**CAMBRIDGE INSTITUTE OF TECHNOLOGY**

K.R. PURAM, BENGALURU-560036

**Department of Computer Science and Engineering**

**Computer Networks Laboratory Manual**

Prepared By

Dr. Josephine Prem Kumar, Prof. CSE

Prof. Subhashini, Asst. Prof. CSE

Prof. Shilpa V., Asst. Prof. CSE

Department OF CSE

**Laboratory Outcomes:**

The student should be able to:

• Analyze and Compare various networking protocols.

• Demonstrate the working of different concepts of networking.

• Implement, analyze and evaluate networking protocols in NS2 / NS3 and JAVA programming language

|  |  |
| --- | --- |
| Sl.No | Program List (PART A) |
|  | 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. |
| 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. |
| 3. | Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination. |
| 4. | . Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets. |
| 5. | Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment. |
| 6. | Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment |

|  |  |
| --- | --- |
| Sl.No | Program List (PART B) |
| 7. | Write a program for error detecting code using CRC-CCITT (16- bits). |
| 8. | Write a program to find the shortest path between vertices using bellman-ford algorithm. |
| 9. | 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. |
| 10. | Write a program on datagram socket for client/server to display the messages on client side, typed at the server side. |
| 11. | Write a program for simple RSA algorithm to encrypt and decrypt the data. |
| 12. | Write a program for congestion control using leaky bucket algorithm. |

# PART A

# Steps for execution – Program 1

* 1. Open gedit editor and type program. Program name should have the extension “ **.tcl**”

# [root@localhost ~]# gedit lab1.tcl

* 1. Save the program and exit.
  2. Open gedit editor and type **awk** program. Program name should have the extension “**.awk**”

# [root@localhost ~]# gedit lab1.awk

* 1. Save the program and exit.
  2. Run the simulation program

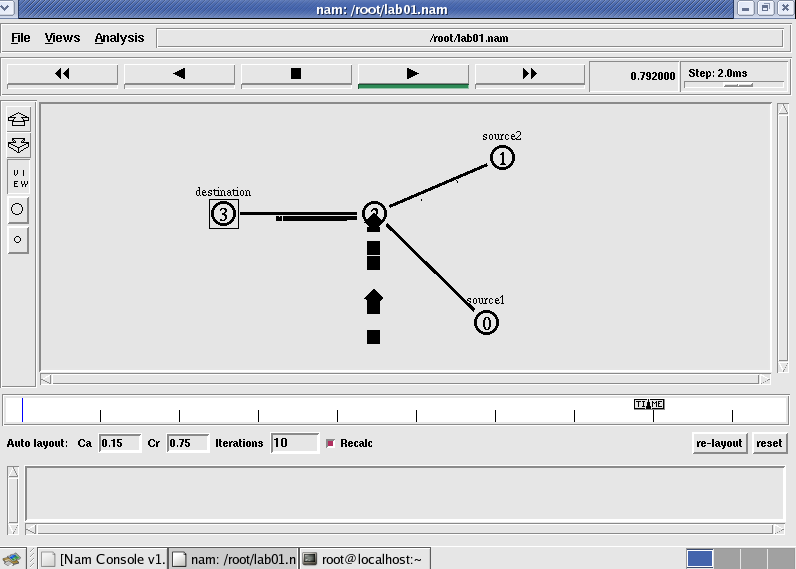
# [root@localhost~]# ns lab1.tcl

* + 1. Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.
    2. Now press the play button in the simulation window and the simulation will begin.
  1. After simulation is completed run **awk file** to see the output,

# [root@localhost~]# awk –f lab1.awk lab1.tr

* 1. To see the trace file contents open the file as,

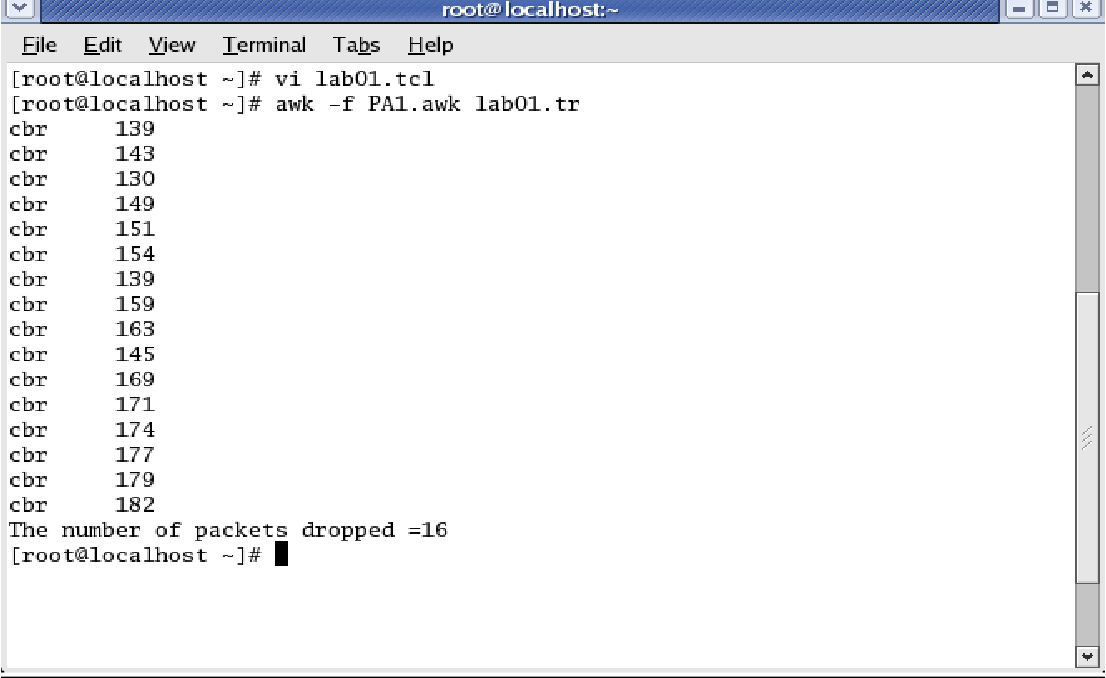
# [root@localhost~]# gedit lab1.tr

**Trace file contains 12 columns:-**

**Event type, Event time, From Node, Source Node, Packet Type, Packet Size, Flags (indicated by --------), Flow ID, Source address, Destination address, Sequence ID, Packet ID**

**Topology**

Output



**Note:**

1. Set the queue size fixed from n0 to n2 as 10, n1-n2 to 10 and from n2-n3 as 5. Syntax: To set the queue size

$ns set queue-limit <from> <to> <size>

Eg: $ns set queue-limit $n0 $n2 10

1. Go on varying the bandwidth from 10, 20 30 . . and find the number of packets dropped at the node2

# Steps for execution – Program 2

* 1. Open gedit editor and type program. Program name should have the extension “ **.tcl**”

# [root@localhost ~]# gedit lab2.tcl

* 1. Save the program and exit.
  2. Open gedit editor and type **awk** program. Program name should have the extension “**.awk**”

# [root@localhost ~]# gedit lab2.awk

* 1. Save the program and exit.
  2. Run the simulationprogram

# [root@localhost~]# ns lab2.tcl

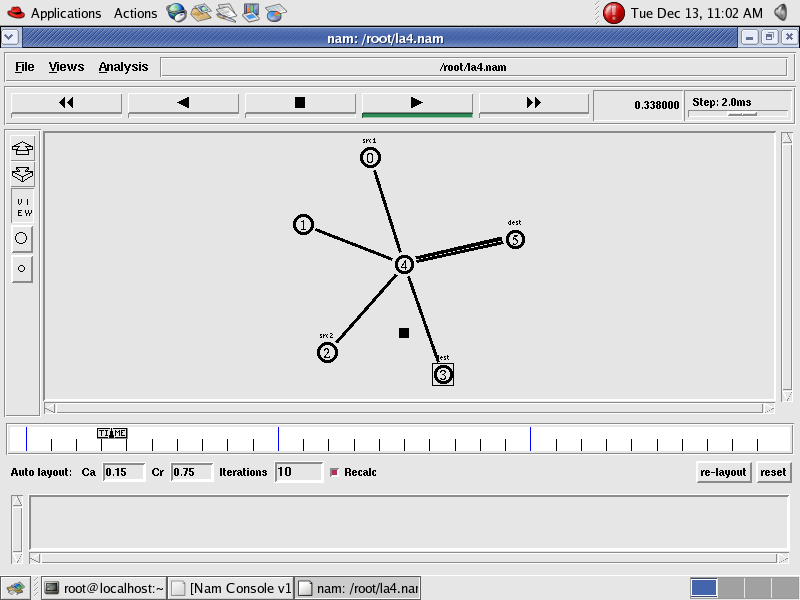
* + 1. Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.
    2. Now press the play button in the simulation window and the simulation will begins.
  1. After simulation is completed run **awk file** to see the output,

# [root@localhost~]# awk –f lab2.awk lab2.tr

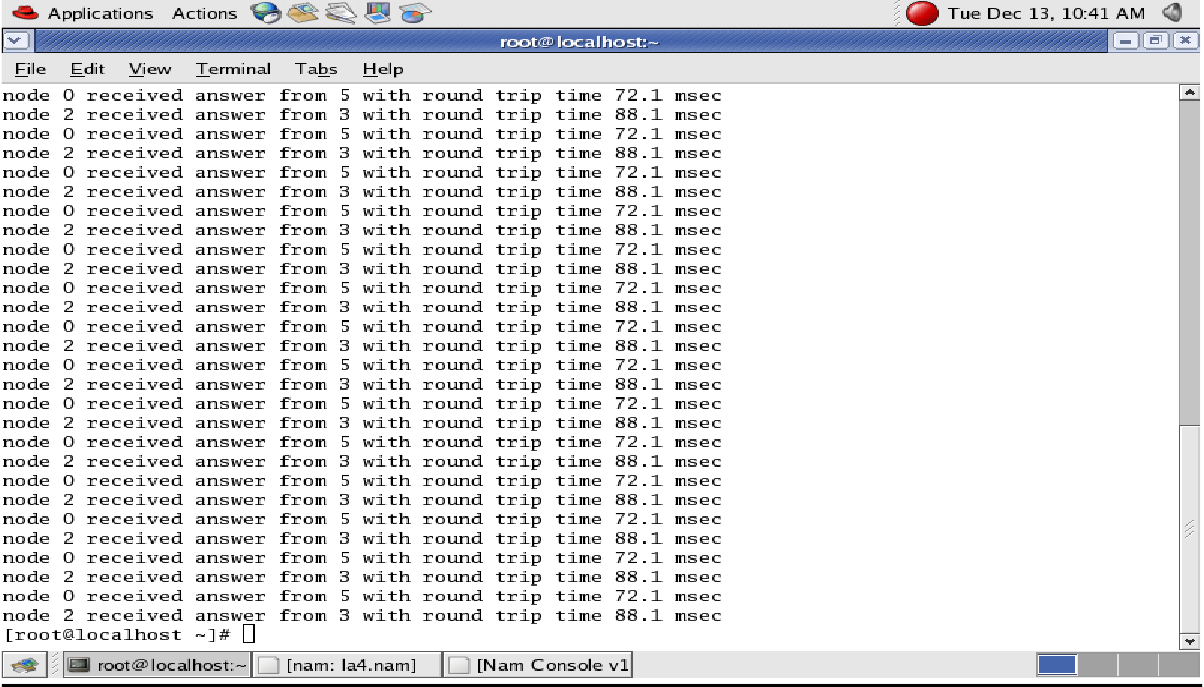
* 1. To see the trace file contents open the file as,

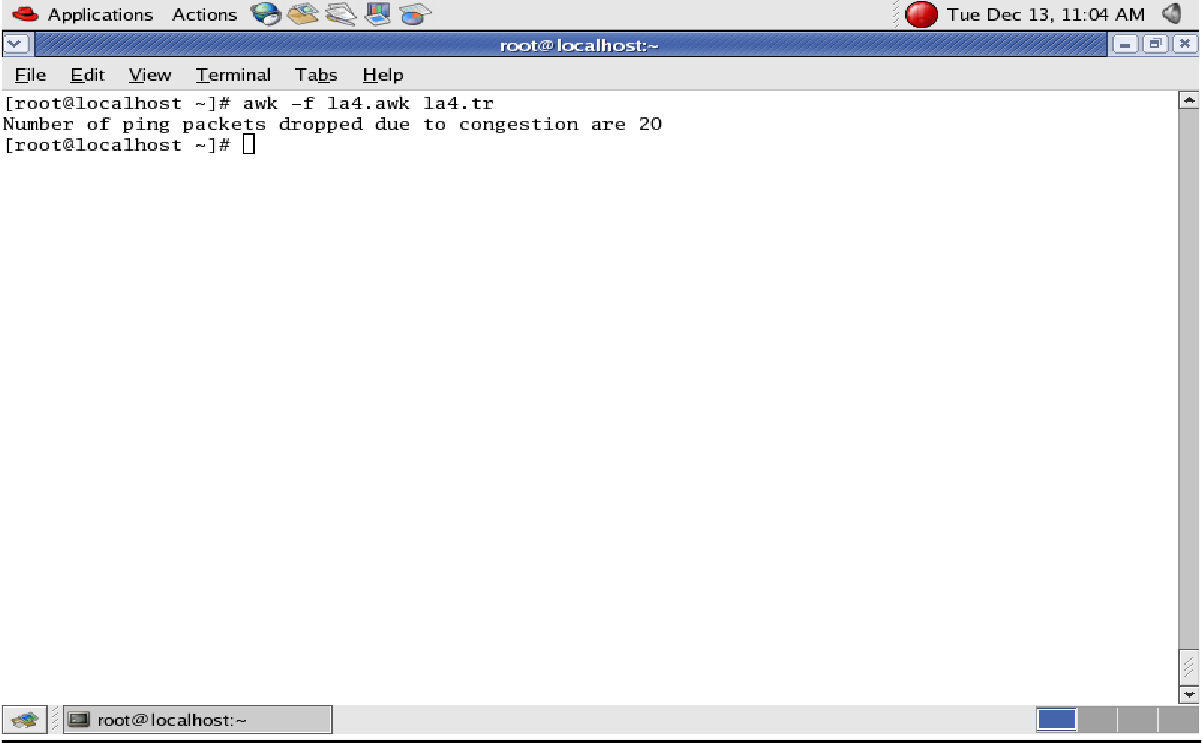
# [root@localhost~]# gedit lab2.tr

**Topology**



**OutputCRC**





# Steps for execution Program 3

1. Open gedit editor and type program. Program name should have the extension “ **.tcl**”

# [root@localhost ~]# gedit lab3.tcl

1. Save the program and exit.
2. Open gedit editor and type **awk** program. Program name should have the extension “**.awk**”

# [root@localhost ~]# gedit lab3.awk

# Save the program and exit

1. Run the simulation program

# [root@localhost~]# ns lab3.tcl

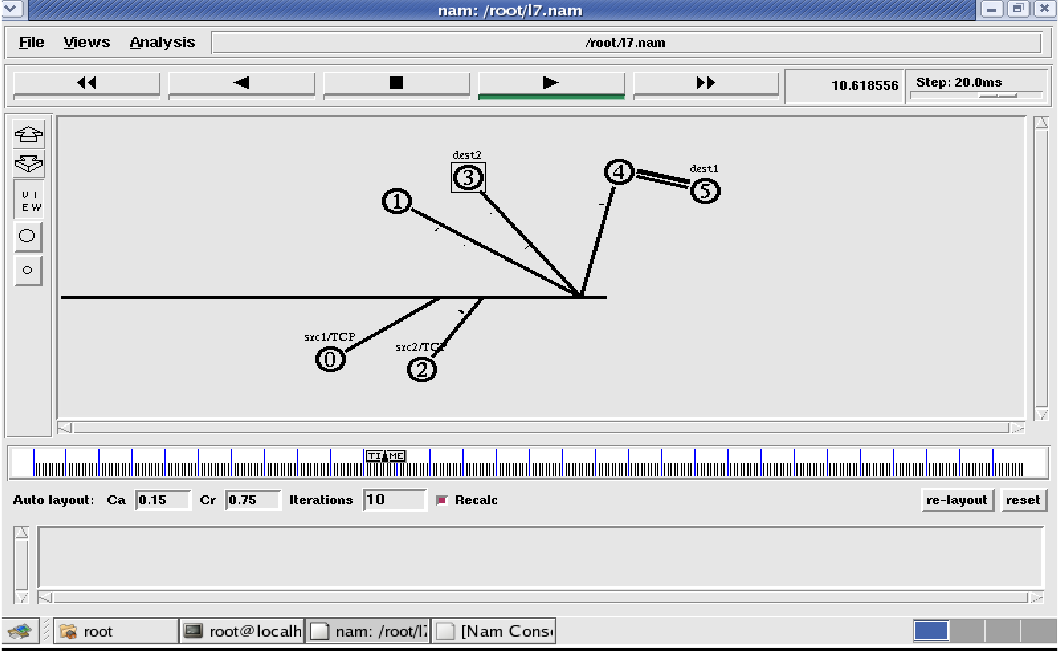
1. After simulation is completed run **awk file** to see the output,

# [root@localhost~]# awk –f lab3.awk file1.tr >a1

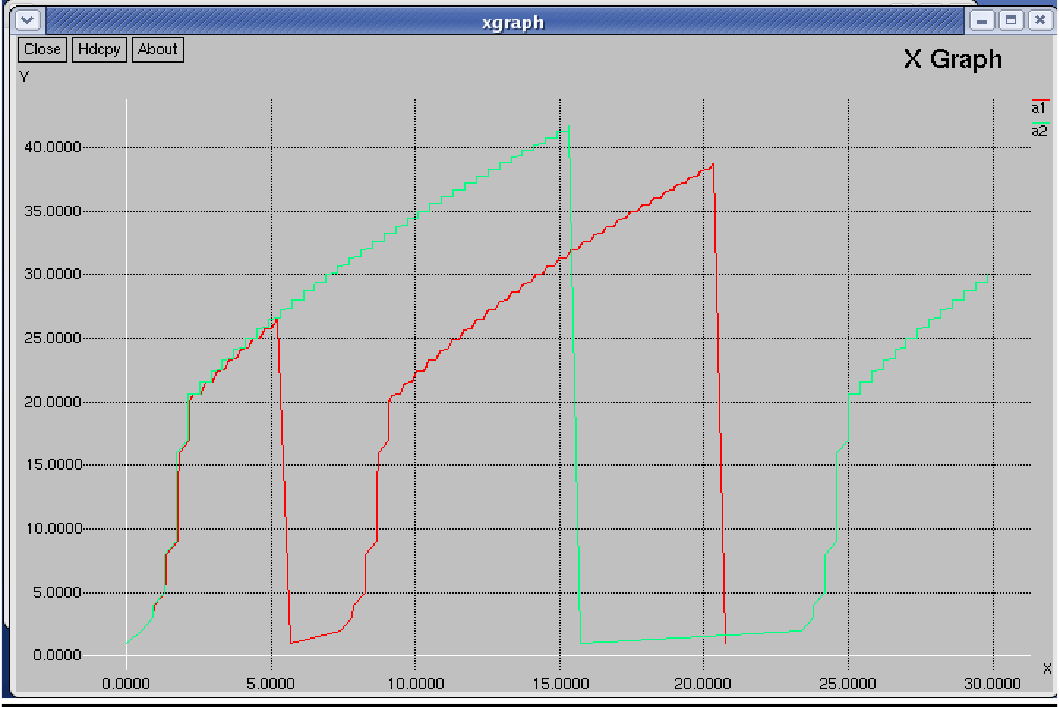
1. **[root@localhost~]# awk –f lab3.awk file2.tr** >**a2**
2. **[root@localhost~]# xgraph a1 a2**
3. Here we are using the congestion window trace files i.e. **file1.tr** and **file2.tr** and we are redirecting the contents of those files to new files say **a1** and **a2** using **output redirection operator (>)**.
4. To see the trace file contents open the file as,

# [root@localhost~]# gedit lab3.tr

# Topology



**Output**



# Steps for execution Program 4

1. Open gedit editor and type program. Program name should have the extension “ **.tcl**”

# [root@localhost ~]# gedit lab4.tcl

1. Save the program and exit.
2. Open vi editor and type **awk** program. Program name should have the extension “**.awk**”

# [root@localhost ~]# gedit lab4.awk

1. Save the program and exit.
2. Run the simulation program

# [root@localhost~]# ns lab4.tcl

* + 1. Here **“ns”** indicates network simulator. We get the topology shown in the snapshot.
    2. Now press the play button in the simulation window and the simulation will begin.

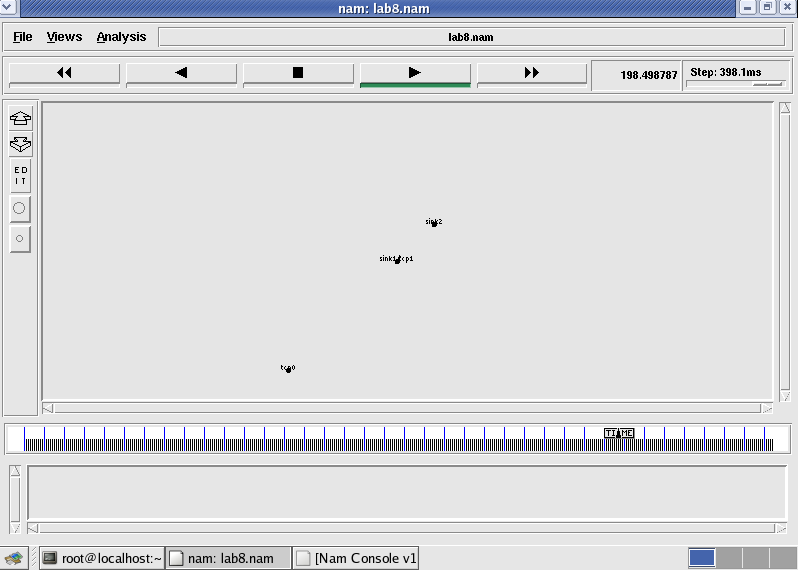
1. After simulation is completed run **awk file** to see the output,

# [root@localhost~]# awk –f lab4.awk lab4.tr

1. To see the trace file contents open the file as,

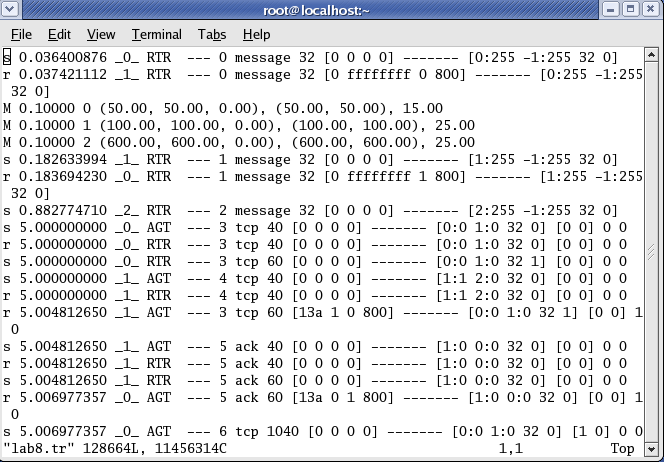
# [root@localhost~]# gedit lab4.tr

**Topology**

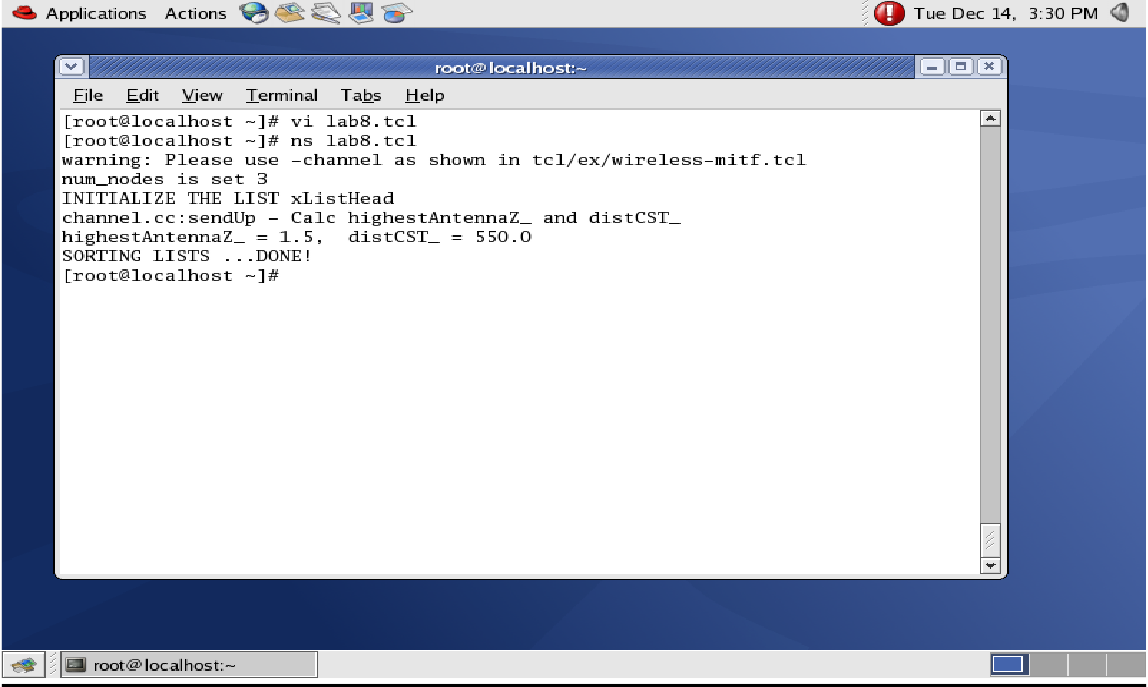


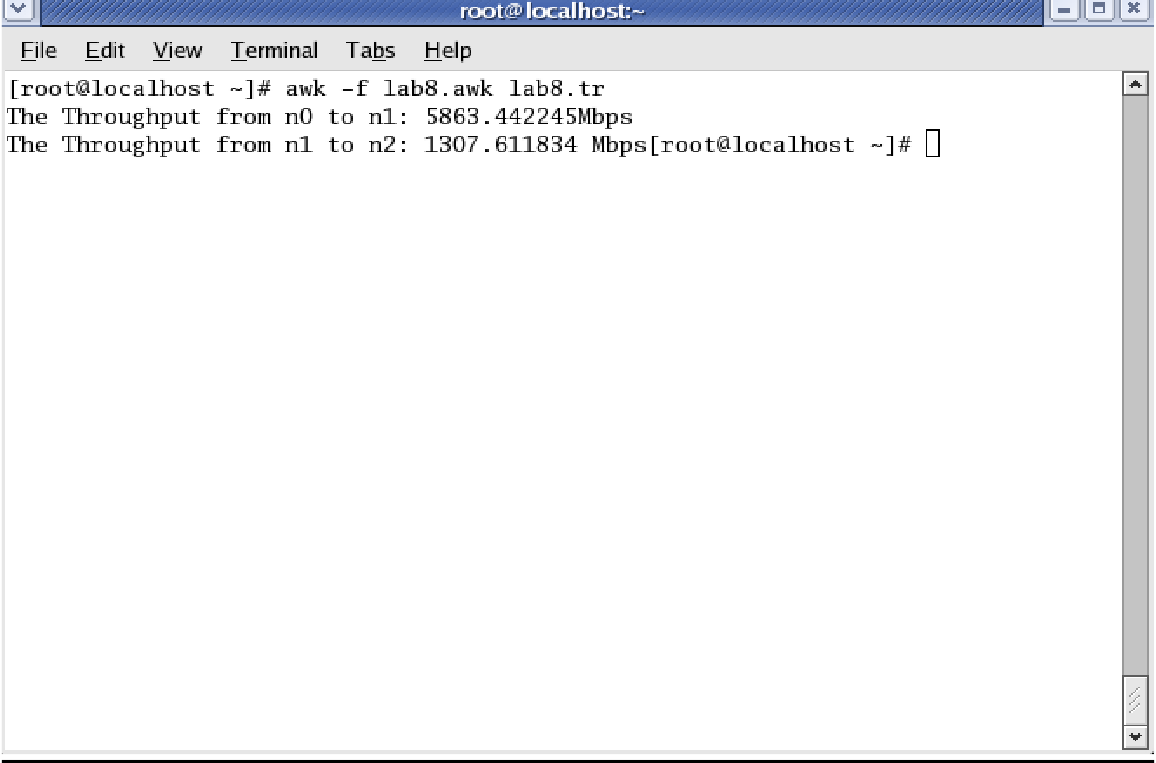
Node 1 and 2 are communicating

# Trace file

Here **“M”** indicates mobile nodes, **“AGT”** indicates Agent Trace, **“RTR”** indicates Router Trace

# Output





# Steps for execution Program 5 - GSM

1. Go to location **Downloads/ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts,** right click in the given location, click on “Open in terminal”, a new terminal would be opened with the location specified above.
2. Open gedit editor and type program. Program name should have the extension “ **.tcl**”

# [root@localhost ~]# gedit lab5.tcl

1. Save the program and exit.
2. Run the simulation program

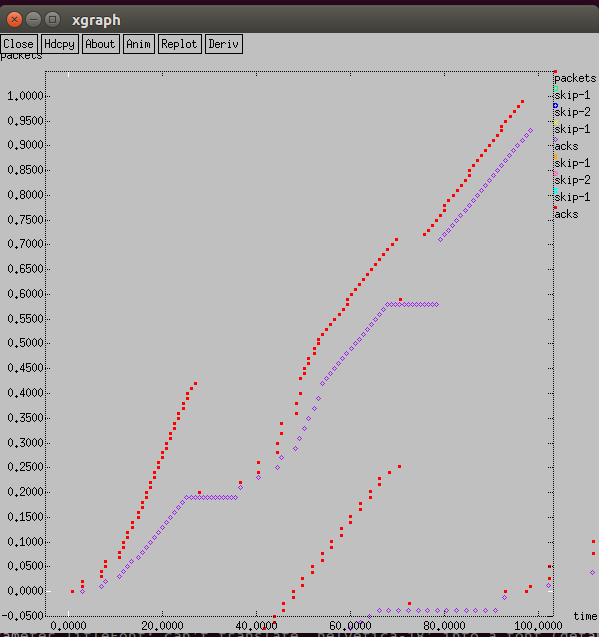
# [root@localhost~]# ns lab5.tcl

Note: Possible errors while executing: If you find an error with something like **“Can’t locate getopts.pl in @INC”**

**Solution**: Install the below command in the terminal

**$sudo apt-get install libperl4-corelibs-perl**

**Output:**

****

# Steps for execution Program 6 - CDMA

1. Go to location **Downloads/ns-allinone-2.35/ns-2.35/tcl/ex/wireless-scripts,** right click in the given location, click on “Open in terminal”, a new terminal would be opened with the location specified above.
2. Open gedit editor and type program. Program name should have the extension “ **.tcl**”

# [root@localhost ~]# gedit lab6.tcl

1. Save the program and exit.
2. Run the simulation program

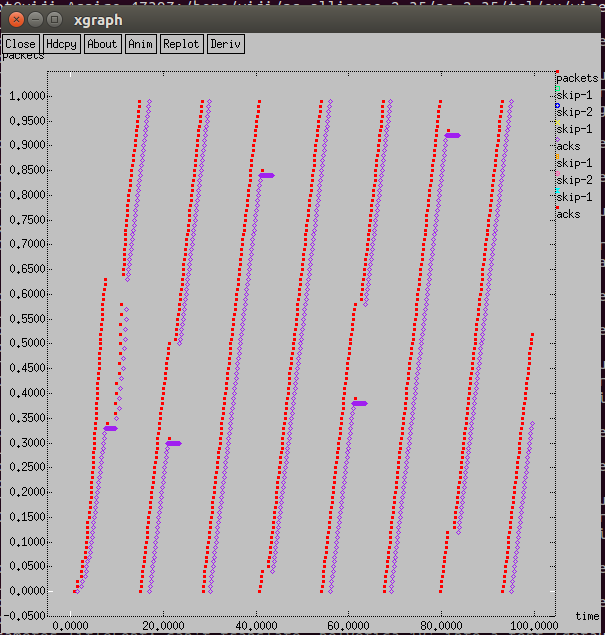
# [root@localhost~]# ns lab6.tcl

Note: Possible errors while executing: If you find an error with something like **“Can’t locate getopts.pl in @INC”**

**Solution**: Install the below command in the terminal

**$sudo apt-get install libperl4-corelibs-perl**

**Output:**

****

**7**. Write a Program for ERROR detecting code using CRC-CCITT (16bit).

**Error**  
A condition when the receiver’s information does not match with the sender’s information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from sender to receiver. That means a 0 bit may change to 1 or a 1 bit may change to 0.

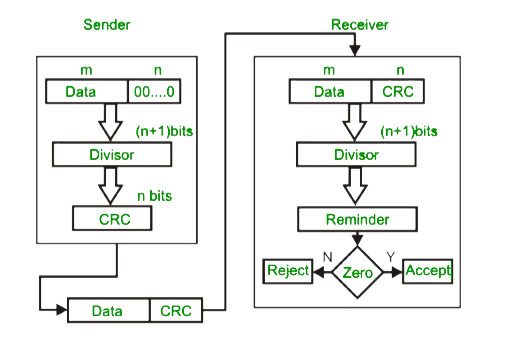
**Error Detecting Codes (Implemented either at Data link layer or Transport Layer of OSI Model)**

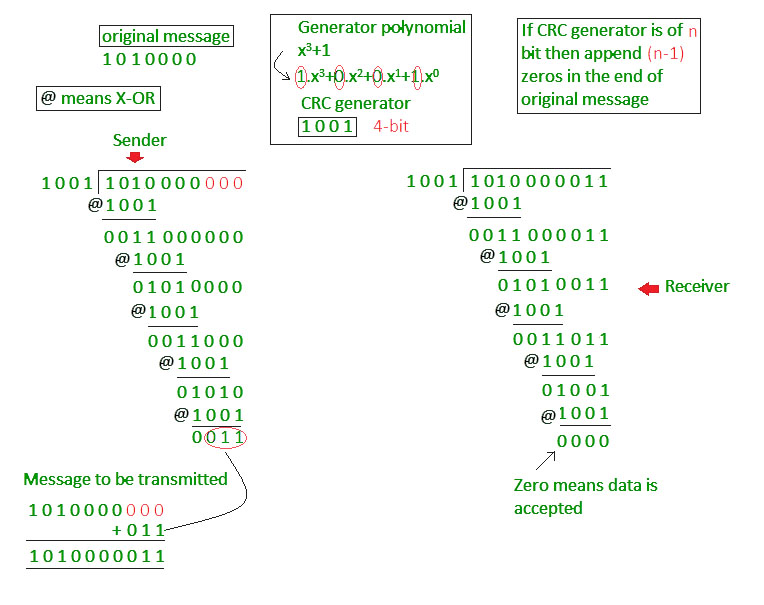
Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if any error has occurred during transmission of the message.

Basic approach used for error detection is the use of redundancy bits, where additional bits are added to facilitate detection of errors.

Some popular techniques for error detection are:  
1. Simple Parity check  
2. Two-dimensional Parity check  
3. Checksum  
4. Cyclic redundancy check  
  
**Cyclic redundancy check (CRC)**

* Unlike checksum scheme, which is based on addition, CRC is based on binary division.
* In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
* At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.
* A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/detect14.jpg)

**Example :**  
[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/detect15.jpg)

import java.io.\*;  
class crc\_gen  
{  
    public static void Bellman(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());  
  
        System.out.println("Enter number of bits in divisor : ");  
        divisor\_bits=Integer.parseInt(br.readLine());  
        divisor=new int[divisor\_bits];  
          
        System.out.println("Enter Divisor bits : ");  
        for(int i=0; i<divisor\_bits; i++)  
            divisor[i]=Integer.parseInt(br.readLine());  
  
        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(div, divisor, rem);  
          
        for(int i=0;i<div.length;i++)           //append dividend and remainder  
        {  
            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(crc, 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 div[],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 :  
Enter number of data bits :  
7  
Enter data bits :  
1  
0  
1  
1  
0  
0  
1  
Enter number of bits in divisor :  
3  
Enter Divisor bits :  
1  
0  
1  
Data bits are : 1011001  
divisor bits are : 101  
Dividend (after appending 0's) are : 101100100  
  
CRC code :  
101100111  
Enter CRC code of 9 bits :  
1  
0  
1  
1  
0  
0  
1  
0  
1  
crc bits are : 101100101  
Error  
THANK YOU.... :)  
Press any key to continue...

8 Write a program to find the shortest path between vertices using Bellman-Ford algorithm.

**Algorithm**  
Following are the detailed steps.

Input: Graph and a source vertex src

Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

**1)** This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.

**2)** This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.  
…..**a)** Do following for each edge u-v  
………………If dist[v] > dist[u] + weight of edge uv, then update dist[v]  
………………….dist[v] = dist[u] + weight of edge uv

**3)** This step reports if there is a negative weight cycle in graph.

Do following for each edge u-v  
……If dist[v] > dist[u] + weight of edge uv, then “Graph contains negative weight cycle”  
The idea of step 3 is, step 2 guarantees shortest distances if graph doesn’t contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

**Example**  
Let us understand the algorithm with following example graph. The images are taken from [this](http://www.cs.arizona.edu/classes/cs445/spring07/ShortestPath2.prn.pdf) source.

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so all edges must be processed 4 times.

Description: Example Graph

Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances. The second row shows distances when edges (B,E), (D,B), (B,D) and (A,B) are processed. The third row shows distances when (A,C) is processed. The fourth row shows when (D,C), (B,C) and (E,D) are processed.

Description: https://media.geeksforgeeks.org/wp-content/uploads/bellmanford2.png

The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time (The last row shows final values).

Description: https://media.geeksforgeeks.org/wp-content/uploads/bellmanford3.png

The second iteration guarantees to give all shortest paths which are at most 2 edges long. The algorithm processes all edges 2 more times. The distances are minimized after the second iteration, so third and fourth iterations don’t update the distances.

import java.util.Scanner;

public class Bellman {

private int d[];

private int n;

public Bellman(int n)

{

this.n = n;

d = new int[n + 1];

}

public void BellmanFordEvaluation(int source, int w[][])

{

for (int i = 1; i <= n; i++)

{

d[i] = 999;

}

d[source] = 0;

for (int i = 1; i <= n - 1; i++)

{

for (int u = 1; u <=n; u++)

{

for (int v = 1; v <=n; v++)

{

if(w[u][v] != 999)

{

if (d[v] >d[u] +w[u][v])

d[v] =d[u] +w[u][v];

}

}

}

}

for (int u = 1; u <= n;u++)

{

for (int v = 1; v <=n; v++)

{

if (w[u][v]!= 999)

{

if (d[v] >d[u] +w[u][v])

{

System.out.println("The Graph contains negative egde cycle");

System.exit(0);

}

}

}

}

for (int vertex = 1; vertex <= n;vertex++)

{

System.out.println("distance of source " + source +" to "+ vertex + " is " + d[vertex]);

}

}

public static void Bellman(String[] args)

{

int n = 0;

int source;

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of vertices");

n = scanner.nextInt();

int w[][] = new int[n +1][n + 1];

System.out.println("Enter the adjacency matrix");

for (int u = 1; u <= n;u++)

{

for (int v = 1; v <=n; v++)

{

w[u][v] =scanner.nextInt();

}

}

System.out.println("Enter the source vertex");

source = scanner.nextInt();

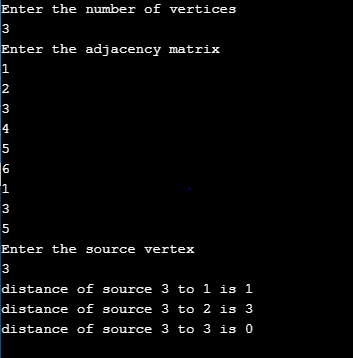
Bellman b = new Bellman(n);

b.BellmanFordEvaluation(source,w);

scanner.close();

}

}



9. Using TCP/IP Sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present. Implement the above program using as message queues or FIFOs as IPC channels.

Socket is an interface which enables the client and the server to communicate and pass on information from one another. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net package provides support for the two common network protocols:

− TCP − TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.

* UDP − UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

**Socket Programming**: Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets −

* The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
* The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
* After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
* The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
* On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket has both an OutputStream and an InputStream. The client's OutputStream is connected to the server's InputStream, and the client's InputStream is connected to the server's OutputStream.

TCP is a two-way communication protocol, hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

**ServerSocket**

This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

**public ServerSocket(int port, int backlog) throws IOException**

Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue.

If the ServerSocket constructor does not throw an exception, it means that your application has successfully bound to the specified port and is ready for client requests.

**public Socket accept() throws IOException**

Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely.

**public InputStream getInputStream() throws IOException**

Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

**public OutputStream getOutputStream() throws IOException**

Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

The **DataOutputStream** stream lets you write the primitives to an output source. The DataInputStream is used in the context of DataOutputStream and can be used to read primitives.

The **java.io.DataInputStream.readUTF()** method reads in a string that has been encoded using a modified UTF-8 format. The string of character is decoded from the UTF and returned as String.

**public static Path get(String first,String... more)**

Converts a path string, or a sequence of strings that when joined form a path string, to a Path.

**public final class Files**

This class consists exclusively of static methods that operate on files, directories, or other types of files.

**public static byte[ ] readAllBytes(Path path) throws IOException**

Reads all the bytes from a file. The method ensures that the file is closed when all bytes have been read or an I/O error, or other runtime exception, is thrown. Note that this method is intended for simple cases where it is convenient to read all bytes into a byte array. It is not intended for reading in large files.

**Parameters:**

**path** - the path to the file

Returns: a byte array containing the bytes read from the file

**String (byte[ ] bytes)**

Constructs a new String by decoding the specified array of bytes using the platform's default charset.

**Source Code**:

//TCP SERVER

import java.io.\*;

import java.net.\*;

import java.nio.file.\*;

public class TCPServer

{

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

{

ServerSocket server;

DataOutputStream out = null;

DataInputStream in;

try

{

server = new ServerSocket(5000, 1); //port number and num of connections

System.out.println("Server Waiting for client");

Socket socket = server.accept();

System.out.println("Client connected ");

in = new DataInputStream(socket.getInputStream());

out = new DataOutputStream(socket.getOutputStream());

String fileName = in.readUTF();

System.out.println("File Requested is : " + fileName);

byte[] filedata = Files.readAllBytes(Paths.get(fileName));

String fileContent = new String(filedata);

out.writeUTF(fileContent.toString());

System.out.println("FILE SENT SUCCESSFULLY");

}

catch (Exception e)

{

System.out.println(e.getMessage());

out.writeUTF("FILE DOESN'T EXISTS");

}

}

}

**//TCP CLIENT**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class TCPClient

{

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

{

DataOutputStream out;

DataInputStream in;

Scanner scanner = new Scanner(System.in);

Socket socket = new Socket("127.0.0.1", 5000); //server IP and Port num

System.out.println("Client Connected to Server");

System.out.print("\nEnter the filename to request\n");

String filename = scanner.nextLine();

in = new DataInputStream(socket.getInputStream());

out = new DataOutputStream(socket.getOutputStream());

out.writeUTF(filename);

String fileContent = in.readUTF();

if (fileContent.length() > 0)

System.out.println(fileContent);

else

System.out.println("FILE IS EMPTY");

}

}

OUTPUT:

Text

Description automatically generated

Text

Description automatically generated

**Note**:

Create two different files Client.java and Server.java. Follow the steps given:

1. Open a terminal run the server program and provide the filename to send

2. Open one more terminal run the client program and provide the IP address of the server. We can give localhost address “127.0.0.1” as it is running on same machine or give the IP address of the machine.

3. Send any start bit to start sending file.

4. Refer https://www.tutorialspoint.com/java/java\_networking.htm for all the parameters, methods description in socket communication.

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

A datagram socket is the one for sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

One of the examples where UDP is preferred over TCP is the live coverage of TV channels. In this aspect, we want to transmit as many frames to live audience as possible not worrying about the loss of one or two frames. TCP being a reliable protocol add its own overhead while transmission.

Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.



Java supports datagram communication through the following classes:

* DatagramPacket
* DatagramSocket

**public class DatagramSocket()**

This class represents a socket for sending and receiving datagram packets. A datagram socket is the sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

The class DatagramPacket contains several constructors that can be used for creating packet object. One of them is: DatagramPacket(byte[] buf, int length, InetAddress address, int port);

This constructor is used for creating a datagram packet for sending packets of length length to the specified port number on the specified host. The message to be transmitted is indicated in the first argument. The key methods of DatagramPacket class are:

* byte[] getData() - Returns the data buffer.
* int getLength() - Returns the length of the data to be sent or the length of the data received.
* void setData(byte[] buf) - Sets the data buffer for this packet.
* void setLength(int length) - Sets the length for this packet.

The class DatagramSocket supports various methods that can be used for transmitting or receiving data a datagram over the network. The two key methods are:

* void send(DatagramPacket p) Sends a datagram packet from this socket.
* void receive(DatagramPacket p) Receives a datagram packet from this socket.

**DatagramSocket()**

Constructs a datagram socket and binds it to any available port on the local host machine.

**DatagramSocket(int port)**

Constructs a datagram socket and binds it to the specified port on the local host machine.

**public void receive(DatagramPacket p) throws IOException**

Receives a datagram packet from this socket. When this method returns, the DatagramPacket's buffer is filled with the data received. The datagram packet also contains the sender's IP address, and the port number on the sender's machine. This method blocks until a datagram is received. The length field of the datagram packet object contains the length of the received message. If the message is longer than the packet's length, the message is truncated.

**Parameters**:

p - The DatagramPacket into which to place the incoming data.

**public final class DatagramPacket**

This class represents a datagram packet.

Datagram packets are used to implement a connectionless packet delivery service. Each message is routed from one machine to another based solely on information contained within that packet. Multiple packets sent from one machine to another might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

**DatagramPacket(byte[ ] buf, int length)**

Constructs a DatagramPacket for receiving packets of length length.

**DatagramPacket(byte[] buf, int length, InetAddress address, int port)**

Constructs a datagram packet for sending packets of length length to the specified port number on the specified host.

**public byte[ ] getData( )**

Returns the data buffer. The data received or the data to be sent starts from the offset in the buffer, and runs for length long.

Returns: the buffer used to receive or send data

**String (byte[ ] bytes, int offset, int length)**

Constructs a new String by decoding the specified subarray of bytes using the platform's default charset.

**public class InetAddress**

This class represents an Internet Protocol (IP) address.

An IP address is either a 32-bit or 128-bit unsigned number used by IP, a lower-level protocol on which protocols like UDP and TCP are built.

**public static InetAddress getByName(String host) throws UnknownHostException**

Determines the IP address of a host, given the host's name.

A simple UDP server program that waits for client’s requests and then accepts the message (datagram) and sends back the same message is given below. Of course, an extended server program can manipulate client’s messages/request and send a new message as a response.

**INPUT:**

a) Enter the message

**EXPECTED OUTPUT**

a) Print the received message

**Source Code:**

**//RECEIVER**

import java.net.\*;

public class DReceiver

{

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

{

byte[] buf = new byte[1024];

System.out.println("Receiver");

DatagramSocket ds = new DatagramSocket(3000);

while(true)

{

DatagramPacket dp = new DatagramPacket(buf, 1024);

ds.receive(dp);

String msg = new String(dp.getData(), 0, dp.getLength());

System.out.println(msg);

}

}

}

**//SENDER**

import java.net.\*;

import java.util.Scanner;

public class DSender

{

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

{

System.out.println("Sender");

DatagramSocket ds = new DatagramSocket();

Scanner scanner = new Scanner(System.in);

System.out.println("\nEnter the Message : ");

while(true)

{

String msg = scanner.nextLine();

InetAddress ip = InetAddress.getByName("127.0.0.1");

DatagramPacket dp = new DatagramPacket(msg.getBytes(), msg.length(), ip, 3000);

ds.send(dp);

}

}

}

**OUTPUT:**

Note: Create two different files DSender.java and DReciever.java. Follow the following steps:

1. Open a terminal run the server program.

2. Open one more terminal run the client program, the sent message will be received.

****

**11. Write a program for simple RSA algorithm to encrypt and decrypt the data.**

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. 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. The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted e and d, respectively) and taking the remainder of the division with N. A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo. The RSA algorithm comprises of three steps, which are depicted below:

**Key Generation Algorithm**

1. Generate two large random primes, p and q, of approximately equal size such that their product n = p\*q

2. Compute n = p\*q and Euler’s totient function (φ) phi(n) = (p-1)(q-1).

3. Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.

4. Compute the secret exponent d, 1 < d < phi, such that e\*d ≡ 1 (mod phi).

5. The public key is (e, n) and the private key is (d, n). The values of p, q, and phi should also be kept secret.

**Encryption**

Sender A does the following:-

1. Using the public key (e,n)

2. Represents the plaintext message as a positive integer M

3. Computes the cipher text C = Me mod n.

4. Sends the cipher text C to B (Receiver).

**Decryption**

Recipient B does the following:-

1. Uses his private key (d, n) to compute M = Cd mod n.

2. Extracts the plaintext from the integer representative m.

RSA is the most widely used public key algorithm. The algorithm is relatively easy to understand and implement. It has been patent free since 2000. RSA is used in security protocols such as IPSEC/IKE -IP data security, TLS/SSL -transport data security (web), PGP -email security, SSH -terminal connection security, SILC -conferencing service security. RSA gets its security from the integer factorization problem. The difficulty of factoring large numbers is the basis of security of RSA.

**INPUT**

a) Enter the Message

**EXPECTED OUTPUT**

1. Print the private key and public key
2. Print the cipher Text
3. Print the plain Text

Example of RSA with small numbers: p = 47, q = 71,

Compute n = pq = 3337

Compute phi = 46 \* 70 = 3220

Let e be 79, compute d = 79-1 mod 3220 = 1019

Public key is n and e, private key d, discard p and q.

Encrypt message m = 688,

c = 68879 mod 3337 = 1570.

Decrypt message c = 1570,

m =15701019 mod 3337 = 688.

**Source Code:**

import java.io.DataInputStream;

import java.io.IOException;

import java.math.BigInteger;

import java.util.Random;

public class RSA

{

private BigInteger p;

private BigInteger q;

private BigInteger N;

private BigInteger phi;

private BigInteger e;

private BigInteger d;

private int bitlength = 1024;

private Random r;

public RSA()

{

r = new Random();

p = BigInteger.probablePrime(bitlength, r);

q = BigInteger.probablePrime(bitlength, r);

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);

}

d = e.modInverse(phi);

System.out.println("public key : "+e+","+N);

System.out.println("private key : "+d+","+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;

}

// Encrypt message

public byte[] encrypt(byte[] message)

{

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

}

// Decrypt message

public byte[] decrypt(byte[] message)

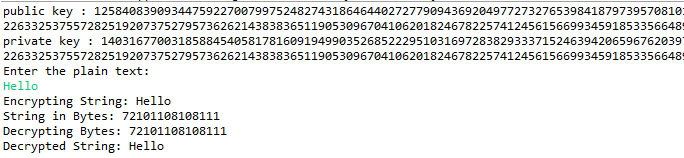
{

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

}

}

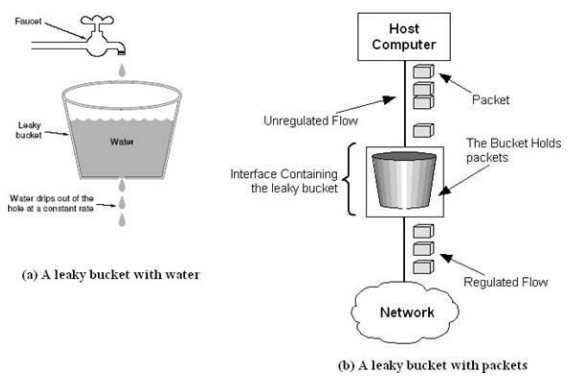
**OUTPUT:**

****

**12. Write a program for congestion control using leaky bucket algorithm.**

**AIM:** To control congestion in computer network

**DESCRIPTION**: The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spillover. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).



While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn’t take into account the idle process of the sender which means that if the host doesn’t transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

**INPUT**

a) Enter the Bucket Size

b) Enter the flow rate

c) Enter the no of packets

d) Enter the size of packets

**EXPECTED OUTPUT**

a) Print the Bucket Size

b) Print the flow rate

c) Print the no of packet and size

**PROGRAM: LEAKY BUCKET**

import java.util.Scanner;

public class Leakybucket {

public static void main(String[] args) {

Scanner sc= new Scanner(System.in);

System.out.println("Enter the bucket size=");

int n=sc.nextInt();

int a[]=new int[n];

System.out.println("Enter the number of packets=");

int num=sc.nextInt();

System.out.println("Enter the Data rate=");

for(int i=0;i<num;i++)

{

a[i]=sc.nextInt();

}

System.out.println("Enter output rate=");

int out=sc.nextInt();

for(int i=0;i<num;i++)

if(a[i]>n)

{

System.out.println("Bucket overflow\n"+a[i]);

}

else

{

if (a[i]==out)

System.out.println("Packet transmitted"+a[i]);

else if(a[i]>out)

{

while(a[i]!=0 && a[i]>out)

{

System.out.println("Packet transmitted "+out);

a[i]=a[i]-out;

}

System.out.println("Packet transmitted "+a[i]);

}

}

}

}

