RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution
Affiliated to Anna University, Chennai,
Rajalakshmi Nagar, Thandalam – 602 105

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

CS19541 - COMPUTER NETWORKS
Laboratory Record Note Book

Name : SARUMATHY J

Register No. : 221501126

Year / Branch / Section : III / BTECH AIML-C

Semester : V

AcademicYear: 2024-2025

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institution
Affiliated to Anna University, Chennai,
Rajalakshmi Nagar, Thandalam – 602 105

BONAFIDE CERTIFICATE

Name: SARUMATHY J

Academic Year: 2024 **Semester:** V **Branch:** BTECH-AIML

Register No. 2116221501126

Certified that this is the bonafide record of work done by the above student in the CS19541- Computer Networks ***Laboratory during the academic year 2024- 2025***

Signature of Faculty in-charge

Submitted for the Practical Examination held on……

Internal Examiner **External Examiner**

Table of Content

24	25.10.24	Capturing and Analyzing Packets Using Wireshark Tool	92	
25	26.10.24	Analyze Weblogs Using Webalizer	96	

Exp. No. 01	Study Of Various Networking Commands Used in Linux and Windows
Date: 19.07.24	

Basic Networking Commands:

1) **arp -a:** ARP is short form of address resolution protocol, It will show the IP address of your computer along with the IP address and MAC address of your router.
   Output:

```
[student@fedora ~]$ arp -a
_gateway (172.16.52.1) at 7c:5a:1c:cf:be:3e [ether] on enp2s0
```

2) **hostname:** This is the simplest of all TCP/IP commands. It simply displays the name of your computer.
   Output:

```
[student@fedora ~]$ hostname
fedora
```

3) **ipconfig/all:** This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your System.
   Output:

```
[student@fedora ~]$ ifconfig
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.52.83  netmask 255.255.254.0  broadcast 172.16.53.255
        inet6 fe80::d193:bbe:929b:17d1  prefixlen 64  scopeid 0x20<link>
        ether 88:ae:dd:15:ec:fb  txqueuelen 1000  (Ethernet)
        RX packets 199637  bytes 100276261 (95.6 MiB)
        RX errors 0  dropped 8587  overruns 0  frame 0
        TX packets 84435  bytes 25372084 (24.1 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 292  bytes 42675 (41.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 292  bytes 42675 (41.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

4) **nbstat -a:** This command helps solve problems with NetBIOS name resolution. (Nbt stands for NetBIOS over TCP/IP)
   Output:

```
[student@fedora ~]$ netstats -r
bash: netstats: command not found...
Similar command is: 'netstat'
[student@fedora ~]$ netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         _gateway        0.0.0.0         UG        0 0          0 enp2s0
172.16.52.0     0.0.0.0         255.255.254.0   U         0 0          0 enp2s0
```

5) **netstat:** (network statistics) netstat displays a variety of statistics about a computers active TCP/IP connections. It is a command line tool for monitoring network connections both incoming and outgoing as well as viewing routing tables, interface statistics etc.
   e.g.:- netstat -r

Output:

```
[student@fedora ~]$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 fedora:57916           maa05s25-in-f10.1:https ESTABLISHED
tcp        0      0 fedora:50068           maa05s19-in-f3.1e:https ESTABLISHED
tcp        0      0 fedora:50080           maa05s19-in-f3.1e:https ESTABLISHED
tcp        0      0 fedora:48494           93.243.107.34.bc.:https ESTABLISHED
tcp        0      0 fedora:37772           maa05s26-in-f14.1:https ESTABLISHED
tcp        0      0 fedora:51330           maa03s45-in-f14.1:https ESTABLISHED
tcp        0      0 fedora:47066           maa05s20-in-f5.1e:https ESTABLISHED
tcp        0      0 fedora:44674           se-in-f84.1e100.n:https ESTABLISHED
tcp        0      0 fedora:48284           maa05s20-in-f5.1e:https TIME_WAIT
tcp        0      0 fedora:57904           maa05s25-in-f10.1:https ESTABLISHED
tcp        0      0 fedora:44036           maa03s26-in-f14.1:https ESTABLISHED
udp        0      0 fedora:bootpc          _gateway:bootps         ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  3      [ ]         STREAM     CONNECTED     34078
unix  3      [ ]         STREAM     CONNECTED     28171
unix  3      [ ]         STREAM     CONNECTED     23299    /run/dbus/system_bus_socket
unix  2      [ ]         DGRAM                    30896
unix  3      [ ]         STREAM     CONNECTED     25354    /run/user/1000/bus
```

6) **nslookup:** (name server lookup) is a tool used to perform DNS lookups in Linux. It is used to display DNS details, such as the IP address of a particular computer, the MX records for a domain or the NS servers of a domain. nslookup can operate in two modes: interactive and Non-interactive.

e.g.:- nslookup www.google.com

Output:

```
[student@fedora ~]$ nslookup www.google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   www.google.com
Address: 142.250.182.4
Name:   www.google.com
Address: 2404:6800:4007:819::2004
```

7) **ping:** (Packet INternet Groper) command is the best way to test connectivity between two nodes. Ping uses ICMP (Internet Control Message Protocol) to communicate to other devices.

1. #ping hostname(ping localhost)
2. #ping ip address (ping 4.2.2.2)
3. #ping fully qualified domain name(ping www.facebook.com

Output:

```
[student@fedora ~]$ ping fedora
PING fedora(fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0)) 56 data bytes
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=2 ttl=64 time=0.068 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=5 ttl=64 time=0.063 ms
```

8) **Route:** route command is used to show/manipulate the IP routing table. It is primarily used to set up static routes to specific hosts or networks via an interface.

Output:

```
[student@fedora ~]$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         _gateway        0.0.0.0         UG    100    0        0 enp2s0
```

Some important Linux networking commands:

1) **ip:**

The ip command is one of the basic commands every administrator will need in daily work, from setting up new systems and assigning IPs to troubleshooting existing systems. The ip command can show address information, manipulate routing, plus display network various devices, interfaces, and tunnels.

ip <OPTIONS> <OBJECT> <COMMAND>

Here are some common use cases for the ip command.

a. To show the IP addresses assigned to an interface on your server: [root@server ~]# ip address show

Output:

```
[student@fedora ~]$ ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 88:ae:dd:15:eb:e8 brd ff:ff:ff:ff:ff:ff
    inet 172.16.53.236/23 brd 172.16.53.255 scope global dynamic noprefixroute enp2s0
       valid_lft 84786sec preferred_lft 84786sec
    inet6 fe80::6802:2582:1fb9:bbf3/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

b. To assign an IP to an interface, for example, enps03:

[root@server ~]# ip address add 192.168.1.254/24 dev enps03

Output:

```
[root@fedora ~]# ip address add 192.168.1.254/24 dev enp2s0
[root@fedora ~]# ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 88:ae:dd:15:eb:9f brd ff:ff:ff:ff:ff:ff
    inet 172.16.53.134/23 brd 172.16.53.255 scope global noprefixroute enp2s0
       valid_lft forever preferred_lft forever
    inet 192.168.1.254/24 scope global enp2s0
       valid_lft forever preferred_lft forever
    inet6 fe80::4ce1:20d3:25c2:b9ad/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

c. To delete an IP on an interface:

[root@server ~]# ip address del 192.168.1.254/24 dev enps03

Output:

```
[root@fedora ~]# ip address del 192.168.1.254/24 dev enp2s0
[root@fedora ~]# ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp2s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 88:ae:dd:15:eb:9f brd ff:ff:ff:ff:ff:ff
    inet 172.16.53.134/23 brd 172.16.53.255 scope global noprefixroute enp2s0
       valid_lft forever preferred_lft forever
    inet6 fe80::4ce1:20d3:25c2:b9ad/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

d. Display the route taken for IP 10.10.1.4: [root@server ~]# ip route get 10.10.1.4

Output:

```
[student@fedora ~]$ ip route get 10.10.1.4
10.10.1.4 via 172.16.52.1 dev enp2s0 src 172.16.53.236 uid 1000
    cache
```

2) ifconfig: The ifconfig command was/is a staple in many sysadmin's tool belt for configuring

and

troubleshooting networks. It has since been replaced by the ip command discussed above.

Output:

```
[student@fedora ~]$ ifconfig
enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.53.126  netmask 255.255.254.0  broadcast 172.16.53.255
        inet6 fe80::e5d3:4ba6:f3f4:5526  prefixlen 64  scopeid 0x20<link>
        ether 88:ae:dd:15:ee:fe  txqueuelen 1000  (Ethernet)
        RX packets 112679  bytes 64994994 (61.9 MiB)
        RX errors 0  dropped 1263  overruns 0  frame 0
        TX packets 26201  bytes 12868430 (12.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 283  bytes 146669 (143.2 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 283  bytes 146669 (143.2 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3) mtr:

MTR (Matt's traceroute) is a program with a command-line interface that serves as a network diagnostic and troubleshooting tool. This command combines the functionality of the ping and traceroute commands. Just like a traceroute, the mtr command will show the route from a computer to a specified host. mtr provides a lot of statistics about each hop, such as response time and percentage. With the mtr command, you will get more information about the route and be able to see problematic devices along the way. If you see a sudden increase in response time or packet loss, then obviously, there is a bad link somewhere.

The syntax of the command is as follows:

mtr <options> hostname/IP

Let's look at some common use cases.

a. The basic mtr command shows you the statistics, including each hop

[root@server ~]# mtr google.com

Output:

```
                                        My traceroute  [v0.95]
fedora (172.16.53.126) -> google.com (142.250.183.238)                  2024-08-03T08:29:16+0530
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                                          Packets               Pings
 Host                                                  Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. _gateway                                            0.0%     6    0.4   0.4   0.3   0.6   0.1
 2. static-41.229.249.49-tataidc.co.in                  0.0%     6    9.0   7.1   3.0  10.3   2.8
 3. 142.250.171.162                                     0.0%     6   10.4  10.2   2.8  14.7   4.3
 4. 142.251.227.217                                     0.0%     5    4.8  10.1   3.8  22.1   7.3
 5. 209.85.253.85                                       0.0%     5    4.9  10.7   3.6  18.7   6.5
 6. maa05s23-in-f14.1e100.net                           0.0%     5    6.5  12.2   2.9  25.6   9.5
```

b. Show the numeric IP addresses and hostnames, too: [root@server ~]# mtr -b google.com

Output:

```
                                        My traceroute  [v0.95]
fedora (172.16.53.126) -> google.com (142.250.183.238)                  2024-08-03T08:32:18+0530
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                                          Packets               Pings
 Host                                                  Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. _gateway (172.16.52.1)                              0.0%     5    0.3   1.7   0.3   4.3   1.9
 2. static-41.229.249.49-tataidc.co.in (49.249.229.41)  0.0%     5    5.0   7.8   3.5  12.9   4.0
 3. 142.250.171.162 (142.250.171.162)                   0.0%     5    4.2  14.8   3.6  39.9  14.8
 4. 142.251.227.217 (142.251.227.217)                   0.0%     4   14.5  11.9   4.3  21.8   7.9
 5. 209.85.253.85 (209.85.253.85)                       0.0%     4   12.3   7.1   3.6  12.3   4.3
 6. maa05s23-in-f14.1e100.net (142.250.183.238)         0.0%     4   18.5  14.2   3.1  19.6   7.6
```

c. Set the number of pings that you want to send: [root@server ~]# mtr -c 10 google.com

Output:

```
                                        My traceroute  [v0.95]
fedora (172.16.53.126) -> google.com (142.250.183.238)                  2024-08-03T08:32:49+0530
Keys:  Help   Display mode   Restart statistics   Order of fields   quit
                                                          Packets               Pings
 Host                                                  Loss%   Snt   Last   Avg  Best  Wrst StDev
 1. _gateway                                            0.0%     4    0.4   0.8   0.4   1.5   0.5
 2. static-41.229.249.49-tataidc.co.in                  0.0%     4    2.6   7.0   2.6   9.6   3.2
 3. 142.250.171.162                                     0.0%     4    7.9  13.8   7.9  24.1   7.2
 4. 142.251.227.217                                     0.0%     3   11.6  16.5  11.6  19.4   4.2
 5. 209.85.253.85                                       0.0%     3   31.6  23.2  13.6  31.6   9.1
 6. maa05s23-in-f14.1e100.net                           0.0%     3   19.0  16.2  10.8  19.0   4.7
```

4) tcpdump:

The tcpdump command is designed for capturing and displaying packets.

a. Before starting any capture, you need to know which interfaces tcpdump can use. You will need to use sudo or have root access in this case.

[root@server ~]# tcpdump -D

Output:

```
[root@fedora ~]# tcpdump -D
1.enp2s0 [Up, Running, Connected]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.bluetooth-monitor (Bluetooth Linux Monitor) [Wireless]
5.usbmon2 (Raw USB traffic, bus number 2)
6.usbmon1 (Raw USB traffic, bus number 1)
7.usbmon0 (Raw USB traffic, all USB buses) [none]
8.nflog (Linux netfilter log (NFLOG) interface) [none]
9.nfqueue (Linux netfilter queue (NFQUEUE) interface) [none]
```

5) ping:

Ping is a tool that verifies IP-level connectivity to another TCP/IP computer by sending Internet Control Message Protocol (ICMP) Echo Request messages. The receipt of corresponding Echo Reply messages are displayed, along with round-trip times. Ping is the primary TCP/IP command used to troubleshoot connectivity, reachability, and name resolution.

Output:

```
[student@fedora ~]$ ping fedora
PING fedora(fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0)) 56 data bytes
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=2 ttl=64 time=0.068 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=4 ttl=64 time=0.068 ms
64 bytes from fedora (fe80::e5d3:4ba6:f3f4:5526%enp2s0): icmp_seq=5 ttl=64 time=0.063 ms
```

Result:

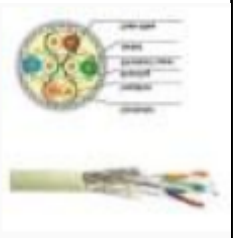
Thus, the Basic Linux commands have been executed successfully.

Exp. No. 02	Study of different types of Network cables
Date: 26.07.24	

a) <u>Understand different types of network cable.</u>

Different type of cables used in networking are:

1. Unshielded Twisted Pair (UTP) Cable
2. Shielded Twisted Pair (STP) Cable
3. Coaxial Cable
4. Fibre Optic Cable

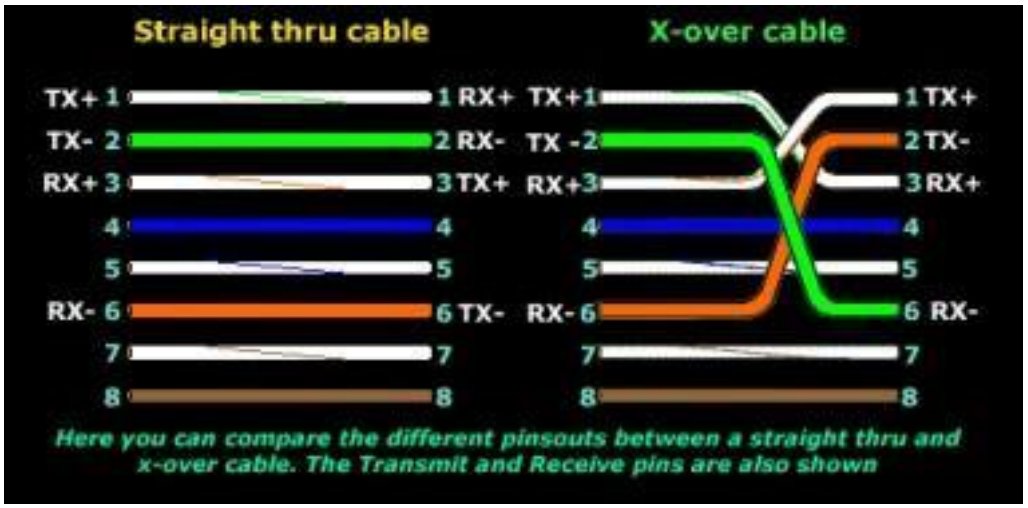
Cable type	Category	Maximum Data Transmission	Advantages/ Disadvantages	Application/Use	Image
UTP	Category 3	10 bps	**Advantages** * Cheaper in cost * Easy to install as they have a smaller overall diameter. **Disadvantages** * More prone to (EMI) Electromagnetic interference and noise	10Base-T Ethernet	
	Category 5	Up to 100 Mbps		Fast Ethernet, Gigabit Ethernet	
	Category 5	1Gbps		Fast Ethernet, Gigabit Ethernet	
STP	Category6,6a	10Gbps	**Advantages** *Shielded. *Faster than UTP *Less susceptible to noise and interference **Disadvantages** *Expensive *Greater installation effort	Gigabit Ethernet, 10G Ethernet (55m) Widely used in data centers	
SSTP	Category 7	10Gbps		Gigabit Ethernet, 10G Ethernet (100m)	
Coaxial cable	RG-6 RG-59 RG-11	10-100Mbps	**Advantages** *High bandwidth *Immune to interference	Speed of signal is 500m Television network High speed internet	

			*Low loss	connections	
			bandwidth * Versatile **Disadvantages** *Limited distance * Cost * Size is bulky		
Fiber optics cable	Single mode Multi mode	100Gbps	**Advantages** *High speed *High bandwidth *High security *Long distance **Disadvantages** *Expensive * Requires skilled installers	Maximum distance of fiber optics cable is around 100 meters	

b) <u>Make Your Own Ethernet Cross-Over Cable/ Straight cable</u>

Tools and parts needed:

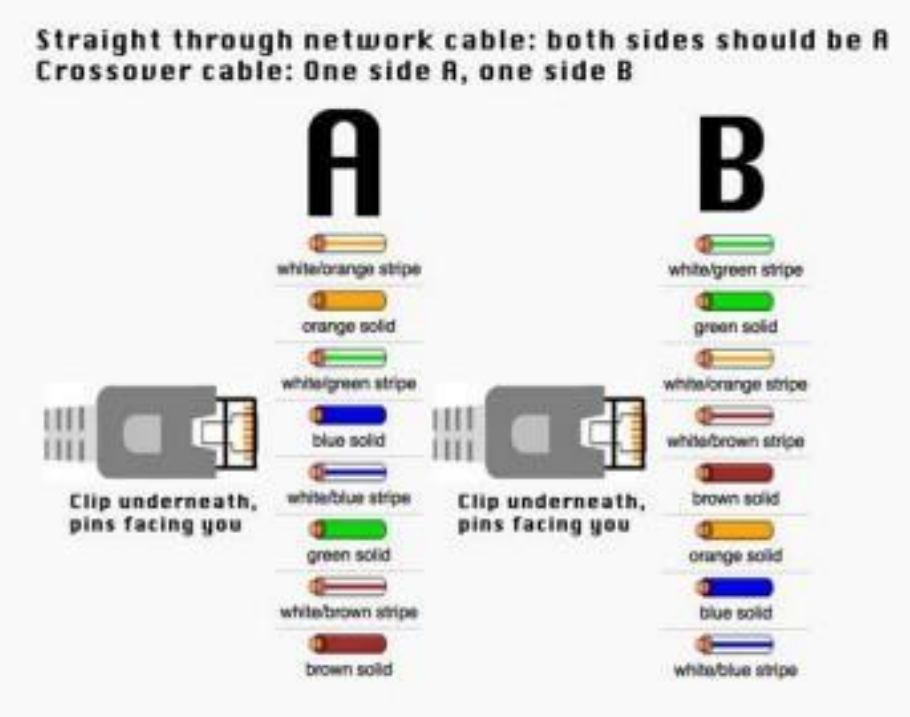
- ➢ Ethernet cabling. CAT5e is certified for gigabit support, but CAT5 cabling works as well, just over shorter distances.
- ➢ A crimping tool. This is an all-in-one networking tool shaped to push down the pins in the plug and strip and cut the shielding off the cables.
- ➢ Two RJ45 plugs.



➢ Optional two plug shields.

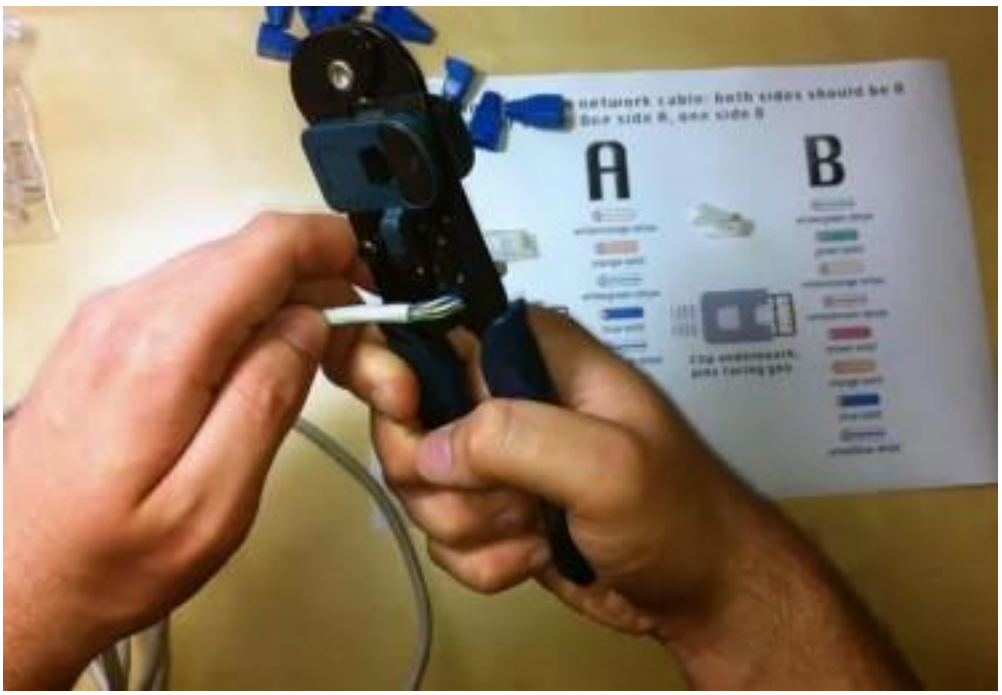
Difference between crossover cable and straight cable

Take a print out the diagram below or have it handy as a reference

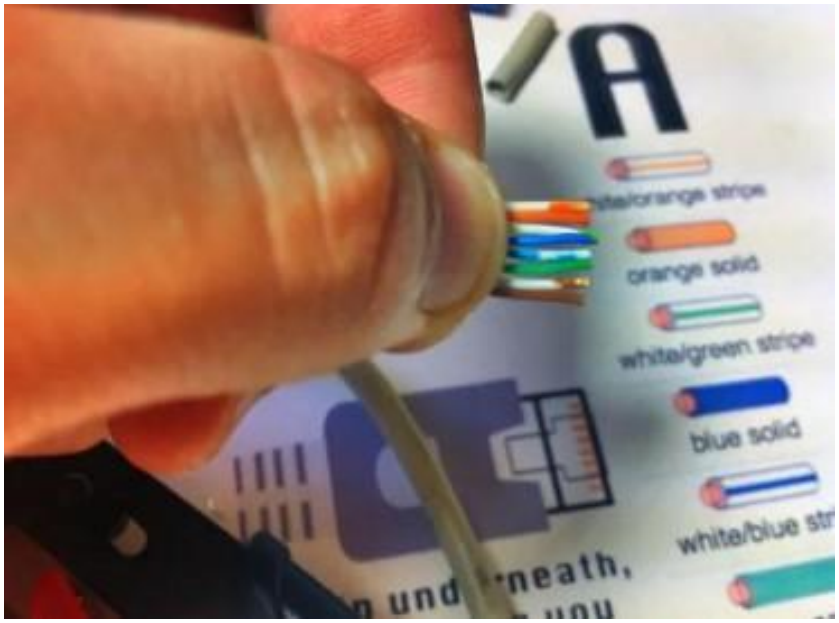


Step 1: To start construction of the device, begin by threading shields onto the cable.

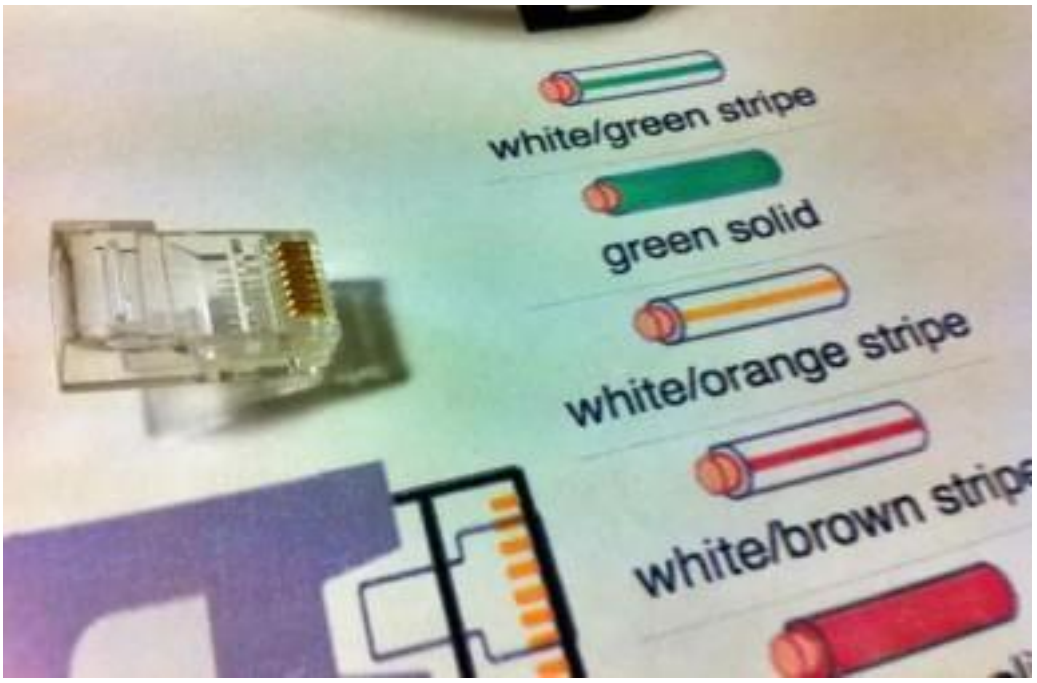
Step 2: Next, strip approximately 1.5 cm of cable shielding from both ends. The crimping tool has a round area to complete this task.



Step 3: After, you will need to untangle the wires; there should be four "twisted pairs." Referencing back to the sheet, arrange them from top to bottom. One end should be in arrangement A and the other in B.



Step 4: Once the order is correct, bunch them together in a line, and if there are any that stick out farther than others, snip them back to create an even level. The difficult aspect is placing these into the RJ45 plug without messing up the order. To do so, hold the plug with the clip side facing away from you and have the gold pins facing toward you, as shown.



Step 5: Next, push the cable right in. The notch at the end of the plug needs to be just over the cable shielding, and if it isn't, that means that you stripped off too much shielding. Simply snip the cables back a little more.

Result:

Thus the study of different types of networking cable has been completed successfully.

Exp. No. 03	Study of Socket Programming
Date: 27.07.24	

What is Socket Programming?

Socket Programming is a way of connecting two nodes on a network to communicate with each other. One socket or node listens on a particular port at an IP, while other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server. They are the real backbones behind web browsing.

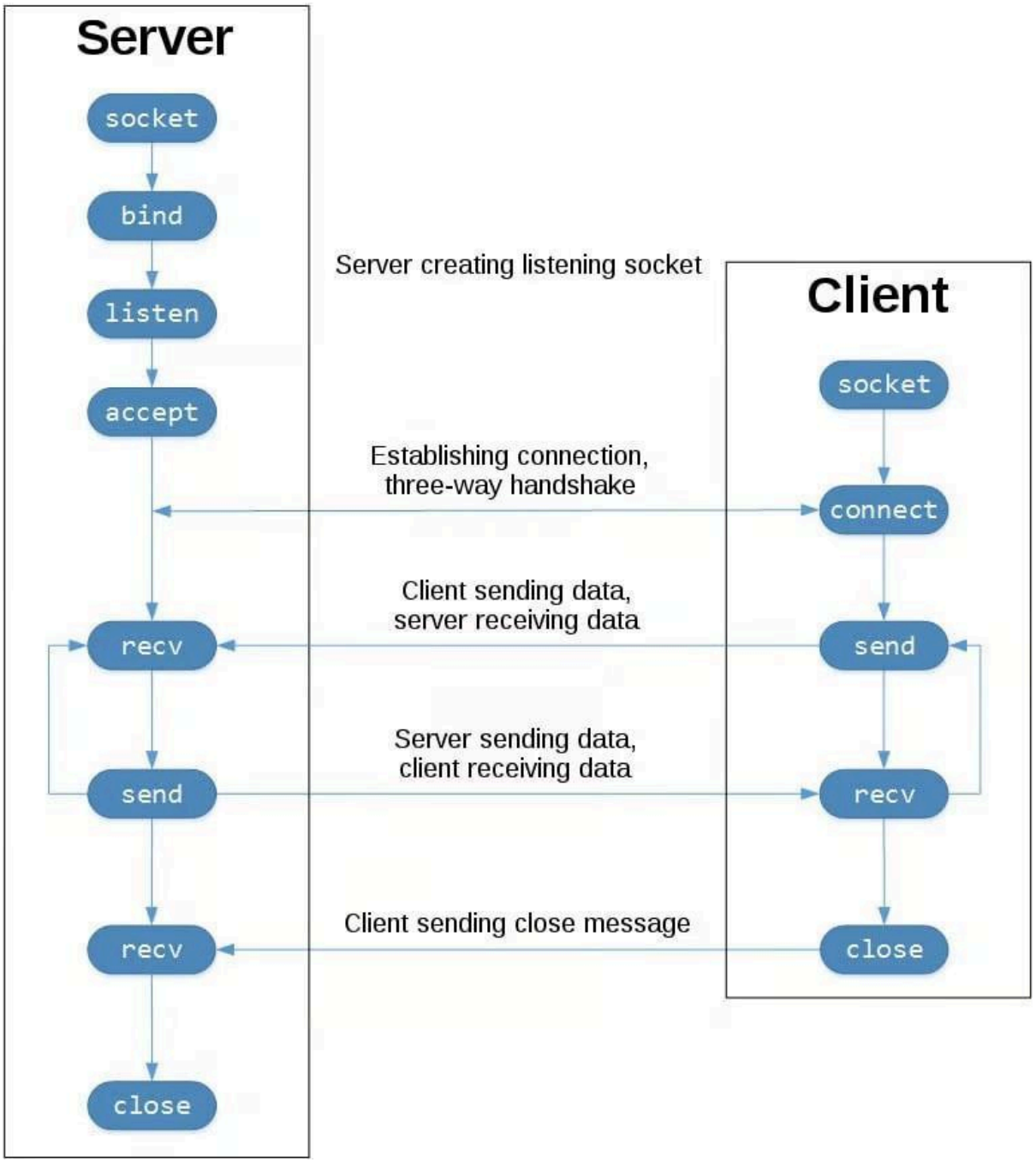
In simpler terms, there is a server and a client. Socket programming is started by importing the socket module or framework. This module consists of built-in methods that are required for creating sockets and help them associate with each other.

Types of Socket Programming:

- **TCP**
  A connection-oriented protocol that's best for direct communication where reliability is important. TCP is used for web browsing, email, text messaging, and file transfers. TCP sockets establish a persistent connection between two processes, and data packets arrive in the correct order.

- **UDP**
  A connectionless protocol that's best for real-time data transmission when speed is more important than reliability. UDP is used for online gaming, live streaming, and DNS protocols. UDP sockets prioritize time over reliability, and there is no session between the client and the server’

TCP Sockets:



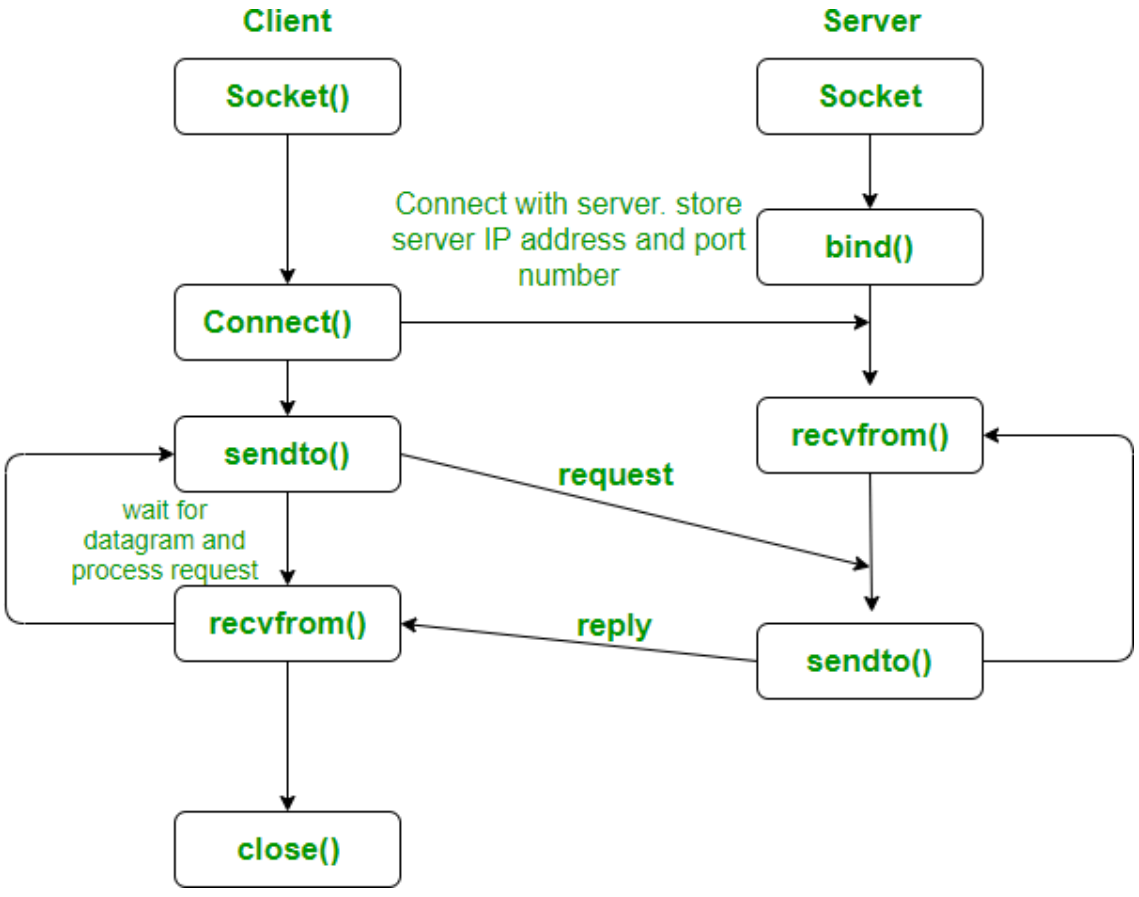
Function Call	Description
Socket()	To create a socket
Bind()	IT's a socket identification like a telephone number to contact
Listen()	Ready to receive a connection
Connect()	Ready to act as a sender

Accept()	Confirmation, it is like accepting to receive a call from a sender
Read()	To receive data
Close()	To close a connection
Write()	To send data

Steps for TCP Connection:

- **Create a socket:** The server creates a socket and listens for connections
- **Connect:** The client creates a socket and connects to the server's socket
- **Establish a connection:** The client and server use a three-way handshake to establish a connection
- **Transfer data:** Data is broken into segments and transmitted over the network
  Acknowledge: The receiving end acknowledges the segments to ensure reliable delivery
- **Regulate flow:** TCP regulates the flow of data to prevent the sender from overwhelming the receiver
- **Terminate connection:** The connection is terminated using a four-way handshake
  Close sockets: The sockets are closed

UDP Sockets:



Function Call	Description
Socket.socket()	Creates a new UDP socket.
Socket.bind(address)	Binds the socket to a specified IP address and port.
Socket.sendto(data, address)	Sends data to a specified address.
Socket.recvfrom(buffsize)	Receives data from the socket, returning the data and the address of the sender.
Socket,close()	Closes the socket to free up resources.

STEPS FOR UDP CONNECTION :

UDP Server :

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to Step 3.

UDP Client :

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until a response from the server is received.
4. Process the reply and go back to step 2, if necessary
5. Close socket descriptor and exit.

Result:

Thus the study of Socket Programming has been completed successfully.

Exp. No. 04	Implementation Of Echo Client/Server Application Using TCP
Date: 02.08.24	

Aim:

To develop Echo server and client program using TCP sockets

Algorithm:

Step 1: Server creates a TCP socket and binds it to port `8000`.

Step 2: Server listens for incoming connections.

Step 3: Client creates a TCP socket and connects to the server.

Step 4: Client sends a message (e.g., `"Hello"`) to the server.

Step 5: Server accepts the client connection.

Step 6: Server receives and decodes the message from the client.

Step 7: Server sends the same message back to the client.

Step 8: Client receives and decodes the server's response.

Step 9: Both client and server print the message to their consoles.

Step 10: Both sockets are closed after communication ends.

Program:

Server.py

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",8000))
s.listen(5)
while True:
   c,a = s.accept()
   print("Received connection
   from",a) data=c.recv(100).decode()
   print(data)
   c.send(data.encode('utf-8')
   ) c.close()
```

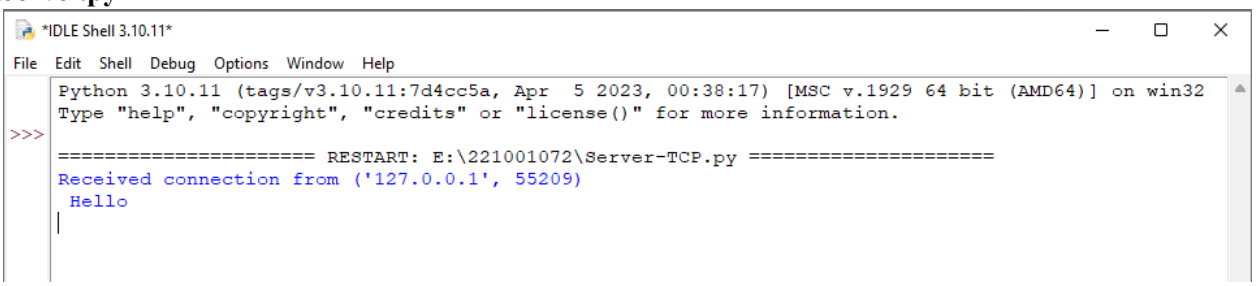
Client.py

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.connect(("127.0.0.1",8000))
op=" Hello "
s.send(op.encode('utf-8'))
data=s.recv(100).decode(
) print(data)
```

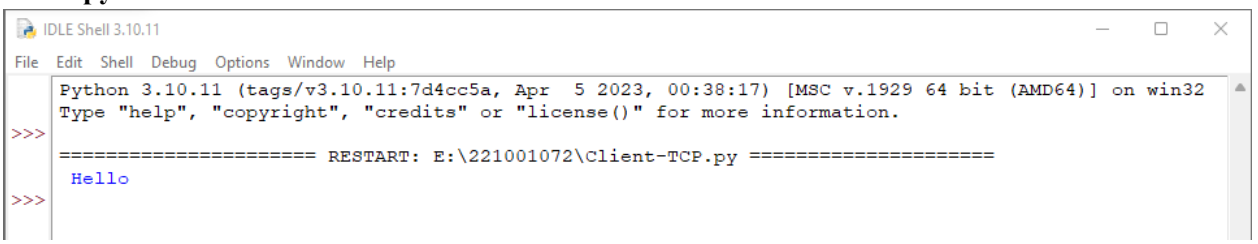
```
s.close()
```

Output:

Server.py



Client.py



Result:

Thus the implementation of the echo server and client program using TCP has been executed successfully.

Exp. No. 05	Implementation Of Echo Client/Server Application Using UDP
Date: 03.08.24	

Aim:

To develop Echo server and client program using UDP sockets

Algorithm:

Step 1: Server creates a UDP socket using socket(AF_INET, SOCK_DGRAM) and binds it to port 12000.

Step 2: Client creates a UDP socket using socket(AF_INET, SOCK_DGRAM).

Step 3: Client sends a message (e.g., "test") to the server at 127.0.0.1 on port 12000 using sendto().

Step 4: Server receives the message using recvfrom(), decodes it, and prints it.

Step 5: Server sends the same message back to the client using sendto().

Step 6: Client receives the response using recvfrom(), decodes it, and prints it.

Step 7: Both the server and client sockets remain open for further communication .

Program:

Server.py

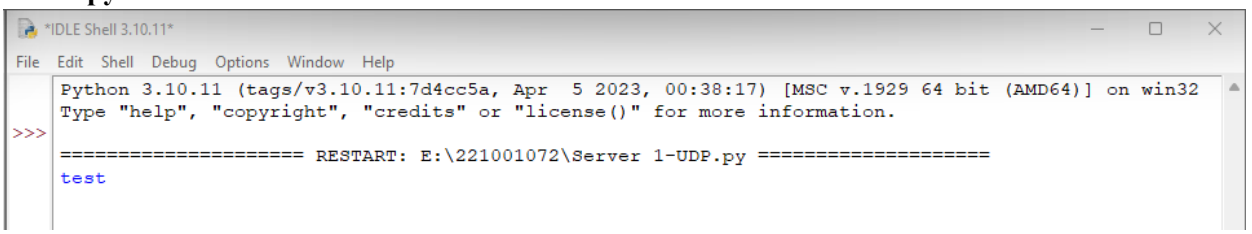
```
import socket
server_socket=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_socket.bind(('',12000))
while True:
   m,address=server_socket.recvfrom(1024)
   n=m.decode()
   print(n)
   server_socket.sendto(n.encode('utf-8'),address)
```

Client.py

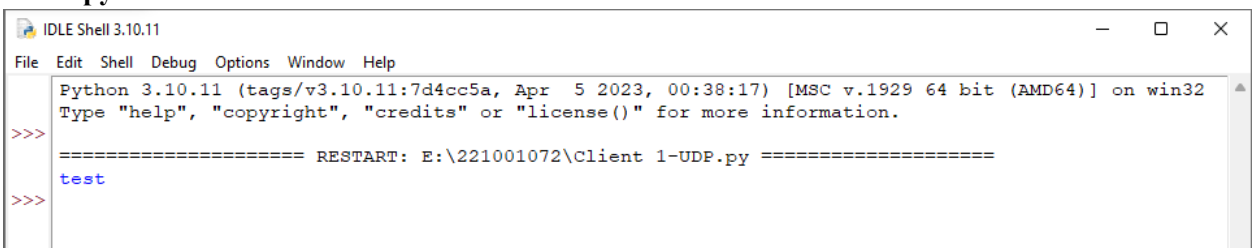
```
import socket
client_socket=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
Message = 'test'
addr=("127.0.0.1",12000)
client_socket.sendto(Message.encode('utf-8'),addr)
data,server=client_socket.recvfrom(1024)
m=data.decode()
print(m)
```

Output:

Server.py



Client.py



Result:

Thus the implementation of the echo server and client program using UDP has been executed successfully.

Exp. No. 06	Implementation Of Chat Application Using TCP
Date: 09.08.24	

Aim:

To develop a Chat server and client program using sockets in pythons.

Algorithm:

Step 1: Server creates a TCP socket using socket(AF_INET, SOCK_STREAM),

binds it to port 12000

Step 2: Client creates a TCP socket using socket(AF_INET, SOCK_STREAM)

and connects to the server at 127.0.0.7 on port 12000.

Step 3: Client sends a message to the server using send(), and

the server receives it using recv(), decodes it, and prints it.

Step 4: Server sends a response back to the client using send(), and the

client receives it using recv(), decodes it, and prints it.

Step 5: Both continue exchanging messages in a loop until either sends "bye".

Once "bye" is exchanged, the chat ends, and both sockets are closed.

Program:

Server.py

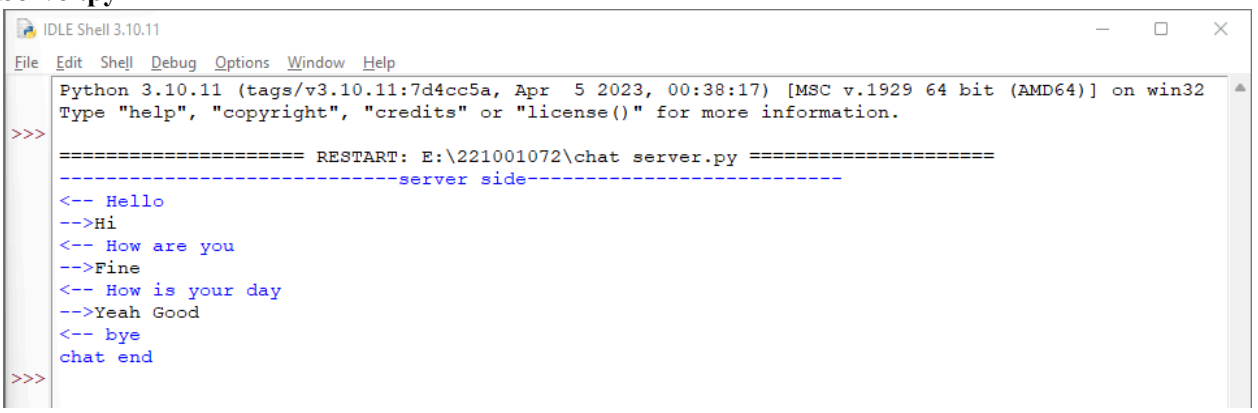
```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",12000))
s.listen(5)
print("--------------------server side--------------------
") c,a = s.accept()
while True:
   data=c.recv(100).decode(
   ) print("<--",data)
   if(data=="bye" or data=="Bye" or data=="BYE"):
      p="bye"
      c.send(p.encode('utf-8'))
      print("chat end")
      break
   msg=input("-->",
   )
   c.send(msg.encode('utf-8'))
```

Client.py

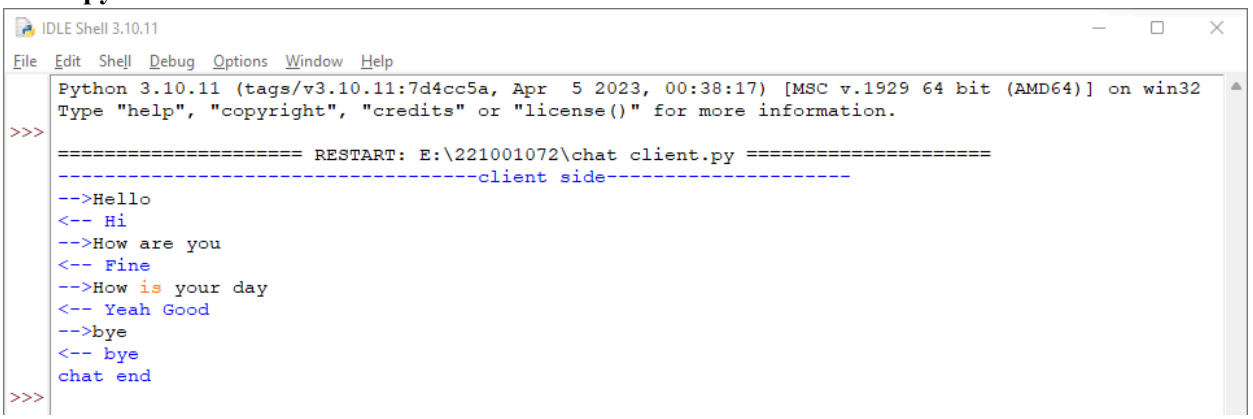
```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.connect(("127.0.0.7",12000))
print("---------- client side----------
") while True:
   msg=input("-->",)
   s.send(msg.encode('utf-8')
   )
   data=s.recv(100).decode()
   print("<--",data)
   if(data=="bye" or data=="Bye" or data=="BYE"):
      print("chat end")
      brea
```

k **Output:**

Server.py



Client.py



Result:

Thus the implementation of a chat application using TCP has been executed successfully.

Exp. No. 07	Uppercase Converter Using UDP Socket
Date: 10.08.24	

Aim:

To develop a program to do uppercase conversion using UDP in sockets.

Algorithm:

Step 1: Server creates a UDP socket, binds it to port 12000, and prints t

he message "UPPERCASE CONVERTER".

Step 2: Client creates a UDP socket and prints "UPPERCASE CONVERTER".

Step 3: Client inputs a message and sends it to the server using sendto().

Step 4: Server receives the message using recvfrom(), converts it to uppercase,

and sends it back to the client.

Step 5: Client receives the uppercase message using recvfrom() and prints it.

Step 6: Client continues to input and send messages in a loop.

Step 7: Server continuously receives messages, converts them to uppercase, and responds.

Step 8: The process repeats until the program is manually stopped.

Program:

Server.py

```
import socket
server_socket = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
server_socket.bind(('',12000))
print("-------------------------UPPERCASE CONVERTER----------------")
while True:
    message, address = server_socket.recvfrom(1024)
    f = message.upper()
    server_socket.sendto(f,address)
```

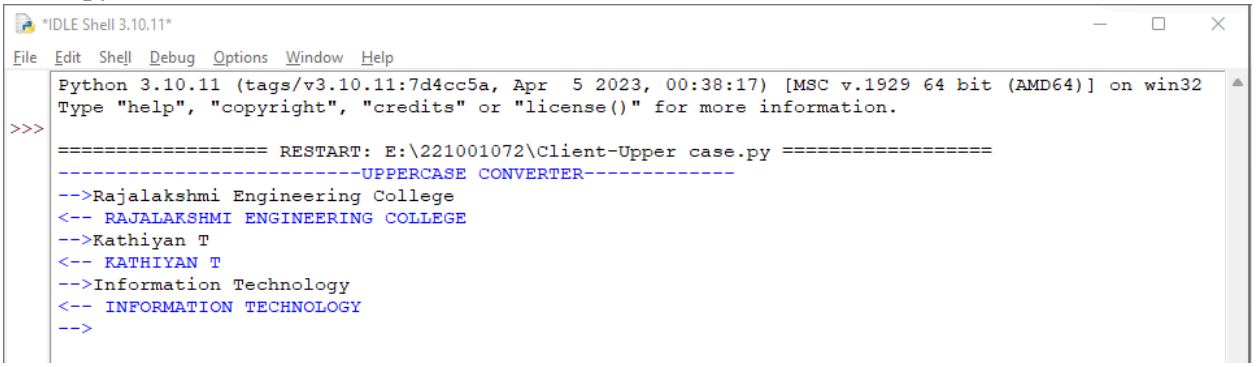
Client.py

```
import socket
client_socket =
socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
print('-------------------------UPPERCASE CONVERTER                ')
while True:
    msg = input("-->")
    m =
    str.encode(msg)
    addr = ("127.0.0.1",12000)
    client_socket.sendto(m,addr)
```

```
data,server =
client_socket.recvfrom(1024)
print("<--",data.decode())
```

Output:

Client.py



Result:

Thus the Python program for Uppercase conversion using UDP has been executed successfully.

Exp. No. 08	Calculator Application Using TCP Socket
Date: 16.08.24	

Aim:

To develop a program to do Calculator Application using TCP in sockets.

Algorithm:

Step 1: Server creates a TCP socket, binds it to port 8000, and

starts listening for client connections.

Step 2: Client connects to the server, and the server accepts the connection.

Step 3: Server displays a menu with calculator options and waits for the

client to select an operation (1 for addition, 2 for subtraction, 3 for multiplication,

4 for division, 5 to end).

Step 4: Depending on the client's choice, the server prompts for two

Program:

Server.py

```
from socket import*
s=socket(AF_INET,SOCK_STREAM)
s.bind(("",8000))
s.listen(5)
print("=======================CALCULATOR============================="
)
c,a=s.accept()
while True:
   data=c.recv(100).decode()
   if(data=='1'):
      f=("Enter 1st value")
      c.send(f.encode('utf-8'))
      d1=c.recv(100).decode(
      ) print(d1)
      d3=int(d1)
      g=("enter 2nd value")
```

```
        c.send(g.encode('utf-8'))
        d2=c.recv(100).decode(
        ) print(d2)
        d4=int(d2)
        ans=d3+d4
        ans1=str(ans)
        c.send(ans1.encode('utf-8'))
    if(data=='2'):
        f = ("Enter 1st value")
        c.send(f.encode('utf-8'))
        d1 =
        c.recv(100).decode()
        print(d1)
        d3=int(d1)
        g=("enter 2nd value")
        c.send(g.encode('utf-8'))
        d2=c.recv(100).decode(
        ) print(d2)
        d4=int(d2)
        ans=d3-d4
        ans1=str(ans
        )
        c.send(ans1.encode('utf-8'))
    if(data=='3'):
        f = ("Enter 1st value")
        c.send(f.encode('utf-8'))
        d1 =
        c.recv(100).decode()
        print(d1)
        d3=int(d1)
        g=("enter 2nd value")
        c.send(g.encode('utf-8'))
        d2=c.recv(100).decode(
        ) print(d2)
        d4=int(d2)
        ans=d3*d4
        ans1=str(ans)
        c.send(ans1.encode('utf-8'))
    if(data=='4'):
        f = ("Enter 1st value")
        c.send(f.encode('utf-8'))
        d1 =
        c.recv(100).decode()
        print(d1)
        d3=int(d1)
        g=("enter 2nd value")
        c.send(g.encode('utf-8'))
        d2=c.recv(100).decode(
        ) print(d2)
        d4=int(d2)
        ans=d3/d4
        ans1=str(ans)
        c.send(ans1.encode('utf-8'))
    if(data=='5'):
        ans1="end"
```

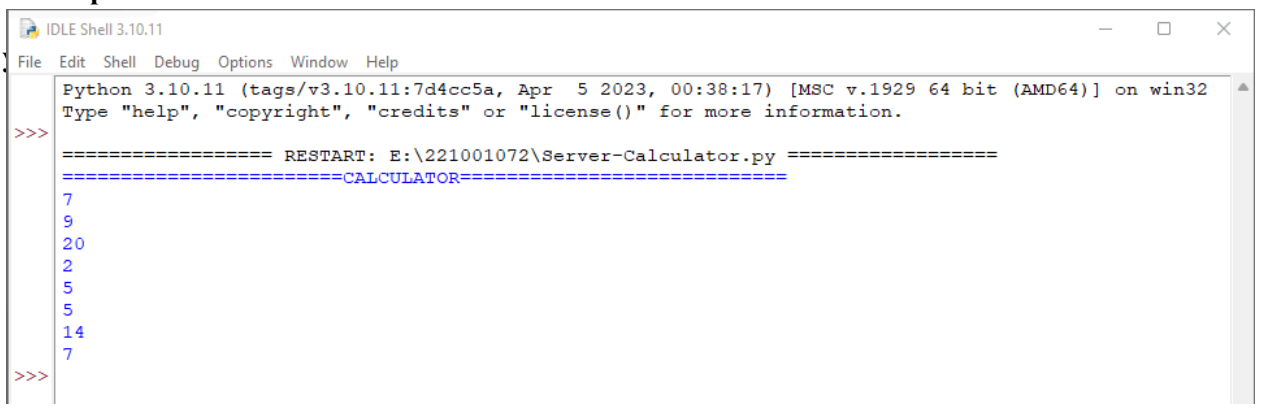
```
c.send(ans1.encode('utf-8')
) break
```

Client.py

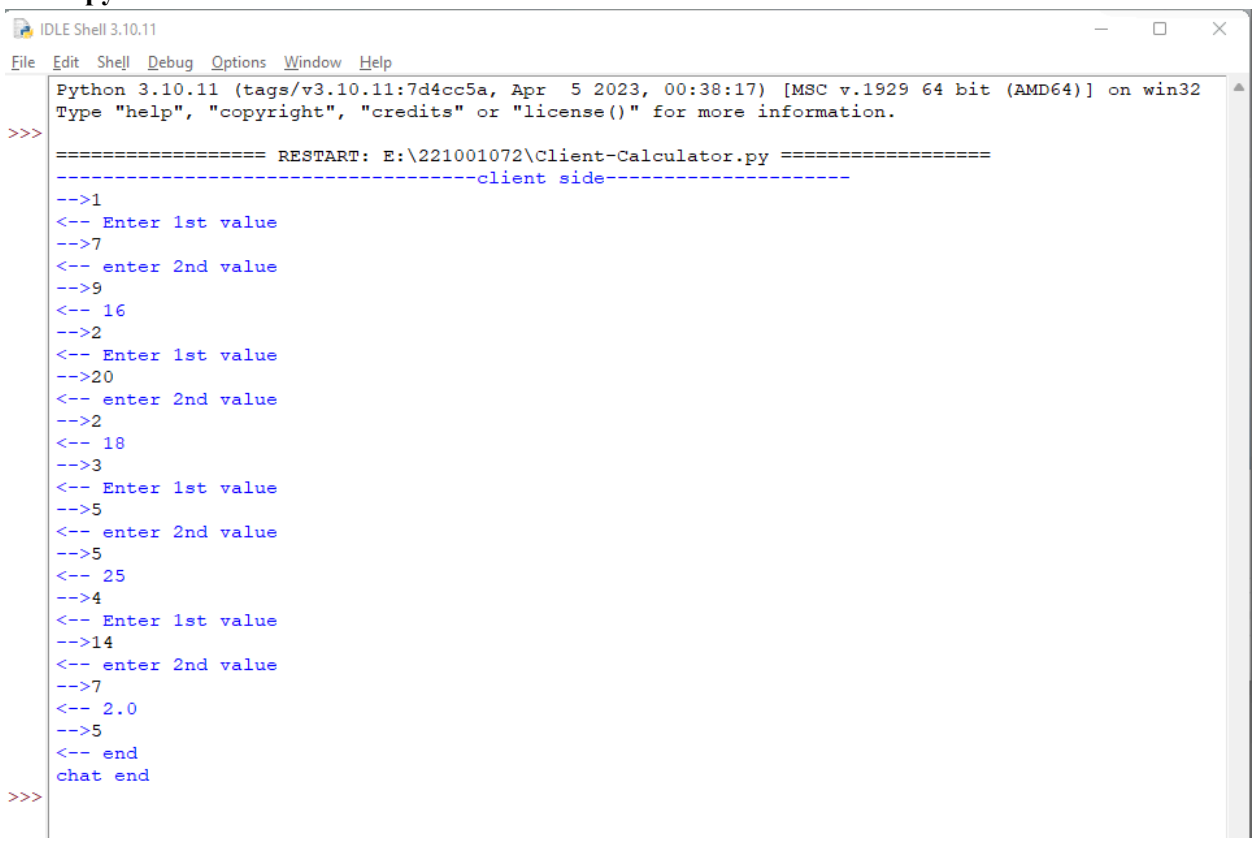
```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.connect(("127.0.0.1",8000))
print("---------------------- client side--------------
") while True:
    msg=input("-->",)
    s.send(msg.encode('utf-8')
    )
    data=s.recv(100).decode()
    print("<--",data)
    if(msg=="5"):
        print("chat
        end") break
    for i in range(1,3):
        msg1=input("-->",
        )
        s.send(msg1.encode('utf-8')
        ) data=s.recv(100).decode()
        print("<--",data)
```

Output:

Server.p



Client.py



Result:

Thus the Python program for Calculator Application using TCP has been executed successfully.

Exp. No. 09	Dictionary Application Using UDP Socket
Date: 17.08.24	

Aim:

To develop a program for Dictionary Application using UDP in sockets.

Algorithm:

Step 1: Server creates a TCP socket, binds it to port 4000, and listens for incoming connections.

Step 2: Client creates a TCP socket and connects to the server at 127.0.0.1 on port 4000.

Step 3: Client prompts the user to enter a word and sends it to the server using send().

Step 4: Server receives the word, looks up its meaning from a predefined dictionary,

and sends the meaning back to the client using send().

Step 5: Client receives the meaning from the server, decodes it, and prints it.

Program:

Server.py

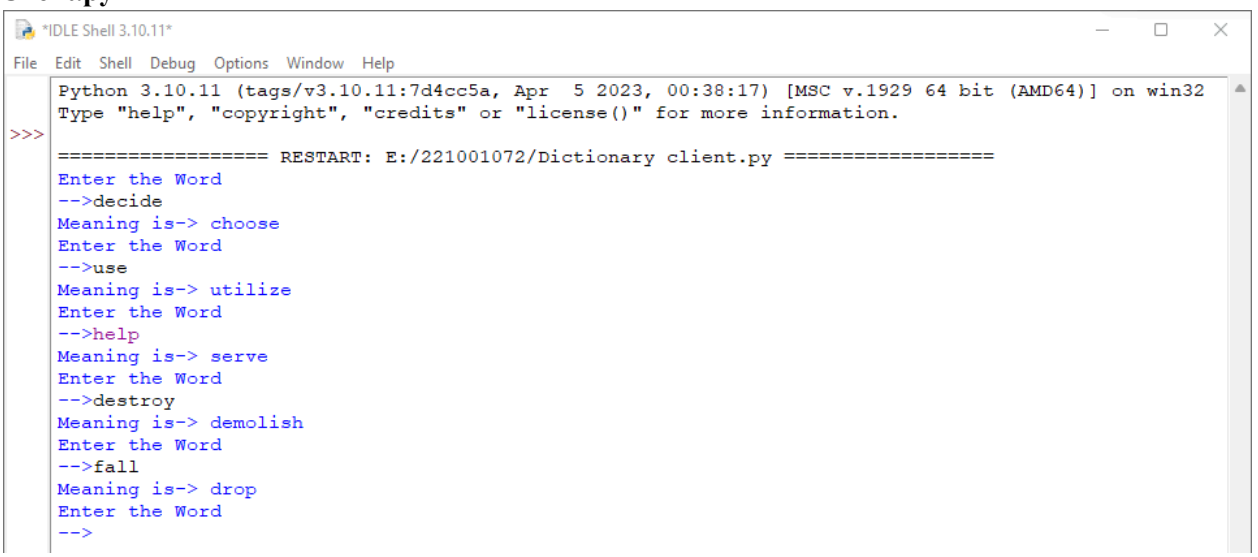
```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.bind(("",4000))
s.listen(5)
c,a = s.accept()
while True:
    data_received = c.recv(100).decode()
    a={'use':'utilize','destroy':'demolish','fall':'drop','decide':'choose','help':'serve'}
    data_tosend = a[data_received]
    c.send(data_tosend.encode('utf-8'))
```

Client.py

```
from socket import *
s = socket(AF_INET,SOCK_STREAM)
s.connect(("127.0.0.1",4000))
while True:
    print("Enter the Word")
    data_tosend=input("-->",)
    s.send(data_tosend.encode('utf'))
    data_received=s.recv(100).decode()
    print("Meaning
    is->",data_received)
```

Output:

Client.py



Result:

Thus the Python program for Calculator Application using TCP has been executed successfully.

Exp. No. 10	Simple Ping Application Using UDP Socket
Date: 23.08.24	

Aim:

To develop a program for Simple Ping Application Using UDP in Sockets.

Algorithm:

Step 1: Server creates a UDP socket, binds it to 127.0.0.1 and port 12345,

and listens for incoming messages.

Step 2: Client creates a UDP socket and sends a Ping message to the server at 127.0.0.1 on port 12345.

Step 3: Server receives the Ping message, prints the message and the client's address,

then responds with a Ping reply.

Step 4: Client receives the Ping reply from the server, calculates the round-trip time,

and prints the reply along with the time taken.

Step 5: If the client does not receive a reply within the specified timeout (2 seconds),

it handles the socket timeout exception and prints an error message.

Step 6: Both client and server continue to repeat the process of sending

and receiving messages until the program is manually stopped.

Program:

Server.py:

```
import socket
def start_server(host='127.0.0.1', port=12345):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.bind((host, port))
        print(f"UDP Server running on {host}:{port}")
        while True:
            data, addr = s.recvfrom(1024)
            print(f"Received message from {addr}: {data.decode()}")
            s.sendto(b'Ping', addr)
if __name__ == "__main__":
    start_server()
```

Client.py:

```
import
socket
import time
```

```
def ping_server(host='127.0.0.1', port=12345):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        try:
            s.settimeout(2)
            start = time.time()
            s.sendto(b'Ping', (host, port))
            data, addr = s.recvfrom(1024)
            end = time.time()
            print(f"Received {data.decode()} from {addr} in {end - start:.2f} seconds")
        except socket.timeout:
```

```
        print("Request timed
out") if __name__ == "__main__":
    ping_server()
```

Output:

Server.py

```
============================== RESTART: D:/server.py ==========================
===
UDP Server running on 127.0.0.1:12345
Received message from ('127.0.0.1', 63929): Ping
Received message from ('127.0.0.1', 63922): Ping
Received message from ('127.0.0.1', 56820): Ping
```

Client.py

```
>>>
============================== RESTART: D:/client.py ==========================
===
Received Ping from ('127.0.0.1', 12345) in 0.01 seconds
>>>
============================== RESTART: D:/client.py ==========================
Received Ping from ('127.0.0.1', 12345) in 0.07 seconds
>>>
============================== RESTART: D:/client.py ==========================
Received Ping from ('127.0.0.1', 12345) in 0.01 seconds
>>>
```

Result:

Thus the Python program for Simple Ping Application Using UDP in Sockets has been executed successfully.

Exp. No. 11	Implementation of Packet Sniffing Using Raw Socket
Date: 24.08.24	

Aim:

To develop a program for Implementation of Packet Sniffing Using Raw Sockets.

Algorithm:

Step 1: Import the necessary library.
Step 2: Capture Network Packet
Step 3: Parse Ethernet Header
Step 4: Parse IP Header
Step 5: Display Parsed Information

Program:

```
import socket
import struct
import
binascii
s = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_IP)
s.bind(("127.0.0.1", 0))
packet = s.recvfrom(65565)
print(packet)
ethernet_header = packet[0][0:14]
eth_header = struct.unpack("!6s6s2s", ethernet_header)
print("ETHERNET HEADER")
print("***************")
print("Destination Address")
print(binascii.hexlify(eth_header[0]))
print("Source Address")
print(binascii.hexlify(eth_header[1]))
print("Type")
print(binascii.hexlify(eth_header[2]))
print("IP HEADER")
print("********")
ipheader = packet[0][14:34]
ip_header = struct.unpack("!12s4s4s", ipheader)
print("Destination Address")
print(socket.inet_ntoa(ip_header[1]))
print("Source Address")
print(socket.inet_ntoa(ip_header[2]))
```

Output:

Result:

Thus the Python program for Implementation of Packet Sniffing Using Raw Sockets has been executed successfully.

Exp. No. 12	Error Correction at Data Link Layer
Date: 30.08.24	

Aim:

To write a python program to implement error detection and correction using the Hamming Code concept.

Algorithm:

Encoding:
Step 1: Accept a 4-bit binary string as input data
Step 2: Check if the input data consists of exactly 4 bits and contains only 0's and 1's
Step 3: Convert the binary string into a list of integers
Step 4: Use a predefined generator matrix G for Hamming (7,4) encoding
Step 5: Multiply the data bits by the generator matrix G to calculate the 7-bit encoded data. Perform bitwise operations and store the result in a new list
Step 6: Return the 7-bit encoded data as a string

Decoding:
Step 1: Accept a 7-bit encoded binary string
Step 2: Check if the input encoded data consists of exactly 7 bits and contains only 0's and 1's
Step 3: Convert the binary string into a list of integers
Step 4: Use a predefined parity-check matrix H for Hamming (7,4)
Step 5: Multiply the received bits by the matrix H to calculate the syndrome, which helps detect errors
Step 6: Convert the syndrome to a decimal value to determine the error position, if any. If an error is found, correct the erroneous bit
Step 7: Extract the original 4 data bits from the corrected 7-bit sequence
Step 8: Return the decoded 4-bit data and the error position (if any)

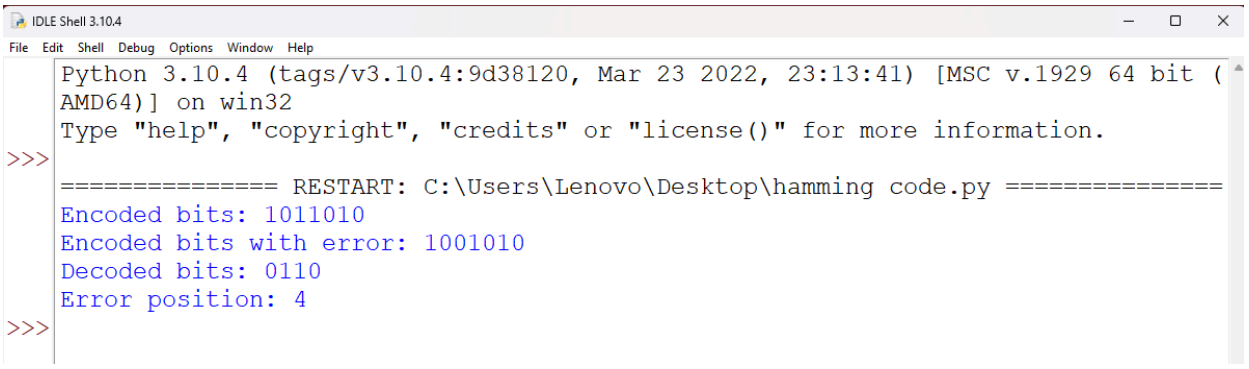
Program:

```
def encode_hamming_7_4(data_bits):
    if len(data_bits) != 4 or any(bit not in '01' for bit in data_bits):
        raise ValueError("Data bits must be a 4-bit binary string.")
    d = list(map(int, data_bits))
    G = [
        [1, 1, 0, 1, 0, 0, 0],
        [1, 0, 1, 1, 0, 0, 0],
        [1, 0, 0, 1, 1, 0, 0],
        [0, 1, 1, 1, 0, 0, 0],
        [0, 1, 0, 1, 1, 0, 0],
        [0, 0, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 1, 1, 1]
    ]
    encoded_bits = [0] * 7
    for i in range(7):
        encoded_bits[i] = sum(G[i][j] * d[j] for j in range(4)) % 2
    return ''.join(map(str, encoded_bits))
def decode_hamming_7_4(encoded_bits):
    if len(encoded_bits) != 7 or any(bit not in '01' for bit in encoded_bits):
        raise ValueError("Encoded bits must be a 7-bit binary string.")
```

```
r = list(map(int, encoded_bits))
```

```
    H = [
        [1, 1, 1, 0, 0, 0, 0],
        [1, 1, 0, 1, 0, 0, 0],
        [1, 0, 1, 0, 1, 0, 0],
        [0, 1, 1, 0, 0, 1, 0],
        [0, 1, 0, 1, 0, 0, 1],
        [0, 0, 1, 1, 1, 0, 0],
        [0, 0, 0, 0, 1, 1, 1]
    ]
    syndrome = [0] * 3
    for i in range(3):
        syndrome[i] = sum(H[i][j] * r[j] for j in range(7)) % 2
    error_pos = int(''.join(map(str, syndrome[::-1])), 2) - 1
    if error_pos >= 0:
        r[error_pos] ^= 1
    data_bits = [r[2], r[4], r[5], r[6]]
    return ''.join(map(str, data_bits)), error_pos
data_bits = '1010'
encoded = encode_hamming_7_4(data_bits)
print(f"Encoded bits: {encoded}")
encoded_with_error = encoded[:2] + ('1' if encoded[2] == '0' else '0') + encoded[3:]
print(f"Encoded bits with error: {encoded_with_error}")
decoded, error_position = decode_hamming_7_4(encoded_with_error)
print(f"Decoded bits: {decoded}")
print(f"Error position: {error_position}" if error_position >= 0 else "No error detected")
```

Output:



Result:

Thus the Python program to implement error detection and correction using Hamming Code has been executed successfully.

Exp. No. 13	Sliding Window Protocol
Date: 31.08.24	

Aim:

Write a program to implement flow control at the data link layer using SLIDING WINDOW PROTOCOL. Simulate the flow of frames from one node to another.

Algorithm:

Step 1: Initialize frames from 1 to total_frames, acknowledged array to False, and set sender_window_start to 0.
Step 2: While sender_window_start is less than total_frames, calculate window_end and display current window.
Step 3: For each unacknowledged frame in the window, simulate sending the frame with a random delay.
Step 4: Simulate ACK reception for each frame with an 80% chance of success.
Step 5: Mark frames as acknowledged if ACK is received; otherwise, leave them unacknowledged.
Step 6: Slide the window by moving sender_window_start to the first unacknowledged frame.
Step 7: Repeat steps 2-6 until all frames are sent and acknowledged.

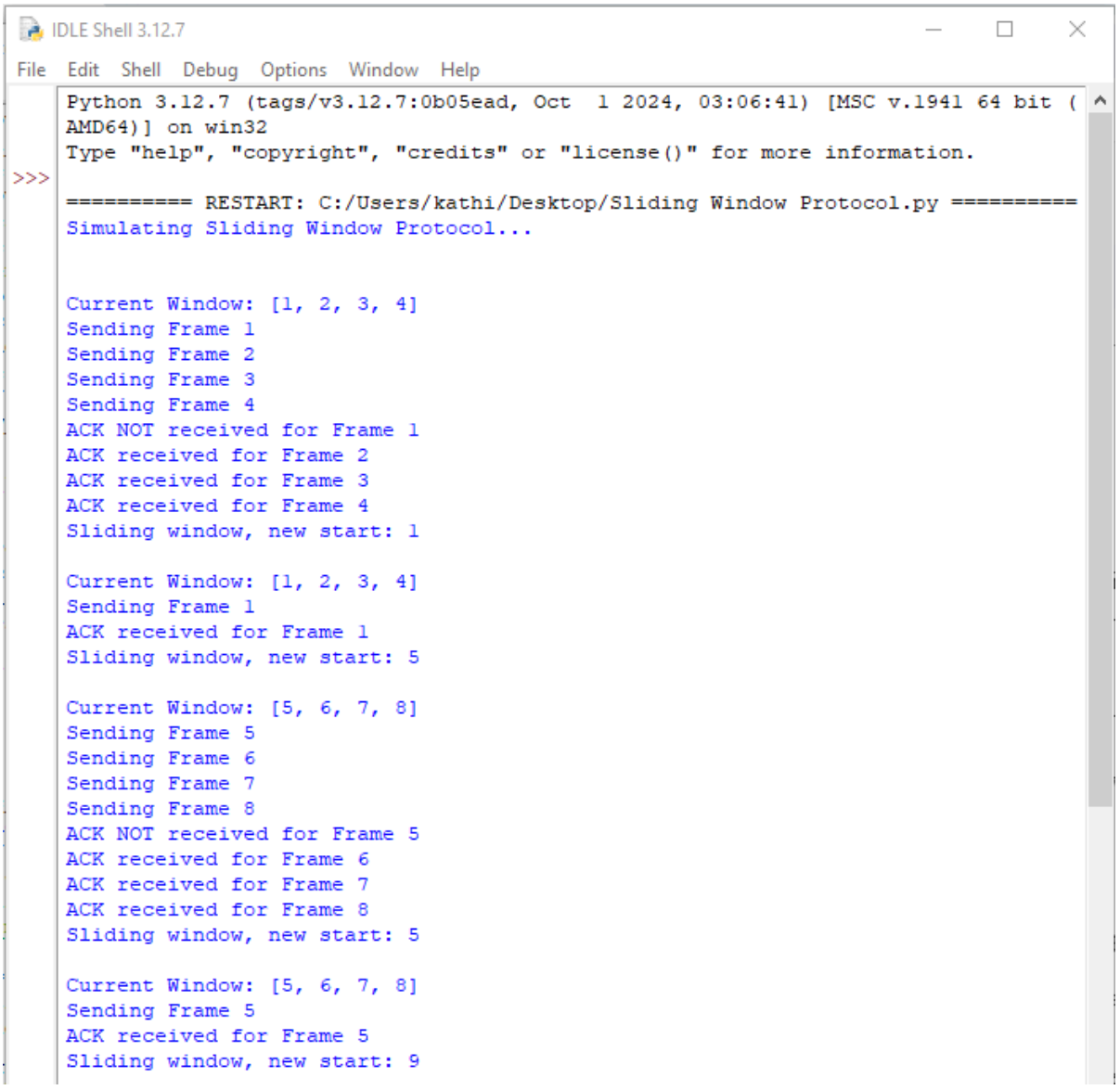
Program:

```
import time
import random
class SlidingWindowProtocol:
    def __init__(self, window_size, total_frames):
        self.window_size = window_size
        self.total_frames = total_frames
        self.frames = list(range(1, total_frames + 1))
        self.acknowledged = [False] * total_frames
        self.sender_window_start = 0
    def send_frame(self, frame_number):
        print(f"Sending Frame {frame_number}")
        time.sleep(random.uniform(0.1, 0.3))
    def receive_ack(self, frame_number):
        return random.random() < 0.8
    def send_frames(self):
        while self.sender_window_start < self.total_frames:
            window_end = min(self.sender_window_start + self.window_size, self.total_frames)
            print(f"\nCurrent Window: {self.frames[self.sender_window_start:window_end]}")
            for i in range(self.sender_window_start, window_end):
                if not self.acknowledged[i]:
                    self.send_frame(self.frames[i])
            self.receive_acks()
    def receive_acks(self):
        window_end = min(self.sender_window_start + self.window_size, self.total_frames)
        for i in range(self.sender_window_start, window_end):
            if not self.acknowledged[i]:
                if self.receive_ack(self.frames[i]):
                    print(f"ACK received for Frame {self.frames[i]}")
                    self.acknowledged[i] = True
                else:
                    print(f"ACK NOT received for Frame {self.frames[i]}")
```

```
        self.slide_window()
```

```
    def slide_window(self):
                                        while self.sender_window_start < self.total_frames and
self.acknowledged[self.sender_window_start]:
            self.sender_window_start += 1
        print(f"Sliding window, new start: {self.sender_window_start +
1}") if _name_== "_main_":
    window_size = 4
    total_frames = 10
    print("Simulating Sliding Window Protocol...\n")
    protocol = SlidingWindowProtocol(window_size, total_frames)
    protocol.send_frames()
    print("\nAll frames sent and acknowledged!")
```

Output:



```
Current Window: [9, 10]
Sending Frame 9
Sending Frame 10
ACK NOT received for Frame 9
ACK NOT received for Frame 10
Sliding window, new start: 9

Current Window: [9, 10]
Sending Frame 9
Sending Frame 10
ACK received for Frame 9
ACK received for Frame 10
Sliding window, new start: 11

All frames sent and acknowledged!
>>>
```

Result:

Thus the Python program to implement Sliding Window Protocol has been executed

successfully.

Exp. No. 14	Study of CISCO Tracker Installation Process
Date: 06.09.24	

Aim:

To study the Packet tracer tool Installation and User Interface Overview.

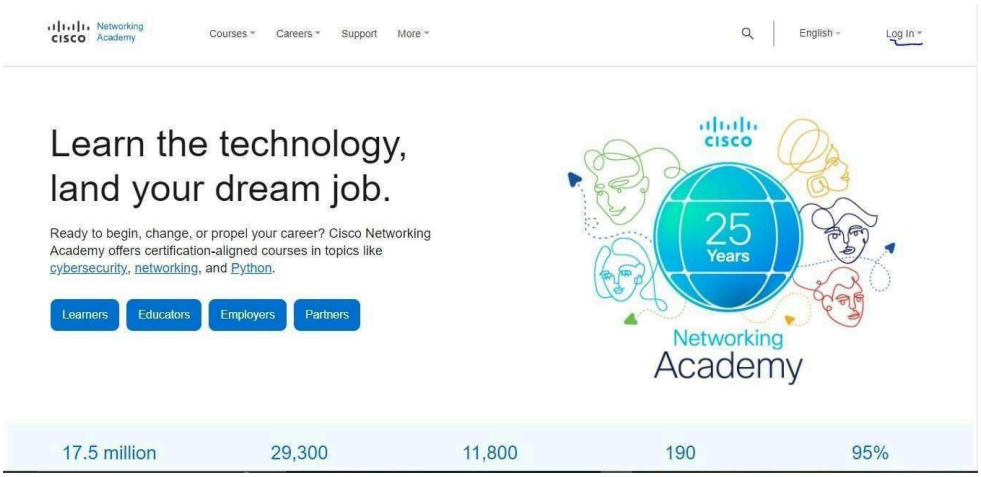
Introduction:

A simulator, as the name suggests, simulates network devices and its environment. Packet Tracer is an exciting network design, simulation and modelling tool.

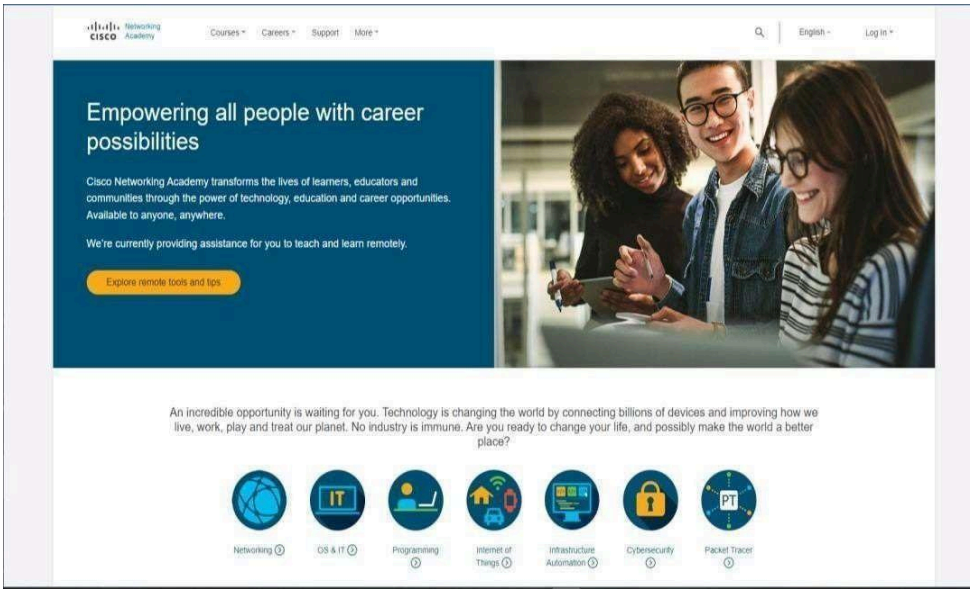
1. It allows you to model complex systems without the need for dedicated equipment.
2. It helps you to practice your network configuration and troubleshooting skills via computer or an Android or iOS based mobile device.
3. It is available for both the Linux and Windows desktop environments.
4. Protocols in Packet Tracer are coded to work and behave in the same way as they would on real hardware.

Installing Packet Tracker:

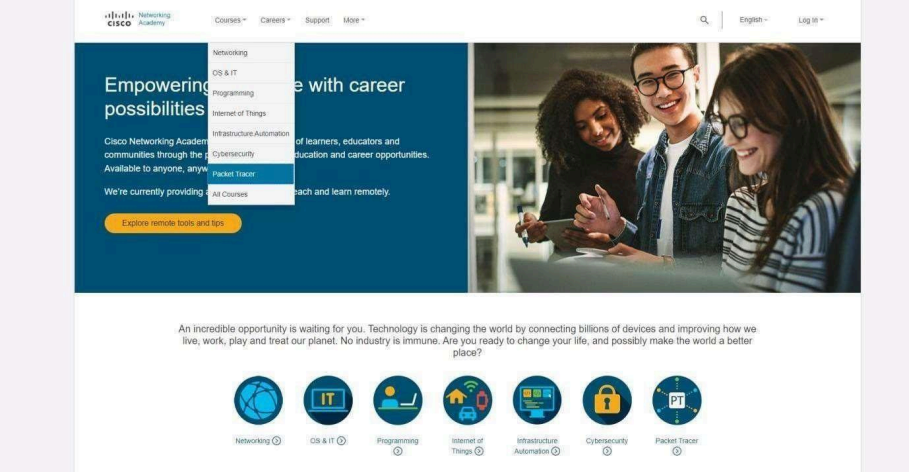
Step 1: To download packet Tracer, go to https://www.netacad.com and log in with your credentials.



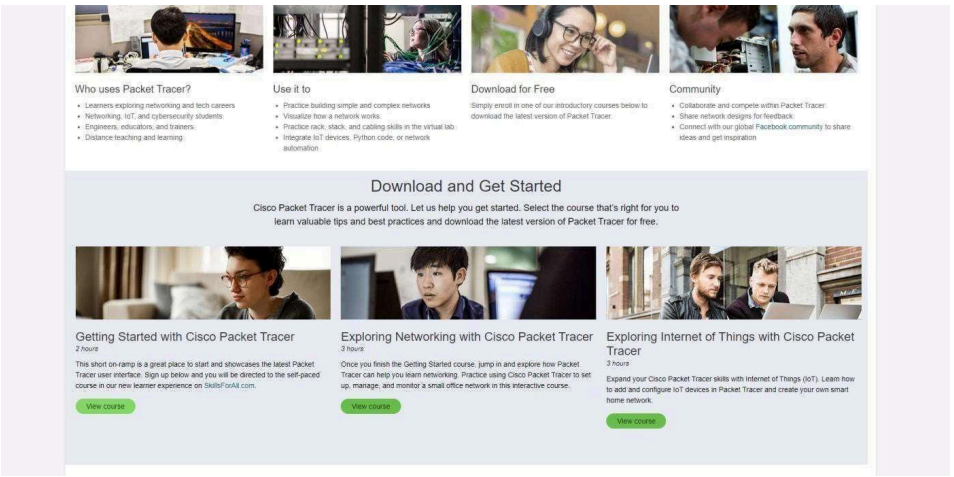
Step 2: By clicking on to link https://www.netacad.com ,a page appears as below.



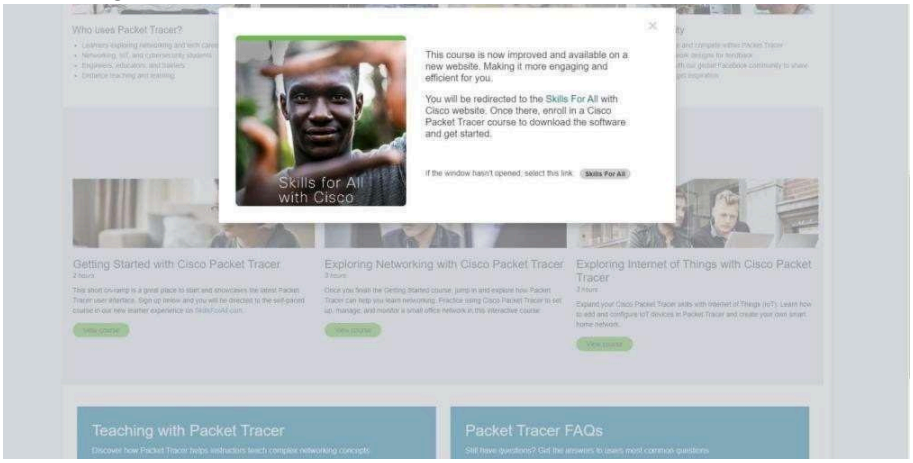
Step 3: As represented as in the below figure choose course -->Packet Tracer.



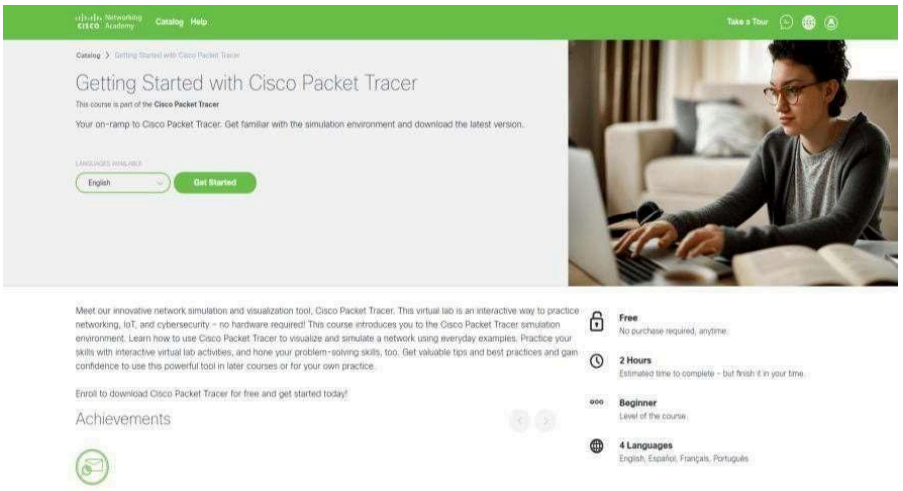
Step 4: By choosing, “Packet Tracer”. The following screen appears as shown in the figure below.



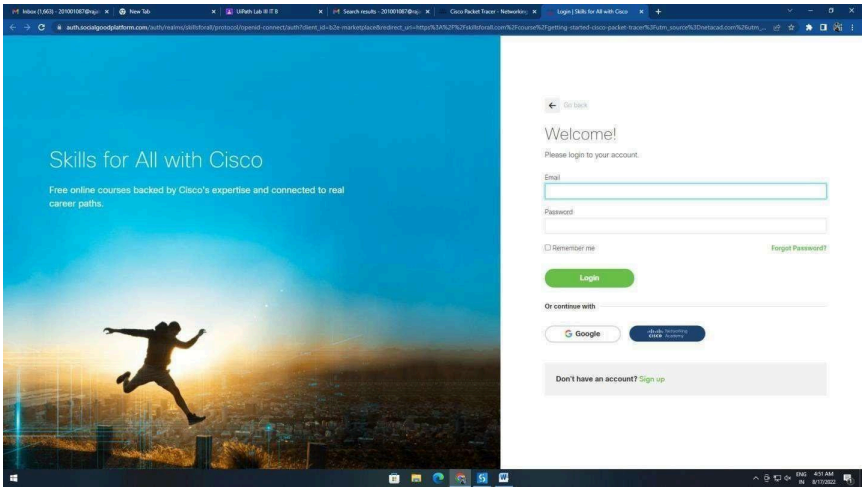
Step 5: By choosing, “getting started with Cisco Packet Tracer”. The following screen appears as shown in the figure below.



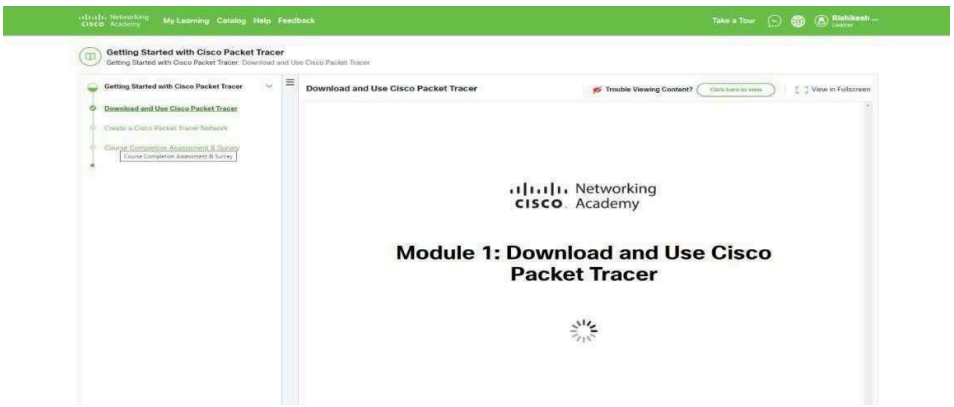
Step 6: By clicking on “skill for all” a new page appears as represented below.



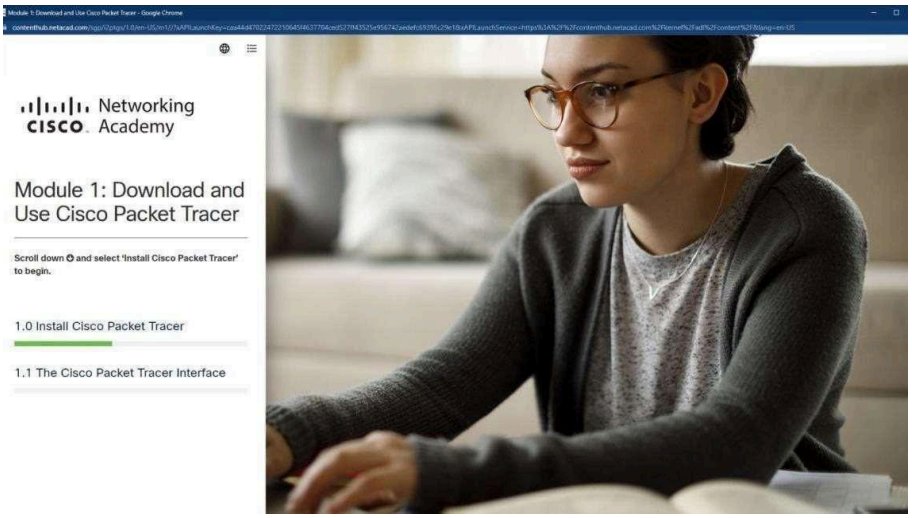
Step 7: Press “get started”. A new screen appears as represented below



Step 8: Login with your Google account or by sign up e-mail and password. Its opens in new screen shown as below.

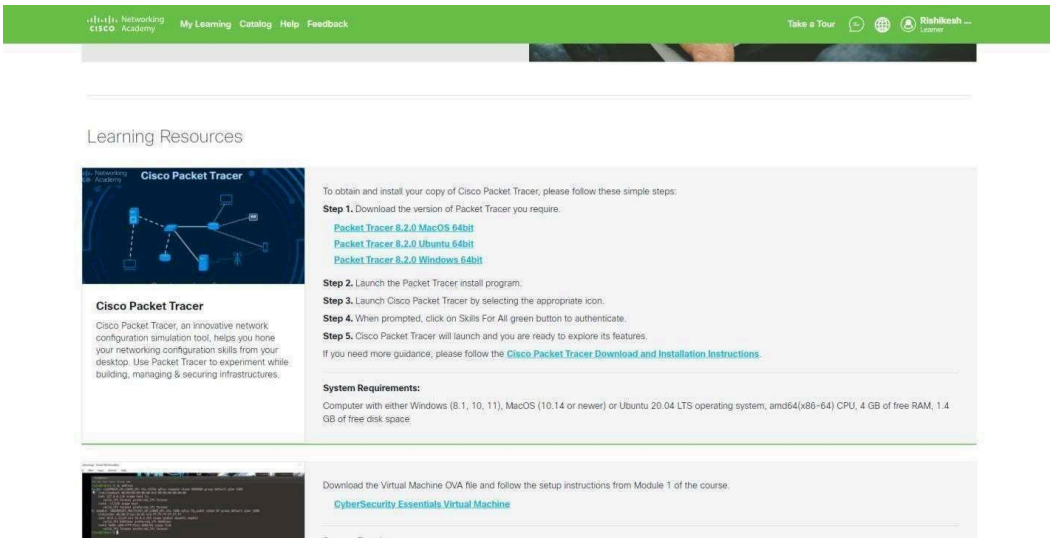


Step 9: By clicking “click here to view”, a new page opens as it is shown below.

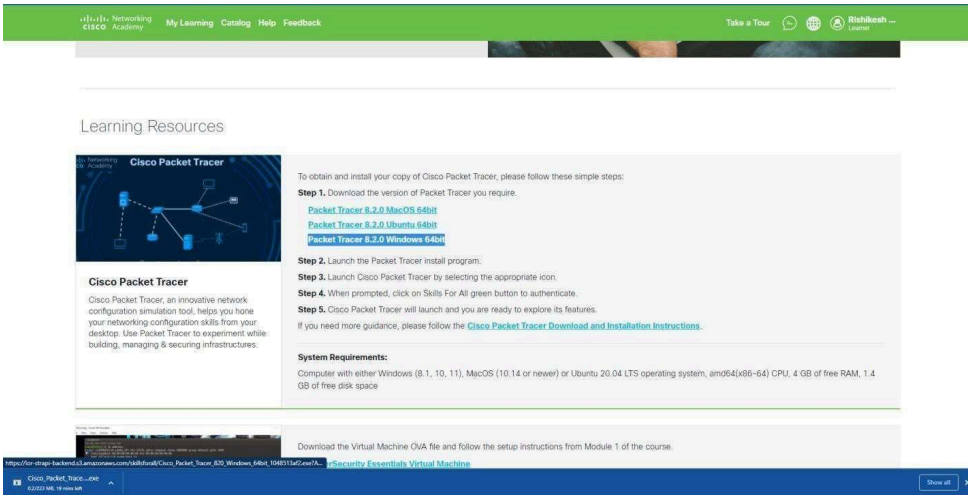


Step 10: By clicking on “1.0 Install Cisco Packet Tracer”, a new page occurs as shown in the figure below.

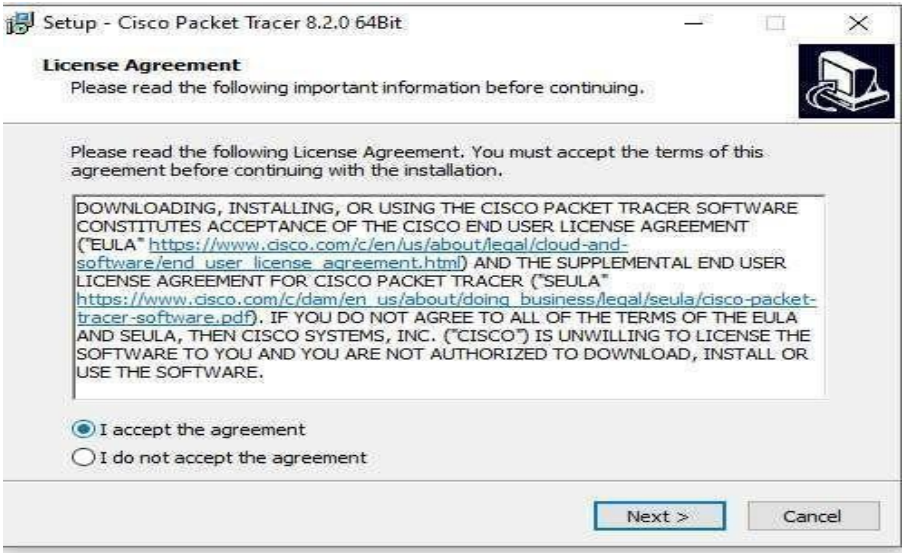
Step 11: On clicking the link- https://skillsforall.com/resources/lab-downloads a new window opens showing the versions of Packet Tracer as required as shown in the below figure.



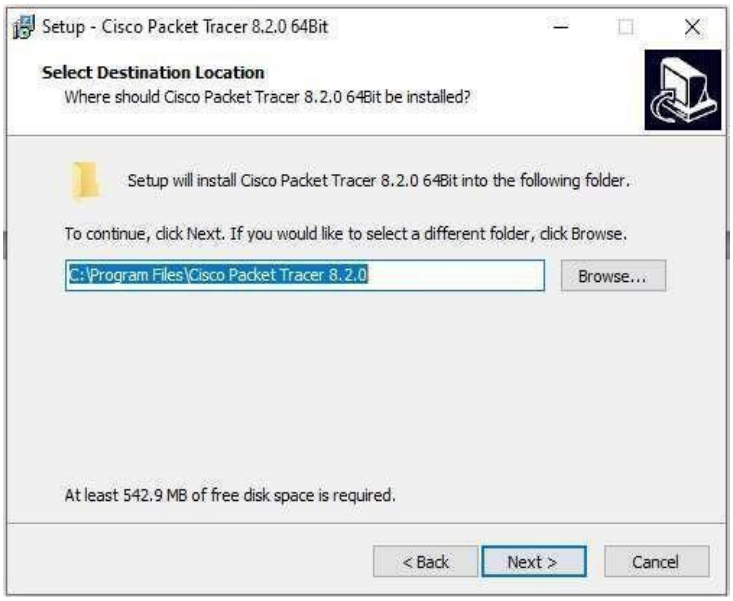
Step 12: Click on the "Packet Tracer 8.2.0 Windows 64bit" and the download will be started automatically.



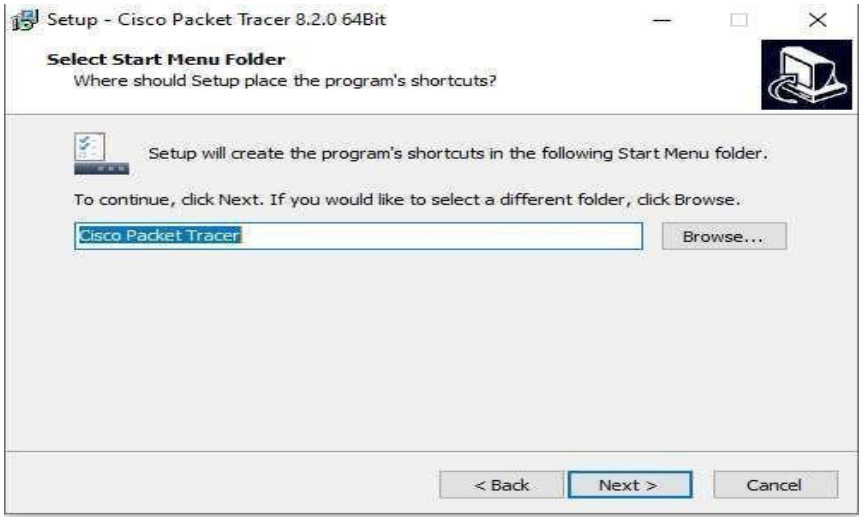
Step 13: After the download is completed, open the packet tracer setup file and agree the condition (License Agreement) and press next.

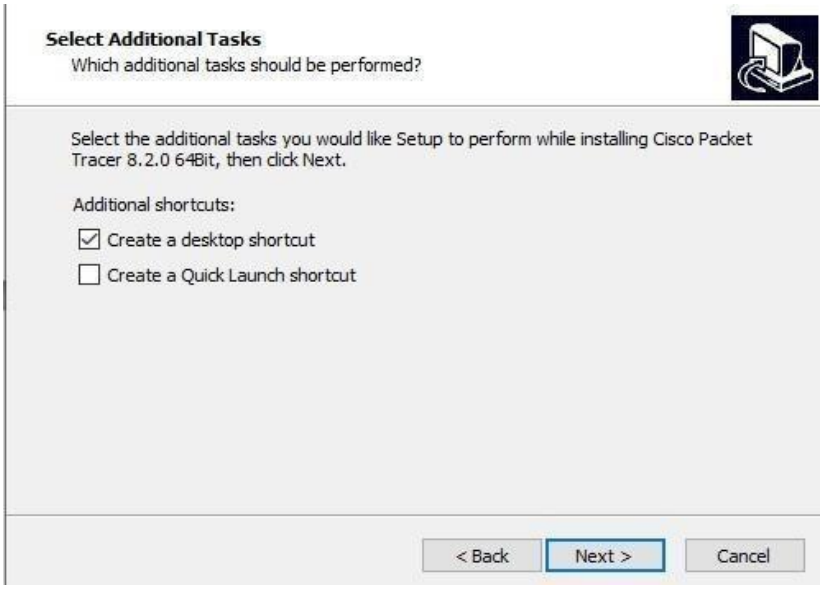


Step 14: Select the file path where the Cisco packet tracer is to be installed as represented below and enternext.

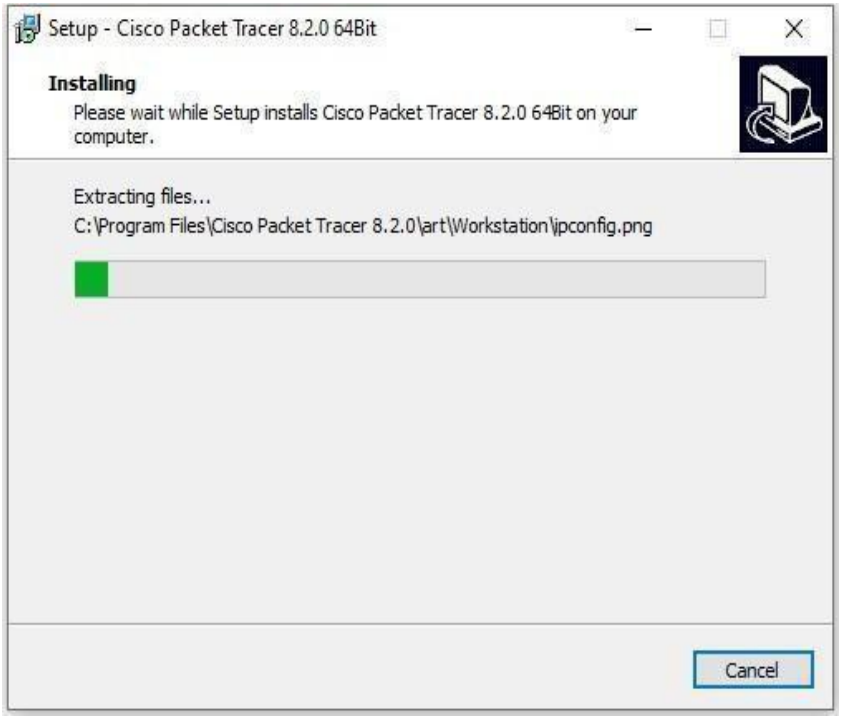


Step 15: Choose shortcut as per your requirement(eg.desktop) and enter next.



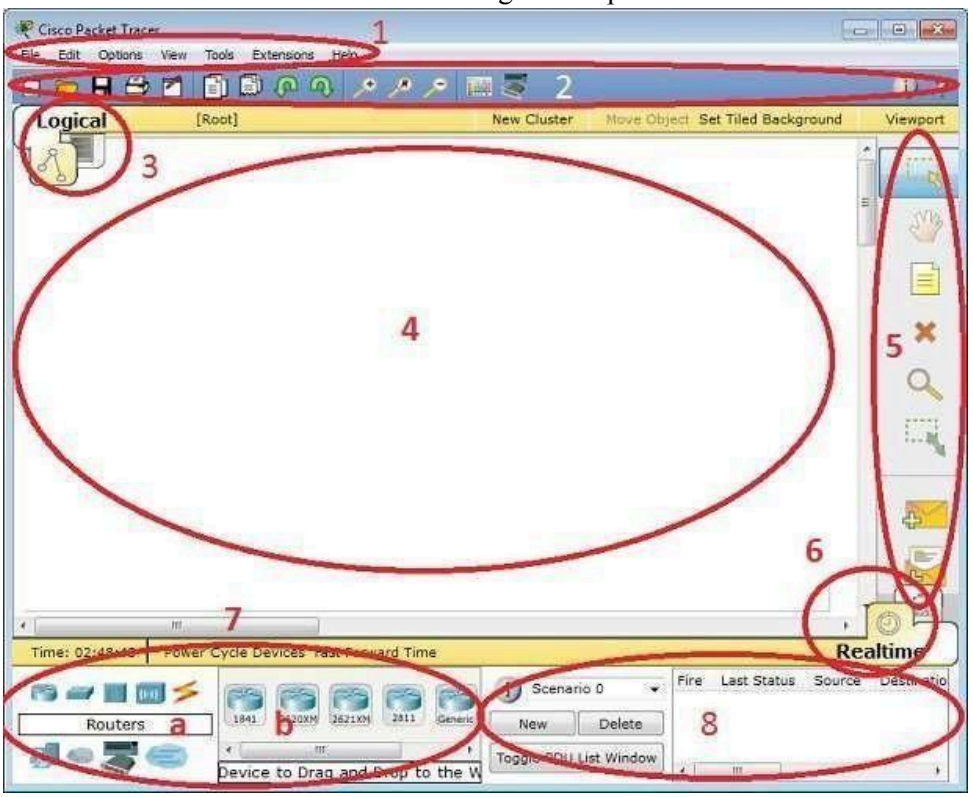


Step 16: Wait for a few minutes until the Cisco Packet Tracer gets installed.



User Interface Overview:

The layout of Packet Tracer is divided into several components. The components of the Packet Tracer interface are as follows: match the numbering with explanations.



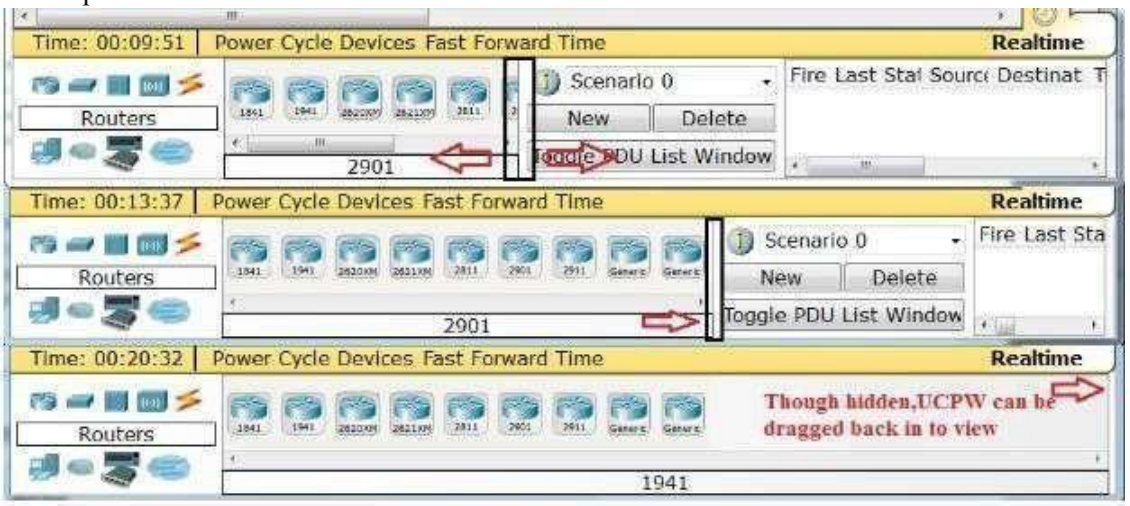
1. **Menu bar** – This is a common menu found in all software applications; it is used to open, save, print, change preferences, and so on.
2. **Main toolbar** – This bar provides shortcut icons to menu options that are commonly accessed, such as open, save, zoom, undo, and redo, and on the right-hand side is an icon for entering network information for the current network.
3. **Logical/Physical workspace tabs** – These tabs allow you to toggle between the Logical

and Physical work areas.

4. Workspace – This is the area where topologies are created and simulations are displayed.
5. **Common tools bar** – This toolbar provides controls for manipulating topologies, such as select, move layout, place note, delete, inspect, resize shape, and add simple/complex PDU.
6. **Real-time/Simulation** tabs – These tabs are used to toggle between the real and simulation modes. Buttons are also provided to control the time, and to capture the packets.
7. **Network component box** – This component contains all of the network and end devices available with Packet Tracer, and is further divided into two areas: Area 7a: Device-type selection box – This area contains device categories Area 7b: Device- specific selection box – When a device category is selected, this selection box displays the different device models within that category
8. **User-created packet box** – Users can create highly-customized packets to test their topology from this area, and the results are displayed as a list.

Workspaces and Modes:

Packet Tracer has two workspaces (Logical and Physical) and two modes (Realtime and Simulation). Upon startup, you are in the Logical Workspace in Realtime Mode. You can build your network and see it run in real time in this configuration. You can switch to Simulation Mode to run controlled networking scenarios. You can also switch to the Physical Workspace to arrange the physical aspects (such as the location) of your devices. Note that you view a simulation while you are in the Physical Workspace. You should return to the Logical Workspace after you are done in the Physical Workspace.



Result:

Thus, the installation of cisco packet tracer is successfully implemented.

Exp. No. 15	Creating a Simple Network with Two End Devices (Simple PDU)
Date: 13.09.24	

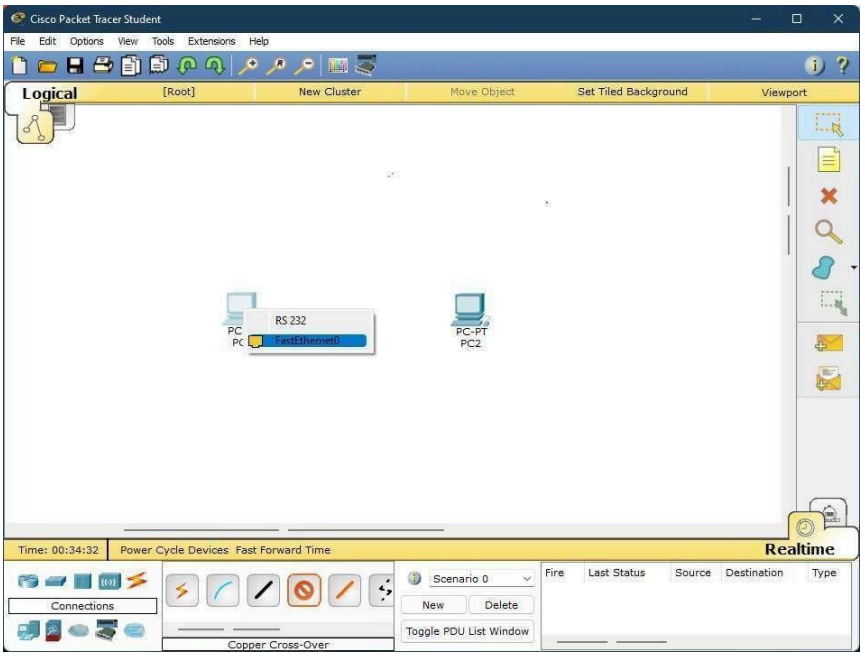
Aim:

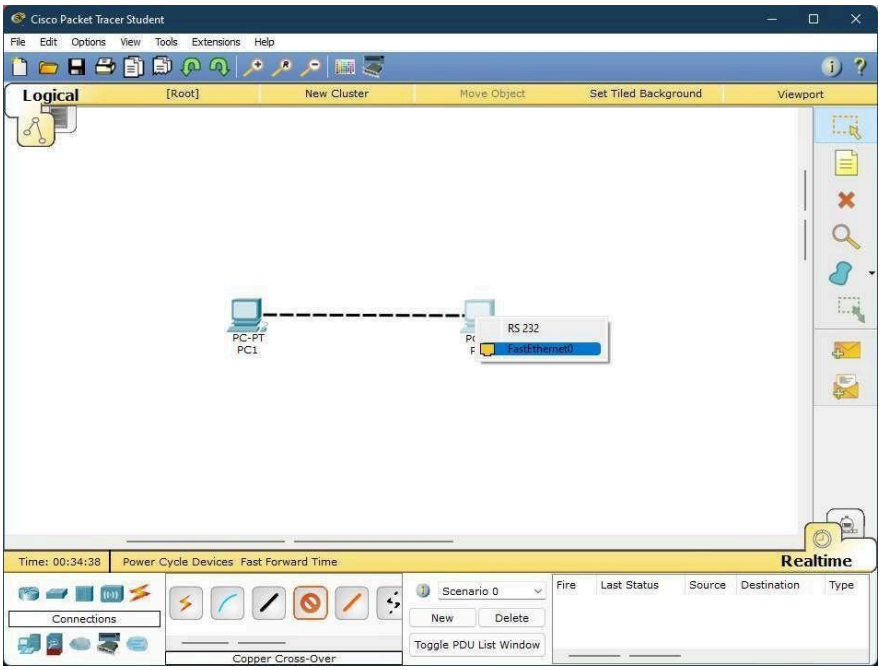
To install and execute CISCO packet for simple PDU

Procedure:

Creating and transmitting a simple PDU (Packet Data Unit) with Cisco Packet.

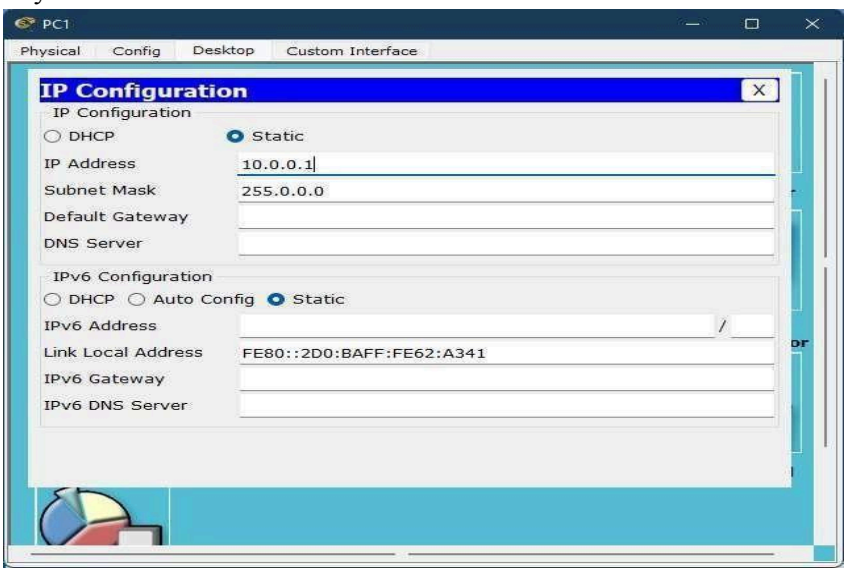
1. **Launch Cisco Packet Tracer:** Start by opening Cisco Packet on your computer.
2. **Create a Network Topology:** Build a simple network topology by dragging and dropping routers, switches and end devices (e.g. PCs) onto the workspace. Connect them using appropriate cables or connections.

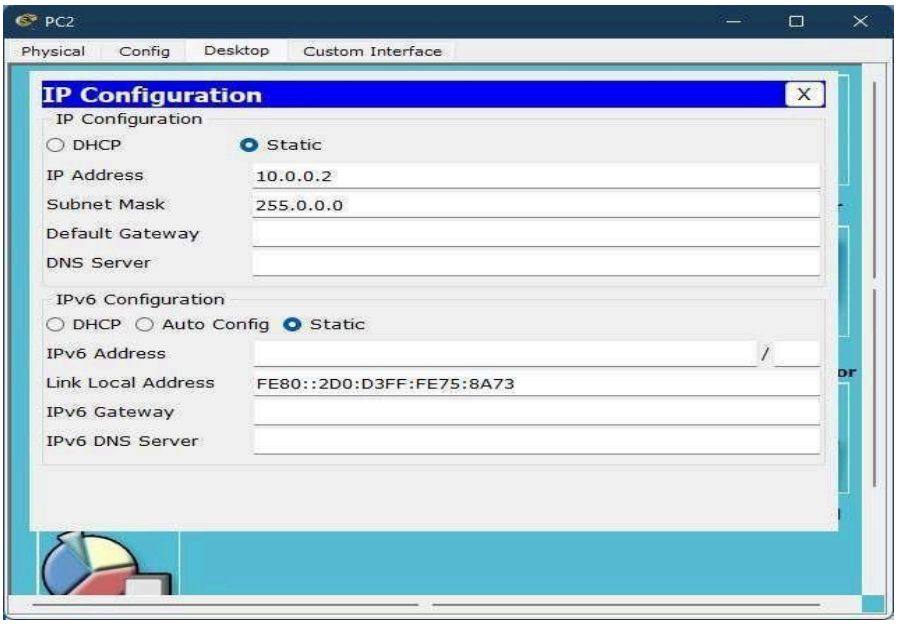




3. **Power on Devices:** Ensure that all devices in your network topology are powered on. You can do this by clicking on each device and selecting the "Power" option.
4. **Configure Device Interfaces:** Configure the interfaces of routers and switches with

appropriate IP addresses, subnet masks, and routing protocols (if needed) to establish connectivity in the network.

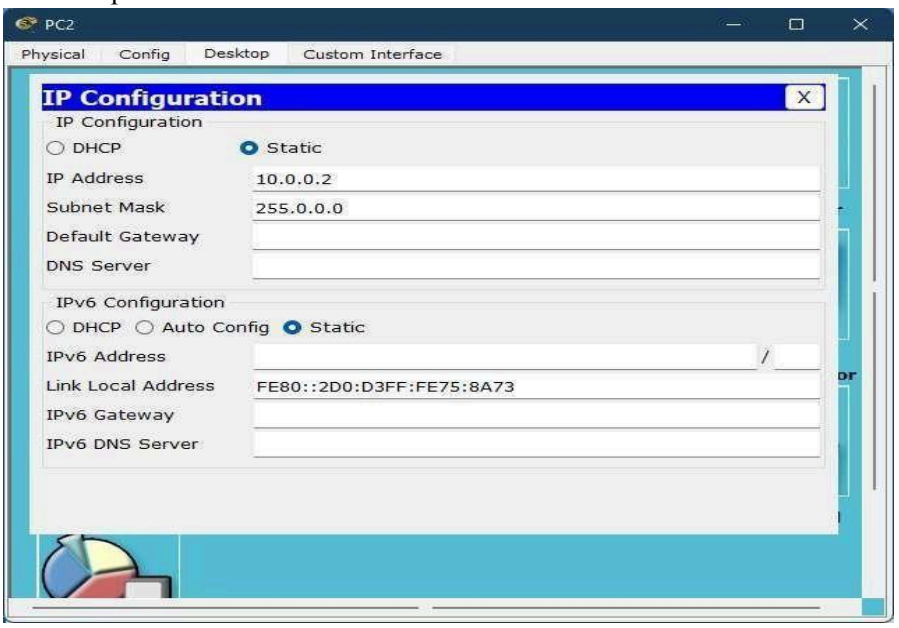


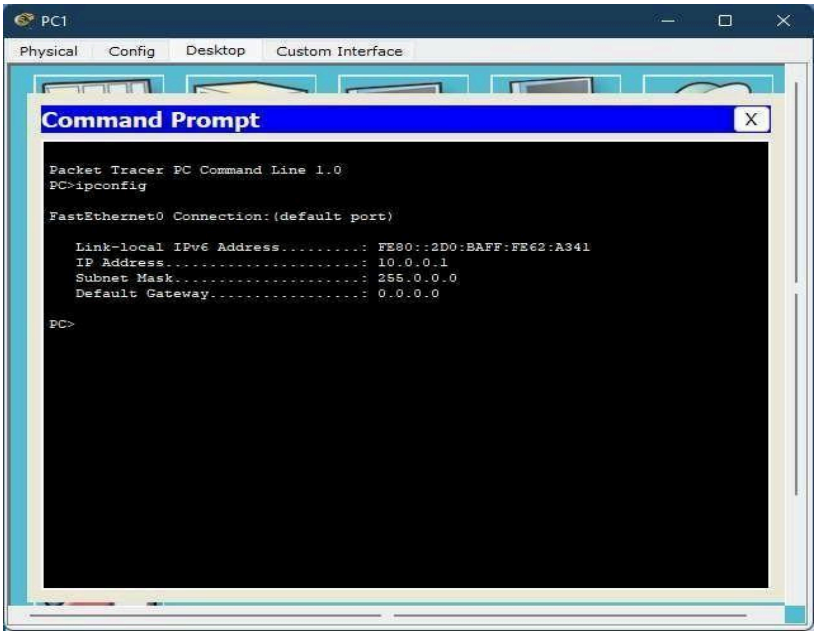


5. **Assign IP Addresses:** On the PCs in your network, assign IP addresses manually to ensure they can communicate with each other. This is crucial for generating a Simple PDU.
6. **Open Command Prompt or Terminal:** On one of the PCs, open the command prompt (Windows) or terminal (Linux). This will be your source device for sending the Simple PDU.

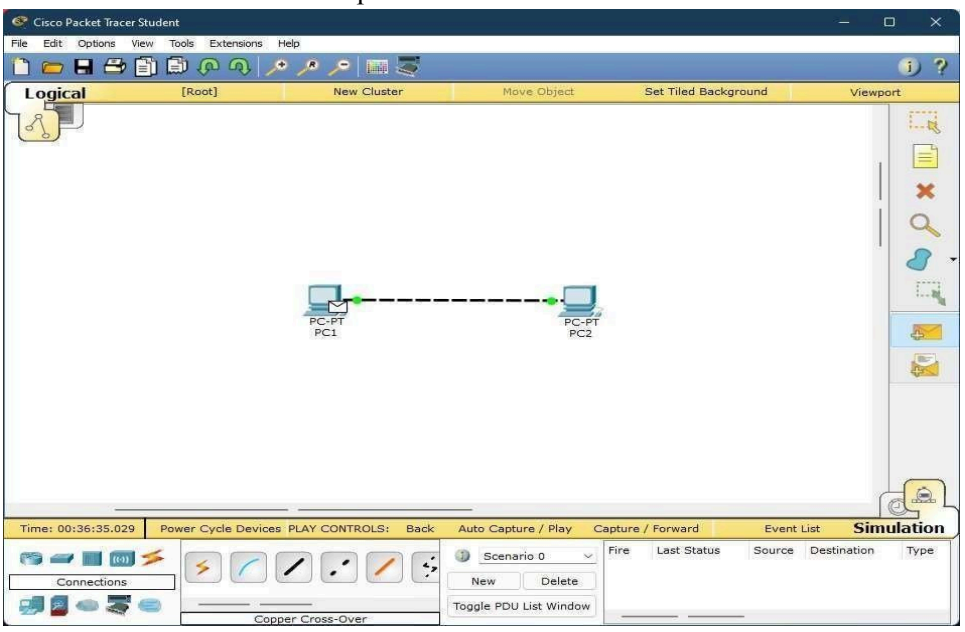
7. **Check Network Connectivity:** Before sending a Simple PDU, confirm that you can ping another device in your network. Use the `ping` command followed by the IP address of the destination device.

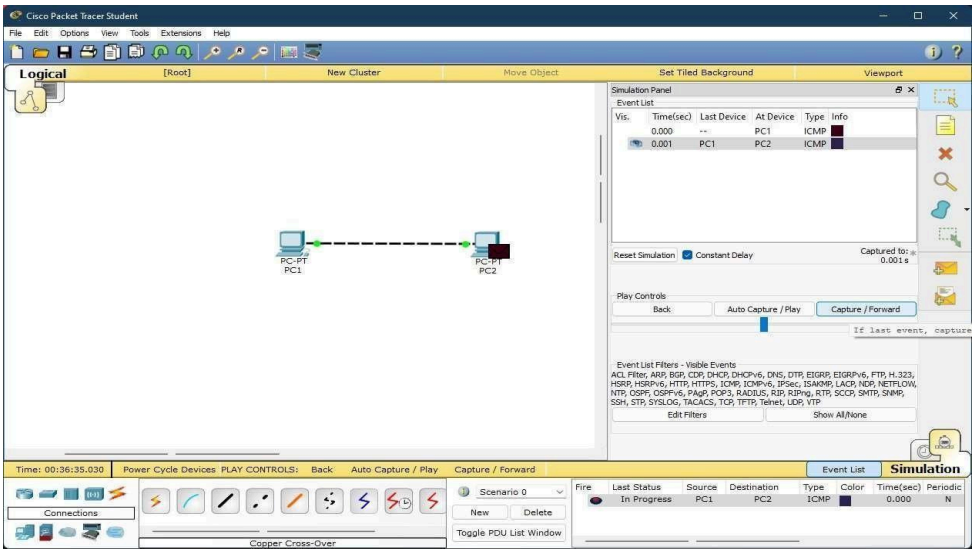
   For example:

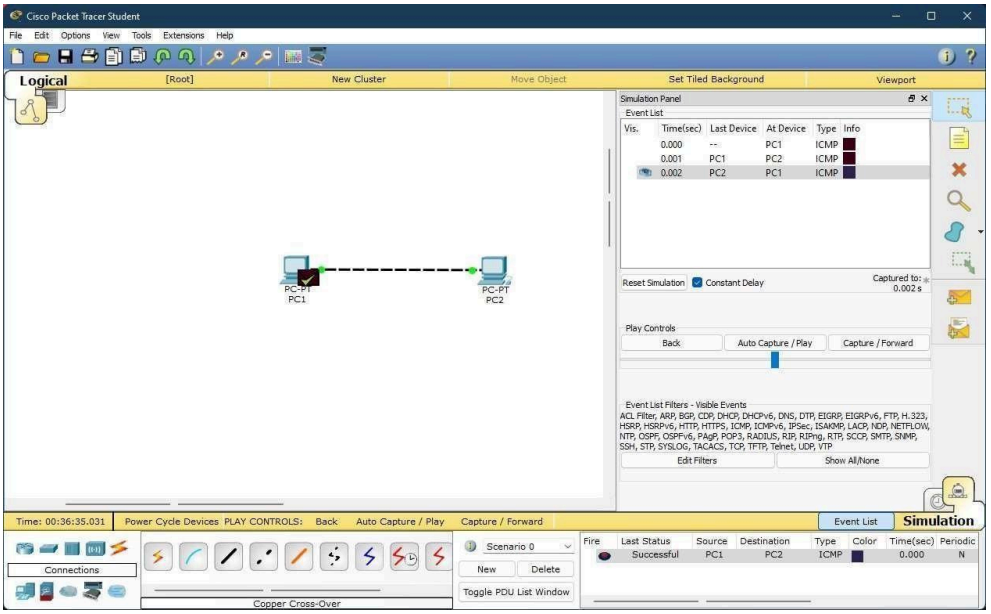




8. **Send:** Click to create and send the simple PDU







Result:

Thus the CISCO Packet for Simple PDU has been executed Successfully.

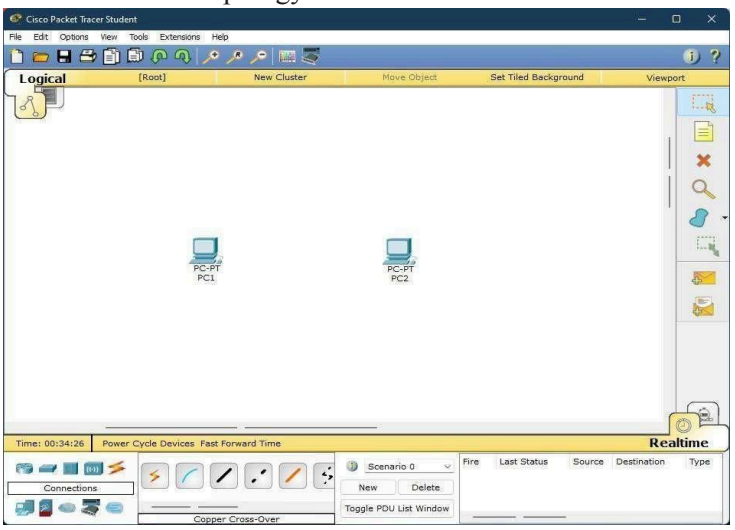
Exp. No. 16	Creating a Simple Network with Two End Devices (CPDU)
Date: 14.09.24	

Aim:

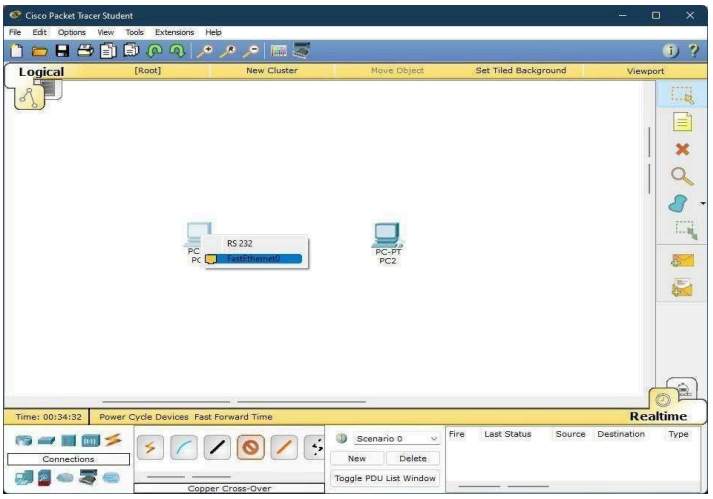
To create a simple network with two end devices with complex protocol data unit.

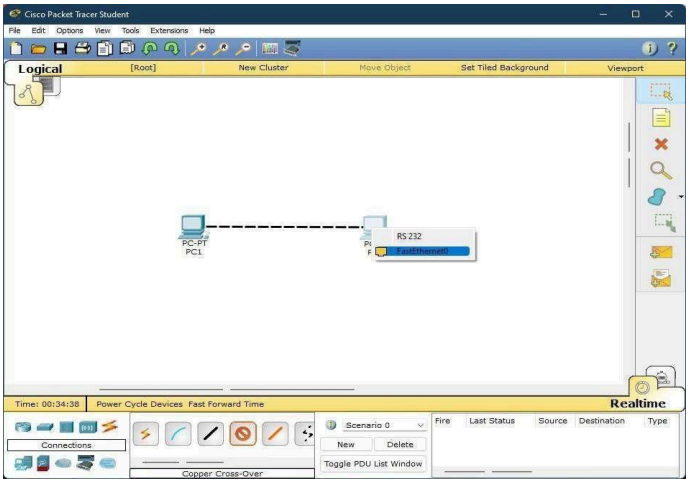
Procedure:

1. Open Cisco Packet Tracer and create a new network topology.
2. Add two PCs in network topology.

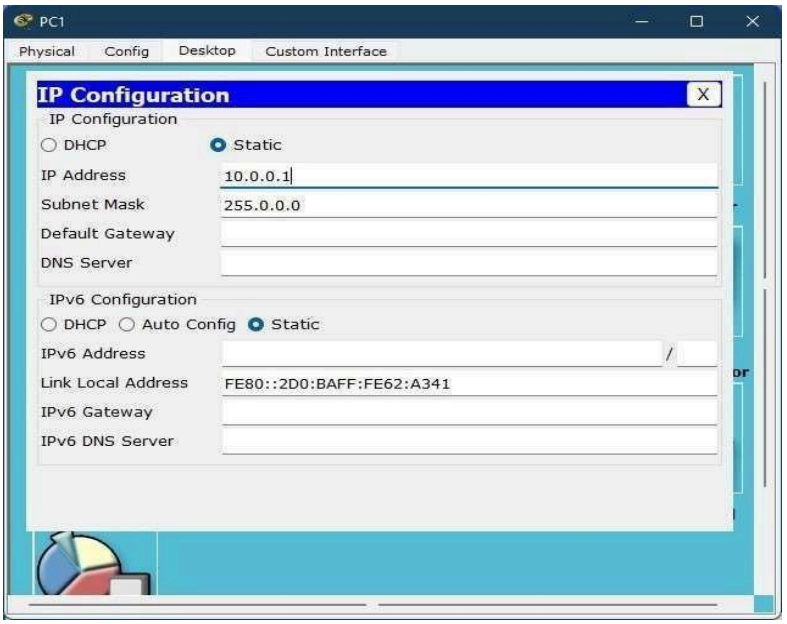


3. Connect the two PCs using a crossover cable.

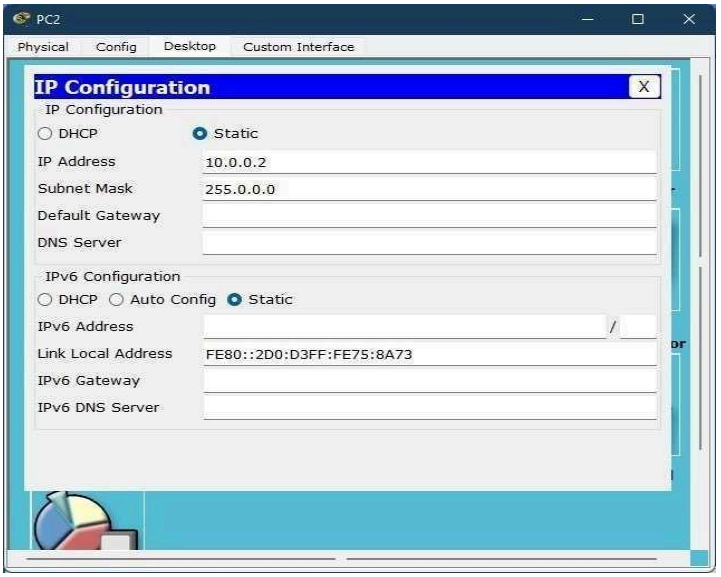




4. Configure first PC as below:

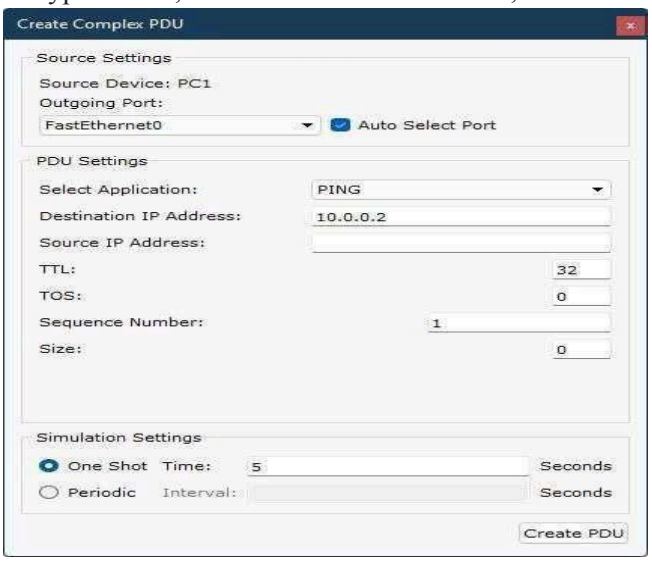


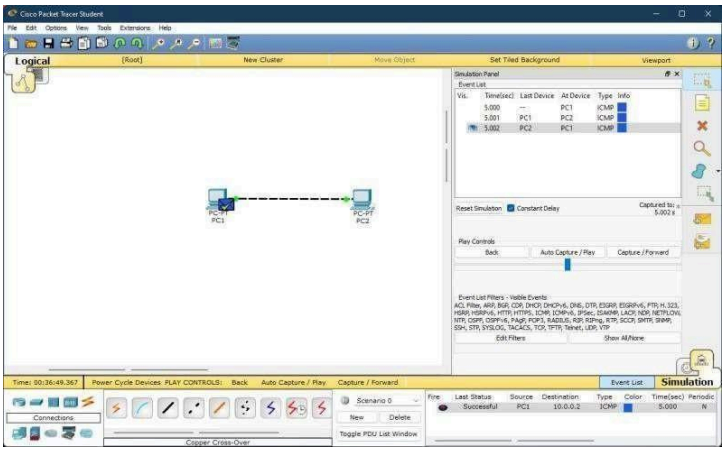
5. Configure second PC as below:



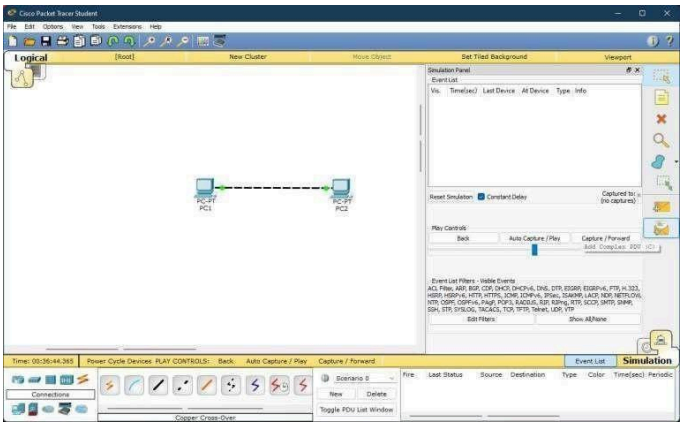
6. On one of the PCs, open the Add Complex PDU window.
7. Click the first PC so it will be the source device of the complex PDU, and then click the second PC.
8. The Create Complex PDU window will display.

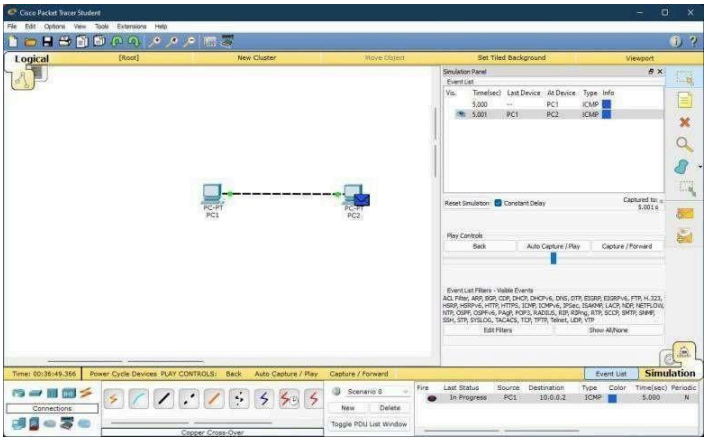
9. Configure the complex PDU settings as follows:
   Protocol type: ICMP, Source IP address: 10.0.0.1, Destination IP address: 10.0.0.2.





10. Click **Create** to send the complex PDU.





Result:

Thus the CISCO Packets for simple network with two end devices with complex protocol

data unit has been executed successfully.

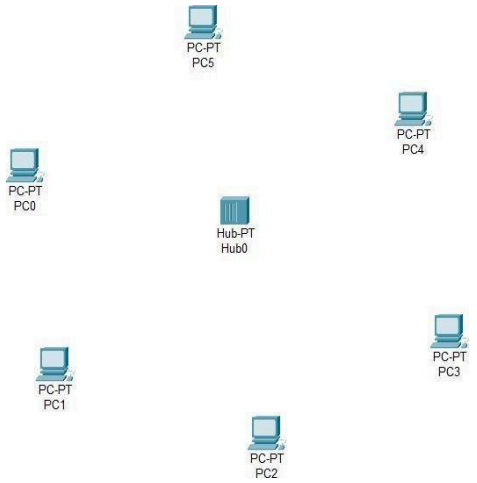
Exp. No. 17	Network Topology using HUB
Date: 20.09.24	

Aim:

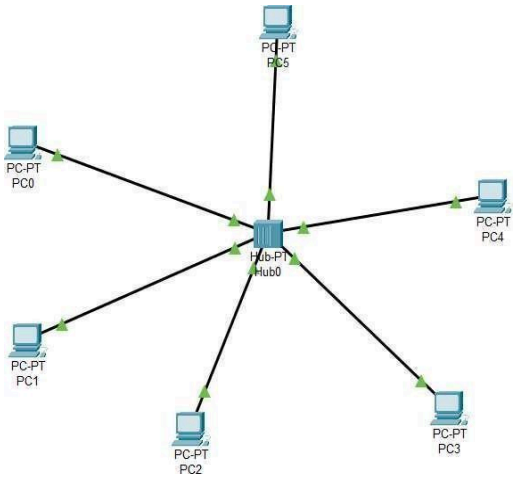
To create a simple network topology using HUB.

Procedure:

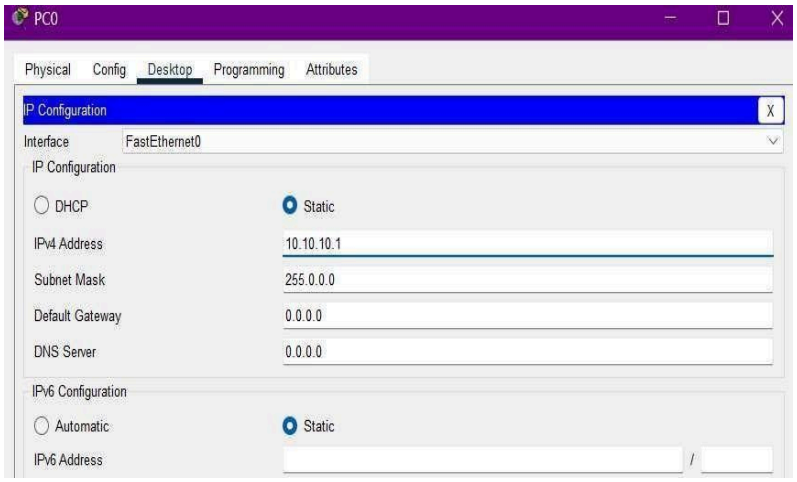
1. From the network component box, click on End Devices and drag and drop one Hub and six pcs onto the workspace.

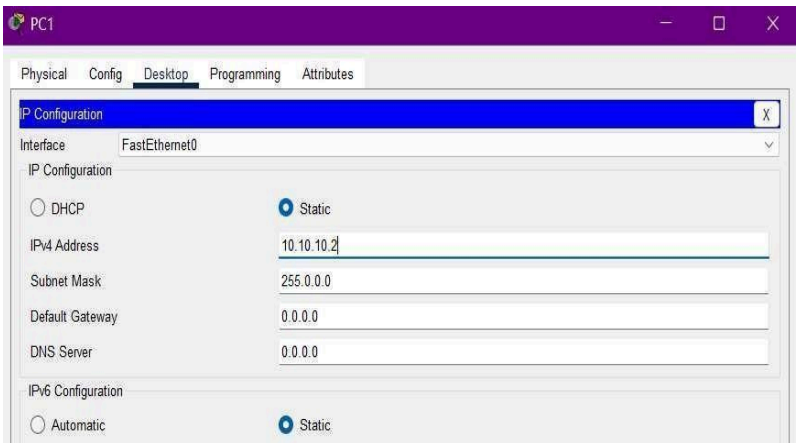


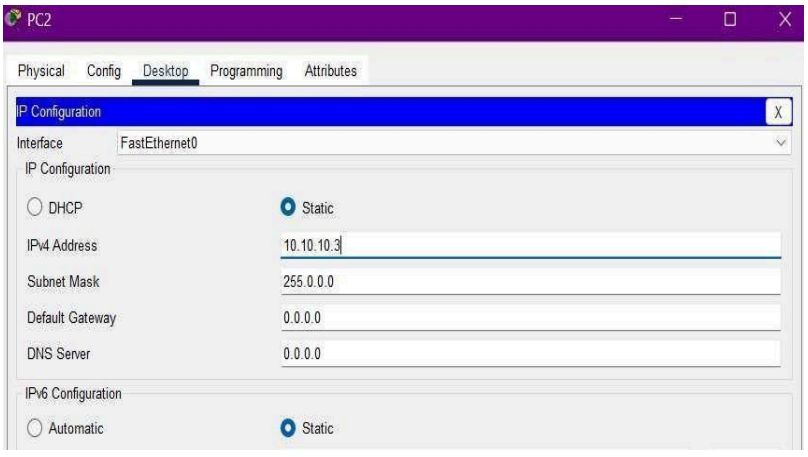
2. Connect each pc's to the hub using copper straight through cable(via fast ethernet port)

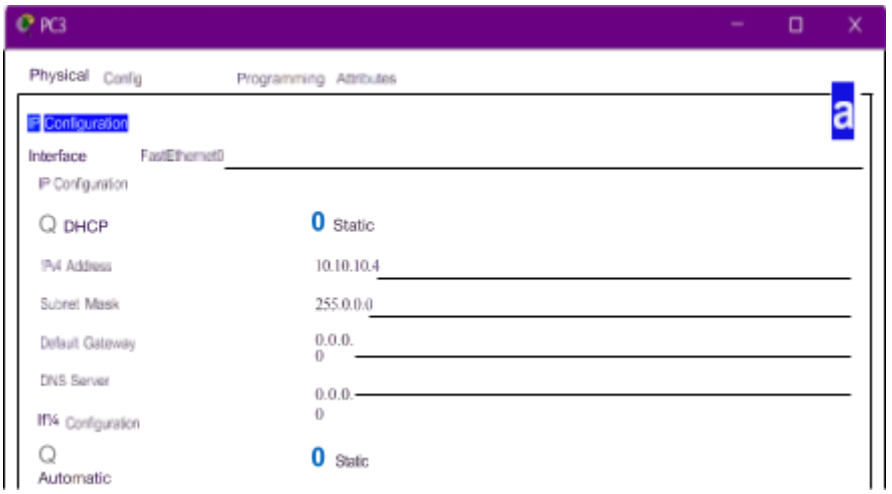


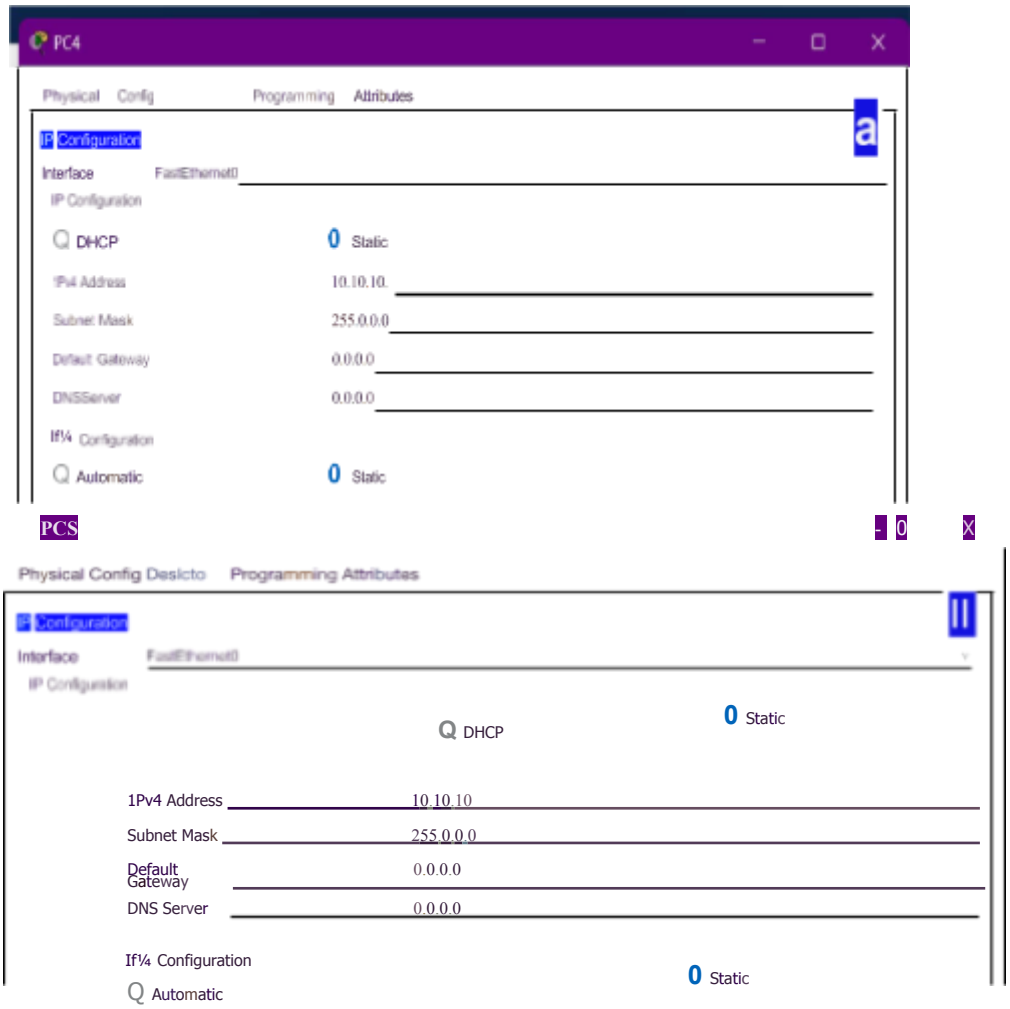
3. Assign IP address to each of the pc by right click the pc-□ Desktop□ipconfig


PC0
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IPv4 Address
10.10.10.1
Subnet Mask
255.0.0.0
Default Gateway
0.0.0.0
DNS Server
0.0.0.0
IPv6 Configuration
Automatic
Static
IPv6 Address

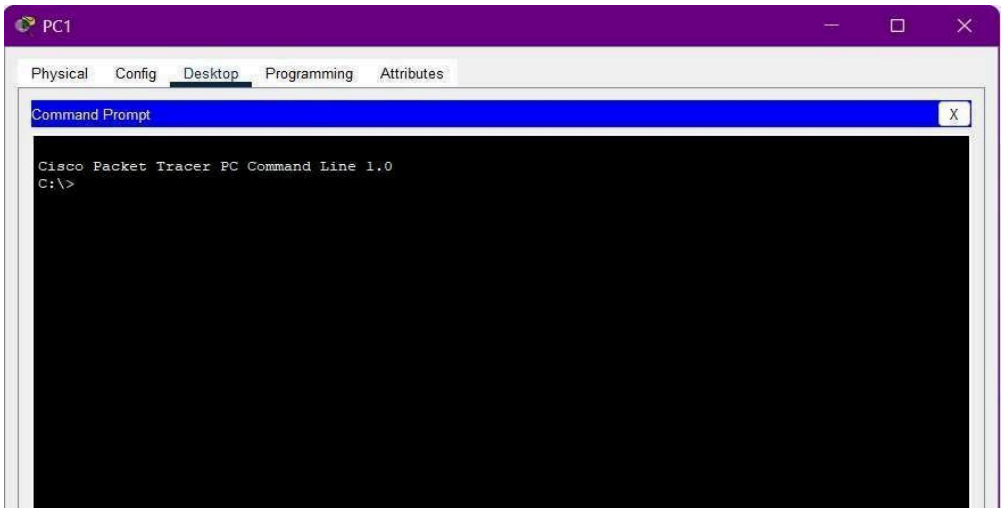

PC1
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IPv4 Address
10.10.10.2
Subnet Mask
255.0.0.0
Default Gateway
0.0.0.0
DNS Server
0.0.0.0
IPv6 Configuration
Automatic
Static


PC2
Physical
Config
Desktop
Programming
Attributes
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IPv4 Address
10.10.10.3
Subnet Mask
255.0.0.0
Default Gateway
0.0.0.0
DNS Server
0.0.0.0
IPv6 Configuration
Automatic
Static

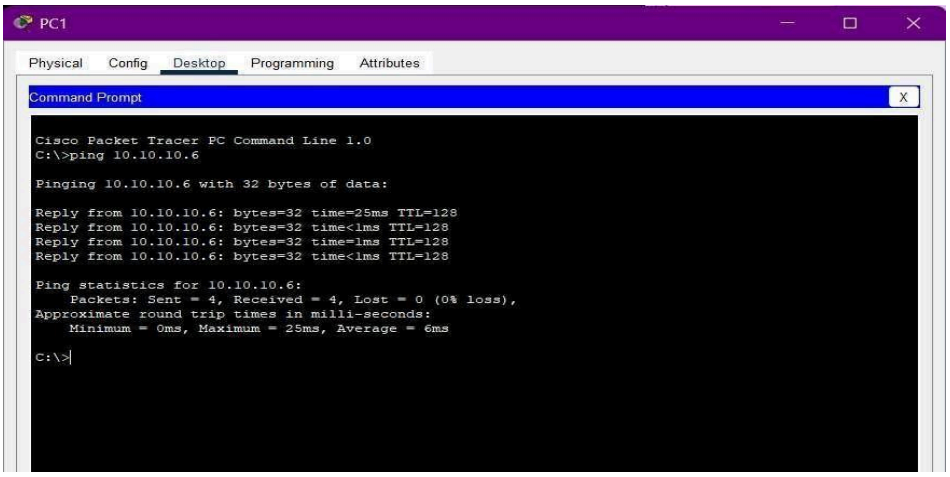

PC3
Physical
Config
Programming
Attributes
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IPv4 Address
10.10.10.4
Subnet Mask
255.0.0.0
Default Gateway
0.0.0.0
DNS Server
0.0.0.0
IPv4 Configuration
Automatic
Static



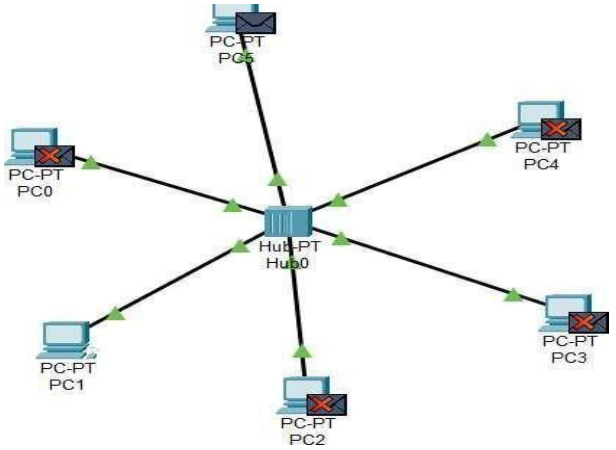
4. To check the connectivity between the pc's, close the ipconfig and open the command prompt from the desktop, for example we have taken the command prompt of pc1.



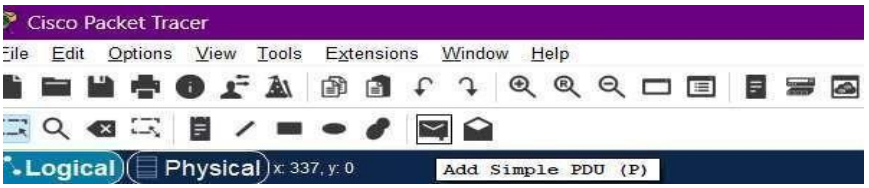
5. Then ping the IP address of the destination pc, here we are going to check with pc5.



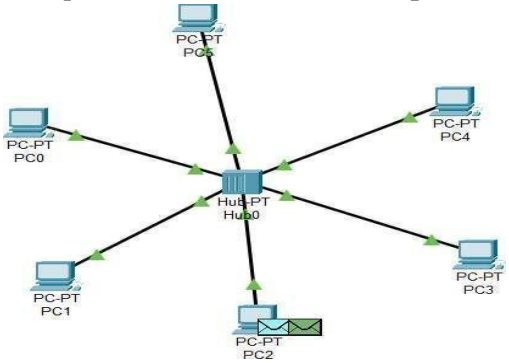
The simulation of ping will be shown like this:



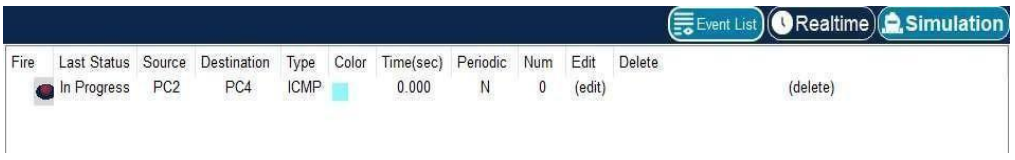
6. Add a simple PDU to one of the PC and check the simulation of the message from the source pc to destination pc via the Hub.



Here we have added a simple PDU where the **source is pc 2** and **destination is pc 4.**

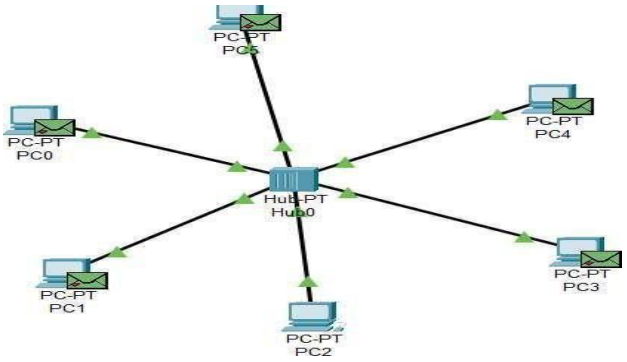


At the bottom of the cisco packet tracer, we can check the status of the PDU.

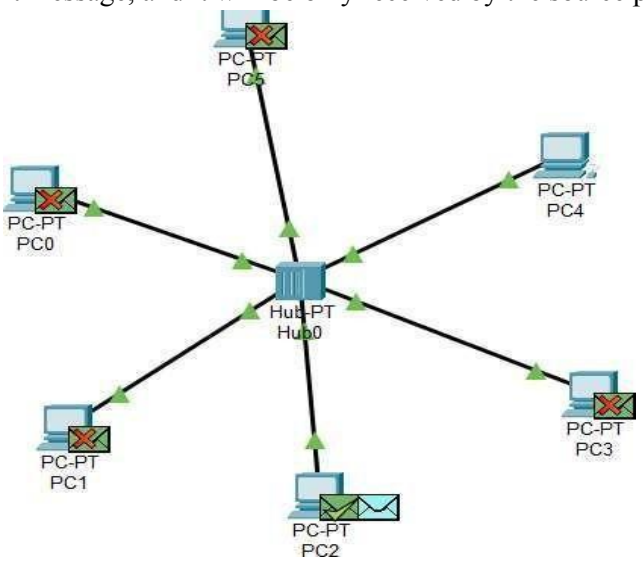


Here the status of the PDU is In Progress.

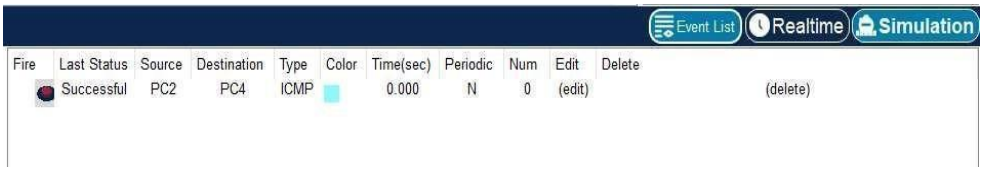
7. The HUB broadcasts the message to all the pc's connected with it, but the only pc which is assigned as destination will receive it.



8. After the message reaches the destination, the destination pc again broadcasts the acknowledgement message, and it will be only received by the source pc.



After the acknowledgement, the status of the PDU will be Successful.



Result:

Thus a simple network topology using a hub is designed, configured, tested and PDU is transmitted successfully using packet tracer.

Exp. No. 18	Creating a Network Topology using Switch
Date: 21.09.24	

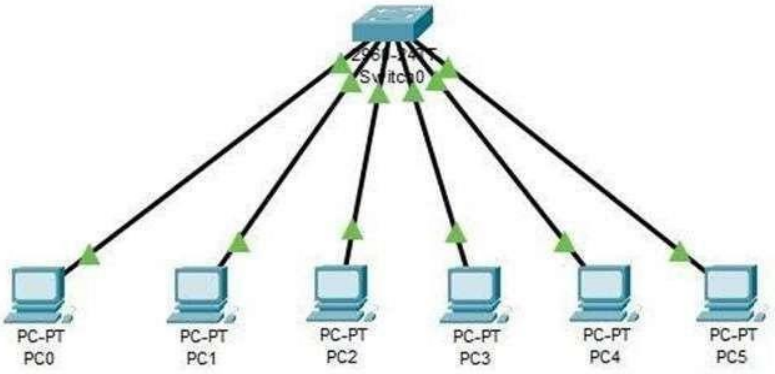
Aim:

To create a network topology using switch.

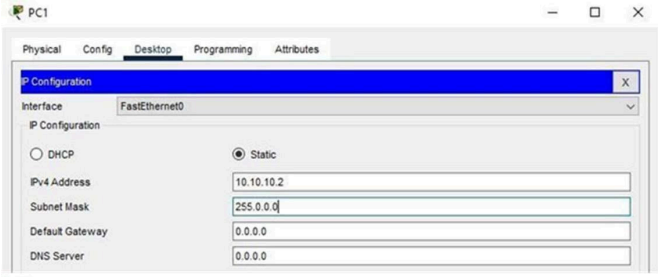
Procedure:

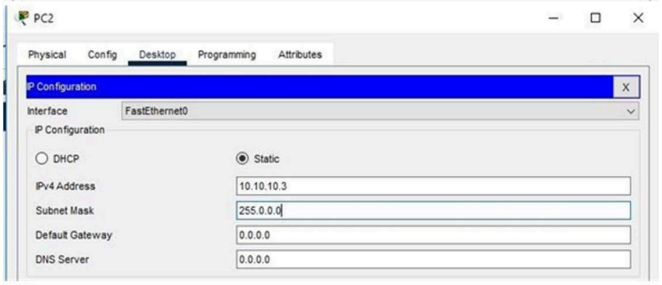
1. From the network component box, click on End Devices and drag and drop one Switch and six PCs on the workspace and connect the PCs with the Switch using copper straight through wire.

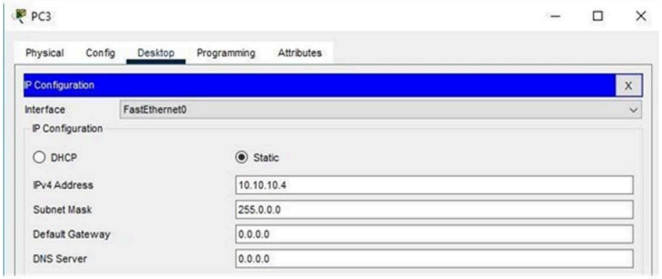
   Switch View:

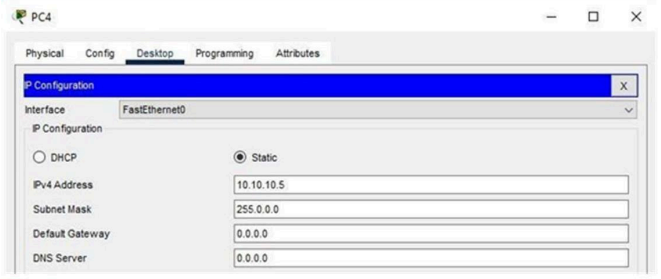


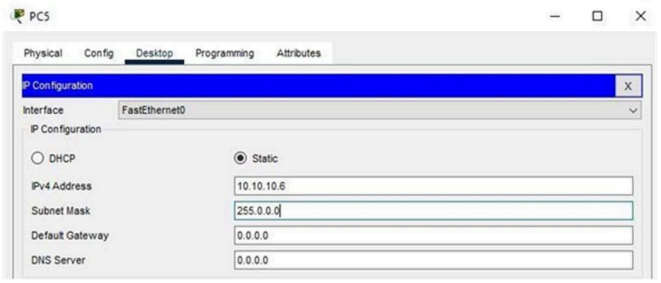
2. Connect the Switch and the PC and assign the IP address to each PC



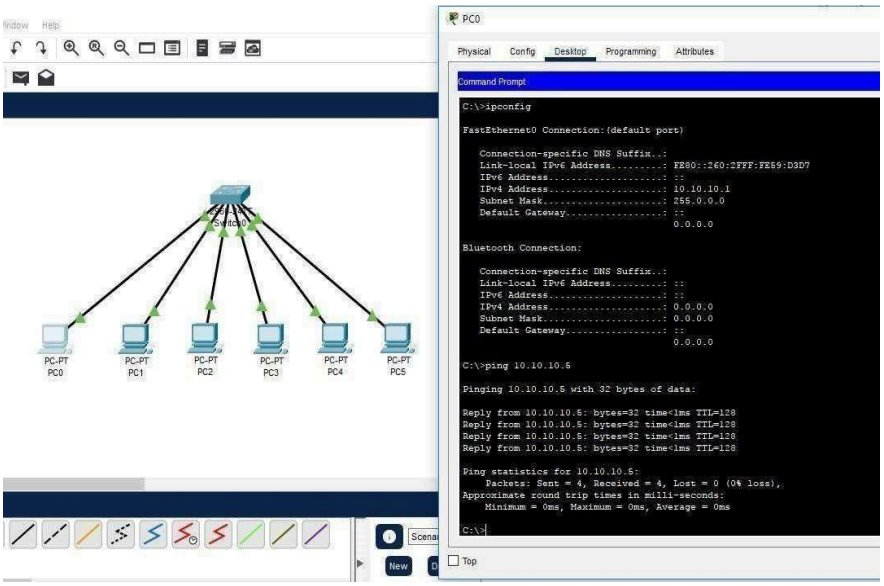


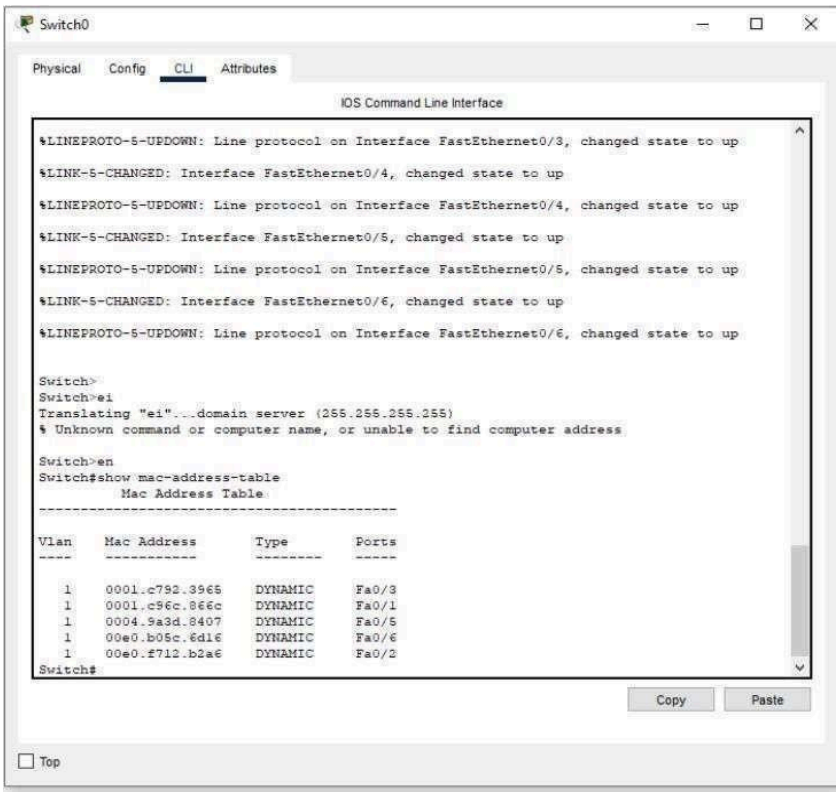




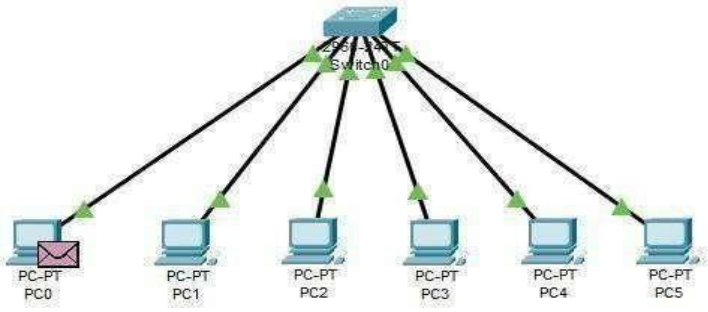


3. Close the IP configuration, open the command prompt and ping the IP address of destination pc.

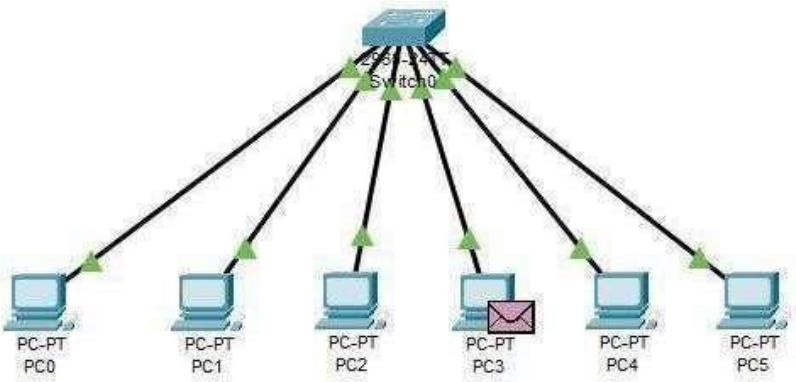




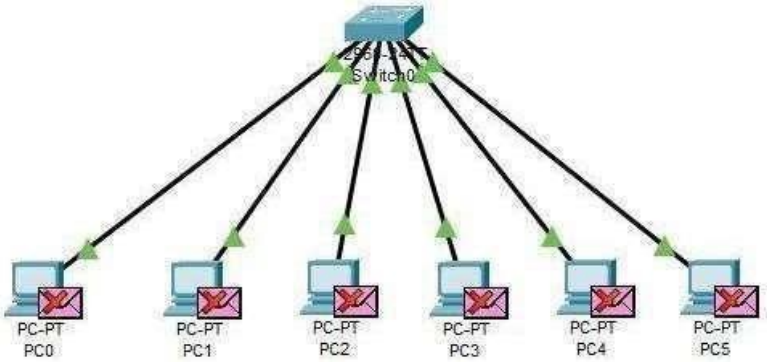
4. Add a simple PDU to one of the PCs and check the stimulation of the message from the source pc to destination pc via to the switch.



5. The message is transmitted from the source PC to the switch and then switch to destination PC



6. The switch by default unicasts the message but at the second time the switch broadcasts the message to all other PCs.



Result:

Thus a simple network topology using switch is designed, configured, tested and PDU is transmitted successfully using packet tracer.

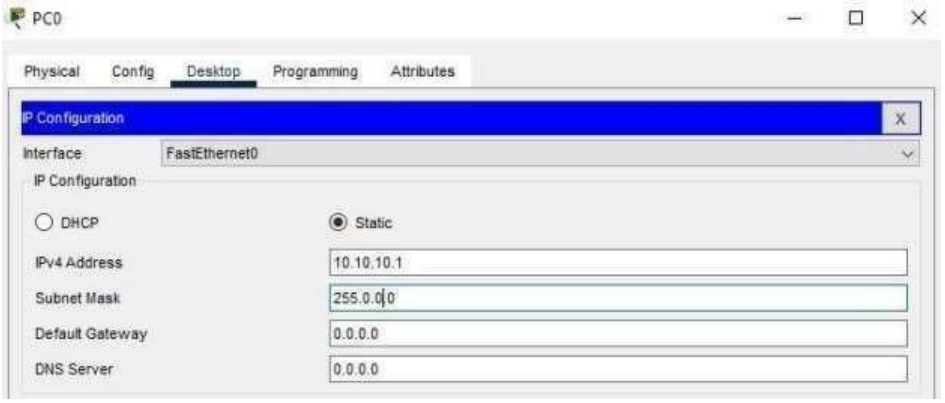
Exp. No. 19	Creating a Network Topology using Repeater
Date: 28.09.24	

Aim:

To create a network topology using repeater.

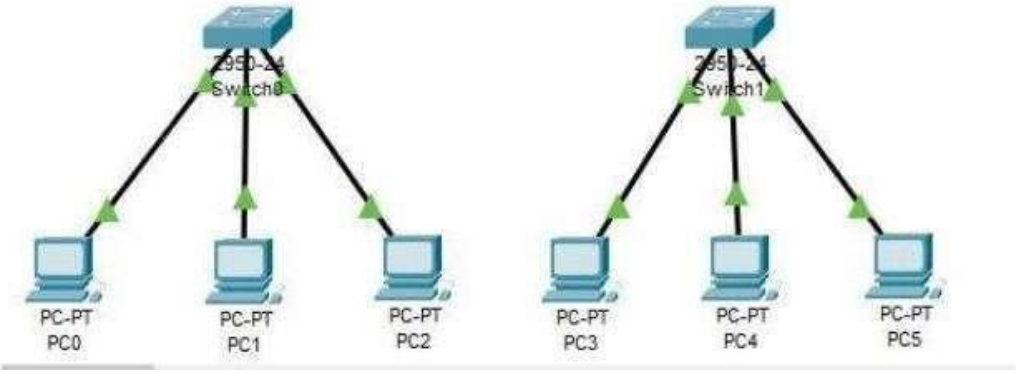
Procedure:

1. From the Network Component Box click on End Devices and drag and drop six Generic pcs into the workspace.
2. And assign IP address to each PC.



Similarly, assign IP address to all other PCs.

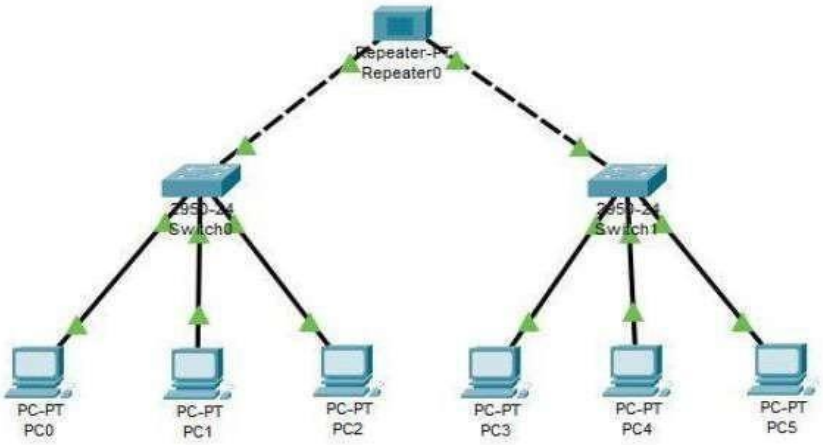
3. Click on the switch from the device-type selection box and insert two switches into the workspace.



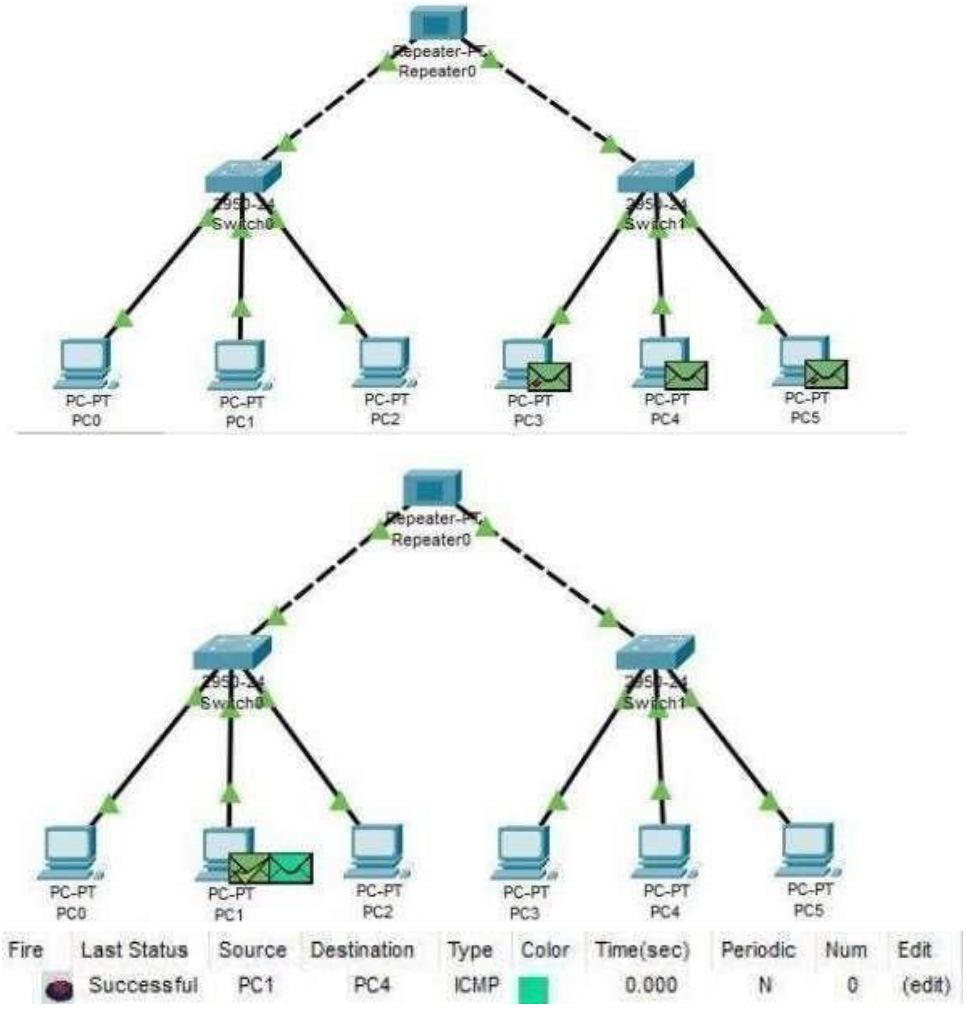
4. Click on the connections, then click on copper straight through and then connect the first three pcs with the first Switch and connect the remaining three pcs with the second switch
5. Physical view of the Repeater.

A repeater is implemented in computer networks to expand the coverage area of the network, repropagate a weak or broken signal and or service remote nodes. Repeaters amplify the received/input signal to a higher frequency domain so that it is reusable, scalable and available.

6. Drag and drop a repeater into the workspace and connect the two switches to the repeater.



7. To check the connection between the pcs, choose a PC as the sender and another PC as the receiver and transmit a simple PDU on the network.



Result:

Thus a simple network topology using repeater is designed, configured, tested and PDU is transmitted successfully using packet tracer.

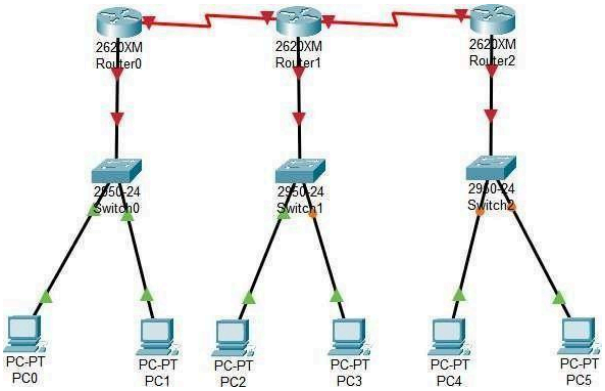
Exp. No. 20	Designing a Network Using Router and Switches
Date: 04.10.24	

Aim:

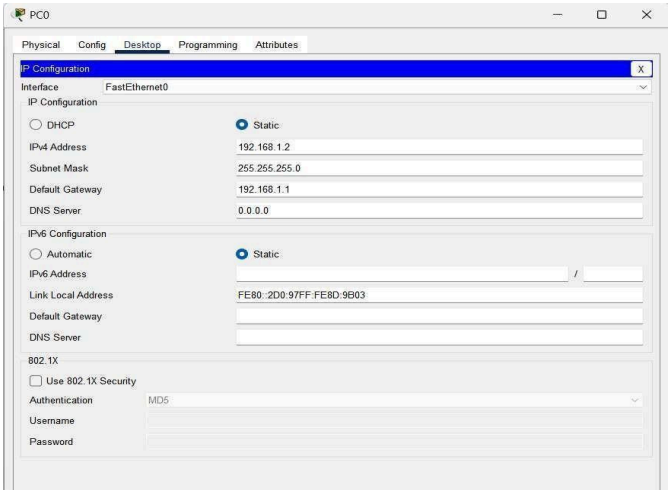
To connect the three different LANs using Router and Switches.

Procedure:

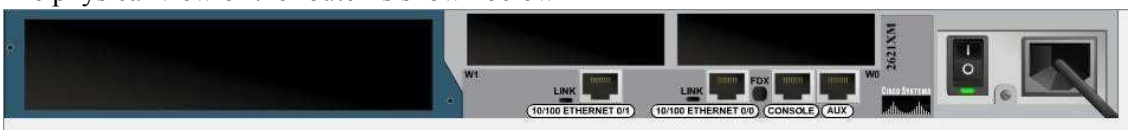
1. Drag and drop six pcs, three switches and routers from the component box into the workspace.



2. Connect the PC and the switch using copper straight through cable and assign the IP address to each PC, the pcs under the Switch 0 contains IP address 192.168.1.2 and 192.168.1.3 and the PCs under switch 1 contains IP address 192.168.2.2 and 192.168.2.3 and the PCs under the switch 2 contains the IP address 192.168.3.2 and 192.168.3.3.



3. The physical view of the router is shown below



4. Then we need to connect each router using serial DTE cable. Before that we need to attachWIC-1T to the router.
5. WIC-1T: The WIC-1T provides a single port serial connection to remote sites or legacy serial network devices such as Synchronous Data Link Control (SDLC) concentrators, alarm systems,and packet over SONET (POS) devices.

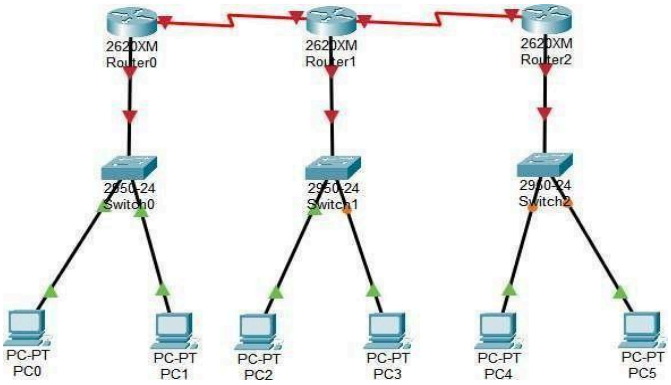
6. We need to attach 1 WIC-1T port on router 0 and router 2 and two WIC- 1T ports on router 1 when the router is off.

Router 0:

Router 1:

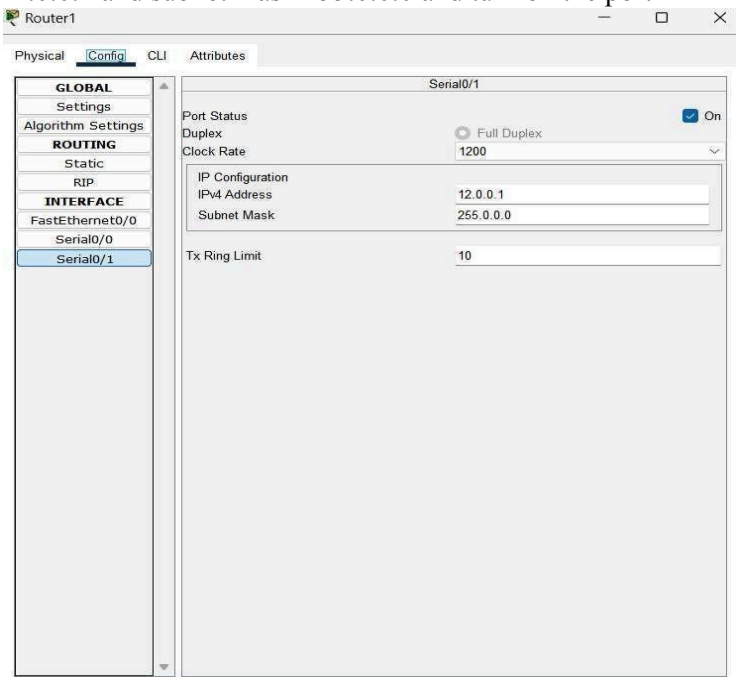
Router 2:

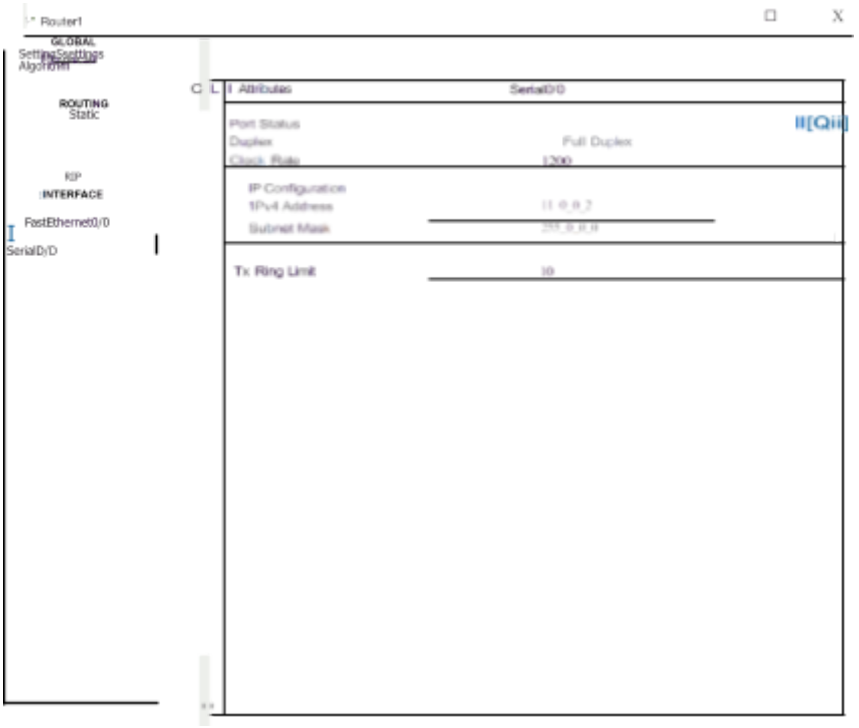
7. Then connected each router using serial DTE cable.

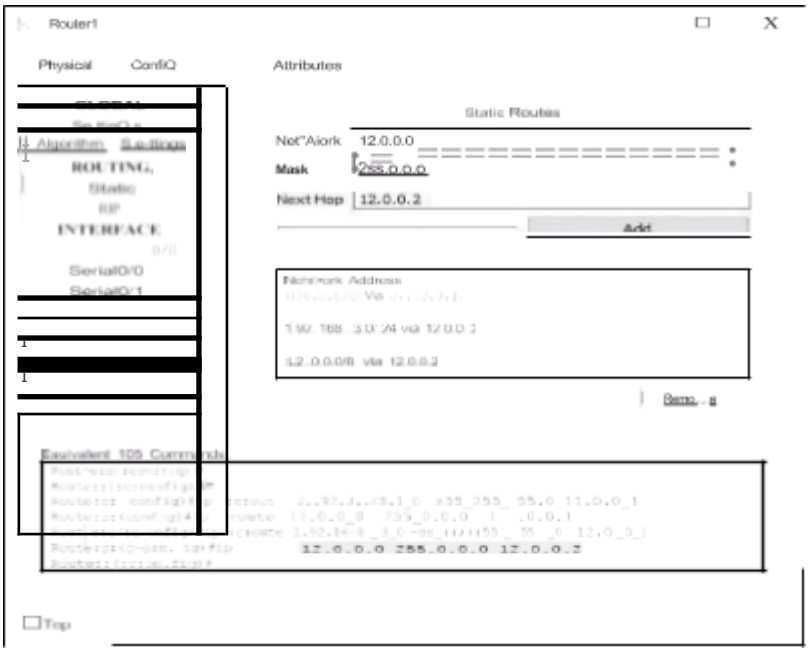


8. To transmit packets, we need to assign IP address to each router and assign routing tables to each router.
9. In Router 0, Go to config -> Interface, under interface go to Fast Ethernet and assign IP address 192.168.1.1 and assign subnet mask 255.255.255.0 and under interface go to serial0/0 and assign IP address

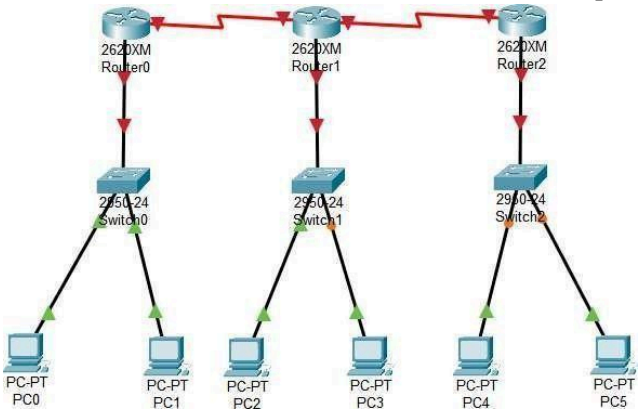
11.0.0.1 and subnet mask 255.0.0.0 and turn on the port





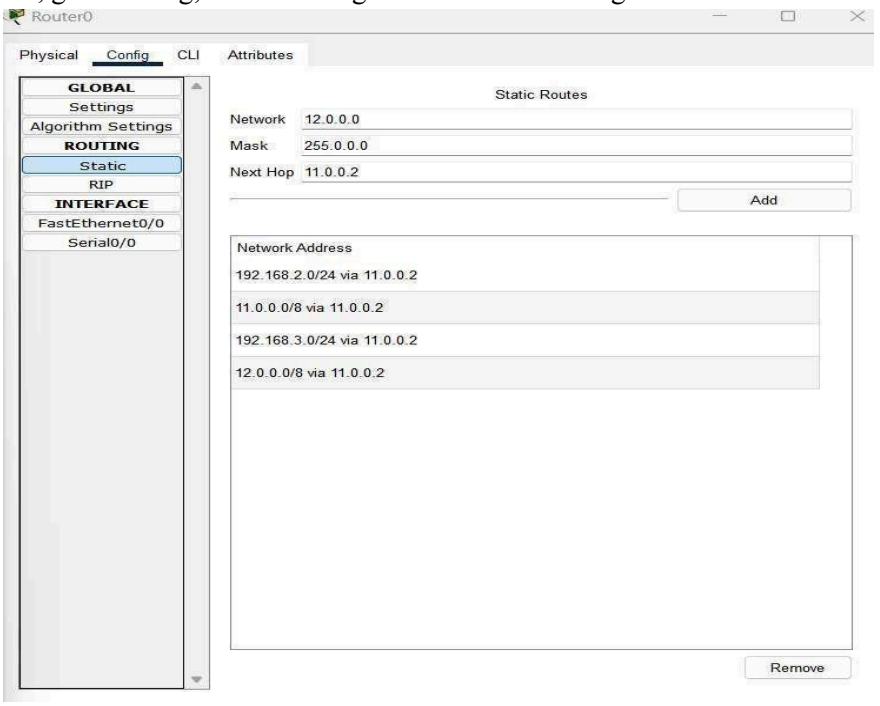


10. Similarly assign IP address to router 1 and router 2 and turn on the port status

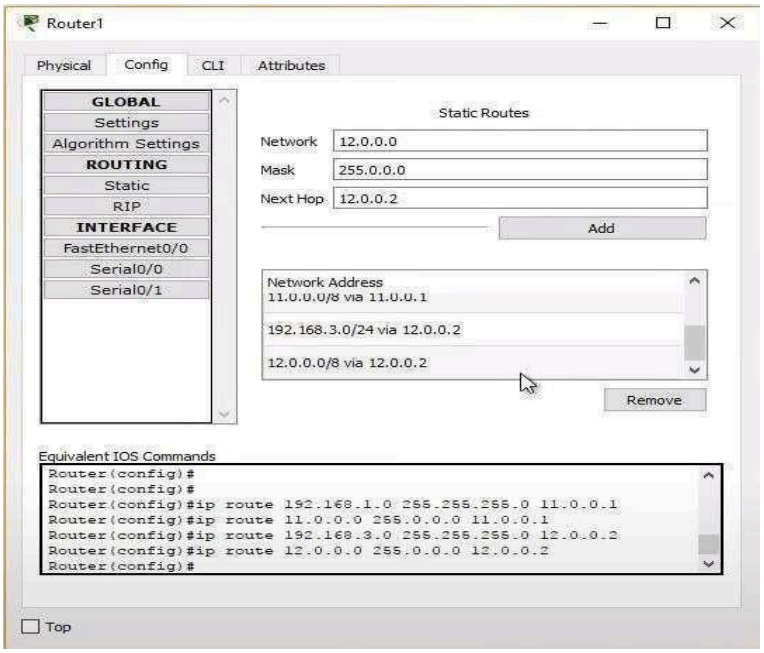


11. Then we need to configure the Routing Table in order to transmit packets from one device to another.

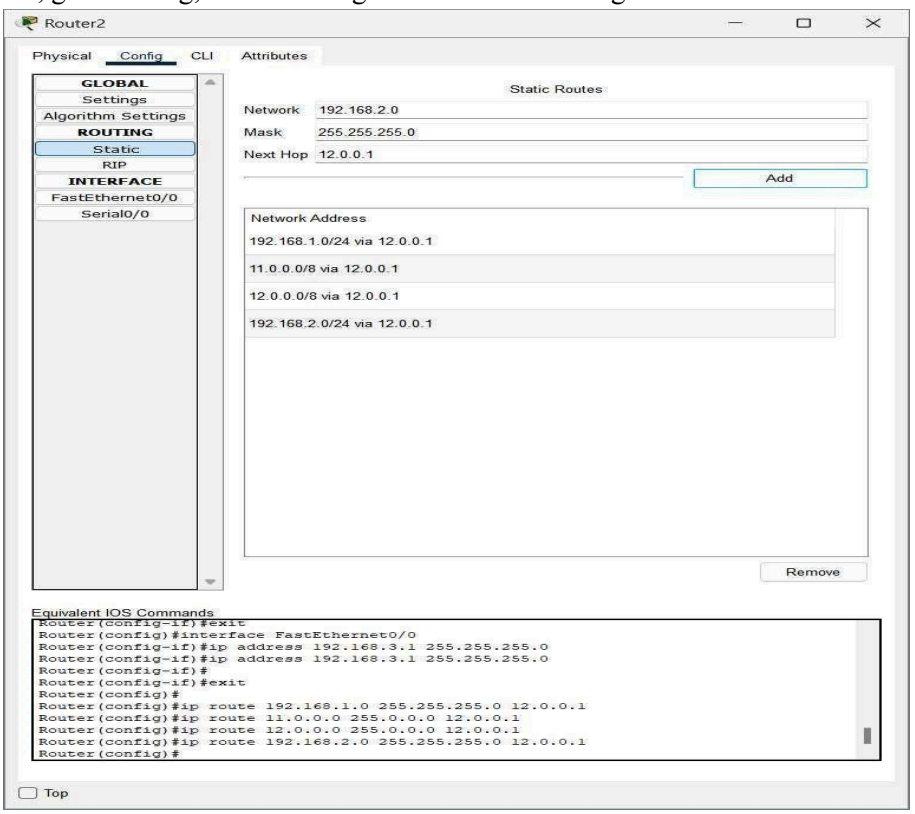
12. In Router 0, go to config, under routing select static and assign static routes.



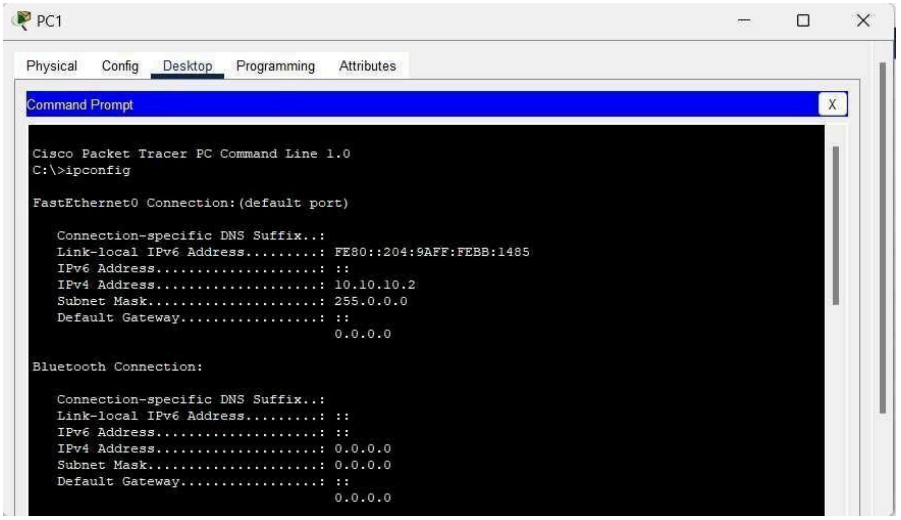
13. In Router 1, go to config, under routing select static and assign static routes.



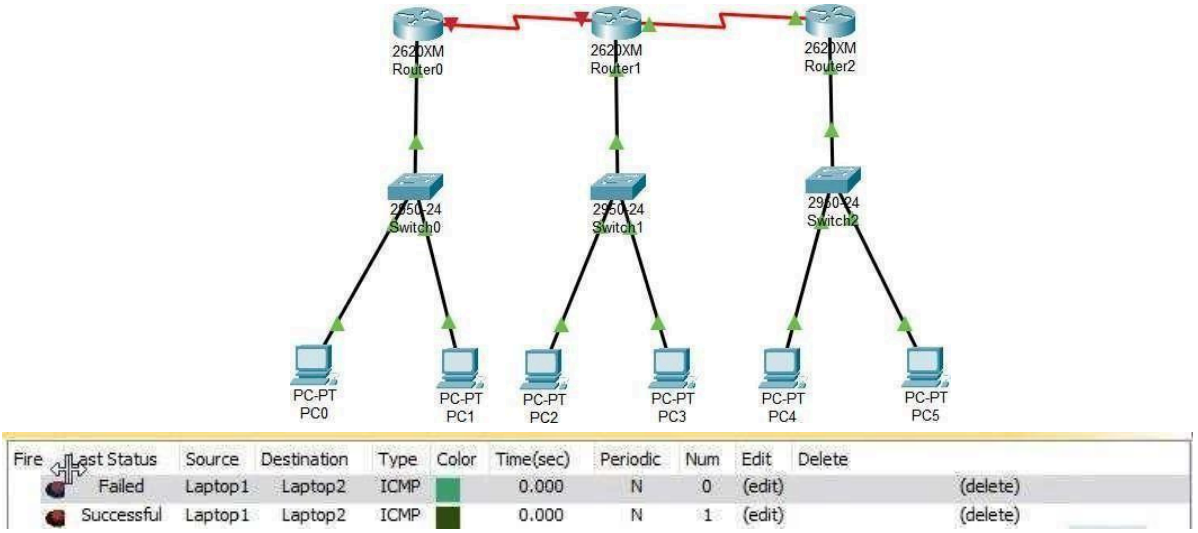
14. In Router 2, go to config, under routing select static and assign static routes.



15. After the routing table is configured for all other Routers, Check the connection by pinging packets from one PC to another. Ping from PC2 (192.168.2.2) to PC5 (192.168.3.3)



16. Add a simple PDU to PC2 and then check the Packet transfer from PC2 to PC5. The Packet isTransferred successfully and the acknowledgment is accepted by the source PC.



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
	Failed	Laptop1	Laptop2	ICMP		0.000	N	0	(edit)	(delete)
	Successful	Laptop1	Laptop2	ICMP		0.000	N	1	(edit)	(delete)

Result:

Thus a simple network topology using 3 routers and 3 switches is designed, configured, tested and PDU is transmitted successfully using packet tracer.

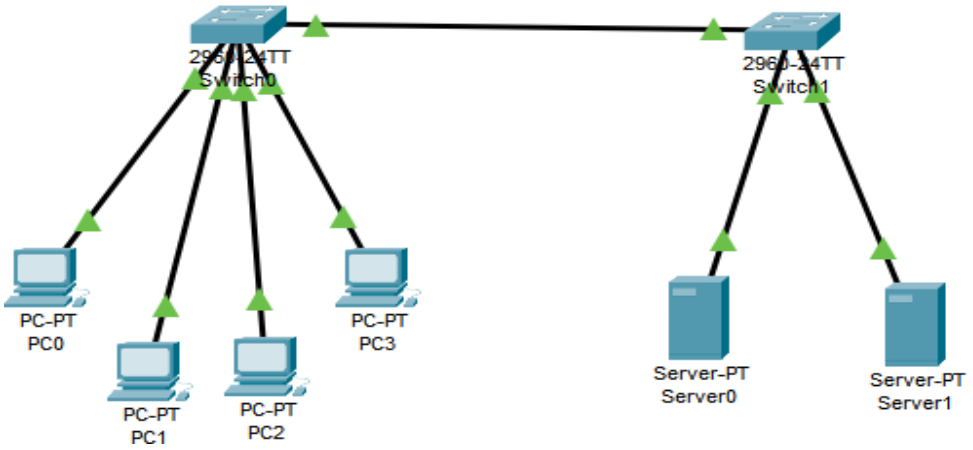
Exp. No. 21	**Designing Simple Virtual LAN Configuration Using CISCO Packet Tracer**
Date: 05.10.24	

Aim:

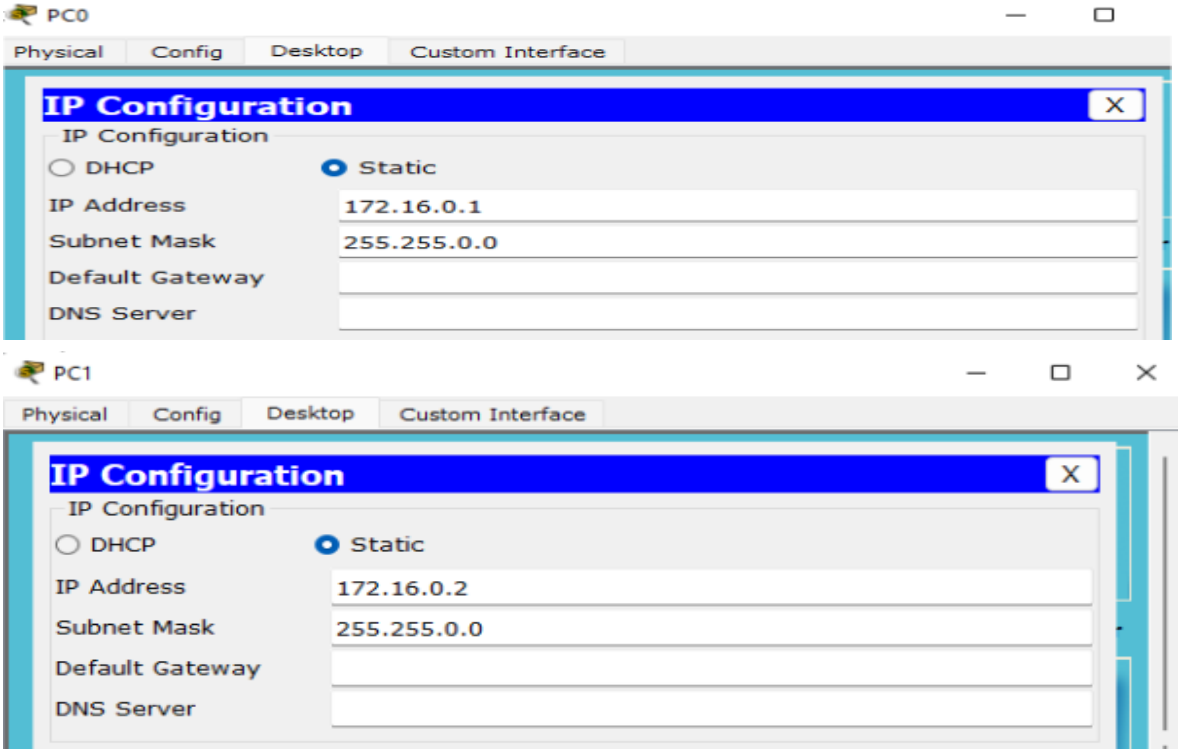
To connect a Simple Virtual LAN Configuration Using CISCO Packet Tracer.

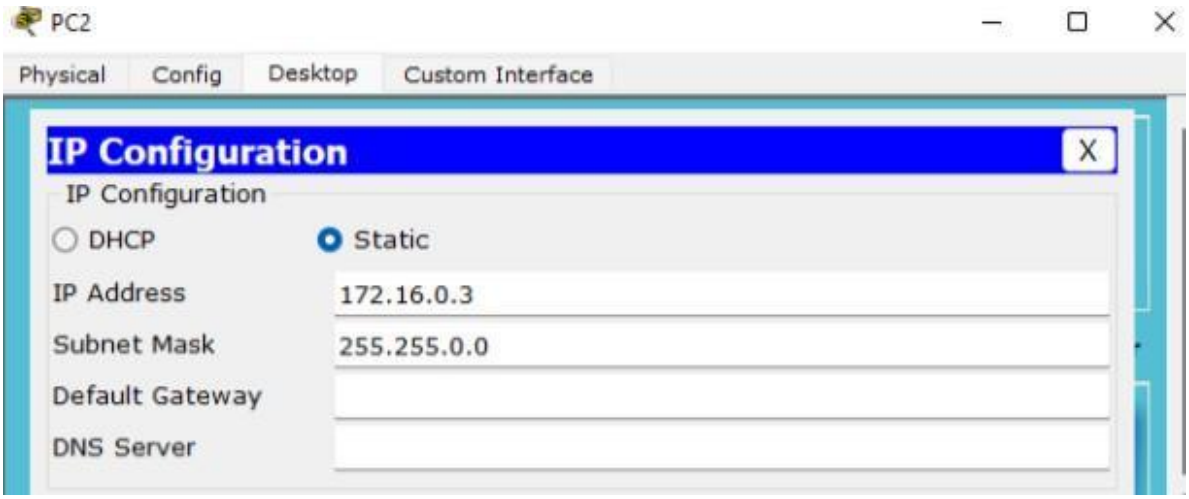
Procedure:

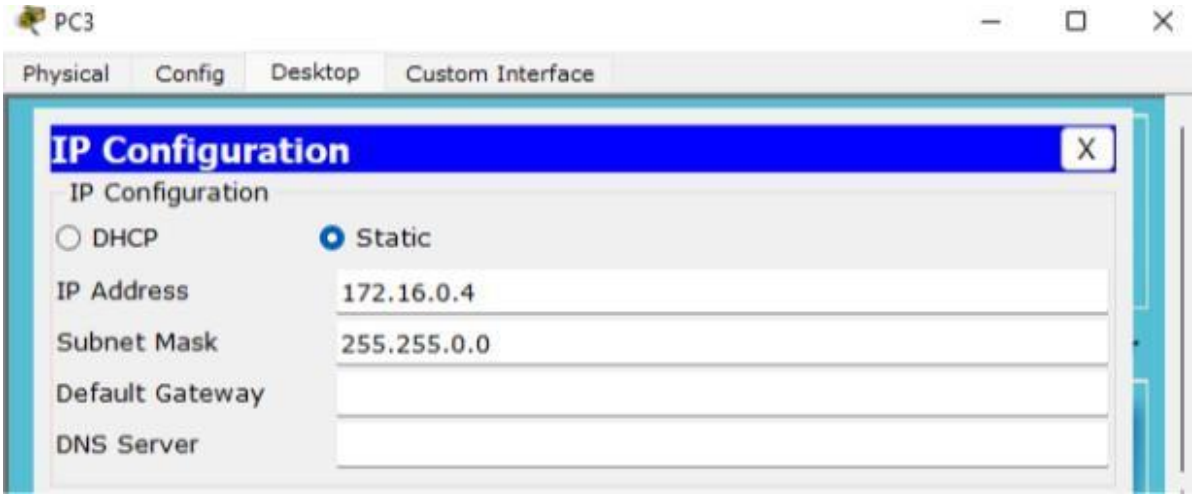
1. Design a simple LAN network with two switches, four PCs and two servers. Connect the four PCs under one switch using copper straight through wire and two servers under another switch using the same wire. Connect the two switches also.

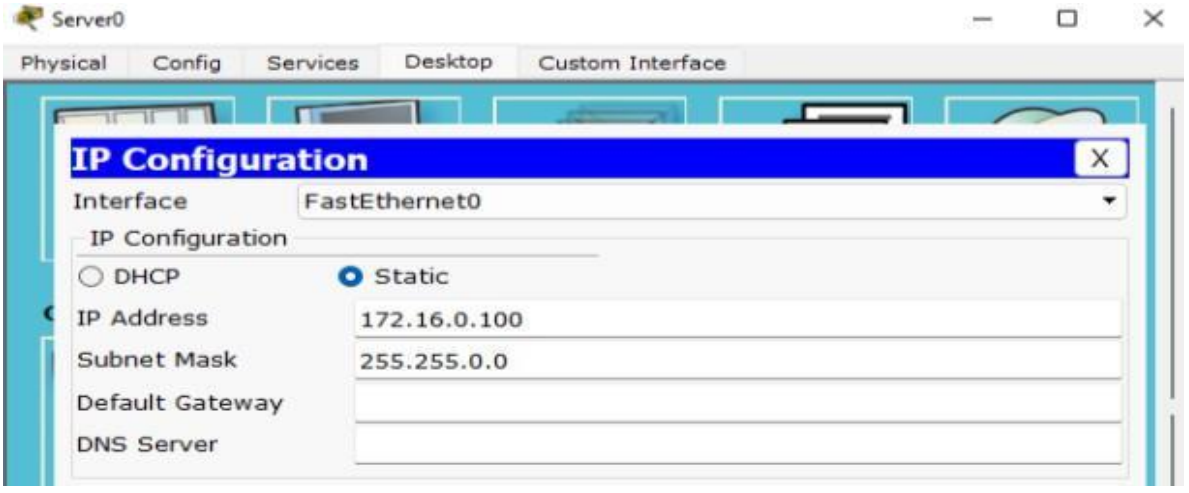


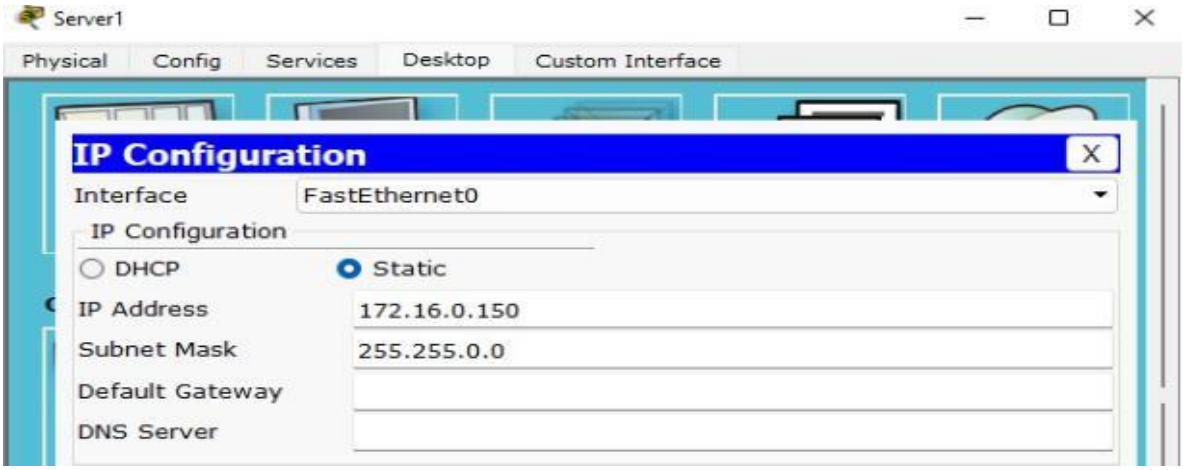
2. Assign the IP address for the PCs and the servers.

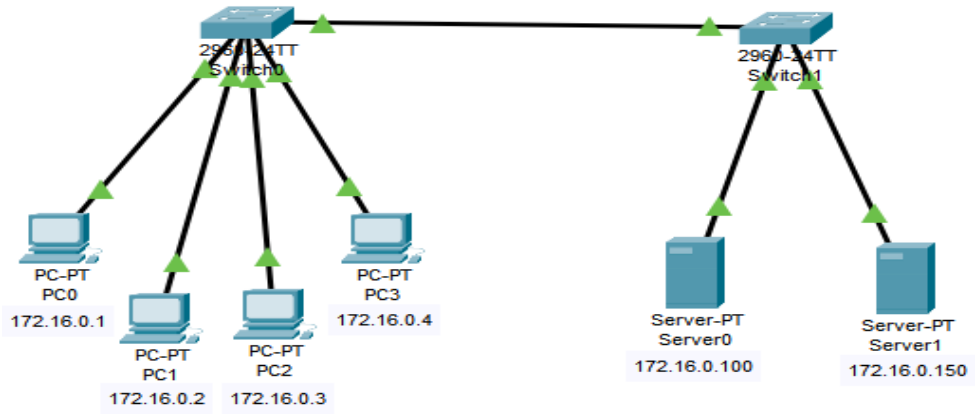



PC2
Physical
Config
Desktop
Custom Interface
IP Configuration
IP Configuration
DHCP
Static
IP Address
172.16.0.3
Subnet Mask
255.255.0.0
Default Gateway
DNS Server

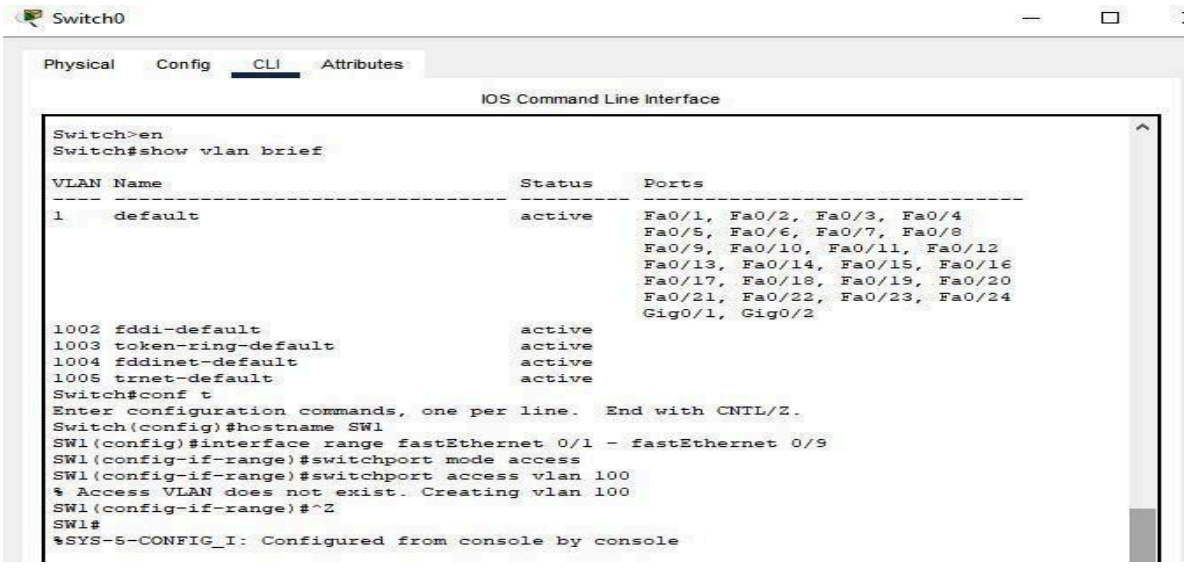

PC3
Physical
Config
Desktop
Custom Interface
IP Configuration
IP Configuration
DHCP
Static
IP Address
172.16.0.4
Subnet Mask
255.255.0.0
Default Gateway
DNS Server


Server0
Physical
Config
Services
Desktop
Custom Interface
IP Configuration
Interface
FastEthernet0
IP Configuration
DHCP
Static
IP Address
172.16.0.100
Subnet Mask
255.255.0.0
Default Gateway
DNS Server





3. In the switch0 command line interface, name the host SW1 and interface the range of fastEthernet 0/1 – fastEthernet 0/9 under vlan 100 and interface the range of fastEthernet 0/10 – fastEthernet 0/19 under vlan 200. During this interfacing, the switchport is enabled to access mode.



```
SW1#show vlan brief

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Fa0/10, Fa0/11, Fa0/12, Fa0/13
                                                 Fa0/14, Fa0/15, Fa0/16, Fa0/17
                                                 Fa0/18, Fa0/19, Fa0/20, Fa0/21
                                                 Fa0/22, Fa0/23, Fa0/24, Gig0/1
                                                 Gig0/2
100  VLAN0100                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                 Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                 Fa0/9
1002 fddi-default                      active
1003 token-ring-default                active
1004 fddinet-default                   active
1005 trnet-default                     active
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#interface range fastEthernet 0/10 - fastEthernet 0/19
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 200
% Access VLAN does not exist. Creating vlan 200
SW1(config-if-range)#^Z
SW1#
%SYS-5-CONFIG_I: Configured from console by console
```

```
SW1#show vlan brief

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Fa0/10, Fa0/11, Fa0/12, Fa0/13
                                                 Fa0/14, Fa0/15, Fa0/16, Fa0/17
                                                 Fa0/18, Fa0/19, Fa0/20, Fa0/21
                                                 Fa0/22, Fa0/23, Fa0/24, Gig0/1
                                                 Gig0/2
100  VLAN0100                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                 Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                 Fa0/9
1002 fddi-default                      active
1003 token-ring-default                active
1004 fddinet-default                   active
1005 trnet-default                     active
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#interface range fastEthernet 0/10 - fastEthernet 0/19
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 200
% Access VLAN does not exist. Creating vlan 200
SW1(config-if-range)#^Z
SW1#
%SYS-5-CONFIG_I: Configured from console by console
```

4. In Switch Command line interface, name the host SW2 and interface the range of fast Ethernet 0/1 – fastEthernet 0/4 under vlan 100 and fastEthernet 0/5 – fastEthernet 0/9 under vlan 200. During this interfacing, the switchport is enabled to access mode.

```
SW1#show vlan brief

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Fa0/10, Fa0/11, Fa0/12, Fa0/13
                                                 Fa0/14, Fa0/15, Fa0/16, Fa0/17
                                                 Fa0/18, Fa0/19, Fa0/20, Fa0/21
                                                 Fa0/22, Fa0/23, Fa0/24, Gig0/1
                                                 Gig0/2
100  VLAN0100                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                 Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                 Fa0/9
1002 fddi-default                      active
1003 token-ring-default                active
1004 fddinet-default                   active
1005 trnet-default                     active
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#interface range fastEthernet 0/10 - fastEthernet 0/19
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 200
% Access VLAN does not exist. Creating vlan 200
SW1(config-if-range)#^Z
SW1#
%SYS-5-CONFIG_I: Configured from console by console
```

```
SW1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/10, Fa0/11, Fa0/12, Fa0/13
                                                Fa0/14, Fa0/15, Fa0/16, Fa0/17
                                                Fa0/18, Fa0/19, Fa0/20, Fa0/21
                                                Fa0/22, Fa0/23, Fa0/24, Gig0/1
                                                Gig0/2
100  VLAN0100                         active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9
1002 fddi-default                     active
1003 token-ring-default               active
1004 fddinet-default                  active
1005 trnet-default                    active
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#interface range fastEthernet 0/10 - fastEthernet 0/19
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 200
% Access VLAN does not exist. Creating vlan 200
SW1(config-if-range)#^Z
SW1#
%SYS-5-CONFIG_I: Configured from console by console
```

```
SW1#show vlan brief

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/10, Fa0/11, Fa0/12, Fa0/13
                                                Fa0/14, Fa0/15, Fa0/16, Fa0/17
                                                Fa0/18, Fa0/19, Fa0/20, Fa0/21
                                                Fa0/22, Fa0/23, Fa0/24, Gig0/1
                                                Gig0/2
100  VLAN0100                         active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9
1002 fddi-default                     active
1003 token-ring-default               active
1004 fddinet-default                  active
1005 trnet-default                    active
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#interface range fastEthernet 0/10 - fastEthernet 0/19
SW1(config-if-range)#switchport mode access
SW1(config-if-range)#switchport access vlan 200
% Access VLAN does not exist. Creating vlan 200
SW1(config-if-range)#^Z
SW1#
%SYS-5-CONFIG_I: Configured from console by console
```

5. Interface the gigabit Ethernet 0/1 in both the switches to enable the connection between the two switch networks by enabling the switchport in the trunk mode.

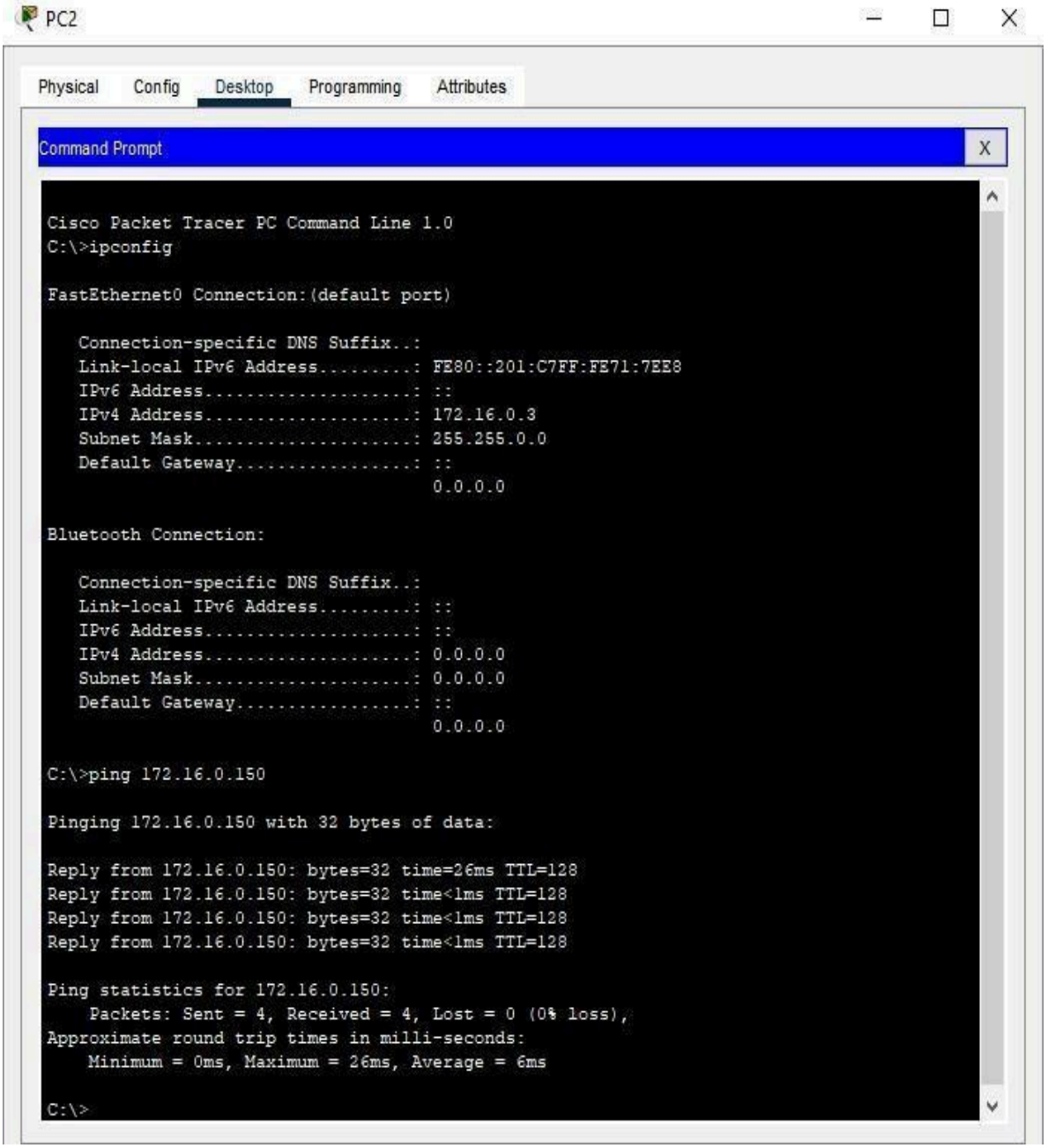
```
SW1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW1(config)#interface gigabitEthernet 0/1
SW1(config-if)#switchport mode trunk

SW1(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to
down

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to
up
```

```
SW1#show interfaces trunk
Port        Mode         Encapsulation  Status        Native vlan
Gig0/1      on           802.1q         trunking      1

Port        Vlans allowed on trunk
Gig0/1      1-1005

Port        Vlans allowed and active in management domain
Gig0/1      1,100,200

Port        Vlans in spanning tree forwarding state and not pruned
Gig0/1      1,100,200

SW1#
```

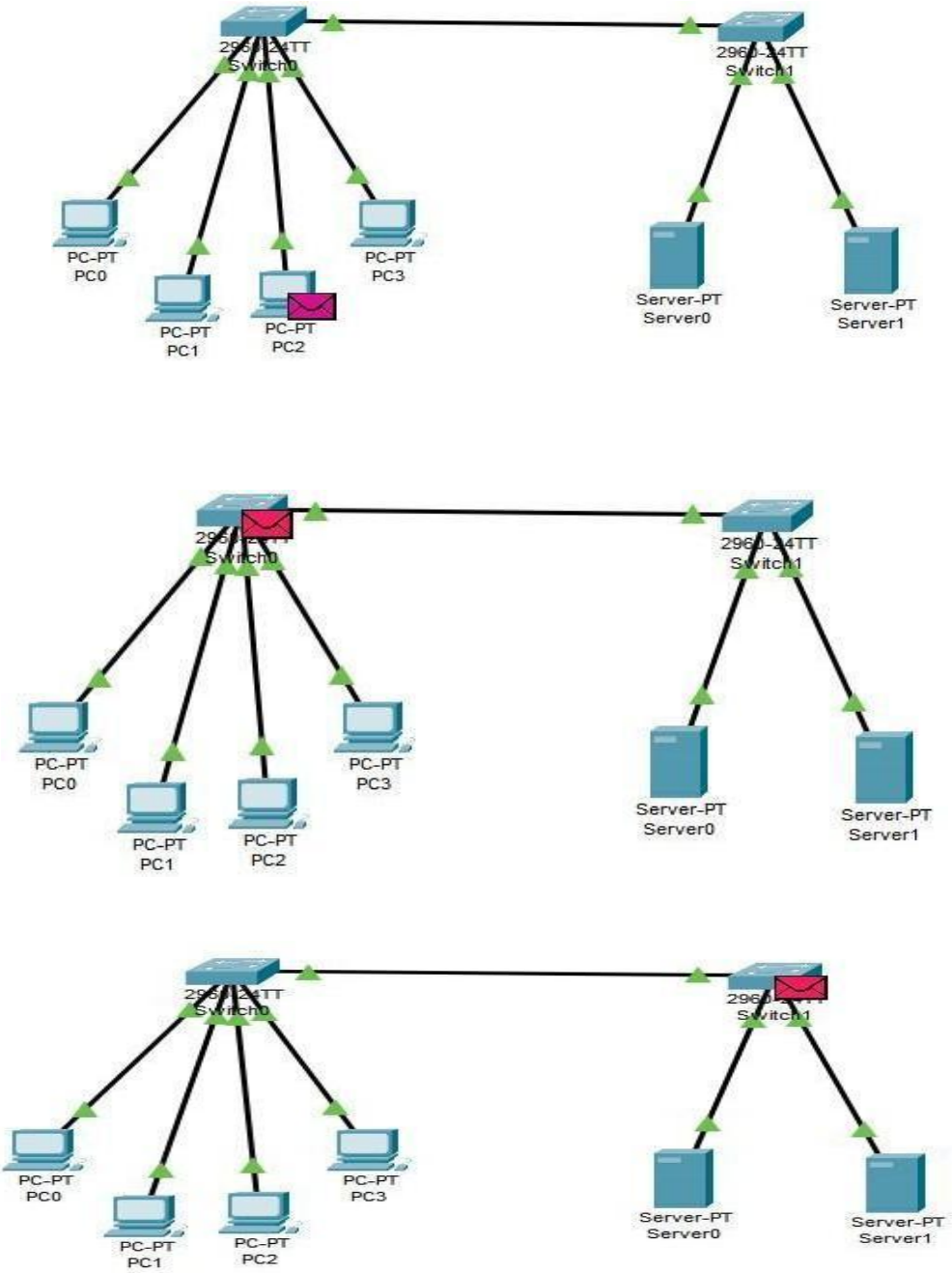
```
SW2#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
SW2(config)#interface gigabitEthernet 0/1
```

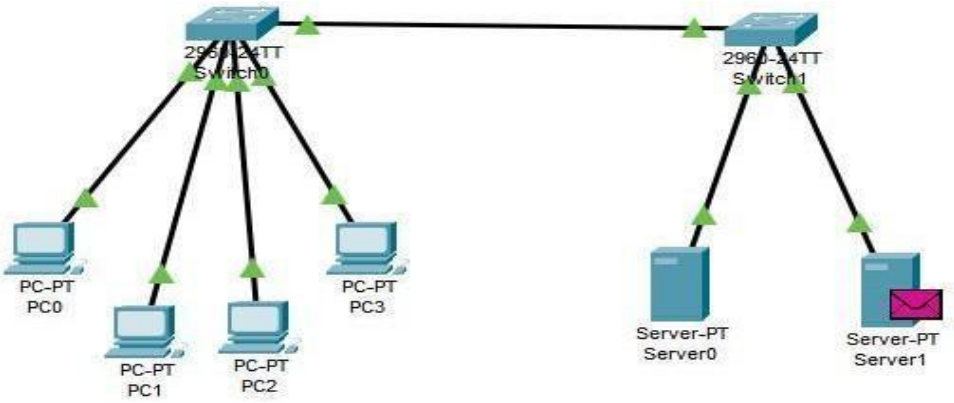
```
SW2(config-if)#switchport mode trunk

SW2(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to down

%LINEPROTO-5-UPDOWN: Line protocol on Interface GigabitEthernet0/1, changed state to up

SW2(config-if)#^Z
SW2#
%SYS-5-CONFIG_I: Configured from console by console

SW2#show interfaces trunk
Port        Mode         Encapsulation  Status        Native vlan
Gig0/1      on           802.1q         trunking      1

Port        Vlans allowed on trunk
Gig0/1      1-1005

Port        Vlans allowed and active in management domain
Gig0/1      1,100,200

Port        Vlans in spanning tree forwarding state and not pruned
Gig0/1      1,100,200

SW2#
```

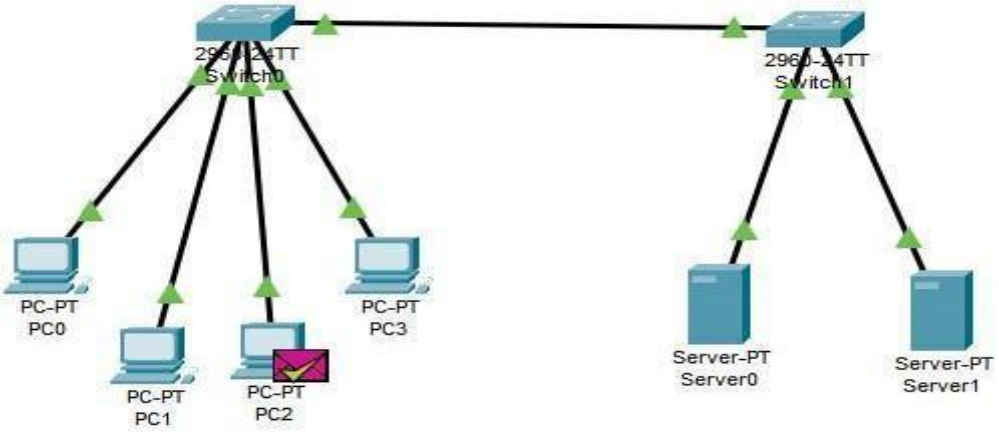
6. Ping the PC of the first network to the Server of another network which is connected through the switches.



7. Place the packets in the Source and the Destination. In simulation mode, check the transmission of the packets is successful.







Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit
	Successful	PC2	Server1	ICMP		0.000	N	0	(edit)

Result:

Thus a Simple Virtual LAN Configuration Using CISCO Packet Tracer has been executed Successfully.

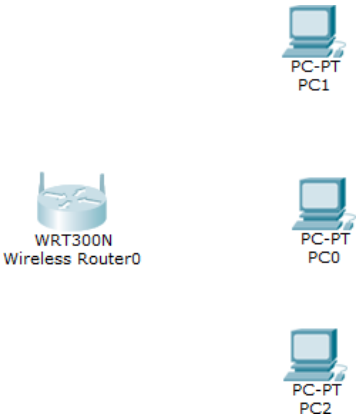
Exp. No. 22	Study of Stimulation of Wireless Connection Between Router and PC
Date: 18.10.24	

Aim:

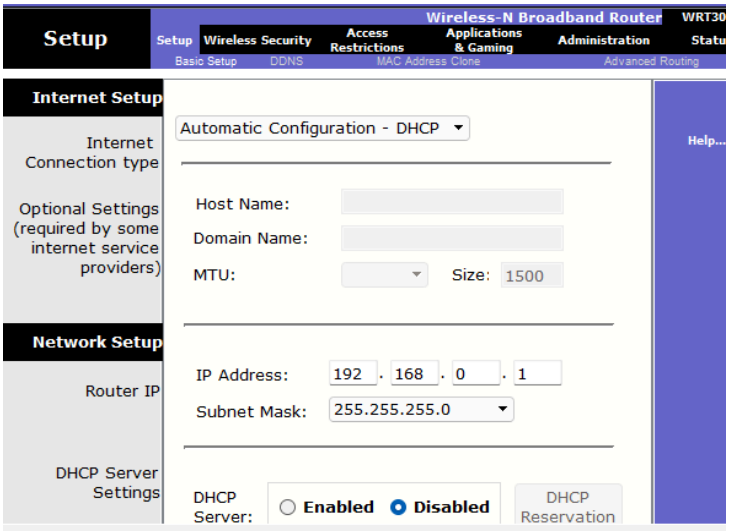
Configuration of Wireless LAN Using CISCO Packet Tracer.

Algorithm:

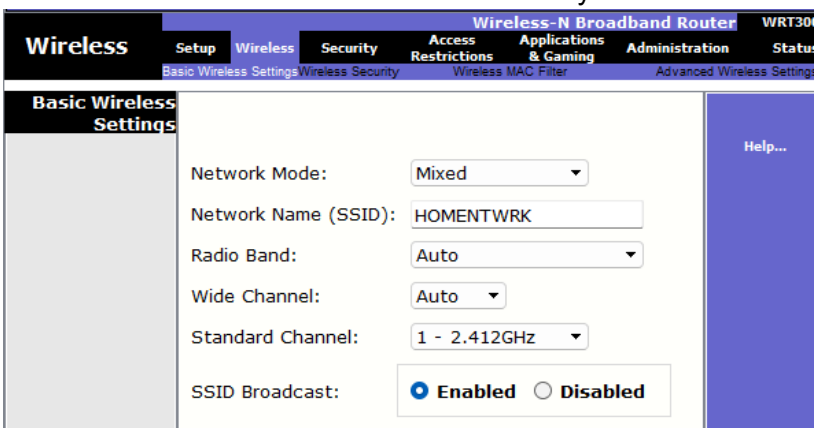
1. Arrange the Pcs and WRT300N Wireless Router.



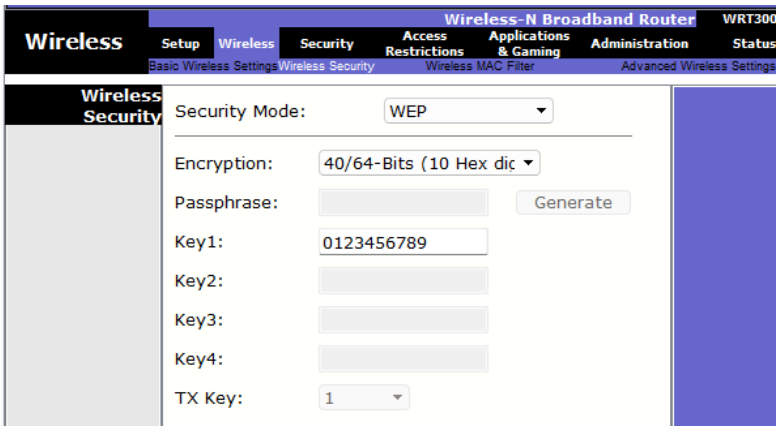
2. Click on the Wireless Router. Set the DHCP server to Disabled and save changes.



3. Go to the Wireless tab and set the Network Name of your choice. Save settings.

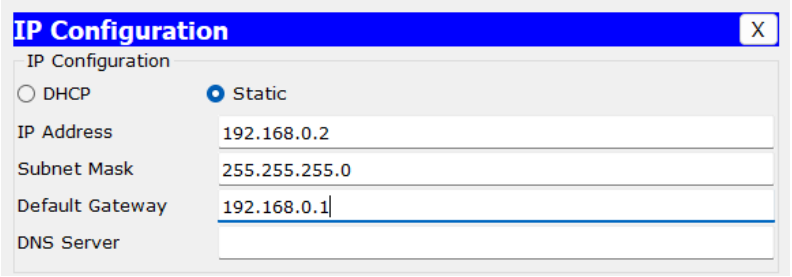


4. Click on Wireless Security and select WEP Security Mode. Set Key 1 as 10 digit HEX. Save settings.

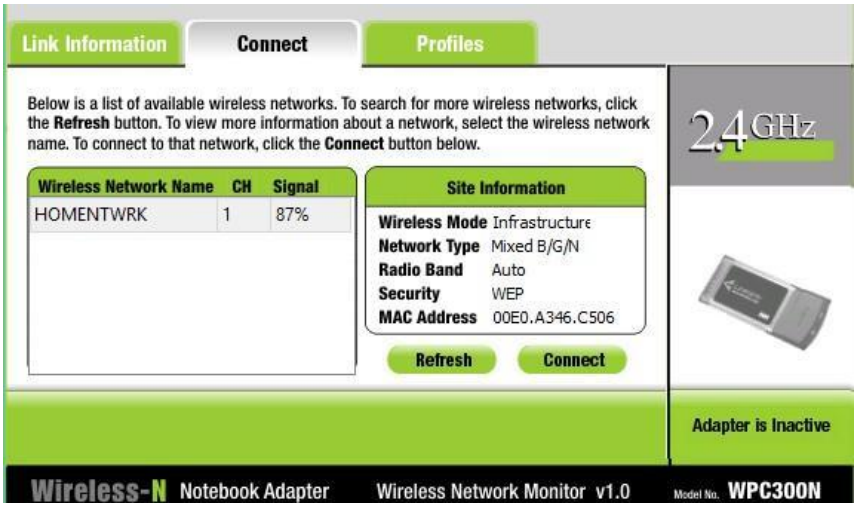


5. Select PC0. Replace Wired LAN adapter with WMP300N.

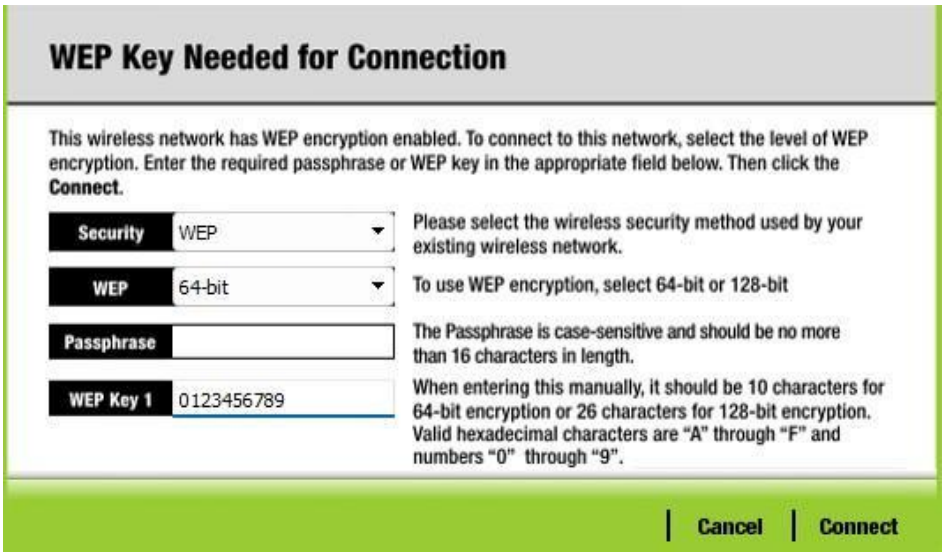
6. Configure PC0. Set IP as Static and assign IP address. Provide Default Gateway as IP of router.



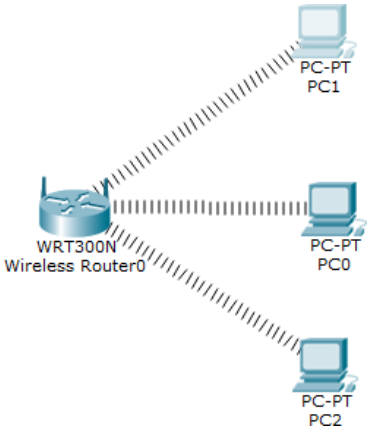
7. Go to PC Wireless in GUI. Here click on Connect tab. Select the Wireless Router to connect.



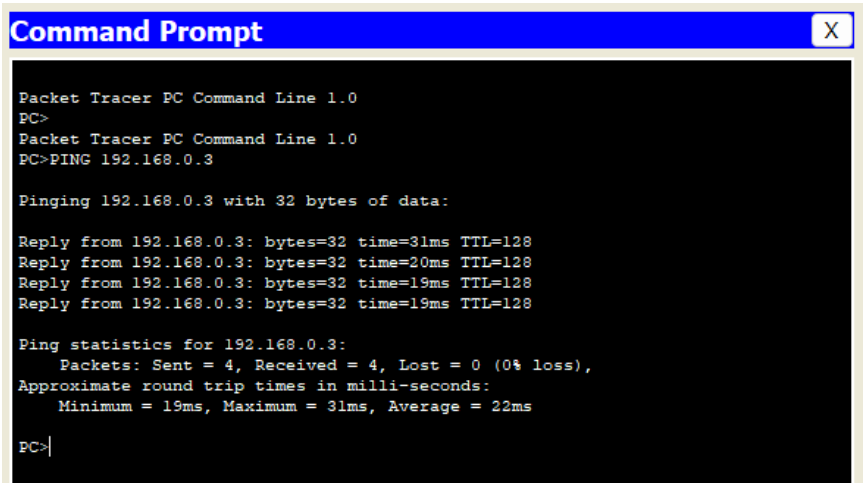
8. Enter the WEP Key 1 entered in the router and press Connect.



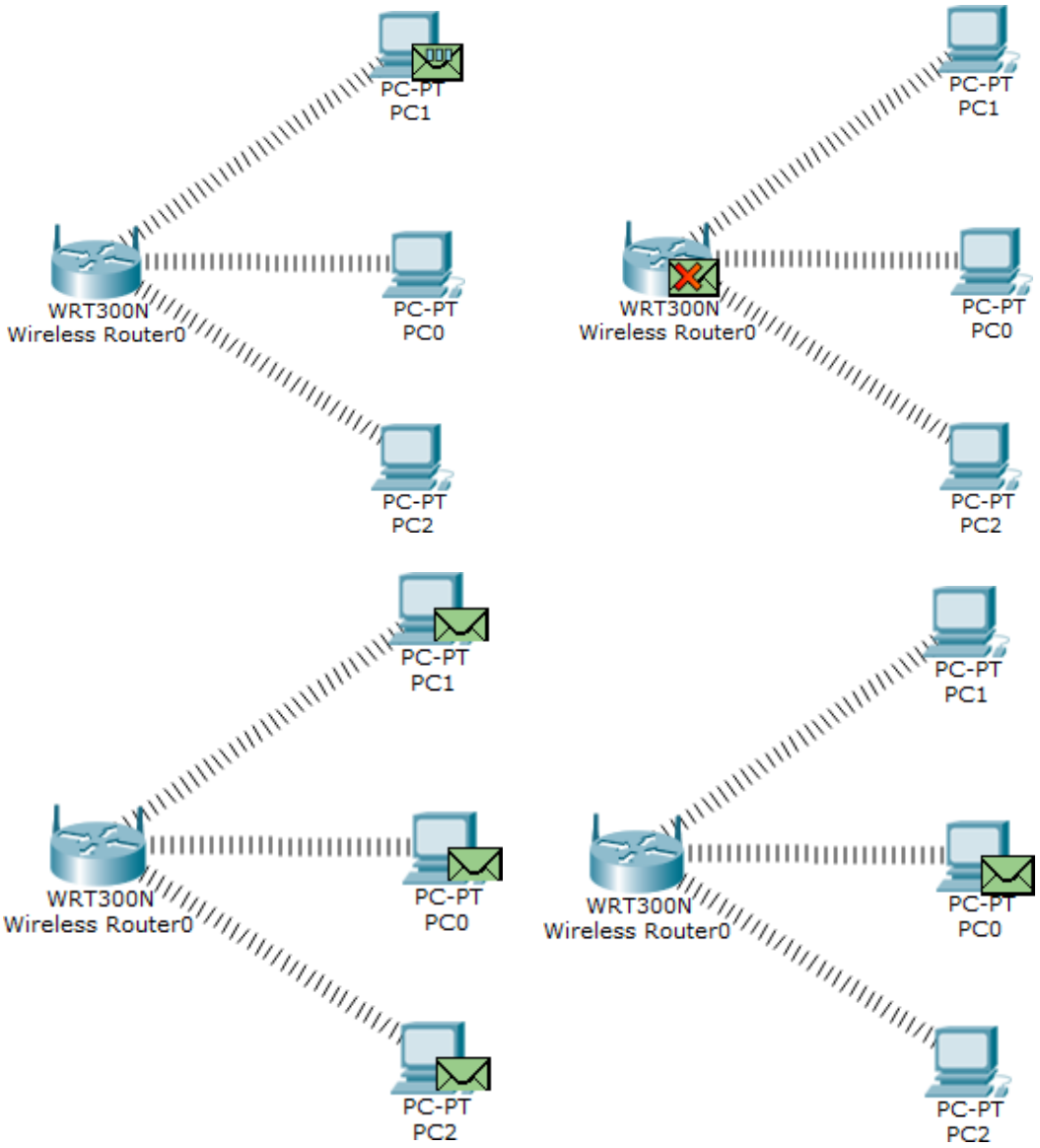
9. Repeat Steps 5 to 8 for PC1 and PC2
   As you can see below, all 3 PCs are connected to the router wirelessly.



10. Ping another PC using command prompt.



11. Visualize packet flow using simulation:



Result:

Thus a simple Wireless LAN Using CISCO Packet Tracer has been executed successfully.

Exp. No. 23	Study of Wireshark Tool
Date: 19.10.24	

Aim:

To analyze and study the network analysis tool wireshark.

Description:

Wireshark

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color coding, and other features that let you dig deep into network traffic and inspect individual packets. You can use Wireshark to inspect a suspicious program's network traffic, analyze the traffic flow on your network, or troubleshoot network problems.

What we can do with Wireshark:

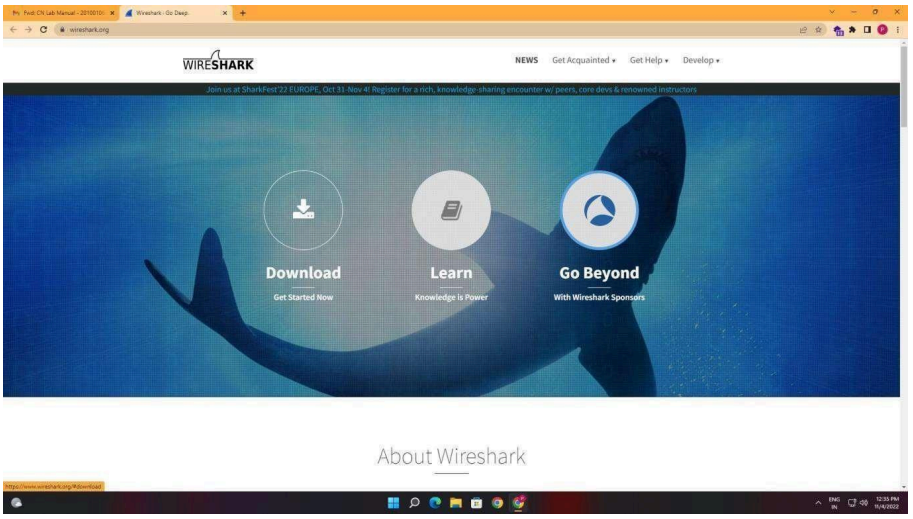
1. Capture network traffic
2. Decode packet protocols using dissectors
3. Define filters – capture and display
4. Watch smart statistics
5. Analyze problems
6. Interactively browse that traffic

Wireshark used for:

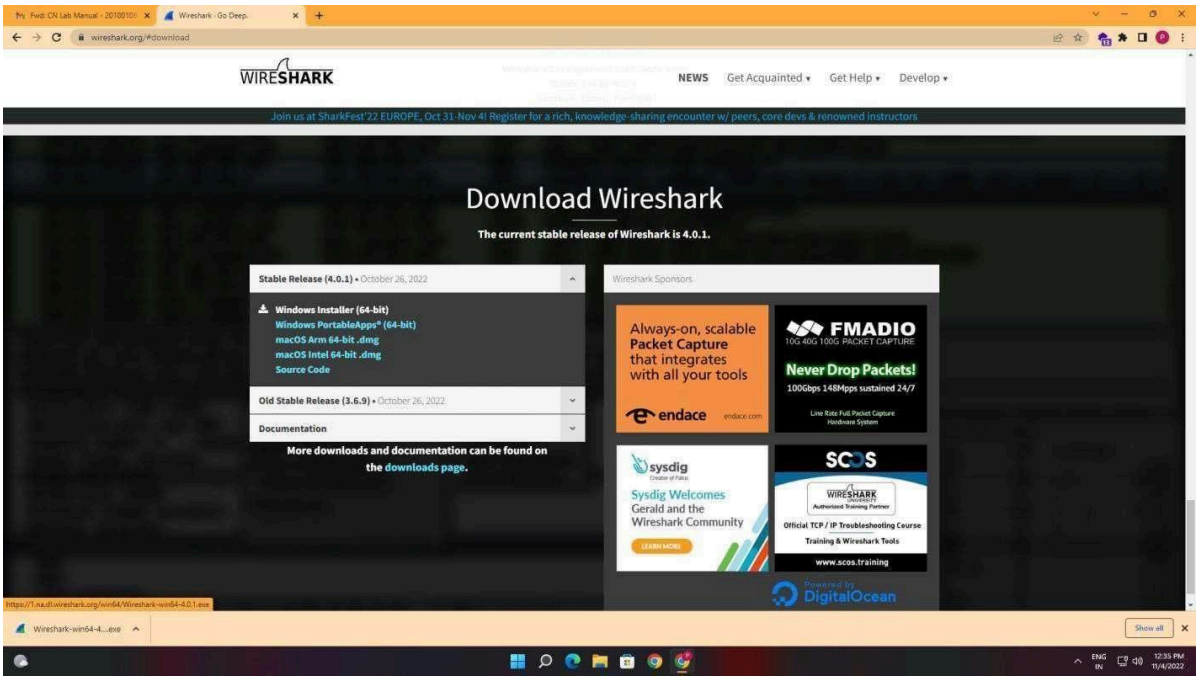
1. Network administrators: troubleshoot network problems
2. Network security engineers: examine security problems
3. Developers: debug protocol implementations
4. People: learn network protocolinternals.

Getting Wireshark:

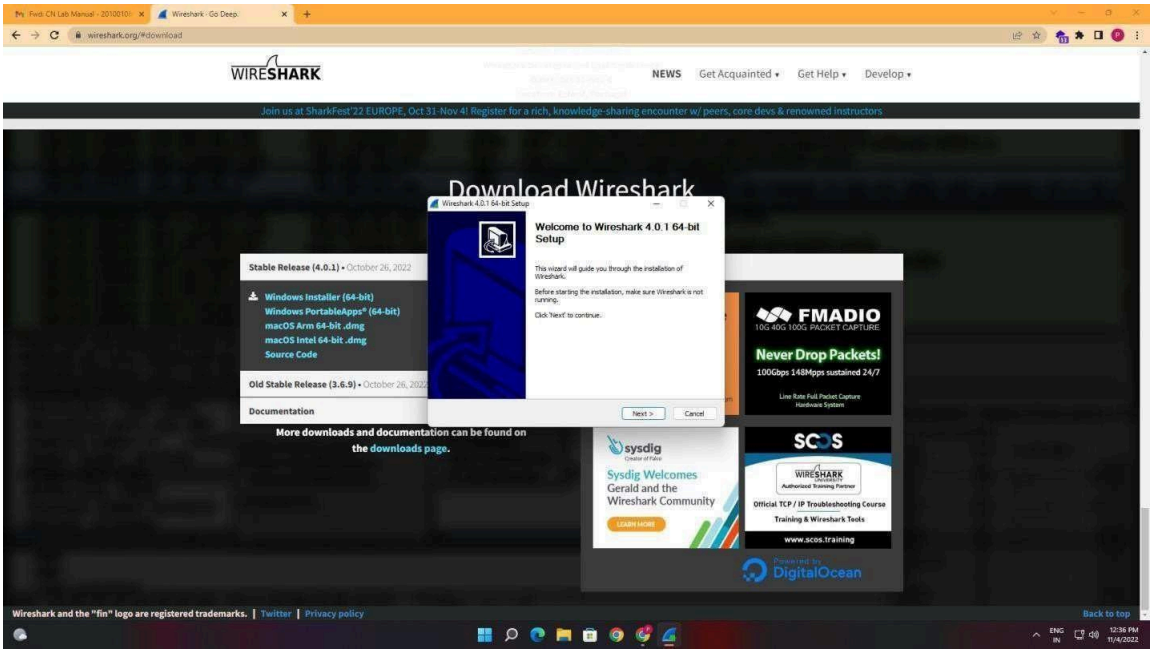
Step 1: Go to the official website of Wireshark.

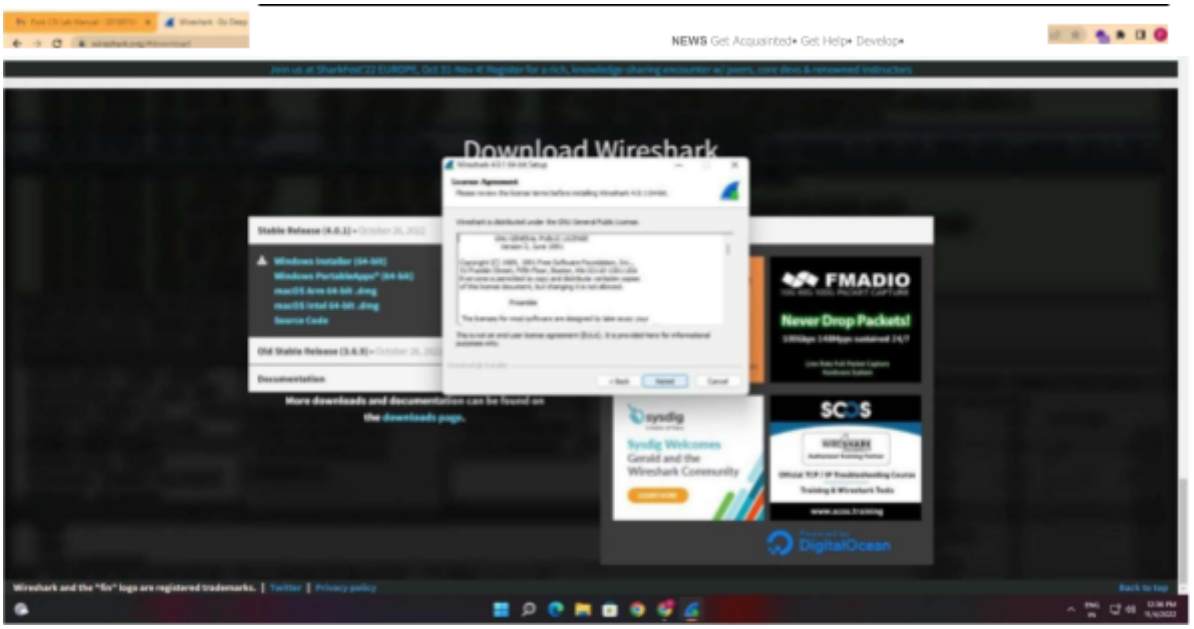


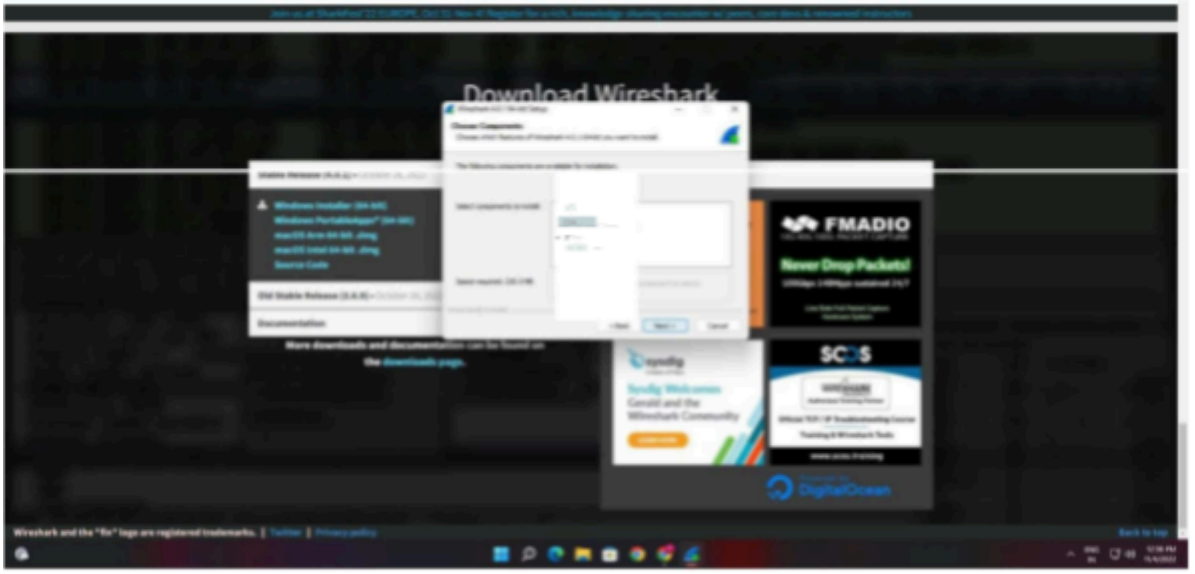
Step 2: Download the stable version of wireshark.

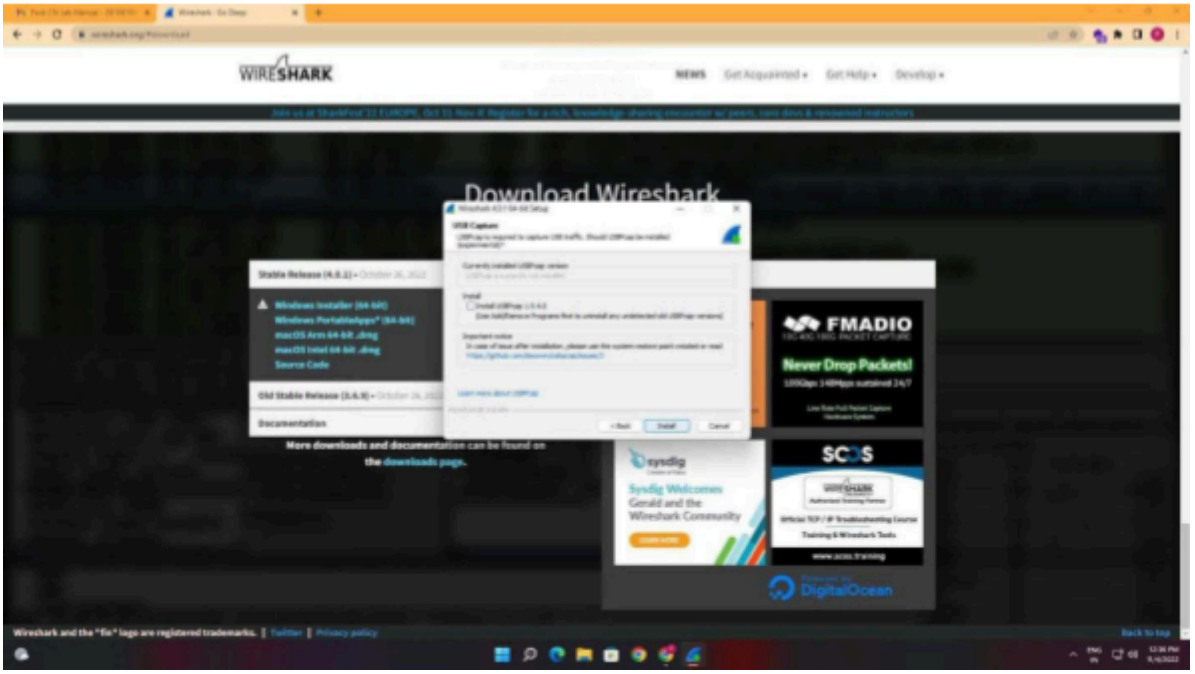


Step 3: Install wireshark by following the below steps.



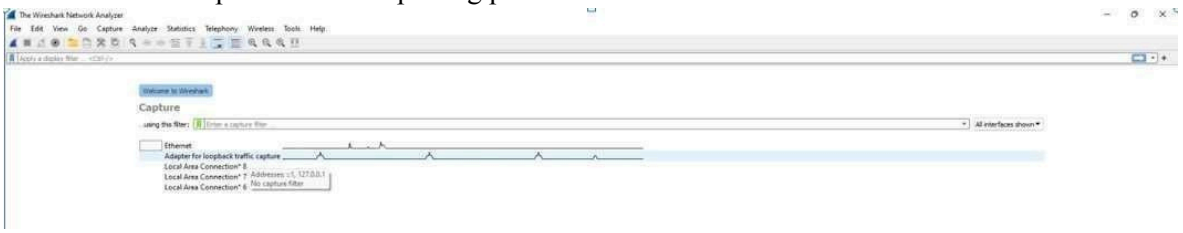

NEWS Get Acquainted▾ Get Help▾ Develop▾
Download Wireshark
FMADIO
Never Drop Packets!
SCOS
sysdig
Sysdig Welcomes Gerald and the Wireshark Community
DigitalOcean
Wireshark and the "fin" logo are registered trademarks.


Download Wireshark
FMADIO
Never Drop Packets!
SCOS
sysdig
DigitalOcean
Wireshark and the "fin" logo are registered trademarks.


WIRESHARK
NEWS Get Acquainted▾ Get Help▾ Develop▾
Download Wireshark
FMADIO
Never Drop Packets!
SCOS
sysdig
DigitalOcean
Wireshark and the "fin" logo are registered trademarks.

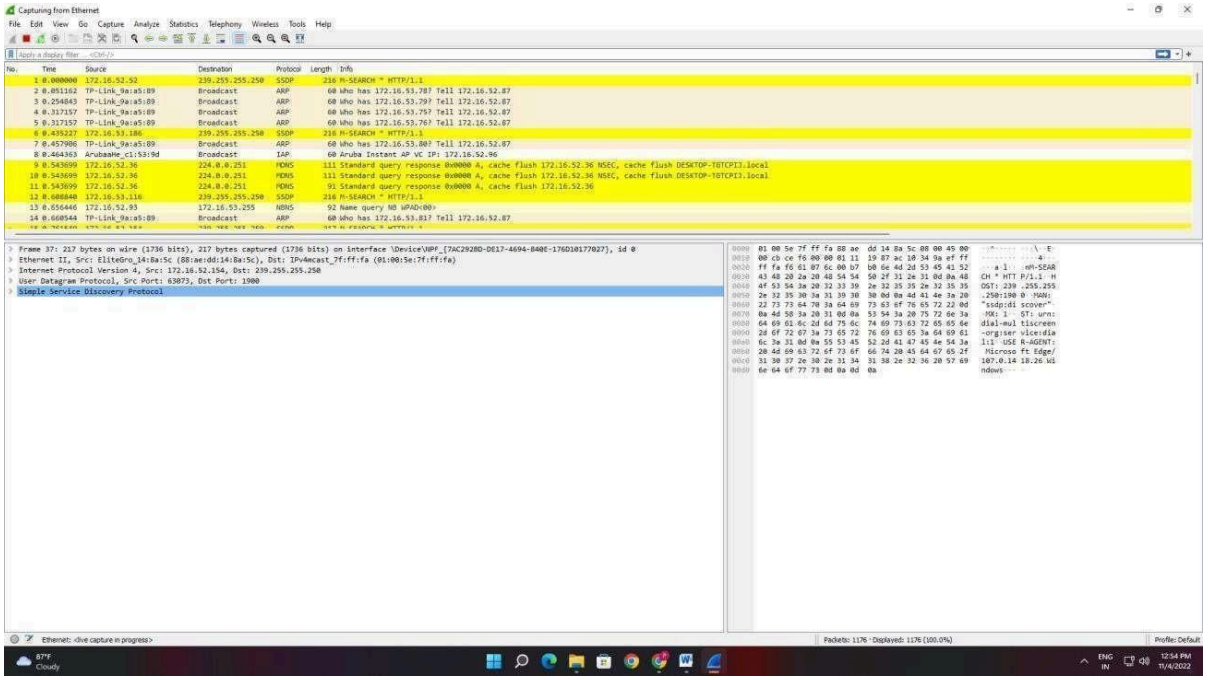
Capturing Packets:

After downloading and installing wireshark, launch it and double-click the name of a network interface under Capture to start capturing packets on that interface.



As soon as you click the interface name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your systems.

If you have promiscuous mode enabled—it's enabled by default—you'll also see all the other packets on the network instead of only packets addressed to your network adapter. To check if promiscuous mode is enabled, click Capture > Options and verify the "Enable promiscuous mode on all interfaces" checkbox is activated at the bottom of this window.



Click the red "Stop" button near the top left corner of the window when you want to stop capturing traffic.

The "Packet List" Pane

The packet list pane displays all the packets in the current capture file. The "Packet List" pane Each line in the packet list corresponds to one packet in the capture file. If you select a line in this pane, more details will be displayed in the "Packet Details" and "Packet Bytes" panes.

The "Packet Details" Pane

The packet details pane shows the current packet (selected in the "Packet List" pane) in a more detailed form. This pane shows the protocols and protocol fields of the packet selected in the "Packet List" pane. The protocols and fields of the packet shown in a tree which can be expanded and collapsed.

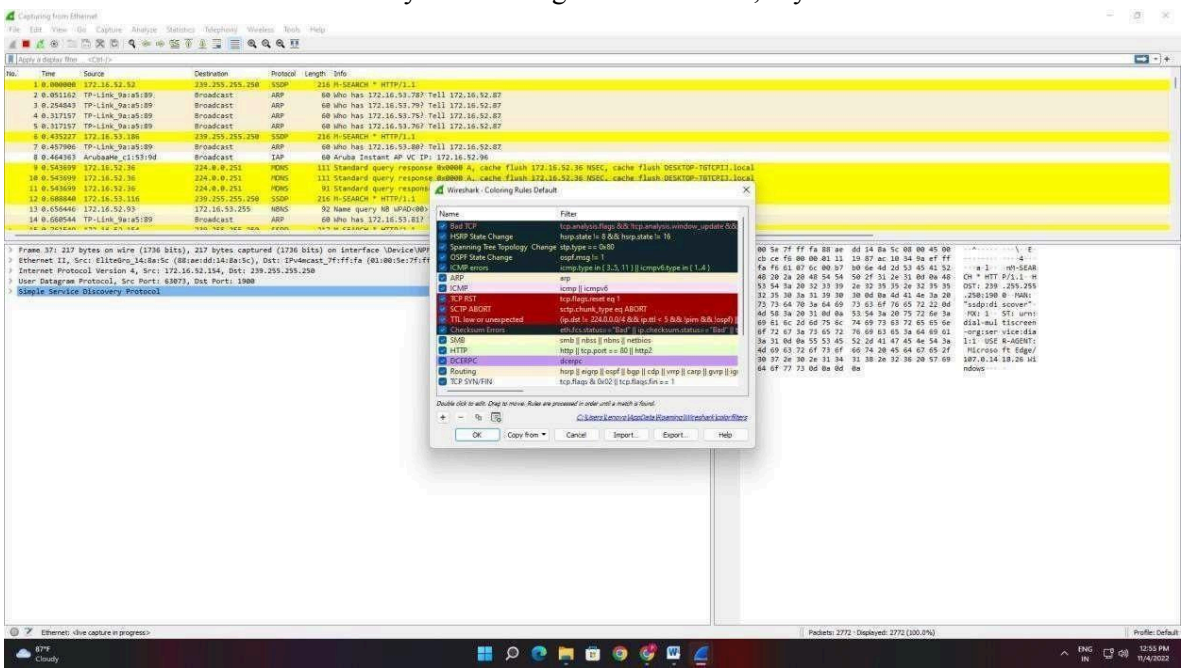
The "Packet Bytes" Pane

The packet bytes pane shows the data of the current packet (selected in the "Packet List" pane) in a hexdump style.

Color Coding

You'll probably see packets highlighted in a variety of different colors. Wireshark uses colors to help you identify the types of traffic at a glance. By default, light purple is TCP traffic, light blue is UDP traffic, and black identifies packets with errors—for example, they could have been delivered out of order.

To view exactly what the color codes mean, click View > Coloring Rules.
You can also customize and modify the coloring rules from here, if you like.



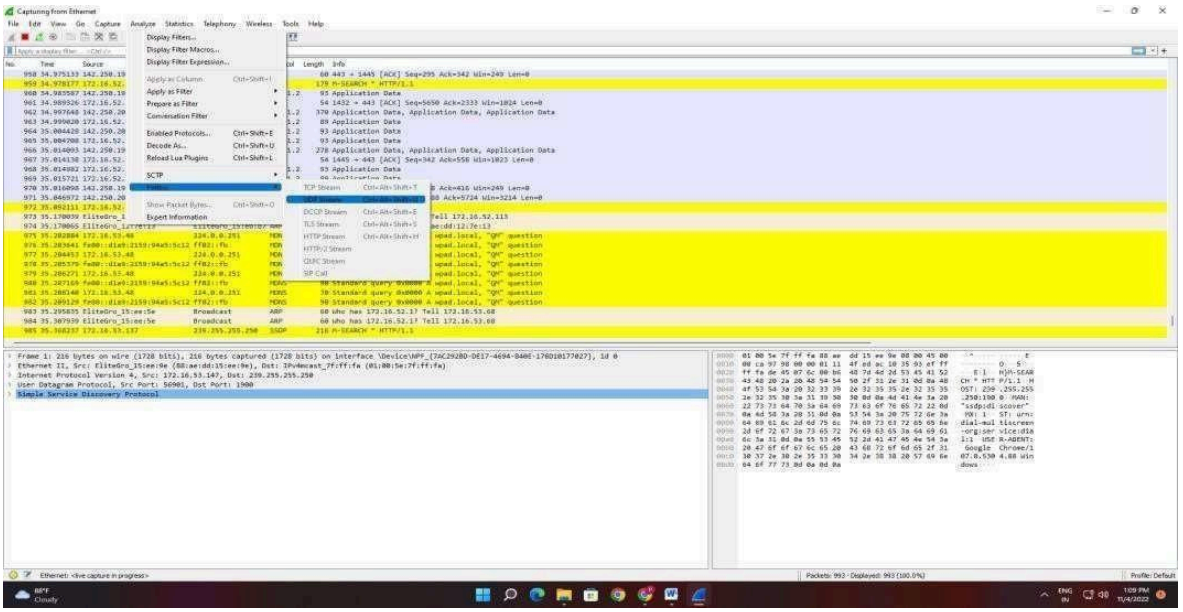
Filtering Packets:

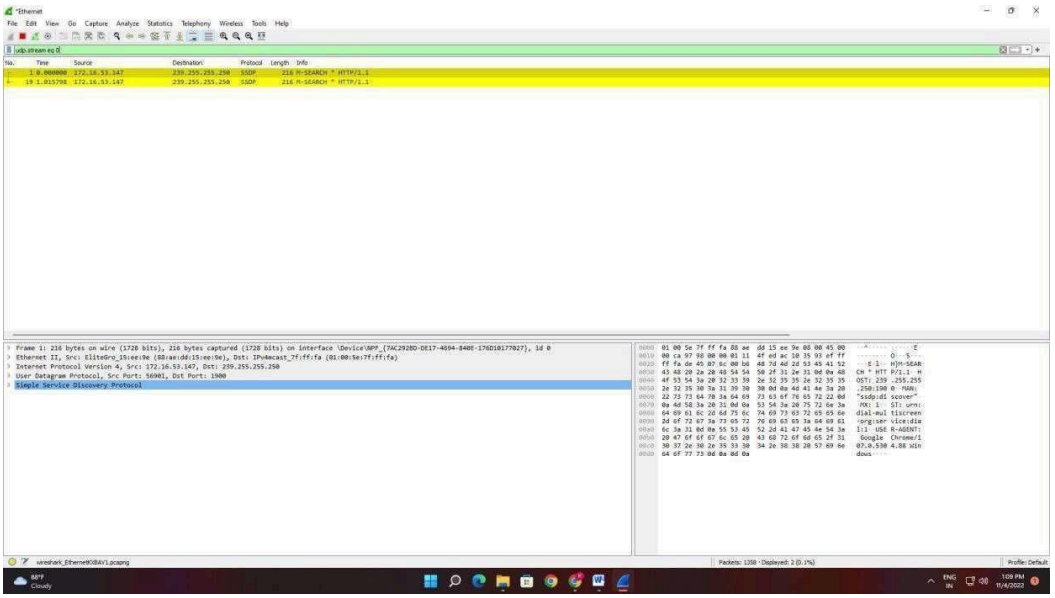
If you're trying to inspect something specific, such as the traffic a program sends when phoning home, it helps to close down all other applications using the network so you can narrow down the traffic. Still, you'll likely have a large amount of packets to sift through. That's where Wireshark's filters come in.

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type "dns" and you'll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.

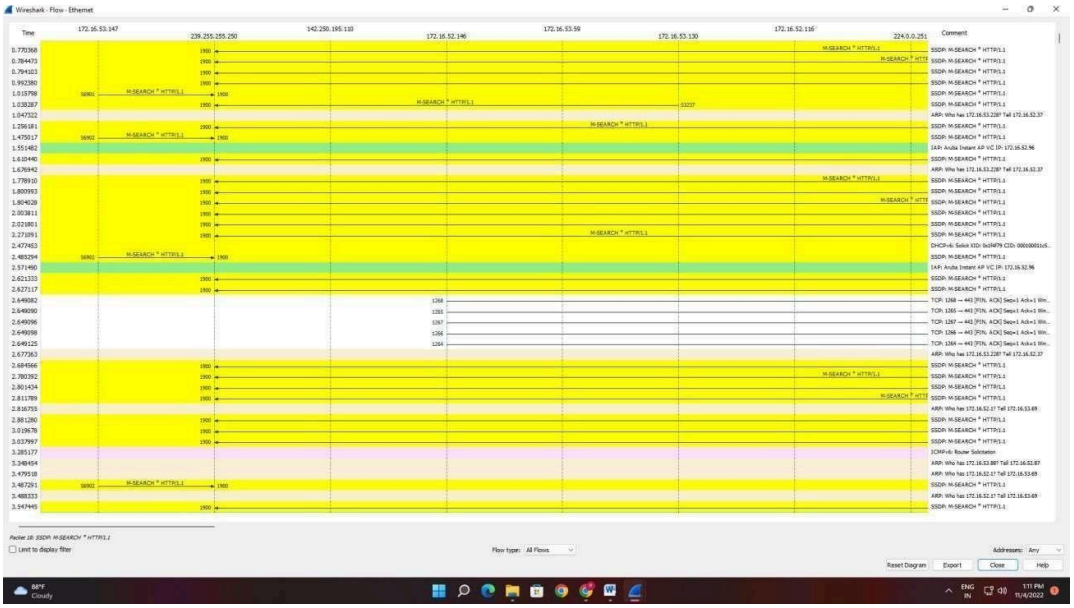
You can also click Analyze > Display Filters to choose a filter from among the default filters included in Wireshark. From here, you can add your own custom filters and save them to easily access them in the future.

Another interesting thing you can do is right-click a packet and select Follow > UDP Stream. You'll see the full UDP conversation between the client and the server. You can also click other protocols in the Follow menu to see the full conversations for other protocols, if applicable.





Flow Graph: Gives a better understanding of what we see.



Result:

Thus the study of wireshark tools has been completed successfully.

Exp. No. 24	Capturing and Analyzing Packets Using Wireshark Tool
Date: 25.10.24	

Aim:

To capture and analyze packets using wireshark tool.

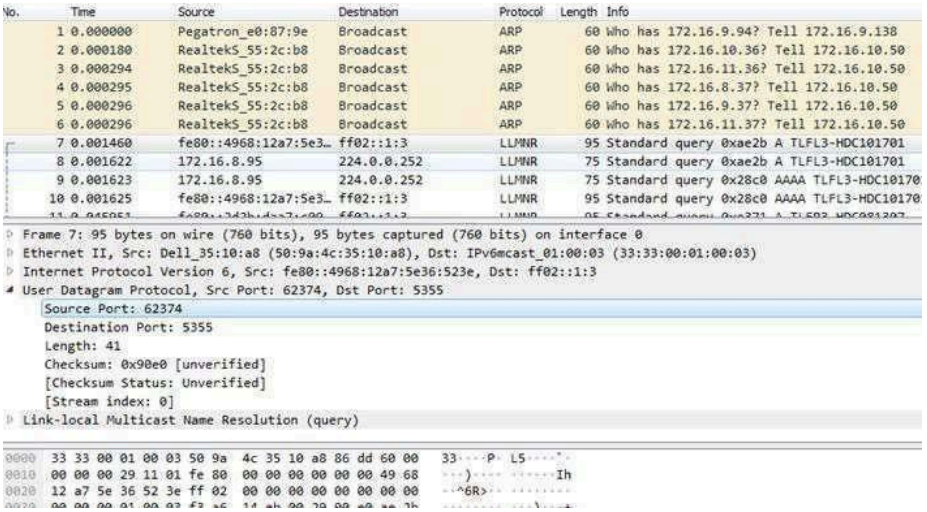
Exercises:

1. Capture 100 packets from the Ethernet: IEEE 802.3 LAN Interface and save it.

   Procedure

   - Select Local Area Connection in wireshark.
   - Go to capture option
   - Select stop capture automatically after 100 packets.
   - Then click start capture.
   - Save the packets.

   Output

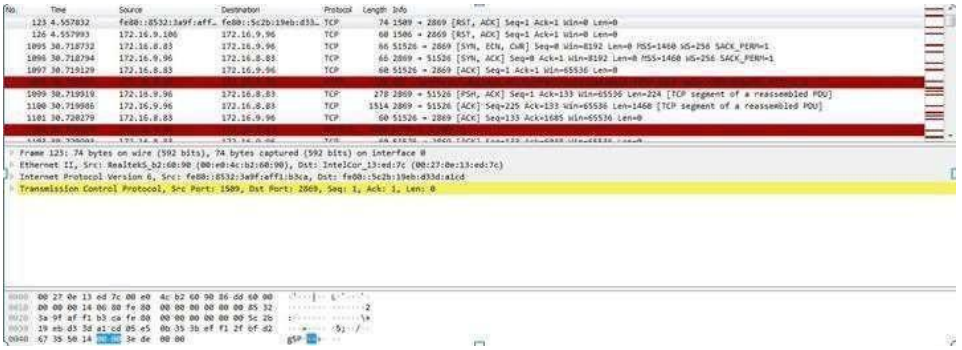


2. Create a Filter to display only TCP/UDP packets, inspect the packets and provide the flow graph.

   Procedure

   - Select Local Area Connection in wireshark.
   - Go to capture option
   - Select stop capture automatically after 100 packets.
   - Then click start capture.
   - Search TCP packets in the search bar.
   - To see the flow graph click the Statistics Flow graph.
   - Save the packets.

   Output



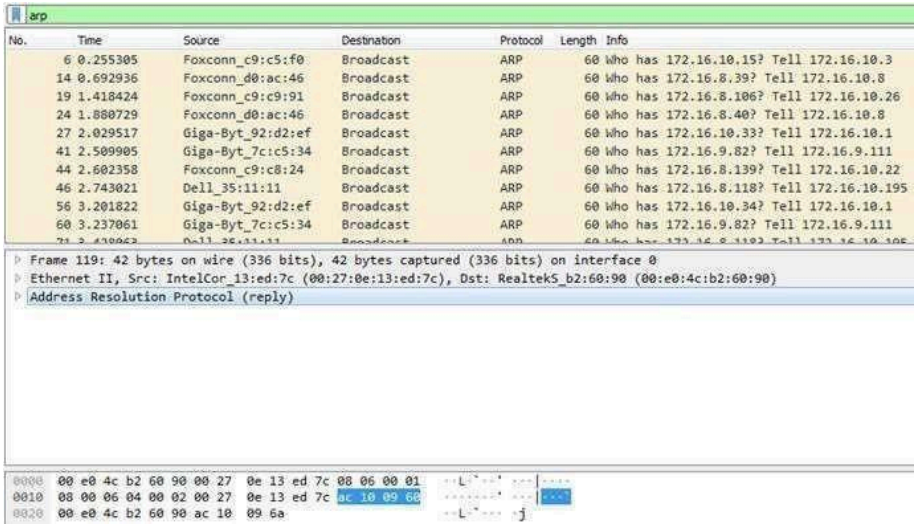
Flow Graph

3. Create a filter to display only ARP Packets and Inspect the Packet.

Procedure

- Select Local Area Connection in wireshark.
- Go to capture option
- Select stop capture automatically after 100 packets.
- Then click start capture.
- Search ARP packets in the search bar.
- Save the packets.

Output

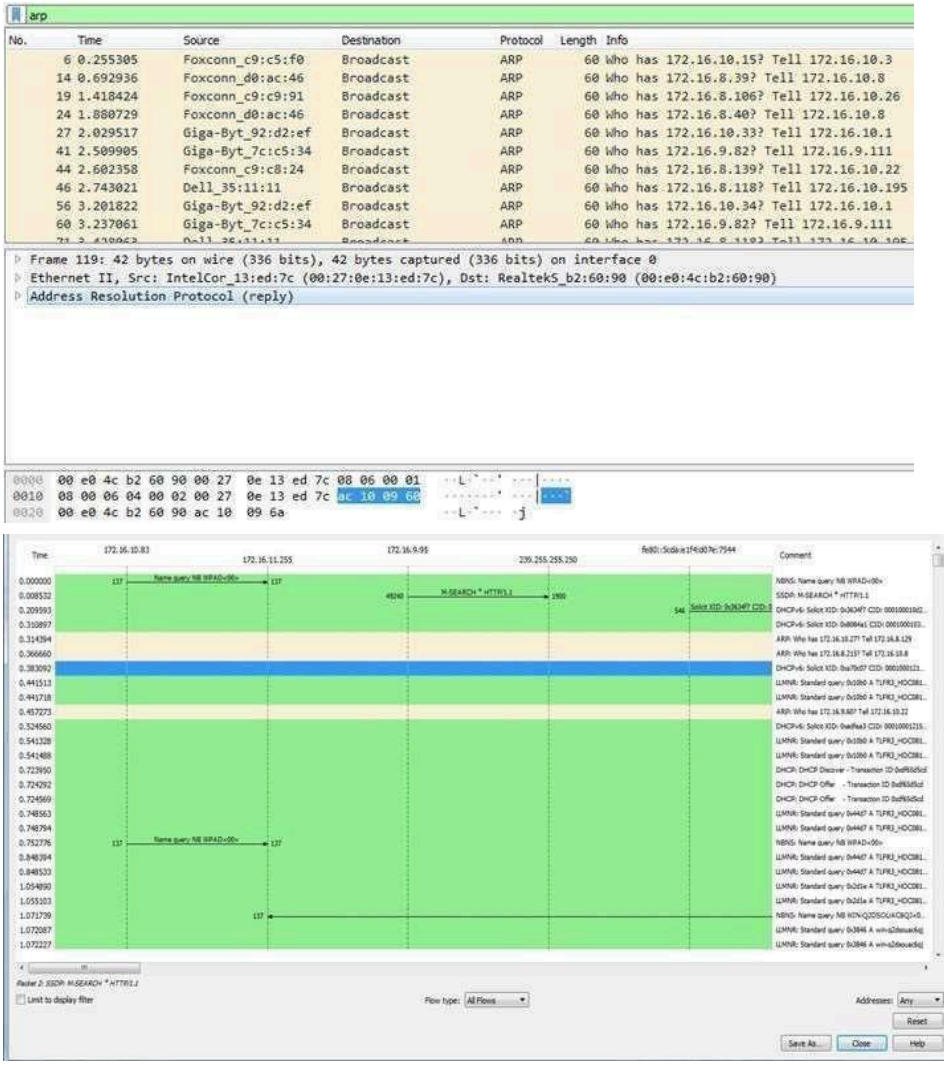


4. Create a Filter to display only DNS packets and provide the Flowgraph.

Procedure

- Select Local Area Connection in wireshark.
- Go to capture option
- Select stop capture automatically after 100 packets.
- Then click start capture.
- Search DNS packets in the search bar.
- To see the flow graph click the Statistics Flow graph.
- Save the packets.

Output

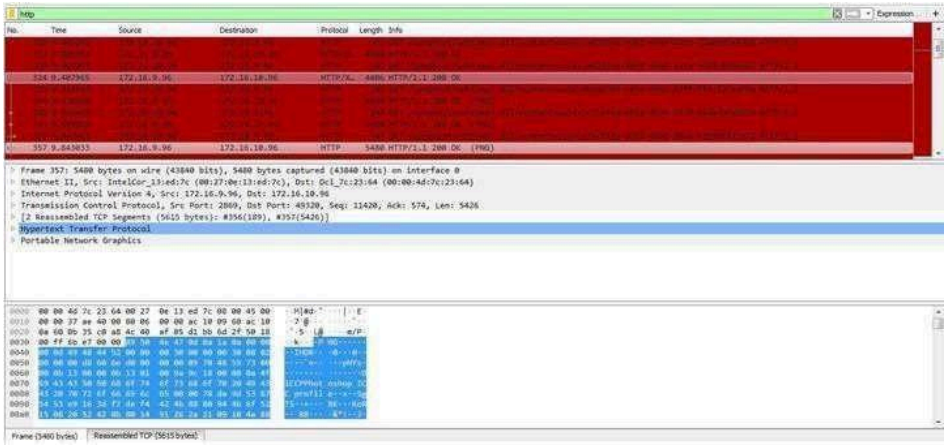


5. Create a Filter to display only HTTP Packets and inspect the packet.

Procedure

- Select Local Area Connection in wireshark.
- Go to capture option
- Select stop capture automatically after 100 packets.
- Then click start capture.
- Search HTTP packets in the search bar.
- Save the packets.

Output

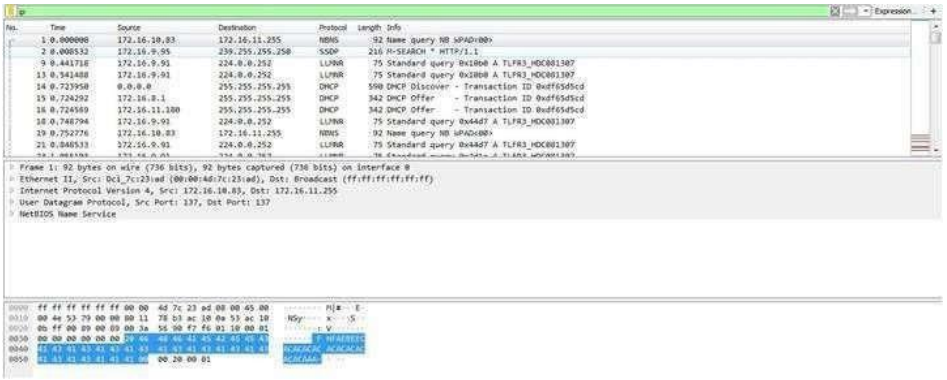


6. Create a Filter to display only IP/ICMP Packets and inspect the packet.

Procedure

- Select Local Area Connection in wireshark.
- Go to capture option
- Select stop capture automatically after 100 packets.
- Then click start capture.
- Search ICMP/IP packets in the search bar.
- Save the packets.

Output

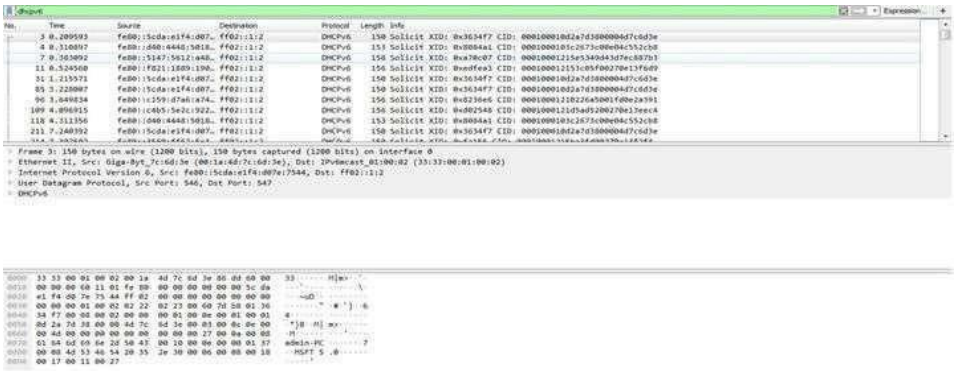


7. Create a Filter to Display only the DHCP Packets and inspect the packet.

Procedure

- Select Local Area Connection in wireshark.
- Go to capture option
- Select stop capture automatically after 100 packets.
- Then click start capture.
- Search HTTP packets in the search bar.
- Save the packets.

Output



Result:

Thus the capturing and analyzing of packets using wireshark tools has been executed successfully.

Exp. No. 25	Analyze Weblogs Using Webalizer
Date: 26.10.24	

Aim:

To analyze weblogs using webalizer.

Description:

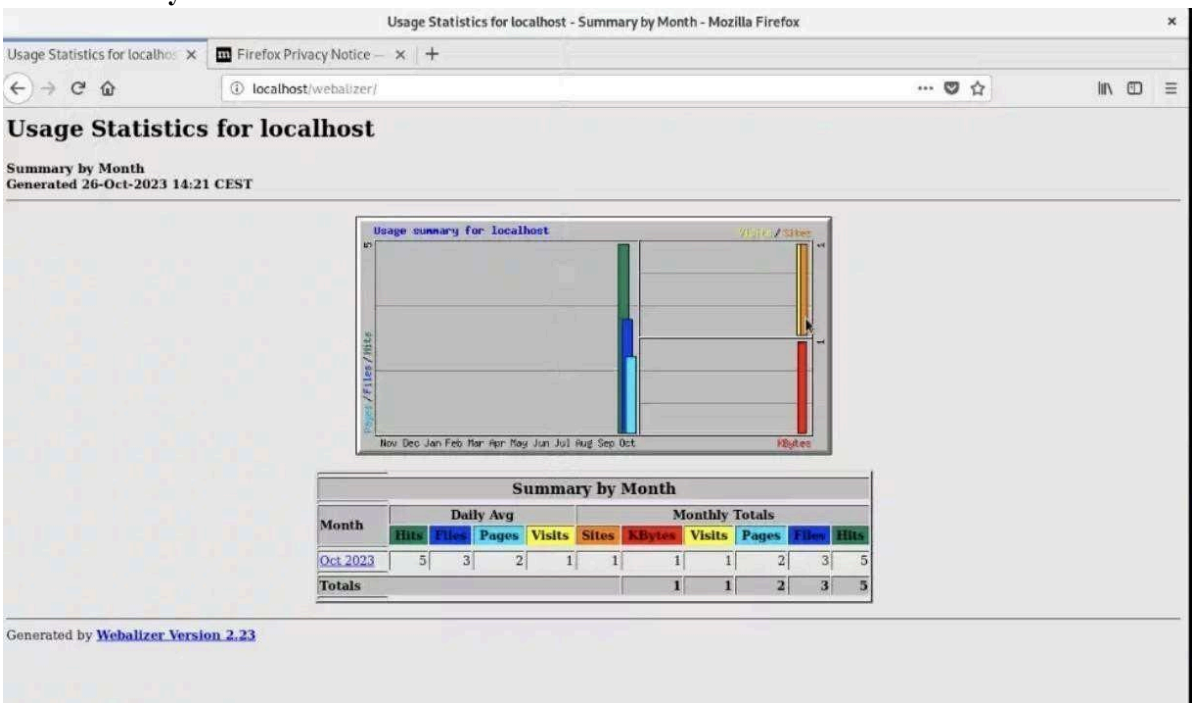
The Webalizer is a web log analysis software, which generates webpages of analysis, from access and usage logs. It is one of the most commonly used web server administrative tools.

Procedure:

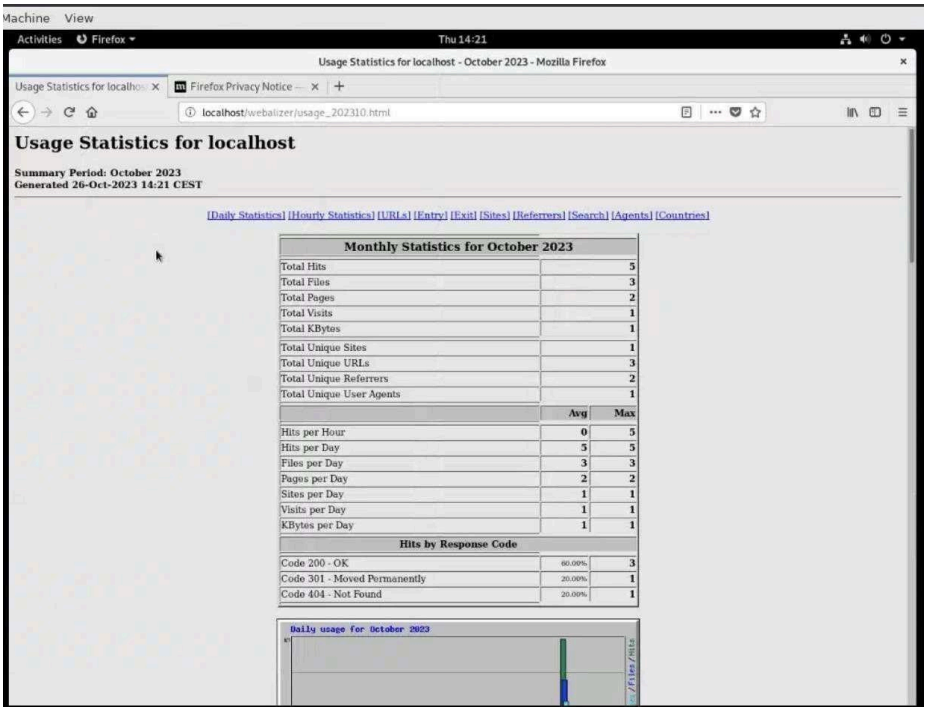
1. Open your Linux/Fedora terminal.
2. Switch to the root user using the 'su' command.
3. Install the Apache HTTP server using the DNF package manager command: dnf install httpd
4. Open the SELinux configuration file using the command: vi /etc/sysconfig/selinux
5. Change the SELinux configuration to 'permissive' mode from 'enforcing' mode as: SELinux=permissive
6. Enable the Apache HTTP service using the command: systemctl enable httpd.service
7. Start the Apache HTTP service using the command: systemctl start httpd.service
8. Create an output directory for the webalizer reports using the command: mkdir /var/www/html/webalizer
9. Install Webalizer using DNF as: dnf install webalizer
10. Copy all the files of the usage directory to the html directory using the command:cp* /var/www/usage /var/www/html
11. Open the Webalizer configuration file using the command: vi /etc/webalizer.conf
12. Inside the file,
    a. Uncomment the command: LogType clf
    b. Uncomment the command: Hostname localhost
    c. Change the OutputDir to: OutputDir /var/www/html/webalizer
    d. Ensure that the following line is present: LogFile /var/log/httpd/access_log
13. After that, restart the Apache HTTP service using the command: systemctl restart httpd.service
14. Open a browser and navigate to the Link http://localhost/webalizer
15. Click on a month to get its reports.

Output:

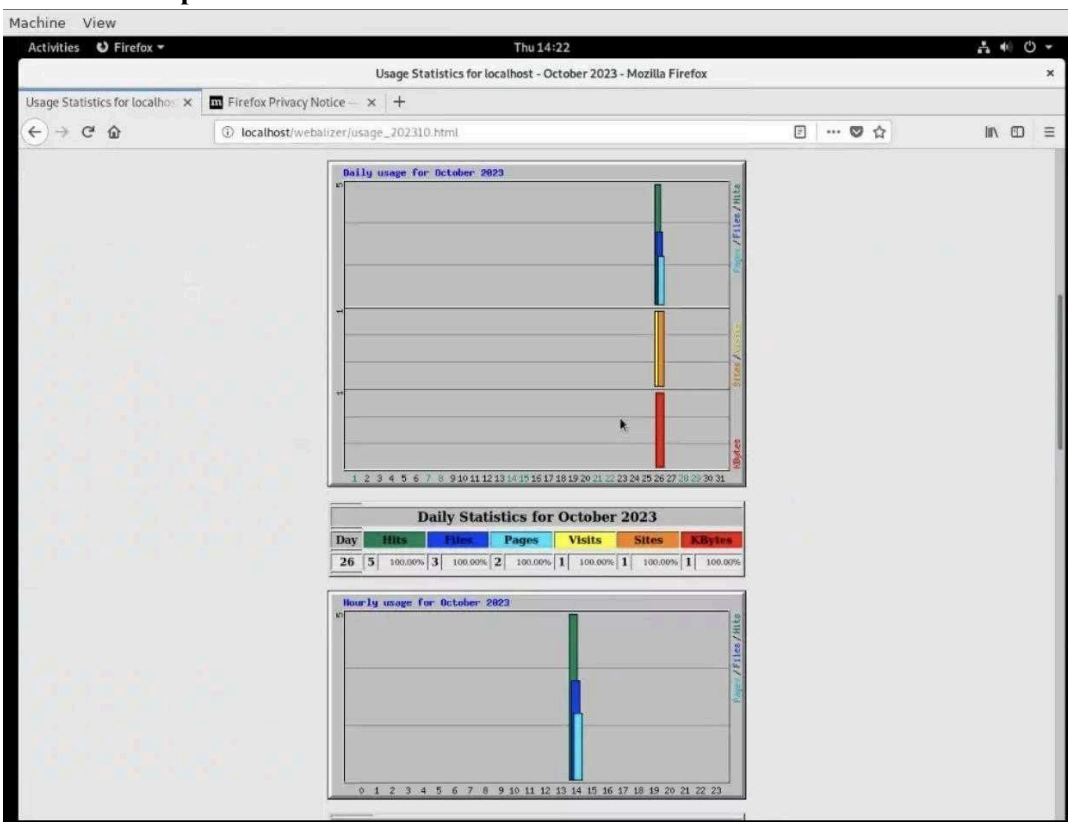
Month-wise Summary:



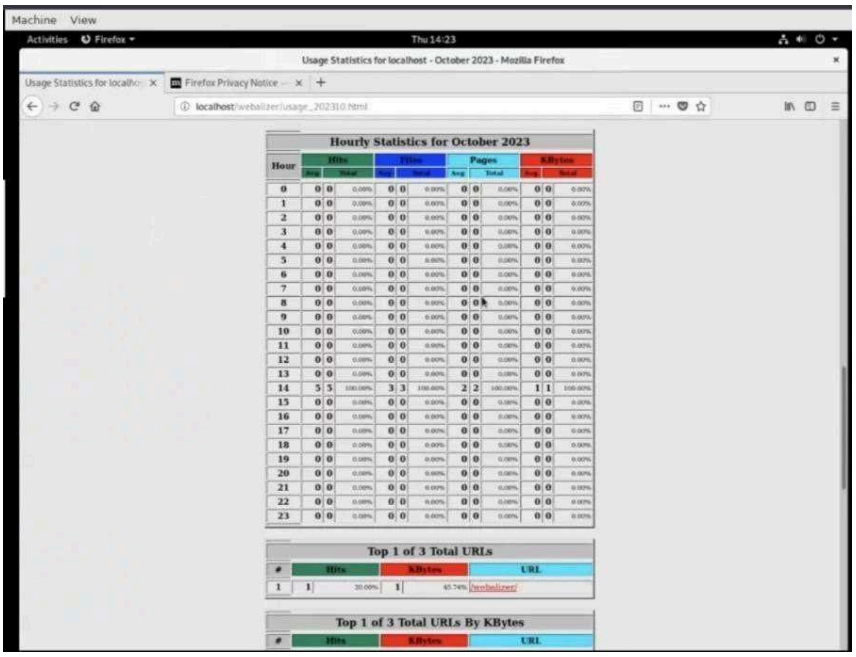
Monthly Statistics for a Specific Month:



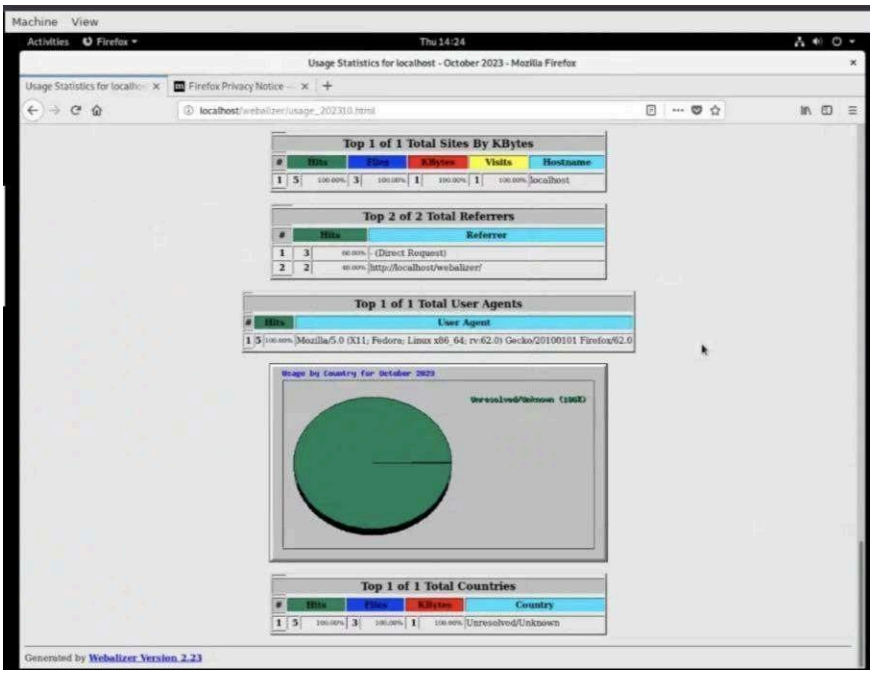
Daily Statistics for a Specific Month:



Hourly Statistics:



User Agents and Countries:



Result:

Thus the analyze of weblogs using webalizer has been executed successfully.