in=new Scanner(System.in);
        System.out.println("Enter the number of Packets");
        int n=in.nextInt();
        System.out.println("Enter the Packets");
        for(i=1;i<=n;i++)
            a[i]=in.nextInt();
        System.out.println("Clock \t Packet size \t accept \t
sent \t remaining");
        for(i=1;i<=n;i++)
        {
            if(a[i]!=0)
            {
                if(buck_rem+a[i]>buck_cap)
                    recv=-1;
                else
                {

```

```
        recv=a[i];
        buck_rem+=a[i];
    }
}
else
    recv=0;
    if(buck_rem!=0)
    {
        if(buck_rem<rate)
        {
            sent=buck_rem;
            buck_rem=0;
        }
        else
        {
            sent=rate;
            buck_rem=buck_rem-rate;
        }
    }
    else
        sent=0;
    if(recv== -1)
        System.out.println(+i+ "\t\t" +a[i]+ "\t
dropped \t" + sent + "\t" +buck_rem);
    else
        System.out.println(+i+ "\t\t" +a[i] +"\t\t"
+recv +"\t" +sent +"\t" +buck_rem);
    }
}
}
```


OUTPUT:

Enter the number of packets

3

Enter the packets

4

3

2

Clock	packet size	accept	sent	remaining
1	4	4	3	1
2	3	3	3	1
3	2	2	3	0

Enter the number of packets

4

Enter the packets

4

3

2

5

Clock	packet size	accept	sent	remaining
1	4	4	3	1
2	3	3	3	1
3	2	2	3	0
4	5	dropped	0	0