EX.NO : 1 SIMPLE CLIENT SERVER PROGRAM-TCP
DATE :

AIM:

To write a java program for TCP CHAT.

ALGORITHM:

Client

1. Start
2. Create the TCP datagram socket
3. Get the request message to be sent from the user
4. Send the request message to the server
5. If the request message is “END” go to step 10
6. Wait for the reply message from the server
7. Receive the reply message sent by the server
8. Display the reply message received from the server
9. Repeat the steps from 3 to 8
10. Stop

Server

1. Start
2. Create TCP datagram socket, make it a listening socket
3. Receive the request message sent by the client
4. If the received message is “END” go to step 10
5. Retrieve the client’s IP address from the request message received
6. Display the received message
7. Get the reply message from the user
8. Send the reply message to the client
9. Repeat the steps from 3 to 8.
10. Stop

PROGRAM:

//TCPServer.java

```
import java.io.*;
import java.net.*;
classTCPServer
{
```

```
public static void main (String argv[]) throws Exception
{
String fromclient;
String toclient;

ServerSocket Server = new ServerSocket (5000);
System.out.println("TCPServerWaitingforclientonport5000");
while(true)
            {
Socket connected = Server.accept();
System.out.println("THECLIENT"+""+
connected.getInetAddress() +":"+connected.getPort()+" IS CONNECTED");

BufferedReaderin FromUser=new BufferedReader(new
InputStreamReader(System.in));

BufferedReaderin FromClient=new
BufferedReader(newInputStreamReader(connected.getInputStream()));

PrintWriter outToClient=new
PrintWriter(connected.getOutputStream(),true);

while (true) {

        System.out.println("SEND (Type Q or q to Quit):");
        toclient=inFromUser.readLine();

        if ( toclient.equals ("q") || toclient.equals("Q") )
        {
                outToClient.println(toclient);
                connected.close();
                break;
        }
        else
        {
                outToClient.println(toclient);
        }
        fromclient = inFromClient.readLine();

        if(fromclient.equals("q")||fromclient.equals("Q") ) {

                connected.close();
                break;
        }
        else
        {
        System.out.println("RECIEVED:" + fromclient );
        }
        }
        }
}
}
```

//TCPClient.java

```
import java.io.*;
import java.net.*;
class TCPClient
{
public static void main (String argv[]) throws Exception
{
String FromServer;
String ToServer;
Socket clientSocket = new Socket ("localhost", 5000);

BufferedReaderin FromUser=new BufferedReader(newInputStreamReader(System.in));

PrintWriterout ToServer=new PrintWriter(clientSocket.getOutputStream(),true);

BufferedReaderin FromServer=new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

while (true)
{

FromServer=inFromServer.readLine();

if(FromServer.equals("q")||FromServer.equals("Q"))
{
        clientSocket.close();
        break;
}
else
{
System.out.println("RECIEVED:" + FromServer);
System.out.println("SEND (Type Q or q to Quit):");
ToServer=inFromUser.readLine();
if(ToServer.equals("Q")||ToServer.equals("q"))
{
        outToServer.println(ToServer);
        clientSocket.close();
        break;
}

else
{
        outToServer.println(ToServer);
}
}
}
}
}
```

OUTPUT:

//***TCPSERVER***

```
S:\tcpchat>javaTCPServer
TCPServerWaitingforclientonport5000
THE CLIENT/127.0.0.1:58755 IS CONNECTED
SEND (Type Q or q to quit):
hey
RECEIVED: hello
```

//***TCPCLIENT***

```
S:\tcpchat>java
TCPClient RECEIVED:
hey
SEND (Type Q or q to quit):
hello
```

RESULT:

Thus the java program for TCP CHAT is successfully executed.

EX.NO: 2 SOCKET PROGRAM FOR ECHO

DATE :

Aim:

To implement TCP ECHO using java programming

ALGORITHM:

Client:

1. Start
2. Create the TCP socket
3. Establish connection with the server
4. Get the message to be echoed from the user
5. Send the message to the server
6. Receive the message echoed by the server
7. Display the message received from the server
8. Terminate the connection
9. Stop

Server:

1. Start
2. Create TCP socket, make it a listening socket
3. Accept the connection request sent by the client for connection establishment
4. Receive the message sent by the client
5. Display the received message
6. Send the received message to the client from which it receives
7. Close the connection when client initiates termination and server becomes a listening server, waiting for clients.
8. Stop.

PROGRAM:

//EchoServer.java

```
import java.io.*;
import java.net.*;
public class EchoServer{
public EchoServer()
        {
                try
                {
                        server = new ServerSocket(9999);
                }
                catch (Exception err)
                {
                        System.out.println(err);
                }
        }
        public void serve()
        {
                try
                {
                        while (true)
                        {
                        Socket client = server.accept();
                        BufferedReader r = new BufferedReader(new
                        InputStreamReader(client.getInputStream()));
                        PrintWriter w=new PrintWriter(client.getOutputStream(),true);
                        w.println("Welcome to the Java Echo Server. Type 'bye' to close.");
                        String line;
                                do
                                {
                                   Line=r.readLine();
                                   if (line !=null){w.println("Got:"+line);
                                    System.out.println("received from client:"+line);
                                }

                                 }
                                 while ( !line.trim().equals("bye") );
                                client.close();
                          }
                }
                catch (Exception err)
                {
                        System.err.println(err);
                }
        }public static void main(String[] args)
```

```
            {
                    EchoServer s = new EchoServer();
                    s.serve();
            }
    private ServerSocket server;
    }


    //EchoClient.java
    import  java.io.*;
    import  java.net.*;
    public class EchoClient
    {
            public static void main(String[] args)
            {
                    try
                    {
                            Socket s = new Socket("localhost", 9999);
                            BufferedReaderr=new BufferedReader(new
                            InputStreamReader(s.getInputStream()));
                            PrintWriter w=new PrintWriter(s.getOutputStream().true());
                            BufferedReaderc on=new BufferedReader(new
                            InputStreamReader(System.in));
                            String line;
                            do
                            {
                                    line=r.readLine();
                                    if(line!=null)
                                            System.out.println(line);
                                    line=con.readLine();
                                    w.println(line);
                            }while(!line.trim().equals("bye"));
                    }
                    catch(Exception err)
                    {
                            System.err.println(err);
                    }
}}
```

OUTPUT:

#EchoServer.java

```
D:\BECSE\Sem4\Lab\NetworksLab\Programs>javac EchoServer.java

D:\BE CSE\Sem 4\Lab\Networks Lab\Programs>java EchoServer
received from client: Hello
```

#EchoClient.java

```
D:\BE CSE\Sem 4\Lab\Networks Lab\Programs>javac EchoClient.java

D:\BE CSE\Sem 4\Lab\Networks Lab\Programs>java EchoClient
Welcome to the Java Echo Server. Type 'bye' to close.
Hello
Got: Hello
```

RESULT:

Thus the java program for TCP ECHO is successfully executed and verified.

EX.NO :3 SOCKET PROGRAM FOR PING COMMANDS
DATE :

AIM :

To write the java program for simulating ping commands.

ALGORITHM:

Server
1: Start the program.
2: Import necessary packages.
3: Initialize the ping server with both sockets as null value.
4: Start the server socket.
5: At the client give the IP address of the server (by using if config command in command prompt).
6: The client program is then started by starting socket.
7: At the receiver end, the client is pinged and traced.
8: Stop the program.

Client
1: Start the program.
2: Import necessary packages.
3: Initialize the ping client with both sockets as null value.
4: Start the socket.
5: Get the IP address of the server.
6: Ping the server.
7: At the receiver end, the server is pinged and traced.
8: Stop the program.

PROGRAM:

//Pingserver.java
```
import java.io.*;
import java.net.*;
class pingserver
{
public static void main (String args[])
{
try
{
String str;
System.out.print(" Enter the IP Address to be Ping: ");
BufferedReader buf1=new BufferedReader(new
InputStreamReader(System.in));
String ip=buf1.readLine();
Runtime H=Runtime.getRuntime();
Process p=H.exec("ping " + ip);
InputStream in=p.getInputStream();
BufferedReader buf2=new BufferedReader(new
InputStreamReader(in));
while((str=buf2.readLine())!=null)
{
System.out.println(" " + str);
} }
catch (Exception e)
{
System.out.println(e.getMessage());
} } }
```

OUTPUT:

```
S:\ping> java pingserver

 Enter the IP Address to be Ping:172.16.17.9

 Pinging 172.16.17.9 with 32 bytes of data:
 Replyfrom172.16.17.9:bytes=32time<1msTTL=64Replyf
 rom172.16.17.9:bytes=32time<1msTTL=64Replyfrom17
 2.16.17.9:bytes=32time<1msTTL=64Replyfrom172.16.1
 7.9:bytes=32time<1msTTL=64

 Ping statistics for 172.16.17.9:
          Packets: sent = 4, Received = 4, Lost = 0 (0% loss),
 Approximate round trip times in milli-seconds:
          Minimum=0ms, Maximum= 0ms, Average =0ms
```

RESULT:

To write the java program for simulating ping commands is executed successfully

EX.NO :4 LINK STATE ROUTING ALGORITHM

DATE :

AIM:

To simulate the Link state routing protocols.

ALGORITHM:

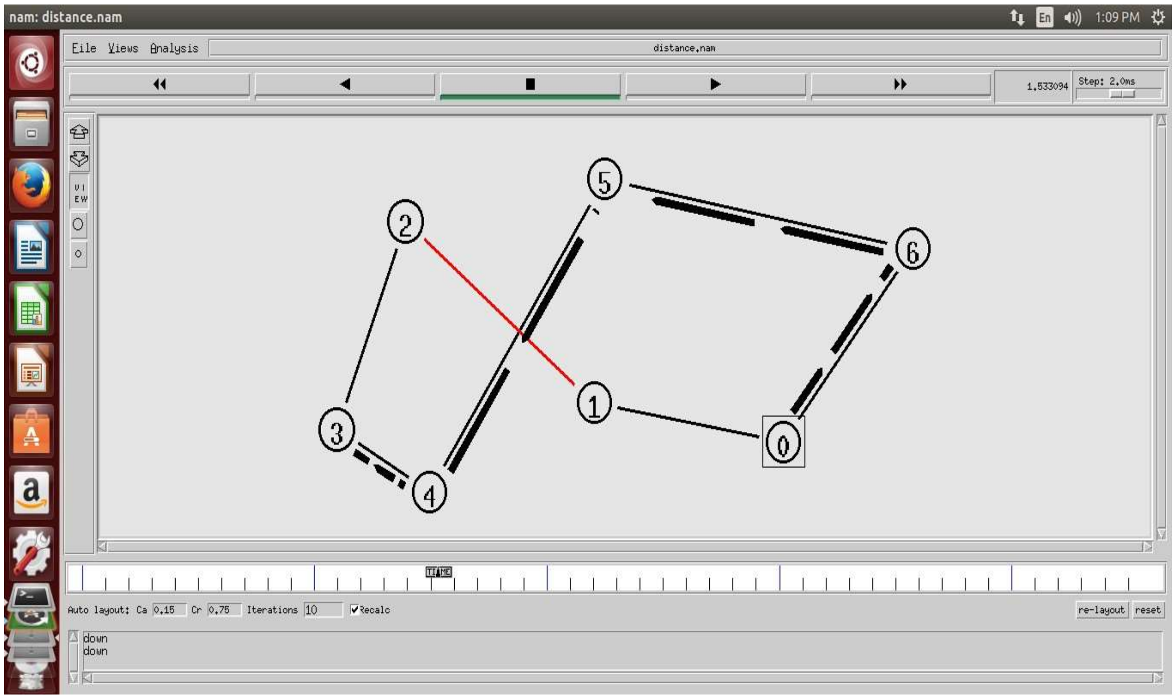
1. Create a Simulator object.
2. Set routing as dynamic.
3. Open the trace and nam trace files.
4. Define the finish procedure.
5. Create nodes and the links between them.
6. Create the agents and attach them to the nodes
7. Create the applications and attach them to the udp agent.
8. Connect UDP and null.
9. At 1 sec the link between node 1 and 2 is broken.
10. At 2 sec the link is up again.
11. Run the simulation.

LINK STATE ROUTING PROGRAM:

```
#Create a simulator object set
ns [new Simulator]
#Tell the simulator to use dynamic routing
$ns rtproto DV
$nsmacTypeMac/802_3
#Open the nam trace file
set nf [open distance.nam w]
$ns namtrace-all $nf
#Open the output files
setf0[opendistance.trw]
$ns trace-all $f0
#Define a 'finish' procedure
proc finish {} {
global ns f0 nf
$ns flush-trace
#Close the trace file close
$f0
close $nf
```

```
  #Call xgraph to display the resultsexec
  namdistance.nam&
  exit 0
}
  #Create seven nodes
  for {set i 0} {$i< 7} {incri} {
  set n($i) [$ns node]
  }
  #Create links between the nodes
  for {set i 0} {$i< 7} {incri} {
  $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb
  10msDropTail}
  #Create a UDP agent and attach it to node n(0)
  set udp0 [newAgent/UDP]
  $ns attach-agent $n(0) $udp0
  #Create a CBR traffic source and attach it to udp0
  set cbr0 [newApplication/Traffic/CBR]
  $cbr0 set packetSize_ 500
  $cbr0 set interval_ 0.005
  $cbr0 attach-agent $udp0
  #Create a Null agent (a traffic sink) and attach it to node n(3)
  set null0 [newAgent/Null]
  $ns attach-agent $n(3) $null0
  #Connectthetrafficsourcewiththetrafficsink
  $ns connect $udp0 $null0
  #Schedule events for the CBR agent and the network dynamics
  $ns at 0.5 "$cbr0 start"
  $ns rtmodel-at 1.0 down $n(1) $n(2)
  $ns rtmodel-at 2.0 up $n(1) $n(2)
  $ns at 4.5 "$cbr0 stop"
  #Call the finish procedure after 5 seconds of simulation time
  $ns at 5.0 "finish"
  #Run the simulation
```

OUTPUT:



```
1.533094   Step: 2.0ms
Auto layout: Ca 0.15  Cr 0.75  Iterations 10  ✓Recalc
down
down
```

RESULT:

To simulate the Link state routing protocols using NS2 is successfully executed.

EX.NO :5 DISTANCE VECTOR ROUTING ALGORITHM
DATE :

AIM:
To simulate the Distance vector routing protocols using NS2.

ALGORITHM:

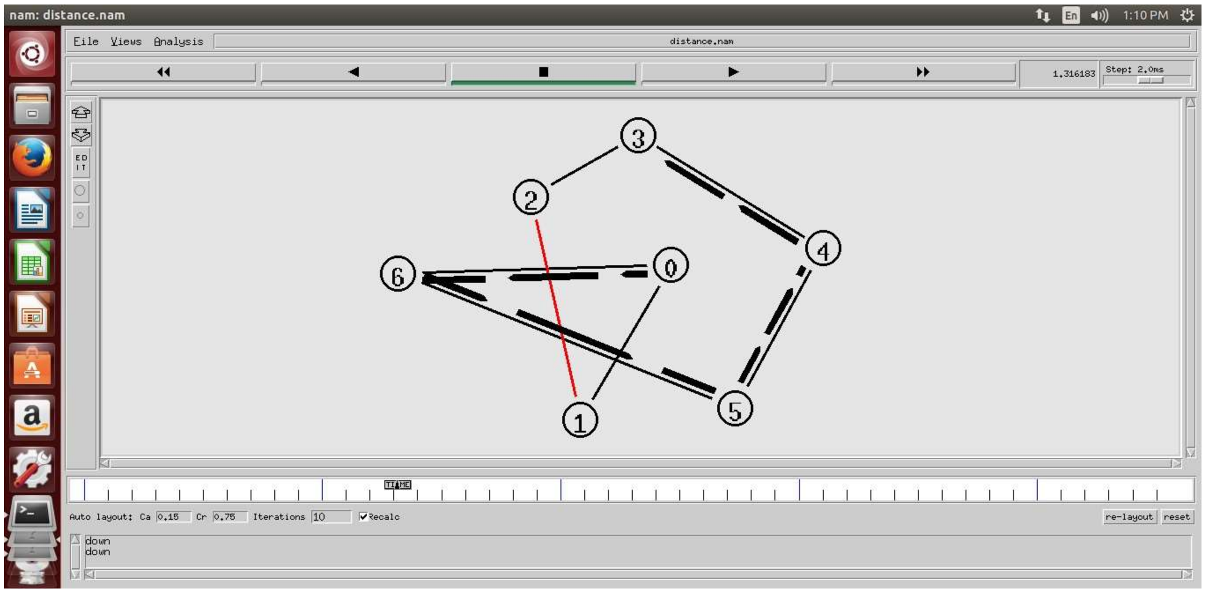
1. Create a simulator object
2. Set routing protocol to Distance Vector routing
3. Trace packets on all links on to NAM trace and text trace file
4. Define finish procedure to close files, flush tracing and run NAM
5. Create eight nodes
6. Specify the link characteristics between nodes
7. Describe their layout topology as a octagon
8. Add UDP agent for node n1
9. Create CBR traffic on top of UDP and set traffic parameters.
10. Add a sink agent to node n4
11. Connect source and the sink
12. Schedule events as follows:
    a. Starttrafficflowat0.5
    b. Down the link n3-n4 at 1.0
    c. Up the link n3-n4 at 2.0
    d. Stoptrafficat3.0
    e. Call finish procedure at 5.0
13. Start the scheduler
14. Observe the traffic route when link is up and down
15. View the simulated events and trace file analyze it
16. Stop

PROGRAM:

```
#Create a simulator object set
ns [new Simulator]
#Tell the simulator to use dynamic routing
$ns rtproto DV
$nsmacTypeMac/802_3
#Open the nam trace file
set nf [open distance.nam w]
$ns namtrace-all $nf
#Open the output files
```

```
set f0 [open distance.tr w]
$ns trace-all $f0
#Define a 'finish' procedure
proc finish {} {
    global ns f0 nf
    $ns flush-trace
    #Close the trace file
    close $f0
    close $nf
   #Call xgraph to display the results
    exec namdistance.nam&
          exit 0
}
#Create seven nodes
for {set i 0} {$i< 7} {incri}
     {set n($i) [$ns node]
}
#Create links between the nodes
for {set i 0} {$i< 7} {incri} {
    $ns duplex-link $n($i) $n([expr ($i+1)%7]) 1Mb
10msDropTail}
#Create a UDP agent and attach it to node n(0)set
udp0 [newAgent/UDP]
$ns attach-agent $n(0) $udp0
#Create a CBR traffic source and attach it to udp0 set
cbr0 [newApplication/Traffic/CBR]
$cbr0 set packet Size_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
#Create a Null agent (a traffic sink) and attach it to node n(3)
set null0 [new Agent/Null]
$ns attach-agent $n(3) $null0
#Connect the traffic source with the traffic sink
$ns connect $udp0 $null0
#Schedule events for the CBR agent and the network
dynamics $ns at 0.5 "$cbr0 start"
$ns rtmodel-at 1.0 down $n(1) $n(2)
$ns rtmodel-at 2.0 up $n(1) $n(2)
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

OUTPUT:



```
1.316183   Step: 2.0ms
Auto layout: Ca 0.15  Cr 0.75  Iterations 10  ✓Recalc
down
down
```

RESULT:

To simulate the Distance vector routing protocols using NS2 is successfully executed.

EX.NO:6 STUDY OF NETWORK SIMULATOR(NS2orNS3) AND
DATE : SIMULATION OF CONGESTION CONTROL ALGORITHM USING NS.

Aim:

To study about the Network Simulator (NS).

Introduction:

Simulation is a very important modern technology. It can be applied to different science, engineering, or other application fields for different purposes. Computer assisted simulation can model hypothetical and real-life objects or activities on a computer so that it can be studied to see how the system function. Different variables can be used to predict the behavior of the system. Computer simulation can be used to assist the modeling and analysis in many natural systems. Typical application areas include physics, chemistry, biology, and human-involved systems in economics, finance or even social science. Other important applications are in the engineering such as civil engineering, structural engineering, mechanical engineering, and computer engineering. Application of simulation technology into networking area such as network traffic simulation, however, is relatively new.

For network simulation, more specifically, it means that the computer assisted simulation technologies are being applied in the simulation of networking algorithms or systems by using software engineering. The application field is narrower than general simulation and it is natural that more specific requirements will be placed on network simulations. For example, the network simulations may put more emphasis on the performance or validity of a distributed protocol or algorithm rather than the visual or real-time visibility features of the simulations. Moreover, since network technologies is keeping developing very fast and so many different organizations participate in the whole process and they have different technologies or products running on different software on the Internet. That is why the network simulations always require open platforms which should be scalable enough to include different efforts and different packages in the simulations of the whole network.

Internet has also a characteristic that it is structured with a uniformed network stack (TCP/IP) that all the different layers technologies can be implemented differently but with a uniformed interface with their neighbored layers. Thus the network simulation tools have to be able to incorporate this feature and allow different future new packages to be included and run transparently without harming existing components or packages. Thus the negative impact of some packages will have no or little impact to the other modules or packages.

Network simulators are used by people from different areas such as academic researchers, industrial developers, and Quality Assurance (QA) to design, simulate, verify, and analyze the performance of different networks protocols. They can also be used to evaluate the effect of the different parameters on the protocols being studied. Generally a network simulator will comprise of a wide range of networking technologies and protocols and help users to build complex networks from basic building blocks like clusters of nodes and links. With their help, one can design different network topologies using various types of nodes such as end-hosts, hubs, network bridges, routers, optical link-layer devices, and mobileunits.

The following sections of the paper are organized as follows. In section 2, we introduce some basic concepts about the network simulations and network simulators. After that, we briefly give an overview of current developments in section 3. From section 4 to 7, we discuss four typical network simulators respectively. A summary is given in section 8.

2) Basic concepts in network simulation

In this section, we will introduce some basic concepts in the area of network simulation. We also try to differentiate and clarify those that easy to cause confusion among readers. We generally introduce the basic idea of the network simulation and simulator and then discuss the difference between simulation and emulation.

Network simulation and simulator

Generally speaking, network simulators try to model the real world networks. The principal idea is that if a system can be modeled, then features of the model can be changed and the corresponding results can be analyzed. As the process of model modification is relatively cheap than the complete real implementation, a wide variety of scenarios can be analyzed at low cost (relative to making changes to a real network). Network simulator always contain the

However, network simulators are not perfect. They can not perfectly model all the details of the networks. However, if well modeled, they will be close enough so as to give there searcher a meaningful insight into the network under test, and how changes will affect its operation.

Simulation and emulation

In the research area of computer and communications networks, simulation is a useful technique since the behavior of a network can be modeled by calculating the interaction between the different network components (they can be end-host or network entities such as routers, physical links or packets) using mathematical formulas. They can also be modeled by actually or virtually capturing and playing back experimental observations from a real production networks.

After we get the observation data from simulation experiments, the behavior of the network and protocols supported can then be observed and analyzed in a series of offline test experiments. All kinds of environmental attributes can also be modified in a controlled manner to assess how the network can behave under different parameters combinations or different configuration conditions. Another characteristic of network simulation that worth noticing is that the simulation program can be used together with different applications and services in order to observe end-to-end or other point-to-point performance in the networks.

Network emulation, however, means that network under planning is simulated in order to assess its performance or to predict the impact of possible changes, or optimizations. The major difference lying between them is that a network emulator means that end-systems such as computers can be attached to the emulator and will act exactly as they are attached to a real network.

The major point is that the network emulator's job is to emulate the network which connects end-hosts, but not the end-hosts themselves. Typical network emulation tools include NS2whichisapopularnetworksimulatorthatcanalsobeusedasalimited-functionality emulator. In contrast, a typical network emulator such as WANsim [WANSIM] is a simple bridged WAN emulator that utilizes some Linux functionality.

Type of network simulators

Different types of network simulators can be categorized and explained based on somecriteria such as if they are commercial or free, or if they are simple ones or complex ones.

a) Commercial and open source simulators

Some of the network simulators are commercial which means that they would not provide the source code of its software or the affiliated packages to the general users for free. All the user s have to pay to get the license to use their software or pay to order specific packages for their own specific usage requirements. One typical example is the OPNET [OPNET]. Commercial simulator has its advantage and disadvantage. The advantage is that it generally has complete and up-to-date documentations and they can be consistently maintained by some specialized staff in that company. However, the open source network simulator is disadvantageous in this aspect, and generally there are not enough specialized people working on the documentation. This problem can be serious when the different versions come with many new things and it will become difficult to trace or understand the previous codes without appropriate documentations.

On the contrary, the open source network simulator has the advantage that everything is very open and everyone or organization can contribute to it and find bugs in it. The interface is also open for future improvement. It can also be very flexible and reflect the most new recent developments of new technologies in a faster way than commercial network simulators. We can see that some advantages of commercial network simulators, however, are the disadvantage for the open source network simulators. Lack of enough systematic and complete documentations and lack of version control supports can lead some serious problems and can limit the applicability and life-time of open source network simulators. Typical open source network simulators include NS2 [NS-2],[NS-2 Wiki] NS3 [NS-3] We will introduce and analyze them in great detail in the following section.

Table1Networksimulators

	Network simulators name
Commercial	OPNET,QualNet
Opensource	NS2,NS3, OMNeT++, SSFNet , J-Sim

Note that limited by the length, not all of them will be introduced in detail in this paper. However, we will focus on some typical ones and introduce others briefly.

b) Simple vs. complex

Currently there are a great variety of network simulators, ranging from the simple ones to the complex ones. Minimally, a network simulator should enable users to represent a network topology, defining the scenarios, specifying the nodes on the network, the links between those nodes and the traffic between the nodes. More complicated systems may allow the user to specify everything about the protocols used to process network traffic. Graphical applications also allow users to easily visualize the workings of their simulated environment. Some of them may be text-based and can provide a less visual or intuitive interface, but may allow more advanced forms of customization. Others may be programming-oriented and can provide a programming framework that allows the users customize to create an application that simulates the networking environment for testing.

c) Overview of current developments

Currently there are many network simulators that have different features in different aspects. A short list of the current network simulators include OPNET, NS-2, NS-3, OMNeT++ [OMNeT], REAL[REAL], SSFNet [SSFNet], J-Sim [J-Sim], and QualNet [QualNet]. However, in this paper, we do not intend to cover all the available network simulators. We only select some typical ones (the first 4 simulators) and do some analysis and compare some from the others slightly to get a better view of the main features of a certain network simulator. What we select are the 4 typical network simulators that represent the recent development status in this area.

The network simulators we select for discussion in this paper include OPNET, NS2, NS3 and OMNet++. Of them, the OPNET is commercial software and is a little different from others and we will introduce in the first place. NS2 are the most popular one in academia because of its open-source and plenty of components library. A lot of non-benefit organizations contribute a lot in the components library and it has been proved that the development mode of NS2 is very successful.

However, because of some natural design limitation of NS2, the NS3 is currently under development and test. Compared with NS2,NS3 put more emphasis on the documentation works and some specialized people are volunteered to manage different components. Moreover, NS3 is not just an updated version of NS2. NS3 redesigns a lot of mechanisms based on the successful and unsuccessfulexperiencesofNS2.OMNet++is an other important network simulator which has very powerful graphical interface and modular core design. OMNet++ is also open sourced and widely acknowledged in academia.

4) OPNET

OPNET [OPNET] is the registered commercial trademark and the name of product presented by OPNET Technologies incorporation. It is one of the most famous and popular commercial network simulators by the end of 2008. Because of it has been used for a longtime in the industry, it become mature and has occupied a big market share.

4.1) Overview

OPNET's software environment is called â€œ Model erâ€, which is specialized for network research and development. It can be flexibly used to study communication networks, devices, protocols, and applications. Because of the fact of being a commercial software provider, OPNET offers relatively much powerful visual or graphical support for the users. The graphical editor interface can be used to build network topology and entities from the application layer to the physical layer. Object-oriented programming technique is used to create the mapping from the graphical design to the implementation of the real systems. An example of the graphical GUI of OPNET can be seen in figure 1. We can see all the topology configuration and simulation results can be presented very intuitively and visually. The parameters can also be adjusted and the experiments can be repeated easily through easy operation through the GUI.

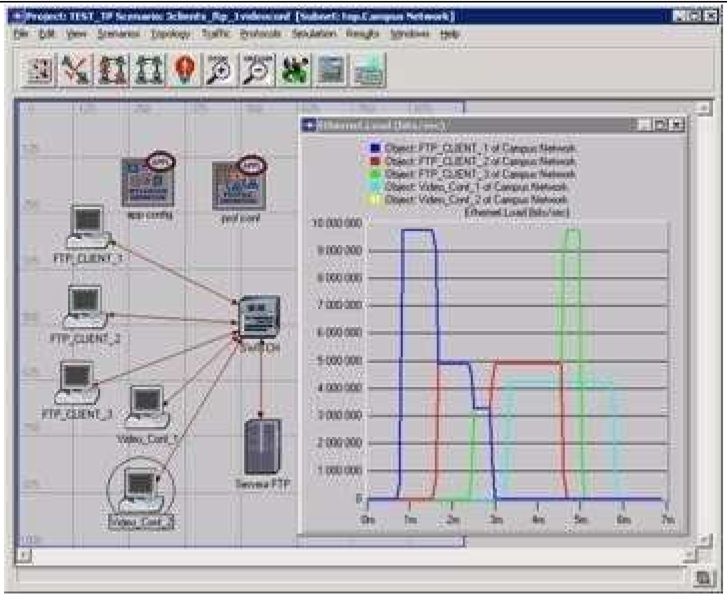


Figure 1. OPNET GUI[OPNET]

OPNET is based on a mechanism called discrete event system which means that the system behavior can simulate by modeling the events in the system in the order of the scenarios the user has set up. Hierarchical structure is used to organize the networks. As other network simulators, OPNET also provides programming tools for users to define the packet format of the protocol. The programming tools are also required to accomplish the tasks of defining the state transition machine, defining network model and the process module.

As of all, OPNET is a popular simulator used in industry for network research and development. The GUI interface and the programming tools are also useful to help the user to build the system they want.

Main features

OPNET inherently has three main functions: modeling, simulating, and analysis. For modeling, it provides intuitive graphical environment to create all kinds of models of protocols. For simulating, it uses 3 different advanced simulations technologies and can be used to address a wide range of studies. For analysis, the simulation results and data can be analyzed and displayed very easily. User friendly graphs, charts, statistics, and even animation can be generated b y OPNET for users ' convenience. According to the OPNET white paper, OPNET's detailed features include:

1. Fast discrete event simulation engine
2. Lot of component library with source code
3. Object-oriented modeling
4. Hierarchical modeling environment
5. Scalable wireless simulations support
6. 32-bit and 64-bit graphical user interface
7. Customizable wireless modeling
8. Discrete Event, Hybrid, and Analytical simulation
9. 32-bit and 64-bit parallel simulation kernel
10. Grid computing support

11. Integrated, GUI-based debugging and analysis
12. Open interface for integrating external component libraries

Recent development and its future

Recently, about at August 7, 2008, OPNET Technologies announced the addition of two major application performance management capabilities. These capabilities include end-to-end visibility into application performance for organizations using WAN optimization solutions and the ability to capture and analyze NetFlow data.

OPNET recently upgrades its ACE Analyst software includes functionality and it is announced to â€œ allow end-user organizations using Riverbed, Cisco, or Juniper WAN optimization appliances to maintain end-to-end visibility into application performance while deploying WAN acceleration solutions â€ [OPNET]. OPNET also provides a module to collect and analyze NetFlow data.

Because of the consistent endeavor and operation of OPNET Inc., OPNET is becoming mature and its product maintain a high acknowledge in the industry. Moreover, OPNET always keeps an eye on the most recent user 's requirements and keeps improving their product which make it very competitive compared with other commercial network simulators in the near expectable future.

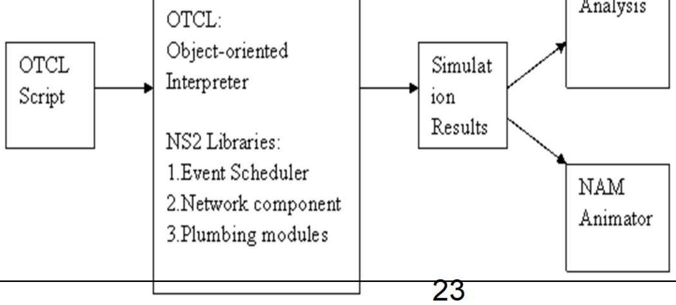
5) Network Simulator 2 (NS2)

NS2 is one of the most popular open source network simulators. The original NS is a discrete event simulator targeted at networking research. In this section, we will give a brief introduction to the NS2 system.

Overview

NS2 is the second version of NS (Network Simulator). NS is originally based on REAL network simulator [REAL]. The first version of NS was developed in 1989 and evolved a lot over the past few years. The current NS project is supported through DARPA. The current second version NS2 is widely used in academic research and it has a lot of packages contributed by different non-benefit groups. For NS2 documentation on recent changes, refer to the NS 2 official webpage [NS2].

Main features

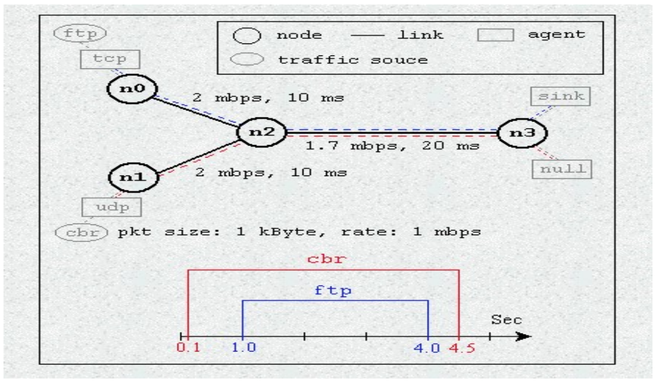

OTCL Script
OTCL: Object-oriented Interpreter
NS2 Libraries: 1.Event Scheduler 2.Network component 3.Plumbing modules
Simulat ion Results
Analysis
NAM Animator

First and foremost, NS2 is an object-oriented, discrete event driven network simulator which was originally developed at University of California-Berkely. The programming it uses is C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT). The usage of these two programming language has its reason. The biggest reason is due to the internal characteristics of these two languages. C++ is efficient to implement a design but it is not very easy to be visual and graphically shown. It's not easy to modify and assembly different components and to change different parameters without a very visual and easy-to-use descriptive language. Moreover, f or efficiency reason, NS2 separates control path implementations from the data path implementation. The event scheduler and the basic network component objects in the data path are written and compiled using C++ to reduce packet and event processing time.

OTcl happens to have the feature that C++ lacks. So the combination of these two languages proves to be very effective. C++ is used to implement the detailed protocol and OTcl is used for users to control the simulation scenario and schedule the events. A simplified user's view of NS2 is shown in figure 2. The OTcl script is used to initiate the event scheduler, set up the network topology, and tell traffic source when to start and stop sending packets through event scheduler. The scenes can be changed easily by programming in the OTcl script. When a user wants to make a new network object, he can either write the new object or assemble a compound object from the existing object library, and plumb the data path through the object. This plumbing makes NS2 very powerful.

Figure2.SimplifiedUser'sViewofNS2

Another feature of NS2 is the event scheduler. In NS2, the event scheduler keeps track of simulation time and release all the events in the event queue by invoking appropriate network components. All the network components use the event scheduler by issuing an event for the packet and waiting for the event to be released before doing further action on the packet.



Recent developments and its future

The most recent version of NS2 is NS 2.33 version which was released on Mar 31, 2008. Compared with the previous version, this newest version [NS2] has integrated the most recent extension on new 802.11 models which include the Ilango Purushothaman's infrastructure mode extensions, the 802.11Ext models from a Mercedes-Benz R&D, NA and University of Karlsruhe team, and the dynamic libraries patch and multi rate 802.11 library from Nicola Baldo and Federico Maguolo of the SIGNET group, University of Padova. NS is now developed in collaboration between some different researchers and institutions, including SAMAN (supported by DARPA), CONSER (through the NSF), and ICIR (formerly ACIRI). Contributions have also come from Sun Microsystems and the UCB and Carnegie Mellon Monarch projects. Generation 3 of NS (NS3) has begun development as of July 1, 2006 and is projected to take four years. It is deemed as the future of NS2, and we will discuss the new generation NS 3 in detail in the following section.

Result:

Thus we have studied about the Network Simulator.

EX.NO :7 SIMPLE CLIENTSERVERPROGRAM-UDP
DATE :

AIM:

To write a java program for UDP

Algorithm:

Client
1. Start
2. Create the UDP datagram socket
3. Get the request message to be sent from the user
4. Send the request message to the server
5. If the request message is “END” go to step 10
6. Wait for the reply message from the server
7. Receive the reply message sent by the server
8. Display the reply message received from the server
9. Repeat the steps from 3 to 8
10. Stop

Server
1. Start
2. Create UDP datagram socket, make it a listening socket
3. Receive the request message sent by the client
4. If the received message is “END” go to step 10
5. Retrieve the client’s IP address from the request message received
6. Display the received message
7. Get the reply message from the user
8. Send the reply message to the client
9. Repeat the steps from 3 to 8.
10. Stop

PROGRAM:

//UDPServer.java

```
import java.io.*;
import java.net.*:
class UDPServer
{
```

```
public static DatagramSocketserversocket;
public static DatagramPacketdp;
public static BufferedReaderdis;
public static InetAddressia;
public static bytebuf[]=newbyte[1024]; public static int cport=789,sport=790;
public static void main(String[]a)throwsIOException

{

serversocket=newDatagramSocket(sport);
dp=newDatagramPacket(buf,buf.length);
dis=newBufferedReader(newInputStreamReader(System.in));
ia=InetAddress.getLocalHost();
System.out.println("ServerisRunning...");
while(true) {
serversocket.receive(dp);
Stringstr=newString(dp.getData(),0,dp.getLength());
 if(str.equals("STOP"))
{
System.out.println("Terminated...");
break;
}
System.out.println("Client: " + str);
 Stringstr1=newString(dis.readLine());
buf=str1.getBytes();
serversocket.send(newDatagramPacket(buf,str1.length(),ia,cport));
}
}
}
```

//UDPClient.java

```
import java.io.*; import java.net.*:

class UDPClient
{

public static DatagramSocketclientsocket;
public static DatagramPacketdp;
public static BufferedReaderdis; public static InetAddressia;
public static bytebuf[]=newbyte[1024]; public static intcport=789,sport=790;

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

{

clientsocket=newDatagramSocket(cport); dp=newDatagramPacket(buf,buf.length);
dis = new BufferedReader(newInputStreamReader(System.in));
ia=InetAddress.getLocalHost();

System.out.println("Client is Running... Type 'STOP' to quit");
while(true)
{
String str=new String(dis.readLine());
buf = str.getBytes(); if(str.equals("STOP"))
{
System.out.println("Terminated...");
clientsocket.send(new DatagramPacket(buf,str.length(), ia,sport));
break;
}
clientsocket.send(new DatagramPacket(buf,str.length(),ia,sport));
clientsocket.receive(dp);
Stringstr2=newString(dp.getData(),0,dp.getLength());
System.out.println("Server:"+str2);
}
}}
```

OUTPUT:

#UDPServer.java

```
D:\BECSE\Sem4\Lab\NetworksLab\Programs>javacUDPServer.java

D:\BECSE\Sem4\Lab\NetworksLab\Programs>javaUDPServerServerisRunning…
Client: Hii Hello
```

#UDPClient.java

```
D:\BECSE\Sem4\Lab\NetworksLab\Programs>javacUDPClient.java

D:\BECSE\Sem4\Lab\NetworksLab\Programs>javaUDPClient Client is Running... Type 'STOP' to
quit


Hii Hello
```

RESULT:

Thus the java program for UDP is successfully executed

EX.NO: 8 SIMULATION OF ERROR CORRECTION CODE (like CRC)
DATE:

Aim:

To write a java program for the simulation of error correction code (like CRC).

Program:

```
 import java.io.*;
 class crcgen
{
public static void main (String args[]) throws IOException
{
BufferedReaderbr=new BufferedReader(newInputStreamReader(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());

/*  System.out.print("Data bits are : ");

for(int i=0; i<data_bits; i++)
System.out.print(data[i]);
 System.out.println();

System.out.print("divisor bits are :");
for(int i=0; i<divisor_bits; i++)
System.out.print(divisor[i]);
 System.out.println();
```

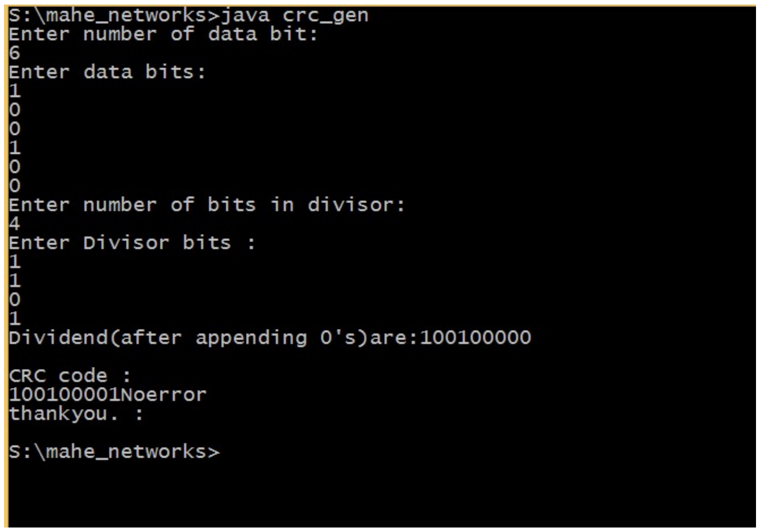
```
*/ tot_length=data_bits+divisor_bits-1;
  div=new int[tot_length];
  rem=new int[tot_length];
  crc=new int[tot_length];
   /*------------------CRCGENERATION------------------*/
  for(int i=0;i<data.length;i++)
     div[i]=data[i];
  System.out.print("Dividend(afterappending0's)are:");
  for(inti=0; i<div.length ;i++)
      System.out.print(div[i]);
      System.out.println();
      for (intj=0;j<div.length;j++)
      {
           rem[j] = div[j];
      }
      rem=divide(div,divisor,rem);
      for (inti=0;i<div.length;i++)      //append dividend and remainder
      {
         crc[i]=(div[i]^rem[i]);
      }
      System.out.println();
      System.out.println("CRCcode:");
       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());
/*    System.out.print("crc bits are : ");
      for (int i=0; i<crc.length; i++)
           System.out.print(crc[i]);
           System.out.println();
*/
      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("NoError");
        }

        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 bitsare : 101
Dividend(afterappending0's)are:101100100CRCcode:101100111
Enter CRC code of 9 bits: 10
1
1
0
```

0
1
0
1
crcbitsare:101100101ErrorTHANKYOU.   ) Press any key to continue...



```
S:\mahe_networks>java crc_gen
Enter number of data bit:
6
Enter data bits:
1
0
0
1
0
0
Enter number of bits in divisor:
4
Enter Divisor bits :
1
1
0
1
Dividend(after appending 0's)are:100100000

CRC code :
100100001Noerror
thankyou. :

S:\mahe_networks>
```

RESULT:

Thus a java program for the simulation of error correction code(like CRC) is executed.

EX.NO:9 STUDY OF TCP/UDP PERFORMANCE USING
DATE: SIMULATION TOOLS

AIM:
To perform a comparative study TCP and UDP Protocols in different simulation scenarios.

Comparison Study between TCP and UDP:

TCP is represented as a connection-oriented protocol. TCP presents end-to-end communications. Moreover, when the communication is created among the transmitter and receiver, the data can be send over that communication. While the UDP is a simple connectionless protocol. UDP does not constitute a dedicated end-to-end communication among the transmitter and the receiver before the real communication takes place. However, the data is being transported in one trend from the transmitter to the receiver with no need to verify the receiver case. Figure 1 shows the segment fields of TCP and UDP.

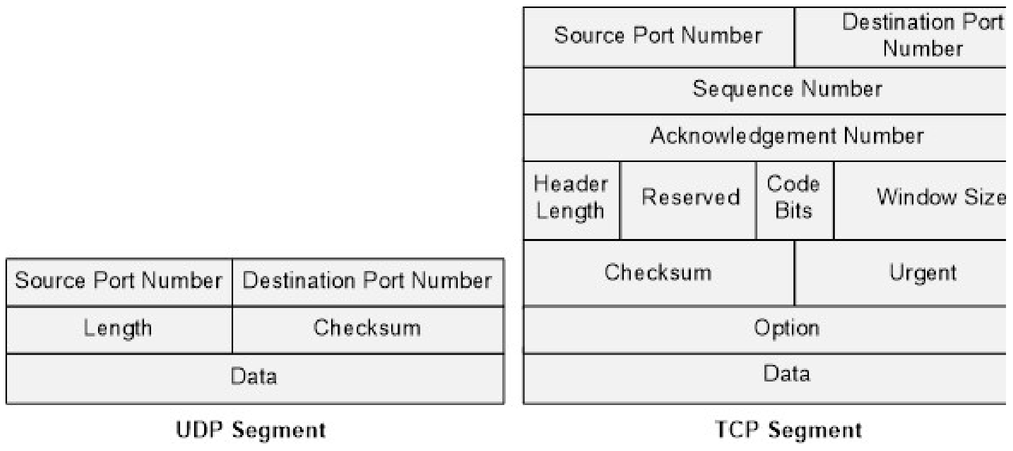
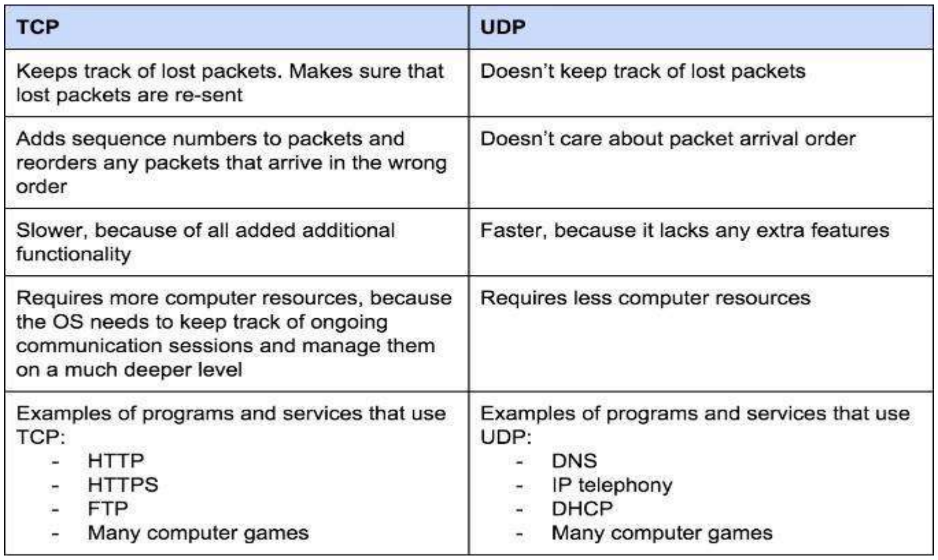


Fig. 1. The Segment Fields of TCP and UDP

The UDP and TCP are various on the basic operations and applications. The differences in the data transmission, the TCP presents ordered and reliable delivery of data from the user to the server and vice versa. UDP is considered as a connectionless protocol and does not provide the reliable delivery for the data. TCP is more reliable compared to the UDP, where TCP uses there transmissions and the message acknowledgment if there is some loss in the packets. Therefore, there is no losing data in the network. While in the case of UDP does not guarantee that the data has arrived to the receiver or not.

Also, in UDP there is no retransmission, timeout and message acknowledgment. TCP transmits the messages in an order and these messages are received in the same order at the destination. If the packets of the data reach in the wrong order, TCP can reorder the data packets. Whilst in UDP, the sequence of the message is not maintained over the transmission. TCP records the data as a stream of bytes and sending the message as segments. The messages in UDP are sending as datagrams in the network. So, both of TCP and UDP have various approaches of sending and receiving the data. Figure 2 shows a comparative among TCP and UDP.



TCP	UDP
Keeps track of lost packets. Makes sure that lost packets are re-sent	Doesn't keep track of lost packets
Adds sequence numbers to packets and reorders any packets that arrive in the wrong order	Doesn't care about packet arrival order
Slower, because of all added additional functionality	Faster, because it lacks any extra features
Requires more computer resources, because the OS needs to keep track of ongoing communication sessions and manage them on a much deeper level	Requires less computer resources
Examples of programs and services that use TCP: - HTTP - HTTPS - FTP - Many computer games	Examples of programs and services that use UDP: - DNS - IP telephony - DHCP - Many computer games

Fig. 2. The Comparison between TCP and UDP [17,18]

Simulation Scenarios:
NS2 simulator is used in this study to evaluate and analyze the behavior of both TCP and UDP protocols. This simulation has been presented two wired scenarios to carefully verify the behavior of these protocols. Where in the first scenario the bandwidth is varied from 0.1 Mb/ms to 0.5 Mb/ms and the packet size is fixed at 64 bytes. While in the second scenario the packet size is varied from 800 bytes to 1000 bytes and the bandwidth is fixed at
Mb/ms. Simulation parameters shows in table1.The nodes number in this study is 8 and the simulation time is 64 second in both scenarios [19,20]. Figure3 illustrates the wired simulation environment.

Parameters	Values
Simulator	NS2.35
Number of Nodes	8
Simulation Time	64 sec
Protocols	TCP and UDP
Bandwidth in Ist scenario	0.1, 0.2, 0.3, 0.4, 0.5
Packet Size in Ist scenario	64
Bandwidth in IInd scenario	0.3
Packet Size in IInd scenario	800, 850, 900, 950, 1000

Table 1. Simulation Parameters

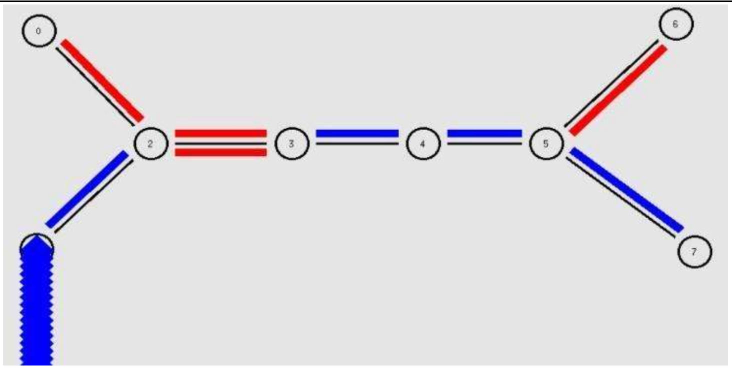


Fig. 3. The Simulation Environment

Performance Metrics

In our comparison, we have utilized various network behavior metrics among UDP and TCP.These metrics are applied toevaluate and analyze protocols performance.

Packet Delivery Ratio(PDR)

PDR is the percentage of data packets transported to the destination to those produced by the sources. PDR is calculated as follow:

$$PDR(\%) = \frac{\sum \text{No.of packets received}}{\sum \text{No.of packets sent}} * 100$$

Average Throughput (TP)

It is the bytes successfully received number and it is calculated as follow:

$$TP = \text{No. of Bytes Received} * 8 * \text{Simulation Time} * 1000\text{kbpsb}$$

Average End-to-End Delay (e2edelay)

It is the mean time of the successfully transmitted data packet over the network from the source to the destination. It is computed as follow:

$$e2edelav = \frac{\sum \text{arrivetime} - \text{sendtime}}{\sum \text{number of connection}}$$

Packet Loss (PL)

It is the difference among the data packets transmitted and the data packets received. It is calculated as follow:

PL=No. of Data Packets Sent – No. of Data Packets Receive

Network Metrics

In this our simulation, there are two various kinds of network parameters which are varying through the simulation experiments:

- Bandwidth: It is the data number that transfer from the source to the destination.
- Packet size: A packet is the unit of data which is routed between the source and destination.

Simulation Results and Discussion

In this simulation, we have analyzed and compared the TCP and UDP protocols utilizing NS2 in two scenarios. The first scenario has been simulated in different bandwidth and in the second scenario we have used different packet size. According to the results obtained in the first scenario, the performance of both protocols has been much different from each other. Where in Figure 4 the TCP has achieved 700.71 of throughput and UDP has achieved 687.1 that means TCP receives data more than UDP. Figure 5 indicates the behavior of the protocols in the e2e delay, where TCP has been achieved average e2edelay to 0.62018sec and UDP has been achieved 0.98376 sec. It is obvious, there is no huge difference between the protocols in e2e delay. In terms of PDR as shown in Figure 6, the behavior of TCP showed much better than UDP, where TCP has been obtained 100% of the network and UDP has obtained 4.04 of PDR. However, TCP has 0% of PL and UDP has 95.96 as shown in Figure 7. Therefore, the results in the first scenario have shown that TCP is much better than UDP in terms of all performance measures.

In the second scenario, it has been used different packet size inboth protocols. The results have shown that the behavior of TCP is outperformed behavior of UDP in mean throughput as shown in Figure 8, where TCP has 580.67 of throughput and UDP has 302.67. Figure9 shows the performance of the protocols in average e2e delay,

TCP has achieved 0.93883 sec and UDP has achieved 2.84602sec.Thus, TCP performance is faster than UDP in send and receive the data. In Figure 10, the performance of TCP is out performed UDP in the PDR, where TCP has been achieved 95.7 value of PDR and UDP has achieved 3 of PDR. However, Figure 11illustrates the behavior of the protocols in PL, TCP has achieved 4.74 of PL and UDP has achieved 97 value of PL. Therefore, the results in the second scenario have shown that TCP is much better than UDP in terms of all performance metrics.

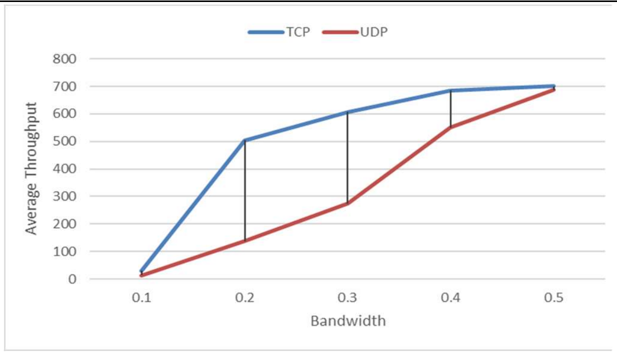


Fig. 4. TP versus Bandwidth

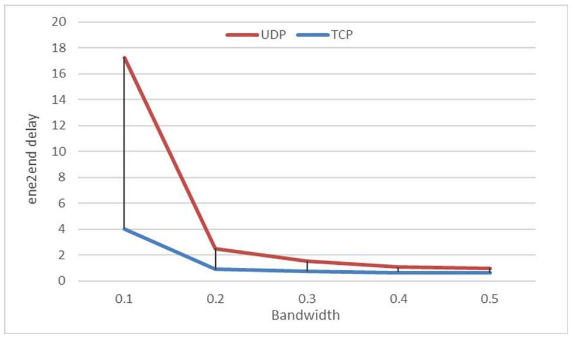


Fig. 5. e2eDelay versus Bandwidth

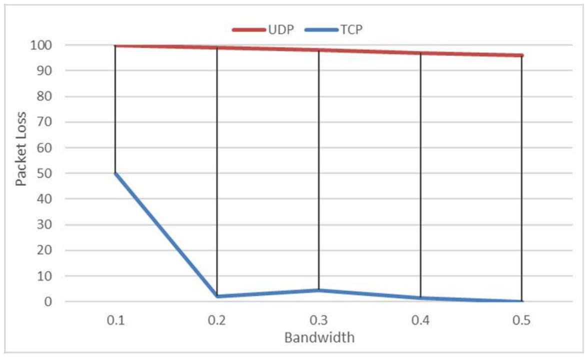


Fig. 6. PL versus Bandwidth

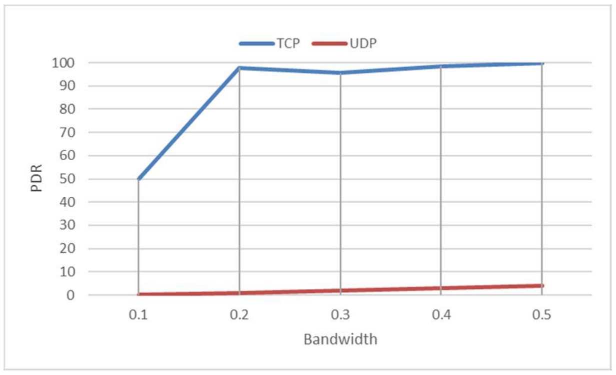


Fig. 7. PDR versus Bandwidth

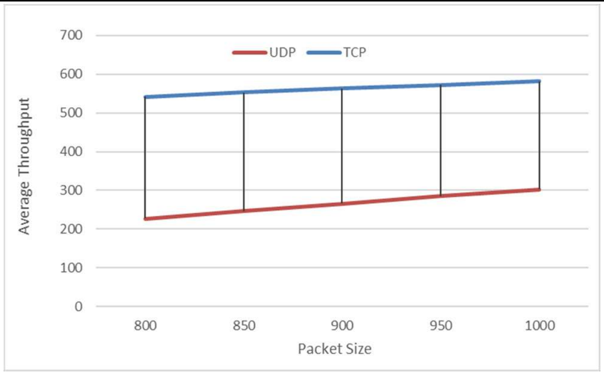


Fig. 8. TP versus Packet Size

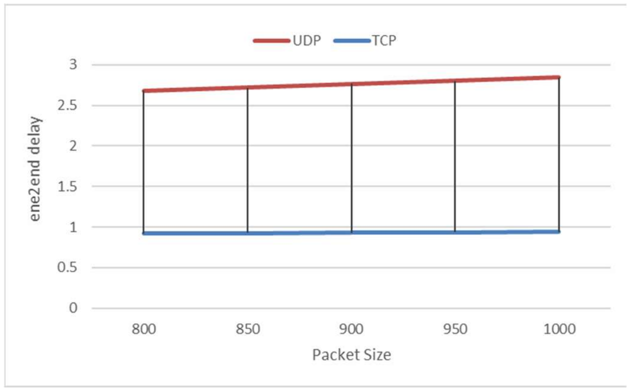


Fig. 9. e2edelay versus Packet Size

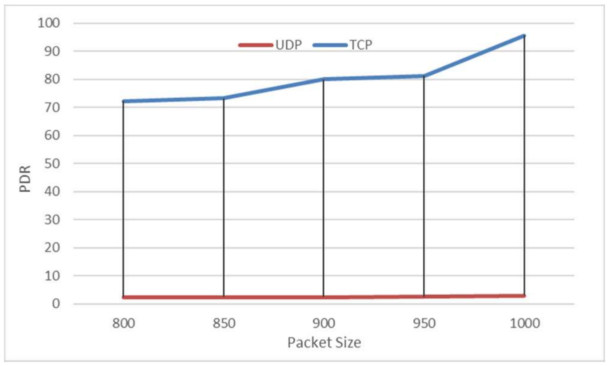


Fig. 10. PDR versus Packet Size

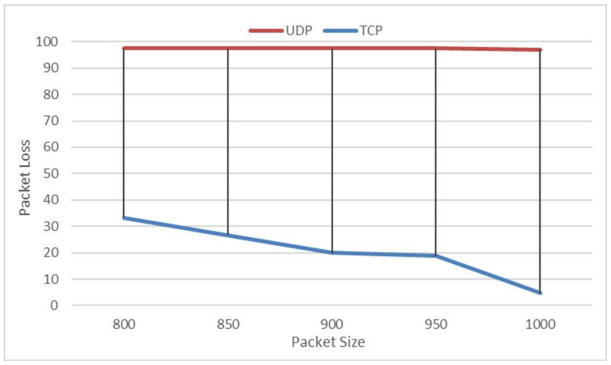


Fig. 11. PL versus Packet Size

Conclusion:

TCP and UDP are a transportation layer protocols which are considered of the basic protocols of the internet. The performance of these protocols in various network parameters and scenarios is still not so clear. Therefore, in this paper, we have analyzed and compared the behavior of both TCP and UDP in two different scenarios to accurately determine which of these protocol is better. The simulation has been used NS2 to assess the behavior of TCP and UDP in varying packet size and bandwidth. These two protocols were measured in terms of the mean end-to-end delay, mean throughput, packet delivery percentage, and packet. The results have shown that the performance of TCP is out performed the UDP in both of the two scenarios.