INDEX

CLIENT SERVER SOCKET PROGRAM TO DISPLAY DATE AND TIME

Ex.No: 1a

Date :

AIM:

Implement a Socket program to display date and time using TCP socket

ALGORITHM:

SERVER:

Step 1: Create a server socket and bind it to the port.

Step 2: Listen for a new connection and when a connection arrives,accept it.

Step 3: Send server's date and time to the client.

Step 4: Read the client's IP address sent by the client.

Step 5: Display the client details.

Step 6: Repeat steps 2-5 until the server is terminated.

Step 7: Close all streams**.**

Step 8: Close the server socket.

Step 9: Stop.

CLIENT:

Step 1: Create a client socket and connect it to the server's port number.

Step 2: Retrieves its own IP address using built-in function.

Step 3: Send its address to the server.

Step 4: Display the date and time sent by the server.

Step 5: Close input and output streams.

Step 6: Close the client socket.

Step 7: Stop.

PROGRAM:

//TCP Date Server

```
import java.net.*;

import java.io.*;

import java.util.*;

class tse

{

public static void main(String[] args)

{

ServerSocket ss=null;

Socket cs;

PrintStream ps;

BufferedReader dis;

String inet;

try

{

ss=new ServerSocket(4444);

System.out.println("Press Ctrl+C to quit");

while(true)
{

cs=ss.accept();

ps=new PrintStream(cs.getOutputStream());

Date d=new Date();

ps.println(d);

dis=new BufferedReader(new InputStreamReader(cs.getInputStream()));

inet=dis.readLine();

System.out.println("ClientSystem I/P address is:"+inet);

ps.close();

dis.close();

}}

catch(IOException e)

{

System.out.println("The exception is:"+e);

}}}
```

//TCP Date Client

```
import java.net.*;

import java.io.*;

class tcl

{

public static void main(String[] args){

Socket soc;
```

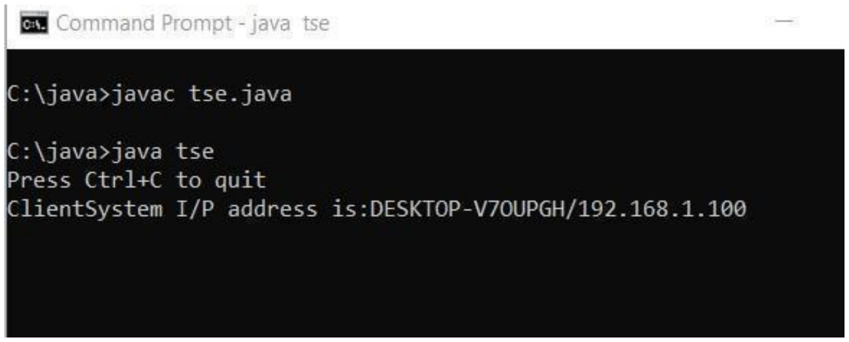
```
BufferedReader dis;  //BufferedReader used to read text from a character-input stream to provide
the efficient reading of sequence of characters.

String sdate;

PrintStream ps;    //    PrintStream provides  methods to write data to another stream and it
automatically flushes the data.

try

{

InetAddress ia=InetAddress.getLocalHost();

if(args.length==0)

soc=new Socket(InetAddress.getLocalHost(),4444);

else

soc=new Socket(InetAddress.getByName(args[0]),4444);   //determine the ip address of the
host,given the host name

dis=new BufferedReader(new InputStreamReader(soc.getInputStream()));

sdate=dis.readLine();

System.out.println("The date/time on server is:"+sdate);

ps=new PrintStream(soc.getOutputStream());

ps.println(ia);

ps.close();

dis.close();

}

catch(IOException e)

{

System.out.println("The exception is:"+e);

}
```

}

}

OUTPUT:

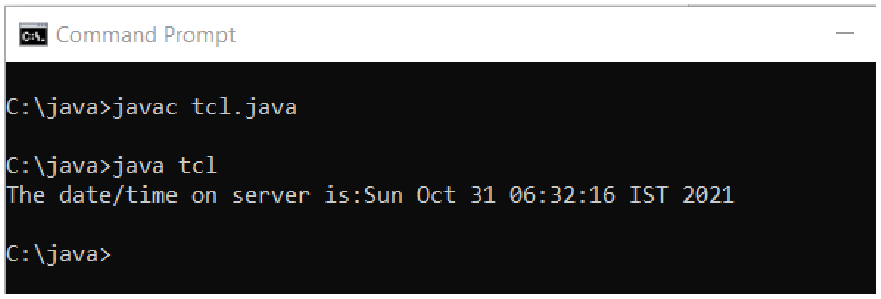
<u>SERVER:</u>



Command Prompt - java tse

```
C:\java>javac tse.java

C:\java>java tse
Press Ctrl+C to quit
ClientSystem I/P address is:DESKTOP-V7OUPGH/192.168.1.100
```

<u>CLIENT:</u>



Command Prompt

```
C:\java>javac tcl.java

C:\java>java tcl
The date/time on server is:Sun Oct 31 06:32:16 IST 2021

C:\java>
```

RESULT:

The given program was successfully executed

IMPLEMENTATION OF CHAT APPLICATION USING TCP SOCKETS

Ex.No:1b

Date:

AIM :Implementation of chat application using TCP Sockets

ALGORITHM :

SERVER:

Step 1: Start the program

Step 2: Create server and client sockets.

Step 3: Use input streams to get the message from user.

Step 4: Use output streams to send message to the client.

Step 5: Wait for client to display this message and write a new one to be displayed by the server.

Step 6: Display message given at client using input streams read from socket.

Step 7: Stop the program.

CLIENT :

Step 1: Start the program

Step 2: Create a client socket that connects to the required host and port.

Step 3: Use input streams read message given by server and print it.

Step 4: Use input streams; get the message from user to be given to the server.

Step 5: Use output streams to write message to the server.

Step 6: Stop the program.

PROGRAM:

CLIENT:

//talkclient.java

```
import java.io.*;

import java.net.*;

public class talkclient

{

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

{

Socket c=null;

DataInputStream usr_inp=null;

DataInputStream din=new DataInputStream(System.in);

DataOutputStream dout=null;

try

{

c=new Socket("127.0.0.1",1234);

usr_inp=new DataInputStream(c.getInputStream());

dout=new DataOutputStream(c.getOutputStream());

}

catch(IOException e) {}
if(c!=null || usr_inp!=null || dout!=null)

{

String unip;

System.out.println("\nEnter the message for server:");

while((unip=din.readLine())!=null)

{

dout.writeBytes(""+unip);

dout.writeBytes("\n");

System.out.println("reply");

System.out.println(usr_inp.readLine());

System.out.println("\nEnter your message:");

}

System.exit(0);

}

din.close();

usr_inp.close();

c.close();

}

}
```

SERVER:

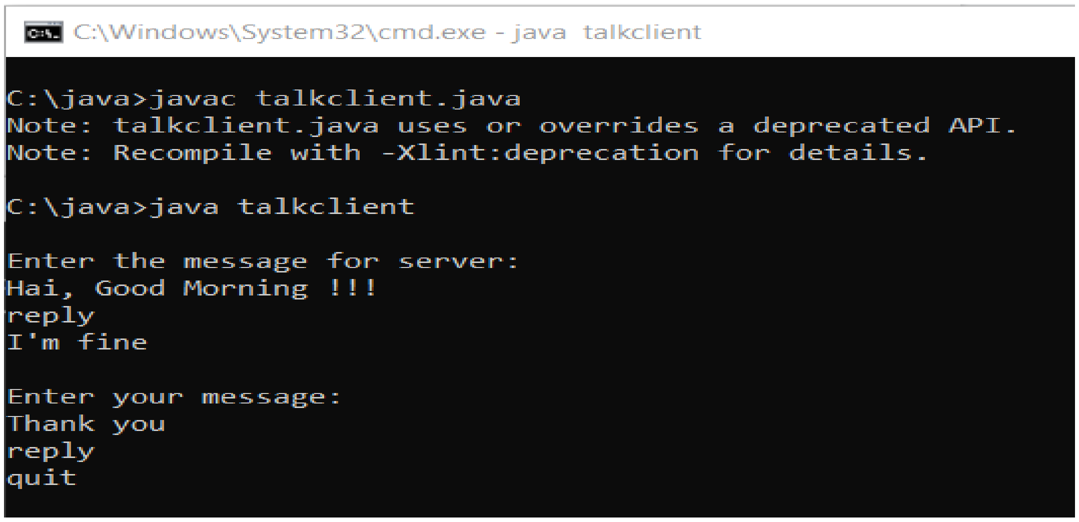
//talkserver.java

```
import java.io.*;

import java.net.*;

public class talkserver

{

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

{

ServerSocket m=null;

Socket c=null;

DataInputStream usr_inp=null;

DataInputStream din=new DataInputStream(System.in);

DataOutputStream dout=null;

try

{

m=new ServerSocket(1234);

c=m.accept();

usr_inp=new DataInputStream(c.getInputStream()); dout=new

DataOutputStream(c.getOutputStream());

}

catch(IOException e) {}

if(c!=null||usr_inp!=null)

{
```

```
String unip; while(true)

{

System.out.println("\nMessage from client:");

String m1=usr_inp.readLine();

System.out.println(m1);

System.out.println("Enter your message:");

unip=din.readLine();

dout.writeBytes(""+unip);

dout.writeBytes("\n");

}

}

dout.close();

usr_inp.close();

c.close();

}

}
```

OUTPUT:

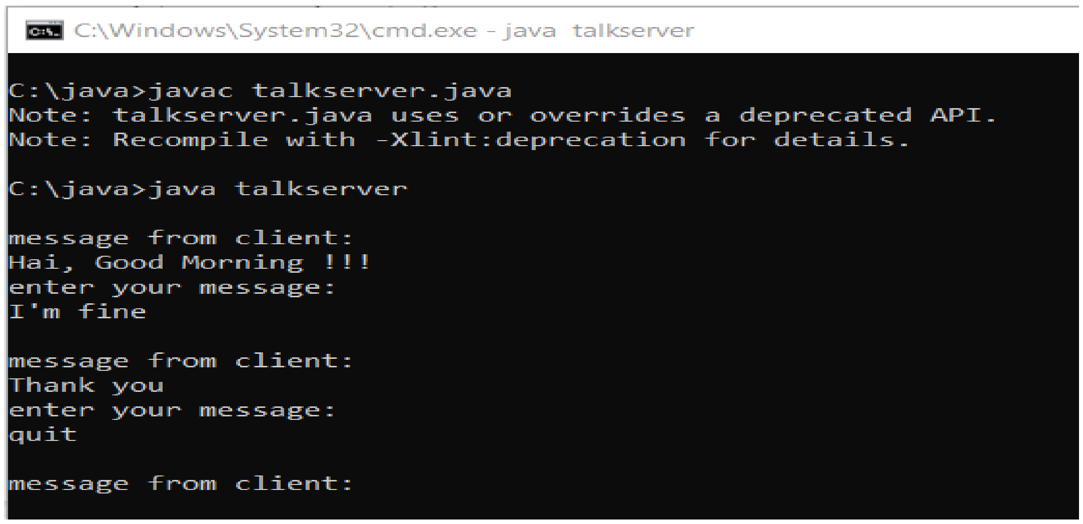
CLIENT:



C:\Windows\System32\cmd.exe - java talkclient

```
C:\java>javac talkclient.java
Note: talkclient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\java>java talkclient

Enter the message for server:
Hai, Good Morning !!!
reply
I'm fine

Enter your message:
Thank you
reply
quit
```

SERVER:



C:\Windows\System32\cmd.exe - java talkserver

```
C:\java>javac talkserver.java
Note: talkserver.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\java>java talkserver

message from client:
Hai, Good Morning !!!
enter your message:
I'm fine

message from client:
Thank you
enter your message:
quit

message from client:
```

RESULT:

The given program was successfully executed

IMPLEMENTATION OF FILE TRANSFER APPLICATION USING TCP SOCKETS

Ex.No:1c

Date:

AIM: Implementation of File Transfer using TCP Sockets

ALGORITHM:

Step 1: Develop the server and client side programs separately. Compile and execute them on two different console windows.

Step 2: Run the server side program first and allow it to wait for a client connection. Then, when the client program is executed, it connects to the server automatically, provided the sockets used in client and server program both use the same port.

Step 3: Simulate the file transfer from the client side using a simple textfile.

PROGRAM:

SERVER:

```
import java.io.BufferedInputStream;

import java.io.File;

import java.io.FileInputStream;

import java.io.OutputStream;

import java.net.InetAddress;

import java.net.ServerSocket;

import java.net.Socket;

public class Fileserver{

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

ServerSocket ssock =new ServerSocket(5000);

Socket socket = ssock.accept();

InetAddress IA = InetAddress.getByName("localhost");
```

```
File file=new File("file1.txt");

FileInputStream fis=new FileInputStream(file);

BufferedInputStream bis =new BufferedInputStream(fis);

OutputStream os =socket.getOutputStream();

byte[] contents;

long fileLength=file.length();

long current=0;

long start =System.nanoTime();

while(current!=fileLength){

        int size=10000;

        if(fileLength-current >= size){

        current+=size;

        }

        else{

        size=(int)(fileLength - current);

        current=fileLength;

        contents= new byte[size];

        bis.read(contents,0,size);

        os.write(contents);

        System.out.print("Sending file ..."+ (current*100)/fileLength+"% completed.");

        }

        os.flush();

        socket.close();
```

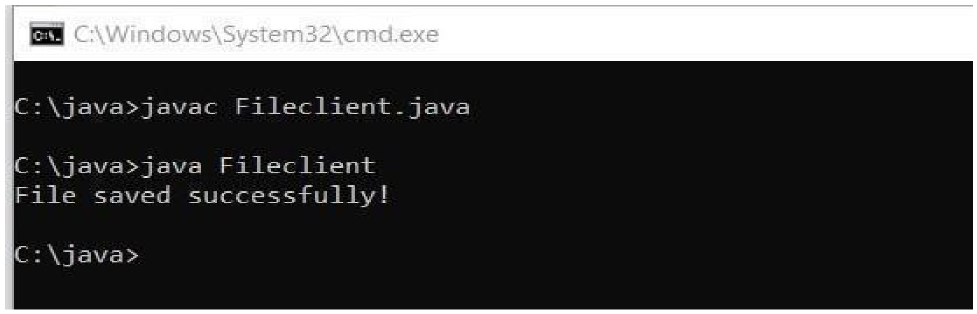
```
            ssock.close();

        System.out.println("File sent successfully");

}

}}
```

CLIENT:

```
import java.io.FileOutputStream;
import java.io.InputStream;

import java.net. InetAddress;

import java.net.Socket;

public class Fileclient {

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

Socket socket = new Socket (InetAddress.getByName("localhost"),5000);

byte[] contents = new byte[10000];

FileOutputStream fos = new FileOutputStream("file2.txt");

BufferedOutputStream bos = new BufferedOutputStream(fos);

InputStream is = socket.getInputStream();

int bytesRead = 0;

while((bytesRead=is.read(contents)) !=-1)

bos.write(contents, 0, bytesRead);

bos.flush();

socket.close();

System.out.println("File saved successfully!");

}}
```

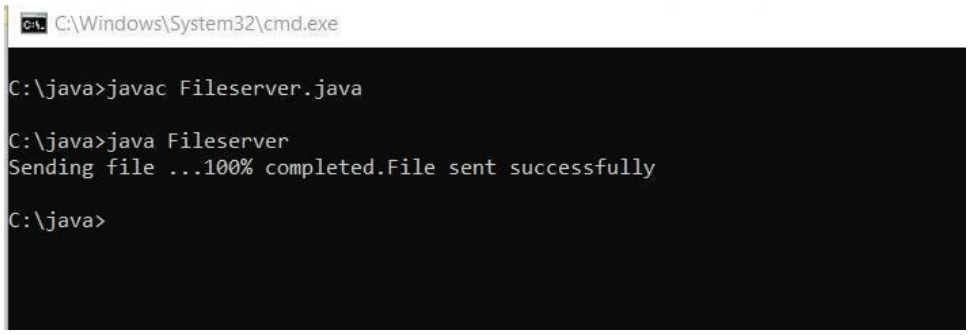
OUTPUT:

CLIENT:

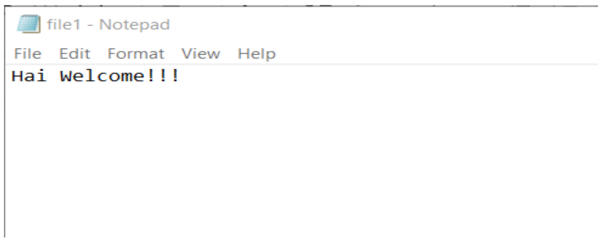


```
C:\java>javac Fileclient.java

C:\java>java Fileclient
File saved successfully!

C:\java>
```

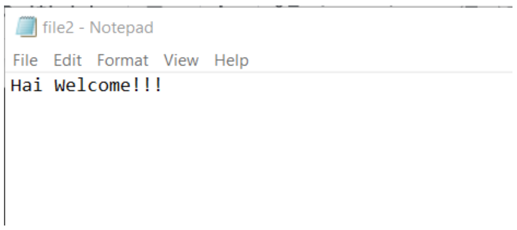
SERVER:



```
C:\java>javac Fileserver.java

C:\java>java Fileserver
Sending file ...100% completed.File sent successfully

C:\java>
```



```
Hai Welcome!!!
```



```
Hai Welcome!!!
```

RESULT: The given program was successfully executed

IMPLEMENTATION OF ECHO CLIENT / SERVER APPLICATION USING TCP SOCKETS

Ex.No: 2

Date :

AIM:

ALGORITHM:

<u>CLIENT:</u>

Step 1: Start the program.

Step 2: Get the frame size from the user

Step 3: To create the frame based on the user request.

Step 4: To send frames to the server.

Step 5: If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.

Step 6 : Stop the program.

<u>SERVER:</u>

Step 1: Start the program.

Step 2: Get the frame size from the user

Step 3: To create the frame based on the user request.

Step 4: To receive frame from client side.

Step 5: If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.

Step 6: Stop the program.

PROGRAM:

CLIENT:

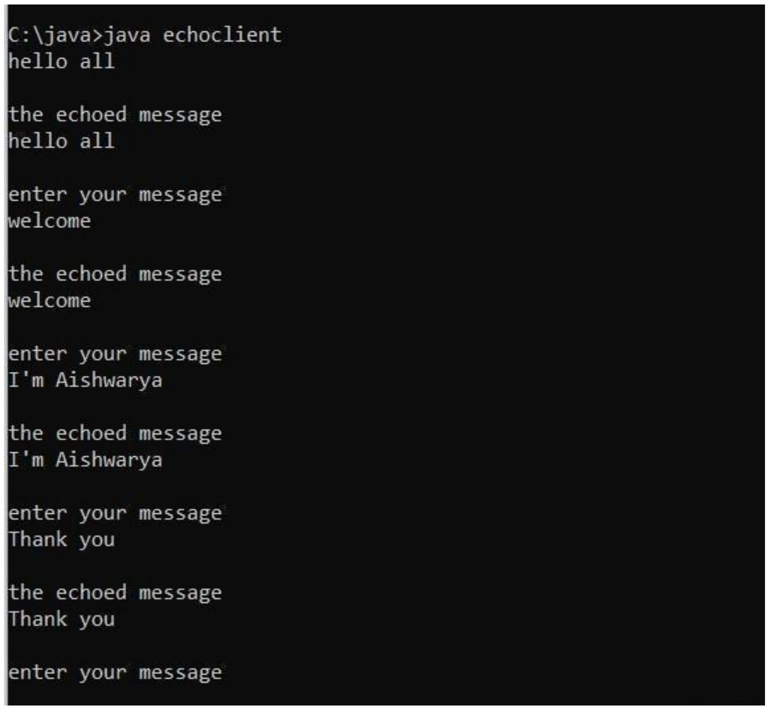
```
import java.io.*;

import java.net.*;

import java.util.*;

public class echoclient

{

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

{

Socket c=null;

DataInputStream usr_inp=null;

DataInputStream din=new DataInputStream(System.in);

 DataOutputStream dout=null;

try{

c=new Socket("127.0.0.1",5678);

usr_inp=new DataInputStream(c.getInputStream());

dout=new DataOutputStream(c.getOutputStream());

}

catch(IOException e) {}

if(c!=null || usr_inp!=null || dout!=null)

{

String unip;

while((unip=din.readLine())!=null){

dout.writeBytes(""+unip);

dout.writeBytes("\n");

System.out.println("\nthe echoed message");
System.out.println(usr_inp.readLine());
System.out.println("\nEnter your message");
}
System.exit(0);
}
din.close();
usr_inp.close();
c.close();
}}
```

SERVER:

```
import java.io.*;
import java.net.*;
public class echoserver{
public static void main(String args[])throws Exception{
         ServerSocket m=null;
        Socket c=null;
        DataInputStream usr_inp=null;
        DataInputStream din=new DataInputStream(System.in);
        DataOutputStream dout=null;
try{
m=new ServerSocket(5678);
c=m.accept();
usr_inp=new DataInputStream(c.getInputStream());
dout=new DataOutputStream(c.getOutputStream());
}
catch(IOException e) {}
if(c!=null || usr_inp!=null)
{
String unip;
while(true)
{
        System.out.println("\nMessage from Client...");
        String m1=(usr_inp.readLine());
        System.out.println(m1);
        dout.writeBytes(""+m1);
        dout.writeBytes("\n");
}
}
dout.close();
usr_inp.close();
c.close();
}
}
```

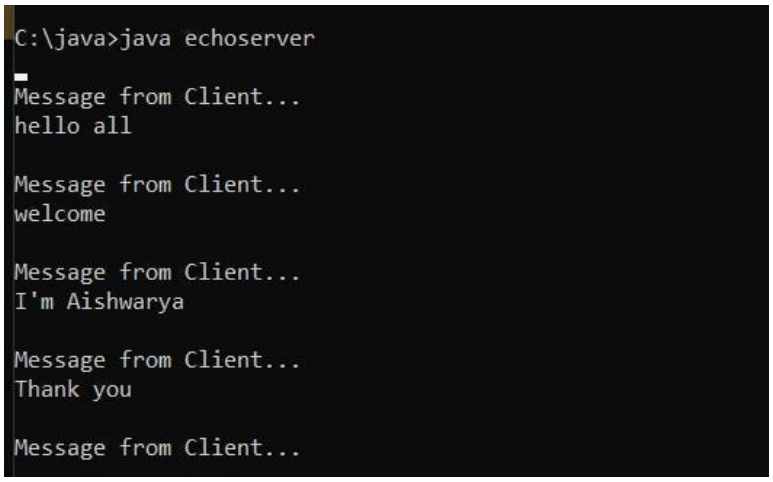
OUTPUT:

CLIENT:



```
C:\java>java echoclient
hello all

the echoed message
hello all

enter your message
welcome

the echoed message
welcome

enter your message
I'm Aishwarya

the echoed message
I'm Aishwarya

enter your message
Thank you

the echoed message
Thank you

enter your message
```

SERVER:



```
C:\java>java echoserver

Message from Client...
hello all

Message from Client...
welcome

Message from Client...
I'm Aishwarya

Message from Client...
Thank you

Message from Client...
```

RESULT: The given program was successfully executed

SIMULATION OF LINK STATE ROUTING ALGORITHM

EX.NO:3

DATE:

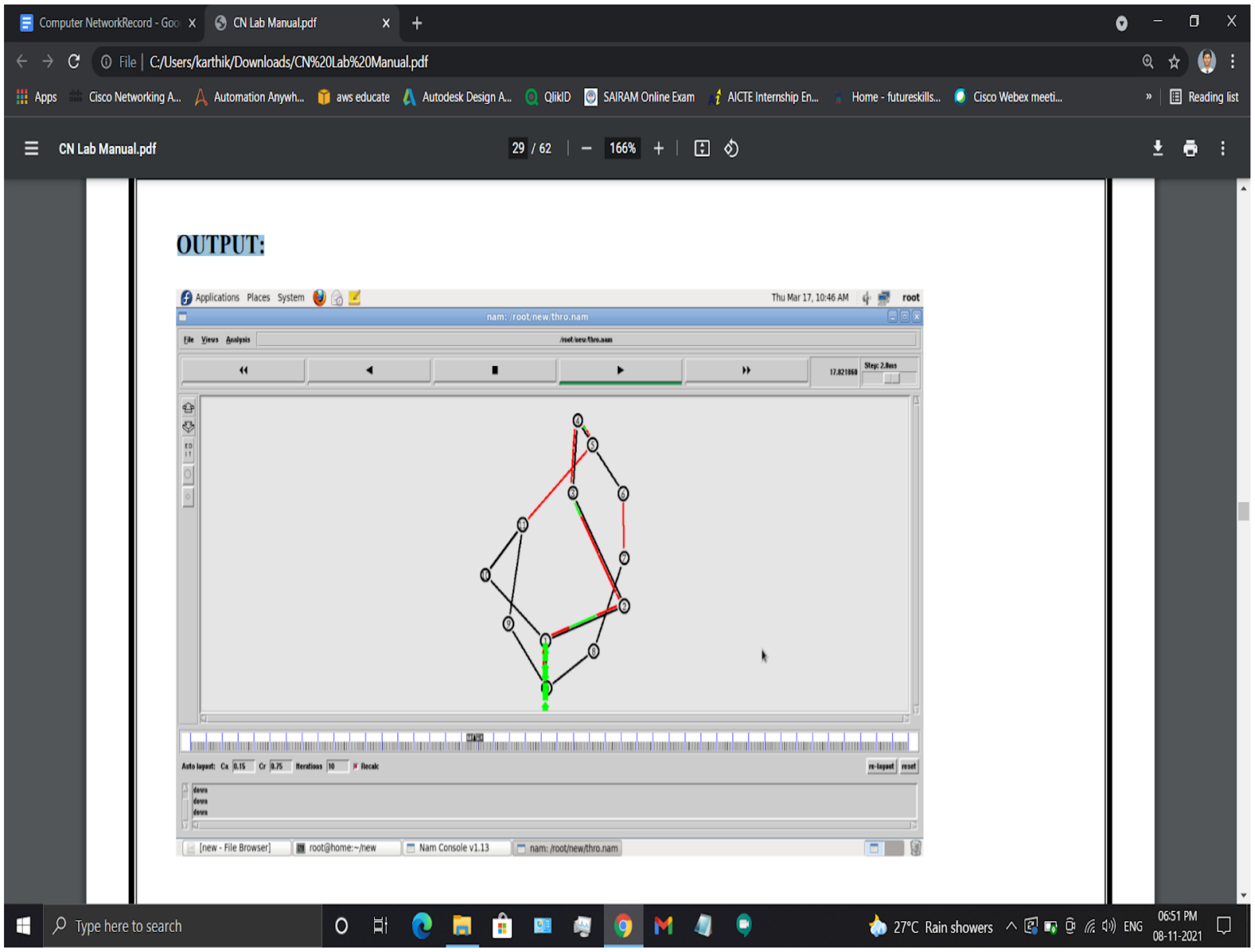
AIM:
To simulate and study the link state routing algorithm using simulation.

ALGORITHM:
Step 1. Create a simulator object
Step 2. Define different colors for different data flows
Step 3. Open a nam trace file and define the finish procedure then close the trace file, and execute nam on trace file.
Step 4. Create n number of nodes using for loop
Step 5. Create duplex links between the nodes
Step 6. Setup UDP Connection between n(0) and n(5)
Step 7. Setup another UDP connection between n(1) and n(5)
Step 8. Apply CBR Traffic over both UDP connections
Step 9. Choose Link state routing protocol to transmit data from sender to receiver.
Step 10. Schedule events and run the program.

PROGRAM:
```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]

$ns namtrace-all $nf
proc finish { } {
global ns nr nf
      $ns flush-trace
close $nf
close $nr
exec nam thro.nam &
   exit 0
            }

for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
 $ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail
```

```
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

 $ns rtproto LS
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
 $ns color 2 Green

$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45 "finish"
$ns run
```

OUTPUT:



RESULT:

The given program was successfully executed

SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

EX.NO: 4

DATE:

AIM: To simulate and study the Distance Vector routing algorithm using simulation

ALGORITHM:

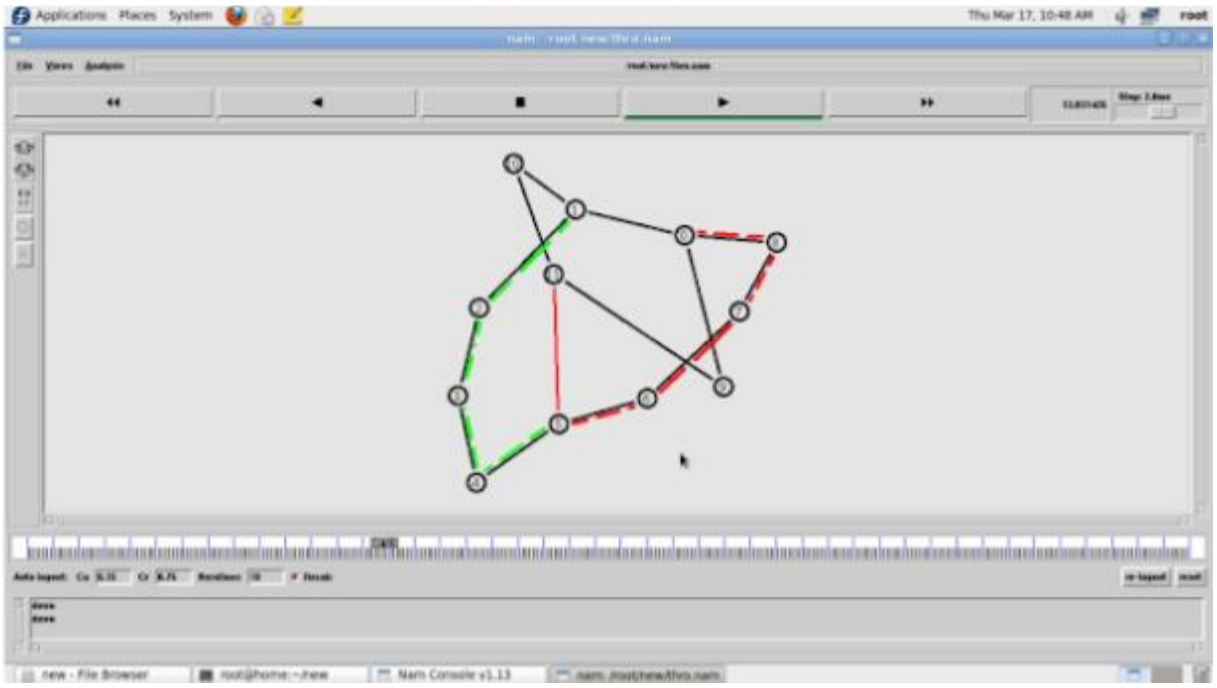
1. Create a simulator object
2. Define different colors for different data flows
3. Open a nam trace file and define the finish procedure then close the trace file, and execute nam on trace file.
4. Create n number of nodes using for loop
5. Create duplex links between the nodes
6. Setup UDP Connection between n(0) and n(5)
7. Setup another UDP connection between n(1) and n(5)
8. Apply CBR Traffic over both UDP connections
9. Choose distance vector routing protocol to transmit data from sender to receiver.
10. Schedule events and run the program.

PROGRAM:

```
set ns [new Simulator]
set nr [open thro.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf
proc finish { } {
global ns nr nf
   $ns flush-trace
close $nf
close $nr
exec nam thro.nam & exit 0 }
for { set i 0 } { $i < 12} { incr i 1 } {
set n($i) [$ns node]}
for {set i 0} {$i < 8} {incr i} {
$ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail }
$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
set null0 [new Agent/Null]
```

```
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

$ns rtproto DV $ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
 $ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1
$udp1 set fid_ 2
$ns color 1 Red
$ns color 2 Green

$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"

$ns at 45 "finish"
$ns run
```

OUTPUT:



RESULT:

The given program was successfully executed

STUDY OF NETWORK SIMULATOR (NS)

EX.NO: 5

DATE:

AIM: To study about the Network Simulator(NS).

1) 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 functions. 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 engineering such as civil engineering, structural engineering, mechanical engineering, and computer engineering. Application of simulation technology into networking areas 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 keep 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 on 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 network 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 mobile units.

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 easily 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 the researcher 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 network. 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 is 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 a network under planning is simulated in order to assess its performance or to predict the impact of possible changes, or optimizations. The major difference 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 NS2 which is a popular network simulator that can also be used as a limited-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 some criteria such as if they are commercial or free, or if they are simple ones or complex ones.

1. 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 users 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 simulators have their advantages and disadvantages. The advantage is that it generally has complete and up-to-date documentation 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 documentation.

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 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 disadvantages for the open source network simulators. Lack of enough systematic and complete documentations and lack of version control support can lead to some serious problems and can limit the applicability and life-time of the open source network simulators. Typical open source network simulators include NS2 [NS2][NS2-wiki], NS3[NS3]. We will introduce and analyze them in great detail in the following sections.

Table 1 Network simulators

	Network simulators name
Commercial	OPNET, QualNet
Open source	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.

2 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 to customize to create an application that simulates the networking environment for testing.

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 it in the first place. NS2 is the most popular one in academia because of its open-source and library components. 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 limitations of NS2, the NS3 is currently under development and testing. Compared with NS2, NS3 puts 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 unsuccessful experiences of NS2. OMNet++ is another important network simulator which has a very powerful graphical interface and modular core design. OMNet++ is also open sourced and widely acknowledged in academia.

OPNET

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

Overview

OPNET ' s software environment is called “Modeler”, 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.

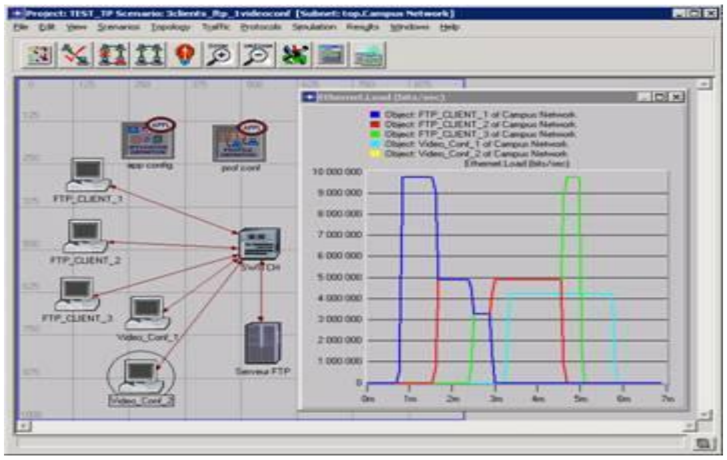


Figure 1. OPNET GUI [OPNET]

Main features

OPNET inherently has three main functions: modeling, simulating, and analysis. For modeling, it provides an 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 whitepaper, 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, on 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 upgraded its ACE Analyst software includes functionality and it is announced to 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" [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 maintains a high acknowledgement in the industry. Moreover, OPNET always keeps an eye on the most recent user's requirements and keeps improving their product which makes 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 the 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

First and foremost, NS2 is an object-oriented, discrete event driven network simulator which was originally developed at University of California-Berkeley. The programming it uses is C++ and OTcl (Tcl script language with Object-oriented extensions developed at MIT). The usage of these two programming languages 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 assemble different components and to change different parameters without a very visual and easy-to-use descriptive language. Moreover, for efficiency reasons, 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 the 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.

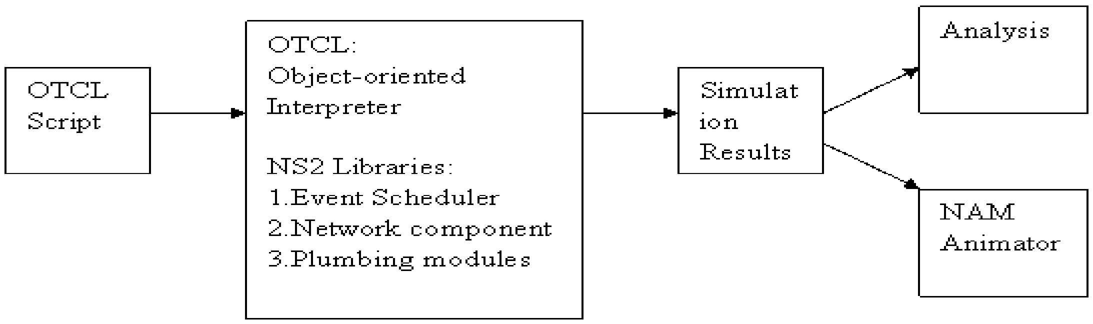


Figure 2. Simplified User's View o f NS2

Another feature of NS2 is the event scheduler. In NS2, the event scheduler keeps track of simulation time and releases 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 the 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 multirate 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.

STUDY OF TCP/UDP USING SIMULATION TOOL

EX.NO: 6

DATE:

AIM: To study about the TCP /UDP.

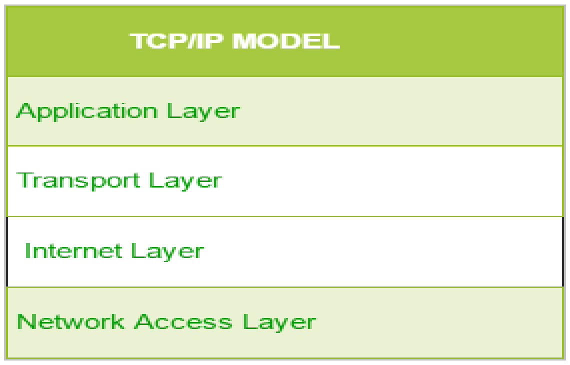
TCP/IP Model

The **OSI Model** we just looked at is just a reference/logical model. It was designed to describe the functions of the communication system by dividing the communication procedure into smaller and simpler components. But when we talk about the TCP/IP model, it was designed and developed by Department of Defense (DoD) in 1960s and is based on standard protocols. It stands for Transmission Control Protocol/Internet Protocol. The **TCP/IP model** is a concise version of the OSI model.

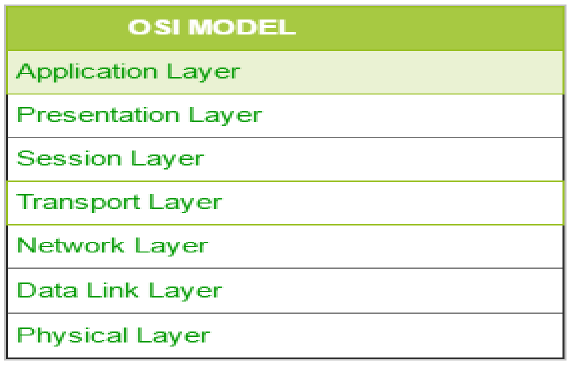
It contains four layers, unlike seven layers in the OSI model. The layers are:

1. **Process/Application Layer**
2. **Host-to-Host/Transport Layer**
3. **Internet Layer**
4. **Network Access/Link Layer**

The diagrammatic comparison of the TCP/IP and OSI model is as follows :



TCP/IP MODEL
Application Layer
Transport Layer
Internet Layer
Network Access Layer



OSI MODEL
Application Layer
Presentation Layer
Session Layer
Transport Layer
Network Layer
Data Link Layer
Physical Layer

Host-to-Host Layer –

This layer is analogous to the transport layer of the OSI model. It is responsible for end-to-end communication and error-free delivery of data. It shields the upper-layer applications from the complexities of data. The two main protocols present in this layer are :

1. **Transmission Control Protocol (TCP) –** It is known to provide reliable and error-free communication between end systems. It performs sequencing and segmentation of data. It also has acknowledgment feature and controls the flow of the data through flow control mechanism. It is a very effective protocol but has a lot of overhead due to such features. Increased overhead leads to increased cost.

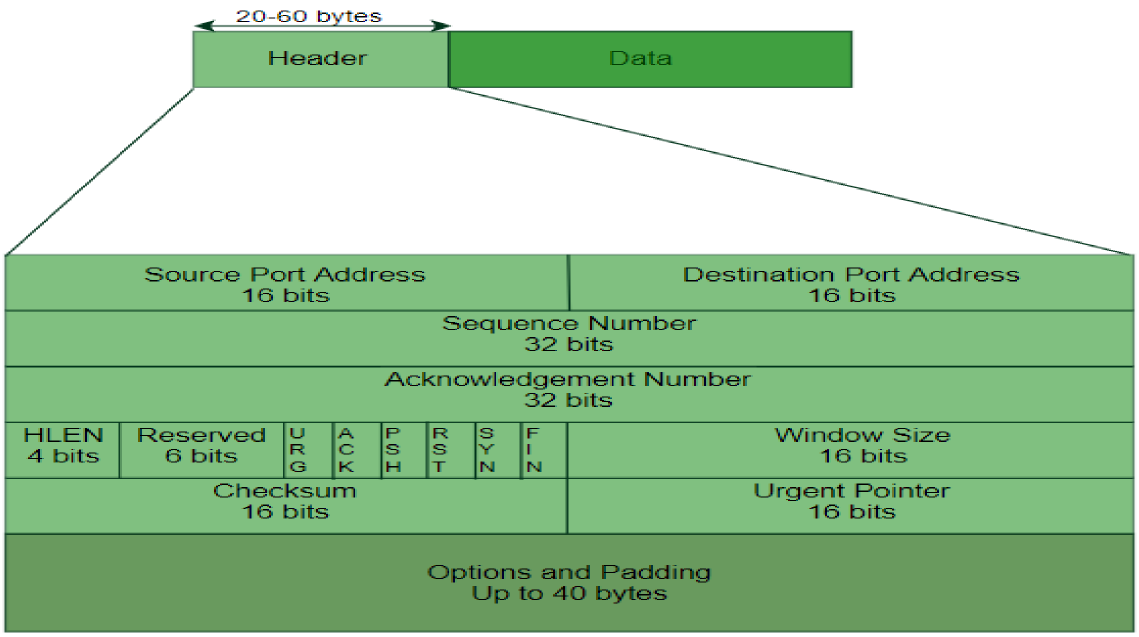
2. **User Datagram Protocol (UDP) –** On the other hand does not provide any such features. It is the go-to protocol if your application does not require reliable transport as it is very cost-effective. Unlike TCP, which is connection-oriented protocol, UDP is connectionless.

TCP Header

The header of a TCP segment can range from 20-60 bytes. 40 bytes are for options. If there are no options, a header is 20 bytes else it can be of upmost 60 bytes. A 16-bit field that holds the port address of the application that is sending the data segment.

TCP Segment structure –

A TCP segment consists of data bytes to be sent and a header that is added to the data by TCP as shown:



The header of a TCP segment can range from 20-60 bytes. 40 bytes are for options. If there are no options, a header is 20 bytes else it can be of upmost 60 bytes. Header fields: **Source Port Address –** A 16-bit field that holds the port address of the application that is sending the data segment.

- **Destination Port Address –** A 16-bit field that holds the port address of the application in the host that is receiving the data segment.
- **Sequence Number –** A 32-bit field that holds the sequence number, i.e, the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end of the segments that are received out of order.

- **Acknowledgement Number –** A 32-bit field that holds the acknowledgement number, i.e, the byte number that the receiver expects to receive next. It is an acknowledgement for the previous bytes being received successfully.

- **Header Length (HLEN) –** This is a 4-bit field that indicates the length of the TCP header by a number of 4-byte words in the header, i.e if the header is 20 bytes(min length of TCP header), then this field will hold 5 (because 5 x 4 = 20) and the maximum length: 60 bytes, then it'll hold the value 15(because 15 x 4 = 60). Hence, the value of this field is always between 5 and 15.
- **Control flags –** These are 6 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. Their function is:
    - URG: Urgent pointer is valid
    - ACK: Acknowledgement number is valid(used in case of cumulative acknowledgement)
    - PSH: Request for push
    - RST: Reset the connection
    - SYN: Synchronize sequence numbers
    - FIN: Terminate the connection
- **Window size –**
  This field tells the window size of the sending TCP in bytes.
- **Checksum** –
  This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP.
- **Urgent pointer –** This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.

TCP Connection Establishment

TCP is a connection-oriented protocol and every connection-oriented protocol needs to establish a connection in order to reserve resources at both the communicating ends.

Connection Establishment –

1. Sender starts the process with the following:

- Sequence number (Seq=521): contains the random initial sequence number generated at the sender side.
- Syn flag (Syn=1): request the receiver to synchronize its sequence number with the above-provided sequence number.

- Maximum segment size (MSS=1460 B): sender tells its maximum segment size, so that receiver sends datagram which won’t require any fragmentation. MSS field is present inside Option field in TCP header.
- Window size (window=14600 B): sender tells about his buffer capacity in which he has to store messages from the receiver.

2. TCP is a full-duplex protocol so both sender and receiver require a window for receiving messages from one another.

- Sequence number (Seq=2000): contains the random initial sequence number generated at the receiver side.
- Syn flag (Syn=1): request the sender to synchronize its sequence number with the above-provided sequence number.
- Maximum segment size (MSS=500 B): sender tells its maximum segment size, so that receiver sends datagram which won’t require any fragmentation. MSS field is present inside Option field in TCP header.
  Since MSSreceiver < MSSsender, both parties agree for minimum MSS i.e., 500 B to avoid fragmentation of packets at both ends.
- Therefore, receiver can send maximum of 14600/500 = 29 packets.
  This is the receiver's sending window size.
- Window size (window=10000 B): receiver tells about his buffer capacity in which he has to store messages from the sender.
- Therefore, sender can send a maximum of 10000/500 = 20 packets.
  This is the sender's sending window size.
- Acknowledgment Number (Ack no.=522): Since sequence number 521 is received by the receiver so, it makes a request for the next sequence number with Ack no.=522 which is the next packet expected by the receiver since Syn flag consumes 1 sequence no.
- ACK flag (ACk=1): tells that the acknowledgment number field contains the next sequence expected by the receiver.

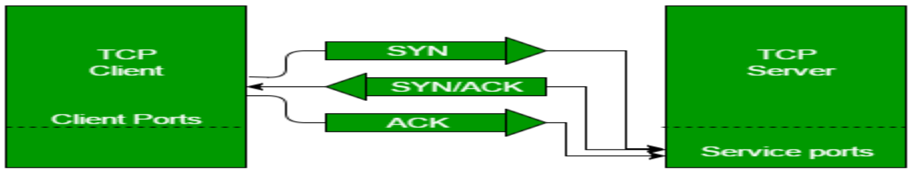
3. Sender makes the final reply for connection establishment in the following way:

- Sequence number (Seq=522): since sequence number = 521 in 1st step and SYN flag consumes one sequence number hence, the next sequence number will be 522.

- Acknowledgment Number (Ack no.=2001): since the sender is acknowledging SYN=1 packet from the receiver with sequence number 2000 so, the next sequence number expected is 2001.
- ACK flag (ACK=1): tells that the acknowledgment number field contains the next sequence expected by the sender.

TCP 3-Way Handshake Process

The process of communication between devices over the internet happens according to the current **TCP/IP** suite model(stripped out version of OSI reference model). The Application layer is a top pile of a stack of TCP/IP models from where network referenced applications like web browsers on the client-side establish a connection with the server. From the application layer, the information is transferred to the transport layer where our topic comes into the picture. The two important protocols of this layer are – TCP, **UDP(User Datagram Protocol)** out of which TCP is prevalent(since it provides reliability for the connection established). However, you can find an application of UDP in querying the DNS server to get the binary equivalent of the Domain Name used for the website.



Traditional TCP

Transmission Control Protocol (TCP) is the transport layer protocol that serves as an interface between client and server. The TCP/IP protocol is used to transfer the data packets between transport layer and network layer. Transport protocol is mainly designed for fixed end systems and fixed, wired networks. In simple terms, the traditional TCP is defined as a wired network while classical TCP uses wireless approach. Mainly TCP is designed for fixed networks and fixed, wired networks.

The main research activities in TCP are as listed below.

1) Congestion Control

2) Slow start

3) Fast re-transmission

1. Congestion control:

During data transmission from sender to receiver, sometimes the data packet may be lost. It is not because of hardware or software problem. Whenever the packet loss is confirmed, the probable reason might be the temporary overload at some point in the transmission path. This temporary overload is otherwise called as Congestion.

Congestion is caused often even when the network is designed perfectly. The transmission speed of receiver may not be equal to the transmission speed of the sender. if the capacity of the sender is more than the capacity of output link, then the packet buffer of a router is filled and the router cannot forward the packets fast enough. The only thing the router can do in this situation is to drop some packets.

The receiver sense the packet loss but does not send message regarding packet loss to the sender. Instead, the receiver starts to send acknowledgement for all the received packets and the sender soon identifies the missing acknowledgement. The sender now notices that a packet is lost and slows down the transmission process. By this, the congestion is reduced. This feature of TCP is one of the reason for its demand even today.

2. Slow start:

The behavior TCP shows after the detection of congestion is called as slow start. The sender always calculates a congestion window for a receiver. At first the sender sends a packet and waits for the acknowledgement. Once the acknowledgement is back it doubles the packet size and sends two packets. After receiving two acknowledgements, one for each packet, the sender again doubles the packet size and this process continues. This is called Exponential growth.

It is dangerous to double the congestion window each time because the steps might become too large. The exponential growth stops at congestion threshold. As it reaches congestion threshold, the increase in transmission rate becomes linear (i.e., the increase is only by 1). Linear increase continues until the sender notices gap between the acknowledgments. In this case, the sender sets the size of congestion window to half of its congestion threshold and the process continues.

3. Fast re-transmission:

In TCP, two things lead to a reduction of the congestion threshold. One of those is sender receiving continuous acknowledgements for the single packet. By this it can convey either of two things. One such thing is that the receiver received all the packets up to the acknowledged one and the other thing is the gap is due to packet loss. Now the sender immediately re-transmit the missing packet before the given time expires. This is called as Fast re-transmission.

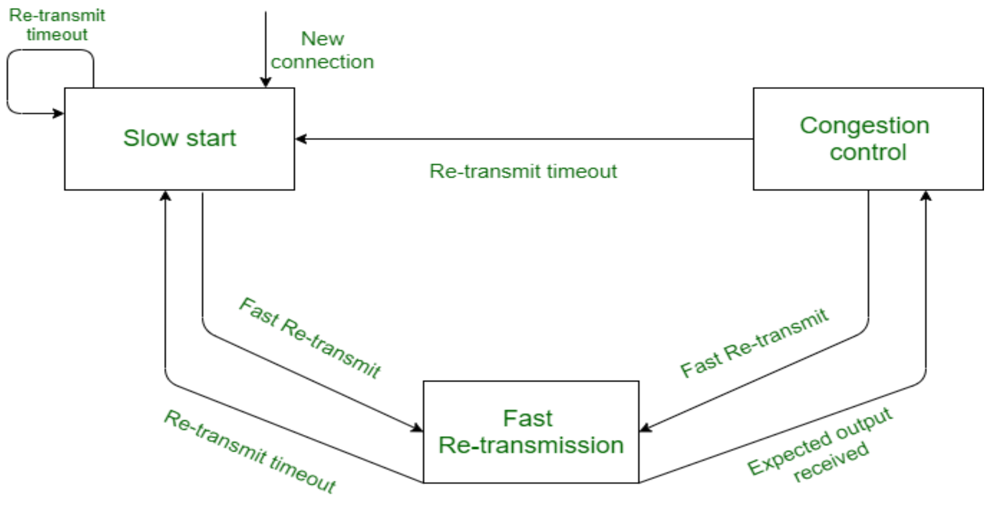


Figure: Traditional TCP

Advantages :

Here, we will discuss the advantages as follows.

1. **The end-to-end TCP semantic is preserved –**

   The packet is not acknowledged by the FA. And if the foreign agent (FA) or base station (BS) fails, the solution reverts to standard TCP.

2. **No Modifications at Fixed Host –**

   The fixed computer TCP does not need any changes. The majority of the changes are made at the foreign agent (FA).

3. **No packet loss during handovers –**

   In the case of a handover, if any data is not passed to the new foreign agent, there will be a time-out at the fixed host and activating retransmission of the packet, via mobile IP, to a new COA.

Disadvantages

Here, we will discuss the disadvantages as follows.

1. **The behavior of the wireless link –** Snooping TCP does not isolate the behavior of the wireless link or I-TCP. Transmission errors can spread to the correspondent nodes (CH).

2. **A mobile node needs additional mechanisms –** The use of NACK between the foreign agent and the mobile node requires the mobile node to have additional mechanisms. For arbitrary mobile nodes, this method is no longer transparent.
3. **Encryption at end-to-end –**If such encryption schemes are used end-to-end between the correspondent node and mobile node, snooping and buffering data can be considered worthless. Snooping TCP may be used if encryption is used above the transport layer (e.g. SSL/TLS).

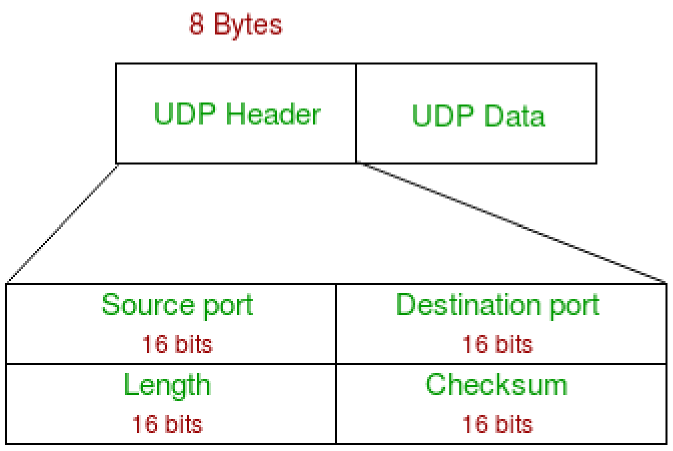
User Datagram Protocol (UDP)

User Datagram Protocol (UDP) is a Transport Layer protocol. UDP is a part of the Internet Protocol suite, referred to as UDP/IP suite. Unlike TCP, it is an **unreliable and connectionless protocol.** So, there is no need to establish a connection prior to data transfer.

Though Transmission Control Protocol (TCP) is the dominant transport layer protocol used with most of the Internet services; provides assured delivery, reliability, and much more but all these services cost us additional overhead and latency. Here, UDP comes into the picture. For real-time services like computer gaming, voice or video communication, live conferences; we need UDP. Since high performance is needed, UDP permits packets to be dropped instead of processing delayed packets. There is no error checking in UDP, so it also saves bandwidth. User Datagram Protocol (UDP) is more efficient in terms of both latency and bandwidth.

UDP Header –

UDP header is an **8-bytes** fixed and simple header, while for TCP it may vary from 20 bytes to 60 bytes. The first 8 Bytes contains all necessary header information and the remaining part consist of data. UDP port number fields are each 16 bits long, therefore the range for port numbers is defined from 0 to 65535; port number 0 is reserved. Port numbers help to distinguish different user requests or processes.



1. **Source Port:** Source Port is a 2 Byte long field used to identify the port number of the source.
2. **Destination Port:** It is a 2 Byte long field, used to identify the port of the destined packet.
3. **Length:** Length is the length of UDP including the header and the data. It is a 16-bits field.
4. **Checksum:** Checksum is 2 Bytes long field. It is the 16-bit one's complement of the one's complement sum of the UDP header, the pseudo-header of information from the IP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

Notes – Unlike TCP, the Checksum calculation is not mandatory in UDP. No Error control or flow control is provided by UDP. Hence UDP depends on IP and ICMP for error reporting.

Applications of UDP:

- Used for simple request-response communication when the size of data is less and hence there is lesser concern about flow and error control.
- It is a suitable protocol for multicasting as UDP supports packet switching.
- UDP is used for some routing update protocols like RIP(Routing Information Protocol).
- Normally used for real-time applications which can not tolerate uneven delays between sections of a received message.
- Following implementations uses UDP as a transport layer protocol:
    - NTP (Network Time Protocol)
    - DNS (Domain Name Service)
    - BOOTP, DHCP.
    - NNP (Network News Protocol)
    - Quote of the day protocol
    - TFTP, RTSP, RIP.

- The application layer can do some of the tasks through UDP-
    - Trace Route
    - Record Route
    - Timestamp
- UDP takes a datagram from Network Layer, attaches its header, and sends it to the user. So, it works fast.
- Actually, UDP is a null protocol if you remove the checksum field.

  90082512. Reduce the requirement of computer resources.

  90082513. When using the Multicast or Broadcast to transfer.

  90082514. The transmission of Real-time packets, mainly in multimedia applications.

SIMULATION OF ERROR CORRECTION CODE – CRC

Ex.No:7

Date :

AIM : Implement a CRC Application

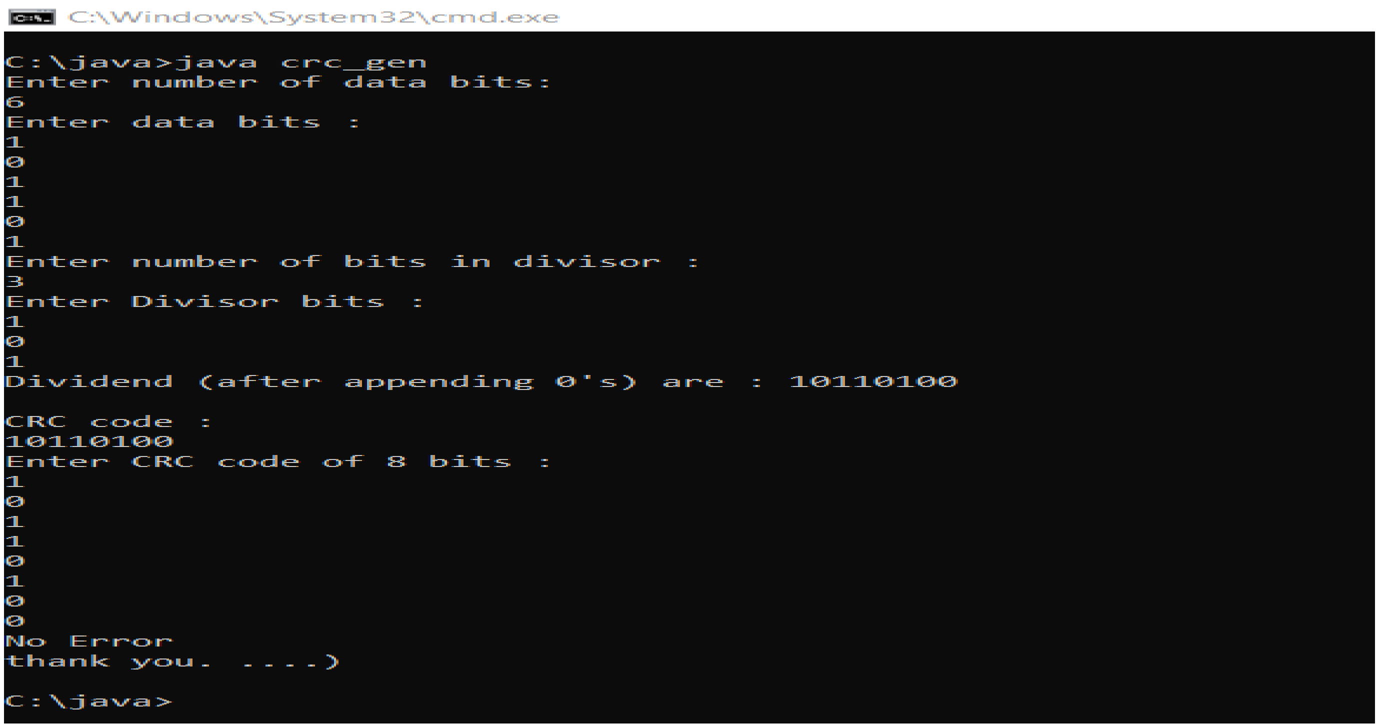
ALGORITHM:
Step 1: Import the necessary java libraries.
Step 2: Write and compile the server and client side programs separately.
Step 3: Execute both the programs side by side on different console windows, with the server program executed first.
Step 4: Verify the output.

PROGRAM:
```
import java.io.*;
public class crc_gen
{
public static void main(String args[]) throws IOException
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
int[] data;
int[] div;
int[] divisor;
int[] rem;
int[] crc;
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits: ");
data_bits=Integer.parseInt(br.readLine());
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)

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



```
C:\Windows\System32\cmd.exe

C:\java>java crc_gen
Enter number of data bits:
6
Enter data bits :
1
0
1
1
0
1
Enter number of bits in divisor :
3
Enter Divisor bits :
1
0
1
Dividend (after appending 0's) are : 10110100

CRC code :
10110100
Enter CRC code of 8 bits :
1
0
1
1
0
1
0
0
No Error
thank you. ....)

C:\java>
```

RESULT: The given program was successfully executed

TRAFFIC ANALYSIS USING WIRESHARK

Ex.No:8

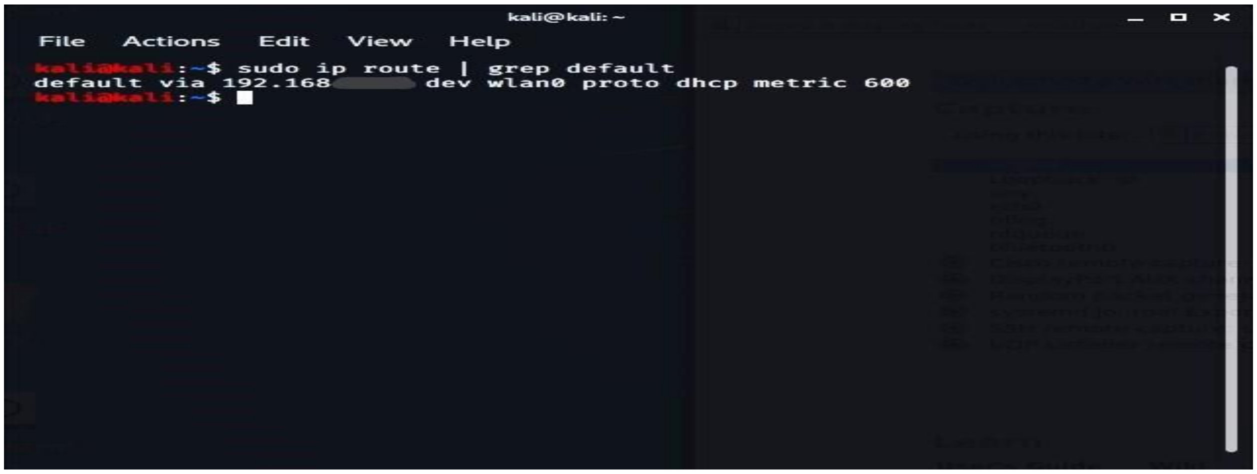
Date :

AIM : Implement a Traffic Analysis Application using WireShark

ARP traffic analysis

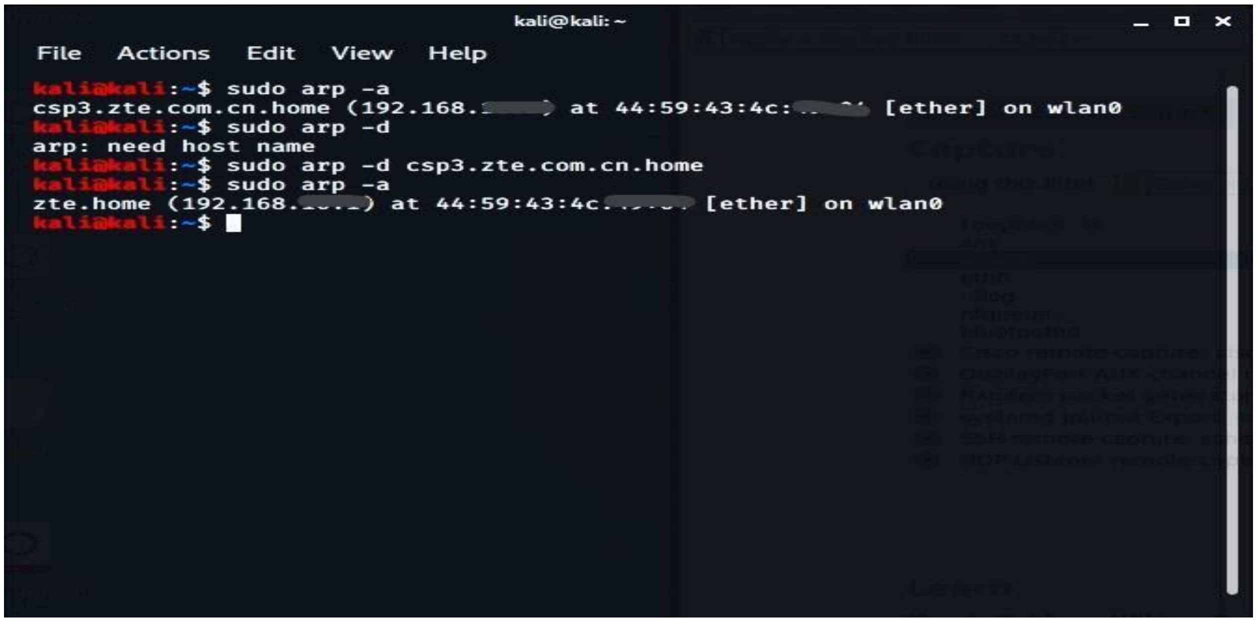
Address resolution protocol (ARP) is generally used to find the MAC address of the target machine. In this demo, let's try capturing and analyzing ARP traffic.

First things first, know the target machine IP. In our case, it's going to be the default gateway address.



```
kali@kali:~$ sudo ip route | grep default
default via 192.168[redacted] dev wlan0 proto dhcp metric 600
kali@kali:~$ 
```

Find existing ARP cache -> Delete the existing one to understand the demo -> Check ARP cache for verification.



```
kali@kali:~$ sudo arp -a
csp3.zte.com.cn.home (192.168.[redacted]) at 44:59:43:4c:[redacted] [ether] on wlan0
kali@kali:~$ sudo arp -d
arp: need host name
kali@kali:~$ sudo arp -d csp3.zte.com.cn.home
kali@kali:~$ sudo arp -a
zte.home (192.168.[redacted]) at 44:59:43:4c[redacted] [ether] on wlan0
kali@kali:~$ 
```

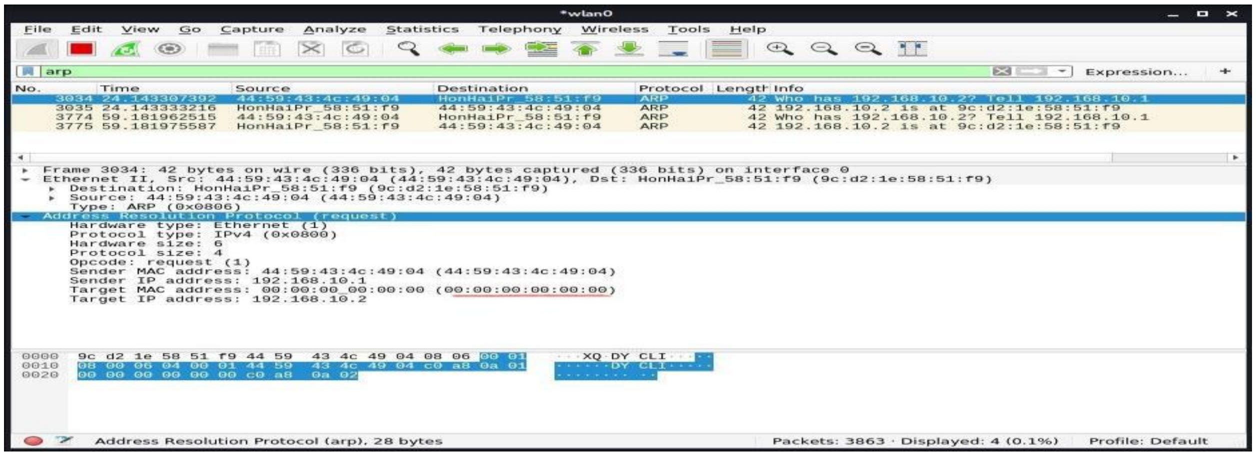
Start Wireshark data capturing, and ping the default gateway address -> Now, let's analyze what happens after removing the ARP entry and pinging a new IP address in the meantime.

Analyze an ARP Request

Using the *'arp'* filter, analyze the captured traffic in Wireshark.

Observe the packet request details from Ethernet and ARP; observe the source and destination IP and sender MAC and IP address.

Monitor the victim's MAC address. Since the destination MAC address is unavailable at the request packet stage, the victim's MAC address is zero, and the destination IP is the local system IP address.


*wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp
Expression...

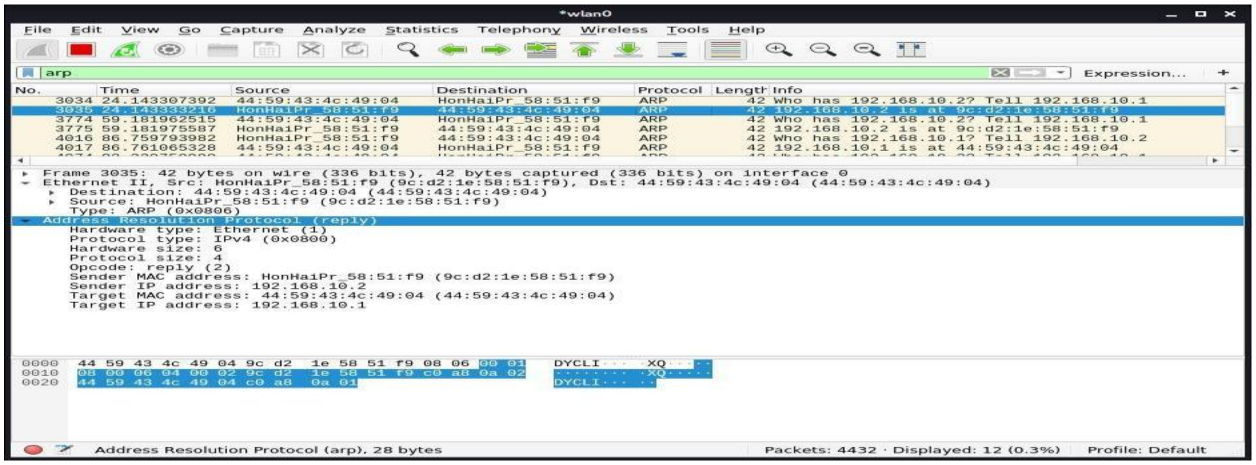
```
No.     Time            Source               Destination          Protocol Length Info
   3034 24.143307392    44:59:43:4c:49:04    HonHaiPr_58:51:f9    ARP        42 Who has 192.168.10.2? Tell 192.168.10.1
   3035 24.143333216    HonHaiPr_58:51:f9    44:59:43:4c:49:04    ARP        42 192.168.10.2 is at 9c:d2:1e:58:51:f9
   3774 59.181962515    44:59:43:4c:49:04    HonHaiPr_58:51:f9    ARP        42 Who has 192.168.10.2? Tell 192.168.10.1
   3775 59.181975587    HonHaiPr_58:51:f9    44:59:43:4c:49:04    ARP        42 192.168.10.2 is at 9c:d2:1e:58:51:f9

▸ Frame 3034: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: 44:59:43:4c:49:04 (44:59:43:4c:49:04), Dst: HonHaiPr_58:51:f9 (9c:d2:1e:58:51:f9)
  ▸ Destination: HonHaiPr_58:51:f9 (9c:d2:1e:58:51:f9)
  ▸ Source: 44:59:43:4c:49:04 (44:59:43:4c:49:04)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: 44:59:43:4c:49:04 (44:59:43:4c:49:04)
    Sender IP address: 192.168.10.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.10.2

0000  9c d2 1e 58 51 f9 44 59  43 4c 49 04 08 06 00 01   ···XQ·DY CLI····
0010  08 00 06 04 00 01 44 59  43 4c 49 04 c0 a8 0a 01   ······DY CLI····
0020  00 00 00 00 00 00 c0 a8  0a 02                     ········ ··

Address Resolution Protocol (arp), 28 bytes        Packets: 3863 · Displayed: 4 (0.1%)        Profile: Default
```

Analyze an ARP Response

Observe the packet replay details from Ethernet and ARP; observe the change in source and destination IP and MAC addresses.

The destination and source MAC address are switched in the response packet.

Everything is similar as before, except the target MAC address, which was all zeroes before. Now, that has turned into your MAC address.


*wlan0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp
Expression...

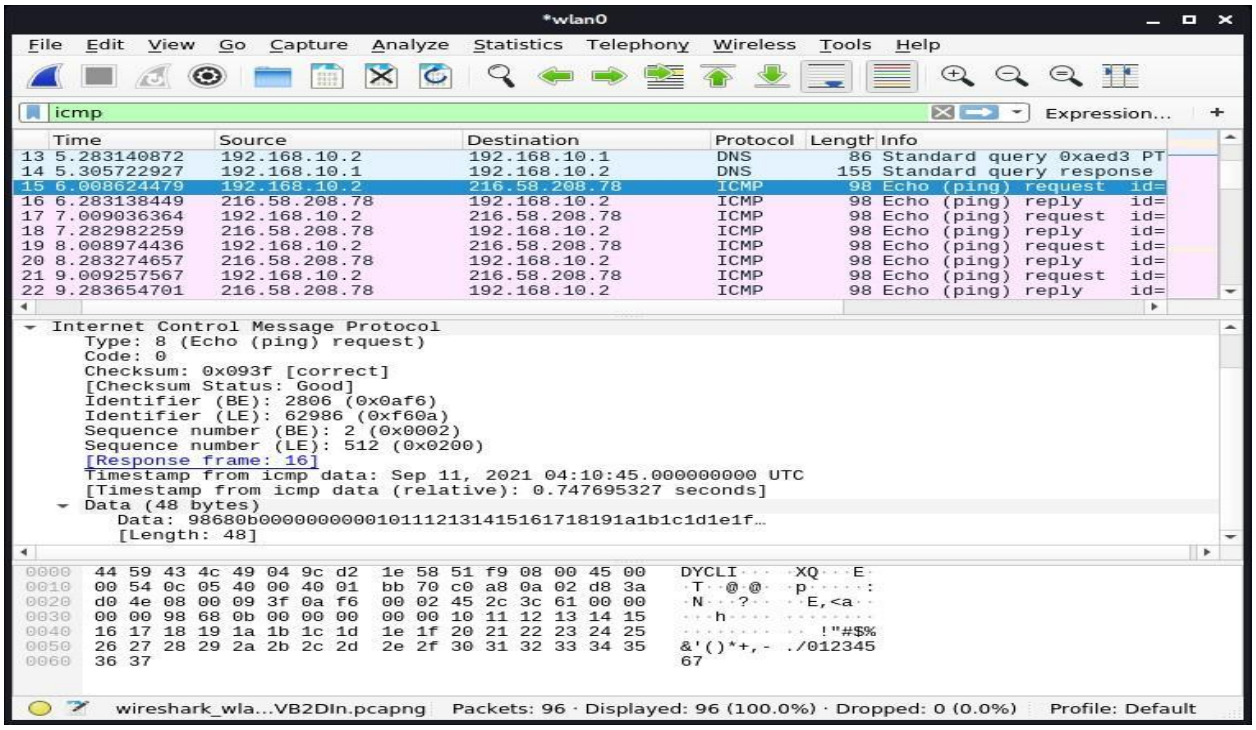
```
No.     Time            Source               Destination          Protocol Length Info
   3034 24.143307392    44:59:43:4c:49:04    HonHaiPr_58:51:f9    ARP        42 Who has 192.168.10.2? Tell 192.168.10.1
   3035 24.143333216    HonHaiPr_58:51:f9    44:59:43:4c:49:04    ARP        42 192.168.10.2 is at 9c:d2:1e:58:51:f9
   3774 59.181962515    44:59:43:4c:49:04    HonHaiPr_58:51:f9    ARP        42 Who has 192.168.10.2? Tell 192.168.10.1
   3775 59.181975587    HonHaiPr_58:51:f9    44:59:43:4c:49:04    ARP        42 192.168.10.2 is at 9c:d2:1e:58:51:f9
   4016 86.759793982    HonHaiPr_58:51:f9    44:59:43:4c:49:04    ARP        42 Who has 192.168.10.1? Tell 192.168.10.2
   4017 86.761065328    44:59:43:4c:49:04    HonHaiPr_58:51:f9    ARP        42 192.168.10.1 is at 44:59:43:4c:49:04

▸ Frame 3035: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: HonHaiPr_58:51:f9 (9c:d2:1e:58:51:f9), Dst: 44:59:43:4c:49:04 (44:59:43:4c:49:04)
  ▸ Destination: 44:59:43:4c:49:04 (44:59:43:4c:49:04)
  ▸ Source: HonHaiPr_58:51:f9 (9c:d2:1e:58:51:f9)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: HonHaiPr_58:51:f9 (9c:d2:1e:58:51:f9)
    Sender IP address: 192.168.10.2
    Target MAC address: 44:59:43:4c:49:04 (44:59:43:4c:49:04)
    Target IP address: 192.168.10.1

0000  44 59 43 4c 49 04 9c d2  1e 58 51 f9 08 06 00 01   DYCLI··· ·XQ····
0010  08 00 06 04 00 02 9c d2  1e 58 51 f9 c0 a8 0a 02   ········ ·XQ····
0020  44 59 43 4c 49 04 c0 a8  0a 01                     DYCLI··· ··

Address Resolution Protocol (arp), 28 bytes        Packets: 4432 · Displayed: 12 (0.3%)        Profile: Default
```

ICMP traffic analysis

ICMP is used for error alerting and monitoring to verify whether data arrives in a timely basis at its desired destination.

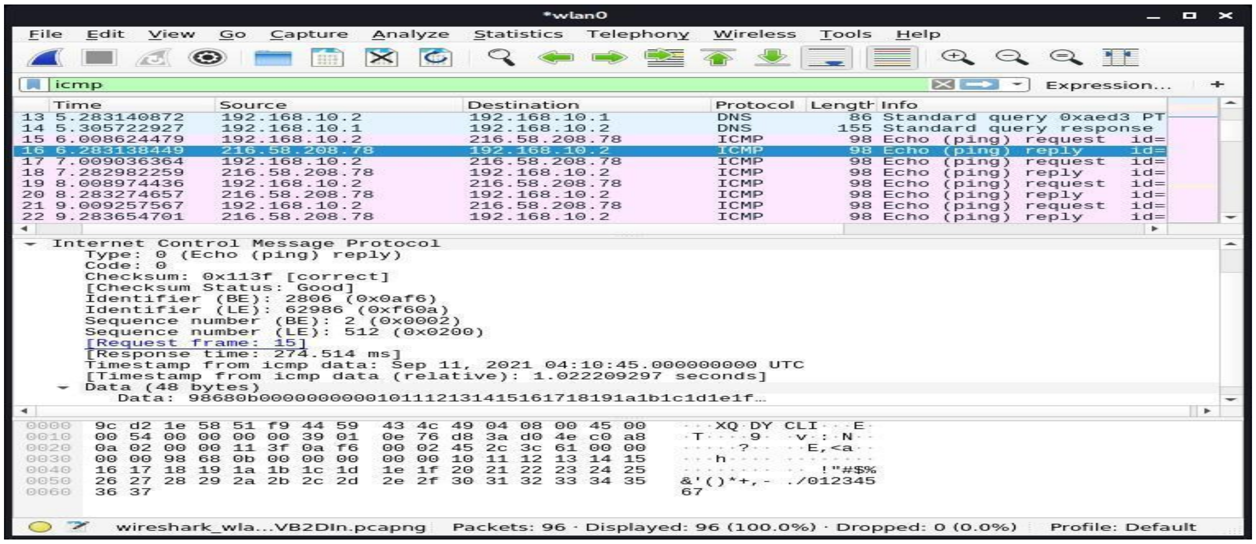
To capture ICMP traffic, ping Google.com. Use the *'ICMP'* filter to see ICMP traffic. Click the ICMP echo-request packet from the Wireshark capture window and start observing the information.

In the ***request packet,*** the source IP is your (requestor) IP address. Whereas the destination IP is that of Google. You can also analyze the ICMP details like Checksum, Identifier Number, Sequence Number, etc.



```
icmp
    Time         Source          Destination     Protocol Length Info
 13 5.283140872  192.168.10.2    192.168.10.1    DNS          86 Standard query 0xaed3 PT
 14 5.305722927  192.168.10.1    192.168.10.2    DNS         155 Standard query response
 15 6.008624479  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 16 6.283138449  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=
 17 7.009036364  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 18 7.282982259  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=
 19 8.008974436  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 20 8.283274657  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=
 21 9.009257567  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 22 9.283654701  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=

Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x093f [correct]
    [Checksum Status: Good]
    Identifier (BE): 2806 (0x0af6)
    Identifier (LE): 62986 (0xf60a)
    Sequence number (BE): 2 (0x0002)
    Sequence number (LE): 512 (0x0200)
    [Response frame: 16]
    Timestamp from icmp data: Sep 11, 2021 04:10:45.000000000 UTC
    [Timestamp from icmp data (relative): 0.747695327 seconds]
  Data (48 bytes)
      Data: 98680b0000000000101112131415161718191a1b1c1d1e1f…
      [Length: 48]

0000  44 59 43 4c 49 04 9c d2  1e 58 51 f9 08 00 45 00   DYCLI···  ·XQ···E·
0010  00 54 0c 05 40 00 40 01  bb 70 c0 a8 0a 02 d8 3a   ·T··@·@·  ·p·····:
0020  d0 4e 08 00 09 3f 0a f6  00 02 45 2c 3c 61 00 00   ·N···?··  ··E,<a··
0030  00 00 98 68 0b 00 00 00  00 00 10 11 12 13 14 15   ···h····  ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········  ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,-  ./012345
0060  36 37                                              67

wireshark_wla...VB2DIn.pcapng   Packets: 96 · Displayed: 96 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

In the ***response packet***, observe the swapping of IPs between source and destination. You can also compare both request and response details, as they are similar.



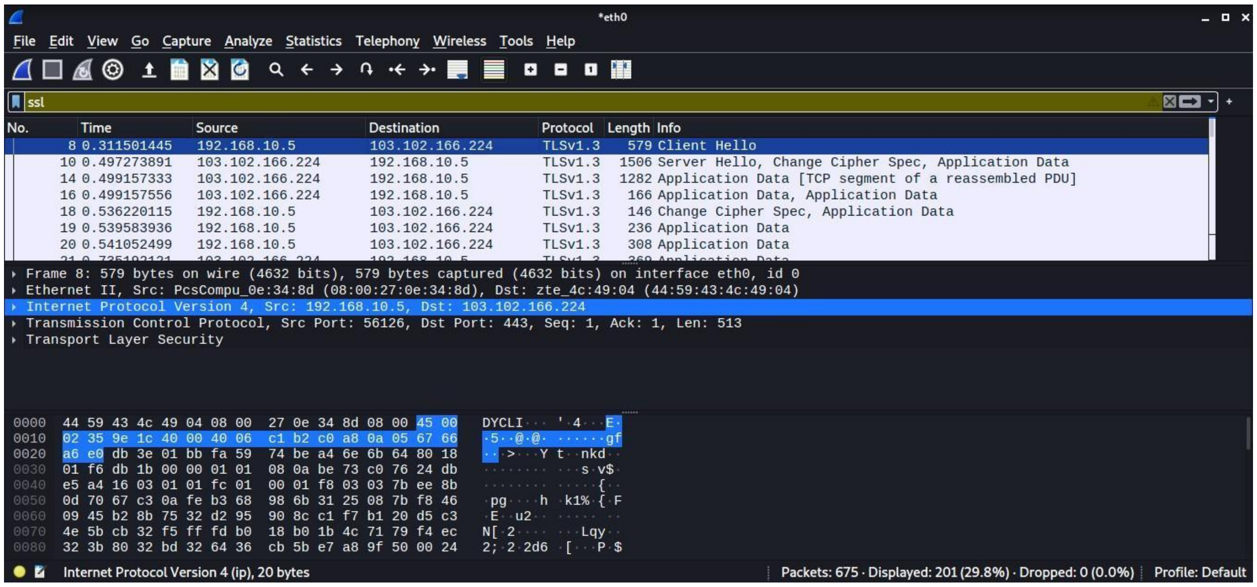
```
icmp
    Time         Source          Destination     Protocol Length Info
 13 5.283140872  192.168.10.2    192.168.10.1    DNS          86 Standard query 0xaed3 PT
 14 5.305722927  192.168.10.1    192.168.10.2    DNS         155 Standard query response
 15 6.008624479  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 16 6.283138449  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=
 17 7.009036364  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 18 7.282982259  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=
 19 8.008974436  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 20 8.283274657  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=
 21 9.009257567  192.168.10.2    216.58.208.78   ICMP         98 Echo (ping) request  id=
 22 9.283654701  216.58.208.78   192.168.10.2    ICMP         98 Echo (ping) reply    id=

Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0x113f [correct]
    [Checksum Status: Good]
    Identifier (BE): 2806 (0x0af6)
    Identifier (LE): 62986 (0xf60a)
    Sequence number (BE): 2 (0x0002)
    Sequence number (LE): 512 (0x0200)
    [Request frame: 15]
    [Response time: 274.514 ms]
    Timestamp from icmp data: Sep 11, 2021 04:10:45.000000000 UTC
    [Timestamp from icmp data (relative): 1.022209297 seconds]
  Data (48 bytes)
      Data: 98680b0000000000101112131415161718191a1b1c1d1e1f…

0000  9c d2 1e 58 51 f9 44 59  43 4c 49 04 08 00 45 00   ···XQ·DY  CLI···E·
0010  00 54 00 00 00 00 39 01  0e 76 d8 3a d0 4e c0 a8   ·T····9·  ·v·:·N··
0020  0a 02 00 00 11 3f 0a f6  00 02 45 2c 3c 61 00 00   ·····?··  ··E,<a··
0030  00 00 98 68 0b 00 00 00  00 00 10 11 12 13 14 15   ···h····  ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········  ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,-  ./012345
0060  36 37                                              67

wireshark_wla...VB2DIn.pcapng   Packets: 96 · Displayed: 96 (100.0%) · Dropped: 0 (0.0%)   Profile: Default
```

HTTPS traffic analysis

The Hypertext Transfer Application Layer Protocol (HTTP) utilizes the internet to establish protocols whenever the HTTP client/server transmits/receives HTTP requests.

Start a Wireshark capture -> Open a web browser -> Navigate to any HTTPS-based website -> Stop the Wireshark capture.

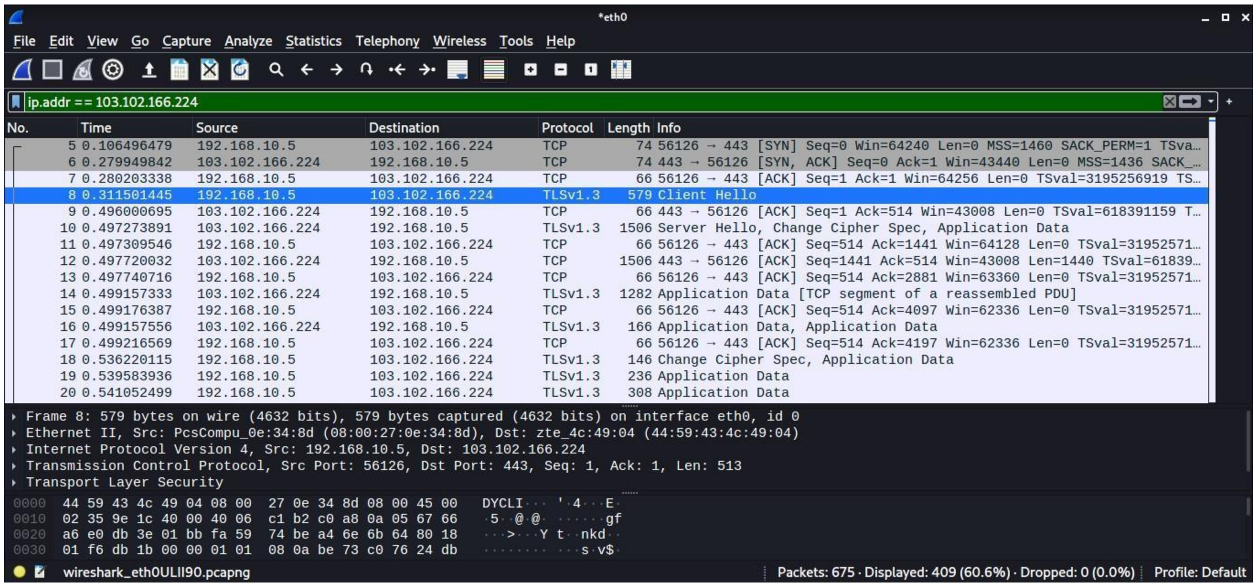
Input ' ssl' in the filter box to monitor only HTTPS traffic -> Observe the first TLS packet -> The destination IP would be the target IP (server).


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ssl

No.  Time         Source          Destination     Protocol Length Info
   8 0.311501445  192.168.10.5    103.102.166.224 TLSv1.3   579 Client Hello
  10 0.497273891  103.102.166.224 192.168.10.5    TLSv1.3  1506 Server Hello, Change Cipher Spec, Application Data
  14 0.499157333  103.102.166.224 192.168.10.5    TLSv1.3  1282 Application Data [TCP segment of a reassembled PDU]
  16 0.499157556  103.102.166.224 192.168.10.5    TLSv1.3   166 Application Data, Application Data
  18 0.536220115  192.168.10.5    103.102.166.224 TLSv1.3   146 Change Cipher Spec, Application Data
  19 0.539583936  192.168.10.5    103.102.166.224 TLSv1.3   236 Application Data
  20 0.541052499  192.168.10.5    103.102.166.224 TLSv1.3   308 Application Data

› Frame 8: 579 bytes on wire (4632 bits), 579 bytes captured (4632 bits) on interface eth0, id 0
› Ethernet II, Src: PcsCompu_0e:34:8d (08:00:27:0e:34:8d), Dst: zte_4c:49:04 (44:59:43:4c:49:04)
› Internet Protocol Version 4, Src: 192.168.10.5, Dst: 103.102.166.224
› Transmission Control Protocol, Src Port: 56126, Dst Port: 443, Seq: 1, Ack: 1, Len: 513
› Transport Layer Security

0000  44 59 43 4c 49 04 08 00  27 0e 34 8d 08 00 45 00   DYCLI··· '·4···E·
0010  02 35 9e 1c 40 00 40 06  c1 b2 c0 a8 0a 05 67 66   ·5··@·@· ······gf
0020  a6 e0 db 3e 01 bb fa 59  74 be a4 6e 6b 64 80 18   ···>···Y t··nkd··
0030  01 f6 db 1b 00 00 01 01  08 0a be 73 c0 76 24 db   ········ ···s·v$·
0040  e5 a4 16 03 01 01 fc 01  00 01 f8 03 03 7b ee 8b   ········ ·····{··
0050  0d 70 67 c3 0a fe b3 68  98 6b 31 25 08 7b f8 46   ·pg····h ·k1%·{·F
0060  09 45 b2 8b 75 32 d2 95  90 8c c1 f7 b1 20 d5 c3   ·E··u2·· ······ ··
0070  4e 5b cb 32 f5 ff fd b0  18 b0 1b 4c 71 79 f4 ec   N[·2···· ···Lqy··
0080  32 3b 80 32 bd 32 64 36  cb 5b e7 a8 9f 50 00 24   2;·2·2d6 ·[···P·$

Internet Protocol Version 4 (ip), 20 bytes    Packets: 675 · Displayed: 201 (29.8%) · Dropped: 0 (0.0%)    Profile: Default
```

To see more traffic of the target IP (destination IP), input the following filter

ip.addr ==

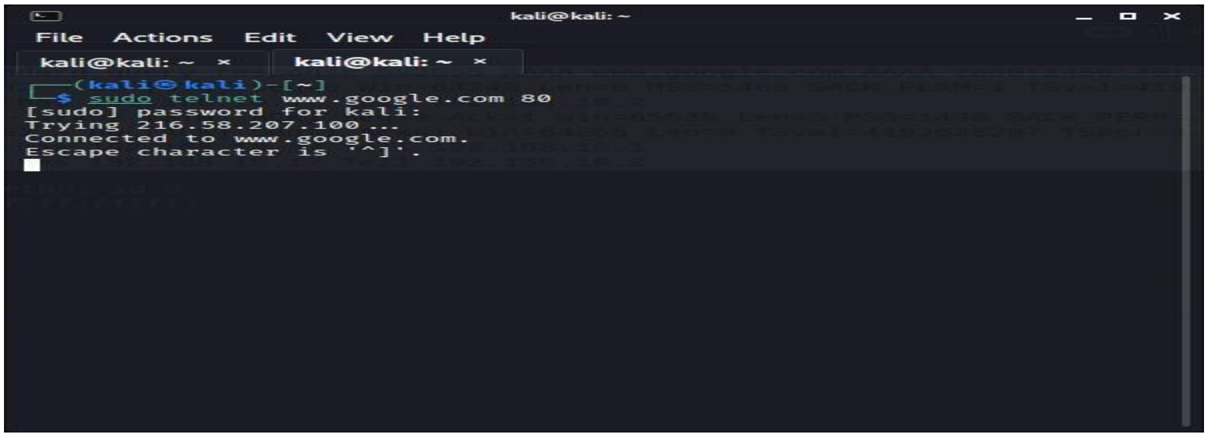

*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

```
ip.addr == 103.102.166.224

No.  Time         Source          Destination     Protocol Length Info
   5 0.106496479  192.168.10.5    103.102.166.224 TCP        74 56126 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSva…
   6 0.279949842  103.102.166.224 192.168.10.5    TCP        74 443 → 56126 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1436 SACK_…
   7 0.280203338  192.168.10.5    103.102.166.224 TCP        66 56126 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3195256919 TS…
   8 0.311501445  192.168.10.5    103.102.166.224 TLSv1.3   579 Client Hello
   9 0.496000695  103.102.166.224 192.168.10.5    TCP        66 443 → 56126 [ACK] Seq=1 Ack=514 Win=43008 Len=0 TSval=618391159 T…
  10 0.497273891  103.102.166.224 192.168.10.5    TLSv1.3  1506 Server Hello, Change Cipher Spec, Application Data
  11 0.497309546  192.168.10.5    103.102.166.224 TCP        66 56126 → 443 [ACK] Seq=514 Ack=1441 Win=64128 Len=0 TSval=31952571…
  12 0.497720032  103.102.166.224 192.168.10.5    TCP      1506 443 → 56126 [ACK] Seq=1441 Ack=514 Win=43008 Len=1440 TSval=61839…
  13 0.497740716  192.168.10.5    103.102.166.224 TCP        66 56126 → 443 [ACK] Seq=514 Ack=2881 Win=63360 Len=0 TSval=31952571…
  14 0.499157333  103.102.166.224 192.168.10.5    TLSv1.3  1282 Application Data [TCP segment of a reassembled PDU]
  15 0.499176387  192.168.10.5    103.102.166.224 TCP        66 56126 → 443 [ACK] Seq=514 Ack=4097 Win=62336 Len=0 TSval=31952571…
  16 0.499157556  103.102.166.224 192.168.10.5    TLSv1.3   166 Application Data, Application Data
  17 0.499216569  192.168.10.5    103.102.166.224 TCP        66 56126 → 443 [ACK] Seq=514 Ack=4197 Win=62336 Len=0 TSval=31952571…
  18 0.536220115  192.168.10.5    103.102.166.224 TLSv1.3   146 Change Cipher Spec, Application Data
  19 0.539583936  192.168.10.5    103.102.166.224 TLSv1.3   236 Application Data
  20 0.541052499  192.168.10.5    103.102.166.224 TLSv1.3   308 Application Data

› Frame 8: 579 bytes on wire (4632 bits), 579 bytes captured (4632 bits) on interface eth0, id 0
› Ethernet II, Src: PcsCompu_0e:34:8d (08:00:27:0e:34:8d), Dst: zte_4c:49:04 (44:59:43:4c:49:04)
› Internet Protocol Version 4, Src: 192.168.10.5, Dst: 103.102.166.224
› Transmission Control Protocol, Src Port: 56126, Dst Port: 443, Seq: 1, Ack: 1, Len: 513
› Transport Layer Security

0000  44 59 43 4c 49 04 08 00  27 0e 34 8d 08 00 45 00   DYCLI··· '·4···E·
0010  02 35 9e 1c 40 00 40 06  c1 b2 c0 a8 0a 05 67 66   ·5··@·@· ······gf
0020  a6 e0 db 3e 01 bb fa 59  74 be a4 6e 6b 64 80 18   ···>···Y t··nkd··
0030  01 f6 db 1b 00 00 01 01  08 0a be 73 c0 76 24 db   ········ ···s·v$·

wireshark_eth0ULII90.pcapng    Packets: 675 · Displayed: 409 (60.6%) · Dropped: 0 (0.0%)    Profile: Default
```

TCP traffic analysis

A standard port scan takes advantage of the TCP three-way handshake. The attacker sends the SYN packet to the target port. The port is considered open when he gets SYN+ACK as a response, whereas the arrival of RST shows the port is closed. After receiving SYN+ACK, the hacker would send an ACK packet to establish a TCP connection.

Let's analyze a TCP network traffic using telnet on Google port 80. Capture the Wireshark traffic while entering the telnet command.



```
┌──(kali㉿kali)-[~]
└─$ sudo telnet www.google.com 80
[sudo] password for kali:
Trying 216.58.207.100...
Connected to www.google.com.
Escape character is '^]'.
```

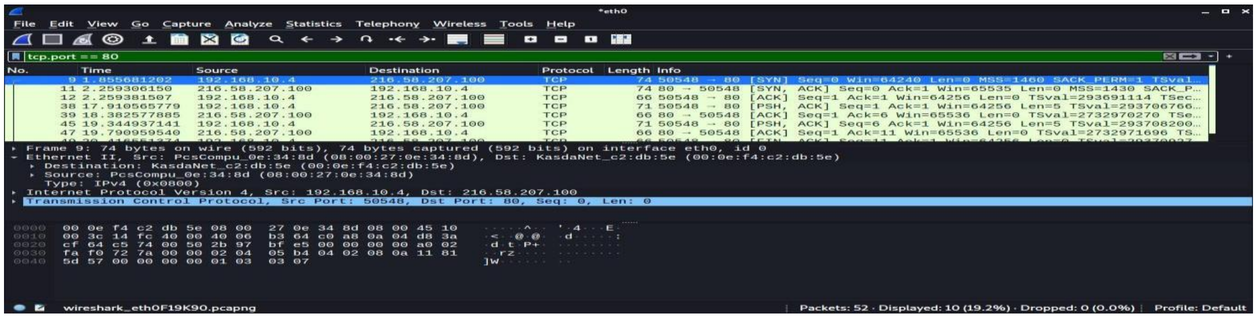
Analyze TCP SYN traffic

Input *'tcp.port == 80'* to see only TCP traffic connected to the webserver connection.

Observe the TCP [SYN] packet. Expand Ethernet and observe the destination address that is the default gateway address; whereas, the source is your own MAC address.

To check the IP details, observe Internet Protocol Version 4; in our case, the destination IP is Googles' web server IP, and the source IP is the local IP address.

To view TCP details, observe Transmission Control Protocol, like port numbers. Monitor the flag values. SYN, which is enabled, shows the initial section of the TCP three-way handshake.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

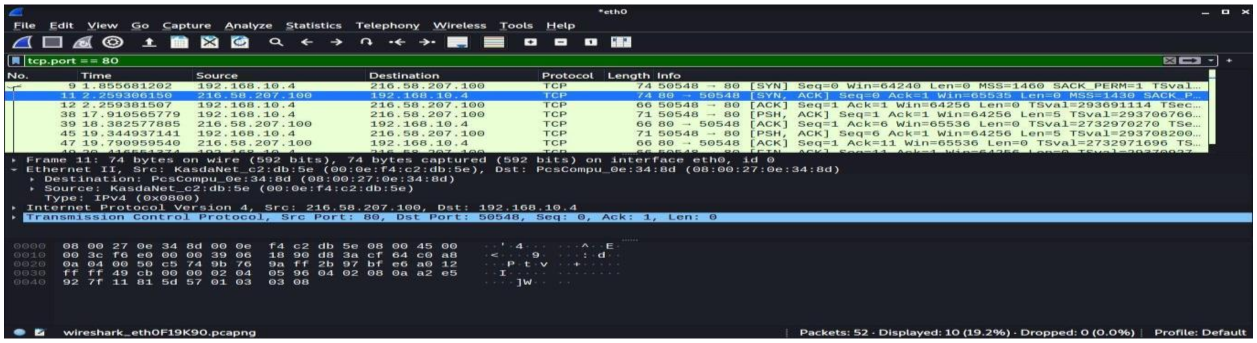
```
tcp.port == 80
No.  Time            Source           Destination      Protocol Length Info
   9 1.855681202     192.168.10.4     216.58.207.100   TCP      74 50548 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval…
  11 2.259306150     216.58.207.100   192.168.10.4     TCP      74 80 → 50548 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1430 SACK_P…
  12 2.259381507     192.168.10.4     216.58.207.100   TCP      66 50548 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=293691114 TSec…
  38 17.910565779    192.168.10.4     216.58.207.100   TCP      71 50548 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=5 TSval=293706766…
  39 18.382577885    216.58.207.100   192.168.10.4     TCP      66 80 → 50548 [ACK] Seq=1 Ack=6 Win=65536 Len=0 TSval=2732970270 TSe…
  45 19.344937141    192.168.10.4     216.58.207.100   TCP      71 50548 → 80 [PSH, ACK] Seq=6 Ack=1 Win=64256 Len=5 TSval=293708200…
  47 19.790959540    216.58.207.100   192.168.10.4     TCP      66 80 → 50548 [ACK] Seq=1 Ack=11 Win=65536 Len=0 TSval=2732971696 TS…
▸ Frame 9: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_0e:34:8d (08:00:27:0e:34:8d), Dst: KasdaNet_c2:db:5e (00:0e:f4:c2:db:5e)
  ▸ Destination: KasdaNet_c2:db:5e (00:0e:f4:c2:db:5e)
  ▸ Source: PcsCompu_0e:34:8d (08:00:27:0e:34:8d)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.10.4, Dst: 216.58.207.100
▸ Transmission Control Protocol, Src Port: 50548, Dst Port: 80, Seq: 0, Len: 0

0000  00 0e f4 c2 db 5e 08 00  27 0e 34 8d 08 00 45 10   .....^.. '.4...E.
0010  00 3c 14 fc 40 00 40 06  b3 64 c0 a8 0a 04 d8 3a   .<..@.@. .d.....:
0020  cf 64 c5 74 00 50 2b 97  bf e5 00 00 00 00 a0 02   .d.t.P+. ........
0030  fa f0 72 7a 00 00 02 04  05 b4 04 02 08 0a 11 81   ..rz.... ........
0040  5d 57 00 00 00 00 01 03  03 07                     ]W...... ..

wireshark_eth0F19K90.pcapng    Packets: 52 · Displayed: 10 (19.2%) · Dropped: 0 (0.0%)    Profile: Default
```

Analyze TCP SYN, ACK traffic

Take a look at the TCP [SYN, ACK] packet. Expand Ethernet and observe the destination address now would be your own MAC address; whereas the source is the default gateway address.

Monitor the acknowledgement code. It's worth noting that the number is one relative ACK number. The real acknowledgement value is one higher than the previous segment's identifier.

Monitor the flag values. [SYN, ACK], which is enabled, shows the second section of the TCP three-way handshake.

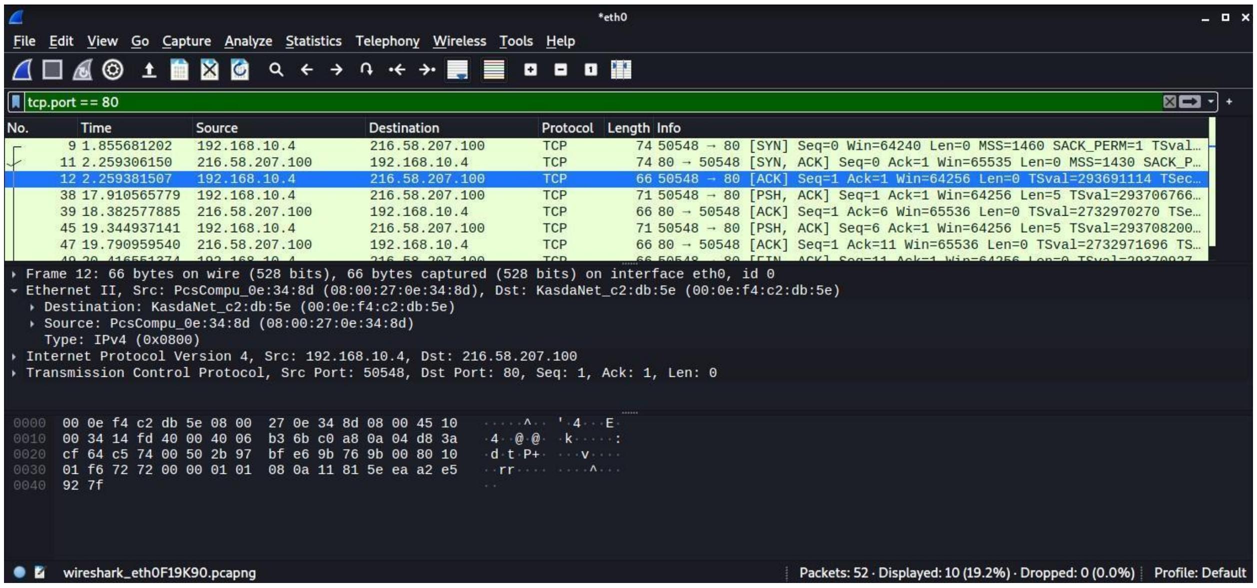

*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 80

```
No.  Time          Source          Destination     Protocol Length Info
   9 1.855681202   192.168.10.4    216.58.207.100  TCP      74 50548 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval…
  11 2.259306150   216.58.207.100  192.168.10.4    TCP      74 80 → 50548 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1430 SACK_P…
  12 2.259381507   192.168.10.4    216.58.207.100  TCP      66 50548 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=293691114 TSec…
  38 17.910565779  192.168.10.4    216.58.207.100  TCP      71 50548 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=5 TSval=293706766…
  39 18.382577885  216.58.207.100  192.168.10.4    TCP      66 80 → 50548 [ACK] Seq=1 Ack=6 Win=65536 Len=0 TSval=2732970270 TSe…
  45 19.344937141  192.168.10.4    216.58.207.100  TCP      71 50548 → 80 [PSH, ACK] Seq=6 Ack=1 Win=64256 Len=5 TSval=293708200…
  47 19.790959540  216.58.207.100  192.168.10.4    TCP      66 80 → 50548 [ACK] Seq=1 Ack=11 Win=65536 Len=0 TSval=2732971696 TS…

▸ Frame 11: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface eth0, id 0
▾ Ethernet II, Src: KasdaNet_c2:db:5e (00:0e:f4:c2:db:5e), Dst: PcsCompu_0e:34:8d (08:00:27:0e:34:8d)
  ▸ Destination: PcsCompu_0e:34:8d (08:00:27:0e:34:8d)
  ▸ Source: KasdaNet_c2:db:5e (00:0e:f4:c2:db:5e)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 216.58.207.100, Dst: 192.168.10.4
▸ Transmission Control Protocol, Src Port: 80, Dst Port: 50548, Seq: 0, Ack: 1, Len: 0

0000  08 00 27 0e 34 8d 00 0e  f4 c2 db 5e 08 00 45 00   ··'·4··· ···^··E·
0010  00 3c f6 e0 00 00 39 06  18 90 d8 3a cf 64 c0 a8   ·<····9· ···:·d··
0020  0a 04 00 50 c5 74 9b 76  9a ff 2b 97 bf e6 a0 12   ···P·t·v ··+·····
0030  ff ff 49 cb 00 00 02 04  05 96 04 02 08 0a a2 e5   ··I····· ········
0040  92 7f 11 81 5d 57 01 03  03 08                     ····]W·· ··

wireshark_eth0F19K90.pcapng    Packets: 52 · Displayed: 10 (19.2%) · Dropped: 0 (0.0%)    Profile: Default
```

Analyze TCP ACK traffic

Now consider the TCP [ACK] packet. Expand Ethernet and observe the destination address that is the default gateway address; whereas the source is your own MAC address. To view TCP details like port numbers, expand Transmission Control Protocol.

The enable ACK flag signals that the TCP three-way handshake has reached the last phase. The client and server have started a TCP session.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tcp.port == 80

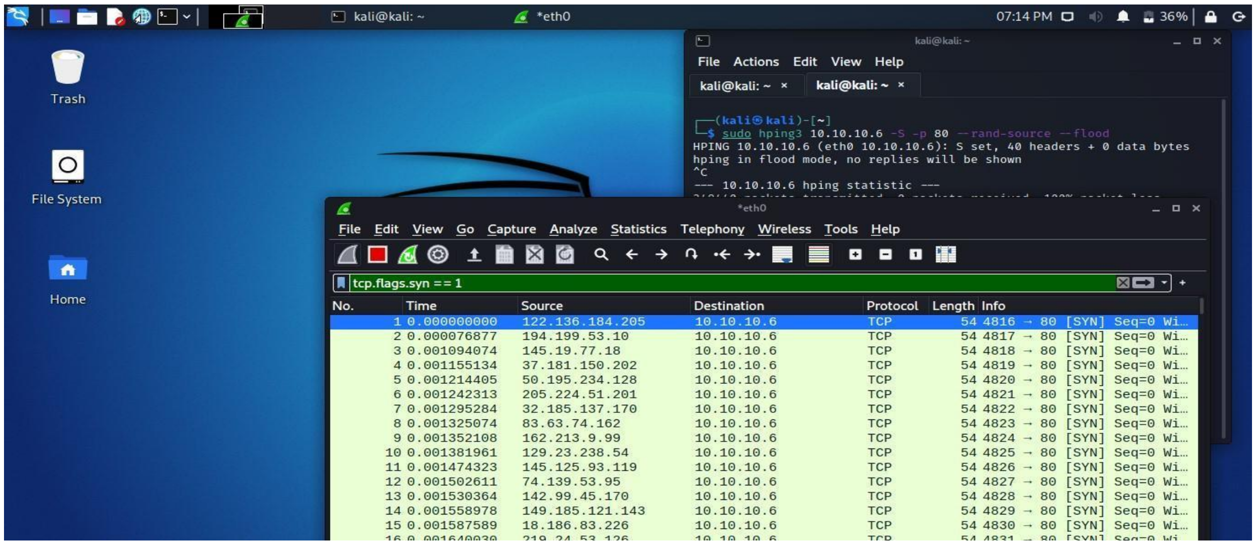
```
No.  Time          Source          Destination     Protocol Length Info
   9 1.855681202   192.168.10.4    216.58.207.100  TCP      74 50548 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval…
  11 2.259306150   216.58.207.100  192.168.10.4    TCP      74 80 → 50548 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1430 SACK_P…
  12 2.259381507   192.168.10.4    216.58.207.100  TCP      66 50548 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=293691114 TSec…
  38 17.910565779  192.168.10.4    216.58.207.100  TCP      71 50548 → 80 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=5 TSval=293706766…
  39 18.382577885  216.58.207.100  192.168.10.4    TCP      66 80 → 50548 [ACK] Seq=1 Ack=6 Win=65536 Len=0 TSval=2732970270 TSe…
  45 19.344937141  192.168.10.4    216.58.207.100  TCP      71 50548 → 80 [PSH, ACK] Seq=6 Ack=1 Win=64256 Len=5 TSval=293708200…
  47 19.790959540  216.58.207.100  192.168.10.4    TCP      66 80 → 50548 [ACK] Seq=1 Ack=11 Win=65536 Len=0 TSval=2732971696 TS…

▸ Frame 12: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0, id 0
▾ Ethernet II, Src: PcsCompu_0e:34:8d (08:00:27:0e:34:8d), Dst: KasdaNet_c2:db:5e (00:0e:f4:c2:db:5e)
  ▸ Destination: KasdaNet_c2:db:5e (00:0e:f4:c2:db:5e)
  ▸ Source: PcsCompu_0e:34:8d (08:00:27:0e:34:8d)
    Type: IPv4 (0x0800)
▸ Internet Protocol Version 4, Src: 192.168.10.4, Dst: 216.58.207.100
▸ Transmission Control Protocol, Src Port: 50548, Dst Port: 80, Seq: 1, Ack: 1, Len: 0

0000  00 0e f4 c2 db 5e 08 00  27 0e 34 8d 08 00 45 10   ·····^·· '·4···E·
0010  00 34 14 fd 40 00 40 06  b3 6b c0 a8 0a 04 d8 3a   ·4··@·@· ·k·····:
0020  cf 64 c5 74 00 50 2b 97  bf e6 9b 76 9b 00 80 10   ·d·t·P+· ···v····
0030  01 f6 72 72 00 00 01 01  08 0a 11 81 5e ea a2 e5   ··rr···· ····^···
0040  92 7f                                              ··

wireshark_eth0F19K90.pcapng    Packets: 52 · Displayed: 10 (19.2%) · Dropped: 0 (0.0%)    Profile: Default
```

Analyze SYN flood attack

SYN flood occurs when an attacker delivers a substantial amount of SYN packets to a server using fake IPs, causing the server to respond with an SYN+ACK and keep its ports partially open, expecting a response from an invisible client.

By overwhelming a victim with SYN packets, an attacker can effectively overrun the victim's resources. In this state, the victim fights with traffic, which causes processor and memory usage to rise, eventually exhausting the victim's resources.

Use the hping3 tool to flood the victim IP. Simultaneously, start capturing the traffic on Wireshark. Input 'tcp.flags.syn == 1' in the filter box to view SYN packets flood.

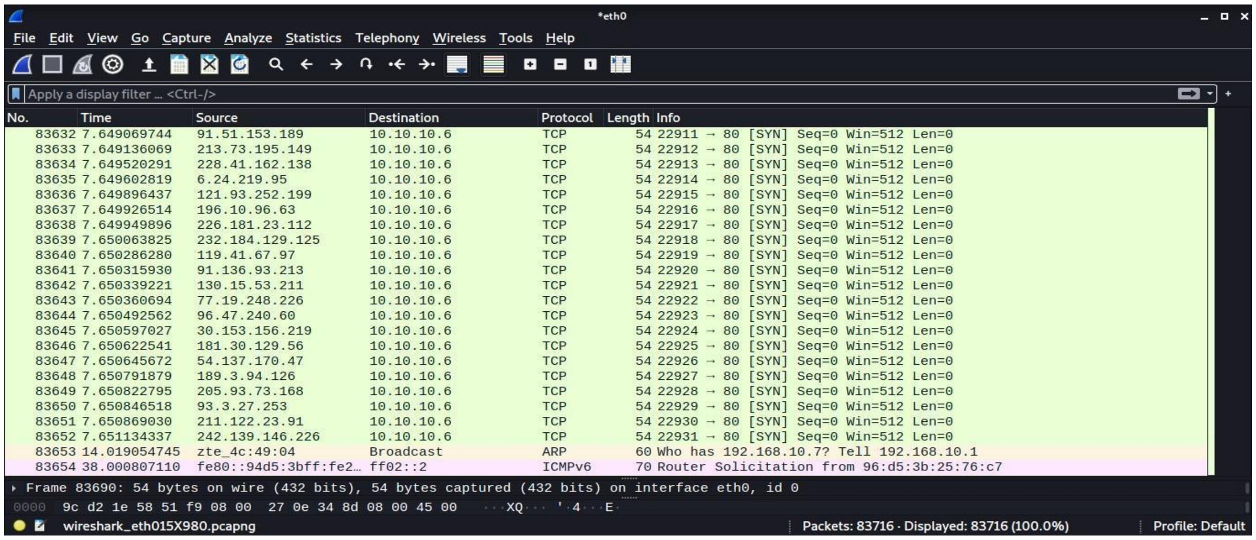


```
┌──(kali㉿kali)-[~]
└─$ sudo hping3 10.10.10.6 -S -p 80 --rand-source --flood
HPING 10.10.10.6 (eth0 10.10.10.6): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.10.10.6 hping statistic ---
```

tcp.flags.syn == 1

```
No.  Time         Source           Destination  Protocol Length Info
   1 0.000000000  122.136.184.205  10.10.10.6   TCP      54 4816 → 80 [SYN] Seq=0 Wi…
   2 0.000076877  194.199.53.10    10.10.10.6   TCP      54 4817 → 80 [SYN] Seq=0 Wi…
   3 0.001094074  145.19.77.18     10.10.10.6   TCP      54 4818 → 80 [SYN] Seq=0 Wi…
   4 0.001155134  37.181.150.202   10.10.10.6   TCP      54 4819 → 80 [SYN] Seq=0 Wi…
   5 0.001214405  50.195.234.128   10.10.10.6   TCP      54 4820 → 80 [SYN] Seq=0 Wi…
   6 0.001242313  205.224.51.201   10.10.10.6   TCP      54 4821 → 80 [SYN] Seq=0 Wi…
   7 0.001295284  32.185.137.170   10.10.10.6   TCP      54 4822 → 80 [SYN] Seq=0 Wi…
   8 0.001325074  83.63.74.162     10.10.10.6   TCP      54 4823 → 80 [SYN] Seq=0 Wi…
   9 0.001352108  162.213.9.99     10.10.10.6   TCP      54 4824 → 80 [SYN] Seq=0 Wi…
  10 0.001381961  129.23.238.54    10.10.10.6   TCP      54 4825 → 80 [SYN] Seq=0 Wi…
  11 0.001474323  145.125.93.119   10.10.10.6   TCP      54 4826 → 80 [SYN] Seq=0 Wi…
  12 0.001502611  74.139.53.95     10.10.10.6   TCP      54 4827 → 80 [SYN] Seq=0 Wi…
  13 0.001530364  142.99.45.170    10.10.10.6   TCP      54 4828 → 80 [SYN] Seq=0 Wi…
  14 0.001558978  149.185.121.143  10.10.10.6   TCP      54 4829 → 80 [SYN] Seq=0 Wi…
  15 0.001587589  18.186.83.226    10.10.10.6   TCP      54 4830 → 80 [SYN] Seq=0 Wi…
```

Notice a lot of SYN packets with no lag time.



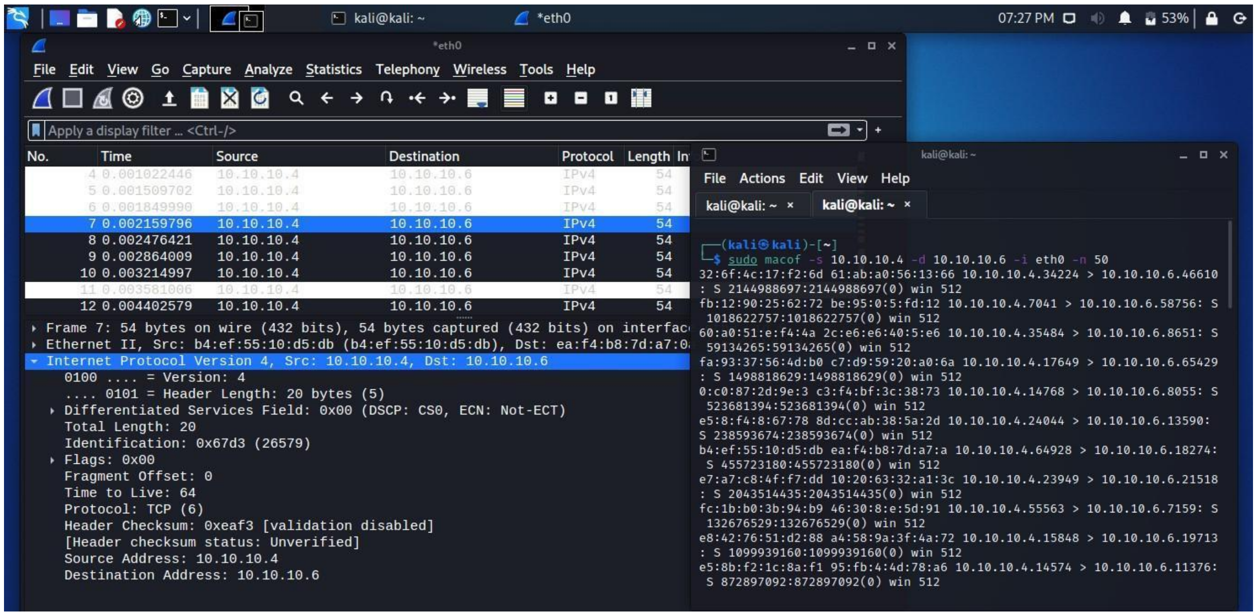
Apply a display filter … <Ctrl-/>

```
No.   Time          Source                Destination  Protocol Length Info
83632 7.649069744   91.51.153.189         10.10.10.6   TCP      54 22911 → 80 [SYN] Seq=0 Win=512 Len=0
83633 7.649136069   213.73.195.149        10.10.10.6   TCP      54 22912 → 80 [SYN] Seq=0 Win=512 Len=0
83634 7.649520291   228.41.162.138        10.10.10.6   TCP      54 22913 → 80 [SYN] Seq=0 Win=512 Len=0
83635 7.649602819   6.24.219.95           10.10.10.6   TCP      54 22914 → 80 [SYN] Seq=0 Win=512 Len=0
83636 7.649896437   121.93.252.199        10.10.10.6   TCP      54 22915 → 80 [SYN] Seq=0 Win=512 Len=0
83637 7.649926514   196.10.96.63          10.10.10.6   TCP      54 22916 → 80 [SYN] Seq=0 Win=512 Len=0
83638 7.649949896   226.181.23.112        10.10.10.6   TCP      54 22917 → 80 [SYN] Seq=0 Win=512 Len=0
83639 7.650063825   232.184.129.125       10.10.10.6   TCP      54 22918 → 80 [SYN] Seq=0 Win=512 Len=0
83640 7.650286280   119.41.67.97          10.10.10.6   TCP      54 22919 → 80 [SYN] Seq=0 Win=512 Len=0
83641 7.650315930   91.136.93.213         10.10.10.6   TCP      54 22920 → 80 [SYN] Seq=0 Win=512 Len=0
83642 7.650339221   130.15.53.211         10.10.10.6   TCP      54 22921 → 80 [SYN] Seq=0 Win=512 Len=0
83643 7.650360694   77.19.248.226         10.10.10.6   TCP      54 22922 → 80 [SYN] Seq=0 Win=512 Len=0
83644 7.650492562   96.47.240.60          10.10.10.6   TCP      54 22923 → 80 [SYN] Seq=0 Win=512 Len=0
83645 7.650597027   30.153.156.219        10.10.10.6   TCP      54 22924 → 80 [SYN] Seq=0 Win=512 Len=0
83646 7.650622541   181.30.129.56         10.10.10.6   TCP      54 22925 → 80 [SYN] Seq=0 Win=512 Len=0
83647 7.650645672   54.137.170.47         10.10.10.6   TCP      54 22926 → 80 [SYN] Seq=0 Win=512 Len=0
83648 7.650791879   189.3.94.126          10.10.10.6   TCP      54 22927 → 80 [SYN] Seq=0 Win=512 Len=0
83649 7.650822795   205.93.73.168         10.10.10.6   TCP      54 22928 → 80 [SYN] Seq=0 Win=512 Len=0
83650 7.650846518   93.3.27.253           10.10.10.6   TCP      54 22929 → 80 [SYN] Seq=0 Win=512 Len=0
83651 7.650869030   211.122.23.91         10.10.10.6   TCP      54 22930 → 80 [SYN] Seq=0 Win=512 Len=0
83652 7.651134337   242.139.146.226       10.10.10.6   TCP      54 22931 → 80 [SYN] Seq=0 Win=512 Len=0
83653 14.019054745  zte_4c:49:04          Broadcast    ARP      60 Who has 192.168.10.7? Tell 192.168.10.1
83654 38.000807110  fe80::94d5:3bff:fe2…  ff02::2      ICMPv6   70 Router Solicitation from 96:d5:3b:25:76:c7
▸ Frame 83690: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface eth0, id 0
0000  9c d2 1e 58 51 f9 08 00  27 0e 34 8d 08 00 45 00   ···XQ··· '·4···E·
wireshark_eth015X980.pcapng          Packets: 83716 · Displayed: 83716 (100.0%)          Profile: Default
```

Analyze DoS attacks

Let’s simulate a Denial of Service (DoS) attack to analyze it via Wireshark. For the demo, I am using the macof tool, the component of the Dsniff suit toolkit, and flooding a surrounding device's switch with MAC addresses.

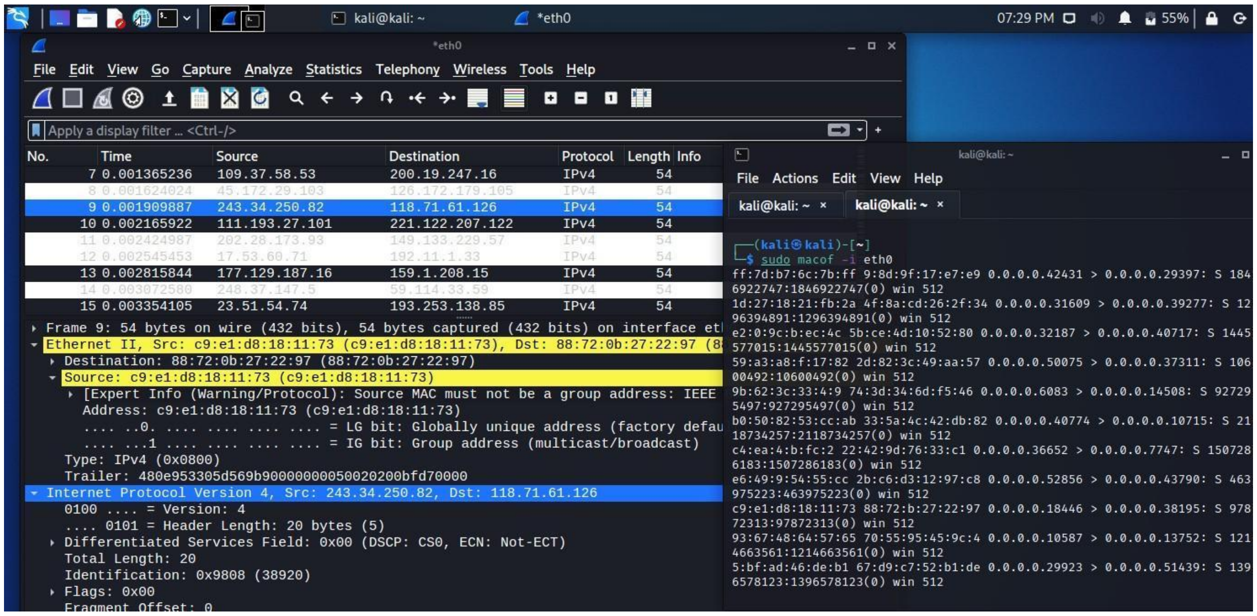
The image below shows IP address is generating requests to another device with the same data size repeatedly. This sort of traffic shows a standard network DoS attack.



```
No.  Time          Source       Destination  Protocol  Length
 4 0.001022446   10.10.10.4   10.10.10.6   IPv4      54
 5 0.001509702   10.10.10.4   10.10.10.6   IPv4      54
 6 0.001849990   10.10.10.4   10.10.10.6   IPv4      54
 7 0.002159796   10.10.10.4   10.10.10.6   IPv4      54
 8 0.002476421   10.10.10.4   10.10.10.6   IPv4      54
 9 0.002864009   10.10.10.4   10.10.10.6   IPv4      54
10 0.003214997   10.10.10.4   10.10.10.6   IPv4      54
11 0.003581006   10.10.10.4   10.10.10.6   IPv4      54
12 0.004402579   10.10.10.4   10.10.10.6   IPv4      54

▸ Frame 7: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interfac
▸ Ethernet II, Src: b4:ef:55:10:d5:db (b4:ef:55:10:d5:db), Dst: ea:f4:b8:7d:a7:0
▾ Internet Protocol Version 4, Src: 10.10.10.4, Dst: 10.10.10.6
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 20
    Identification: 0x67d3 (26579)
  ▸ Flags: 0x00
    Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0xeaf3 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.10.10.4
    Destination Address: 10.10.10.6
```

```
┌──(kali㉿kali)-[~]
└─$ sudo macof -s 10.10.10.4 -d 10.10.10.6 -i eth0 -n 50
32:6f:4c:17:f2:6d 61:ab:a0:56:13:66 10.10.10.4.34224 > 10.10.10.6.46610
: S 2144988697:2144988697(0) win 512
fb:12:90:25:62:72 be:95:0:5:fd:12 10.10.10.4.7041 > 10.10.10.6.58756: S
 1018622757:1018622757(0) win 512
60:a0:51:e:f4:4a 2c:e6:e6:40:5:e6 10.10.10.4.35484 > 10.10.10.6.8651: S
 59134265:59134265(0) win 512
fa:93:37:56:4d:b0 c7:d9:59:20:a0:6a 10.10.10.4.17649 > 10.10.10.6.65429
: S 1498818629:1498818629(0) win 512
0:c0:87:2d:9e:3 c3:f4:bf:3c:38:73 10.10.10.4.14768 > 10.10.10.6.8055: S
 523681394:523681394(0) win 512
e5:8:f4:8:67:78 8d:cc:ab:38:5a:2d 10.10.10.4.24044 > 10.10.10.6.13590:
S 238593674:238593674(0) win 512
b4:ef:55:10:d5:db ea:f4:b8:7d:a7:a 10.10.10.4.64928 > 10.10.10.6.18274:
 S 455723180:455723180(0) win 512
e7:a7:c8:4f:f7:dd 10:20:63:32:a1:3c 10.10.10.4.23949 > 10.10.10.6.21518
: S 2043514435:2043514435(0) win 512
fc:1b:b0:3b:94:b9 46:30:8:e:5d:91 10.10.10.4.55563 > 10.10.10.6.7159: S
 132676529:132676529(0) win 512
e8:42:76:51:d2:88 a4:58:9a:3f:4a:72 10.10.10.4.15848 > 10.10.10.6.19713
: S 1099939160:1099939160(0) win 512
e5:8b:f2:1c:8a:f1 95:fb:4:4d:78:a6 10.10.10.4.14574 > 10.10.10.6.11376:
 S 872897092:872897092(0) win 512
```

For a DDoS attack, use the macof tool again to generate traffic. Observe the fake source and destination IP addresses are sending many packets with similar data sizes.



```
No.  Time          Source            Destination        Protocol  Length  Info
 7 0.001365236   109.37.58.53      200.19.247.16      IPv4      54
 8 0.001624024   45.172.29.103     126.172.179.105    IPv4      54
 9 0.001909887   243.34.250.82     118.71.61.126      IPv4      54
10 0.002165922   111.193.27.101    221.122.207.122    IPv4      54
11 0.002424987   202.28.173.93     149.133.229.57     IPv4      54
12 0.002545453   17.53.60.71       192.11.1.33        IPv4      54
13 0.002815844   177.129.187.16    159.1.208.15       IPv4      54
14 0.003072580   248.37.147.5      59.114.33.59       IPv4      54
15 0.003354105   23.51.54.74       193.253.138.85     IPv4      54

▸ Frame 9: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface et
▾ Ethernet II, Src: c9:e1:d8:18:11:73 (c9:e1:d8:18:11:73), Dst: 88:72:0b:27:22:97 (8
  ▸ Destination: 88:72:0b:27:22:97 (88:72:0b:27:22:97)
  ▾ Source: c9:e1:d8:18:11:73 (c9:e1:d8:18:11:73)
    ▸ [Expert Info (Warning/Protocol): Source MAC must not be a group address: IEEE
      Address: c9:e1:d8:18:11:73 (c9:e1:d8:18:11:73)
      .... ..0. .... .... .... .... = LG bit: Globally unique address (factory defau
      .... ...1 .... .... .... .... = IG bit: Group address (multicast/broadcast)
    Type: IPv4 (0x0800)
    Trailer: 480e953305d569b90000000050020200bfd70000
▾ Internet Protocol Version 4, Src: 243.34.250.82, Dst: 118.71.61.126
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 20
    Identification: 0x9808 (38920)
  ▸ Flags: 0x00
    Fragment Offset: 0
```

```
┌──(kali㉿kali)-[~]
└─$ sudo macof -i eth0
ff:7d:b7:6c:7b:ff 9:8d:9f:17:e7:e9 0.0.0.0.42431 > 0.0.0.0.29397: S 184
6922747:1846922747(0) win 512
1d:27:18:21:fb:2a 4f:8a:cd:26:2f:34 0.0.0.0.31609 > 0.0.0.0.39277: S 12
96394891:1296394891(0) win 512
e2:0:9c:b:ec:4c 5b:ce:4d:10:52:80 0.0.0.0.32187 > 0.0.0.0.40717: S 1445
577015:1445577015(0) win 512
59:a3:a8:f:17:82 2d:82:3c:49:aa:57 0.0.0.0.50075 > 0.0.0.0.37311: S 106
00492:10600492(0) win 512
9b:62:3c:33:4:9 74:3d:34:6d:f5:46 0.0.0.0.6083 > 0.0.0.0.14508: S 92729
5497:927295497(0) win 512
b0:50:82:53:cc:ab 33:5a:4c:42:db:82 0.0.0.0.40774 > 0.0.0.0.10715: S 21
18734257:2118734257(0) win 512
c4:ea:4:b:fc:2 22:42:9d:76:33:c1 0.0.0.0.36652 > 0.0.0.0.7747: S 150728
6183:1507286183(0) win 512
e6:49:9:54:55:cc 2b:c6:d3:12:97:c8 0.0.0.0.52856 > 0.0.0.0.43790: S 463
975223:463975223(0) win 512
c9:e1:d8:18:11:73 88:72:b:27:22:97 0.0.0.0.18446 > 0.0.0.0.38195: S 978
72313:97872313(0) win 512
93:67:48:64:57:65 70:55:95:45:9c:4 0.0.0.0.10587 > 0.0.0.0.13752: S 121
4663561:1214663561(0) win 512
5:bf:ad:46:de:b1 67:d9:c7:52:b1:de 0.0.0.0.29923 > 0.0.0.0.51439: S 139
6578123:1396578123(0) win 512
```

Wireshark is an essential tool that many blue team and network administrators use daily. The objective might differ, but they analyze network traffic using it. In this article, we have explored several network traffic types like HTTPS, TCP, etc. In addition, we have seen few attacks using Wireshark, like the DoS attack.