Experiment no – 5

Title- Implementation of data link layer protocol

Objective- Configuring HDLC protocol for data communication

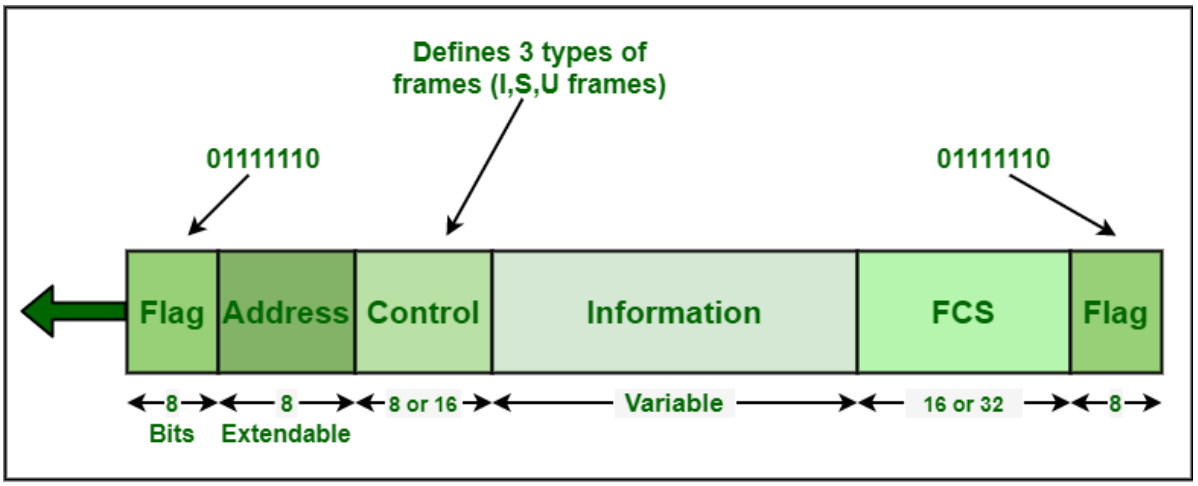
Aim- Implementing HDLC protocol and configuring it for demonstrating data link layers communication

Software used- Cisco Packet Tracer

Theory-

High-Level Data Link Control (HDLC) generally uses term "frame" to indicate and represent an entity of data or a protocol of data unit often transmitted or transferred from one station to another station. Each and every frame on link should begin and end with Flag Sequence Field (F). Each of frames in HDLC includes mainly six fields. It begins with a flag field, an address field, a control field, an information field, an frame check sequence (FCS) field, and an ending flag field. The ending flag field of one frame can serve as beginning flag field of the next frame in multiple-frame transmissions.

The basic frame structure of HDLC protocol is shown below:



Basic Frame Structure

Let us understand these fields in details:

Flag Field –

The flag field is generally responsible for initiation and termination of error checking. In HDLC protocol, there is no start and stop bits. So, the flag field is basically using delimiter 0x7e to simply indicate beginning and end of frame.

It is an 8-bit sequence with a bit pattern 01111110 that basically helps in identifying both starting and end of a frame. This bit pattern also serves as a synchronization pattern for receiver. This bit pattern is also not allowed to occur anywhere else inside a complete frame.

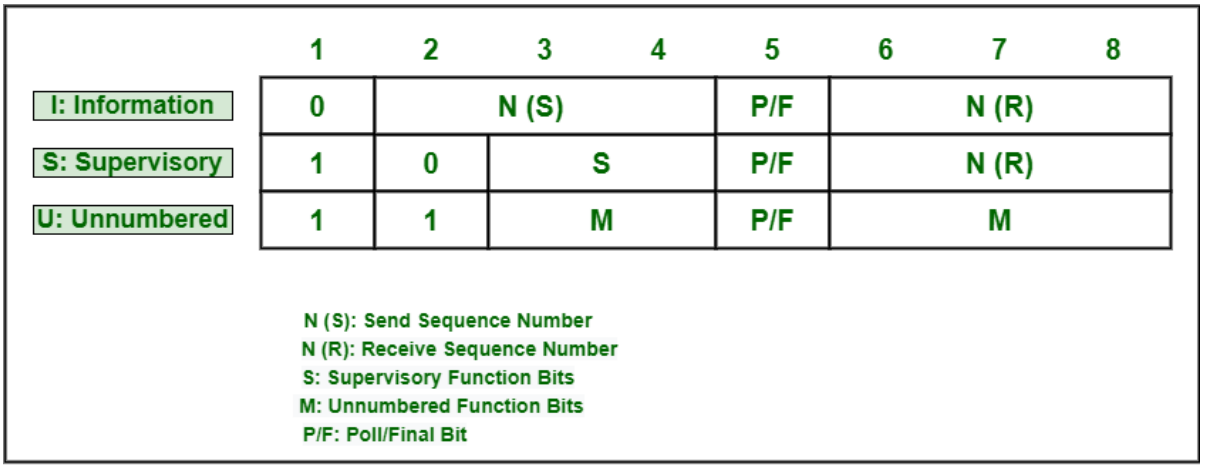
Address Field –

The address field generally includes HDLC address of secondary station. It helps to identify secondary station will sent or receive data frame. This field also generally consists of 8 bits therefore it is capable of addressing 256 addresses. This field can be of 1 byte or several bytes long, it depends upon requirements of network. Each byte can identify up to 128 stations.

This address might include a particular address, a group address, or a broadcast address. A primary address can either be a source of communication or a destination that eliminates requirement of including address of primary.

Control Field –

HDLC generally uses this field to determine how to control process of communication. The control field is different for different types of frames in HDLC protocol. The types of frames can be Information frame (I-frame), Supervisory frame (S-frame), and Unnumbered frame (U- frame).



Control Field Format

This field is a 1-2-byte segment of frame generally requires for flow and error control. This field basically consists of 8 bits but it can be extended to 16 bits. In this field, interpretation of bits usually depends upon the type of frame.

Information Field –

This field usually contains data or information of users sender is transmitting to receiver in an I-frame and network layer or management information in U-frame. It also consists of user's data and is fully transparent. The length of this field might vary from one network to another network.

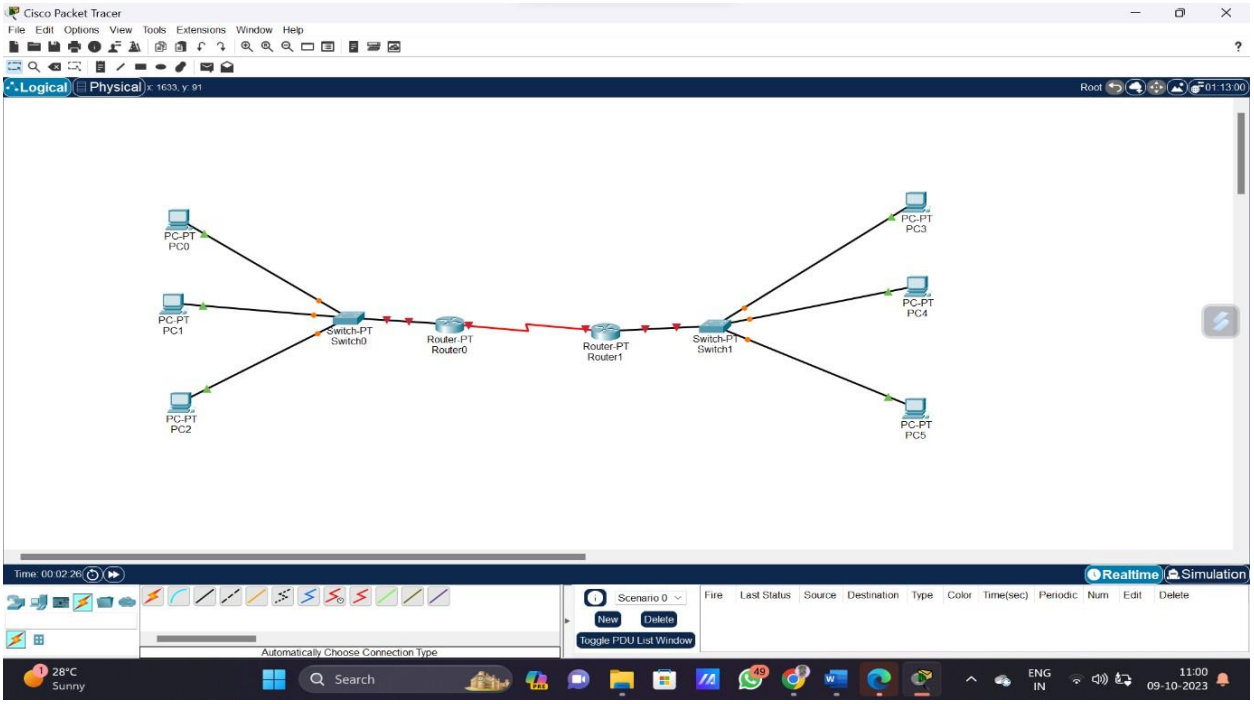
Information field is not always present in an HDLC frame.

Frame Check Sequence (FCS) –

FCS is generally used for identification of errors i.e., HDLC error detection. In FCS, CRC16 (16-bit Cyclic Redundancy Check) or CRC32 (32-bit Cyclic Redundancy Check) code is basically used for error detection. CRC calculation is done again in receiver. If somehow result differs even slightly from value in original frame, an error is assumed.
This field can either contain 2 byte or 4 bytes. This field is a total 16 bit that is required for error detection in address field, control field, and information field. FCS is basically calculated by sender and receiver both of a data frame. FCS is used to confirm and ensure that data frame was not corrupted by medium that is used to transfer frame from sender to receiver.

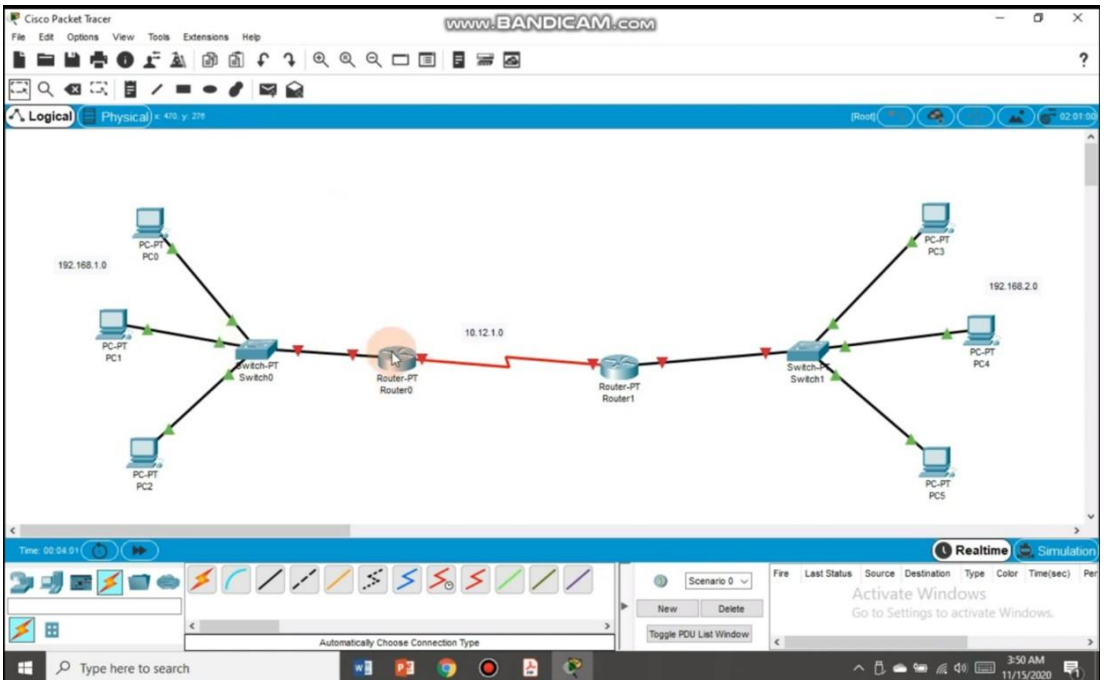
Procedure Step I:
Make a network using 3 end devices and connect them using switch and connect switch to router

Similarly create another network and connect both networks using serial wires.



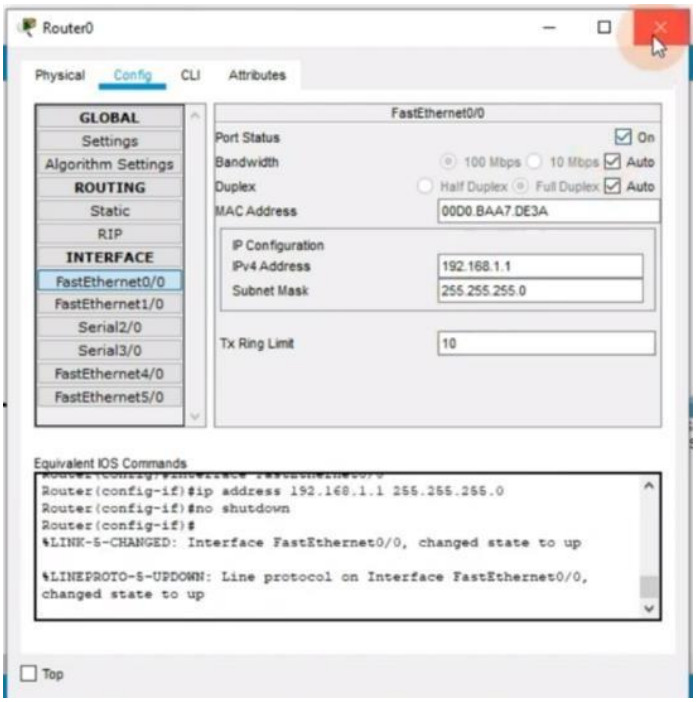
Step2-

Configure each end device with their respective network IP addresses



Step3-

Configure the router in fast ethernet section with network gateway IP address

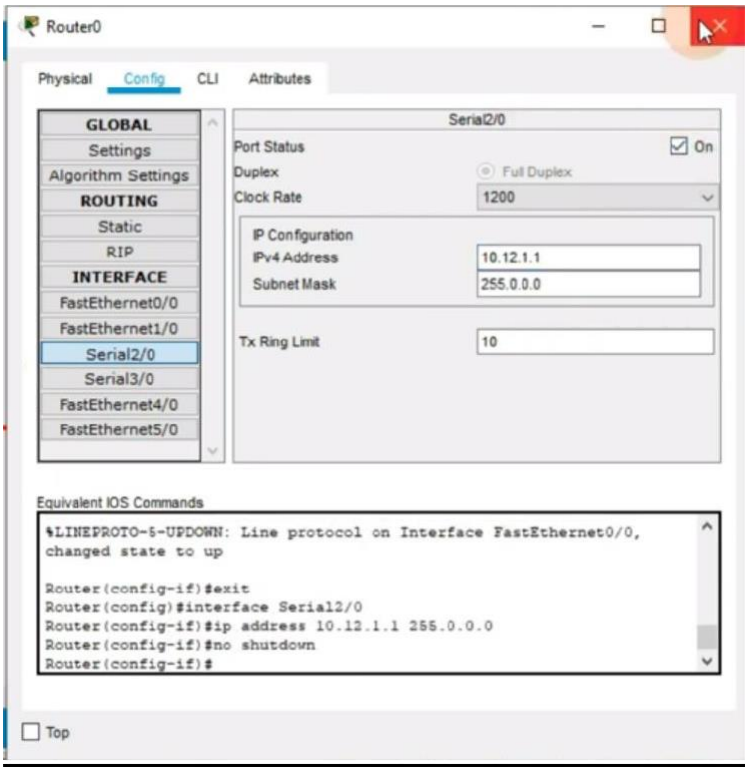


Similarly configure router 2 with its network gateway IP address

Step4-

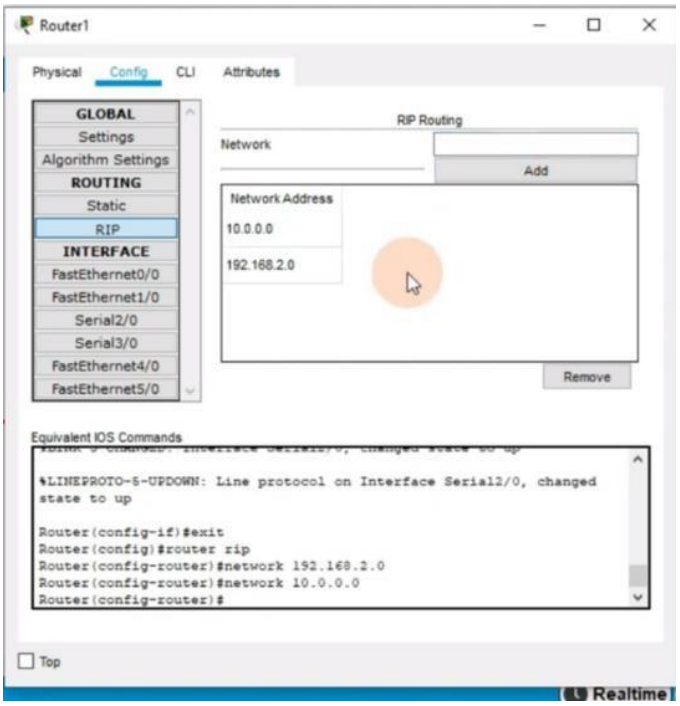
Now in serial 2/0 section configure router to router network ip address and select appropriate clock rate

And make port status on



Similarly do it for router 1

Step5-Configure Routing information in to router 0: add Network IP 192.168.1.0 and 10.0.0.0

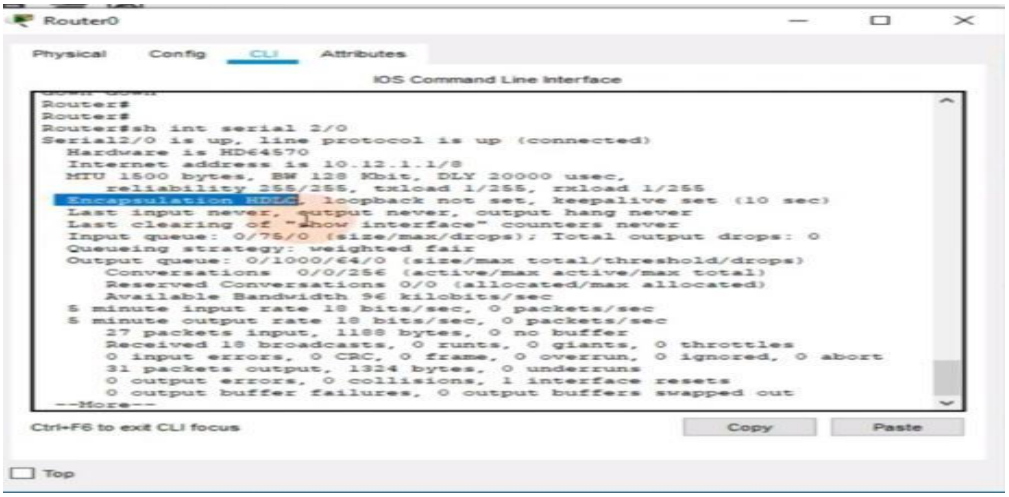


Similarly do it for router 1Network IP-192.168.1.0 and 10.0.0.0

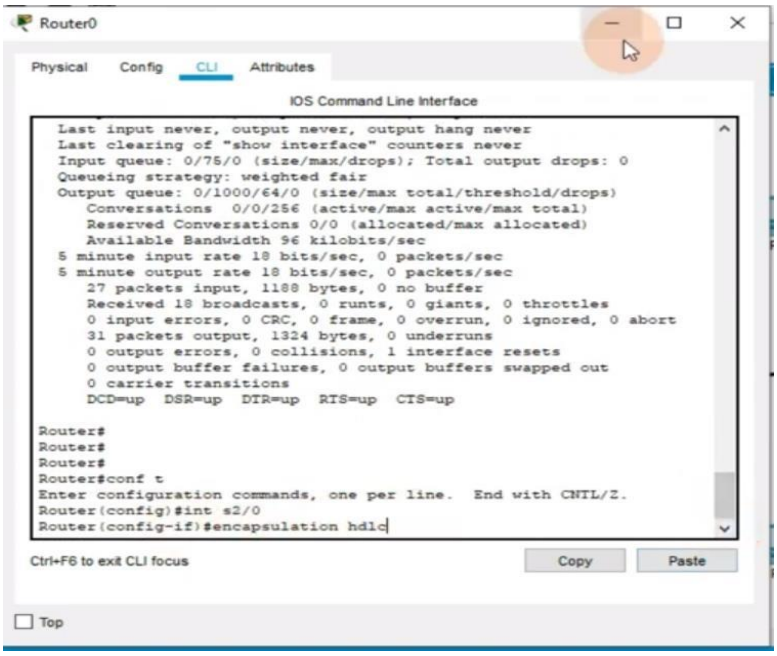
Step6-

Check weather HDLC encapsulation is active in both the router using command line interface of routers

Use command Sh int serial 2/0



Step 7-



In case HDLC encapsulation is not set in your router run configuration commands on CLI window
Conf t
Int S2/0
Encapsulation HDLC
Note do not run if already configured
Similarly do it for router 1

Conclusion:

EXPERIMENT NO.6

TITLE: Socket Programming in python(C/C++) on TCP Client, TCP Server.

OBJECTIVE: To study configuration of socket programming

AIM: To practice the communication between client and server.

SOFTWARE USED: Python 3.6 version

THEORY:

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

This chapter gives you understanding on most famous concept in Networking - Socket Programming.

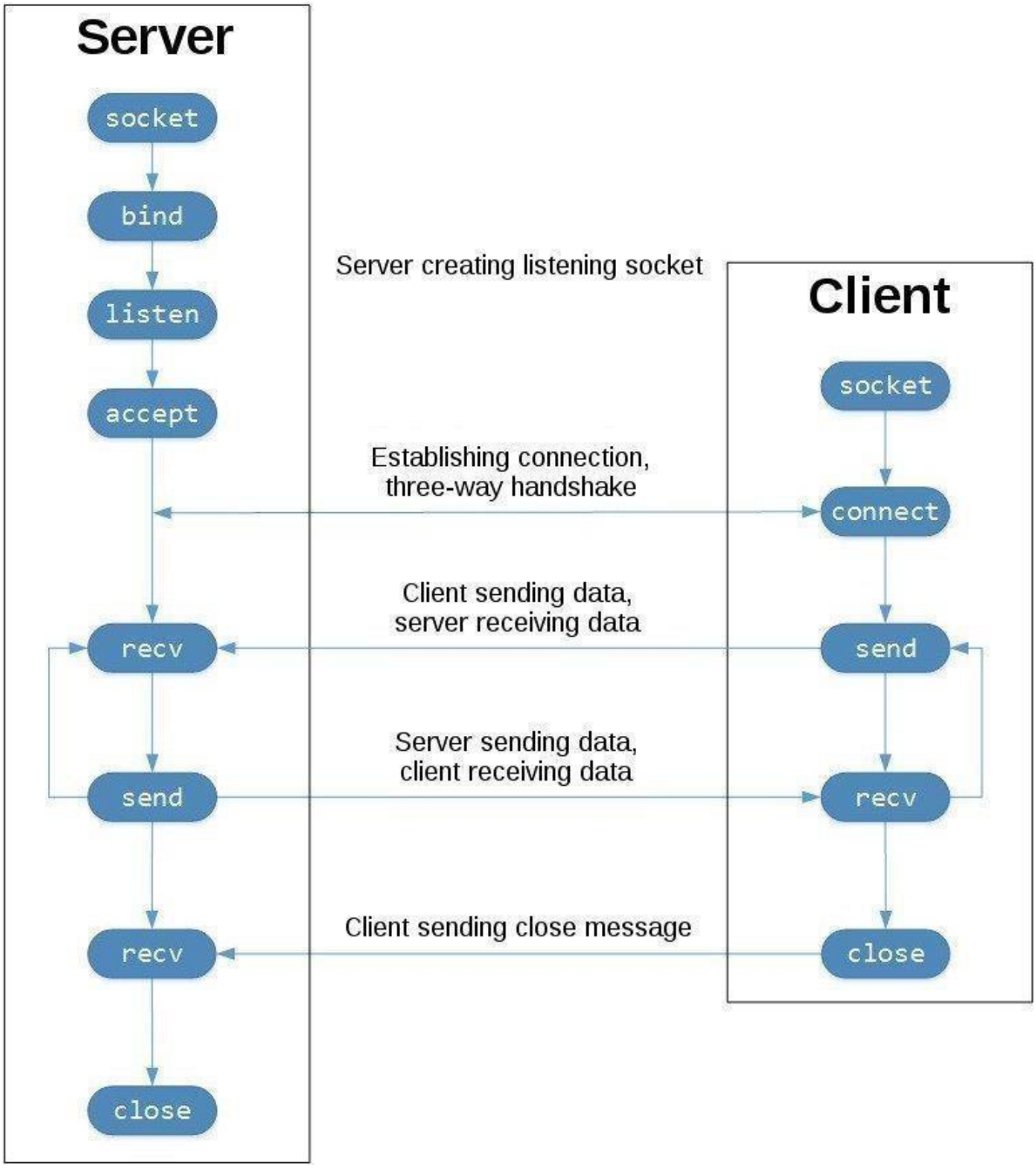
What is Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

TCP Sockets

In the diagram below, let's look at the sequence of socket API calls and data flow for TCP:



TCP Socket Flow

The left-hand column represents the server. On the right-hand side is the client.

- Starting in the top left-hand column, note the API calls the server makes to setup a “listening” socket:
- socket()

- bind()
- listen()
- accept()

A listening socket does just what it sounds like. It listens for connections from clients. When a client connects, the server calls accept() to accept, or complete, the connection.

The client calls connect() to establish a connection to the server and initiate the three-way handshake. The handshake step is important since it ensures that each side of the connection is reachable in the network, in other words that the client can reach the server and vice-versa. It may be that only one host, client or server, can reach the other.

In the middle is the round-trip section, where data is exchanged between the client and server using calls to send() and recv().

At the bottom, the client and server close() their respective sockets. Sockets have their own vocabulary –

Sr.No.	Term & Description
1	**Domain** The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
2	**type** The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
3	**protocol** Typically zero, this may be used to identify a variant of a protocol within a domain and type.

4	**hostname** The identifier of a network interface − • A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation • A string "<broadcast>", which specifies an INADDR_BROADCAST address. • A zero-length string, which specifies INADDR_ANY, or • An Integer, interpreted as a binary address in host byte order.
5	**port** Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The socket *Module*

To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax −

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters −

- **socket_family** − This is either AF_UNIX or AF_INET, as explained earlier.
- **socket_type** − This is either SOCK_STREAM or SOCK_DGRAM.
- **protocol** − This is usually left out, defaulting to 0.

Once you have *socket* object, then you can use required functions to create your client or server program. Following is the list of functions required −

Server Socket Methods

Sr.No.	Method & Description
1	**s.bind()** This method binds address (hostname, port number pair) to socket.
2	**s.listen()** This method sets up and start TCP listener.
3	**s.accept()** This passively accept TCP client connection, waiting until connection arrives (blocking).

Client Socket Methods

Sr.No.	Method & Description
1	**s.connect()-** This method actively initiates TCP server connection.

General Socket Methods

Sr.No.	Method & Description
1	**s.recv()** This method receives TCP message
2	**s.send()** This method transmits TCP message

3	**s.recvfrom()** This method receives UDP message
4	**s.sendto()** This method transmits UDP message
5	**s.close()** This method closes socket
6	**socket.gethostname()** Returns the hostname.

A Simple Server

To write Internet servers, we use the **socket** function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

Now call **bind(hostname, port)** function to specify a *port* for your service on the given host.Next, call the *accept* method of the returned object. This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

```
#!/usr/bin/python          # This is server.py file

import socket              # Import socket module

s = socket.socket()        # Create a socket object
```

```
host = socket.gethostname() # Get local machine name
port = 12345               # Reserve a port for your service.
s.bind((host, port))       # Bind to the port

s.listen(5)                # Now wait for client connection.
while True:
   c, addr = s.accept()    # Establish connection with client.
   print 'Got connection from', addr
   c.send('Thank you for connecting')
   c.close()               # Close the connection
```

A Simple Client

Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's socket module function.

The socket.connect (hosname, port) opens a TCP connection to hostname on the port. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits −

```
#!/usr/bin/python          # This is client.py file

import socket              # Import socket module

s = socket.socket()        # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345               # Reserve a port for your service.

s.connect((host, port))
print s.recv(1024)
s.close()                  # Close the socket when done
```

Now run this server.py in background and then run above client.py to see the result.

```
# Following would start a server in background.
$ python server.py &

# Once server is started run client as follows:
$ python client.py
```

```
Got connection from ('127.0.0.1', 48437)
Thank you for connecting
```

This would produce following result

Output :

Server.py file

```
import socket
s=socket.socket()
print('socket is created' )
s.bind(('localhost',9999))
s.listen(3)
print('waiting for connections')
while True:

    c,addr=s.accept()
    print("connected with",addr)
    c.send(bytes('welcome to AISSMS IOIT','Utf-8'))
    c.close()
```

```
# This is server.py file
import socket                          # Import socket module
s=socket.socket()                      #  Create  a  socket
object print('socket is created' )
s.bind(('localhost',9999))             # Bind to the port on the given host
s.listen(3)                            #  Now  wait  for  client
connection. print('waiting for connections')
while True:
  c,addr=s.accept()                    # Establish connection with client.
  print("connected with",addr)
  c.send(bytes('welcome to AISSMS IOIT','Utf-8'))  # Transmits TCP message to receiver
  c.close()                            # Close the connection
```

client.py

```
import socket
c =socket.socket()
c.connect(('localhost',9999))
print(c.recv(1024).decode())
```

```
# This is client.py file

import socket                  # Import socket module
c =socket.socket()             #  Create  a  socket  object
c.connect(('localhost',9999))  #  Initiates  TCP  server
connection. print(c.recv(1024).decode())
```

Now run this server.py in background and then run above client.py to see the result.

Output of server.py

C:\Users\sandh\pythonProject\venv\Scripts\python.exe
C:/Users/sandh/pythonProject/server.py socket is created
waiting for connections
connected with ('127.0.0.1', 63138)

Output of Client.py

C:\Users\sandh\pythonProject\venv\Scripts\python.exe
C:/Users/sandh/pythonProject/client.py welcome to AISSMS IOIT

Process finished with exit code 0.

Conclusion:

Experiment NO .7

TITLE-Congestion control using Leaky Bucket algorithm

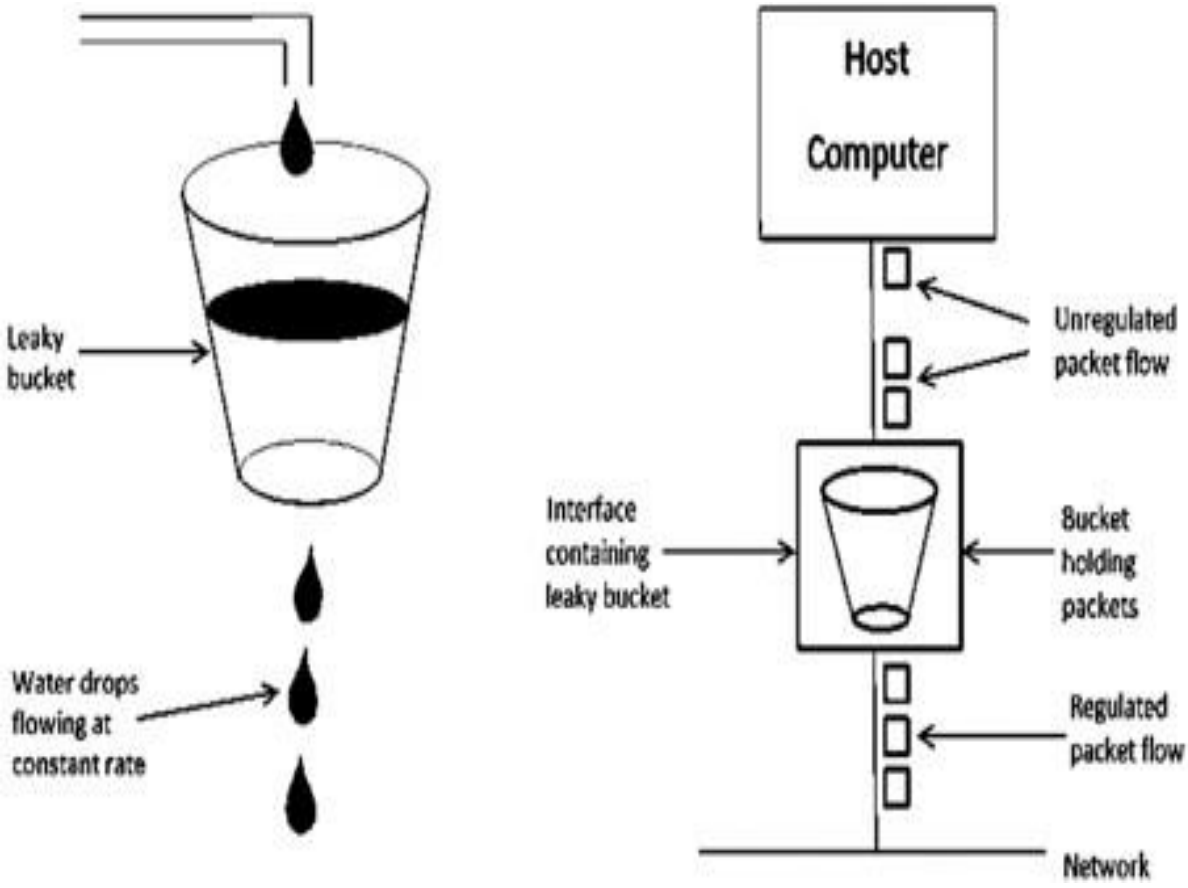
AIM-To write a program for congestion control using Leaky Bucket algorithm

SOFTWARE USED-

THEORY-

The congesting control algorithms are basically divided into two groups: open loop and closed loop. Open loop solutions attempt to solve the problem by good design, in essence, to make sure it does not occur in the first place. Once the system is up and running, midcourse corrections are not made. Open loop algorithms are further divided into ones that act at source versus ones that act at the destination. In contrast, closed loop solutions are based on the concept of a feedback loop if there is any congestion. Closed loop algorithms are also divided into two sub categories: explicit feedback and implicit feedback. In explicit feedback algorithms, packets are sent back from the point of congestion to warn the source. In implicit algorithm, the source deduces the existence of congestion by making local observation, such as the time needed for acknowledgment to come back. The presence of congestion means that the load is (temporarily) greater than the resources (in part of the system) can handle. For subnets that use virtual circuits internally, these methods can be used at the network layer. Another open loop method to help manage congestion is forcing the packet to be transmitted at a more predictable rate. This approach to congestion management is widely used in ATM networks and is called traffic shaping.

The other method is the leaky bucket algorithm. Each host is connected to the network by an interface containing a leaky bucket, that is, a finite internal queue. If a packet arrives at the queue when it is full, the packet is discarded. In other words, if one or more process are already queued, the new packet is unceremoniously discarded. This arrangement can be built into the hardware interface or simulate d by the host operating system. In fact it is nothing other than a single server queuing system with constant service time. The host is allowed to put one packet per clock tick onto the network. This mechanism turns an uneven flow of packet from the user process inside the host into an even flow of packet onto the network, smoothing out bursts and greatly reducing the chances of congestion.



Algorithm

1. Start
2. Set the bucket size or the buffer size,
3. Set the output rate.
4. Transmit the packets such that there is no overflow.
5. Repeat the process of transmission until all packets are transmitted. (Reject packets where its size is greater than the bucket size)
6. Stop

Code

main.c

Run

```c
#include<stdio.h>
int main(){
int incoming, outgoing, buck_size, n, store=0;
printf("Enter bucket size, outgoing rate and no of inputs: ");
scanf("%d %d %d", &buck_size, &outgoing, &n);
while (n!=0){
printf("Enter the incoming packet size: ");
scanf("%d", &incoming);
printf("Incoming packet size %d\n", incoming);
if (incoming <= (buck_size-store)){
store += incoming;
printf("bucket buffer size %d out of %d\n",store,buck_size);
}else{
printf("dropped %d no of packet\n",incoming-(buck_size-store));
printf("Bucket buffer size %d out of %d\n");
store=buck_size;
}
store=store-outgoing;
printf("after outgoing %d packet left out of %d in buffer \n", store, buck_size);
n--;
}
}
```

Output

Output

Clear

```
/tmp/kpEsF6NAF1.o
Enter bucket size, outgoing rate and no of inputs: 32 10 3
Enter the incoming packet size: 12
Incoming packet size 12
bucket buffer size 12 out of 32
after outgoing 2 packet left out of 32 in buffer
Enter the incoming packet size: 4
Incoming packet size 4
bucket buffer size 6 out of 32
after outgoing -4 packet left out of 32 in buffer
Enter the incoming packet size: 14
Incoming packet size 14
bucket buffer size 10 out of 32
after outgoing 0 packet left out of 32 in buffer
_
```

Conclusion:

Experiment NO: 8

TITLE-Analyzing working of protocols

AIM-To observe and note the working of protocols using PING / TRACEROUTE / PATHPING and capture packets in LAN using packet capture and analysis tools.

THEORY-

Traceroute, Ping, and PathPing are network tools or utilities that use the ICMP protocol to perform testing to diagnose issues on a network. Internet Control Message Protocol (ICMP) is an error reporting and diagnostic utility. ICMPs are used by routers, intermediary devices, or hosts to communicate updates or error information to other routers, intermediary devices, or hosts.

1. *Ping:*

The ping command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not. Tracking and isolating hardware and software problems, determining the status of the network and various foreign hosts. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response.

2. *Trace route:*

Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop. Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination. Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values.
The response time of each hop is calculated. To guarantee accuracy, each hop is

queried multiple times (usually three times) to better measure the response of that particular hop. Traceroute sends packets with TTL values that gradually increase from packet to packet. starting with TTL value of one. Routers decrement TTL values of packets by one when routing and discard packets whose TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded. For the first set of packets, the first router receives the packet, decrements the TTL value and drops the packet because it then has TTL value zero. The router sends an ICMP Time Exceeded message back to the source. The next set of packets are given a TTL value of two, so the first router forwards the packets, but the second router drops them and replies with ICMP Time Exceeded. Proceeding in this way, traceroute uses the returned ICMP Time Exceeded messages to build a list of routers that packets traverse, until the destination is reached and returns an ICMP Echo Reply message. With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the hostname.

3. *Pathping:*

This network utility is a more advanced version of the Ping tool, which performs a ping to cach hop along the route to the destination (unlike Ping, which just pings from the originating device to the destination device). It is extremely useful in diagnosing packet loss, and can help with diagnosing slow speed faults.

To Pathping a device, proceed as follows.

1. Open a Windows Command Prompt window.

2. At the command prompt, type, pathping <IP address/Website url>

4. ICMP:

ICMP or Internet Control Message Protocol is Internet or Network layer protocol; it is used to check the reachability of a host or router in a network.

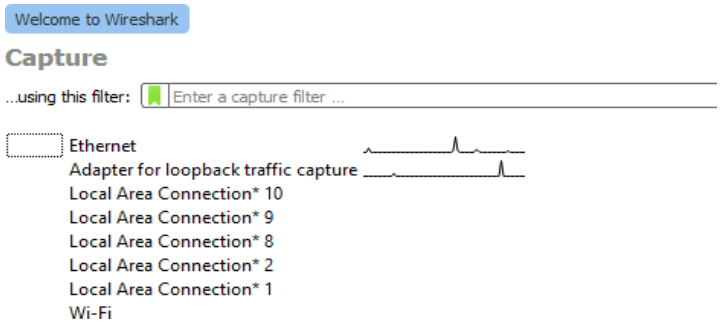
PROCEDURE:

1. Set up the LAN:

   •Connect multiple devices (e.g., computers, laptops) in a LAN network. Ensure that all devices are connected to the same LAN and can communicate with each other.

2. Launch Wireshark:

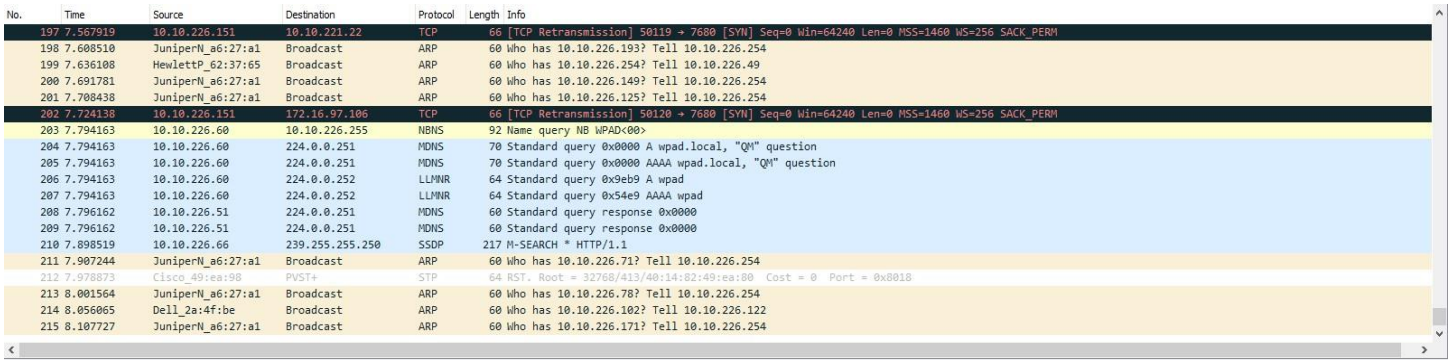
   •Install Wireshark on a computer within the LAN network.



   •Launch Wireshark and select the network interface that is connected to the LAN.

3. Start Packet Capture:

   •Click on the "Capture" button (the one with a shark fin icon) in Wireshark to start capturing packets on the selected interface.



No.	Time	Source	Destination	Protocol	Length	Info
197	7.567919	10.10.226.151	10.10.221.22	TCP	66	[TCP Retransmission] 50119 → 7680 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
198	7.608510	JuniperN_a6:27:a1	Broadcast	ARP	60	Who has 10.10.226.193? Tell 10.10.226.254
199	7.636108	HewlettP_62:37:65	Broadcast	ARP	60	Who has 10.10.226.254? Tell 10.10.226.49
200	7.691781	JuniperN_a6:27:a1	Broadcast	ARP	60	Who has 10.10.226.149? Tell 10.10.226.254
201	7.708438	JuniperN_a6:27:a1	Broadcast	ARP	60	Who has 10.10.226.125? Tell 10.10.226.254
202	7.724138	10.10.226.151	172.16.97.106	TCP	66	[TCP Retransmission] 50120 → 7680 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
203	7.794163	10.10.226.60	10.10.226.255	NBNS	92	Name query NB WPAD<00>
204	7.794163	10.10.226.60	224.0.0.251	MDNS	70	Standard query 0x0000 A wpad.local, "QM" question
205	7.794163	10.10.226.60	224.0.0.251	MDNS	70	Standard query 0x0000 AAAA wpad.local, "QM" question
206	7.794163	10.10.226.60	224.0.0.252	LLMNR	64	Standard query 0x9eb9 A wpad
207	7.794163	10.10.226.60	224.0.0.252	LLMNR	64	Standard query 0x54e9 AAAA wpad
208	7.796162	10.10.226.51	224.0.0.251	MDNS	60	Standard query response 0x0000
209	7.796162	10.10.226.51	224.0.0.251	MDNS	60	Standard query response 0x0000
210	7.898519	10.10.226.66	239.255.255.250	SSDP	217	M-SEARCH * HTTP/1.1
211	7.907244	JuniperN_a6:27:a1	Broadcast	ARP	60	Who has 10.10.226.71? Tell 10.10.226.254
212	7.978873	Cisco_49:ea:98	PVST+	STP	64	RST. Root = 32768/413/40:14:82:49:ea:80 Cost = 0 Port = 0x8018
213	8.001564	JuniperN_a6:27:a1	Broadcast	ARP	60	Who has 10.10.226.78? Tell 10.10.226.254
214	8.056065	Dell_2a:4f:be	Broadcast	ARP	60	Who has 10.10.226.102? Tell 10.10.226.122
215	8.107727	JuniperN_a6:27:a1	Broadcast	ARP	60	Who has 10.10.226.171? Tell 10.10.226.254

4. Capture Packets during PING:

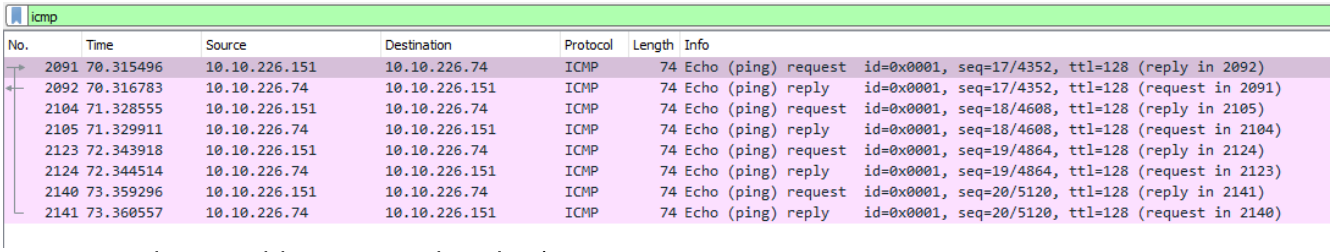
   •Open the command prompt/terminal on a device within the LAN.

   •Use the PING command to send ICMP Echo Request packets to another device in the LAN.

   •For example, on a Windows system, type: ping <IP_address> (replace <IP_address> with the IP address of the target device).

```
Command Prompt
Microsoft Windows [Version 10.0.19045.3570]
(c) Microsoft Corporation. All rights reserved.

C:\Users\admin>ping 10.10.226.74

Pinging 10.10.226.74 with 32 bytes of data:
Reply from 10.10.226.74: bytes=32 time=1ms TTL=128
Reply from 10.10.226.74: bytes=32 time=1ms TTL=128
Reply from 10.10.226.74: bytes=32 time<1ms TTL=128
Reply from 10.10.226.74: bytes=32 time=1ms TTL=128

Ping statistics for 10.10.226.74:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\Users\admin>
```

- While the PING command is running, Wireshark will capture the ICMP packets



exchanged between the devices.

5. Capture Packets during TRACEROUTE:

- Open the command prompt/terminal on a device within the LAN.
- Use the TRACEROUTE command to trace the route to another device in the LAN.
- For example, on a Windows system, type: tracert <IP_address> (replace

```
C:\Users\admin>tracert 10.10.226.74

Tracing route to DESKTOP-P52R3QV [10.10.226.74]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  DESKTOP-P52R3QV [10.10.226.74]

Trace complete.

C:\Users\admin>
```

<IP_address> with the IP address of the target device).

- While the TRACEROUTE command is running, Wireshark will capture the

ICMP packets with increasing TTL values as they traverse through the network.

icmp

No.	Time	Source	Destination	Protocol	Length	Info
349	23.347962	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=138/35328, ttl=1 (no response found!)
350	23.348994	10.10.226.254	10.10.226.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
351	23.349356	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=139/35584, ttl=1 (no response found!)
352	23.353785	10.10.226.254	10.10.226.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
353	23.354108	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=140/35840, ttl=1 (no response found!)
354	23.355022	10.10.226.254	10.10.226.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
365	23.530986	10.10.226.254	10.10.226.151	ICMP	70	Destination unreachable (Port unreachable)
392	25.045861	10.10.226.254	10.10.226.151	ICMP	70	Destination unreachable (Port unreachable)
408	26.558631	10.10.226.254	10.10.226.151	ICMP	70	Destination unreachable (Port unreachable)
448	29.087688	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=141/36096, ttl=2 (no response found!)
449	29.088559	10.13.46.169	10.10.226.151	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
450	29.089987	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=142/36352, ttl=2 (no response found!)
451	29.090801	10.13.46.169	10.10.226.151	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
452	29.091472	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=143/36608, ttl=2 (no response found!)
453	29.092308	10.13.46.169	10.10.226.151	ICMP	134	Time-to-live exceeded (Time to live exceeded in transit)
587	34.805097	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=144/36864, ttl=3 (no response found!)
588	34.817066	10.13.46.218	10.10.226.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
589	34.818187	10.10.226.151	10.4.122.117	ICMP	106	Echo (ping) request id=0x0001, seq=145/37120, ttl=3 (no response found!)
590	34.825892	10.13.46.218	10.10.226.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

1. Capture Packets during PATHPING:

- Open the command prompt/terminal on a device within the LAN.
- Use the PATHPING command to trace the route and measure the round-trip time to another device in the LAN.

```
C:\Users\admin>pathping 10.10.226.74

Tracing route to DESKTOP-P52R3QV [10.10.226.74]
over a maximum of 30 hops:
  0  DESKTOP-MTVU9JE.Unitech.local [10.10.226.151]
  1  DESKTOP-P52R3QV [10.10.226.74]

Computing statistics for 25 seconds...
            Source to Here   This Node/Link
Hop  RTT    Lost/Sent = Pct  Lost/Sent = Pct  Address
  0                                           DESKTOP-MTVU9JE.Unitech.local [10.10.226.151]
                                    0/ 100 =  0%   |
  1    1ms     0/ 100 =  0%     0/ 100 =  0%  DESKTOP-P52R3QV [10.10.226.74]

Trace complete.
```

- For example, on a Windows system, type: pathping <IP_address> (replace <IP_address> with the IP address of the target device).

icmp

No.	Time	Source	Destination	Protocol	Length	Info
161	9.360087	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=37/9472, ttl=1 (reply in 162)
162	9.361737	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=37/9472, ttl=128 (request in 161)
169	9.364819	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=38/9728, ttl=1 (reply in 170)
170	9.366376	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=38/9728, ttl=128 (request in 169)
174	9.619023	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=39/9984, ttl=1 (reply in 175)
175	9.620545	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=39/9984, ttl=128 (request in 174)
180	9.884743	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=40/10240, ttl=1 (reply in 181)
181	9.885866	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=40/10240, ttl=128 (request in 180)
187	10.150136	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=41/10496, ttl=1 (reply in 188)
188	10.151629	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=41/10496, ttl=128 (request in 187)
193	10.415777	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=42/10752, ttl=1 (reply in 194)
194	10.416438	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=42/10752, ttl=128 (request in 193)
200	10.681201	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=43/11008, ttl=1 (reply in 201)
201	10.681715	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=43/11008, ttl=128 (request in 200)
206	10.946815	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=44/11264, ttl=1 (reply in 207)
207	10.947419	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=44/11264, ttl=128 (request in 206)
211	11.212483	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=45/11520, ttl=1 (reply in 212)
212	11.213015	10.10.226.74	10.10.226.151	ICMP	106	Echo (ping) reply id=0x0001, seq=45/11520, ttl=128 (request in 211)
219	11.478080	10.10.226.151	10.10.226.74	ICMP	106	Echo (ping) request id=0x0001, seq=46/11776, ttl=1 (reply in 220)

CONCLUSION:

Experiment NO 9

TITLE: Executing proxy server using simulator

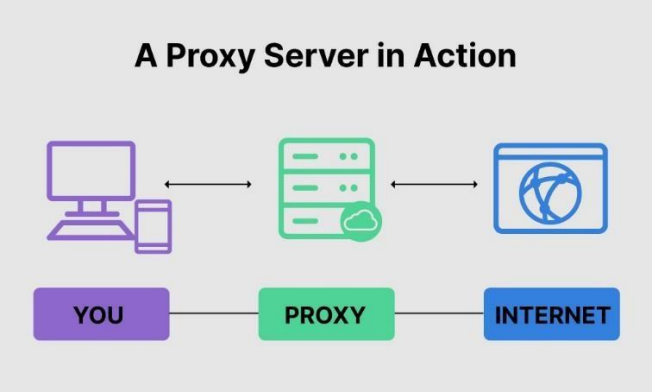
AIM : Designing and simulating proxy server on cisco packet tracer and observing its working

SOFTWARE USED: Cisco packet tracer

THEORY:

Proxy Server Overview:

A proxy server is an intermediary server that plays a critical role in the transmission of data between a client (e.g., a computer, smartphone) and a destination server (e.g., a web server). It functions as a gateway, receiving requests from clients and forwarding those requests to the target server. The target server processes the request and sends a response back to the proxy server, which, in turn, delivers the response to the client.



Purposes of Proxy Servers:

Caching: Proxy servers can store copies of frequently requested resources like web pages, images, and files. By doing so, they reduce the load on the destination server and accelerate content delivery to clients. This is especially useful in networks with limited bandwidth.

Security: Proxy servers enhance network security by serving as a barrier between the client and the internet. They can inspect, filter, or block network traffic, thereby providing an extra layer of protection against malicious content, viruses, and cyber threats.

Anonymity and Privacy: Proxy servers can mask the client's IP address, making it difficult for websites or services to trace a user's identity or location. This feature is valuable for individuals who want to browse the internet anonymously.

Content Filtering: Proxy servers are commonly used in corporate or educational settings to filter and control access to specific websites or types of content. Administrators can implement content policies to restrict or allow access based on organizational rules.

Load Balancing: In the case of reverse proxy servers, they distribute incoming client requests across multiple backend servers. This load balancing improves system reliability and ensures even distribution of workloads.

Types of Proxy Servers:

Forward Proxy: A forward proxy is used by clients to access the internet indirectly. It's often employed in corporate networks to control and monitor outbound traffic and enforce security policies.

Reverse Proxy: Reverse proxies are placed in front of one or more destination servers.

They act as a gateway for incoming client requests and can handle tasks such as load

balancing, SSL termination, and serving as a single entry point to multiple backend servers.

Transparent Proxy: Transparent proxies intercept network traffic without the client's knowledge. They are frequently used for caching and content filtering in network infrastructure.

Anonymous Proxy: These proxies conceal the client's IP address from the destination server, providing a level of anonymity while browsing the web.

Configuration and Operation:

Proxy servers are configured to route and filter traffic based on specific rules and policies. They mainly operate at the application layer (Layer 7) of the OSI model, allowing them to inspect and modify the content of HTTP requests and responses. Clients can be configured to route their traffic through the proxy server.

Advantages and Disadvantages:

Proxy servers offer several advantages, such as improved security, privacy, and network performance. However, they can also introduce complexity, potential single points of failure, and require proper configuration and maintenance.

Understanding the theory of proxy servers is essential for effectively utilizing their capabilities in different network scenarios, from boosting performance to enhancing security and privacy. Network administrators and IT professionals typically configure and manage proxy servers to meet specific network requirements.

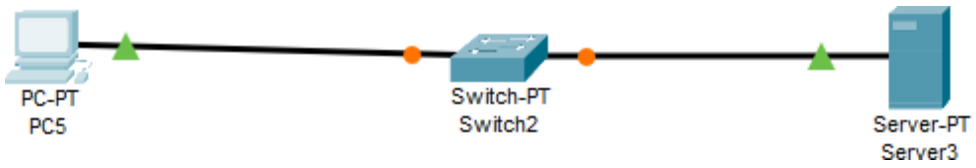
PROCEDURE:

In Packet Tracer, a network simulation tool developed by Cisco, you can create a simple web server using a generic server device and configure it as a proxy server. Here's a step-by-step guide on how to do it:

1. Open Packet Tracer: If you don't have Packet Tracer installed, you can download it from the Cisco Networking Academy website.

2. Set up the network: Create a network topology by dragging and dropping devices from the device list onto the workspace. For this example, you will need at least three devices: a PC, a server (generic server), and a switch to connect them.

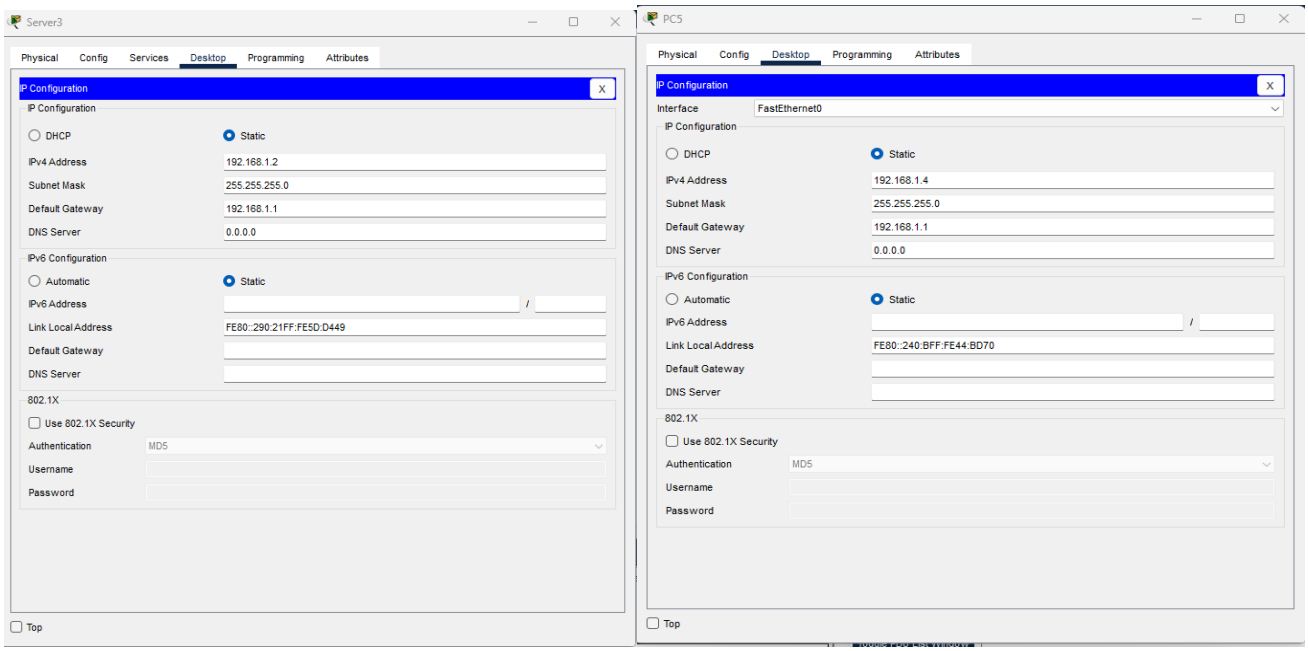
3. Connect the devices: Use the copper straight-through cable to connect the PC to one of the router's LAN ports. Connect the server to another LAN port of the router using the same cable type.



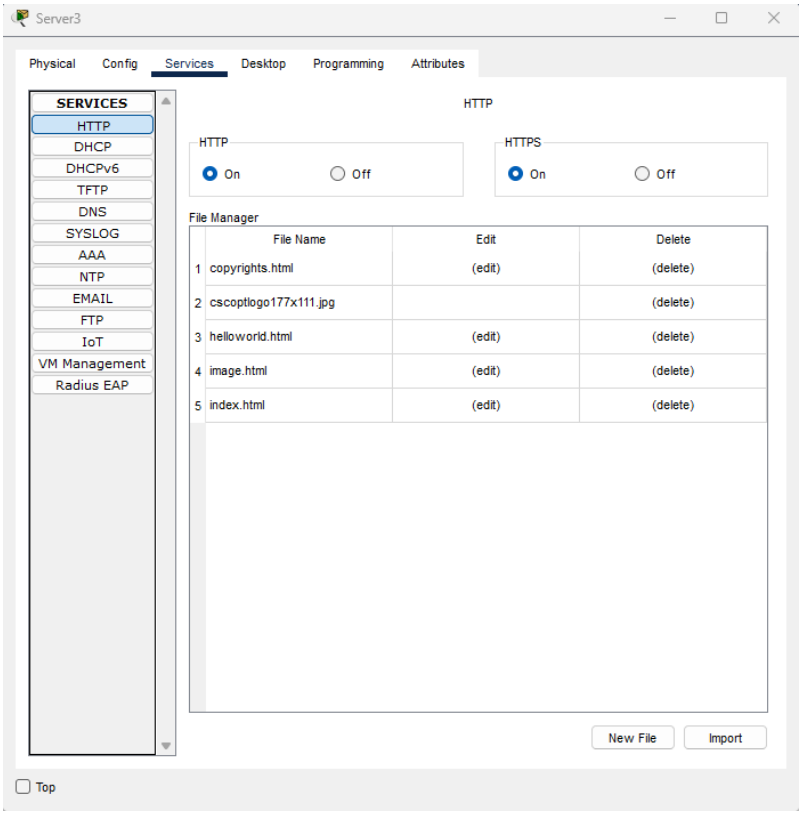
1. Configure IP addresses: Right-click on each device and select "Configure." Set IP addresses on the interfaces connected to the router's LAN ports. For example:

   • PC: IP address 192.168.1.4, subnet mask 255.255.255.0, default gateway 192.168.1.1

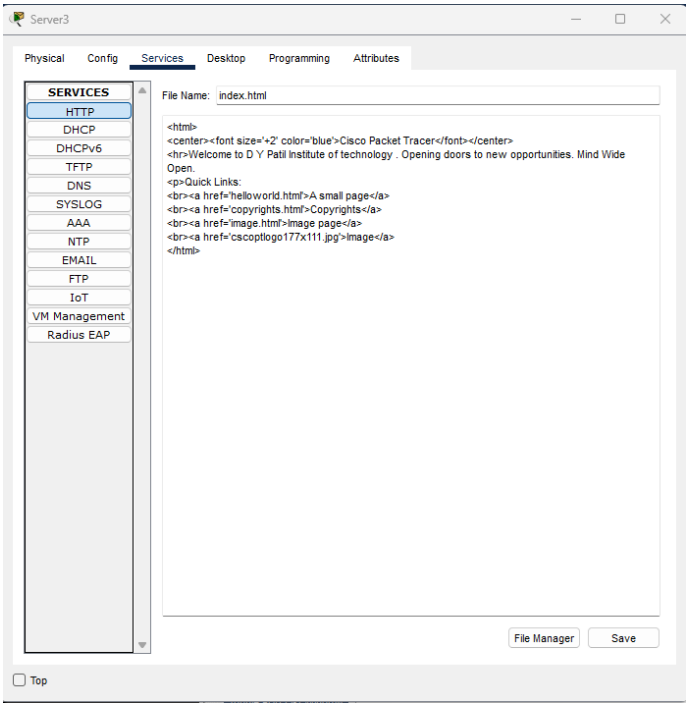
   • Server: IP address 192.168.1.2, subnet mask 255.255.255.0, default gateway 192.168.1.1



2. Set up HTTP service on the server: Double-click on the server to access its configuration. Go to the "Services" tab and enable the HTTP service. This will allow the server to respond to HTTP requests and act as a web server.

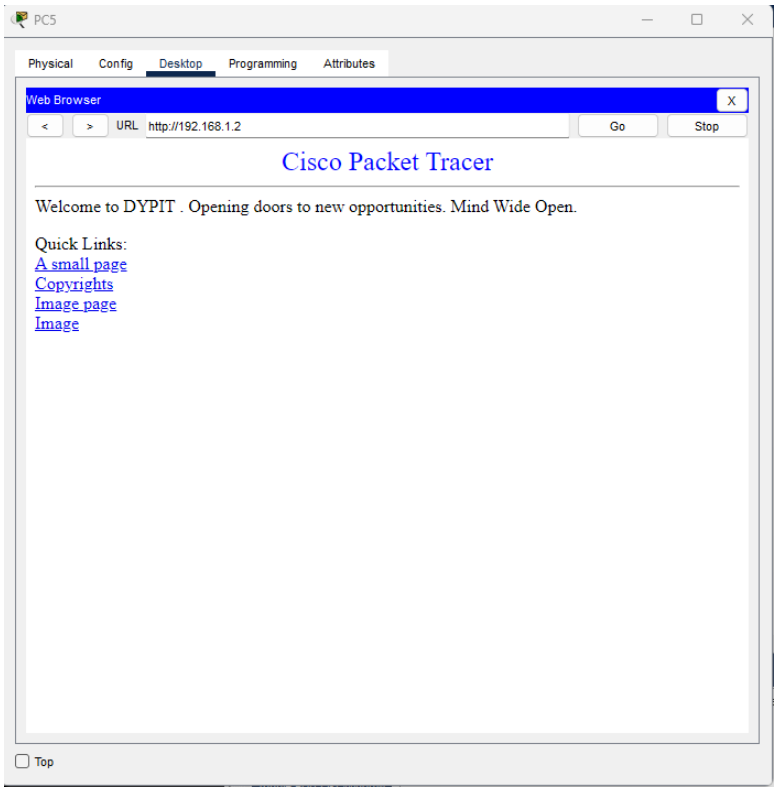


3. You can also configure data on the website by changing scripts of index.html file by clicking on edit option



4. Configure the PC's web browser proxy settings: Open the PC's web browser. Set the HTTP proxy to the IP address of the server (192.168.1.2).

5. Test the setup: Open the web browser on the PC and try to access a website. The PC will send the HTTP requests to the server acting as a proxy. The server will forward the requests to the Internet and return the responses back to the PC through the proxy.


PC5
Physical
Config
Desktop
Programming
Attributes
Web Browser
X
<
>
URL
http://192.168.1.2
Go
Stop
Cisco Packet Tracer
Welcome to DYPIT . Opening doors to new opportunities. Mind Wide Open.
Quick Links:
A small page
Copyrights
Image page
Image
Top

CONCLUSION:

Experiment NO :10

TITLE-Cabling of Network

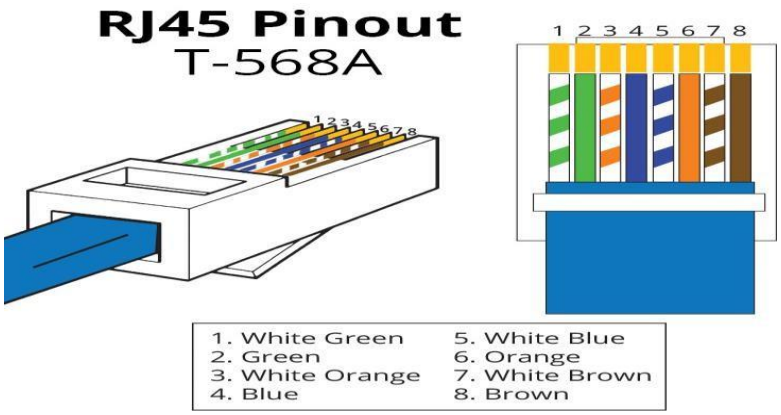
AIM-To do cabling of Network and learn various steps of cabling

1. Cable Crimping
2. Straight Cabling
3. Cross Cabling
4. IO connector crimping
5. Testing crimped cable using a cable tester

HARDWARE- RJ-45 connector, 10 Connector, Crimping Tool, Twisted pair Cable, Cable Tester. Standard Cabling: 1. 10BaseT and 100BaseT, UTP category-5 cable for both modes. A straight cable is used to connect a computer to a hub

RJ45 Pin # (END 1)	Wire Color	Diagram End #1	RJ45 Pin # (END 2)	Wire Color	Diagram End #2
1	White/Orange		1	White/Green	
2	Orange		2	Green	
3	White/Green		3	White/Orange	
4	Blue		4	White/Brown	
5	White/Blue		5	Brown	
6	Green		6	Orange	
7	White/Brown		7	Blue	
8	Brown		8	White/Blue	

Cross Cabling: A cross cable is used to connect 2 computers directly (with ONLY the UTPcable). It is also used to connect 2 hubs with a normal port on both hubs



PROCEDURE:

Cable Crimping:

1. Remove the outmost vinyl shield for 12mm at one end of the cable.

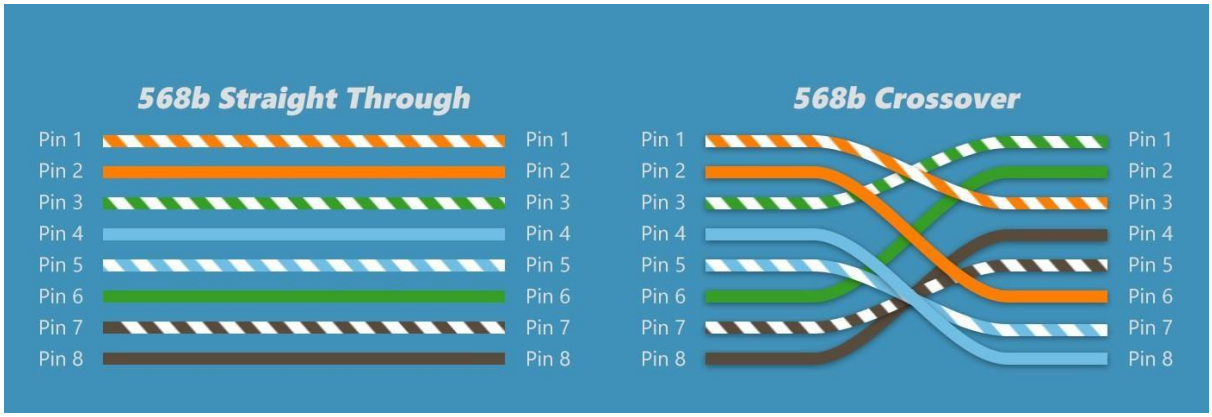
2. Arrange the metal wires in parallel

3. Insert the metal wires into RJ45 connector on keeping the metal wire arrangement.

4. Set the RJ45 connector (with the cable) on the pliers, and squeeze it tightly.

5. Make the other side of the cable in the same way.

6. Once done, no need to take care of the direction of the cable.



I0 connector crimping:

Run the full length of Ethernet cable in place, from endpoint to endpoint, making sure to leave excess. At one end, cut the wire to length leaving enough length to work, but not too much excess. Strip off about 2 inches of the Ethernet cable sheath. Align each of the colored wires according to the layout of the jack. Use the punch down tool to insert each wire into the jack. Repeat the above steps for the second RJ45 jack.

Testing the crimped cable using a cable tester:

1. Skin off the cable jacket 3.0 cm long cable stripper up to cable.
2. Untwist each pair and straighten each wire 190 0.1.5 cm long.
3. Cut all the wires.
4. Insert the wires into the RJ45 connector right white orange left brown the pins facing up.
5. Place the connector into a crimping tool, and squeeze hard so that the handle reaches its full swing.
6. Use a cable tester to test for proper continuity.


3cm
1. Get your RJ45 plug, cable cap, Ethernet cable ready
2. Insert the cable cap
3. Put the Ethernet cable into crimping tool
4. Cut the outer skin
5. Separate the wires
T568B Line order: orange+white | orange | green+white | blue | blue+white | green | brown+white | brown
6. keep the wires neat, T568B is recommended
Reserve a length of about 2.5cm
7. Cut the wires in obluque angle
8. Wear the core wire into the RJ45 plug
push
9. Insert the cable cap into the RJ45 plug
10. Complete as shown
11. Crimp plug
12. Complete

Conclusion: