**PRACTICAL- 1**

**AIM: To implement connection oriented client/server mechanism using socket programming. Mini Project based on Socket programming in group of TWO students. (In any programming language).**

**Hardware requirements:** Pc/Laptop.

**Software requirements:** Command prompt, NetBeans IDE 8.2

**Knowledge requirements:** Basic knowledge about Socket Programming.

**Theory :**

**Socket Programming**

* Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.
* When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.
* The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

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

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

After the connections are established, communication can occur using I/O streams. TCP is a two-way communication protocol; hence data can be sent across both streams at the same time. Following are the useful classes providing complete set of methods to implement sockets.

ServerSocket Class Methods

* The **java.net.ServerSocket** class is used by server applications to obtain a port and listen for client requests.

The ServerSocket class has four constructors −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public ServerSocket(int port) throws IOException**  Attempts to create a server socket bound to the specified port. An exception occurs if the port is already bound by another application. |
| 2 | **public ServerSocket (int port, int backlog) throws IOException**  Similar to the previous constructor, the backlog parameter specifies how many incoming clients to store in a wait queue. |
| 3 | **public ServerSocket (int port, int backlog, InetAddress address) throws IOException**  Similar to the previous constructor, the InetAddress parameter specifies the local IP address to bind to. The InetAddress is used for servers that may have multiple IP addresses, allowing the server to specify which of its IP addresses to accept client requests on. |
| 4 | **public ServerSocket () throws IOException**  Creates an unbound server socket. When using this constructor, use the bind() method when you are ready to bind the server socket. |

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

Following are some of the common methods of the ServerSocket class −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public int getLocalPort()**  Returns the port that the server socket is listening on. This method is useful if you passed in 0 as the port number in a constructor and let the server find a port for you. |
| 2 | **public Socket accept () throws IOException**  Waits for an incoming client. This method blocks until either a client connects to the server on the specified port or the socket times out, assuming that the time-out value has been set using the setSoTimeout() method. Otherwise, this method blocks indefinitely. |
| 3 | **public void setSoTimeout(int timeout)**  Sets the time-out value for how long the server socket waits for a client during the accept(). |
| 4 | **public void bind(SocketAddress host, int backlog)**  Binds the socket to the specified server and port in the SocketAddress object. Use this method if you have instantiated the ServerSocket using the no-argument constructor. |

* When the ServerSocket invokes accept(), the method does not return until a client connects. After a client does connect, the ServerSocket creates a new Socket on an unspecified port and returns a reference to this new Socket. A TCP connection now exists between the client and the server, and communication can begin.

Socket Class Methods

* The **java.net.Socket** class represents the socket that both the client and the server use to communicate with each other. The client obtains a Socket object by instantiating one, whereas the server obtains a Socket object from the return value of the accept() method.

The Socket class has five constructors that a client uses to connect to a server –

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public Socket(String host, int port) throws UnknownHostException, IOException.**  This method attempts to connect to the specified server at the specified port. If this constructor does not throw an exception, the connection is successful and the client is connected to the server. |
| 2 | **public Socket(InetAddress host, int port) throws IOException**  This method is identical to the previous constructor, except that the host is denoted by an InetAddress object. |
| 3 | **public Socket(String host, int port, InetAddress localAddress, int localPort) throws IOException.**  Connects to the specified host and port, creating a socket on the local host at the specified address and port. |
| 4 | **public Socket(InetAddress host, int port, InetAddress localAddress, int localPort) throws IOException.**  This method is identical to the previous constructor, except that the host is denoted by an InetAddress object instead of a String. |
| 5 | **public Socket()**  Creates an unconnected socket. Use the connect() method to connect this socket to a server. |

When the Socket constructor returns, it does not simply instantiate a Socket object but it actually attempts to connect to the specified server and port.

Some methods of interest in the Socket class are listed here. Notice that both the client and the server have a Socket object, so these methods can be invoked by both the client and the server.

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **public void connect(SocketAddress host, int timeout) throws IOException**  This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor. |
| 2 | **public InetAddress getInetAddress()**  This method returns the address of the other computer that this socket is connected to. |
| 3 | **public int getPort()**  Returns the port the socket is bound to on the remote machine. |
| 4 | **public int getLocalPort()**  Returns the port the socket is bound to on the local machine. |
| 5 | **public SocketAddress getRemoteSocketAddress()**  Returns the address of the remote socket. |
| 6 | **public InputStream getInputStream() throws IOException**  Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket. |
| 7 | **public OutputStream getOutputStream() throws IOException**  Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket. |
| 8 | **public void close() throws IOException**  Closes the socket, which makes this Socket object no longer capable of connecting again to any server. |

Socket Client Example

* The following Chatclient is a client program that connects to a server by using a socket and sends a greeting, and then waits for a response.

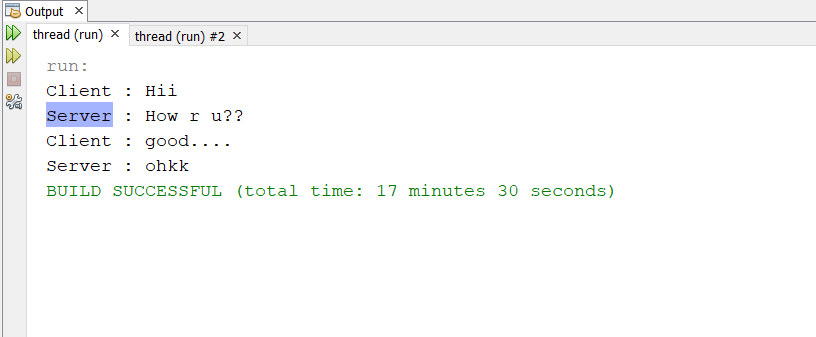
|  |
| --- |
| package Socket;  import java.io.BufferedReader;  import java.io.InputStreamReader;  import java.io.PrintStream;  import java.net.Socket;  public class chatclient {  public static void main(String args[]) throws Exception  {  Socket sk=new Socket("10.20.2.95",2000);  BufferedReader sin=new  BufferedReader(new InputStreamReader(sk.getInputStream()));  PrintStream sout=new PrintStream(sk.getOutputStream());  BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));  String s;  while ( true )  {  System.out.print("Client : ");  s=stdin.readLine();  sout.println(s);  s=sin.readLine();  System.out.print("Server : "+s+"\n");  if ( s.equalsIgnoreCase("BYE") )  break;  }  sk.close();  sin.close();  sout.close();  stdin.close();  }  } |

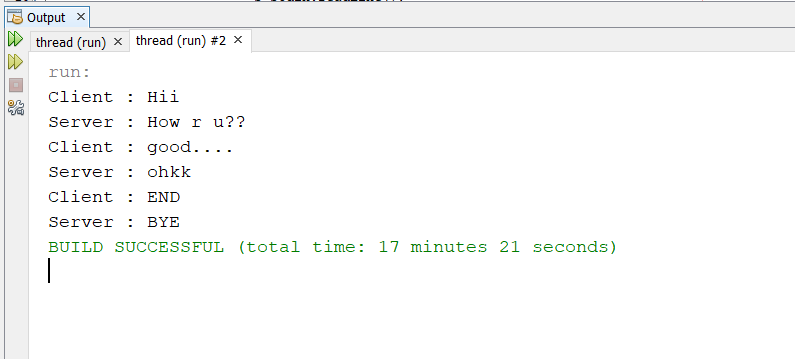
Socket Server

* The following Chatserver program is an example of a server application that uses the Socket class to listen for clients on a port number specified by a command-line argument

|  |
| --- |
| package Socket;  import java.net.\*;  import java.io.\*;    public class chatserver  {  public static void main(String args[]) throws Exception  {  ServerSocket ss=new ServerSocket (2000);  Socket sk=ss.accept();  BufferedReader cin=new  BufferedReader (new InputStreamReader(sk.getInputStream()));  PrintStream cout=new PrintStream(sk.getOutputStream ());  BufferedReader stdin=new BufferedReader (new InputStreamReader(System.in));  String s;  while ( true )  {  s=cin.readLine();  if (s.equalsIgnoreCase("END"))  {  cout.println("BYE");  break;  }  System. out.print("Client : "+s+"\n");  System.out.print("Server : ");  s=stdin.readLine();  cout.println(s);  }  ss.close();  sk.close();  cin.close();  cout.close();  stdin.close();  }  } |

**Output:**

Server

Client

**Conclusion**

Thus we have checked Response of Established Connection from both client and server Side Connection.

**PRACTICAL-2**

**AIM: Introduction of Packet Tracer and configure network topology for Static and Dynamic Routing Protocol using packet tracer.**

**Hardware requirements:** pc/laptop.

**Software requirements:** CISCO packet tracer

**Knowledge requirements:** Basic knowledge about addresses, switch, routers.

**Theory:**

**Packet Tracer**

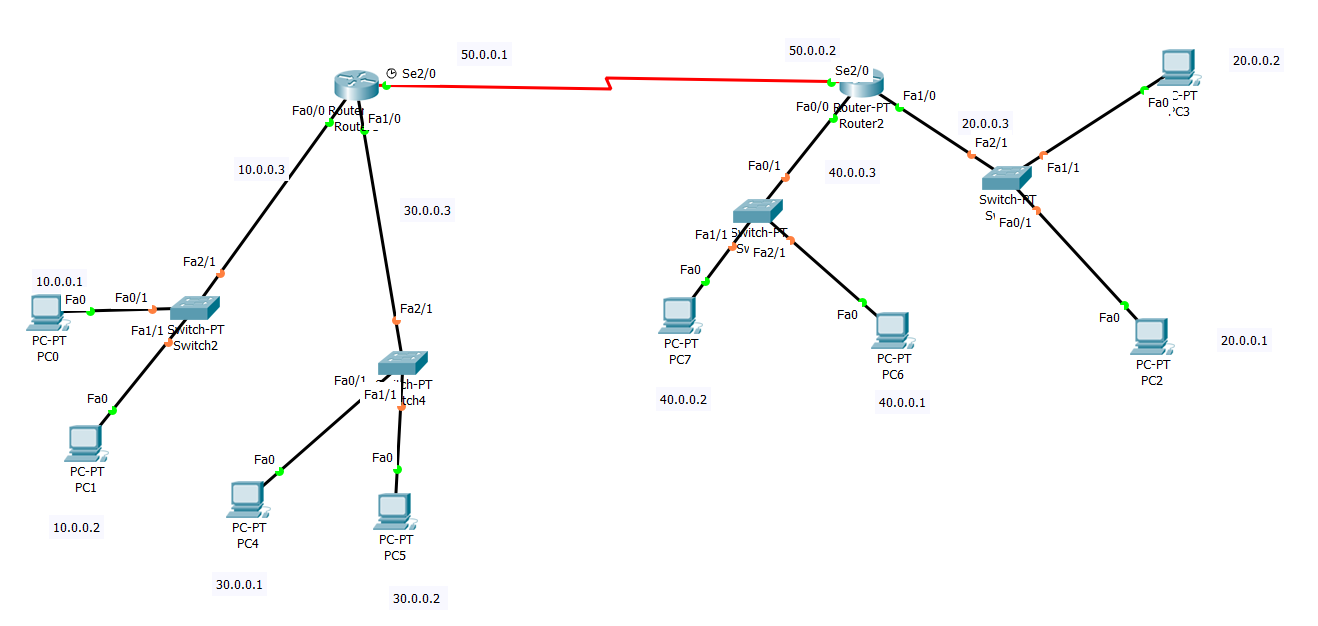
* Packet Tracer is a [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) visual [simulation](https://en.wikipedia.org/wiki/Simulation) tool designed by [Cisco Systems](https://en.wikipedia.org/wiki/Cisco_Systems) that allows users to create [network topologies](https://en.wikipedia.org/wiki/Network_topologies) and imitate modern [computer networks](https://en.wikipedia.org/wiki/Computer_networks).
* The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface.
* Packet Tracer makes use of a [drag and drop](https://en.wikipedia.org/wiki/Drag_and_drop) user interface, allowing users to add and remove simulated network devices as they see fit.
* The software is mainly focused towards Certified Cisco Network Associate Academy students as an educational tool for helping them learn fundamental CCNA concepts.

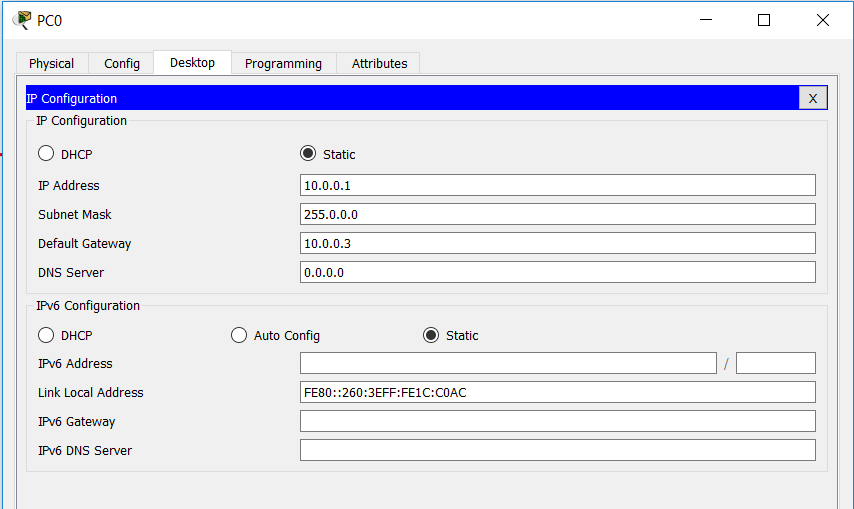
**Overview**

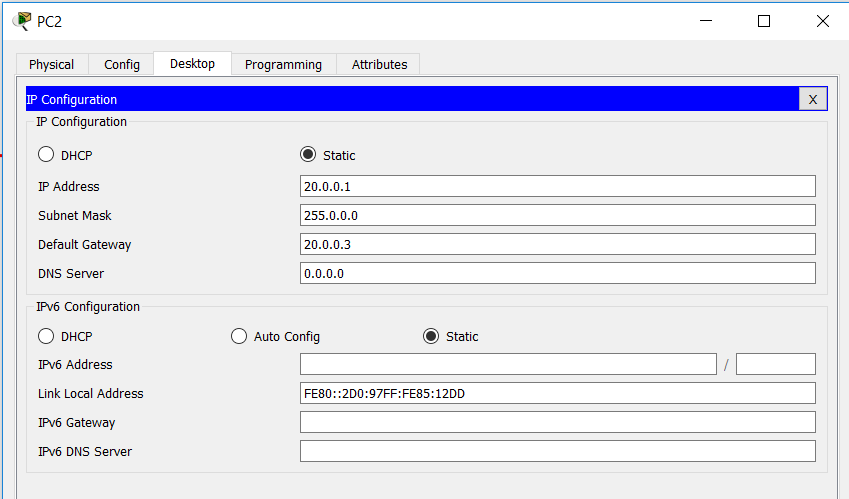
* Packet Tracer can be run on Linux and Microsoft Windows. Similar [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) and [iOS](https://en.wikipedia.org/wiki/IOS) apps are also available. Packet Tracer allows users to create simulated network topologies by dragging and dropping routers, switches and various other types of network devices.
* A physical connection between devices is represented by a "cable" item.
* Packet Tracer supports an array of simulated [Application Layer protocols](https://en.wikipedia.org/wiki/Application_Layer), as well as basic routing with [RIP](https://en.wikipedia.org/wiki/Routing_Information_Protocol), [OSPF](https://en.wikipedia.org/wiki/OSPF), [EIGRP](https://en.wikipedia.org/wiki/EIGRP), [BGP](https://en.wikipedia.org/wiki/BGP), to the extents required by the current [CCNA](https://en.wikipedia.org/wiki/CCNA) curriculum.
* In addition to simulating certain aspects of [computer networks](https://en.wikipedia.org/wiki/Computer_networks), Packet Tracer can also be used for collaboration. As of Packet Tracer 5.0, Packet Tracer supports a multi-user system that enables multiple users to connect multiple topologies together over a [computer network](https://en.wikipedia.org/wiki/Computer_network).
* Packet Tracer also allows instructors to create activities that students have to complete. Packet Tracer is often used in educational settings as a learning aid.

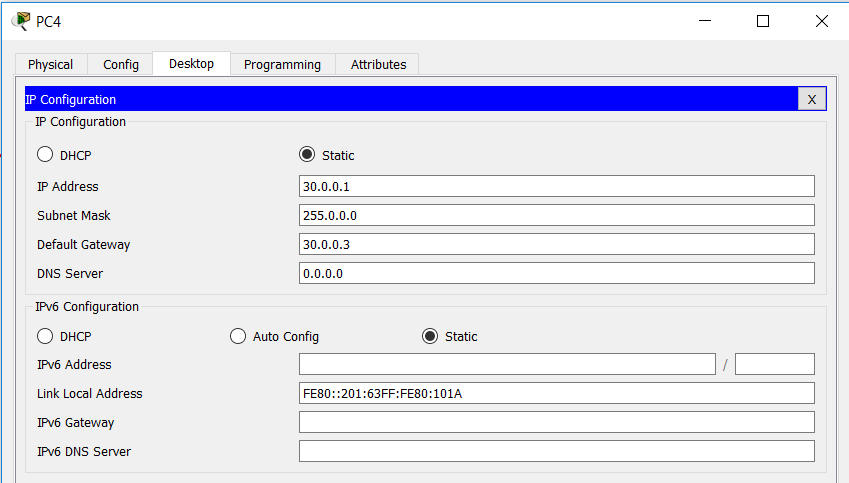
**Static**

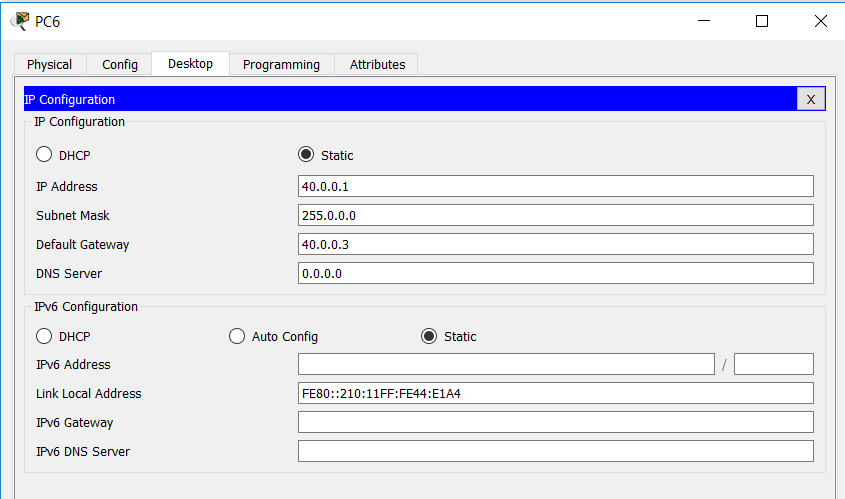
* Routers forward packets using either route information from route table entries that you manually configure or the route information that is calculated using dynamic routing algorithms.
* Static routes, which define explicit paths between two routers, cannot be automatically updated; you must manually reconfigure static routes when network changes occur. Static routes use less bandwidth than dynamic routes. No CPU cycles are used to calculate and analyze routing updates.
* You can supplement dynamic routes with static routes where appropriate. You can redistribute static routes into dynamic routing algorithms, but you cannot redistribute routing information calculated by dynamic routing algorithms into the static routing table.

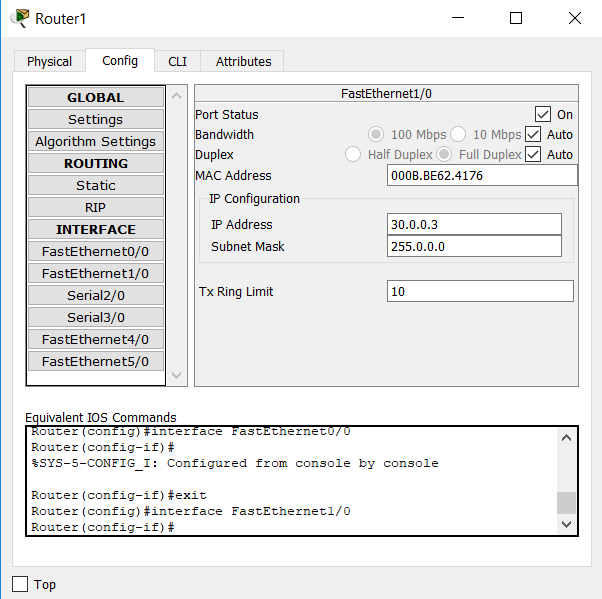


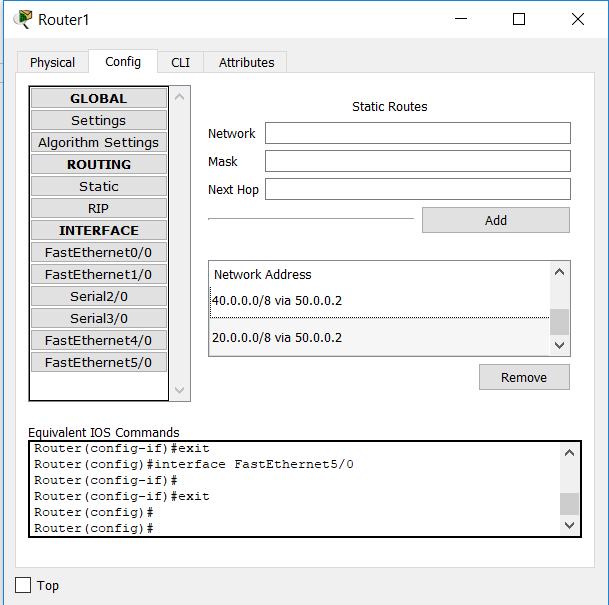


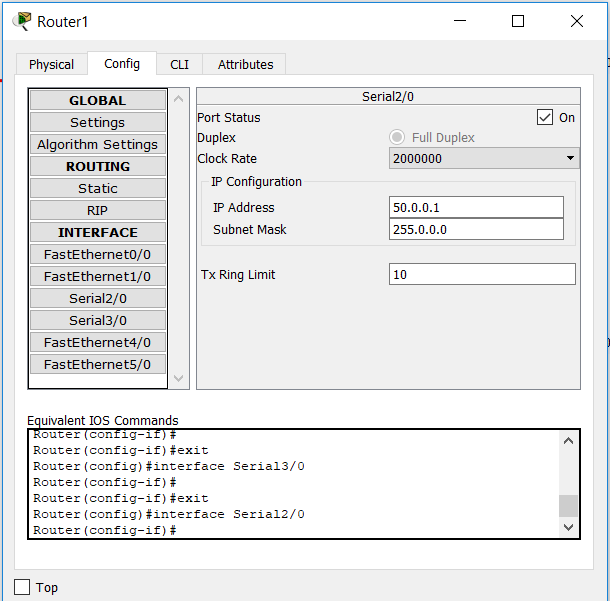


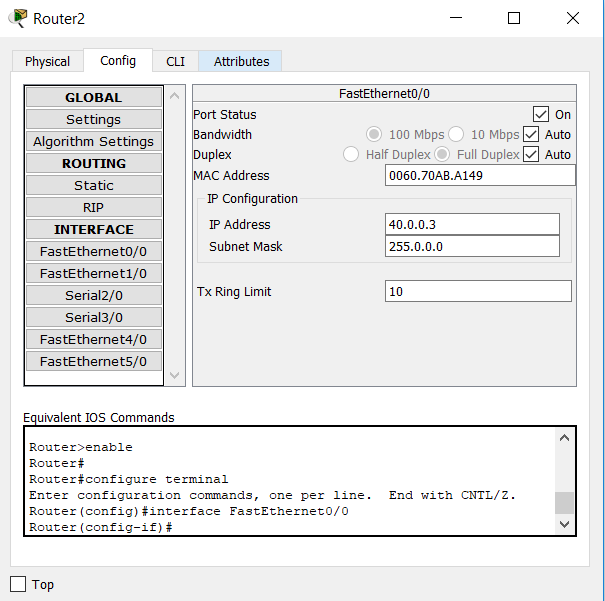


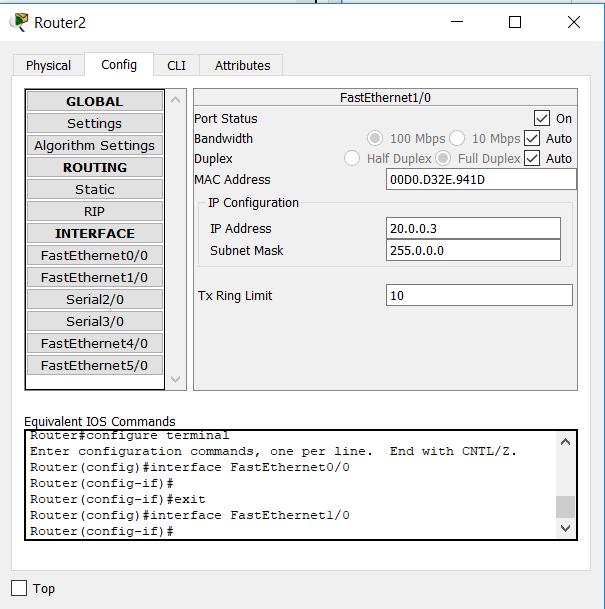


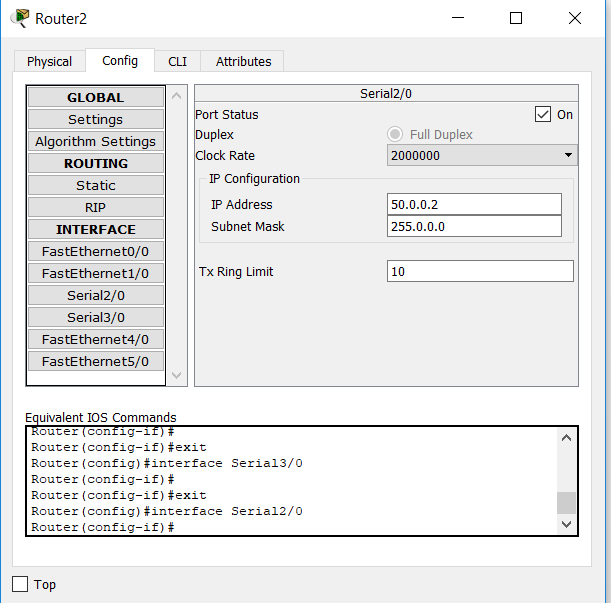


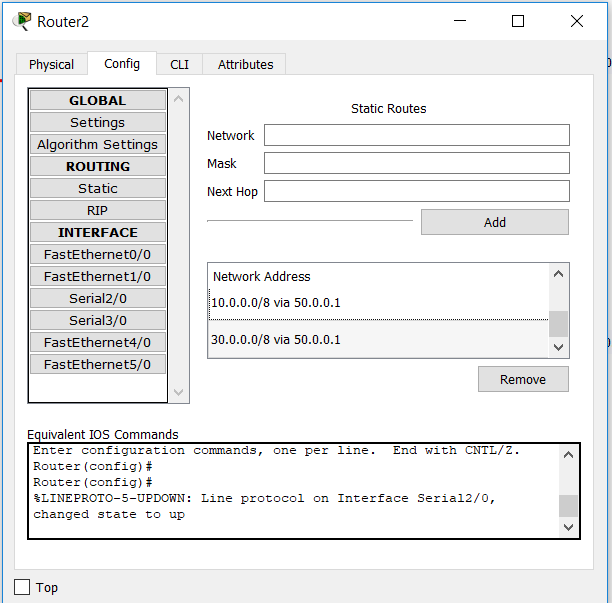


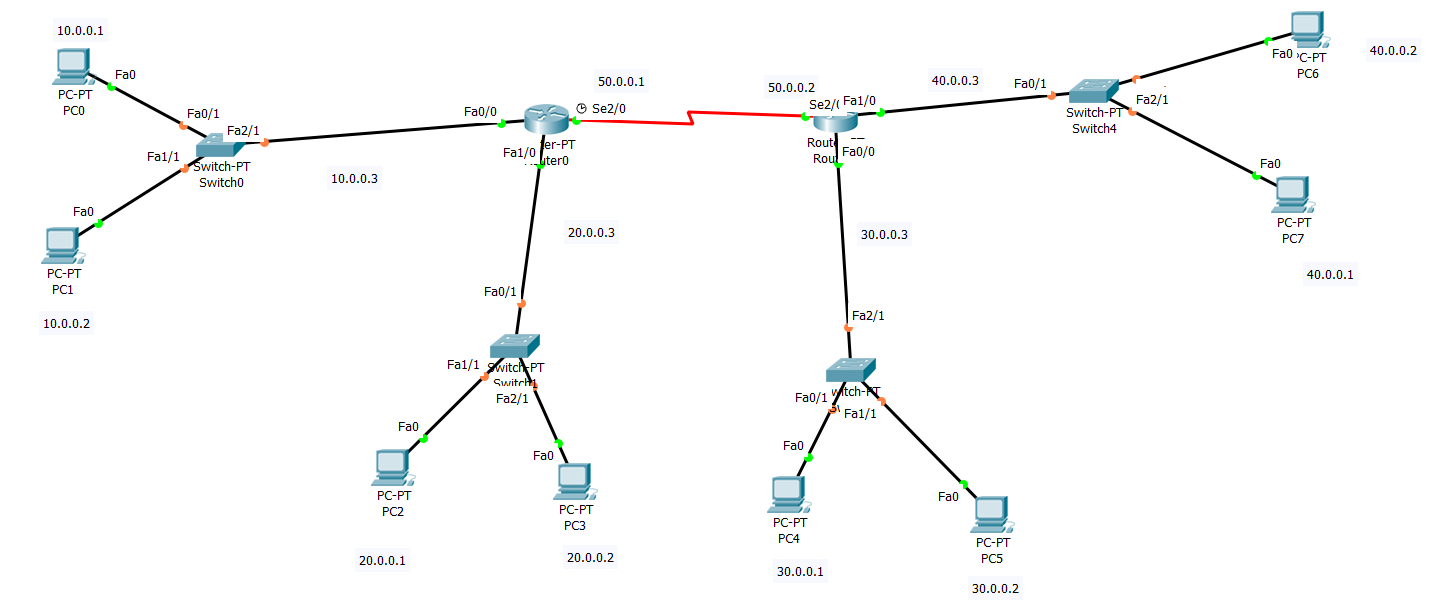








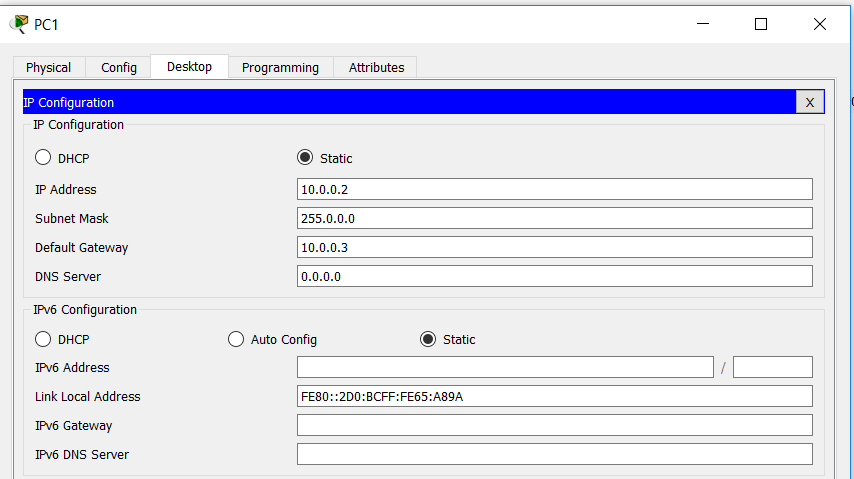


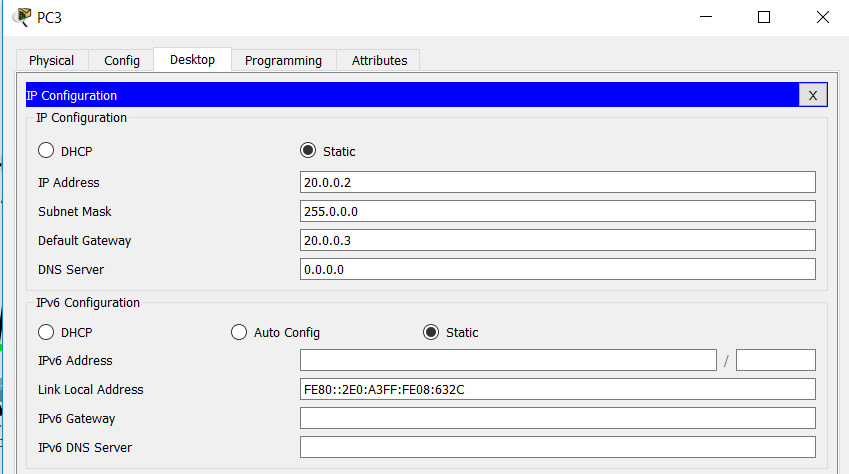
**RIP (Routing Information Protocol)**

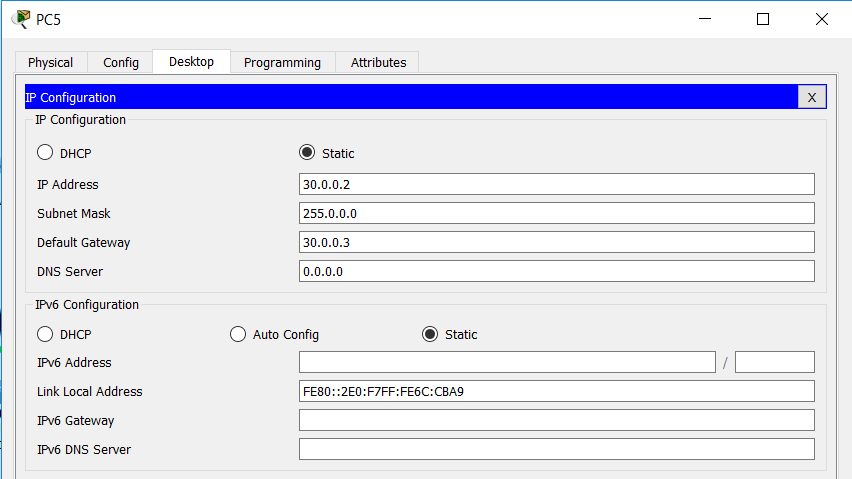
* The Routing Information Protocol (RIP) is one of the oldest [distance-vector routing protocols](https://en.wikipedia.org/wiki/Distance-vector_routing_protocols) which

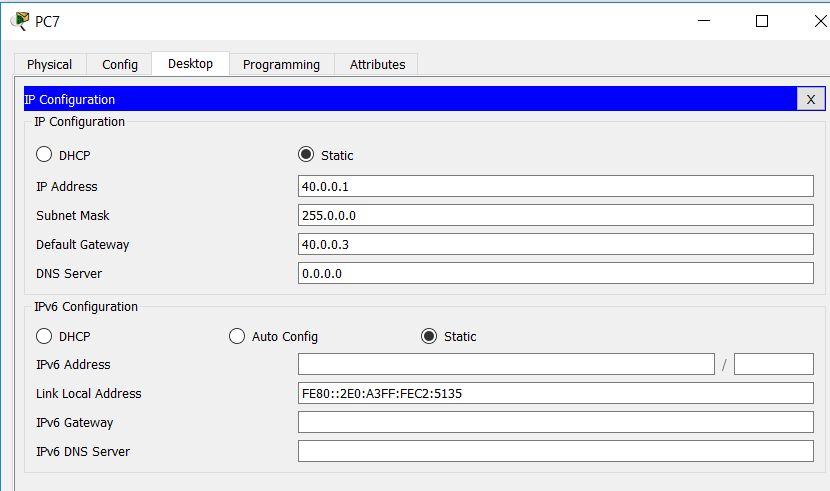
employ the [hop count](https://en.wikipedia.org/wiki/Hopcount) as a [routing metric](https://en.wikipedia.org/wiki/Metrics_(networking)).

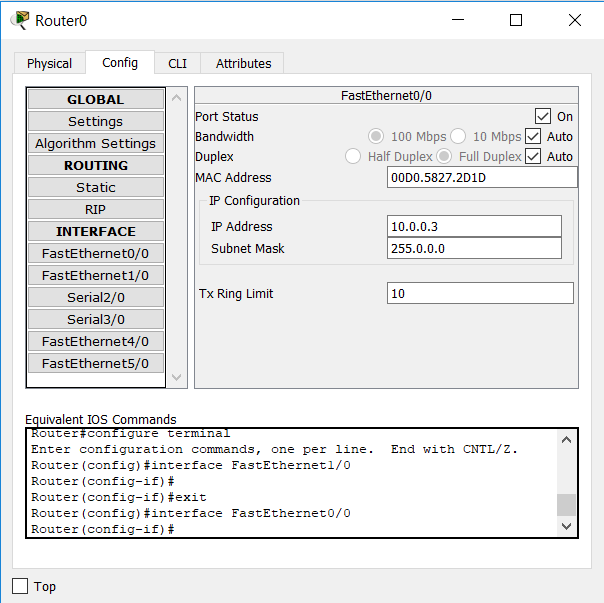
* RIP prevents [routing loops](https://en.wikipedia.org/wiki/Routing_loop_problem) by implementing a limit on the number of [hops](https://en.wikipedia.org/wiki/Hop_(telecommunications)) allowed in a path from source to destination.
* The largest number of hops allowed for RIP is 15, which limits the size of networks that RIP can support.
* RIP implements the [split horizon](https://en.wikipedia.org/wiki/Split_horizon), [route poisoning](https://en.wikipedia.org/wiki/Route_poisoning) and [hold-down](https://en.wikipedia.org/wiki/Holddown) mechanisms to prevent incorrect routing information from being propagated.

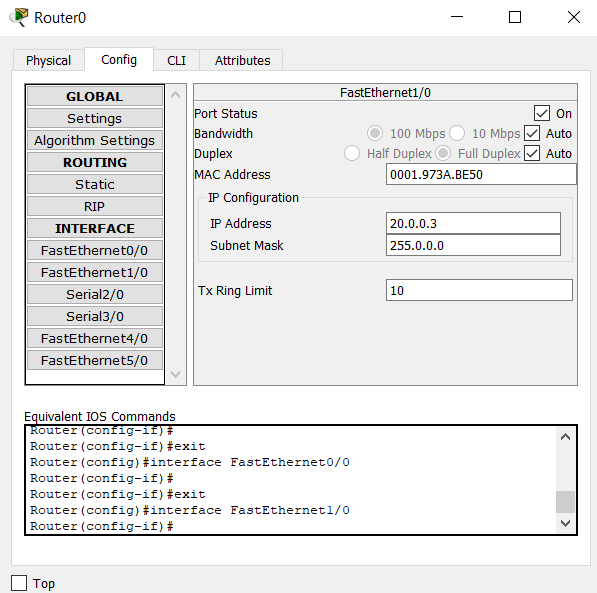


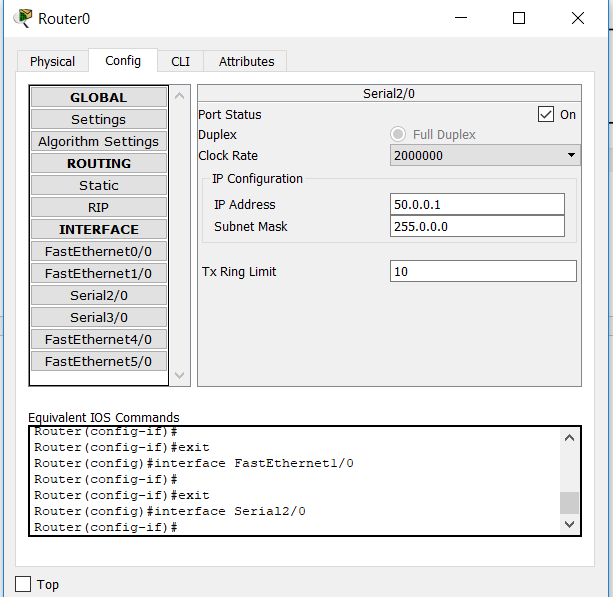


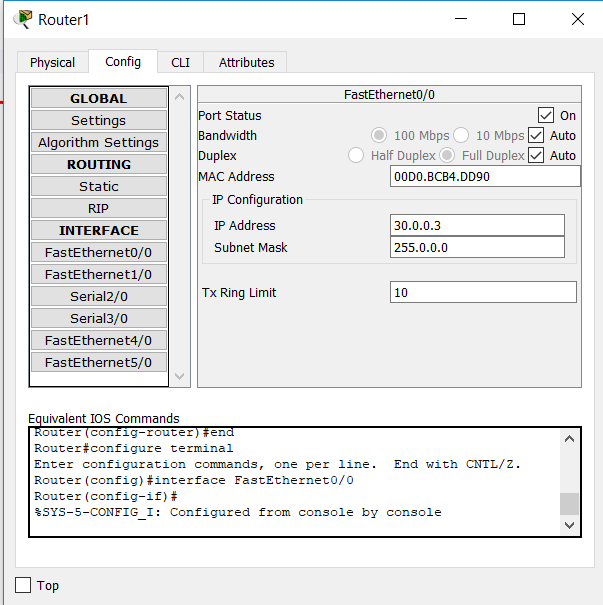


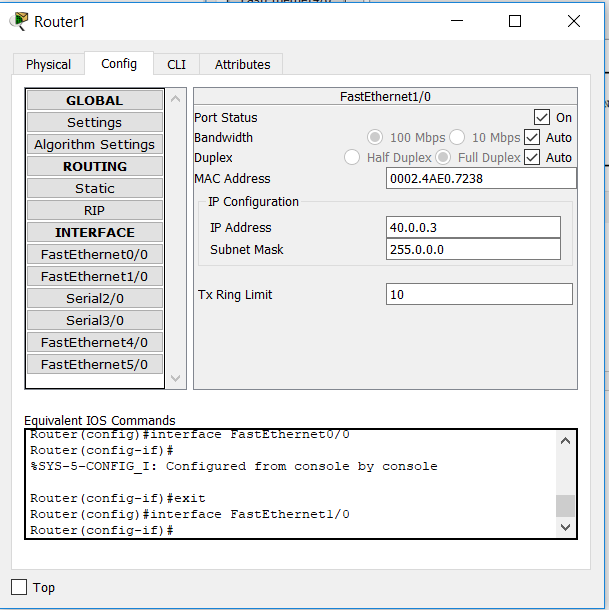


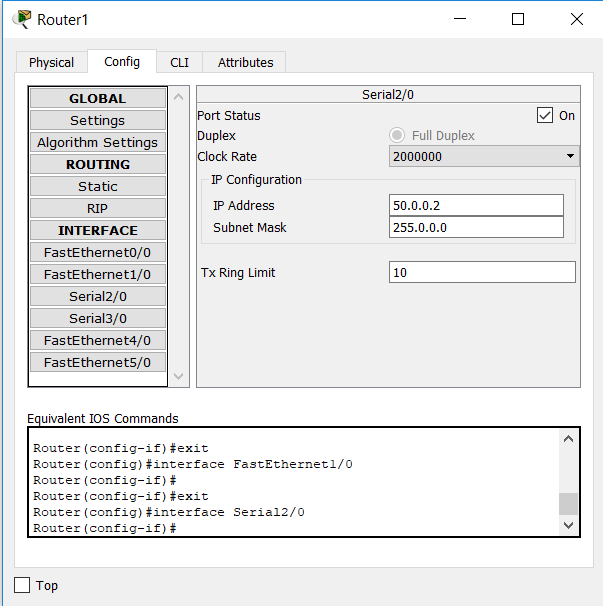


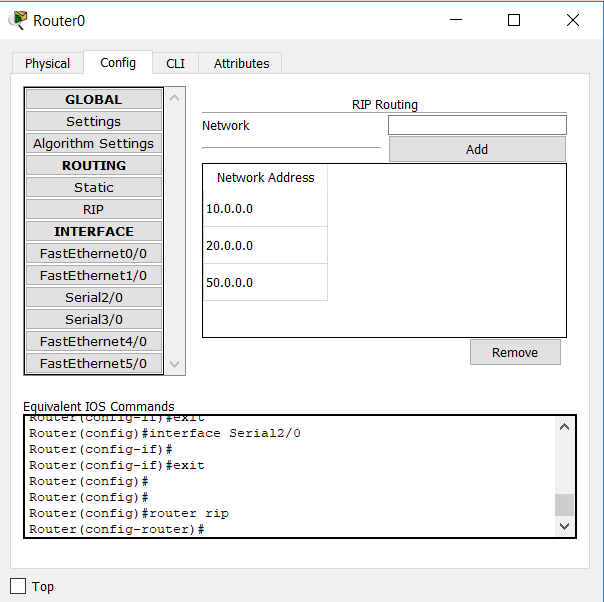


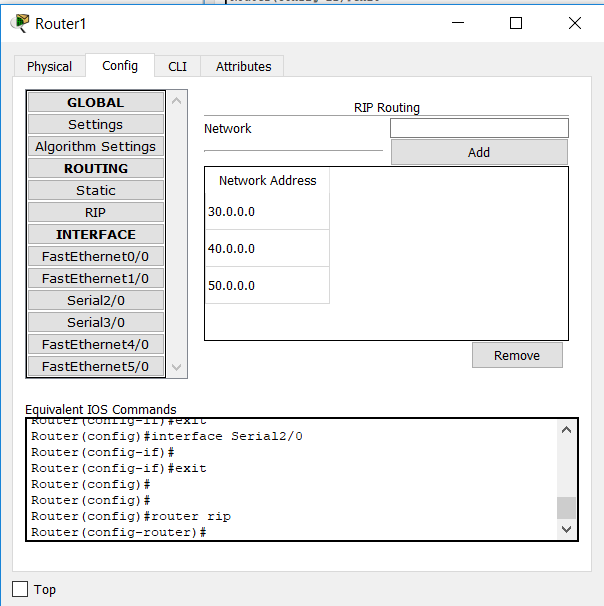












**Conclusion**

We establish connection between routers, PCs and switches and we can able to send message from one Pc to another Pc.

**PRACTICAL-3**

**AIM: To configure the concept of Sub-netting and Issues of Sub-netting.**

**Hardware requirements:** pc/laptop.

**Software requirements:** CISCO packet tracer

**Knowledge requirements:** Basic knowledge about addresses, switch, routers, subnet mask.

**Theory:**

**Subnet :**

An [IP address](https://www.iplocation.net/ip-address) has two components, the network address and the host address. A subnet mask separates the IP address into the network and host addresses (<network><host>). Subnetting further divides the host part of an IP address into a subnet and host address (<network><subnet><host>) if additional subnetwork is needed.

It is called a subnet mask because it is used to identify network address of an IP address by performing a bitwise AND operation on the netmask.

A Subnet mask is a 32-bit number that masks an IP address, and divides the IP address into network address and host address. Subnet Mask is made by setting network bits to all "1"s and setting host bits to all "0"s. Within a given network, two host addresses are reserved for special purpose, and cannot be assigned to hosts. The "0" address is assigned a network address and "255" is assigned to a broadcast address, and they cannot be assigned to hosts.

Examples of commonly used netmasks for classed networks are 8-bits (Class A), 16-bits (Class B) and 24-bits (Class C), and classless networks are as follows:

Subnetting an IP network is to separate a big network into smaller multiple networks for reorganization and security purposes. All nodes (hosts) in a subnetwork see all packets transmitted by any node in a network. Performance of a network is adversely affected under heavy traffic load due to collisions and retransmissions.

Applying a subnet mask to an IP address separates network address from host address. The network bits are represented by the 1's in the mask, and the host bits are represented by 0's. Performing a bitwise logical AND operation on the IP address with the subnet mask produces the network address. For example, applying the Class C subnet mask to our IP address 216.3.128.12 produces the following network address:

IP: 1101 1000. 0000 0011. 1000 0000. 0000 1100 (216.003.128.012)

Mask: 1111 1111. 1111 1111. 1111 1111. 0000 0000 (255.255.255.000)

---------------------------------------------

1101 1000. 0000 0011. 1000 0000. 0000 0000 (216.003.128.000)

**Subnetting Network**

Here is another scenario where subnetting is needed. Pretend that a web host with a Class C network needs to divide the network so that parts of the network can be leased to its customers. Let's assume that a host has a network address of 216.3.128.0 (as shown in the example above). Let's say that we're going to divide the network into 2 and dedicate the first half to itself, and the other half to its customers.

216. 3. 128. (0000 0000) (1st half assigned to the web host)

216. 3. 128. (1000 0000) (2nd half assigned to the customers)

The web host will have the subnet mask of 216.3.128.128 (/25). Now, we'll further divide the 2nd half into eight blocks of 16 IP addresses.

216.3. 128. (1000 0000) Customer 1 -- Gets 16 IPs (14 usable)

216.3. 128. (1001 0000) Customer 2 -- Gets 16 IPs (14 usable)

216.3. 128. (1010 0000) Customer 3 -- Gets 16 IPs (14 usable)

216.3. 128. (1011 0000) Customer 4 -- Gets 16 IPs (14 usable)

216.3. 128. (1100 0000) Customer 5 -- Gets 16 IPs (14 usable)

216.3. 128. (1101 0000) Customer 6 -- Gets 16 IPs (14 usable)

216.3. 128. (1110 0000) Customer 7 -- Gets 16 IPs (14 usable)

216.3. 128. (1111 0000) Customer 8 -- Gets 16 IPs (14 usable)

-----------------------------

255. 255. 255. (1111 0000) (Subnet mask of 255.255.255.240)

**For given IP address 192.168.12.0 create six extra such a way that subnet 1 has 128 host, Subnet 2 has 64 host, subnet 3 has 32 host and subnet 4,5,6 has 4 host each.**

**Subnet 1 (128 Host) : -**

From: -192.168.12.0 to 192.168.12.127

Subnet mask: -255.255.255.128

No. of address: -25

Network Address: - 192.168.12.0

Wildcard: - 0.0.0.127

**Subnet 2 (64 Host): -**

From: -192.168.12.128 to 192.168.12.191

Subnet mask: -255.255.255.192

No. of address: -26

Network Address: - 192.168.12.128

Wildcard: - 0.0.0.63

**Subnet 3 (128 Host): -**

From: -192.168.12.192 to 192.168.12.223

Subnet mask: -255.255.255.224

No. of address: -27

Network Address: - 192.168.12.192

Wildcard: - 0.0.0.31

**Subnet 4 (4 Host): -**

From: -192.168.12.224 to 192.168.12.227

Subnet mask: -255.255.255.252

No. of address: -30

Network Address: - 192.168.12.224

Wildcard: - 0.0.0.3

**Subnet 5 (4 Host): -**

From: -192.168.12.228 to 192.168.12.231

Subnet mask: -255.255.255.252

No. of address: -3

Network Address: - 192.168.12.228

Wildcard: - 0.0.0.3

**Subnet 6 (4 Host): -**

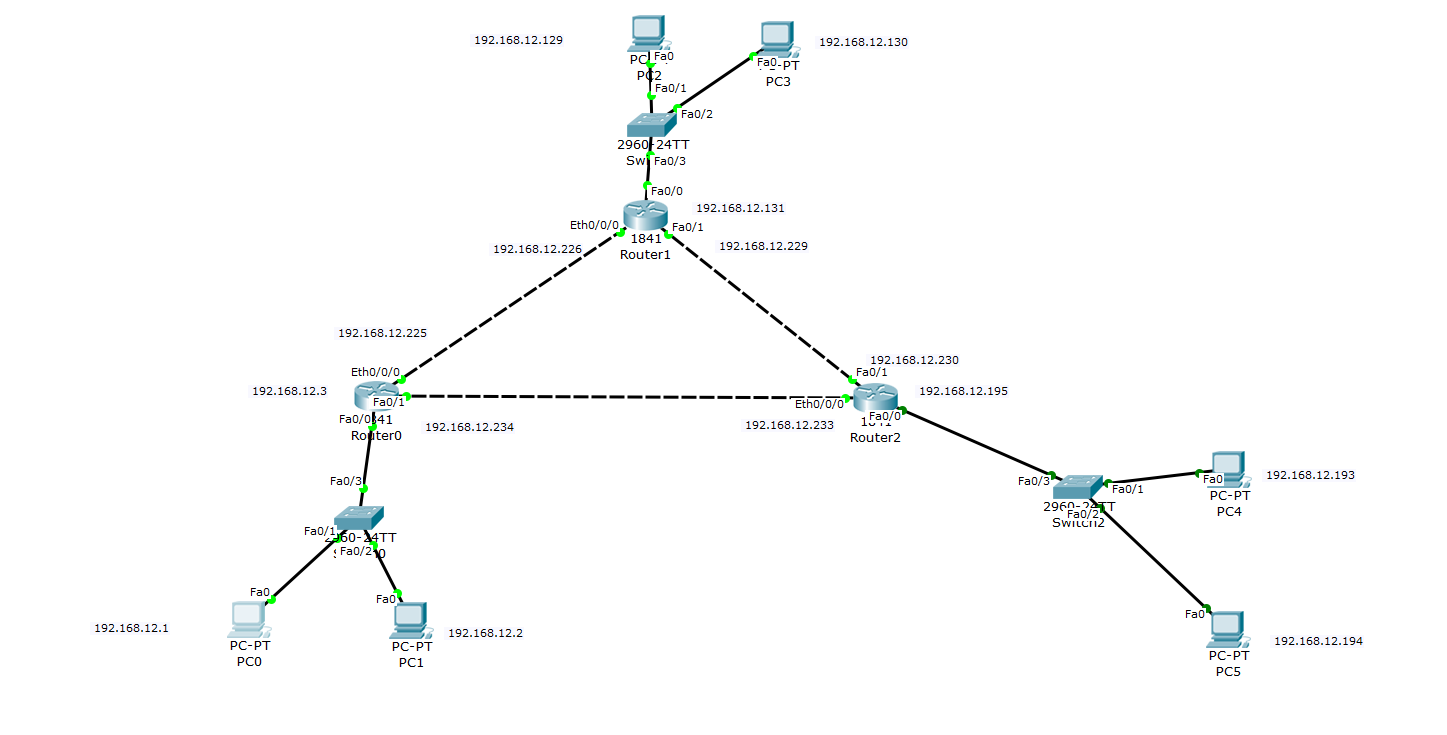
From: -192.168.12.232 to 192.168.12.235

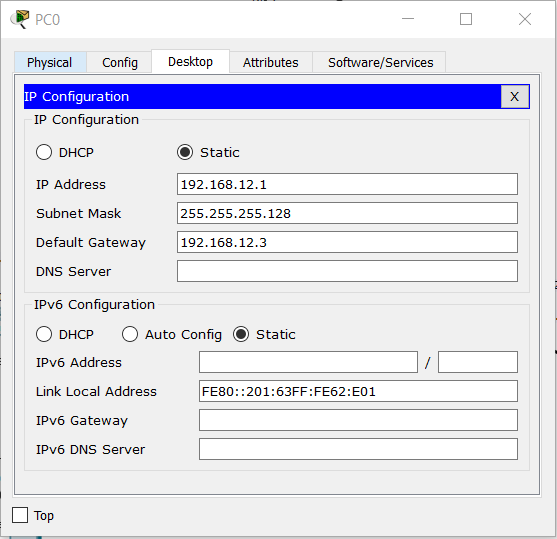
Subnet mask: -255.255.255.252

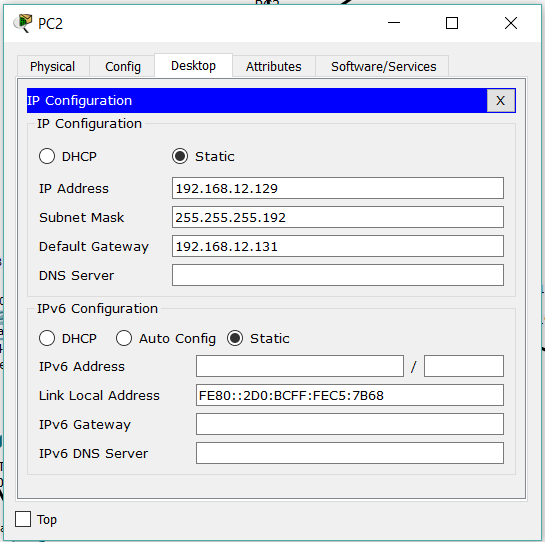
No. of address: -30

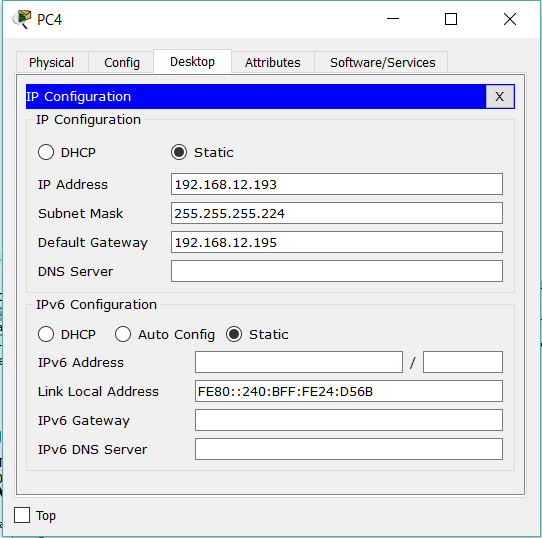
Network Address: - 192.168.12.232

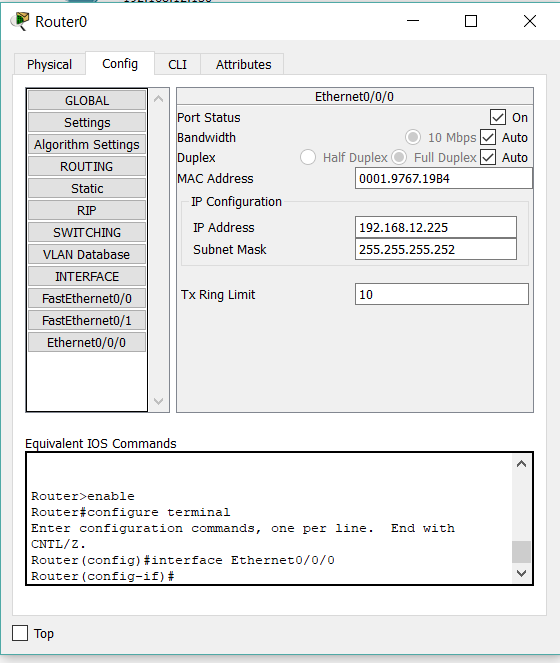
Wildcard: - 0.0.0.3

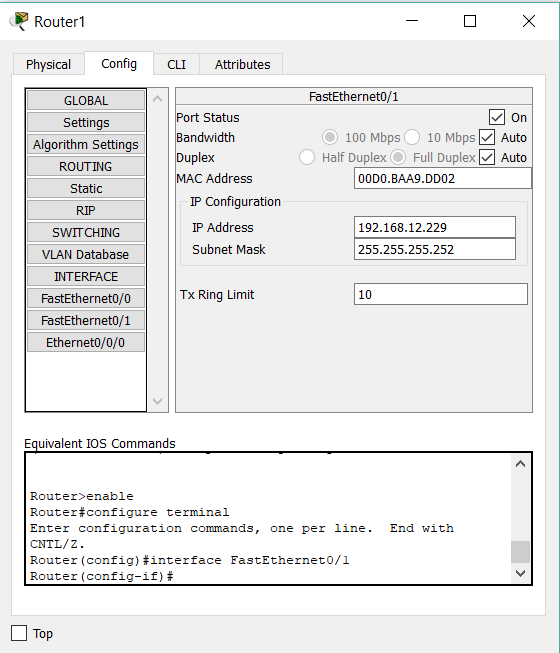
****

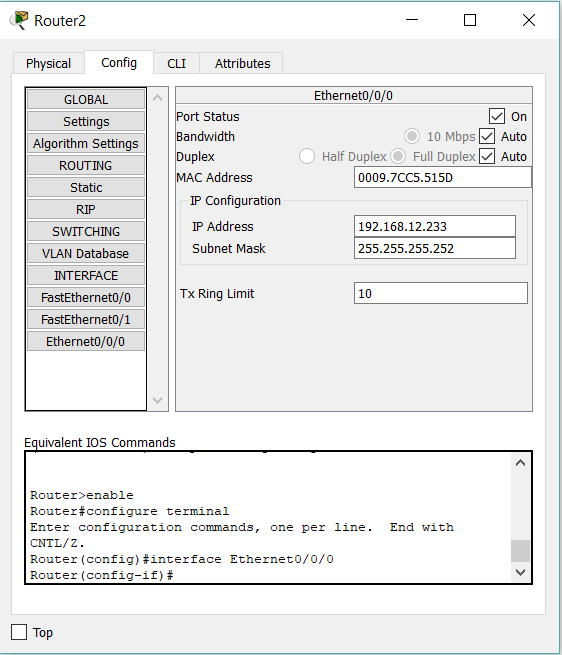
****

****

****

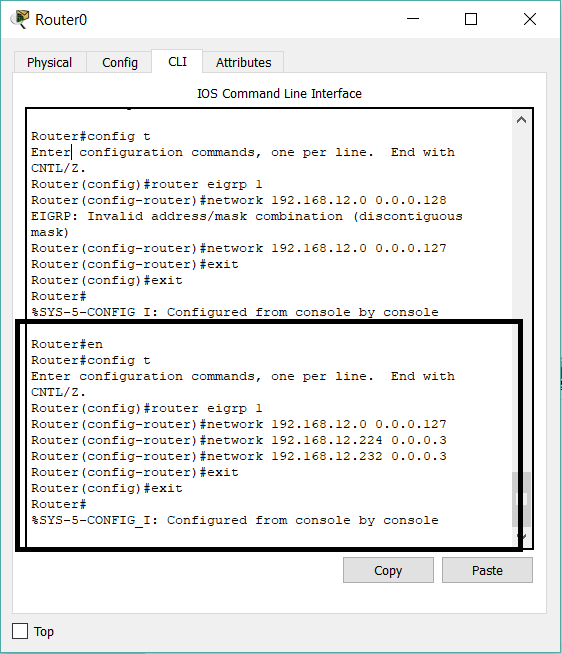
****

****

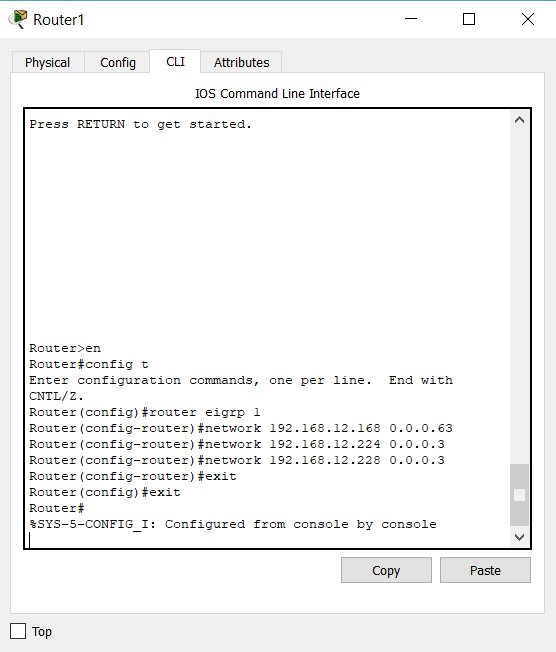
****

* **Routing IP Address using CLI Script: -**

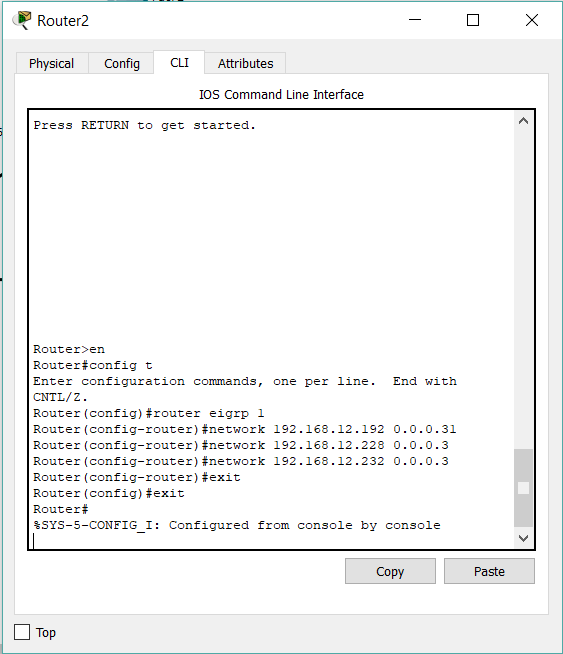
**For Router 0: -**

****

**For Router 1: -**

****

**For Router 2: -**

****

**Conclusion**

In this practical we learn concept of sub-netting.

**PRACTICAL- 4**

**AIM: To configure DHCP and HTTP server.**

**Hardware requirements:** None

**Software requirements:** CISCO packet tracer

**Knowledge requirements:** Basic knowledge about addresses, switch, routers, subnet mask.

**Theory :**

**What is DHCP?**

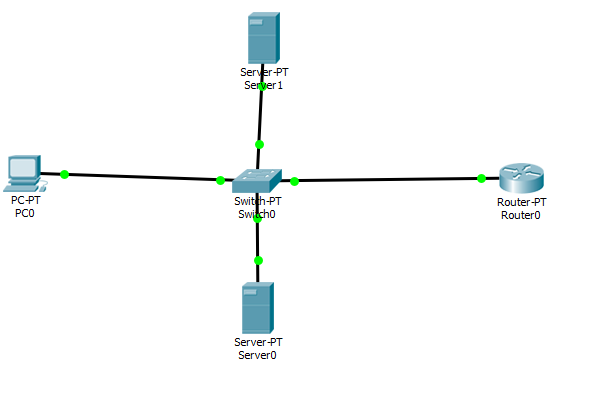
Dynamic Host Configuration Protocol (DHCP) is a network protocol that enables a server to automatically assign an IP address to a computer from a defined range of numbers (i.e., a scope) configured for a given network.

**How DHCP assigns IP addresses**

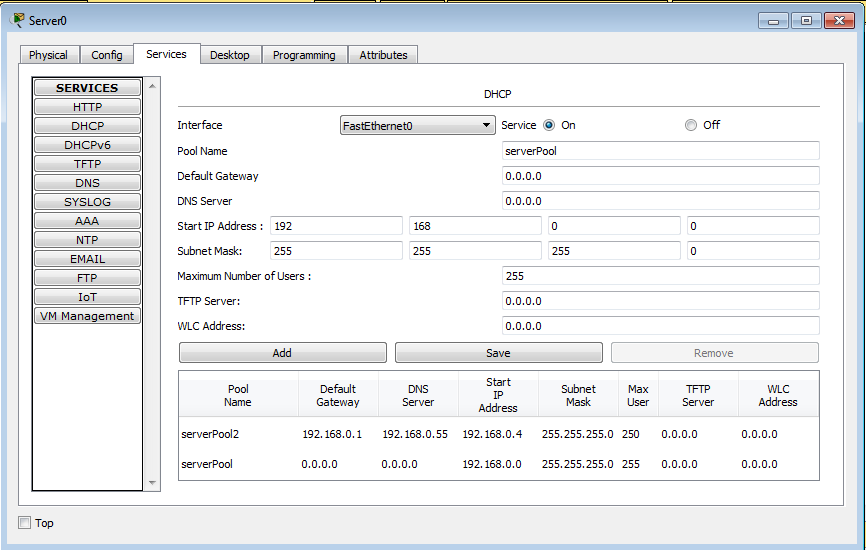
DHCP assigns an IP address when a system is started, for example:

1. A user turns on a computer with a DHCP client.
2. The client computer sends a broadcast request (called a DISCOVER or DHCPDISCOVER), looking for a DHCP server to answer.
3. The router directs the DISCOVER packet to the correct DHCP server.
4. The server receives the DISCOVER packet. Based on availability and usage policies set on the server, the server determines an appropriate address (if any) to give to the client. The server then temporarily reserves that address for the client and sends back to the client an OFFER (or DHCPOFFER) packet, with that address information. The server also configures the client's DNS servers, WINS servers, NTP servers, and sometimes other services as well.
5. The client sends a REQUEST (or DHCPREQUEST) packet, letting the server know that it intends to use the address.
6. The server sends an ACK (or DHCPACK) packet, confirming that the client has a been given a lease on the address for a server-specified period of time.

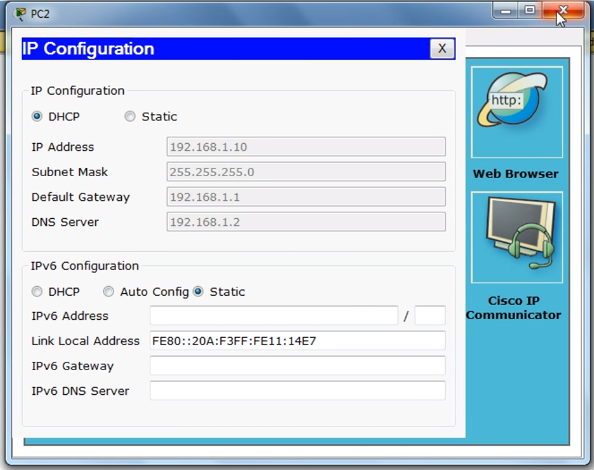
**Step 1** Take DHCP router and required Pc and switch.



**Step 2** Give IP address to all server and PC and switch.



**When we click on DHCP**



**Conclusion**

In this practical we are studied about DHCP server and create configure DHCP.

**PRACTICAL-5**

**AIM: To configure DNS server.**

**Hardware requirements:** None

**Software requirements:** CISCO packet tracer

**Knowledge requirements:** Basic knowledge about addresses, switch, routers, subnet mask.

**Theory :**

## **What does DNS Server mean?**

## A DNS server is a type of name server that manages, maintains and processes Internet domain names and their associated records. In other words, a DNS server is the primary component that implements the DNS (Domain Name System) protocol and provisions domain name resolution services to Web hosts and clients on an IP-based network.

## **Techopedia explains DNS Server**

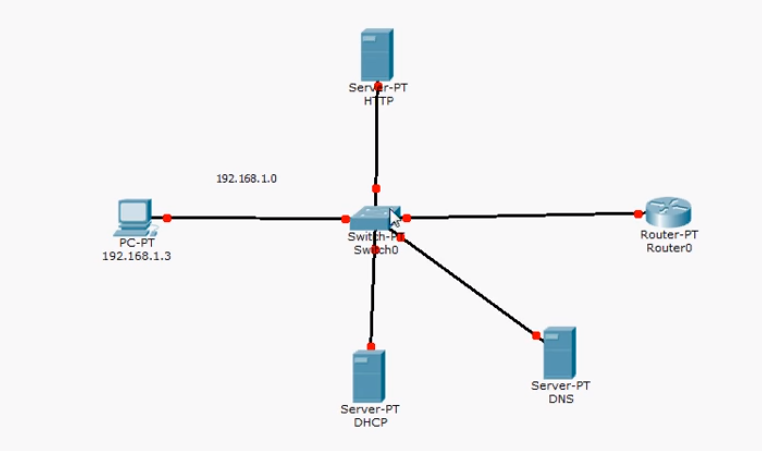
Primarily designed to locate and deliver websites to end users over the Internet or a private network, a DNS server is developed on typical hardware but runs specialized DNS software. It is always connected to the Internet or a network.

A DNS server stores a database of different domain names, network names, Internet hosts, DNS records and other related data. The most basic function of a DNS server is to translate a domain name into its respective IP address. During a domain name resolution query, DNS records are searched, and if found, the domain name record is returned. If the domain name is not registered or added to that DNS server, the query is then passed to other DNS servers until the domain name record is found.

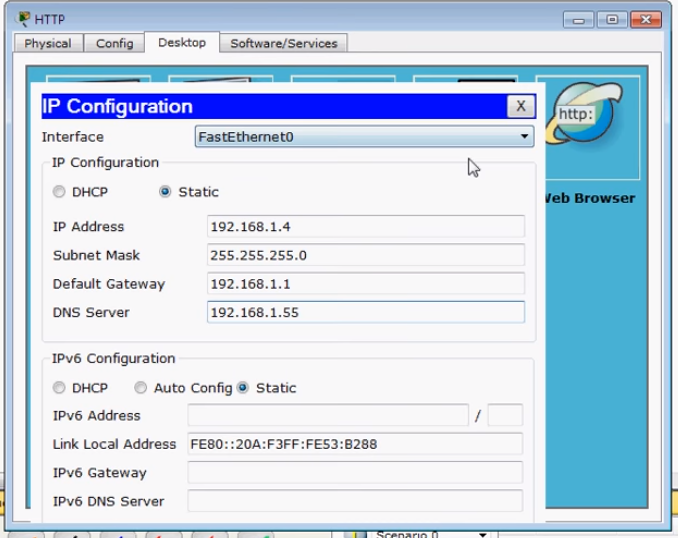
DNS server software comes in dozens, if not hundreds of flavors. The bestknown version is BIND, which is free and distributed with Linux/Unix systems. On Microsoft systems, the Microsoft DNS is bundled as part of many Windows Server releases.

**Screenshot**

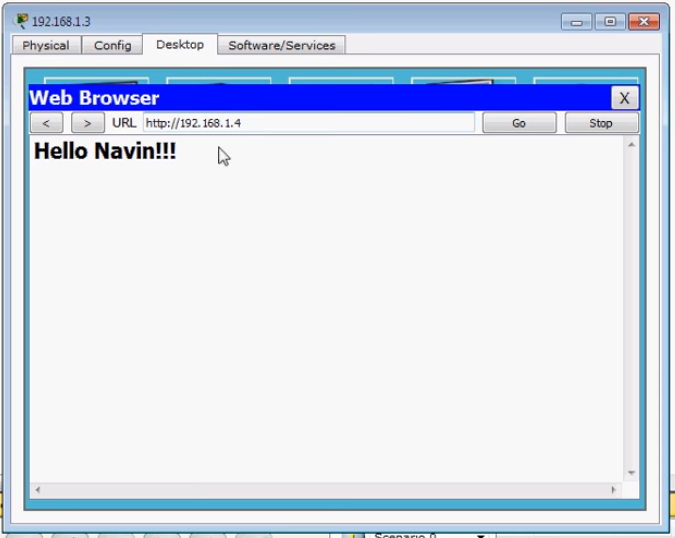
**Step 1 Create This Topology**



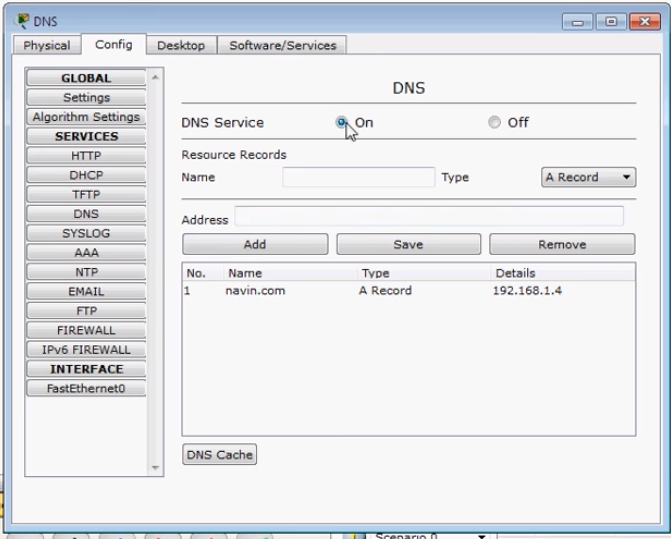
**Step 2 Give to IP Address**



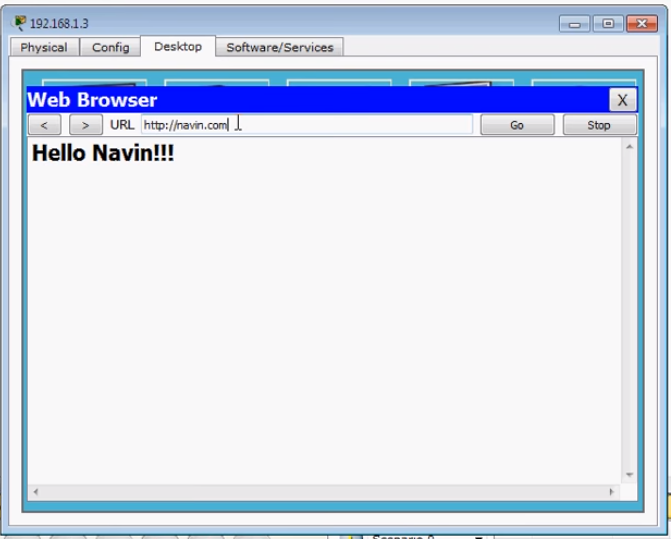
**DNS Server Response by IP Address**



**Add Name of Domain**



**Response by Name**



**Conclusion**

In this practical we are studied What is DNS server and how to configure DNS server.

**PRACTICAL-6**

**AIM: Create a network topology of CE department.**

**Hardware Requirements :** pc / laptop

**Software Requirements :** cisco packet tracer

**Theory :**

**Packet Tracer**

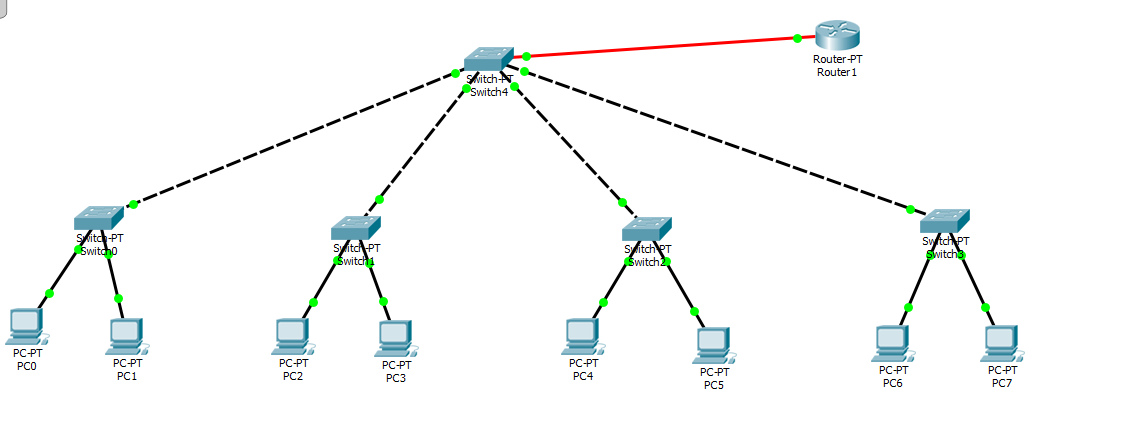
* Packet Tracer is a cross-platform visual simulation tool designed by Cisco Systems that allows users to create network topologies and imitate modern computer networks.
* The software allows users to simulate the configuration of Cisco routers and switches using a simulated command line interface.
* Packet Tracer makes use of a drag and drop user interface, allowing users to add and remove simulated network devices as they see fit.
* The software is mainly focused towards Certified Cisco Network Associate Academy students as an educational tool for helping them learn fundamental CCNA concepts.

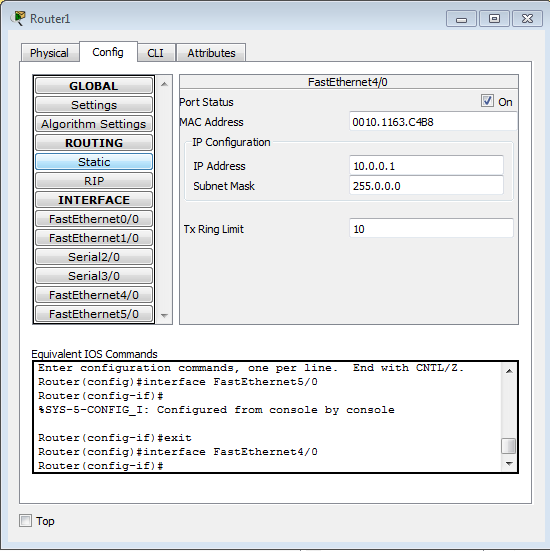
**RIP (Routing Information Protocol)**

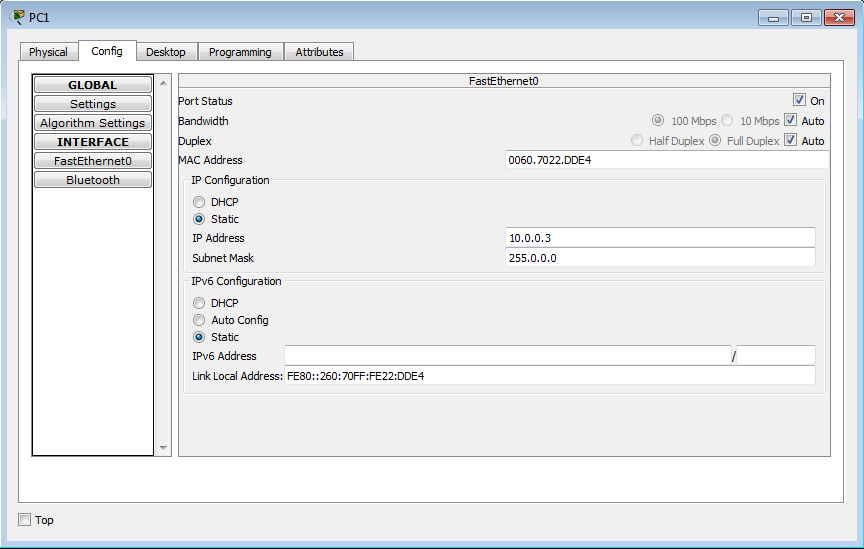
* The Routing Information Protocol (RIP) is one of the oldest distance-vector routing protocols which employ the hop count as a routing metric.
* RIP prevents routing loops by implementing a limit on the number of hops allowed in a path from source to destination.
* The largest number of hops allowed for RIP is 15, which limits the size of networks that RIP can support.
* RIP implements the split horizon, route poisoning and hold-down mechanisms to prevent

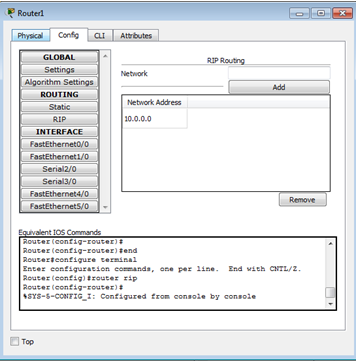
incorrect routing information from being propagated.

**Implementation :**

****

****

****

****

**Conclusion**

In this practical we learn how to implement network topology of any department.

**PRACTICAL-7**

**AIM: Analyze TCP window performance using wireshark.**

**Hardware requirements:** NA.

**Software requirements:** Wireshark

**Knowledge requirements:** Basic knowledge about Wireshark

Theory: Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options.

Wireshark lets the user put network interface controllers into promiscuous mode (if supported by the network interface controller), so they can see all the traffic visible on that interface including unicast traffic not sent to that network interface controller's MAC address. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic. Port mirroring or various network taps extend capture to any point on the network. Simple passive taps are extremely resistant to tampering.

On GNU/Linux, BSD, and macOS, with libpcap 1.0.0 or later, Wireshark 1.4 and later can also put wireless network interface controllers into monitor mode.

If a remote machine captures packets and sends the captured packets to a machine running Wireshark using the TZSP protocol or the protocol used by Omni Peek, Wireshark dissects those packets, so it can analyze packets captured on a remote machine at the time that they are captured.

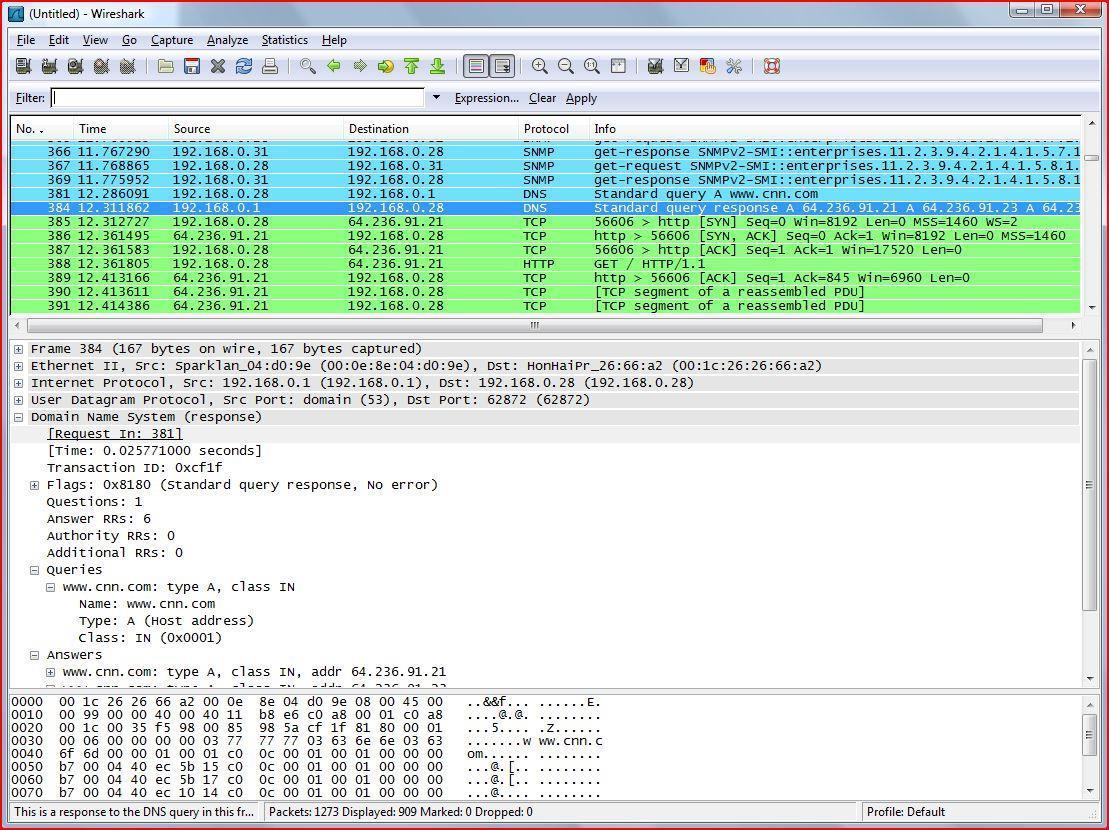
Features:

* Wireshark is a data capturing program that "understands" the structure (encapsulation) of different networking protocols. It can parse and display the fields, along with their meanings as specified by different networking protocols. Wireshark uses pcap to capture packets, so it can only capture packets on the types of networks that pcap supports.
* Data can be captured "from the wire" from a live network connection or read from a file of already-captured packets.
* Live data can be read from different types of networks, including Ethernet, IEEE 802.11, PPP, and loopback.
* Captured network data can be browsed via a GUI, or via the terminal (command line) version of the utility, TShark.
* Captured files can be programmatically edited or converted via command-line switches to the "editcap" program.
* Data display can be refined using a display filter.
* Plug-ins can be created for dissecting new protocols.
* VoIP calls in the captured traffic can be detected. If encoded in a compatible encoding, the media flow can even be played.
* Raw USB traffic can be captured.

When Wire shark’s install is started or the portable version is run for the first time, you will be asked to install the included ‘Windows Packet Capture’ (WinPcap) library and driver as well. This is obviously a drawback if you want a truly portable application but unfortunately, is required to capture the network packets. If WinPcap is already detected on the system, this part will be automatically skipped. Once installed under Windows or Linux, the default interface is pretty much the same.

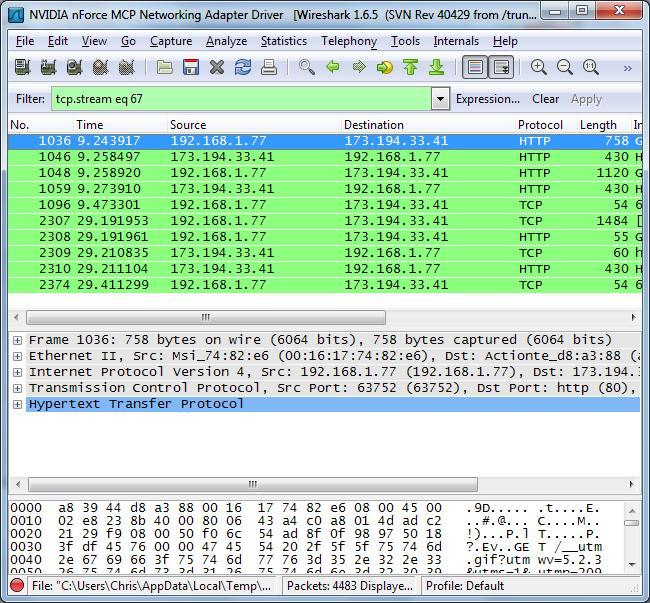
Implementation:

First of all open wireshark tool and then click on LAN, Ethernet or Wi-Fi as per your wish. Click on it and go to the option for starting the scan after it wait for some time. And then it will display something like this



After it there has been mentioned the time, source, destination and the protocols in which they work and it’s some general information.

Click on the any of the option provided here clicking on the Domain name system and so on you can monitor and capture the packets of that network



Here you can see that here there is a possibility that you can only capture packets on the Http or Tcp protocol you cannot capture packets on the Https.After wards you can save It to the .pcap file for further purpose.

The best one you can try on the http website click on login in the website and the start scan you will see that website in it after you click on it will show you the username and password of that site that you have been login.

**Conclusion**

From this Practical I understood about the wireshark tool how tocapture packets etc.

**PRACTICAL-8**

**AIM: Find and Compare latency using Wireshark.**

**Hardware requirements:** NA.

**Software requirements:** Wireshark

**Knowledge requirements:** Basic knowledge about Wireshark

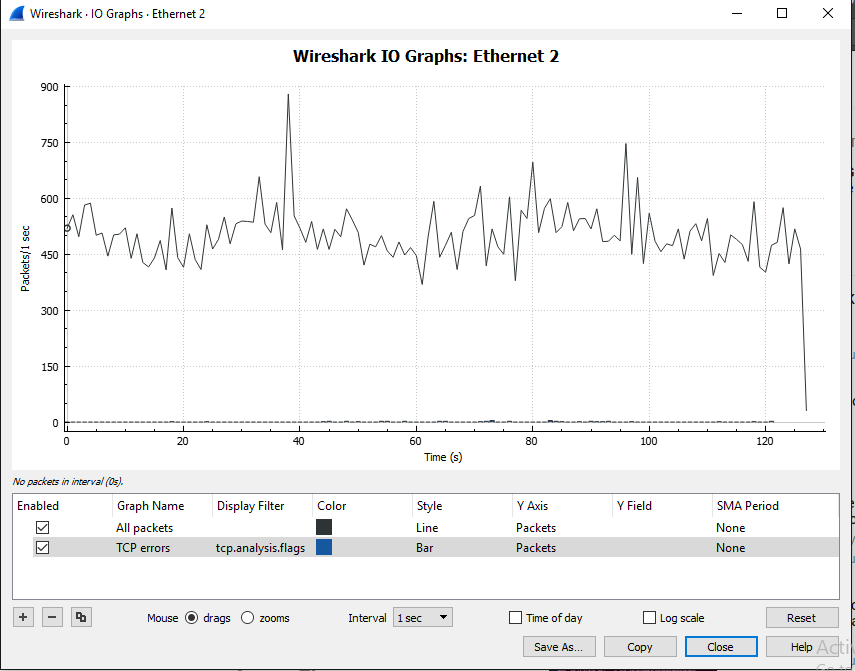
**Theory:**

The first way to show IO information is the Wireshark IO Graph, in Statistics > IO Graph.The IO graph shows a time series of network traffic, which you can change the resolution and scale of, and filter.User configurable graph of the captured network packets.We can define up to five differently colored graphs.The user can configure the following things:

* Graphs
  + Graph 1-5: enable the specific graph 1-5 (only graph 1 is enabled by default)
  + Color: the color of the graph (cannot be changed)
  + Filter: a display filter for this graph (only the packets that pass this filter will be taken into account for this graph)
  + Style: the style of the graph (Line/Impulse/FBar/Dot)
* X Axis
  + Tick interval: an interval in x direction lasts (10/1 minutes or 10/1/0.1/0.01/0.001 seconds)
  + Pixels per tick: use 10/5/2/1 pixels per tick interval
  + View as time of day: option to view x direction labels as time of day instead of seconds or minutes since beginning of capture
* Y Axis
  + Unit: the unit for the y direction (Packets/Tick, Bytes/Tick, Bits/Tick, Advanced…​) [XXX - describe the Advanced feature.]
  + Scale: the scale for the y unit (Logarithmic,Auto,10,20,50,100,200,500,…​)

The Save button will save the currently displayed portion of the graph as one of various file formats.

The Copy button will copy values from selected graphs to the clipboard in CSV (Comma Separated Values) format.



**Conclusion**

Studied to compare the overall traffic in different capture files using wireshark.

**PRACTICAL-9**

**AIM: Create a simple point-to-point link between two nodes and**

**echo a single packet between the nodes using NS-3.**

**Hardware requirements:** NA.

**Software requirements:** Ubuntu with NS-3

**Knowledge requirements:** Basic knowledge about NS-3

**Script:**

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/netanim-module.h"

#include "ns3/mobility-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("FirstScriptExample");

int

main (int argc, char \*argv[]){

CommandLinecmd;

cmd.Parse (argc, argv);

Time::SetResolution (Time::NS);

LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

NodeContainer nodes;

nodes.Create (2);

PointToPointHelperpointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer devices;

devices = pointToPoint.Install (nodes);

InternetStackHelper stack;

stack.Install (nodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interfaces = address.Assign (devices);

UdpEchoServerHelperechoServer (9);

ApplicationContainerserverApps = echoServer.Install (nodes.Get (1));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelperechoClient (interfaces.GetAddress (1), 9);

echoClient.SetAttribute ("MaxPackets", UintegerValue (1));

echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainerclientApps = echoClient.Install (nodes.Get (0));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (10.0));

MobilityHelper mobility;

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

mobility.Install (nodes);

AnimationInterfaceanim ("first.xml");

AnimationInterface::SetConstantPosition (nodes.Get (0), 10, 25);

AnimationInterface::SetConstantPosition (nodes.Get (1), 40, 25);

anim.EnablePacketMetadata (true);

pointToPoint.EnablePcapAll ("first");

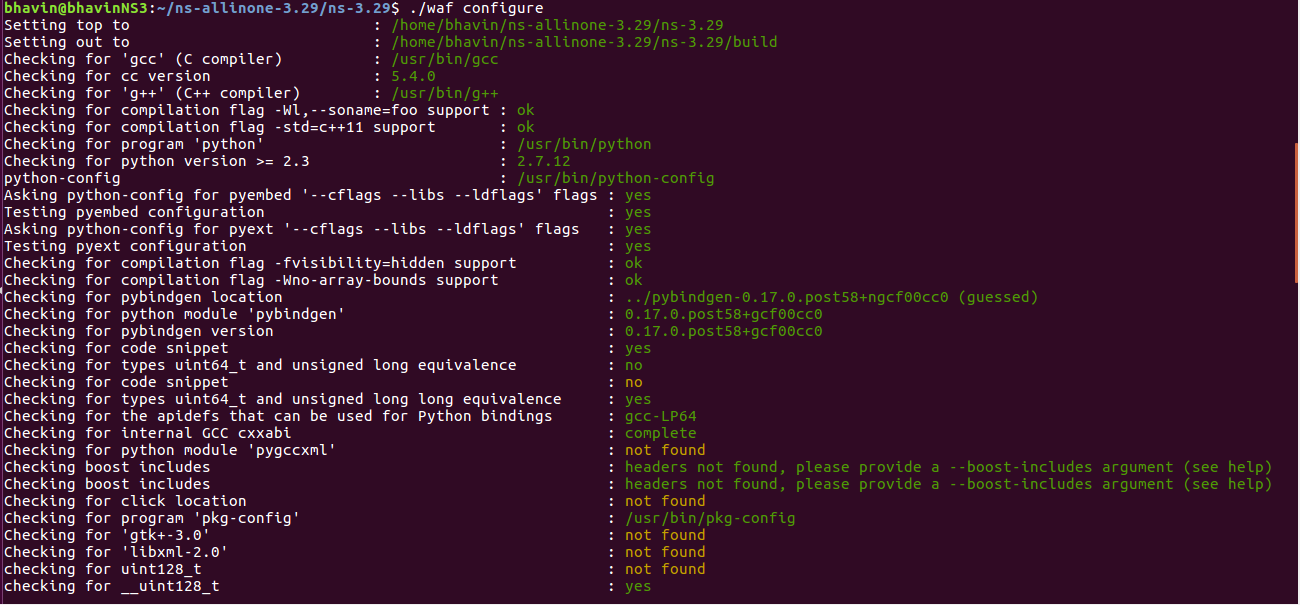
Simulator::Run ();

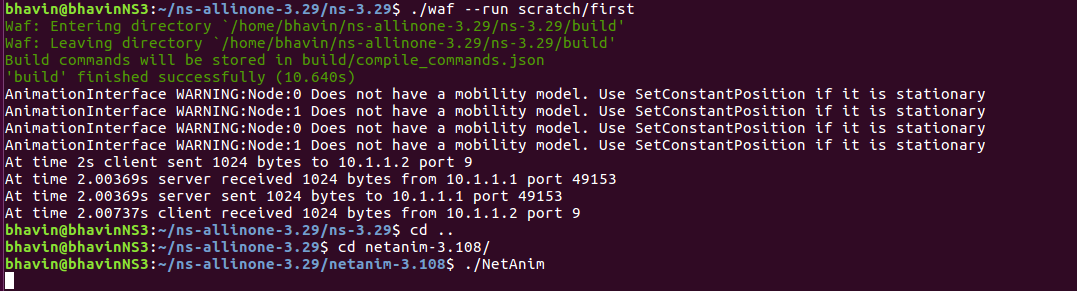
Simulator::Destroy ();

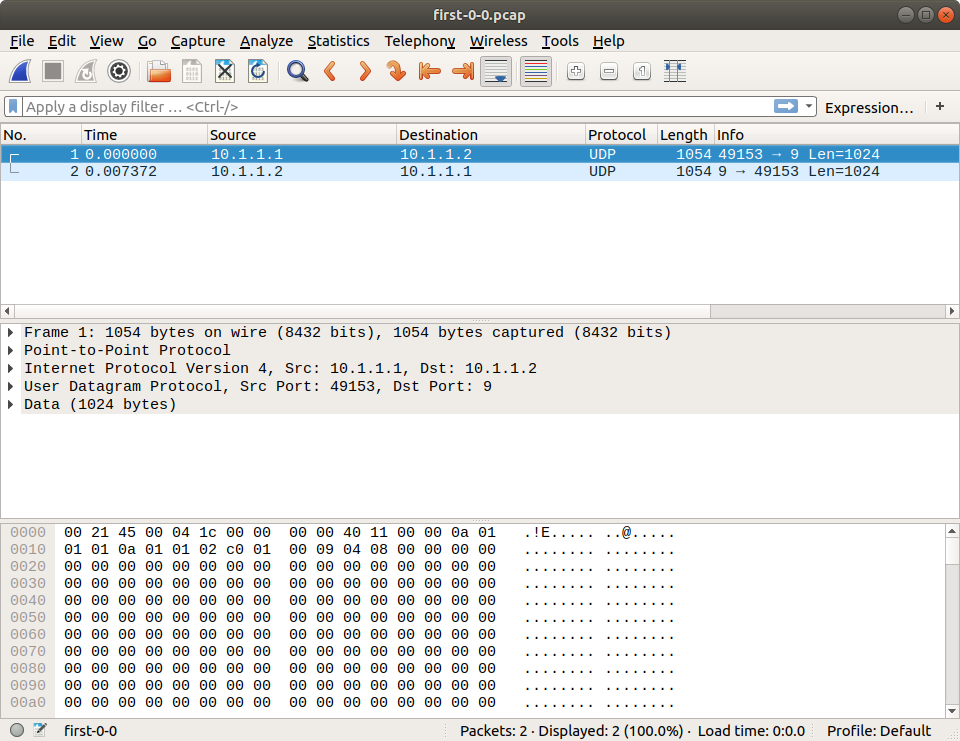
return 0;

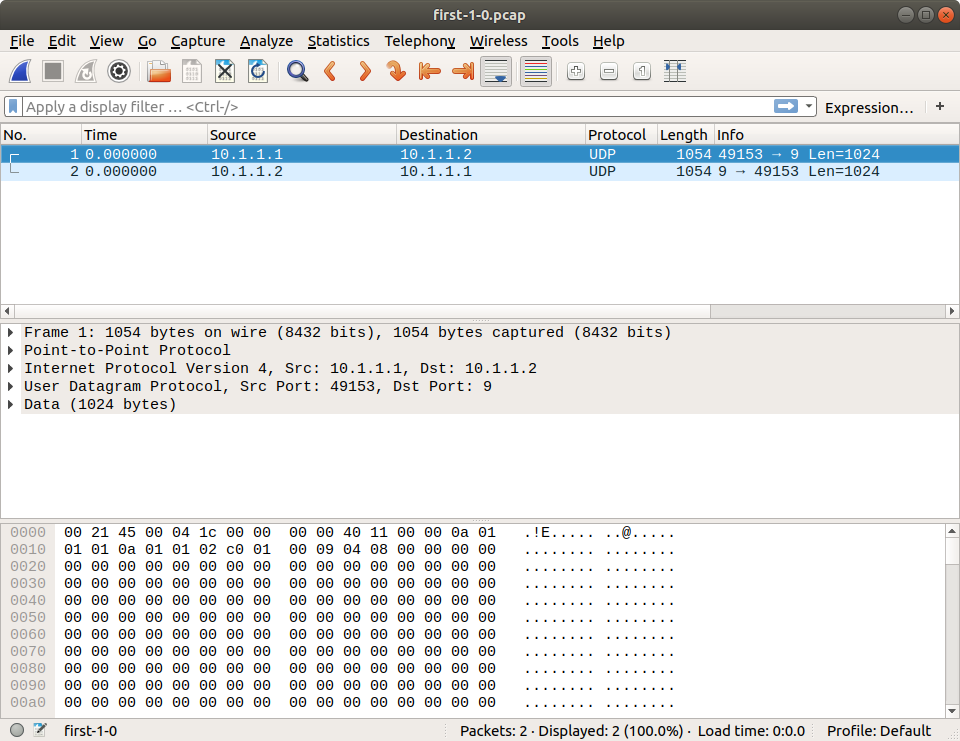
}

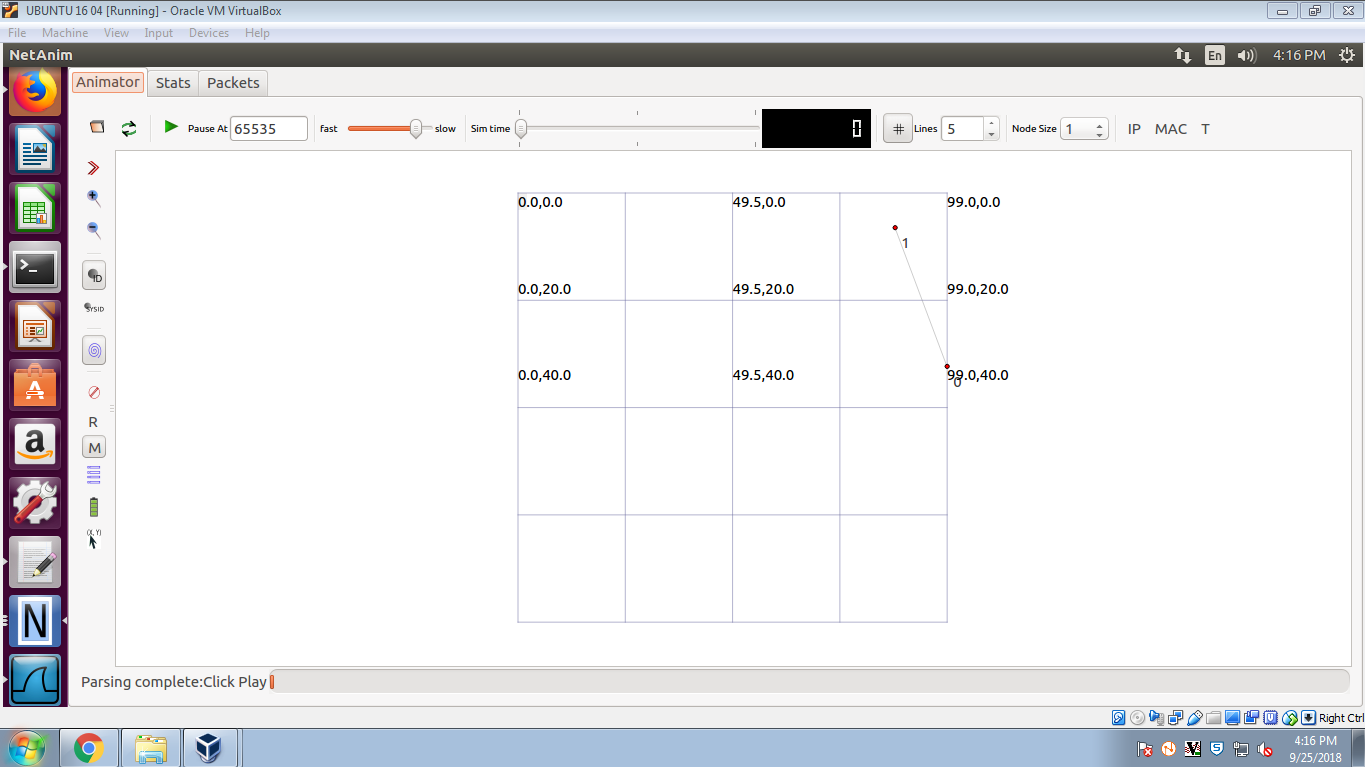
**Output:**











**Conclusion**

Studied how to create a point-to-point link using NS-3 and analyzed it using wireshark.

**PRACTICAL-10**

**AIM: Add a CSMA network to the point-to-point simulation of**

**practical-9.**

**Hardware requirements:** NA.

**Software requirements:** Ubuntu with NS-3

**Knowledge requirements:** Basic knowledge about NS-3

**Script :**

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/netanim-module.h"

#include "ns3/mobility-module.h"

using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("Practical6Example");

int main (int argc, char \*argv[]) {

CommandLinecmd;

cmd.Parse (argc, argv);

Time::SetResolution (Time::NS);

LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);

LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);

NodeContainer nodes;

nodes.Create (3);

PointToPointHelperpointToPoint;

pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));

pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainerdevice,device1;

device = pointToPoint.Install (nodes.Get(0),nodes.Get(1));

device1 = pointToPoint.Install (nodes.Get(2),nodes.Get(1));

InternetStackHelper stack;

stack.Install (nodes);

Ipv4AddressHelper address;

address.SetBase ("10.1.1.0", "255.255.255.0");

Ipv4InterfaceContainer interface = address.Assign (device);

Ipv4InterfaceContainer interface1 = address.Assign (device1);

UdpEchoServerHelperechoServer (9);

ApplicationContainerserverApps = echoServer.Install (nodes.Get (1));

serverApps.Start (Seconds (1.0));

serverApps.Stop (Seconds (10.0));

UdpEchoClientHelperechoClient (interface.GetAddress (1), 9);

echoClient.SetAttribute ("MaxPackets", UintegerValue (1));

echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));

echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainerclientApps = echoClient.Install (nodes.Get (0));

clientApps.Start (Seconds (2.0));

clientApps.Stop (Seconds (5.0));

ApplicationContainer clientApps1 = echoClient.Install (nodes.Get (2));

clientApps1.Start (Seconds (5.0));

clientApps1.Stop (Seconds (10.0));

MobilityHelper mobility;

mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

mobility.Install (nodes);

AnimationInterfaceanim ("prac6.xml");

AnimationInterface::SetConstantPosition (nodes.Get (0), 10, 25);

AnimationInterface::SetConstantPosition (nodes.Get (1), 30, 25);

AnimationInterface::SetConstantPosition (nodes.Get (2), 50, 25);

anim.EnablePacketMetadata (true);

pointToPoint.EnablePcapAll ("prac6");

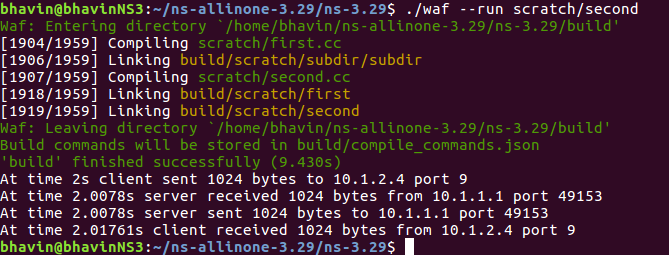
Simulator::Run ();

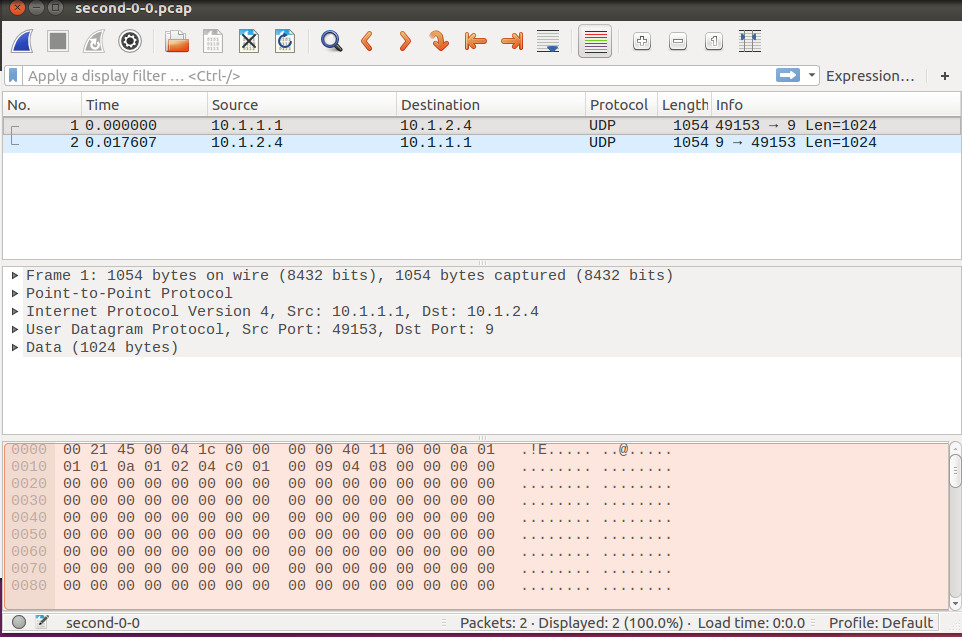
Simulator::Destroy ();

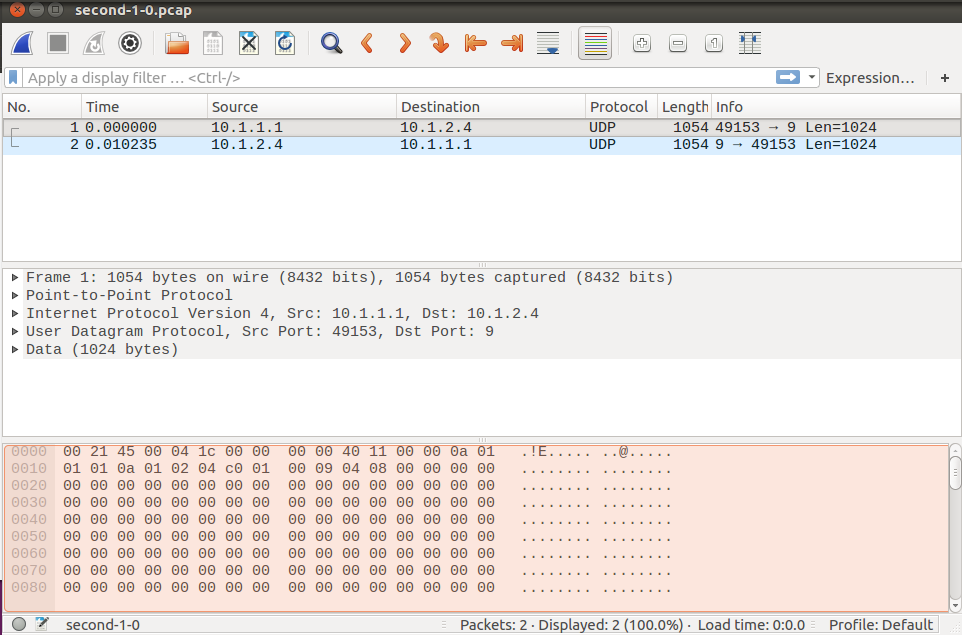
return 0;

}

Output:







**Conclusion**

Studied how to create a point-to-point link using NS-3 and analyzed it using wireshark.

**PRACTICAL-11**

**AIM: Create Wireless Network using Wi-Fi helper object in**

**NS-3.**

**Hardware requirements:** NA.

**Software requirements:** Ubuntu with NS-3

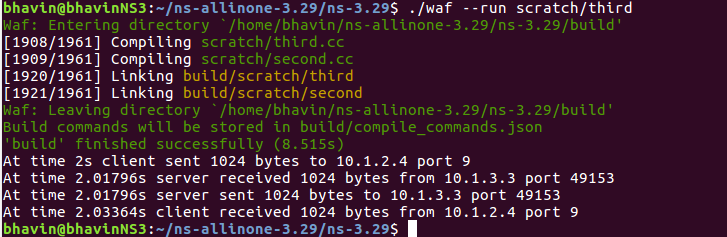
**Knowledge requirements:** Basic knowledge about NS-3

**Script:**

#include "ns3/core-module.h"  
#include "ns3/point-to-point-module.h"  
#include "ns3/network-module.h"  
#include "ns3/applications-module.h"  
#include "ns3/mobility-module.h"  
#include "ns3/csma-module.h"  
#include "ns3/internet-module.h"  
#include "ns3/yans-wifi-helper.h"  
#include "ns3/ssid.h"  
using namespace ns3;

NS\_LOG\_COMPONENT\_DEFINE ("ThirdScriptExample");  
int   
main (int argc, char \*argv[]) {  
  bool verbose = true;  
  uint32\_t nCsma = 3;  
  uint32\_t nWifi = 3;  
  bool tracing = false;  
  CommandLine cmd;  
  cmd.AddValue ("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);  
  cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);  
  cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);  
  cmd.AddValue ("tracing", "Enable pcap tracing", tracing);  
  cmd.Parse (argc,argv);  
   if (nWifi > 18)     {  
      std::cout << "nWifi should be 18 or less; otherwise grid layout exceeds the bounding box" << std::endl;  
      return 1;     }  
  if (verbose)     {  
      LogComponentEnable ("UdpEchoClientApplication", LOG\_LEVEL\_INFO);  
      LogComponentEnable ("UdpEchoServerApplication", LOG\_LEVEL\_INFO);     }  
  NodeContainer p2pNodes;  
  p2pNodes.Create (2);  
  PointToPointHelper pointToPoint;  
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));  
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));  
  NetDeviceContainer p2pDevices;  
  p2pDevices = pointToPoint.Install (p2pNodes);  
  NodeContainer csmaNodes;  
  csmaNodes.Add (p2pNodes.Get (1));  
  csmaNodes.Create (nCsma);  
  CsmaHelper csma;  
  csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));  
  csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));  
  NetDeviceContainer csmaDevices;  
  csmaDevices = csma.Install (csmaNodes);  
  NodeContainer wifiStaNodes;  
  wifiStaNodes.Create (nWifi);  
  NodeContainer wifiApNode = p2pNodes.Get (0);  
  YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();  
  YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();  
  phy.SetChannel (channel.Create ());  
  WifiHelper wifi;  
  wifi.SetRemoteStationManager ("ns3::AarfWifiManager");  
  WifiMacHelper mac;  
  Ssid ssid = Ssid ("ns-3-ssid");  
  mac.SetType ("ns3::StaWifiMac",  
               "Ssid", SsidValue (ssid),  
               "ActiveProbing", BooleanValue (false));  
  NetDeviceContainer staDevices;  
  staDevices = wifi.Install (phy, mac, wifiStaNodes);  
  mac.SetType ("ns3::ApWifiMac",  
               "Ssid", SsidValue (ssid));  
  NetDeviceContainer apDevices;  
  apDevices = wifi.Install (phy, mac, wifiApNode);  
  MobilityHelper mobility;  
  mobility.SetPositionAllocator ("ns3::GridPositionAllocator",  
                                 "MinX", DoubleValue (0.0),  
                                 "MinY", DoubleValue (0.0),  
                                 "DeltaX", DoubleValue (5.0),  
                                 "DeltaY", DoubleValue (10.0),  
                                 "GridWidth", UintegerValue (3),  
                                 "LayoutType", StringValue ("RowFirst"));  
  mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",  
                             "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));  
  mobility.Install (wifiStaNodes);  
  mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");  
  mobility.Install (wifiApNode);  
  InternetStackHelper stack;  
  stack.Install (csmaNodes);  
  stack.Install (wifiApNode);  
  stack.Install (wifiStaNodes);  
  Ipv4AddressHelper address;  
  address.SetBase ("10.1.1.0", "255.255.255.0");  
  Ipv4InterfaceContainer p2pInterfaces;  
  p2pInterfaces = address.Assign (p2pDevices);  
  address.SetBase ("10.1.2.0", "255.255.255.0");  
  Ipv4InterfaceContainer csmaInterfaces;  
  csmaInterfaces = address.Assign (csmaDevices);  
  address.SetBase ("10.1.3.0", "255.255.255.0");  
  address.Assign (staDevices);  
  address.Assign (apDevices);  
  UdpEchoServerHelper echoServer (9);  
  ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));  
  serverApps.Start (Seconds (1.0));  
  serverApps.Stop (Seconds (10.0));  
  UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 9);  
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));  
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));  
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));  
  ApplicationContainer clientApps =   
    echoClient.Install (wifiStaNodes.Get (nWifi - 1));  
  clientApps.Start (Seconds (2.0));  
  clientApps.Stop (Seconds (10.0));  
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();  
  Simulator::Stop (Seconds (10.0));  
  if (tracing == true)     {  
      pointToPoint.EnablePcapAll ("third");  
      phy.EnablePcap ("third", apDevices.Get (0));  
      csma.EnablePcap ("third", csmaDevices.Get (0), true);     }  
  Simulator::Run ();  
  Simulator::Destroy ();  
  return 0;  
}

**Output:**

****

**Conclusion**

Studied how to create a point-to-point link using NS-3 and analyzed it using wireshark.

**PRACTICAL-12**

**AIM: Introduction and demonstration of simulation in Contiki**

**OS with Cooja tool.**

**Hardware Requirement:-** N/A.

**Software Requirement:-** Contiki-OS, Cooja tool

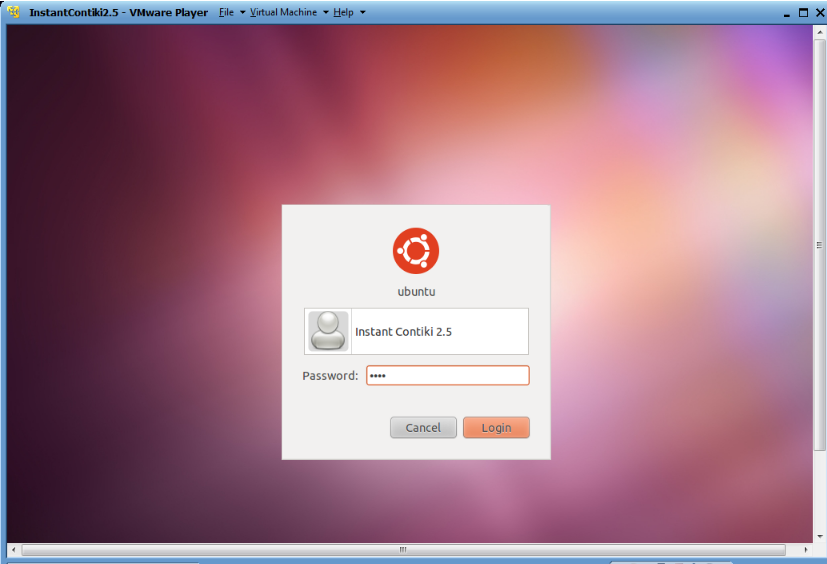
**Theory:-**

**Contiki-OS**

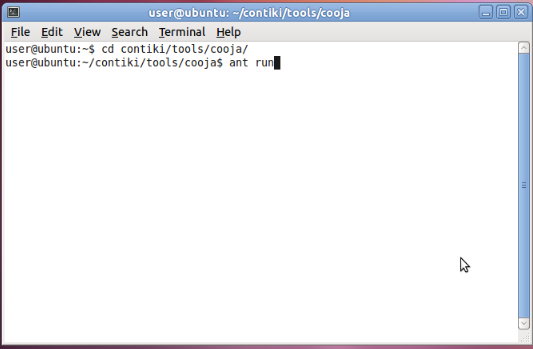
Contiki is an operating system for networked, memory-constrained systems with a focus on low-power wireless Internet of Things devices. Extant uses for Contiki include systems for street lighting, sound monitoring for smart cities, radiation monitoring, and alarms. It is open-source software released under a BSD license.

**Step to simulation in Contiki-OS with Cooja tool:-**

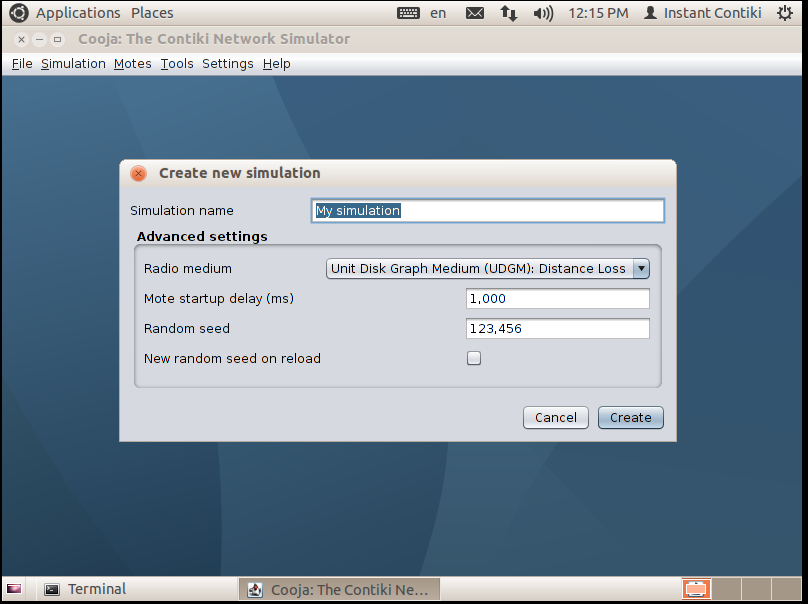
**Step-1:** Start Instant Contiki by running InstantContiki2.6.vmx. Wait for the virtual Ubuntu Linux boot up. Log into Instant Contiki. The password is “user”.



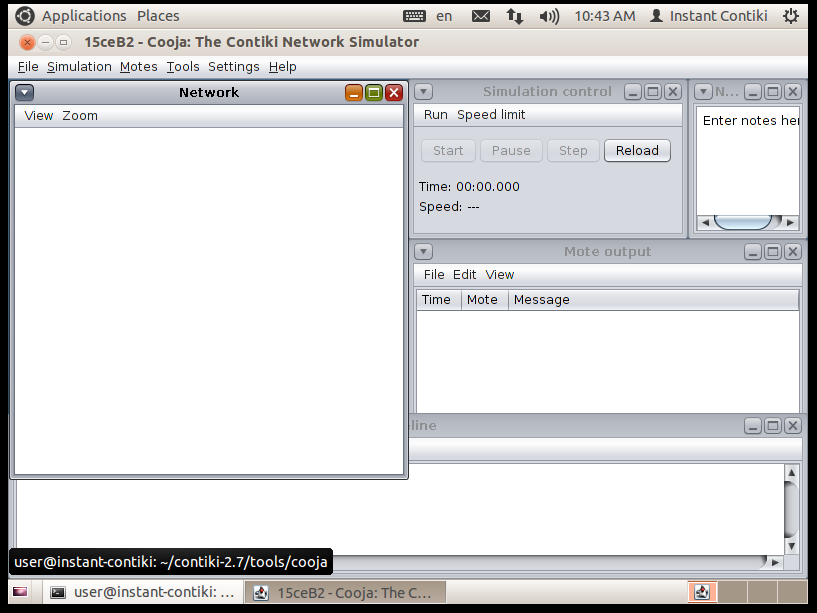
**Step-2:** To start Cooja, first open a terminal window. In the terminal window, go to the Cooja directory. Start Cooja with the command: “ ant run “.

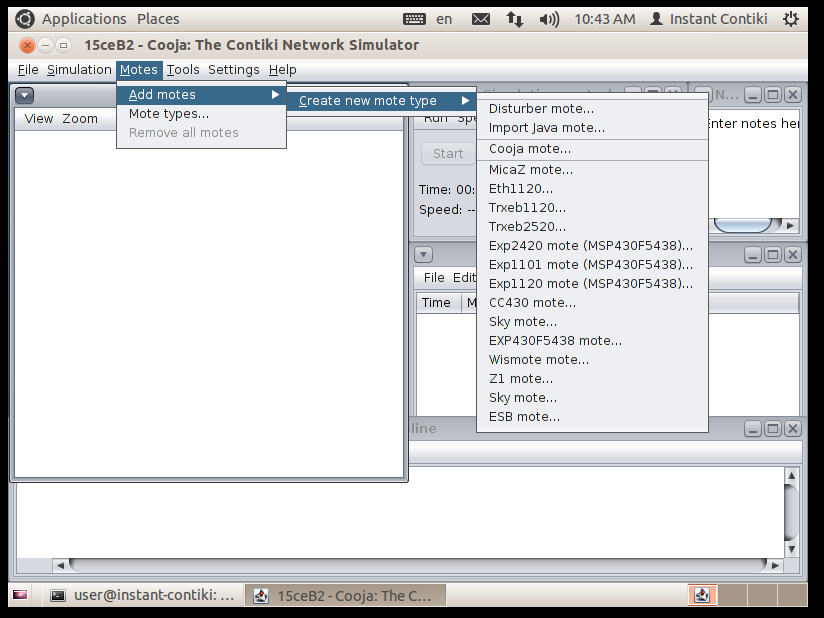


**Step-3:** To create new simulation, click the “File” menu and click “New simulation”. Cooja now opens up the Create new simulation dialog. In this dialog, we may choose to give our simulation a new name. Click the Create button.

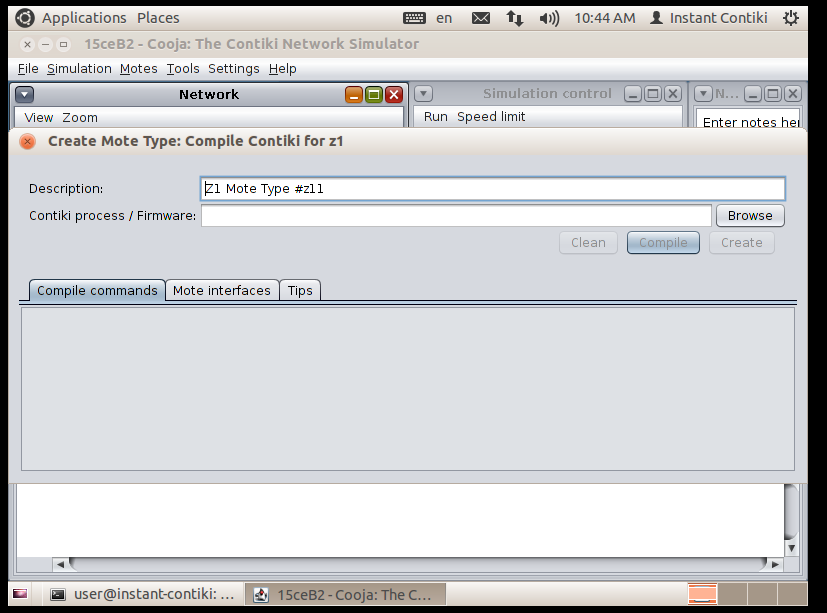


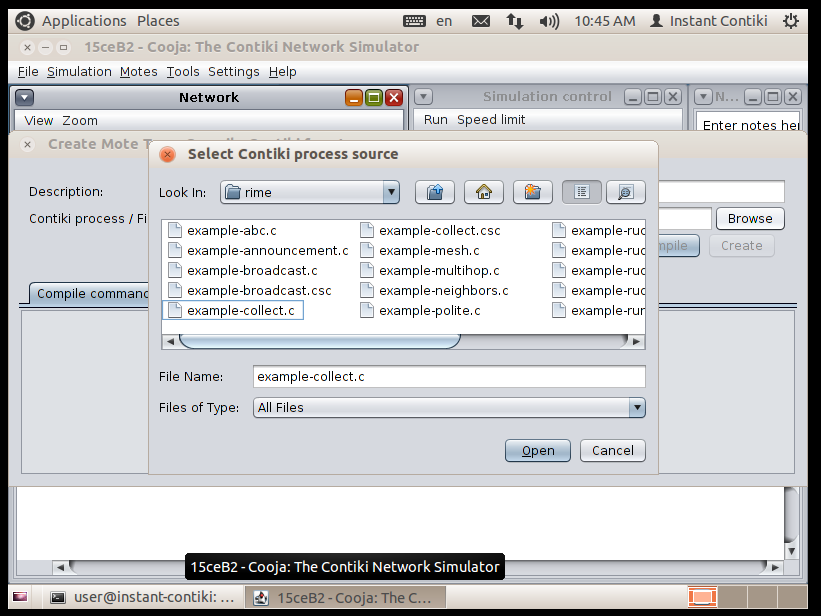
**Step-4:** Cooja brings up the new simulation.



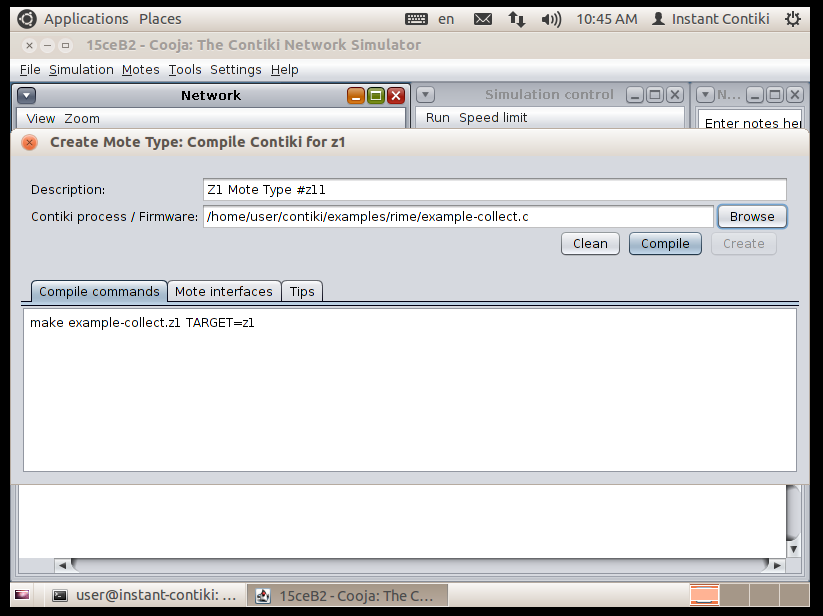
**Step-5:** To add new motes go to the Motes menu, click on Add motes. Click Create new mote type... and select one of the available mote types.

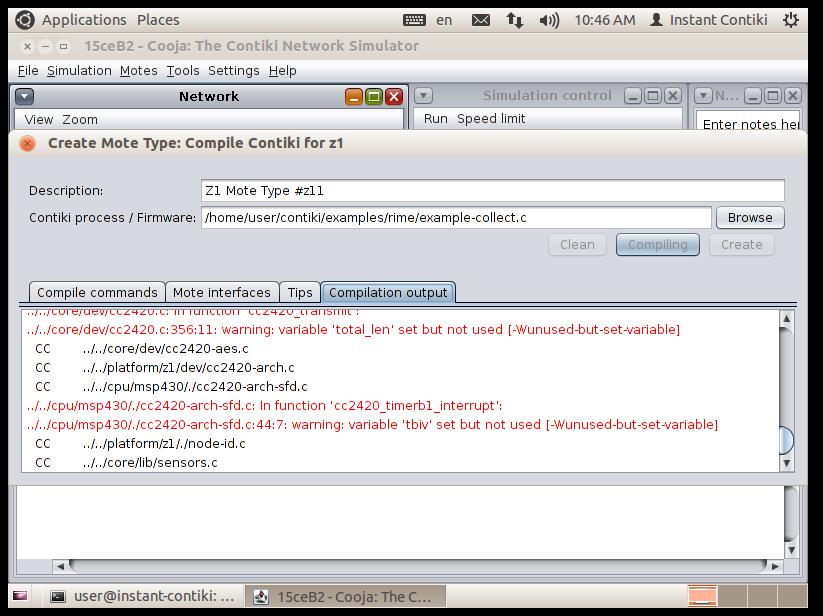
**Step-6:** Cooja opens the Create Mote Type dialog, in which we can choose a name for our mote type as well as the Contiki application that our mote type will run. Click on the Browse... button on the right hand side to choose our Contiki application.



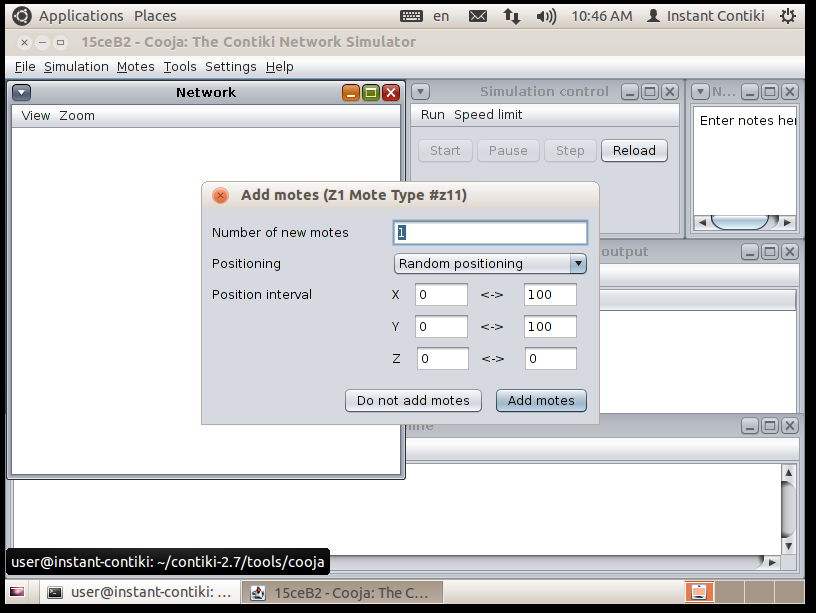


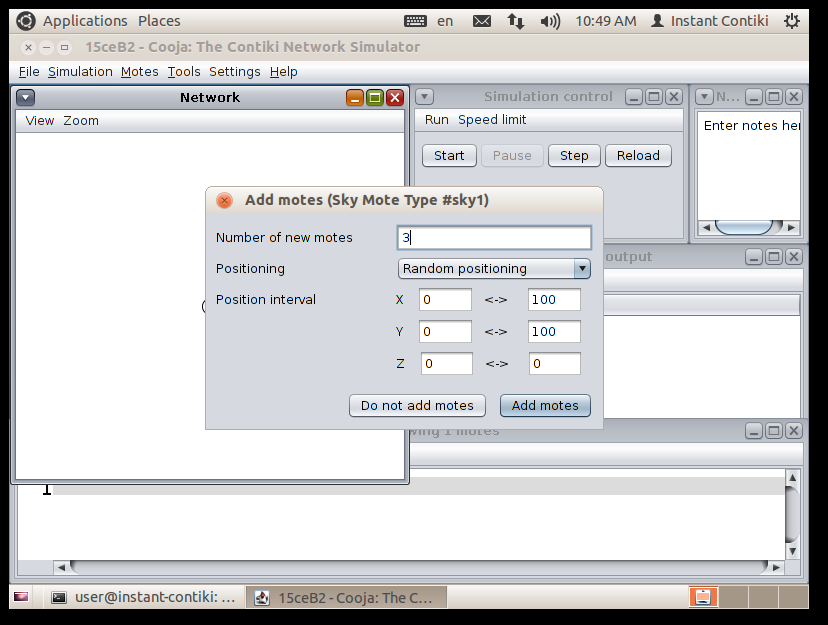
**Step-7:** Now Cooja will verify that the selected Contiki application compiles for the platform that we have selected. Click the Compile button.



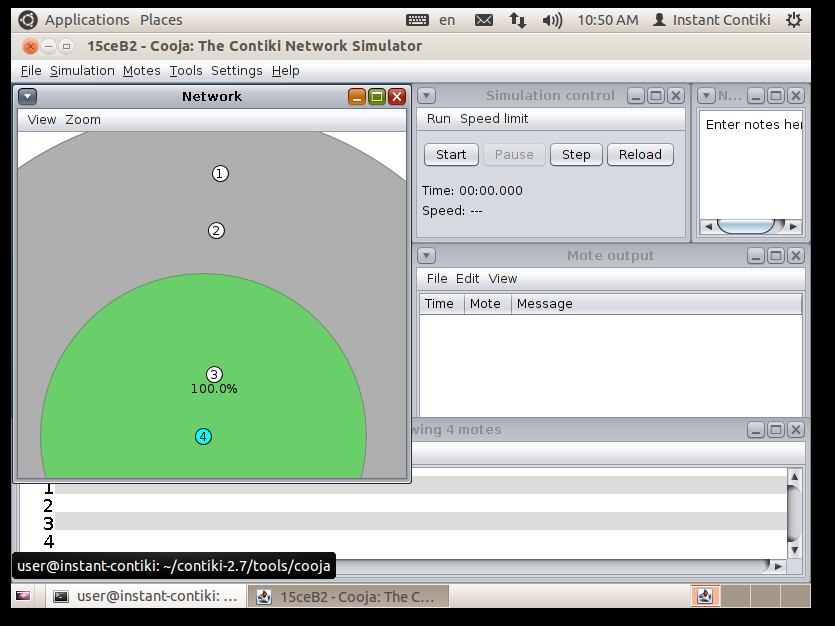


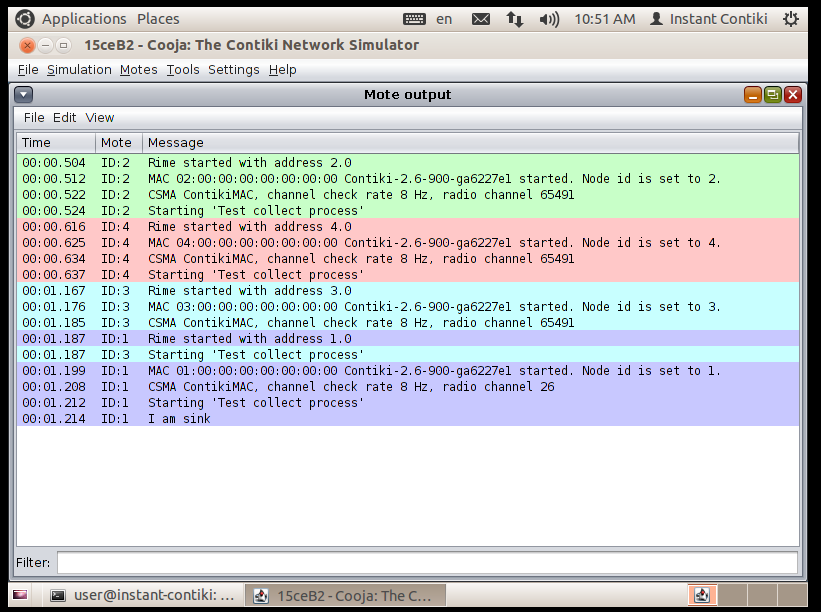
**Step-8:** Cooja will now ask us if we want to add motes from the newly created mote type to the simulation.





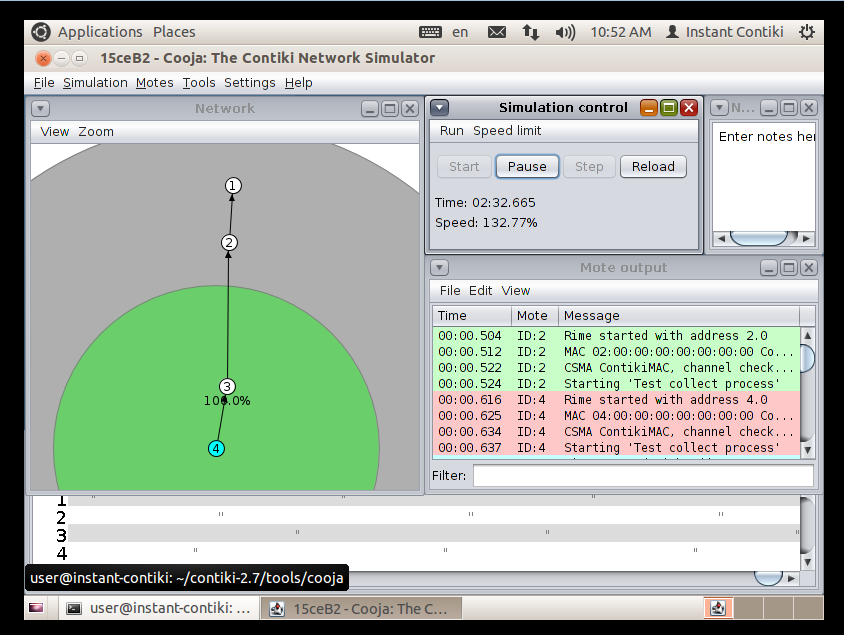
**Step-9:** Click the Start button to start the simulation.

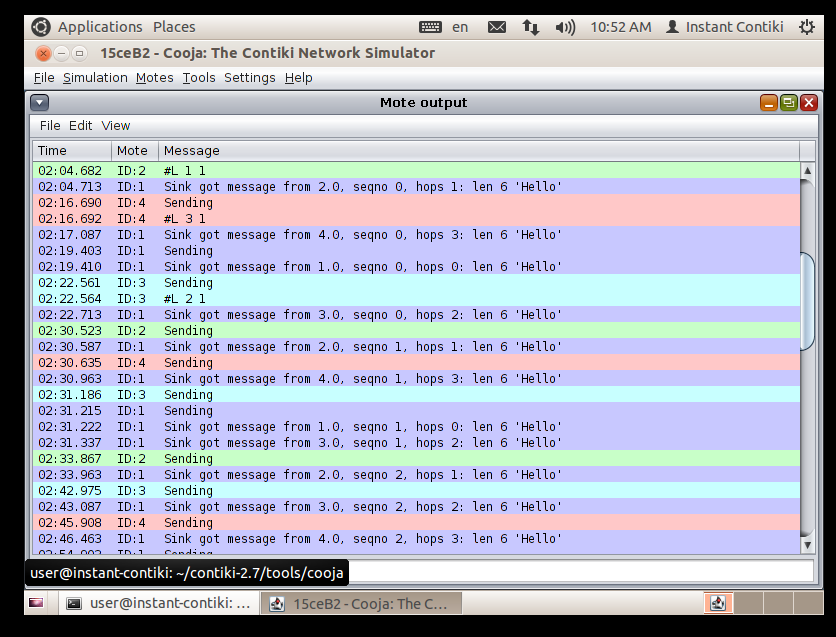




Here, Mote 1 is working as sink so it will receive all the message from this radio environment. All other mote will send message to mote 1.

**Step-10:** We see printouts from the simulated motes appearing in the Mote output window. The Network window shows communication going on in the network. The Timeline window shows communication and radio events over time - the small gray lines are ContikiMAC periodically waking the radio up. We can click the Pause button to pause the simulation.





**Conclusion**

From this practical, we know about how we can perform simulation in Contiki-OS with Cooja tool.