# 1.DEVELOPING CLIENT SERVER BASED TCP APPLICATION USING LINUX SOCKET PROGRAMMING

## Aim:

The aim of this lab is to develop a basic TCP client-server application using Linux socket programming. This lab will guide you through writing, compiling, and running both server and client codes, establishing a simple communication channel between them.

## Algorithm:

Linux socket programming allows communication between different processes on the same or different systems using Internet Protocol (IP). TCP (Transmission Control Protocol) is a reliable, connection-oriented protocol that ensures data integrity. This lab focuses on creating a basic TCP client-server application.

### Server Side:

In the server-side code:

* A socket is created using the socket() function.
* The server's address is set up using a sockaddr\_in struct.
* The socket is bound to the server's address using the bind() function.
* The server listens for incoming connections using the listen() function.
* It accepts an incoming connection using the accept() function.
* Data is read from the client using the read() function.
* The server then prints the received message and closes the connection.

### Client Side:

In the client-side code:

* A socket is created using the socket() function.
* The server's address is configured using a sockaddr\_in struct.
* The client connects to the server using the connect() function.
* The user inputs a message to be sent to the server.
* The message is sent to the server using the send() function.
* The client then closes the connection.

**PROCEDURE:**

Step 1: Writing the Server Code

1. Create a file named server.c in the project directory.

c

// server.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket, valread;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[MAX\_BUFFER\_SIZE] = {0};

// Create a socket

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up server address struct

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = INADDR\_ANY;

address.sin\_port = htons(PORT);

// Bind the socket to the address

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

exit(EXIT\_FAILURE);

}

// Accept incoming connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept failed");

exit(EXIT\_FAILURE);

}

// Read data from the client using TCP

valread = read(new\_socket, buffer, MAX\_BUFFER\_SIZE);

printf("Received message from client: %s\n", buffer);

// Close the connection

close(new\_socket);

close(server\_fd);

return 0;

}

Step 2: Compiling and Running the Server Code

1. Compile the server code.

bash

gcc server.c -o server

1. Run the server.

bash

./server

Part 2: Client Side Step 1: Writing the Client Code

1. Create a file named client.c in the project directory.

c

// client.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_address;

char message[MAX\_BUFFER\_SIZE];

// Create a socket

if ((client\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Configure server address

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_address.sin\_addr) <= 0) {

perror("Invalid address/Address not supported");

exit(EXIT\_FAILURE);

}

// Connect to the server using TCP

if (connect(client\_fd, (struct sockaddr \*)&server\_address, sizeof(server\_address)) < 0) {

perror("Connection Failed");

exit(EXIT\_FAILURE);

}

// Get user input for the message

printf("Enter a message to send to the server: ");

fgets(message, MAX\_BUFFER\_SIZE, stdin);

// Send the message to the server using TCP

send(client\_fd, message, strlen(message), 0);

// Close the connection

close(client\_fd);

return 0;

}

Step 2: Compiling and Running the Client Code

1. Compile the client code.

bash

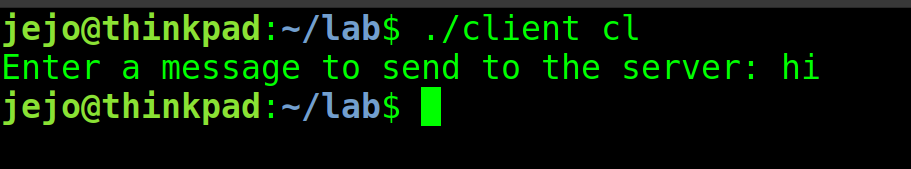
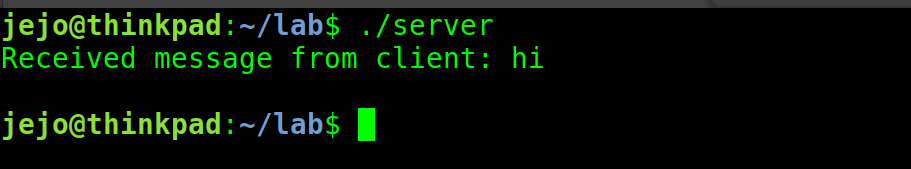
gcc client.c -o client

1. Run the client.

bash

./client

OUTPUT:



# 2.Developing a UDP Client-Server Application using UNIX Socket Programming

## Aim:

The aim of this lab is to develop a basic UDP (User Datagram Protocol) client-server application using UNIX socket programming. This lab will guide you through writing, compiling, and running both server and client codes for establishing communication over UDP.

## Algorithm:

### Part 1: Server Side

**Step 1: Writing the Server Code**

1. Create a file named udp\_server.c in the project directory.
   * Copy and paste the following code into udp\_server.c:

c

// udp\_server.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int server\_fd;

struct sockaddr\_in server\_address, client\_address;

socklen\_t client\_address\_len = sizeof(client\_address);

char buffer[MAX\_BUFFER\_SIZE] = {0};

// Create a UDP socket

if ((server\_fd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Configure server address

server\_address.sin\_family = AF\_INET;

server\_address.sin\_addr.s\_addr = INADDR\_ANY;

server\_address.sin\_port = htons(PORT);

// Bind the socket to the address

if (bind(server\_fd, (struct sockaddr \*)&server\_address, sizeof(server\_address)) == -1) {

perror("Bind failed");

exit(EXIT\_FAILURE);

}

// Server logic here (you can modify this part)

return 0;

}

**Step 2: Compiling and Running the Server Code**

1. Open a terminal.
2. Compile the server code:

bash

gcc udp\_server.c -o udp\_server

1. Run the server:

bash

./udp\_server

### Part 2: Client Side

**Step 1: Writing the Client Code**

1. Create a file named udp\_client.c in the project directory.
   * Copy and paste the following code into udp\_client.c:

c

// udp\_client.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <arpa/inet.h>

#define PORT 8080

#define MAX\_BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_address;

char message[MAX\_BUFFER\_SIZE];

// Create a UDP socket

if ((client\_fd = socket(AF\_INET, SOCK\_DGRAM, 0)) == -1) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Configure server address

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, "127.0.0.1", &server\_address.sin\_addr) <= 0) {

perror("Invalid address/ Address not supported");

exit(EXIT\_FAILURE);

}

// Client logic here (you can modify this part)

return 0;

}

**Step 2: Compiling and Running the Client Code**

1. Open a terminal.
2. Compile the client code:

bash

gcc udp\_client.c -o udp\_client

1. Run the client:

bash

./udp\_client

**OUTPUT:**

## 

# 4.Performance Analysis of TCP and UDP Protocols using Cisco Packet Tracer

## Aim:

The aim of this experiment is to evaluate and compare the performance of TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) in a network environment using Cisco Packet Tracer. The analysis will focus on key performance metrics such as latency, throughput, and reliability under different network conditions.

## Theory:

### TCP (Transmission Control Protocol):

* TCP is a connection-oriented protocol providing reliable and ordered data delivery.
* It employs a three-way handshake for establishing connections and utilizes flow control mechanisms to prevent congestion.
* TCP ensures data integrity through error checking and retransmission of lost or corrupted packets.

### UDP (User Datagram Protocol):

* UDP is a connectionless protocol offering faster but less reliable communication compared to TCP.
* It does not establish a connection before transmitting data and lacks built-in mechanisms for error recovery and flow control.
* UDP is commonly used in scenarios prioritizing low latency and quick data transmission over reliability.

## Procedures:

### 1. Setting Up the Network:

* Create a network topology in Cisco Packet Tracer with at least two devices connected through a network.
* Configure IP addresses and subnet masks for the devices.

### 2. Installing Packet Capturing Software:

* Choose a device (e.g., a PC) as a packet capturing tool.
* Install packet capturing software (e.g., Wireshark) on the capturing device.

### 3. Generating Traffic:

* Configure applications on source and destination devices to generate traffic using both TCP and UDP protocols.
* Define scenarios with varying network loads or introduce network congestion.

### 4. Capturing Packets:

* Start the packet capturing software on the designated device.
* Capture packets during data transmission between source and destination devices.

### 5. Analyzing Performance Metrics:

* Use captured packets to analyze key performance metrics:
  + Latency: Measure round-trip time for packets to travel from source to destination and back.
  + Throughput: Calculate data transmitted per unit of time for both TCP and UDP.
  + Reliability: Examine packet loss and retransmission rates for TCP.

### 6. Comparative Analysis:

* Compare TCP and UDP performance under different network conditions.
* Assess the impact of factors such as network congestion or varying packet sizes.

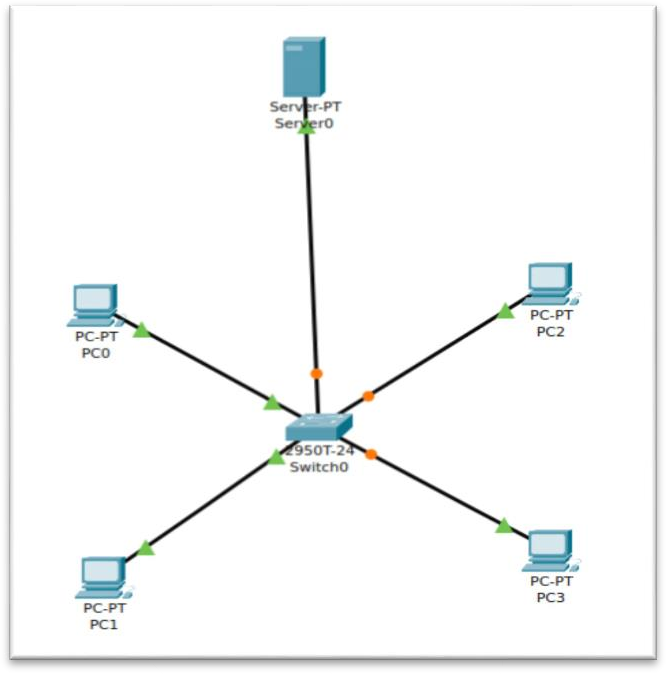
### 7. Documentation and Reporting:

* Compile findings into a report with graphs and charts illustrating performance differences between TCP and UDP.
* Provide insights into the suitability of each protocol for different use cases.

## Lab Procedure:

STEP 1: Open Cisco Packet Tracer and Arrange the components.

STEP 2 : Use "CopperStraightThrough" cables to connect PCs, servers, and switches among other devices.



STEP 3 : Configure every device in this manner: Configure PCs (PC0, PC1, PC2, PC3)

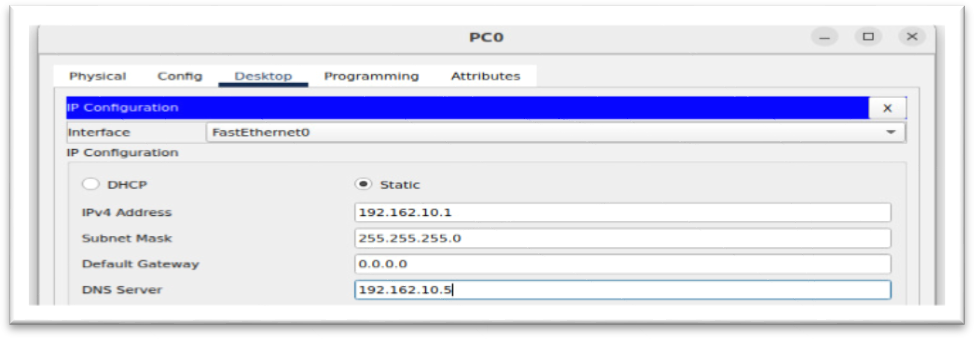
* 1. Configure the PC0 by setting the ipv4:192.162.10.x
  2. where [x+1] is taken from pc(x) number.
  3. So in the case of PC0 we use : 192.162.10.1 SO IN:

PC(1):192.162.10.2

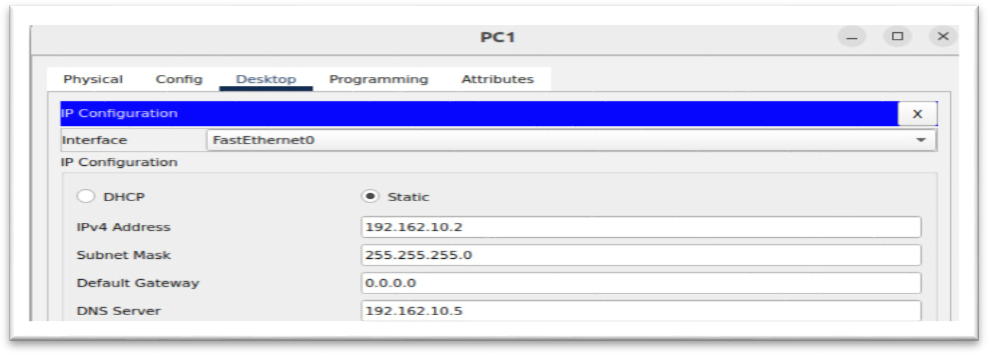
PC(2):192.162.10.3

PC(3):192.162.10.4

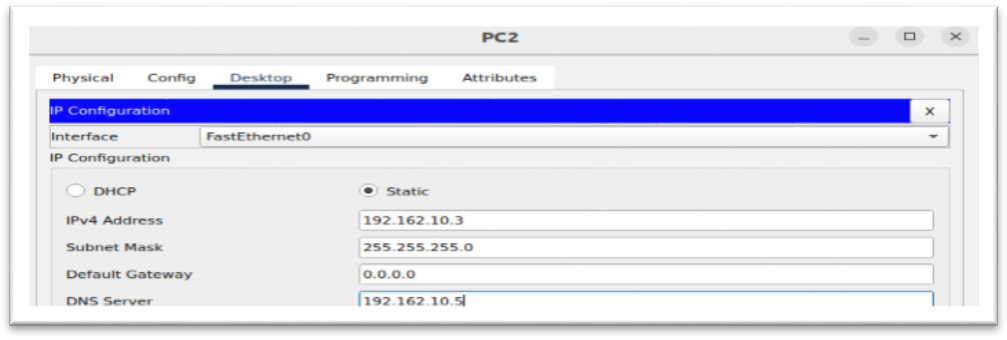
\*CONFIGURE THE DNS WITH THIS COMMON NETOWRK:192.162.10.5



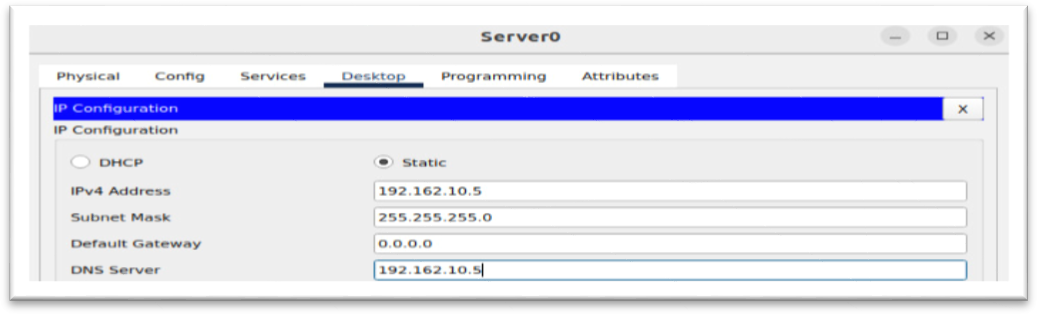
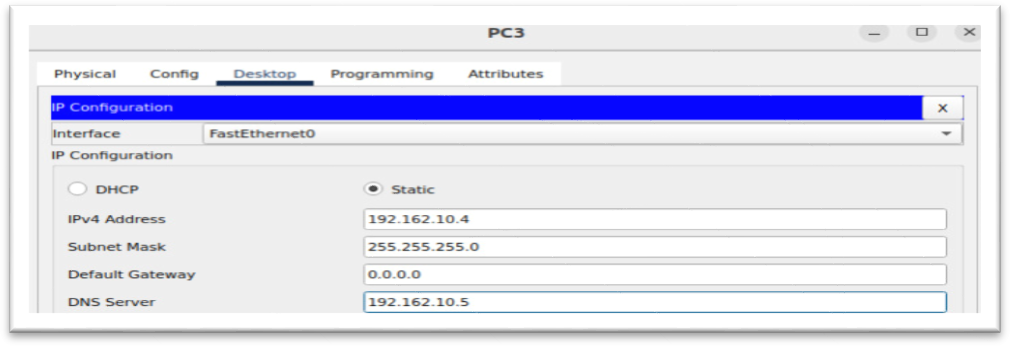
* Configure the PC1.



* + Configure the PC2.



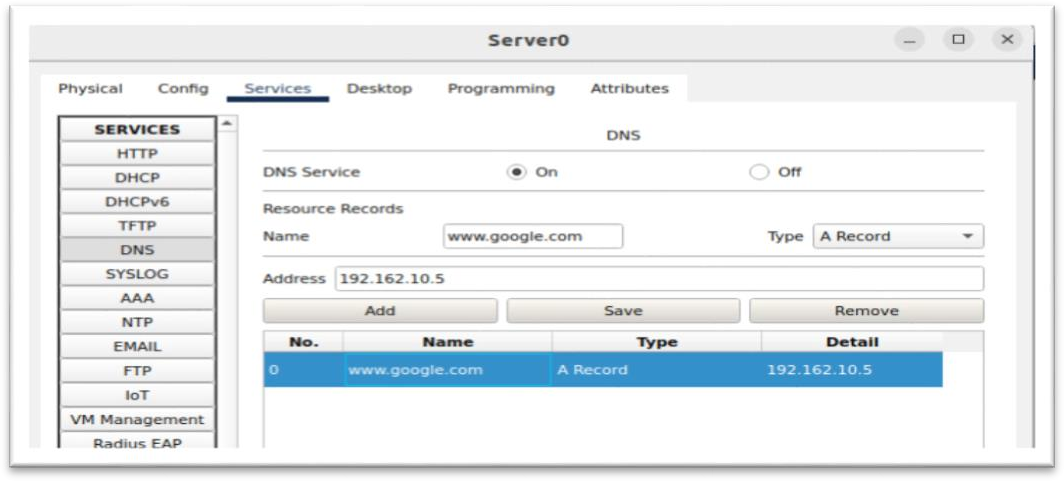
1. Configure the PC3.

  
2.Configure the Server.

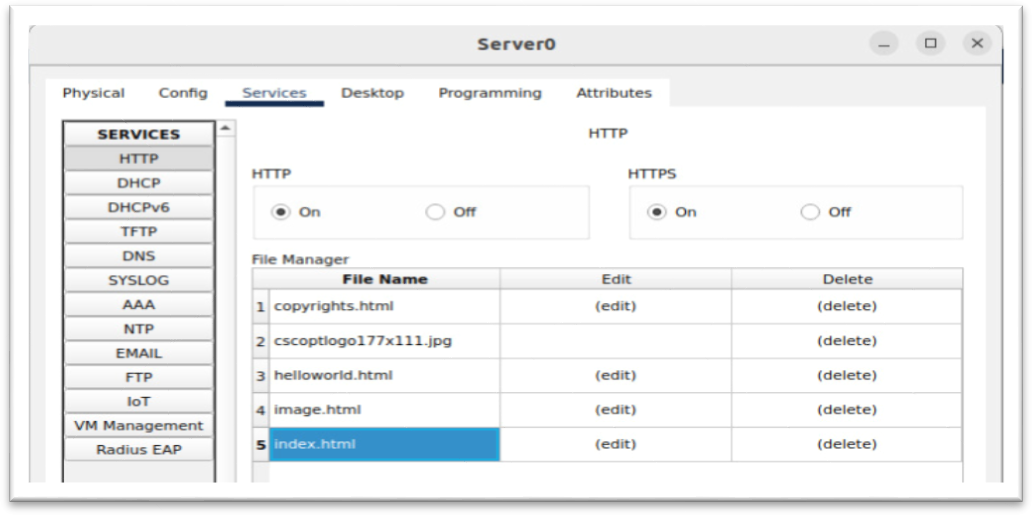
STEP 4 : Configure the Server-Services-DNS

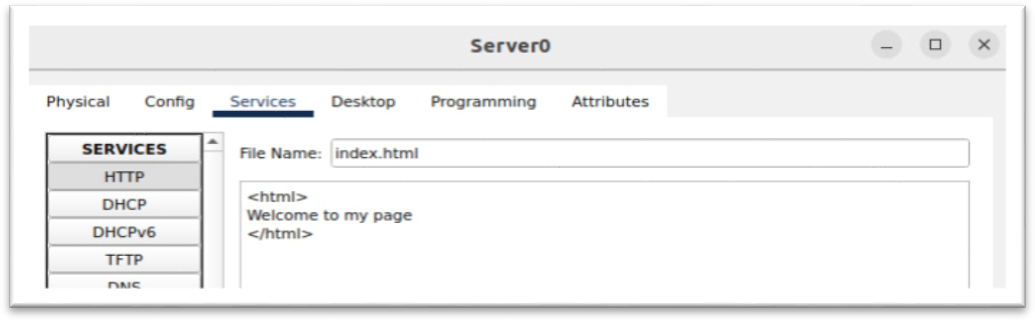
Add Name of the web : [www.google.com](http://www.google.com/)

Add IP Address : 192.162.10.5 , DNS Service want to turn ON.



STEP 5 : Configure the Server-Services-HTTP

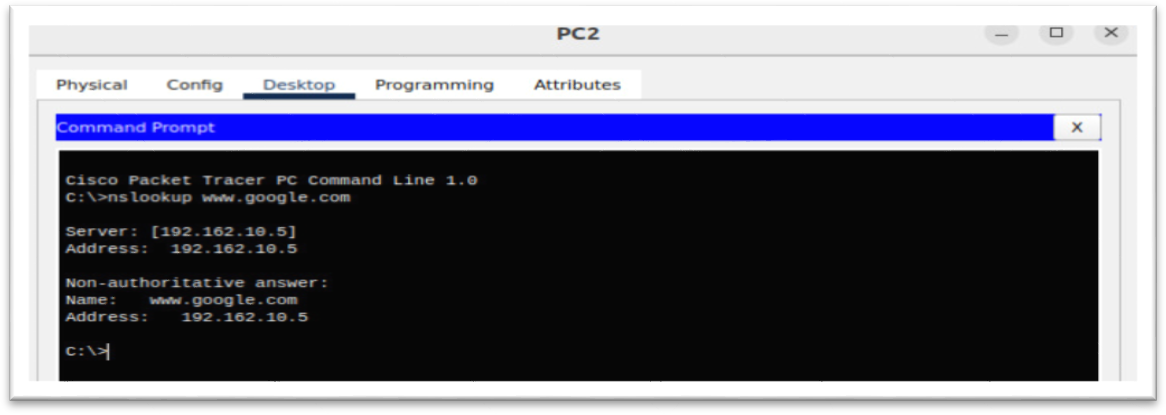
Get into index.html and edit the file, HTTP and HTTPS turn on.



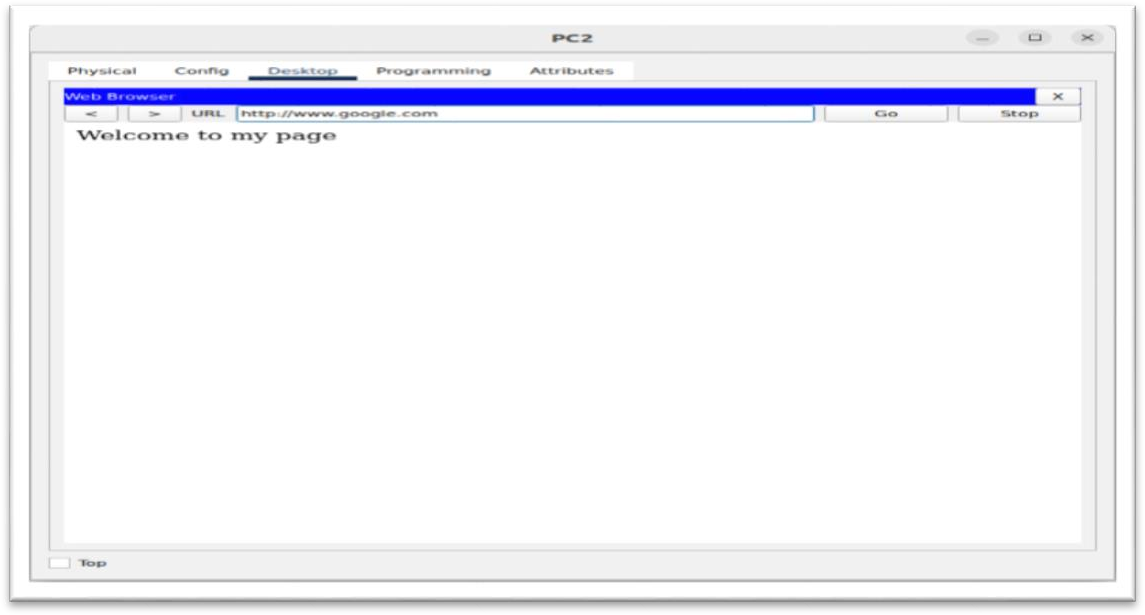
**Add your Data :** Welcome to my page.

STEP 6 : On PC2, open the Command Prompt.

**Comment:** #ns lookup [“www.google.com“](http://www.google.com/) using - (UDP)



STEP 7 : Launch the PC2 web browser. Go to the URL and do a search.(TCP-CONNECTION)



# 5(a).Lab Manual: Performance Analysis of Routing Protocol using Cisco Packet Tracer (OSPF - Dynamic Routing)

## Aim:

The aim of this experiment is to conduct a performance analysis of the OSPF (Open Shortest Path First) routing protocol using Cisco Packet Tracer. The analysis will focus on evaluating OSPF's efficiency in terms of routing table convergence time, network stability, and scalability under varying network conditions.

## Theory:

### OSPF (Open Shortest Path First):

* OSPF is a link-state routing protocol that uses Dijkstra's algorithm to determine the shortest path to reach a destination.
* Designed to provide efficient and scalable routing in large networks by maintaining a detailed link-state database.
* OSPF routers exchange link-state advertisements (LSAs) to build and update their routing tables, allowing for dynamic adaptation to network topology changes.
* OSPF uses metrics such as cost to determine the best path to a destination.

## Procedures:

### 1. Network Topology Setup:

* Create a network topology in Cisco Packet Tracer with multiple routers, switches, and end devices.
* Ensure that OSPF is the chosen routing protocol for the routers.

### 2. Router Configuration:

* Configure OSPF on each router in the network. Specify OSPF areas, router IDs, and network interfaces participating in OSPF.
* Adjust OSPF parameters such as hello and dead intervals, area types, and authentication if needed.

### 3. Traffic Generation:

* Set up applications on end devices to generate traffic across the network.
* Create scenarios that simulate changes in network topology, link failures, or additions of new routers.

### 4. Performance Metrics Collection:

* Use Cisco Packet Tracer's simulation mode to observe OSPF's behavior during network events.
* Record routing table convergence times, including the time it takes for routers to update their routing tables in response to topology changes.

### 5. Link-State Database Analysis:

* Examine the OSPF link-state database on routers to understand how LSAs are exchanged and propagated.
* Evaluate the impact of changes in the network on the link-state database.

### 6. Stability and Scalability Testing:

* Introduce network changes, such as link failures or additions, to assess OSPF's ability to maintain stable routing and adapt to topology changes.
* Test the scalability of OSPF by increasing the size of the network and observing its performance.

### 7. Documentation and Analysis:

* Document the observed convergence times, stability, and scalability of OSPF in the network.
* Analyze the impact of OSPF configurations and parameters on its performance.
* Provide recommendations for optimizing OSPF in different network scenarios.

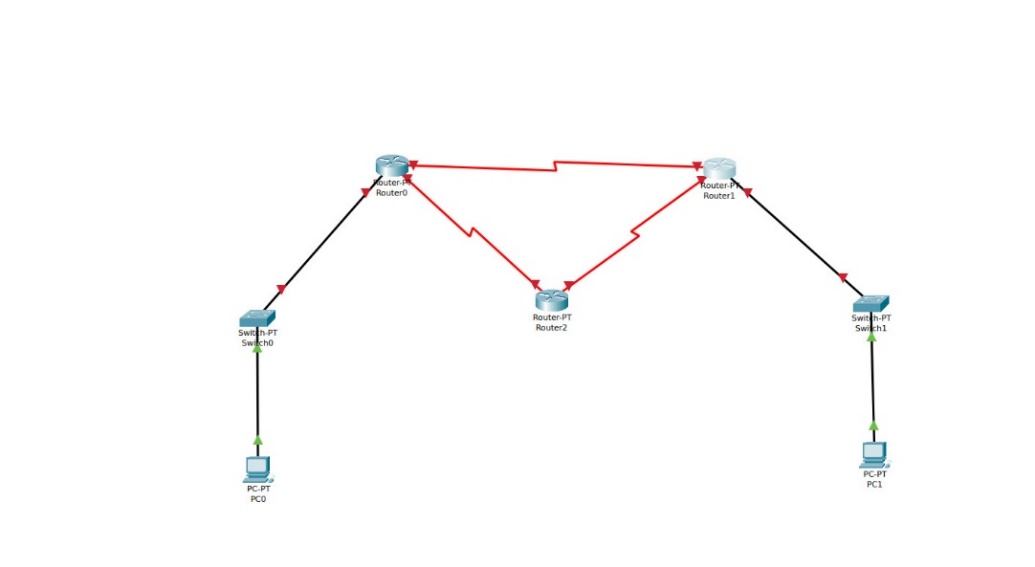
### 8. Reporting:

* Compile the findings into a report, including graphs and charts to illustrate OSPF's performance under various conditions.
* Summarize the strengths and weaknesses of OSPF based on the conducted analysis.

By following these procedures, you can gain insights into OSPF's performance characteristics and its behavior in response to changes in network conditions using Cisco Packet Tracer. This analysis is crucial for network administrators to optimize OSPF configurations for efficient and reliable routing in real-world scenarios.

**Lab Procedure:**

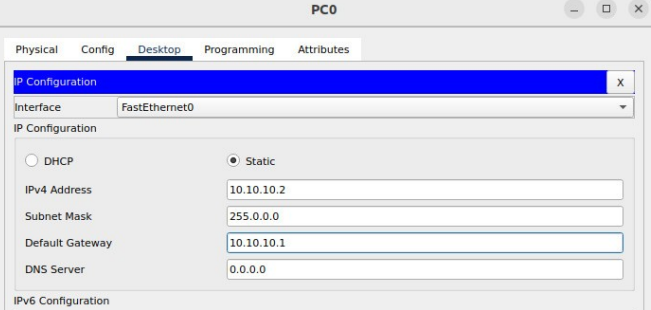
Step 1: Open Cisco Packet Tracer and Arrange the components.



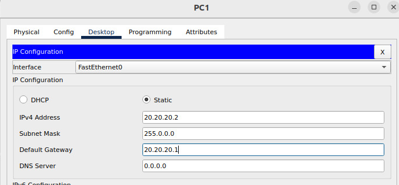
Step 2 : Use "Automatically Choose Connection Type " cables to connect PCs, servers, and router among other devices.

Step 3 : Configure every device in this manner: Configure PCs (PC0, PC1, PC2, PC3)

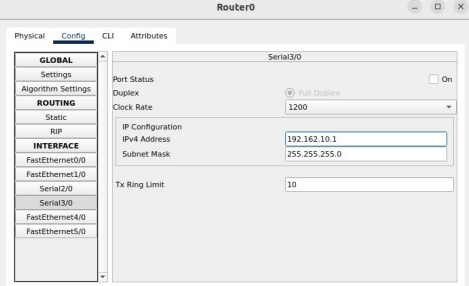
* Configure the PC0



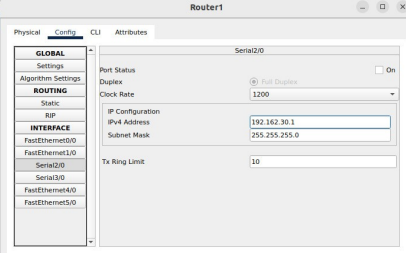
* Configure the PC1



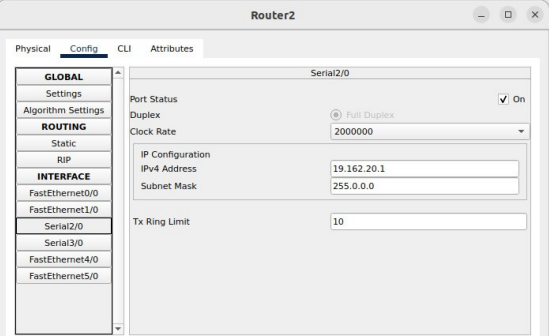
Step 3 : Establishing the connections in router 0

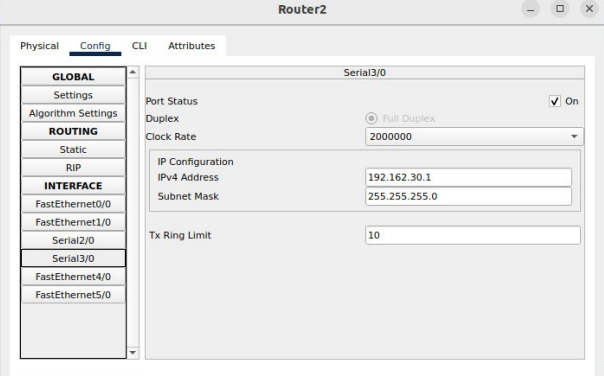


Step 4 : Establishing the connections in router 1

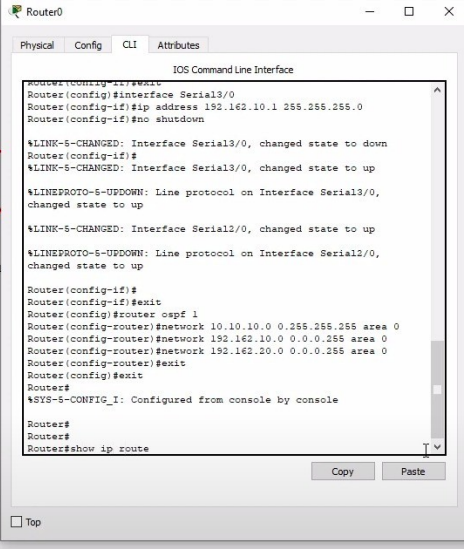


Step 5 : Establishing the connections in router 2





Step 6 : Establishing the connections in router 0



Comment to be follow:

# exit

# router ospf 1

# network 10.10.10.0 0.255.255.255 area 0

# network 192.162.10.0 0.0.0.255 area 0

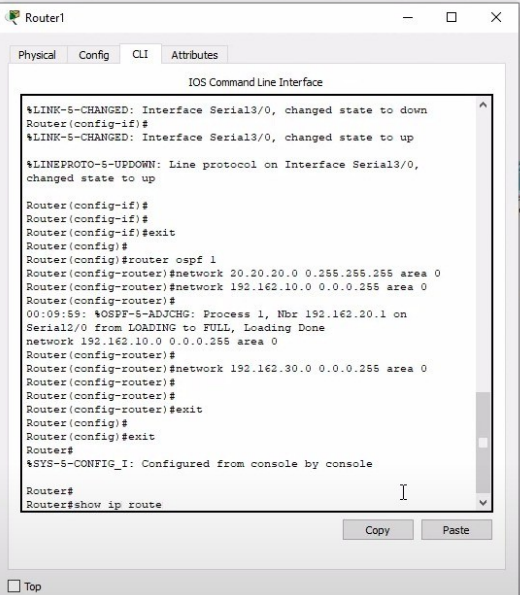
# network 192.162.20.0 0.0.0.255 area 0

# exit

# exit

# show ip route

Step 7 : Establishing the connections in router 1



Comment to be follow:

# exit

# router ospf 1

# network 20.20.20.0 0.255.255.255 area 0

# network 192.162.10.0 0.0.0.255 area 0

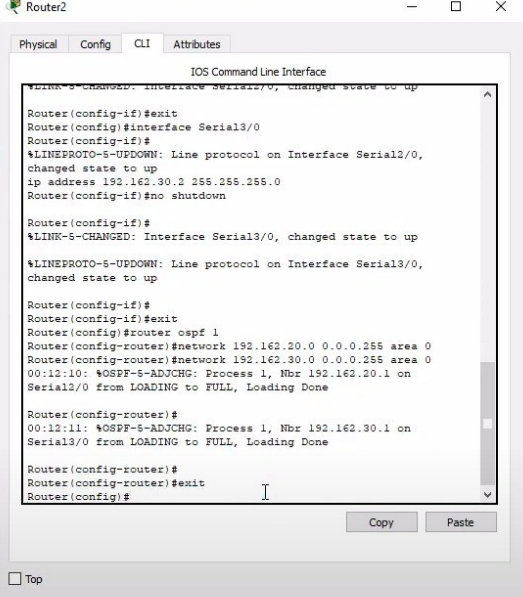
# network 192.162.30.0 0.0.0.255 area 0

# exit

# exit

# show ip route

Step 8 : Establishing the connections in router 2



Comment to be follow:

# exit

# router ospf 1

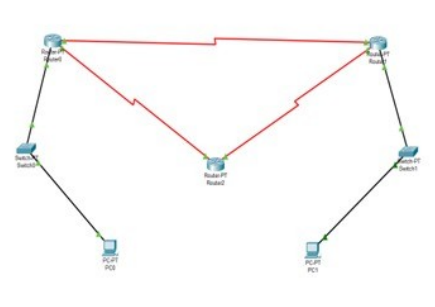
# network 192.162.20.0 0.0.0.255 area 0

# network 192.162.30.0 0.0.0.255 area 0

# exit

# exit

Step 9 : Comment to be follow : tracert 20.20.20.2



# 5(b).Performance Analysis of RIP Dynamic Routing Protocol using Cisco Packet Tracer

## Aim:

The aim of this experiment is to perform a performance analysis of the RIP (Routing Information Protocol) dynamic routing protocol using Cisco Packet Tracer. The analysis will focus on understanding the convergence time, routing table updates, and scalability of RIP in a network environment.

## Theory:

### RIP (Routing Information Protocol):

* RIP is a distance vector routing protocol used to exchange routing information between routers in a network.
* It uses the Bellman-Ford algorithm to determine the best path to a destination based on the number of hops.
* RIP routers periodically broadcast their routing tables to neighboring routers, allowing them to update their own tables.

## Procedures:

### 1. Setting Up the Network:

* Create a network topology in Cisco Packet Tracer with multiple routers interconnected.
* Assign IP addresses to router interfaces and configure RIP as the routing protocol.

### 2. Configuring RIP on Routers:

* Access the command-line interface (CLI) of each router and configure RIP using commands such as router rip and network [network\_address].
* Set the version of RIP (RIP v1 or v2) and configure other relevant parameters.

### 3. Monitoring Routing Tables:

* Access the routing tables of the routers to observe the current state of the network.
* Use commands like show ip route to display the routing table and check the routes learned from neighboring routers.

### 4. Introducing Network Changes:

* Simulate changes in the network, such as link failures or additions, to observe how quickly RIP adapts to these changes.
* Monitor the convergence time, which is the time it takes for the routers to update their routing tables after a topology change.

### 5. Analyzing Routing Updates:

* Capture and analyze RIP routing updates using packet capturing tools like Wireshark.
* Examine the contents of RIP packets to understand the information being exchanged between routers.

### 6. Performance Metrics:

* Measure and analyze performance metrics such as:
  + Convergence Time: Time taken for the routers to adjust to changes in the network topology.
  + Routing Table Updates: Frequency and size of RIP updates exchanged between routers.
  + Stability: Assess the stability of the network under different scenarios.

### 7. Scaling the Network:

* Increase the size of the network by adding more routers and links to evaluate the scalability of RIP.
* Observe how well RIP handles larger networks and if there are any performance degradation issues.

### 8. Documentation and Reporting:

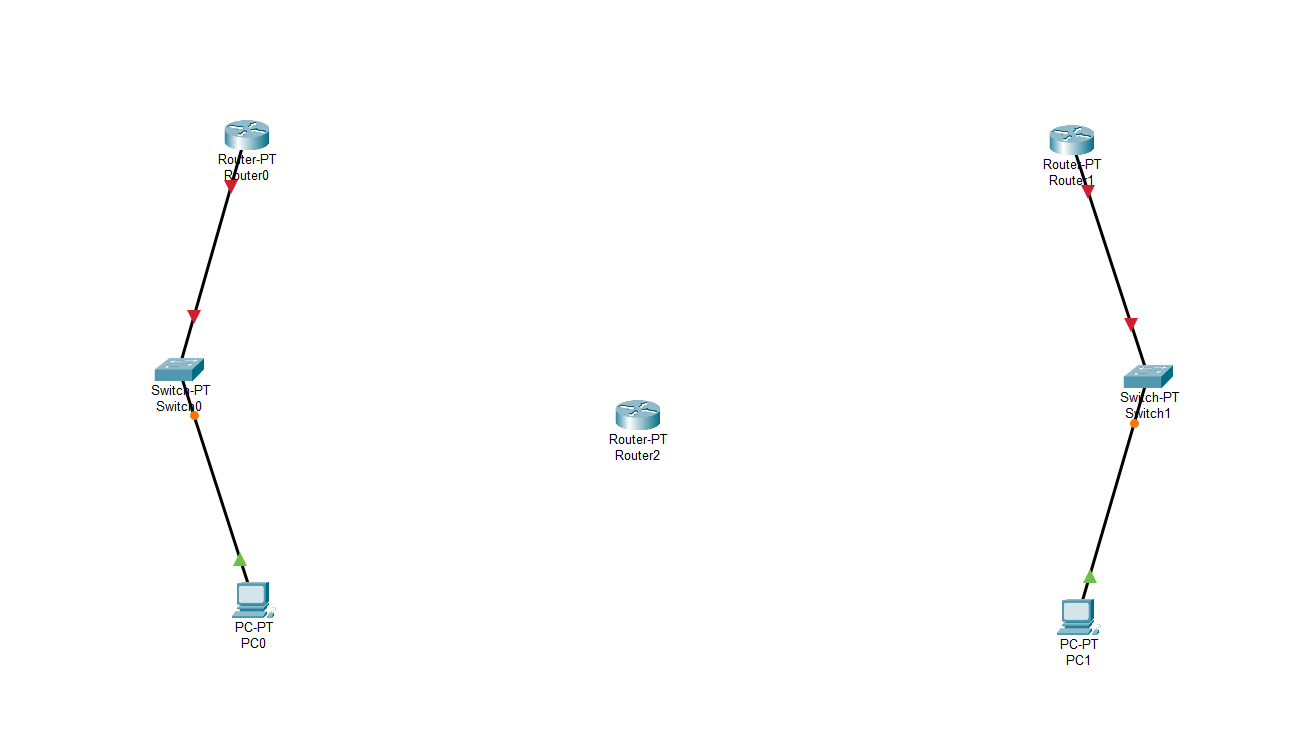
* Document the results and observations in a report.
* Provide insights into the strengths and limitations of RIP in terms of convergence time, routing updates, and scalability.

By following these procedures, you can conduct a thorough performance analysis of the RIP dynamic routing protocol using Cisco Packet Tracer. This analysis will help in understanding how RIP behaves in response to network changes and its suitability for specific network environments.

**Lab Procedure:**

**Step 1:** Open Cisco Packet Tracer and Arrange the components

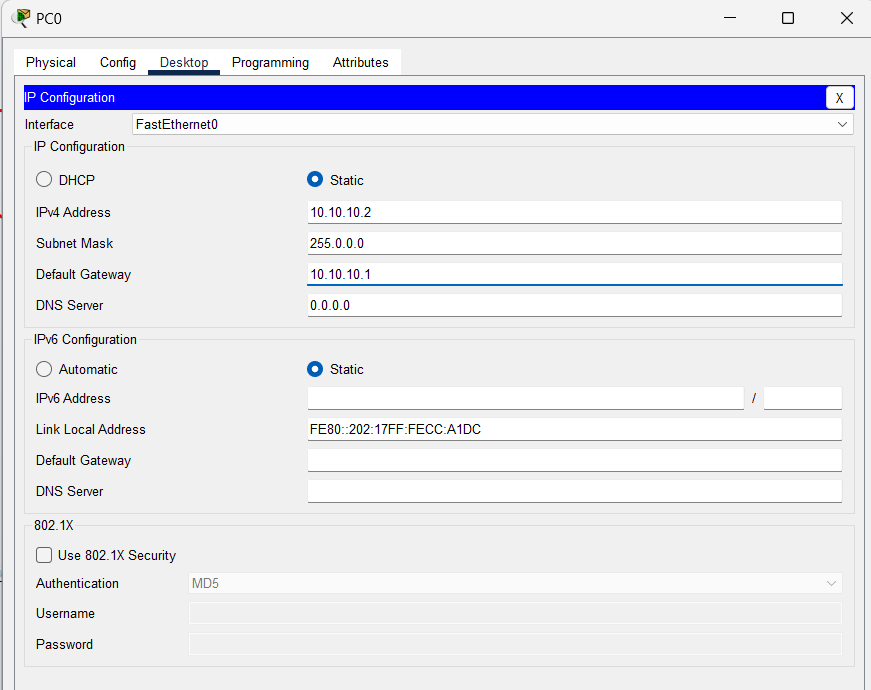
**Step 2:** Use "CopperStraightThrough" cables to connect PCs, servers, and switches among other devices.



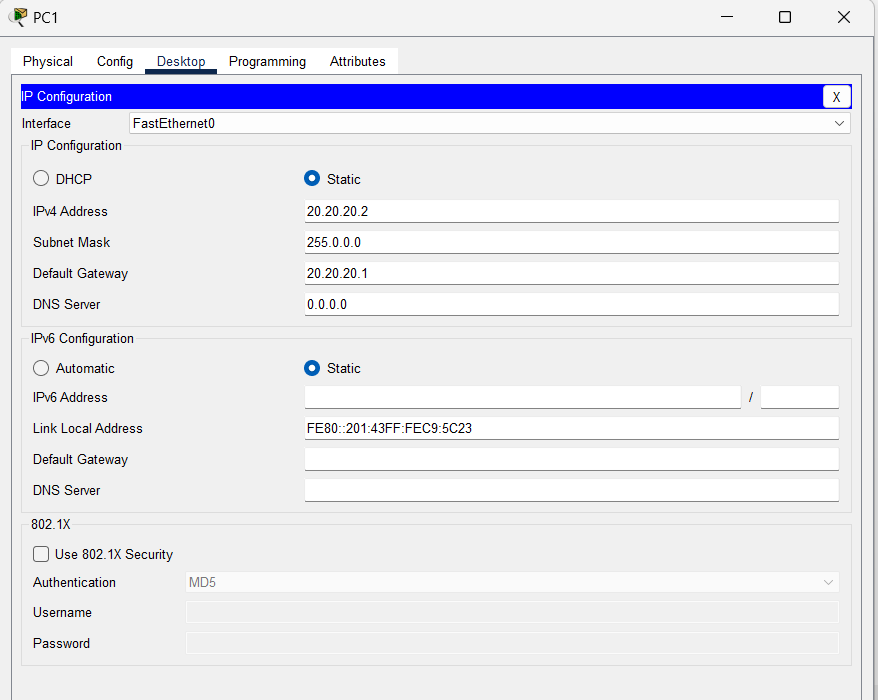
**Step 3:** Connect all three routers using the serial DTE(red) via series.

**Step 4:** Configure every device in this manner: Configure PCs (PC0, PC1,)

* Configure the PC0



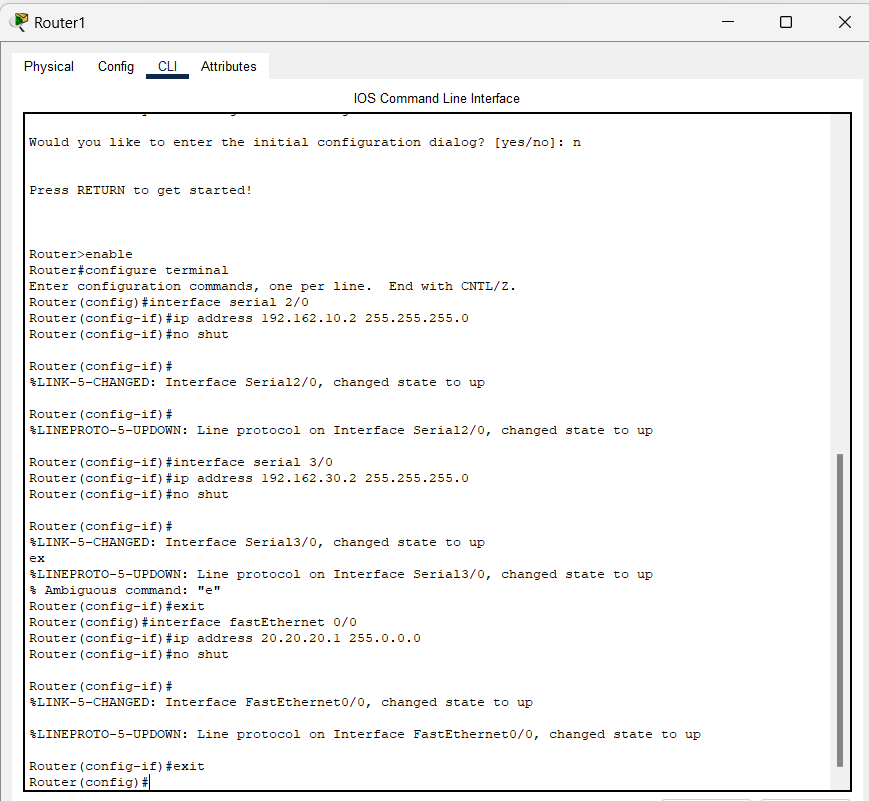
* Configure the PC1



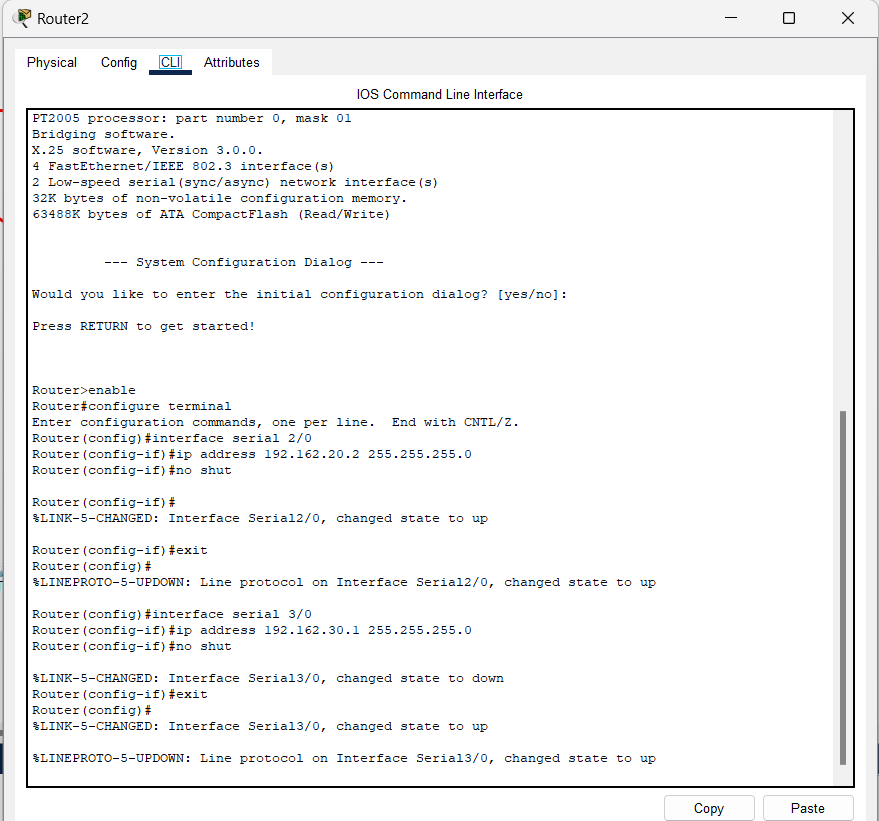
**Step 5:** Establishing the connections in router 0



**Step 6:** Establishing the connections in router 1



**Step 7:** Establishing the connections in router 2



**Step 8:** Here's what it should look like after completing all the stages mentioned above.Click PC0 after selecting the message symbol that appears beneath the undo button. On top of the computer, an icon for a message box shows. Using the same cursor, click PC1 now.

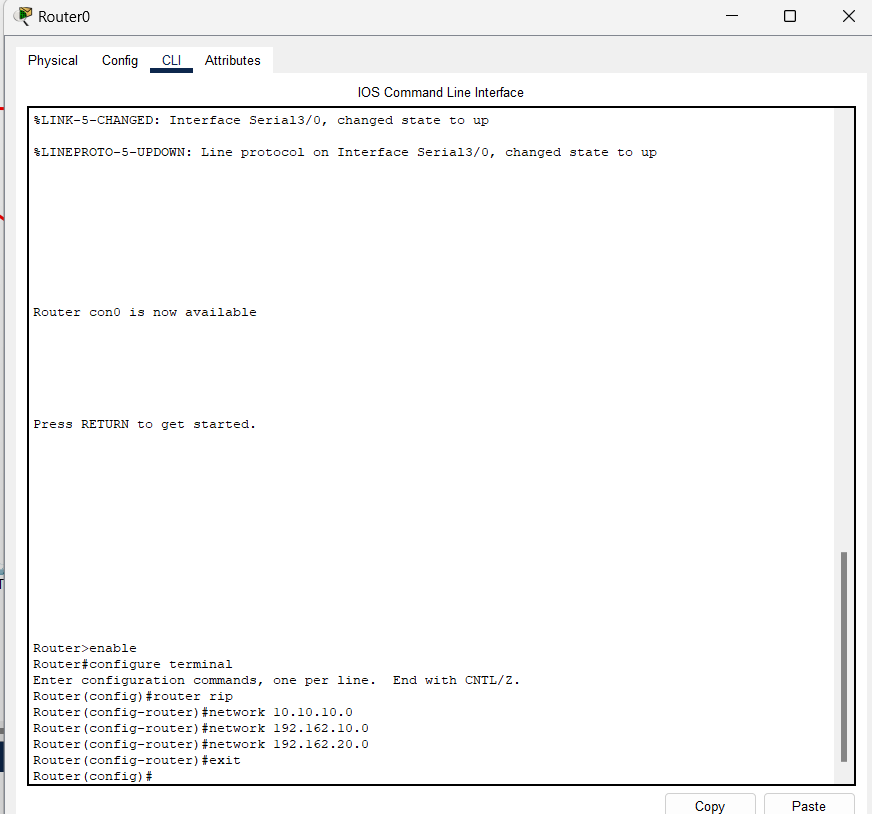


IT SHOULD FAIL

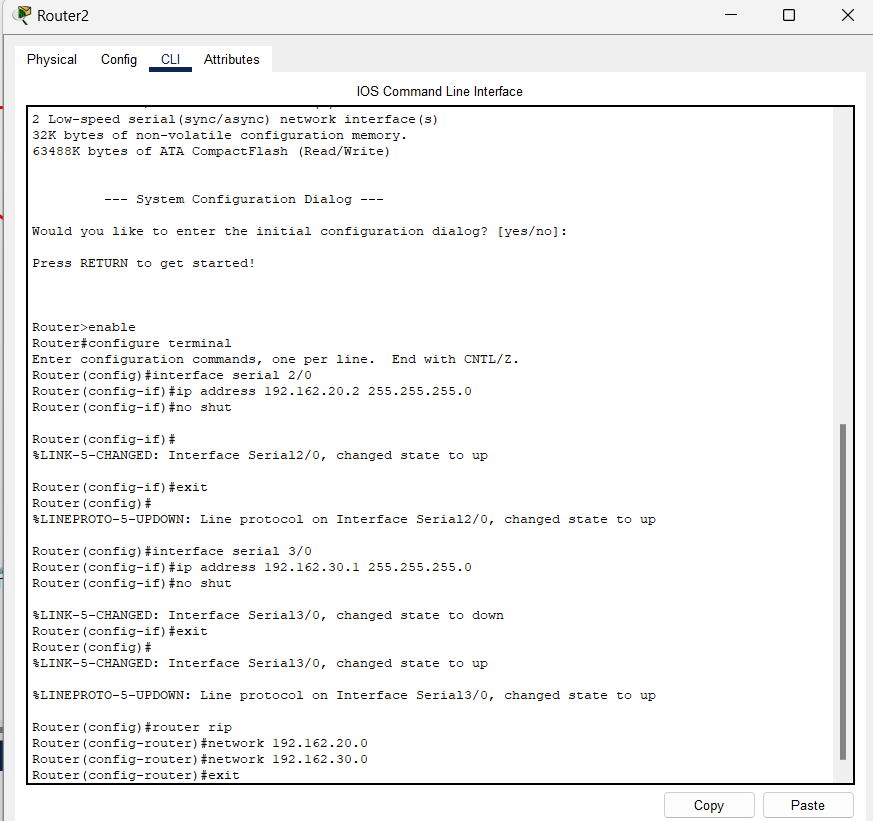
Reason: The routers should know each other, but at this point, they don't. This is when RIP (dynamic routing) comes into action. It introduces the other two routers and their destinations.

[it indicates failed in the bottom right corner of the app]

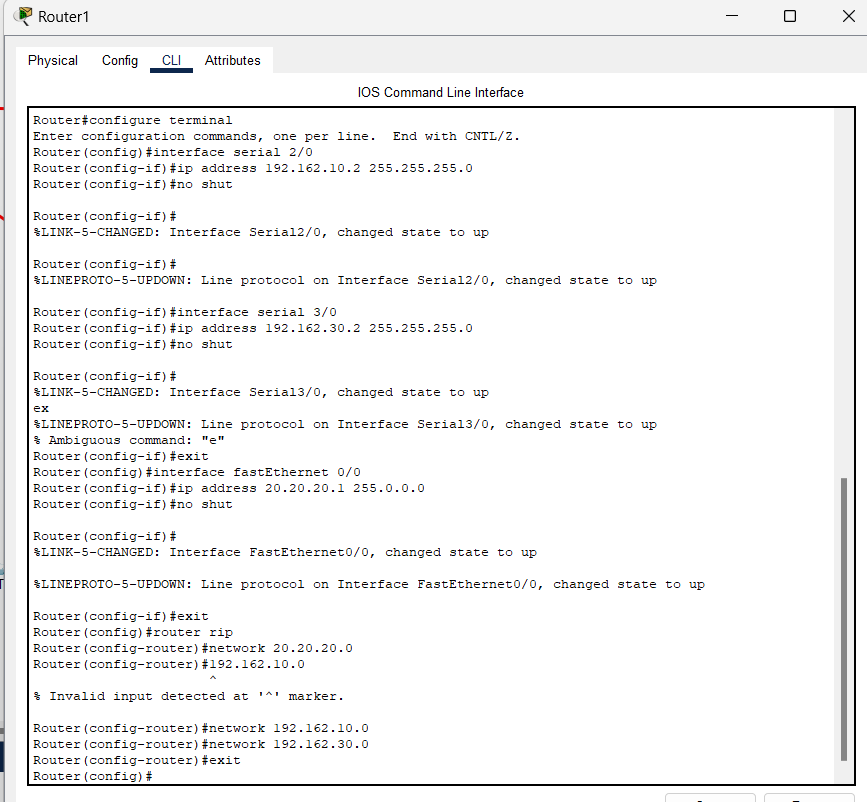
**Step 9:** Use the command router rip for router 0.



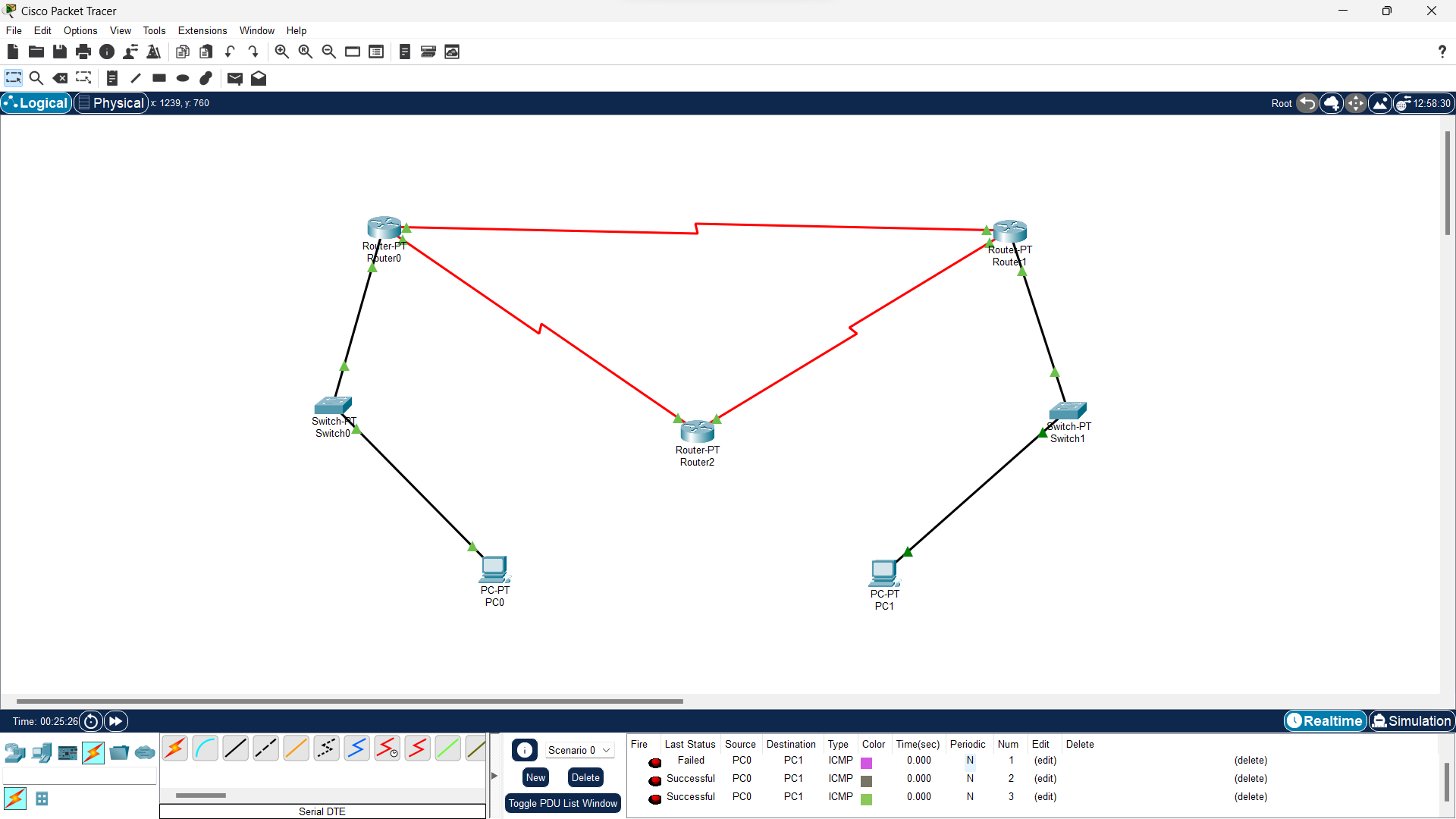
**Step 10:** Use the command router rip for router 2.

.

**Step 11:** Use the command router rip for router 1.



Now repeat the message step and it should be successful



# 6.Network Tools Demonstration

## Aim:

The aim of this experiment is to demonstrate the use of essential network tools, namely Ping, TCPDump, Traceroute, and Netstat. This demonstration will provide an understanding of their functionalities and how they can be employed for network troubleshooting and analysis.

## Theory:

### 1. Ping:

* Ping is a network utility tool used to test the reachability of a host on an Internet Protocol (IP) network.
* It measures the round-trip time for messages sent from the originating host to a destination computer.

### 2. TCPDump:

* TCPDump is a packet analyzer that allows the user to display TCP, UDP, and other packets being transmitted or received over a network.
* It provides a command-line interface for capturing and analyzing network traffic.

### 3. Traceroute:

* Traceroute is a diagnostic tool used to trace the route that packets take to reach a destination host.
* It displays the IP addresses of the routers or intermediary devices that the packets traverse.

### 4. Netstat:

* Netstat (Network Statistics) is a command-line tool that provides information about network connections, routing tables, interface statistics, masquerade connections, and more.
* It helps in monitoring and troubleshooting network-related issues.

## PROCEDURE:

## Ping:

### Command:

ping [www.google.com](http://www.google.com/)

## TCPDump:

### Command:

sudo tcpdump

## Traceroute:

### Command:

traceroute [www.google.com](http://www.google.com/)

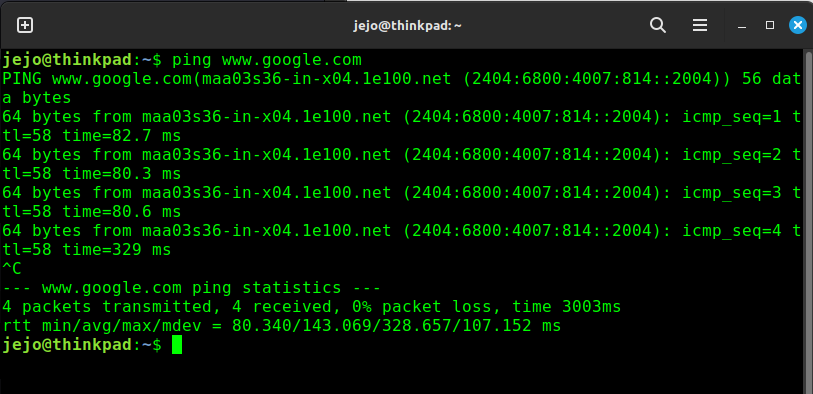
## Netstat:

### Command:

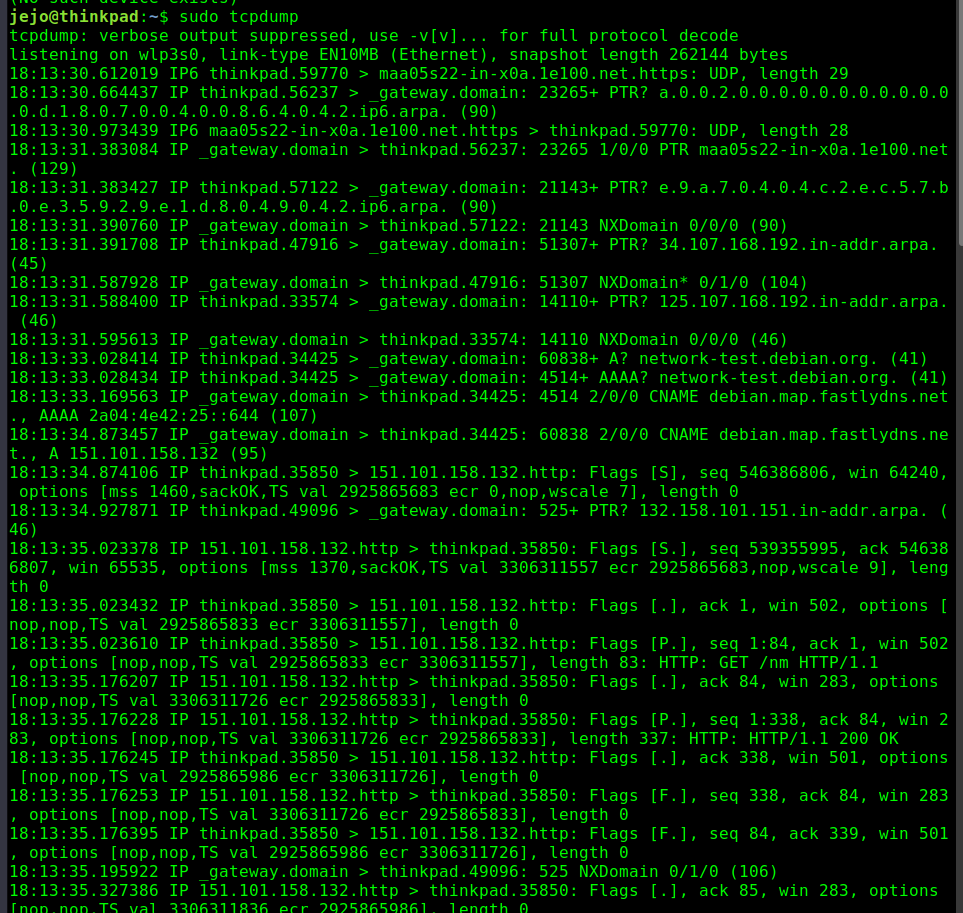
netstat -an

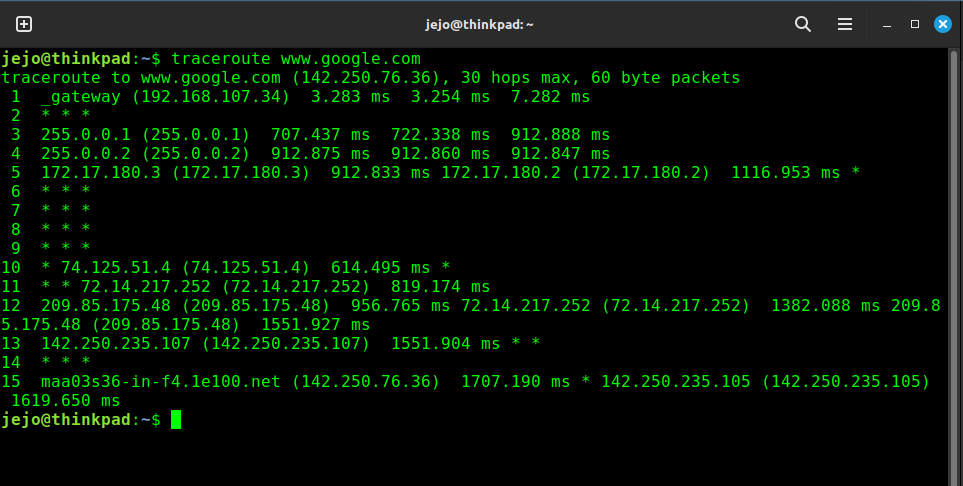
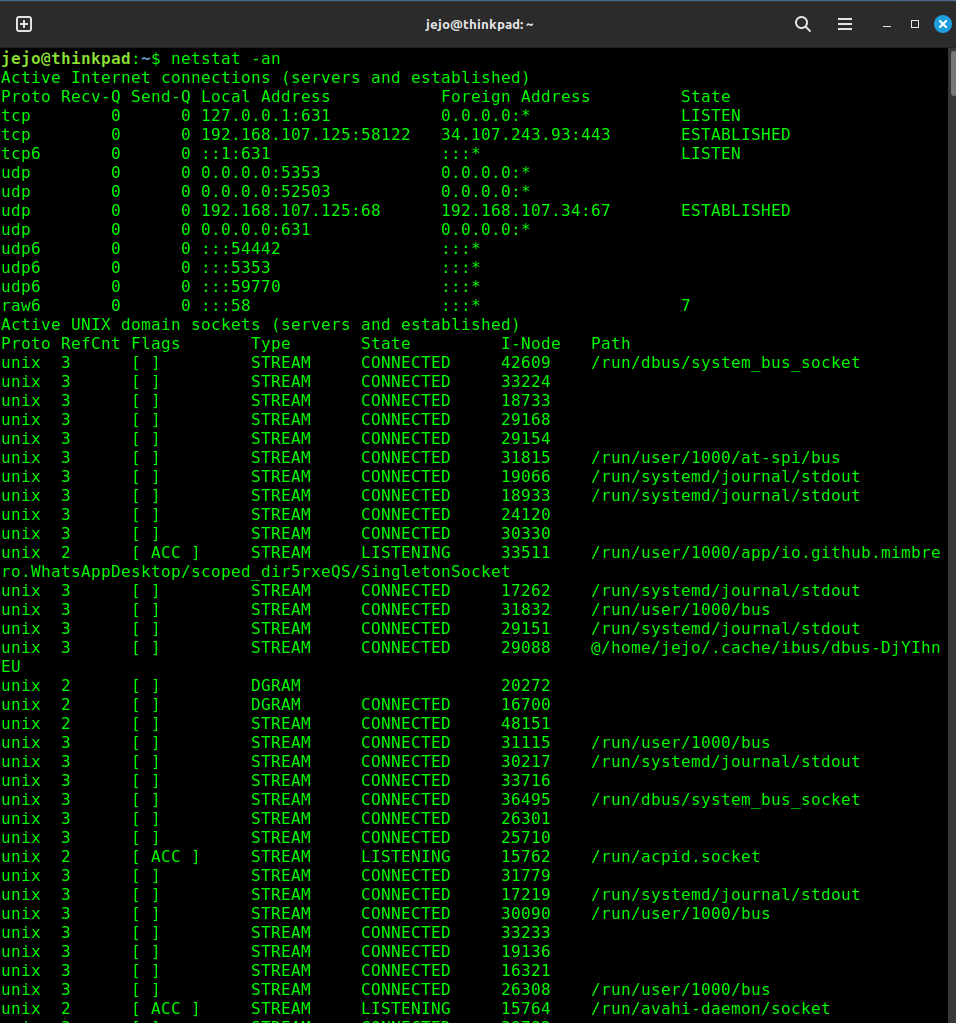
# OUTPUT

**PING:**

****

# TCPDump:

****

**Traceroute:**

# . NETSTAT: 7.NETWORK TRAFFIC ANALYSIS WITH WIRESHARK

## Aim:

The objective of this experiment is to conduct a comprehensive analysis of network traffic using Wireshark, a widely adopted network protocol analyzer. The primary goal is to gain insights into communication patterns, identify potential issues, and understand the protocols and data exchanged within the network.

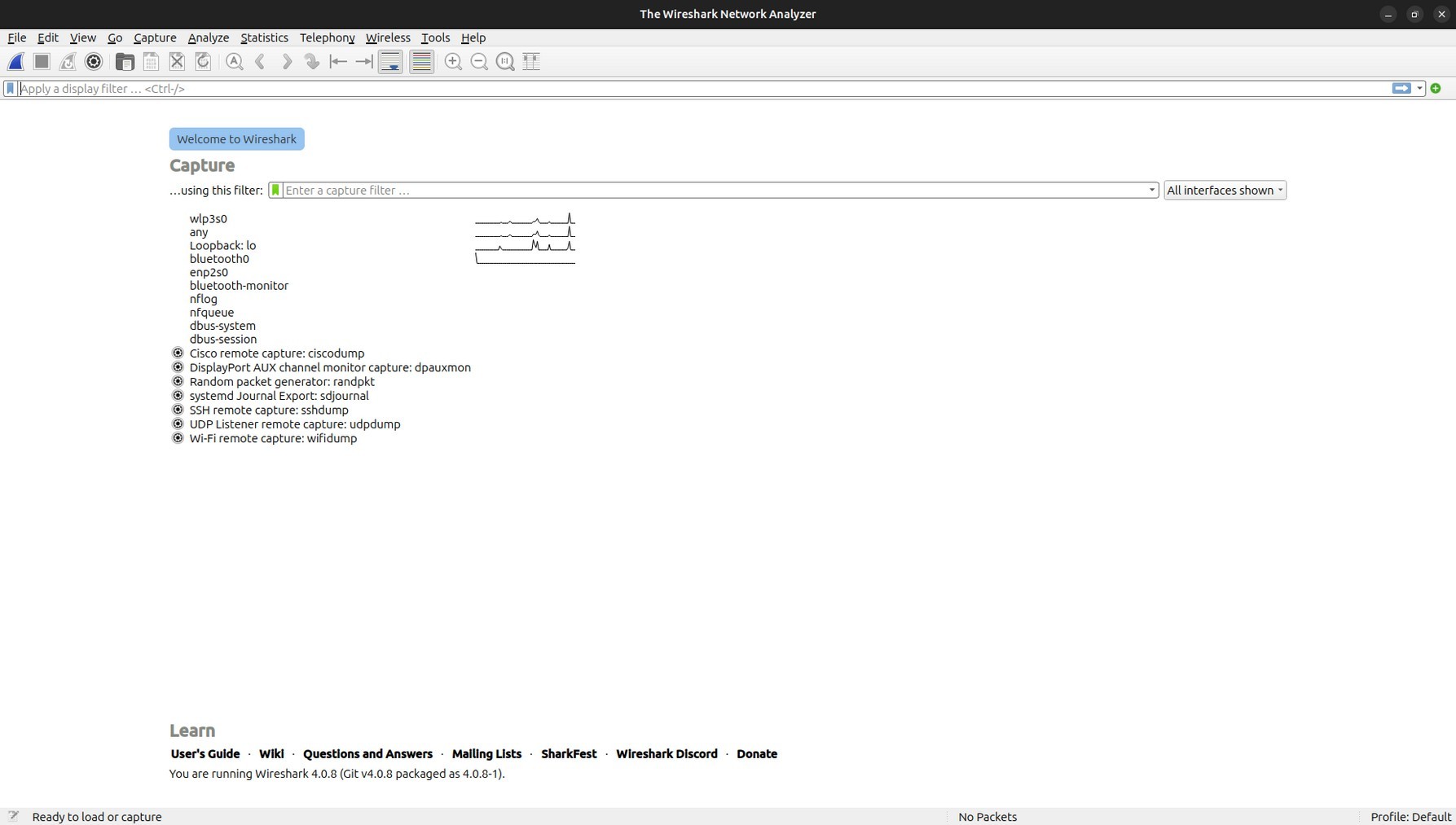
## Theory:

Wireshark serves as a potent open-source network analysis tool that facilitates the capture and inspection of packets on a network in real-time. Key concepts related to network traffic analysis using Wireshark include:

1. **Packet Capture:**
   * Wireshark captures packets flowing through a network interface, offering a detailed view of the data exchanged between devices.
   * Packets contain information about source and destination addresses, protocols used, and payload data.
2. **Protocol Analysis:**
   * Wireshark supports a broad range of network protocols, such as TCP, UDP, HTTP, DNS, and more.
   * It decodes and displays protocol-specific information, enabling analysts to comprehend the nature of communication.
3. **Filtering and Display Options:**
   * Wireshark permits users to apply filters to concentrate on specific types of traffic, source/destination addresses, or protocols.
   * Display options provide various views, including packet details, packet bytes, and packet tree, facilitating in-depth analysis.
4. **Follow Stream:**
   * The "Follow Stream" feature in Wireshark allows analysts to reconstruct and view the entire conversation between two devices for a specific protocol, providing context to the communication.
5. **Statistics and Metrics:**
   * Wireshark provides statistical information encompassing packet counts, data rates, and error counts.
   * Analyzing these metrics aids in identifying patterns, anomalies, and potential performance issues.

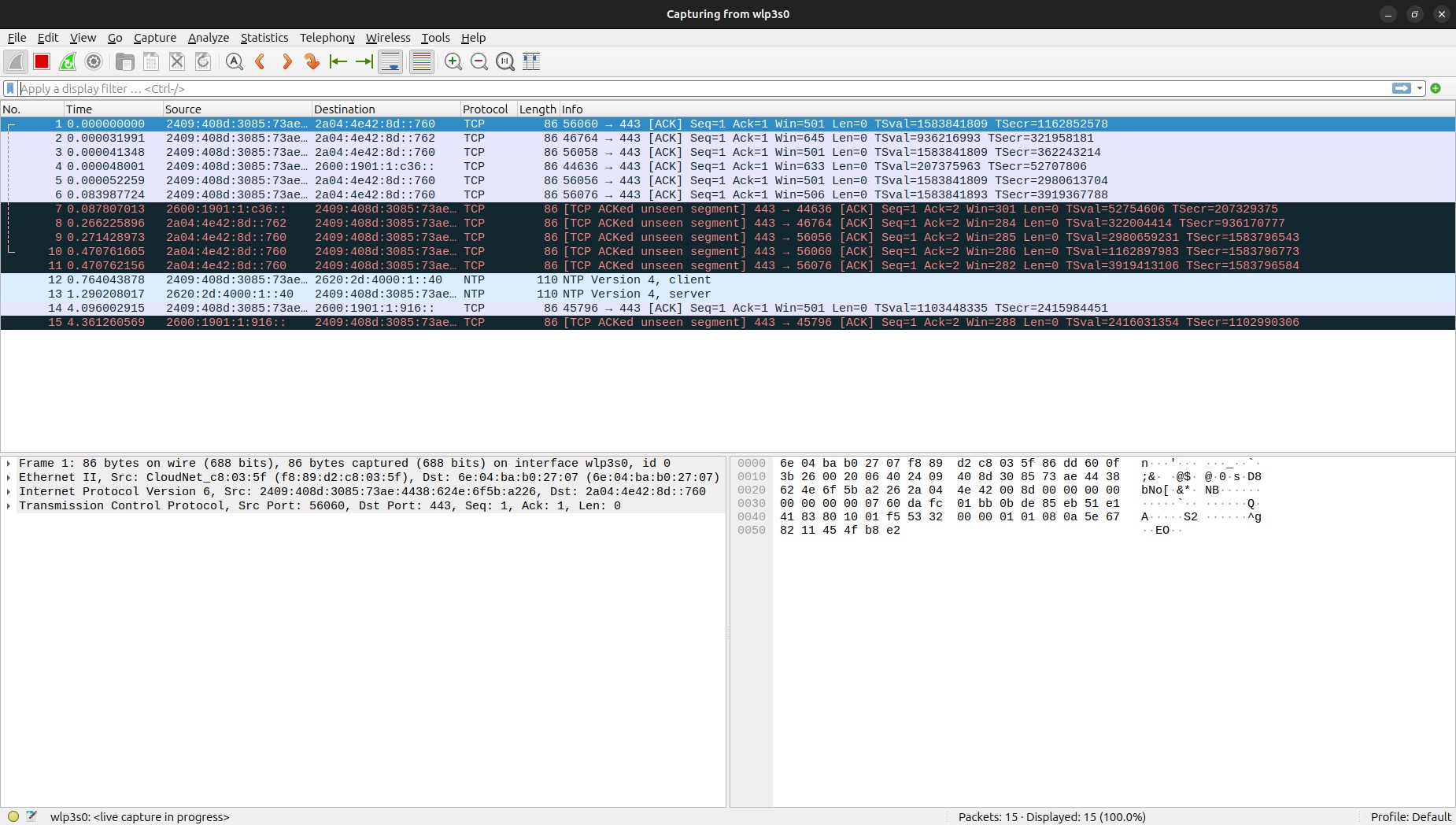
# Lab Procedure:

* 1. **Open Wireshark:**
     + Start Wireshark on your computer.

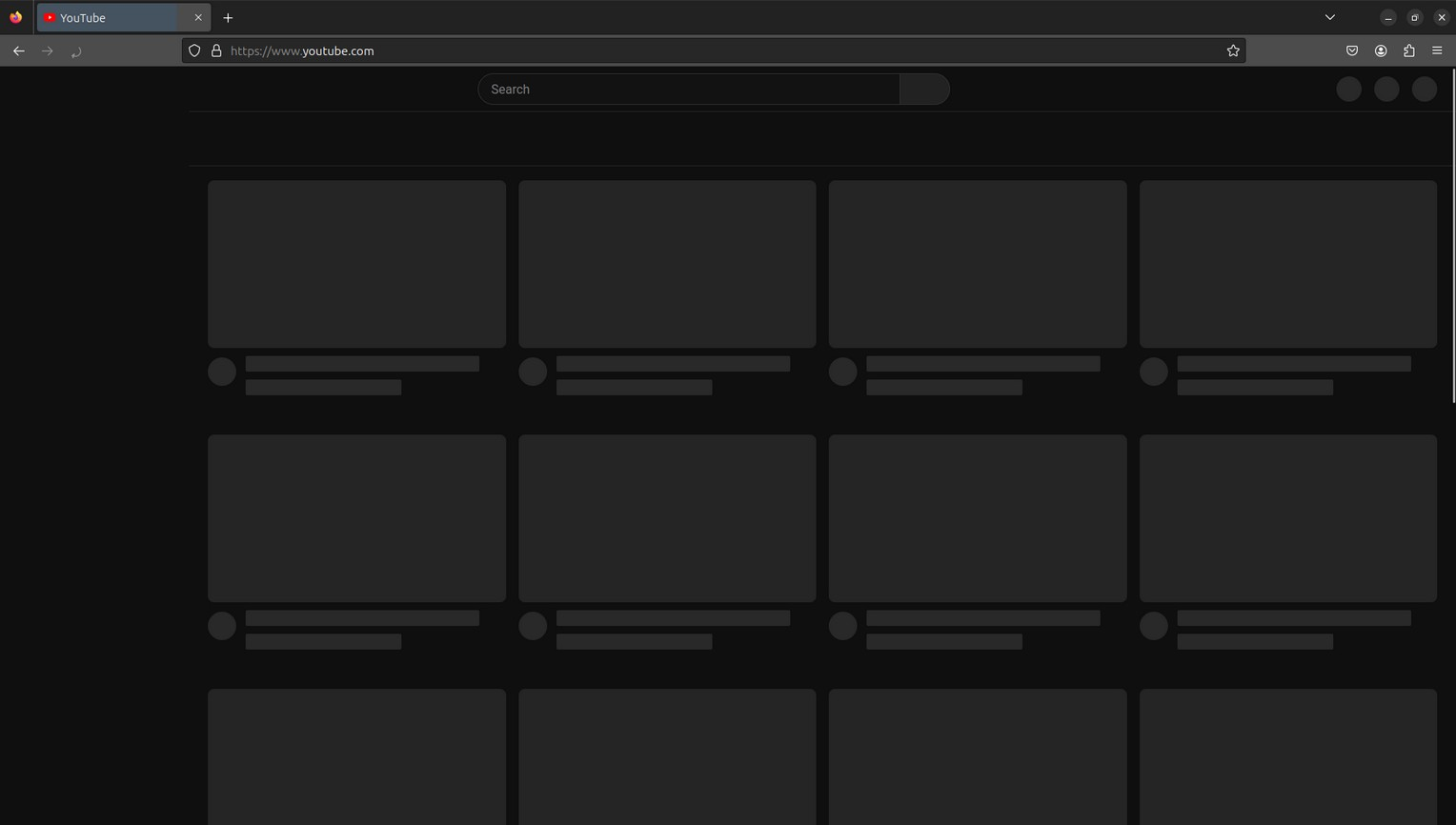


# Capture Traffic:

* + - Select the network interface.
    - Begin capturing packets.



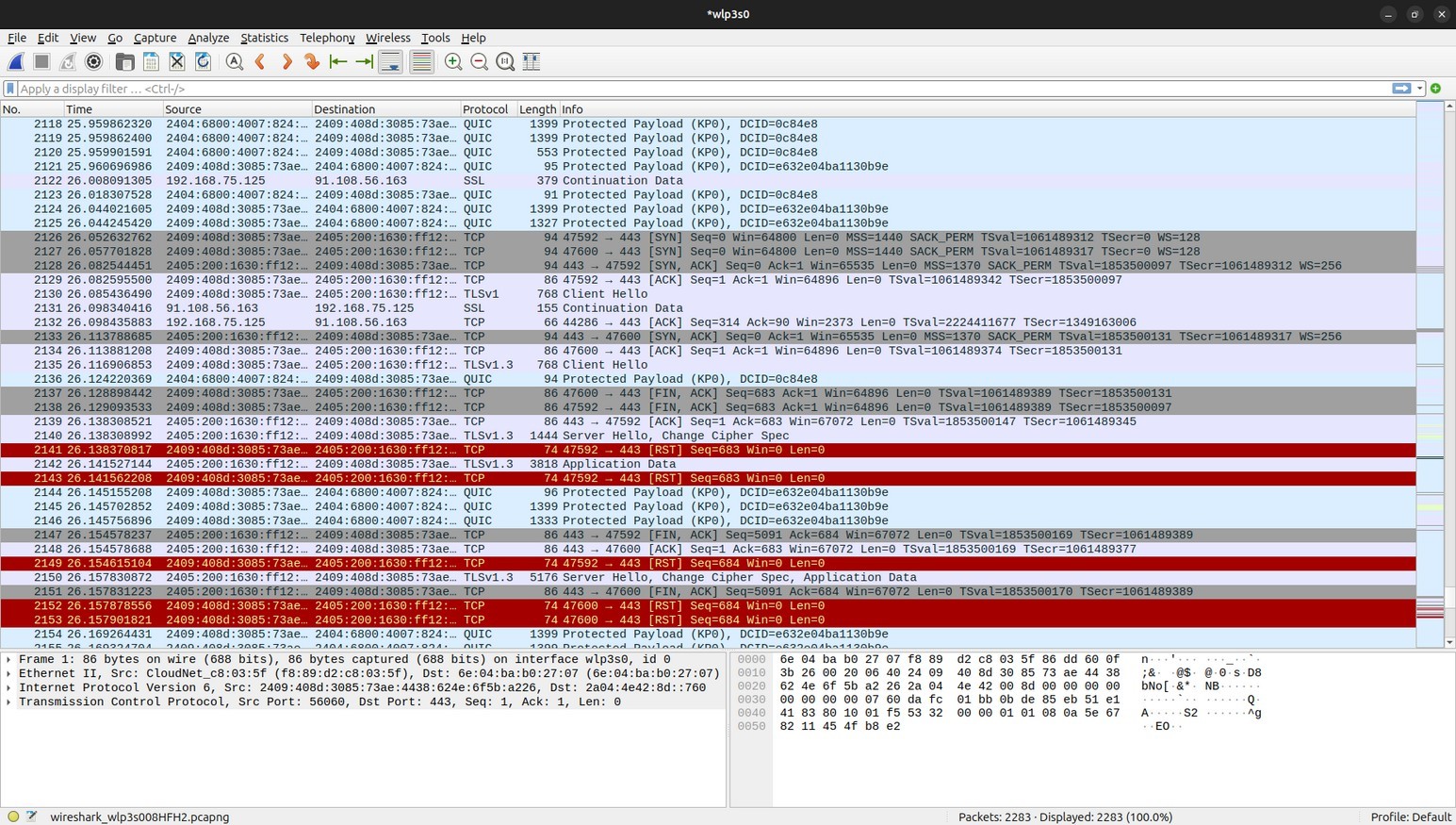
# Analyze Packets:

* + - open a sample website eg: ([www.youtube.com](http://www.youtube.com/))

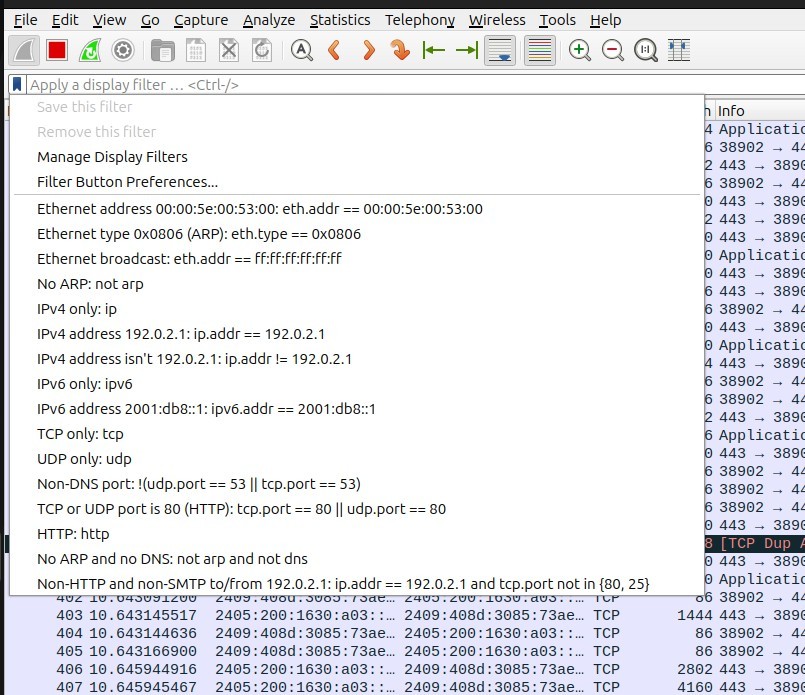
•

\*Stop the capture after a brief duration.

\*Review and analyze captured packets.



# Filtering:

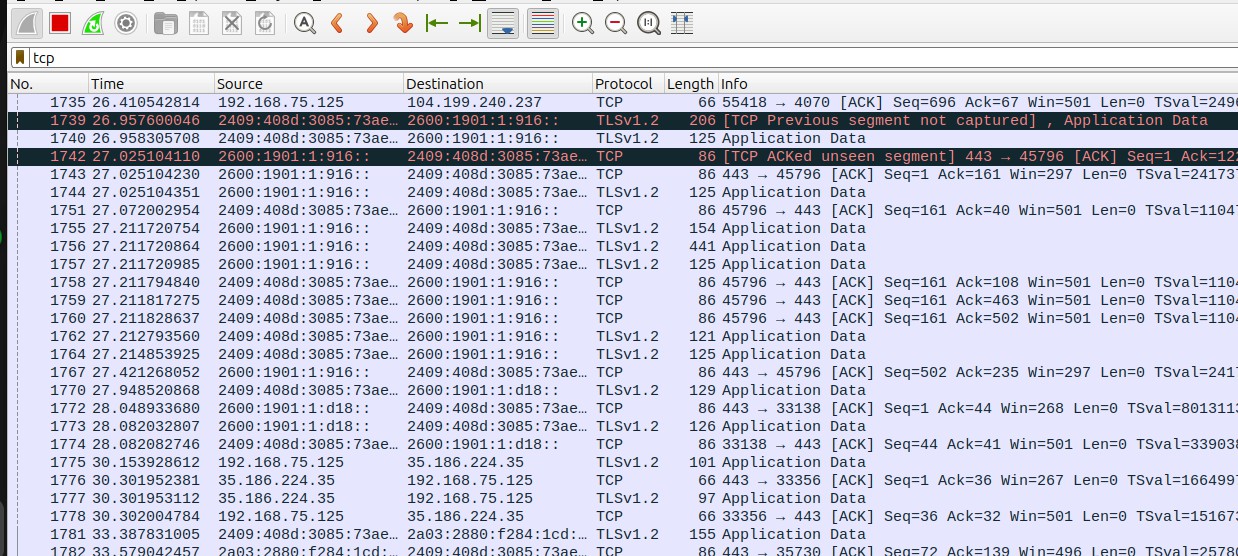
 Introduce basic display filters.

**Packet Details:**

* + - Examine details of selected packets.

# Follow TCP Stream:

* + - Analyze entire TCP conversations.



# saving the scan output:

\*our output will be saved in the /documents/wireshark-scan-result.pcapng form...

# 

# CONTENT BEYOND SYLLABUS

**1.Lab Manual: Developing a TCP Client-Server Application using Python Socket Programming using python**

# Lab Setup:

### Setting up the Environment:

#### Ensure that you have a Linux environment with Python installed.

* + open a terminal.

### Creating the Project Directory:

#### Create a new directory for your project.

bash

2. mkdir tcp\_message\_lab cd tcp\_message\_lab

# Part 1: Server Side

## Step 1: Writing the Server Code

#### 1. Create a file named server.py in the project directory.

python

1. # server.py import socket

PORT = 8080

MAX\_BUFFER\_SIZE = 1024

def main():

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) server\_socket.bind(('0.0.0.0', PORT))

server\_socket.listen(3)

print("Server listening on port", PORT)

client\_socket, client\_address = server\_socket.accept() print("Accepted connection from", client\_address)

data = client\_socket.recv(MAX\_BUFFER\_SIZE).decode('utf-8') print("Received message from client:", data)

client\_socket.close() server\_socket.close()

if name == " main ": main()

**Step 2: Running the Server Code**

#### 1. Run the server.

bash

1. python server.py

# Part 2: Client Side

## Step 1: Writing the Client Code

#### 1. Create a file named client.py in the project directory.

python

1. # client.py import socket

PORT = 8080

MAX\_BUFFER\_SIZE = 1024

def main():

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) server\_address = ('127.0.0.1', PORT)

try:

client\_socket.connect(server\_address) print("Connected to server on port", PORT)

message = input("Enter a message to send to the server: ") client\_socket.sendall(message.encode('utf-8'))

finally:

client\_socket.close()

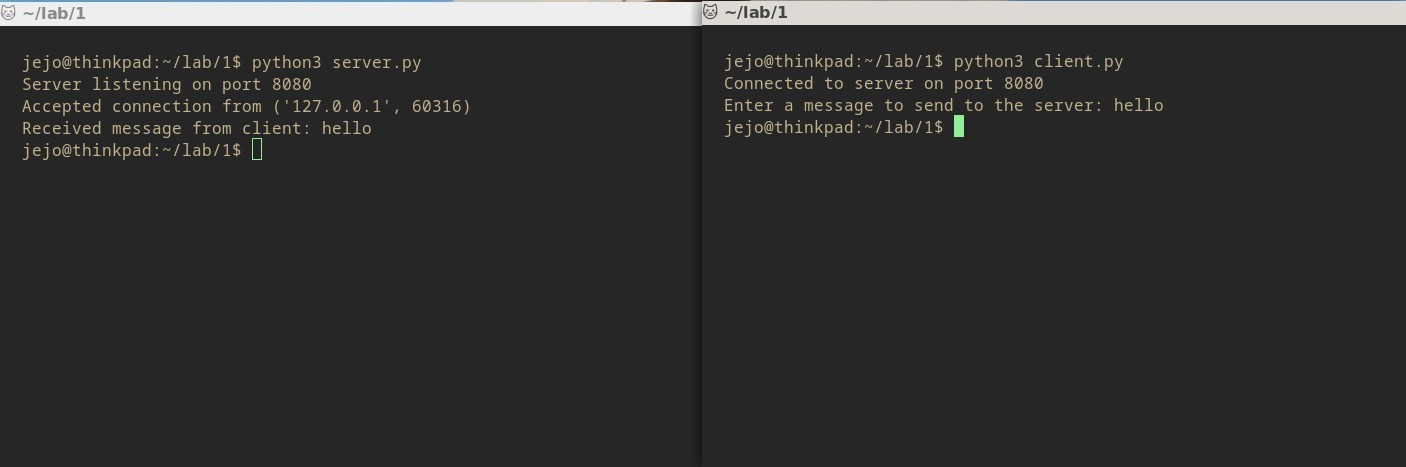
if name == " main ": main()

## Step 2: Running the Client Code

#### Run the client (in a separate terminal).

bash python client.py

**OUTPUT:**



**2. Developing a UDP Client-Server Application using Python Socket Programming and python**

## Lab Setup:

### 1. Setting up the Environment:

* Ensure that you have a UNIX-like operating system.
* Open a terminal.

### 2. Creating the Project Directory:

bash

mkdir udp\_message\_lab

cd udp\_message\_lab

## Part 1: Server Side

### Step 1: Writing the Server Code

1. Create a file named udp\_server.py in the project directory.

python

# udp\_server.py

import socket

PORT = 8080

MAX\_BUFFER\_SIZE = 1024

def main():

sockfd = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

server\_addr = ('0.0.0.0', PORT)

sockfd.bind(server\_addr)

print(f"UDP Server is listening on port {PORT}...")

while True:

data, client\_addr = sockfd.recvfrom(MAX\_BUFFER\_SIZE)

message = data.decode('utf-8')

print(f"Message from client: {message}")

sockfd.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Step 2: Running the Server Code

1. Run the server.

bash

python udp\_server.py

## Part 2: Client Side

### Step 1: Writing the Client Code

1. Create a file named udp\_client.py in the project directory.

python

# udp\_client.py

import socket

PORT = 8080

SERVER\_IP = '127.0.0.1'

MAX\_BUFFER\_SIZE = 1024

def main():

sockfd = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

server\_addr = (SERVER\_IP, PORT)

while True:

message = input("Enter a message to send to the server: ")

sockfd.sendto(message.encode('utf-8'), server\_addr)

sockfd.close()

if \_\_name\_\_ == "\_\_main\_\_":

main()

### Step 2: Running the Client Code

1. Run the client (in a separate terminal).

bash

python udp\_client.py

**OUTPUT:**

# 