

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

**LAB RECORD**

**Computer Network Lab (23CS5PCCON)**

***Submitted by***

**NITHESH (1BM23CS218)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**September 2025 – January 2026**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

****

**CERTIFICATE**

This is to certify that the Lab work entitled “Computer Network (23CS5PCCON)” carried out by **Nithesh (1BM23CS218),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements of the above-mentioned subject and the work prescribed for the said degree.

Dr. Kavitha Sooda Professor & HOD

Department of CSE, BMSCE

Sarala D V Assistant Professor

Department of CSE, BMSCE

Index

**Part - A**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 19/08/25 | Create a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices and demonstrate  ping message. | 1 – 3 |
| 2 | 09/09/25 | Configure DHCP within a LAN and outside LAN. | 4 – 5 |
| 3 | 09/09/25 | Configure Web Server, DNS within a LAN. | 6 – 7 |
| 4 | 09/09/25 | Configure IP address to routers in packet tracer.  Explore the following messages: ping responses, destination unreachable, request timed out, reply. | 8 – 9 |
| 5 | 23/09/25 | Configure default route, static route to the Router. | 10 – 12 |
| 6 | 23/09/25 | Configure RIP routing Protocol in Routers. | 13 – 15 |
| 7 | 14/10/25 | Configure OSPF routing protocol. | 16 – 17 |
| 8 | 14/10/25 | To construct a VLAN and make the PC’s communicate among a VLAN. | 18 – 19 |
| 9 | 11/11/25 | To construct a WLAN and make the nodes communicate wirelessly. | 20 – 22 |
| 10 | 11/11/25 | Demonstrate the TTL/ Life of a Packet. | 23 – 24 |
| 11 | 18/11/25 | To understand the operation of TELNET by accessing the router in server room from a PC in IT office. | 25 – 27 |
| 12 | 18/11/25 | To construct simple LAN and understand the concept and operation of Address Resolution Protocol (ARP). | 28 – 30 |

**Part - B**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 28/10/25 | Write a program for congestion control using Leaky bucket algorithm. | 31 – 34 |
| 2 | 17/11/25 | Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present. | 35 – 36 |
| 3 | 17/11/25 | Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present. | 37 – 38 |
| 4 | 28/10/25 | Write a program for error detecting code using CRC-CCITT (16-bits). | 39 - 43 |

Github Link:

https://github.com/VCnithesh944/ComputerNetworksLAB

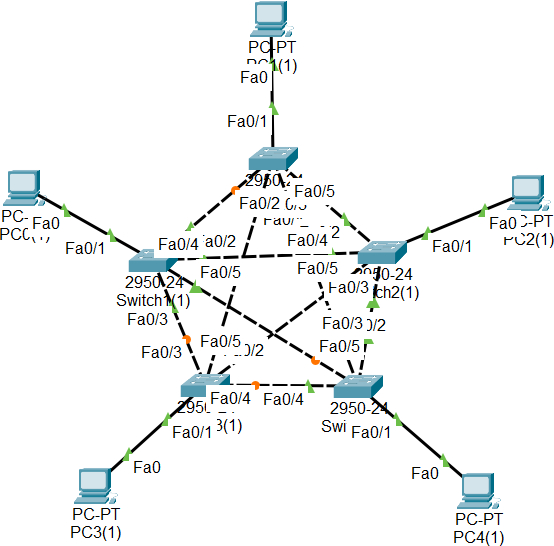
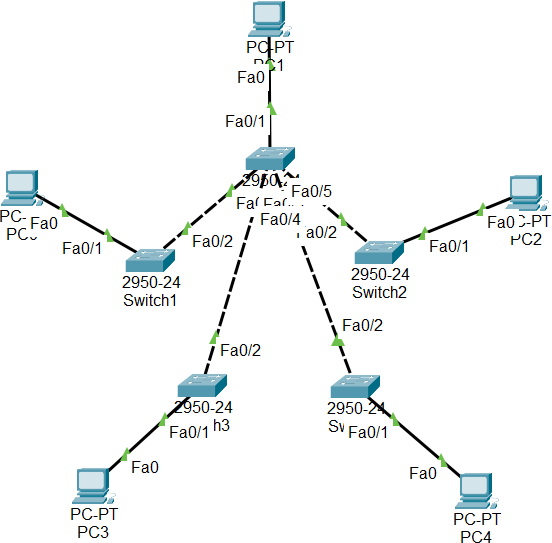
# PART - A

**Program 1:**

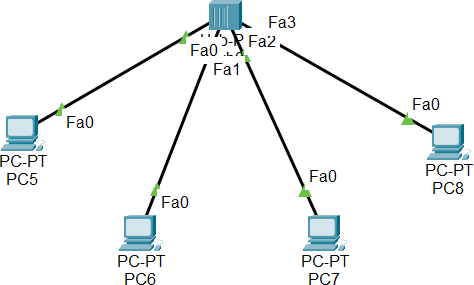
**Aim:** Create a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices and demonstrate ping message.

**Network diagram:**

1. STAR Topology with Switch: 2. MESH Topology with Switch:



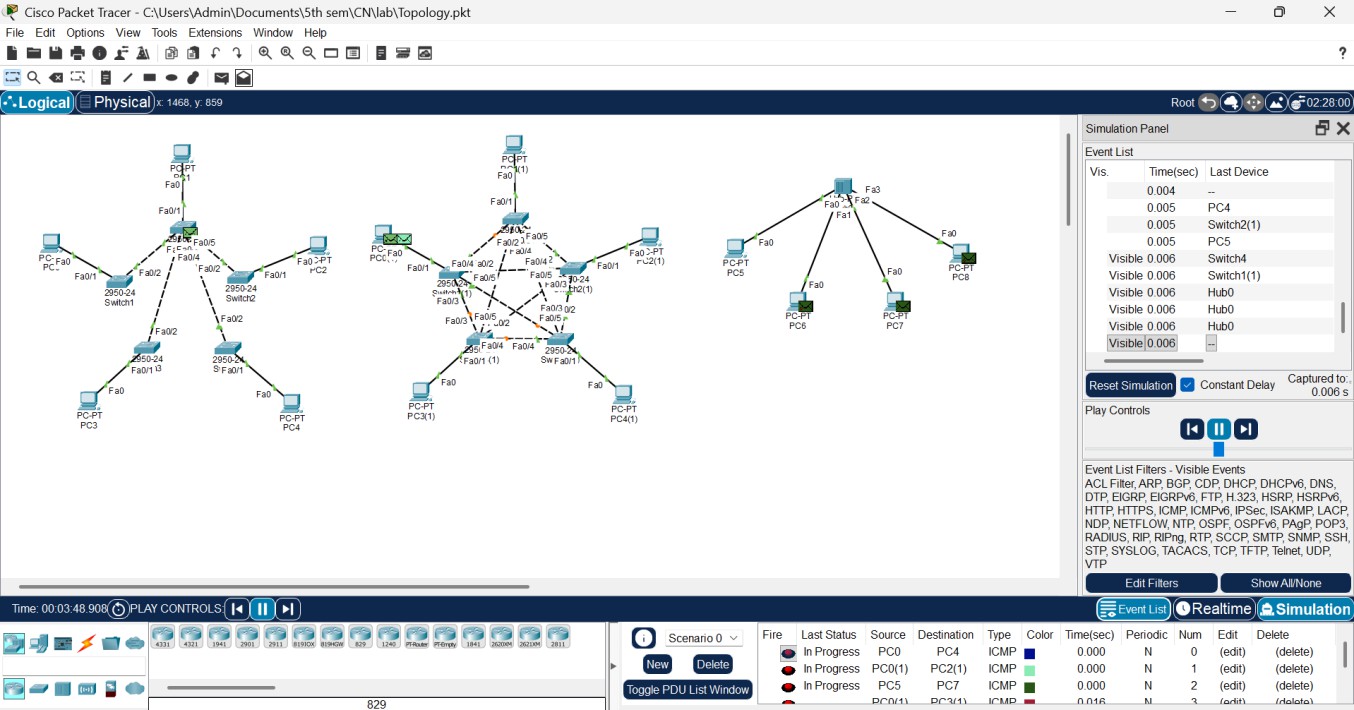
3. HUB-Based Network Topology:



**Configuration:**

|  |  |
| --- | --- |
|  |  |

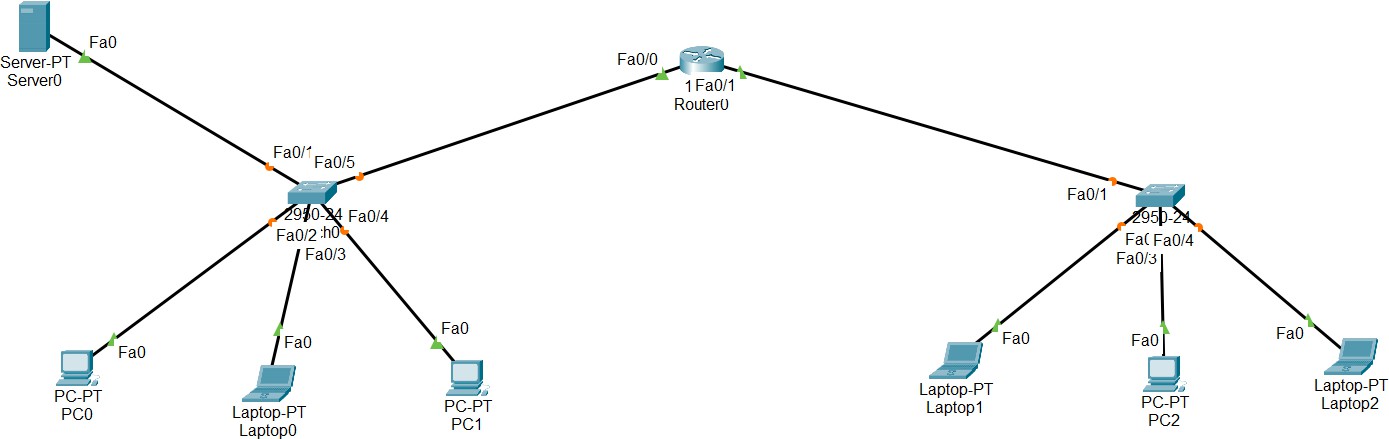
**Output:**



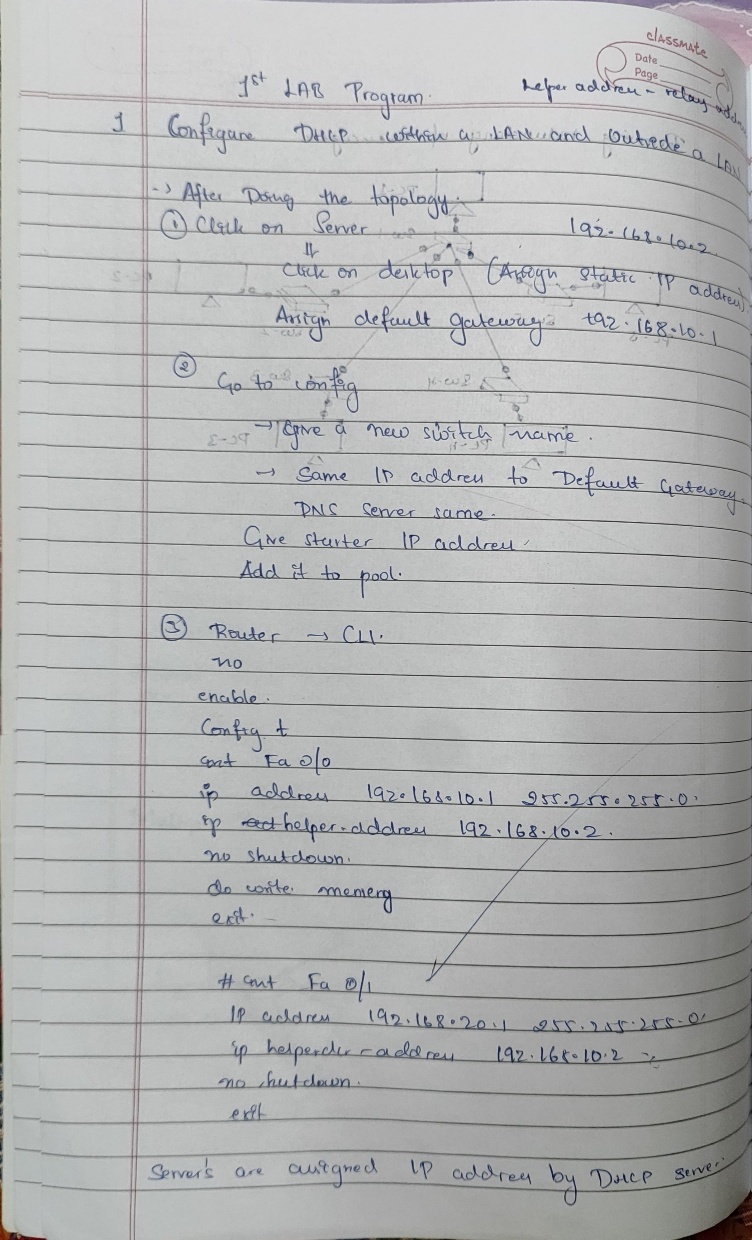
**Program 2:**

**Aim:** Configure DHCP within a LAN and outside LAN.

**Network diagram:**



**Configuration:**

****

**Output:**

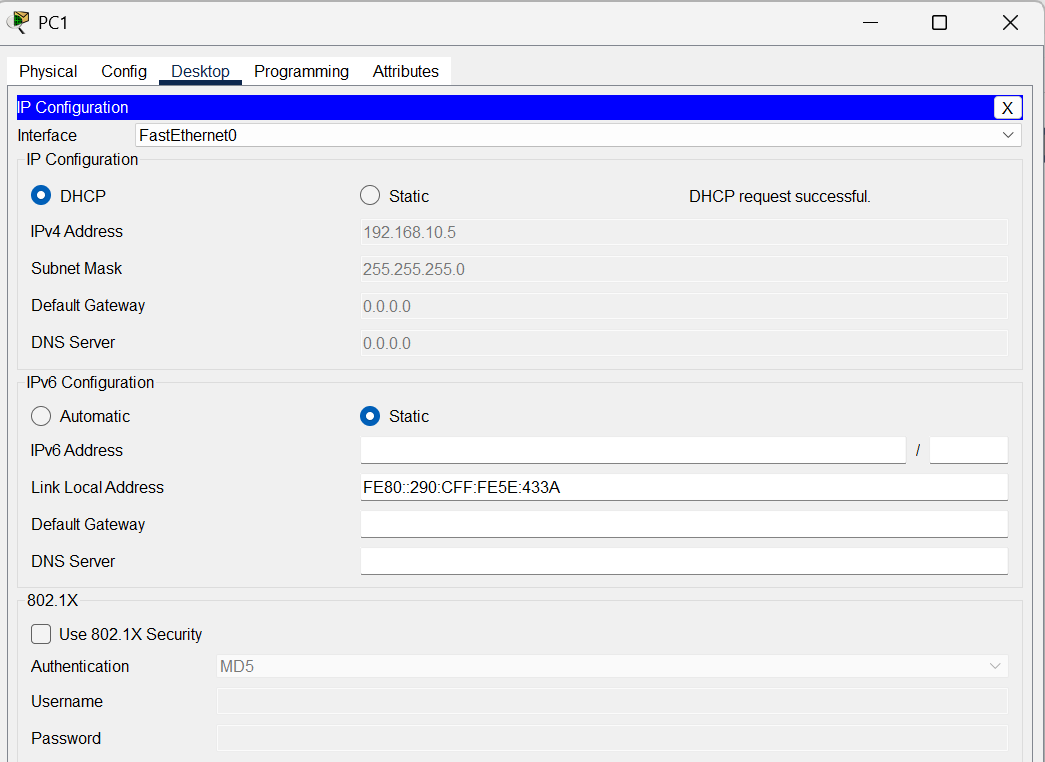
****

Fig 1. Ip address assigned by DHCP server within Lan (PC1)

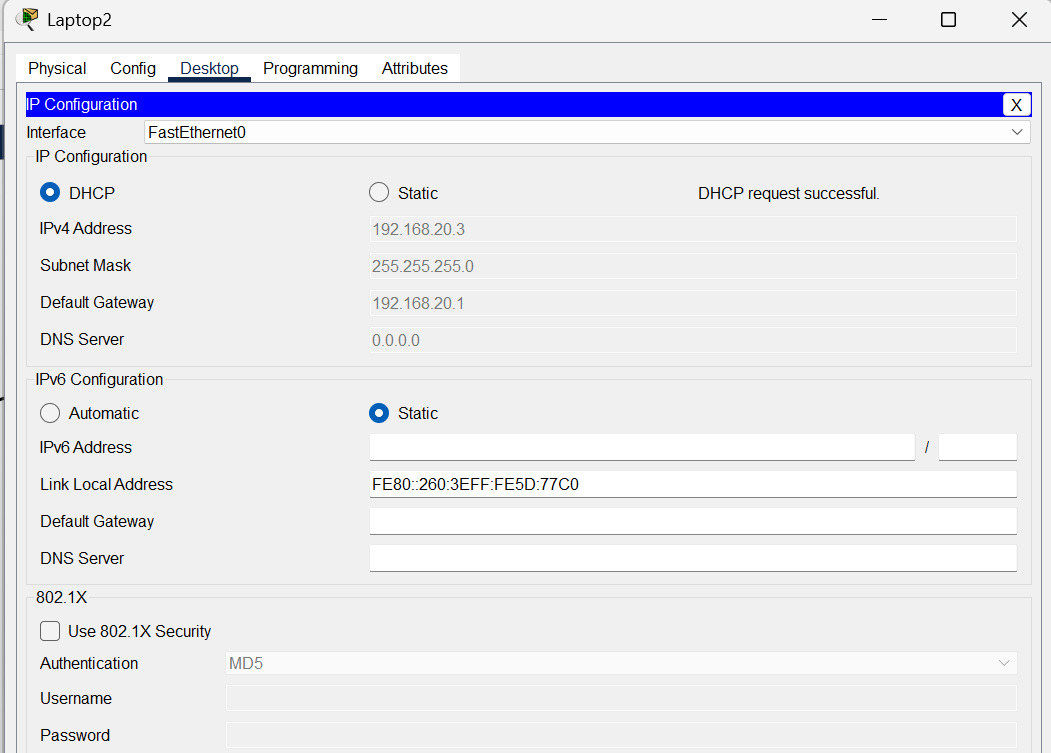
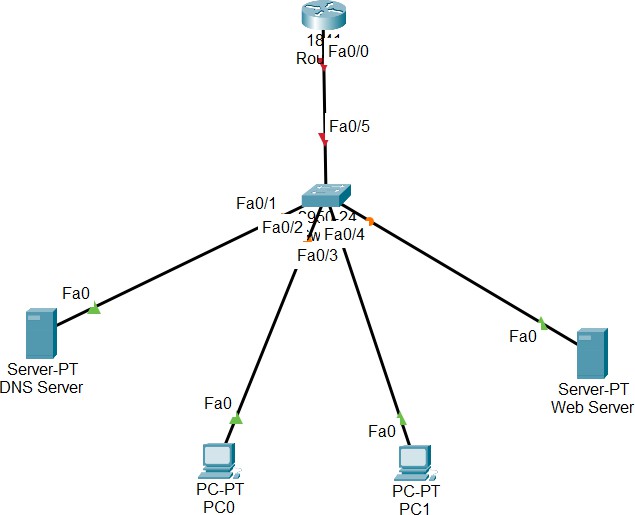


Fig 2. Ip address assigned by DHCP server outside Lan (laptop2)

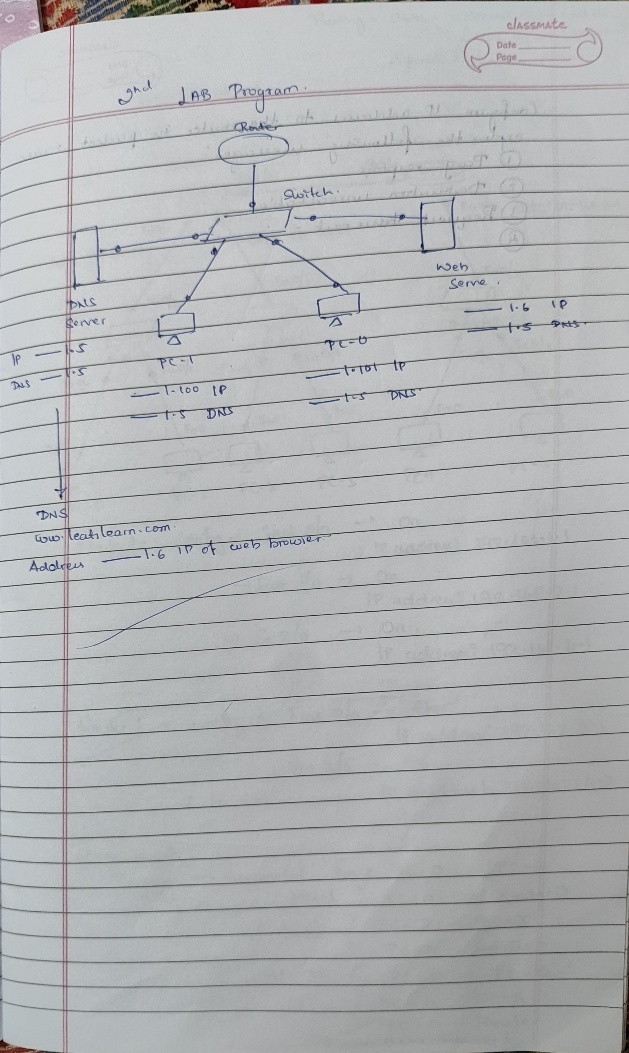
**Program 3:**

**Aim:** Configure Web Server, DNS within a LAN.

**Network diagram:**



**Configuration:**

****

**Output:**

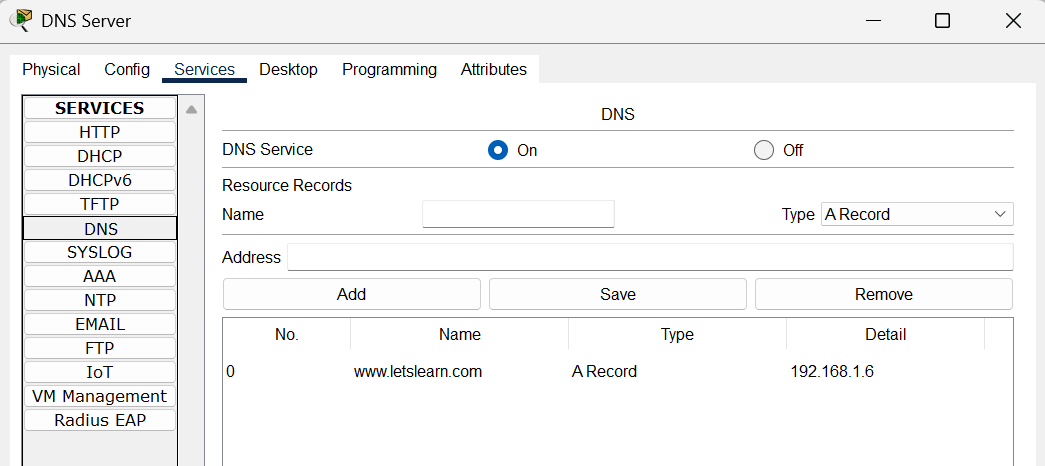
****

Fig 1. DNS server – DNS Services

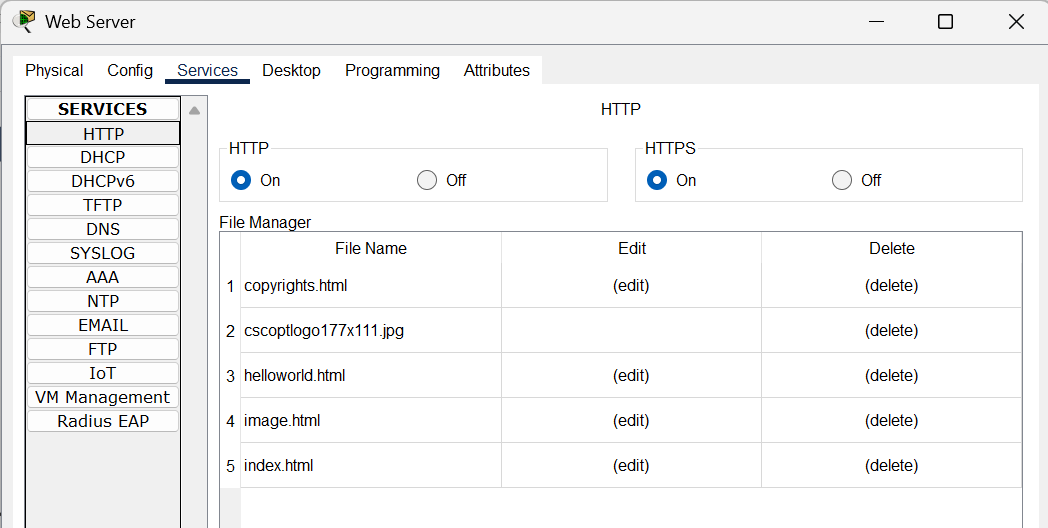


Fig 2. WEB server – HTTP Services

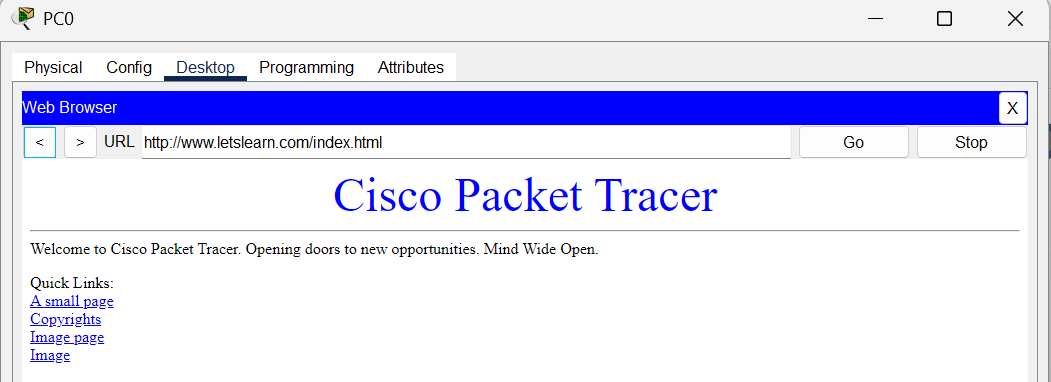
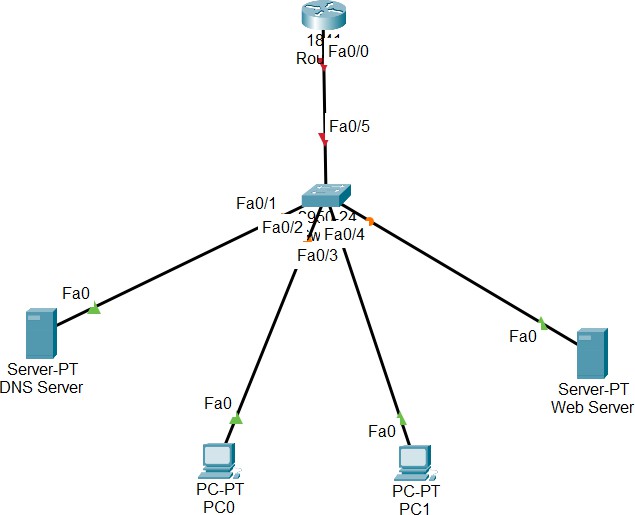


Fig 3. PC0 – accessing data from web browser

**Program 4:**

**Aim:** Configure IP address to routers in packet tracer. Explore the following messages: ping responses, destination unreachable, request timed out, reply.

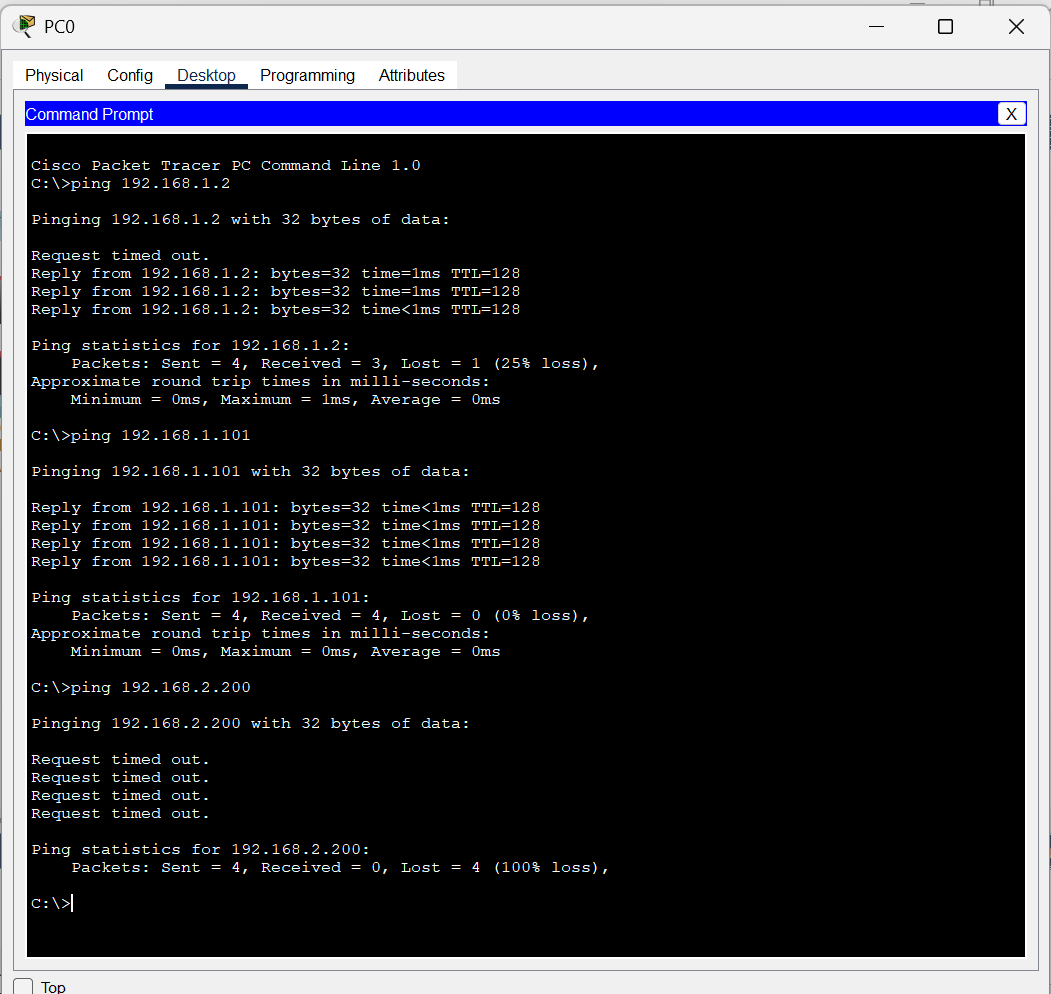
**Network diagram:**



**Configuration:**

|  |
| --- |
|  |

**Output:**

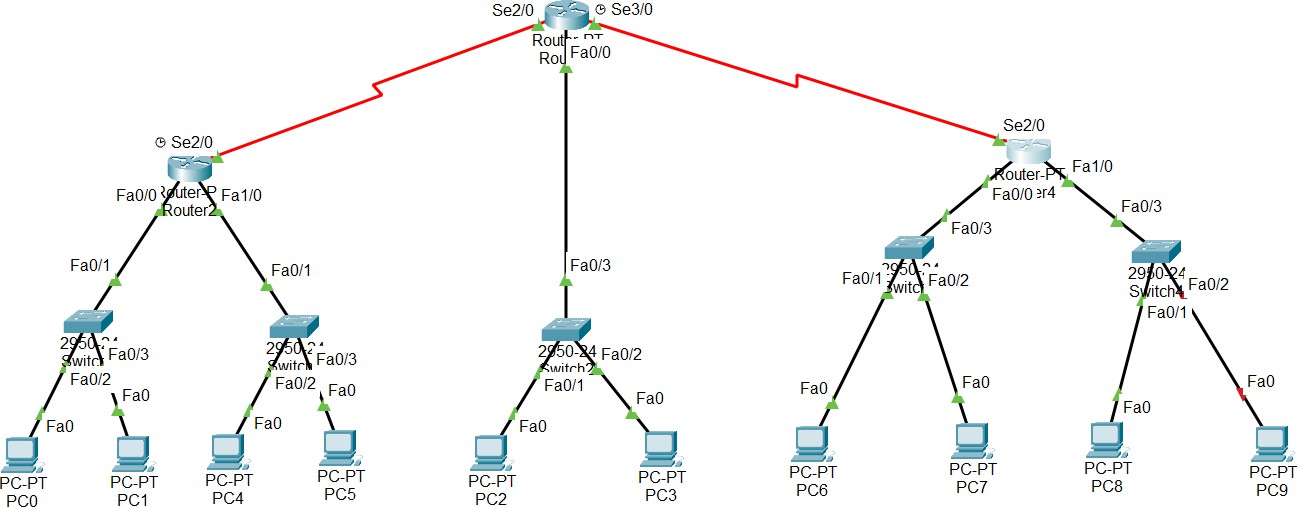
****

**Program 5:**

**Aim:** Configure default route, static route to the Router.

**Network diagram:**

**Configuration:**

****

|  |  |
| --- | --- |
|  |  |

**Output:**

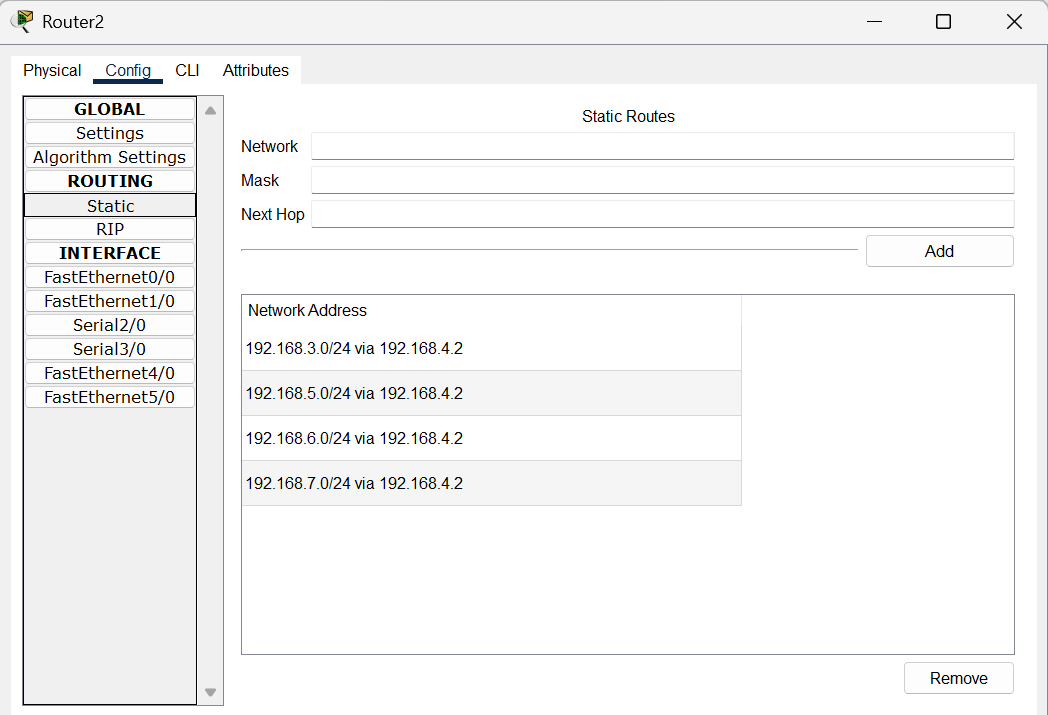
****

Fig 1. Router 2 – Static routing

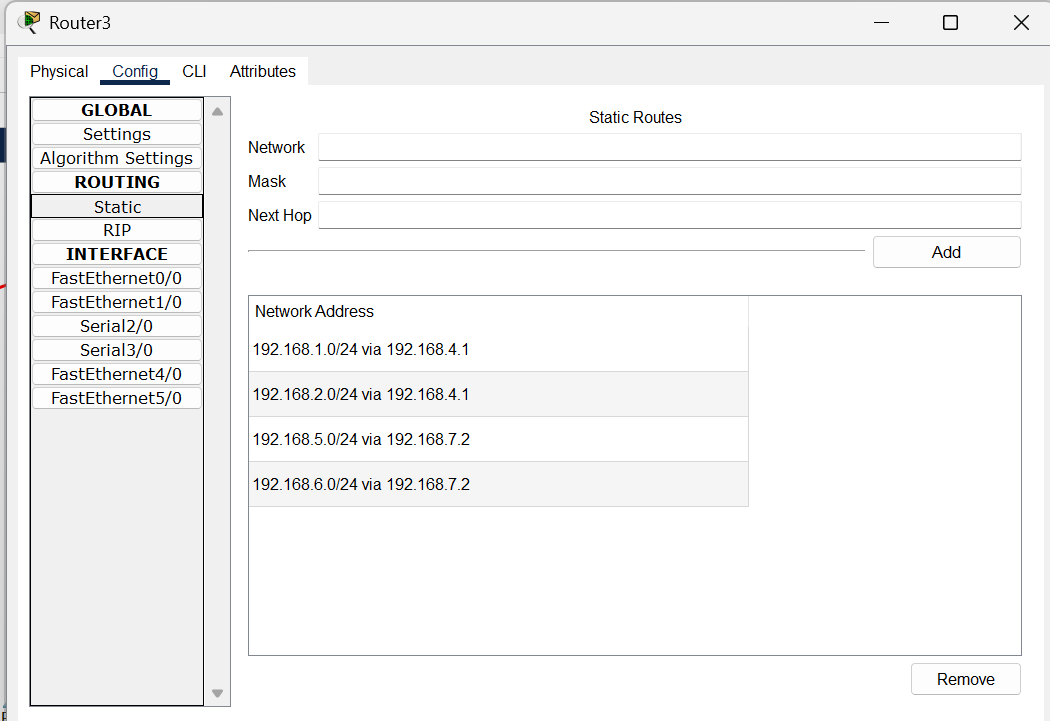


Fig 2. Router 3 – Static routing

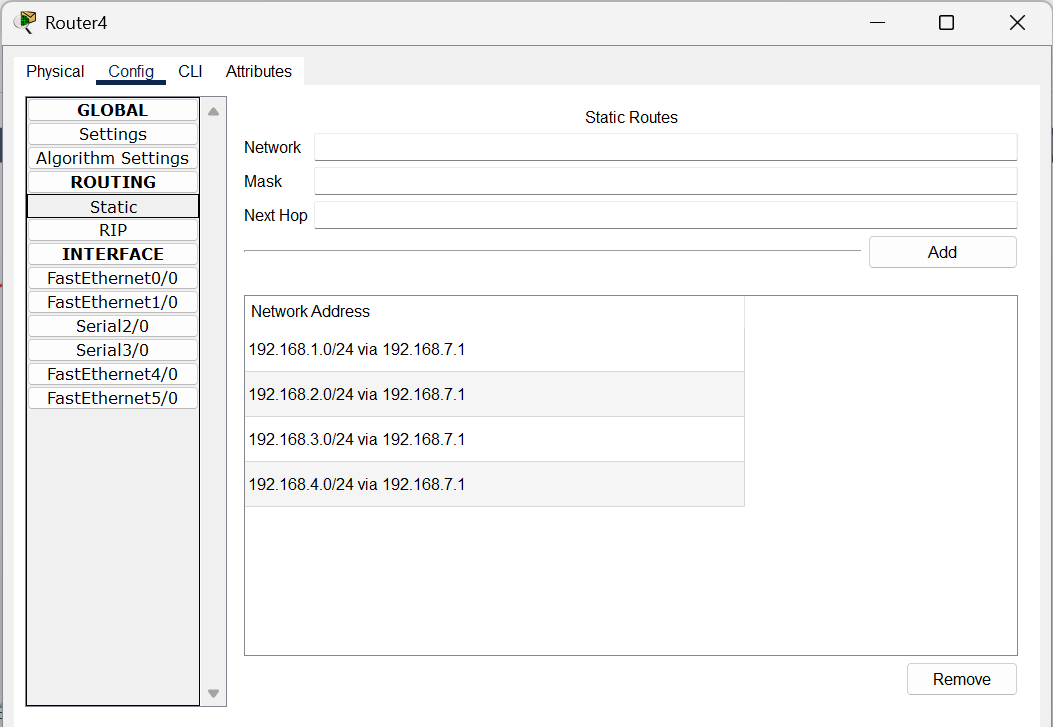
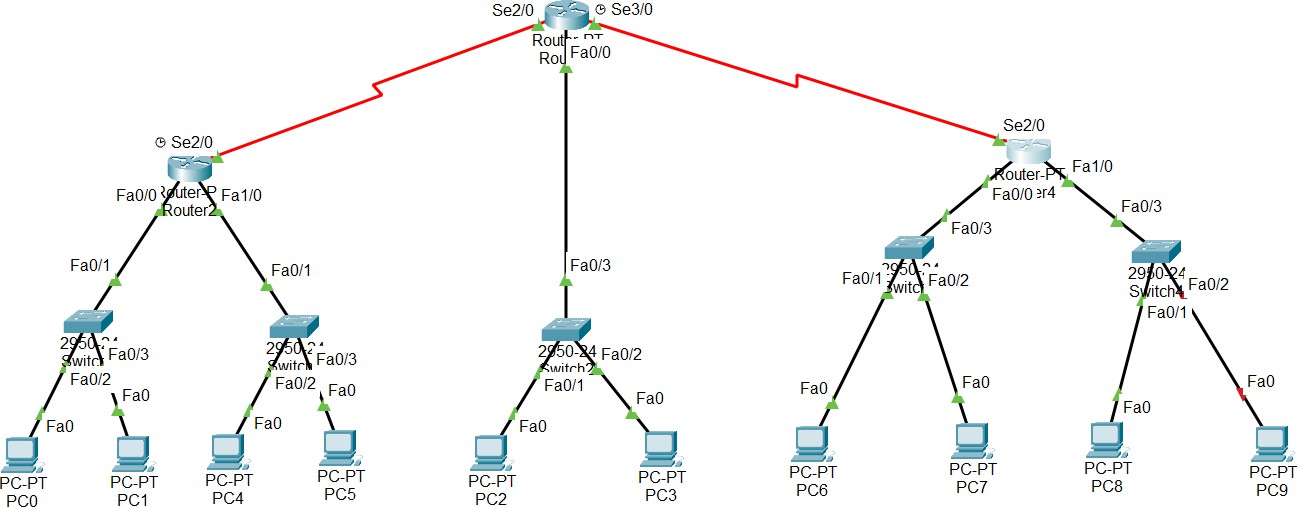


Fig 3. Router 4 – Static routing

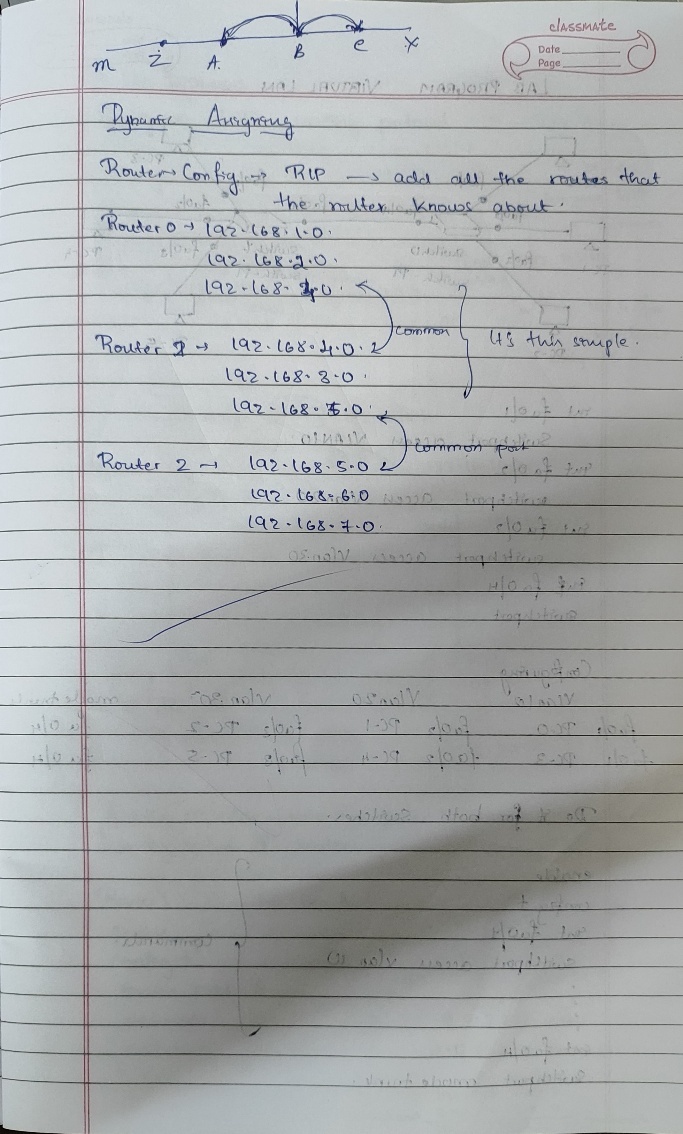
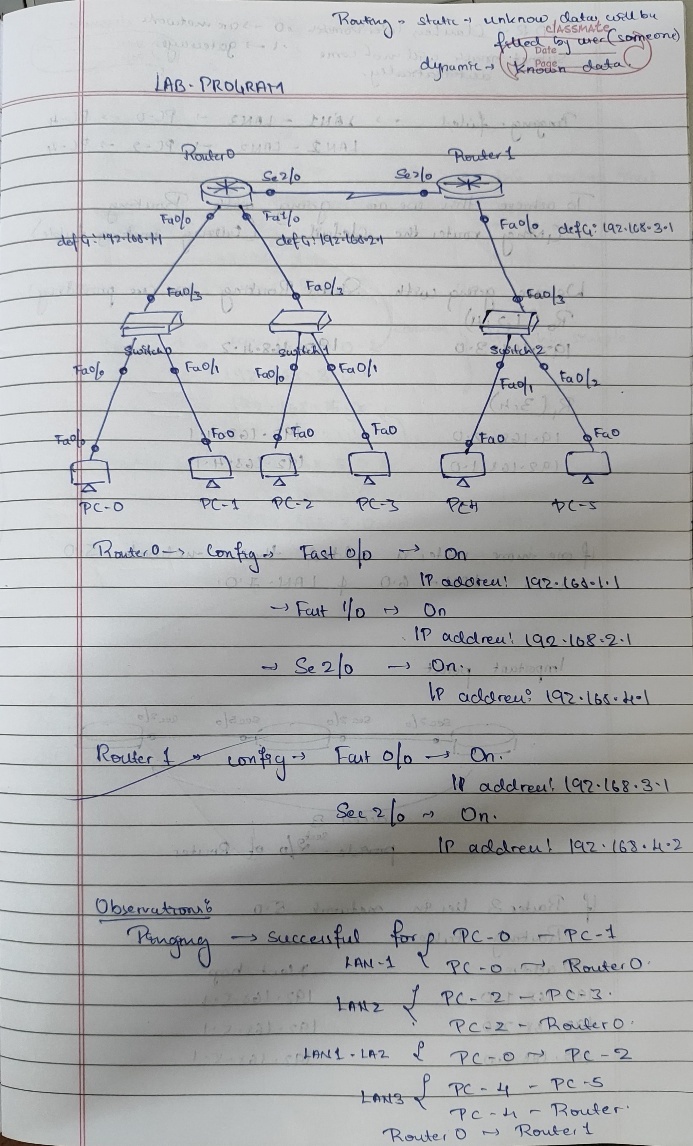
**Program 6:**

**Aim:** Configure RIP routing Protocol in Routers.

**Network diagram:**

****

**Configuration:**

****

**Output:**

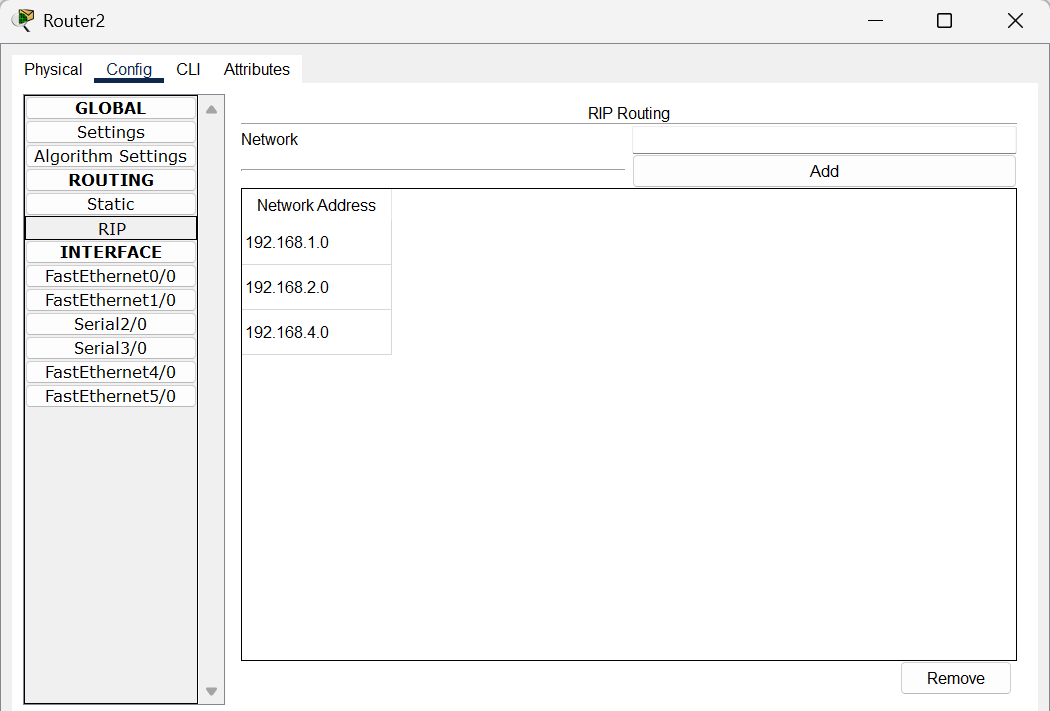
****

Fig 1. Router 2 – RIP routing

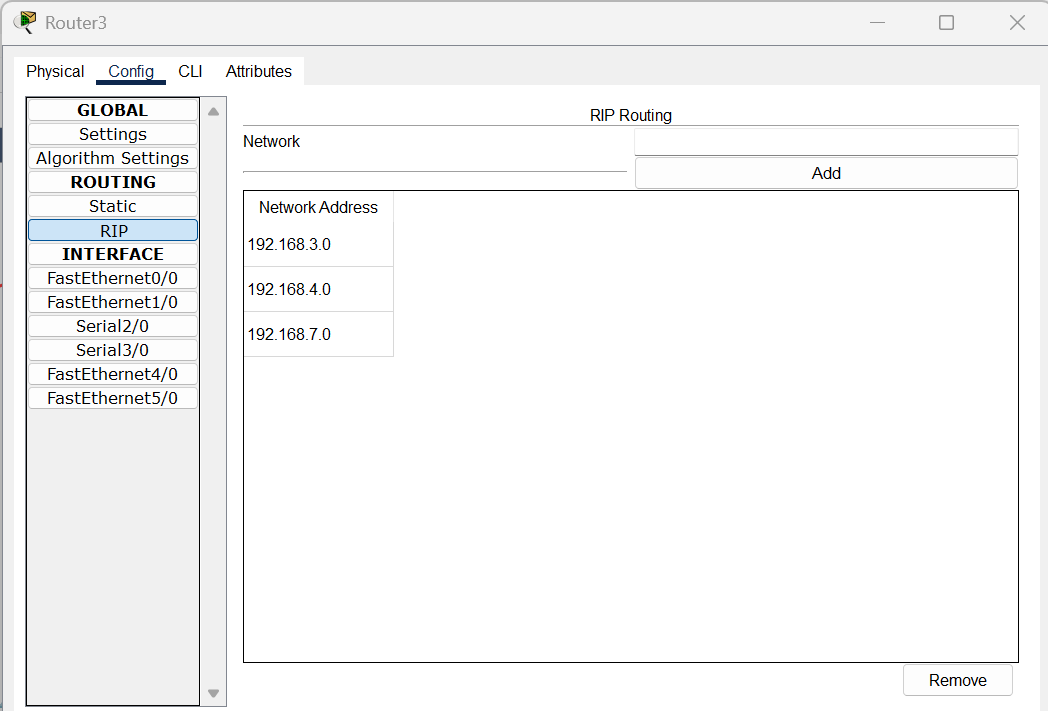


Fig 2. Router 3 – RIP routing

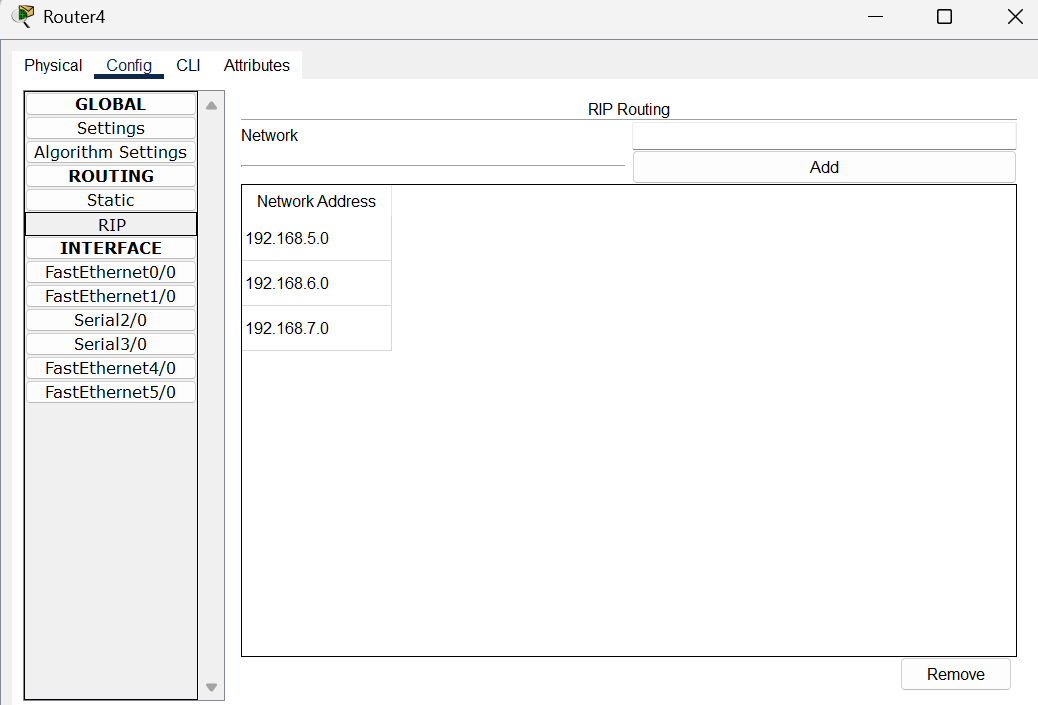
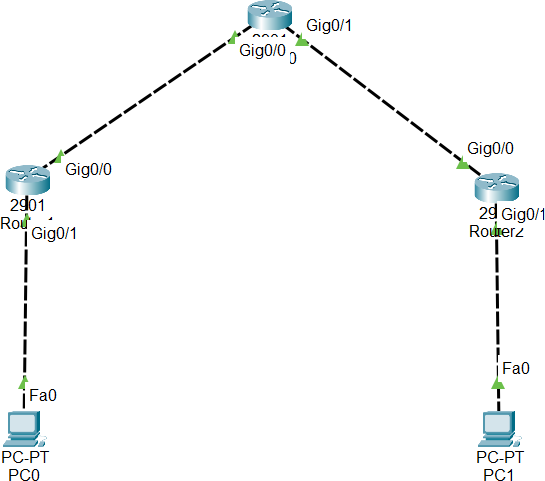


Fig 3. Router 4 – RIP routing

**Program 7:**

**Aim:** Configure OSPF routing protocol.

**Network diagram:**



**Configuration:**

|  |  |
| --- | --- |
|  |  |

**Output:**

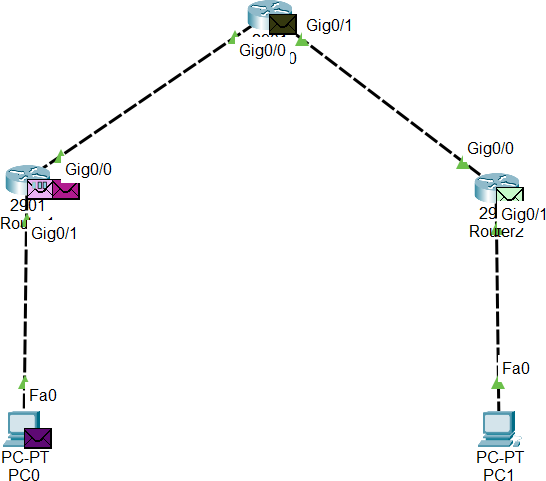
****

Fig 1. Sending PDU message from PC0 to PC1

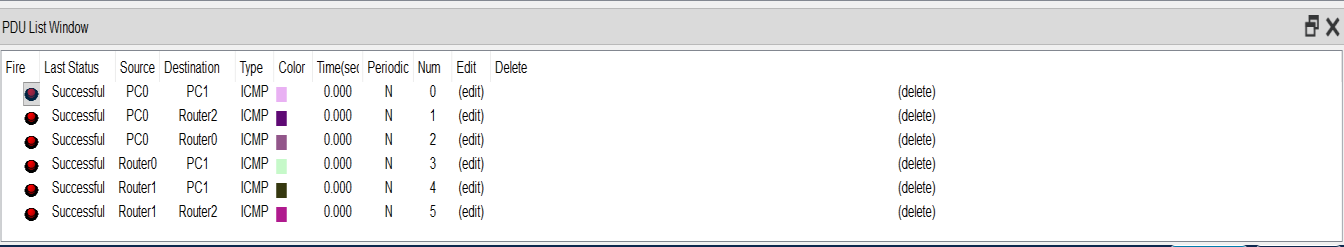
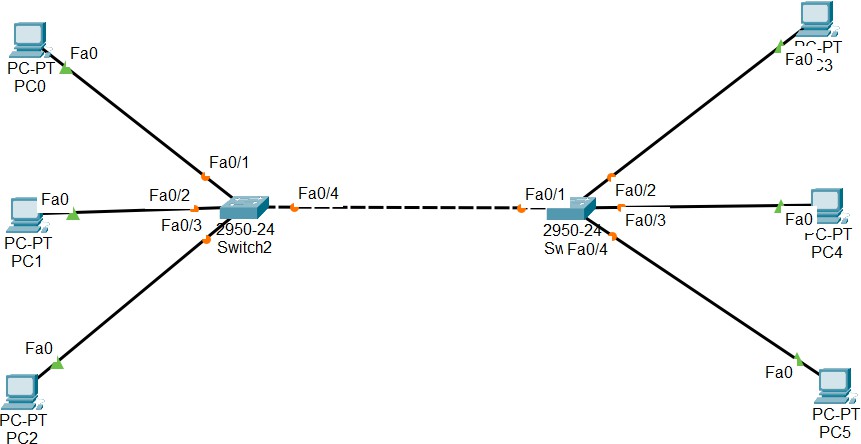


Fig 2. Checking PDU messages

**Program 8:**

**Aim:** To construct a VLAN and make the PC’s communicate among a VLAN.

**Network diagram:**



**Configuration:**

|  |  |
| --- | --- |
|  |  |

**Output:**

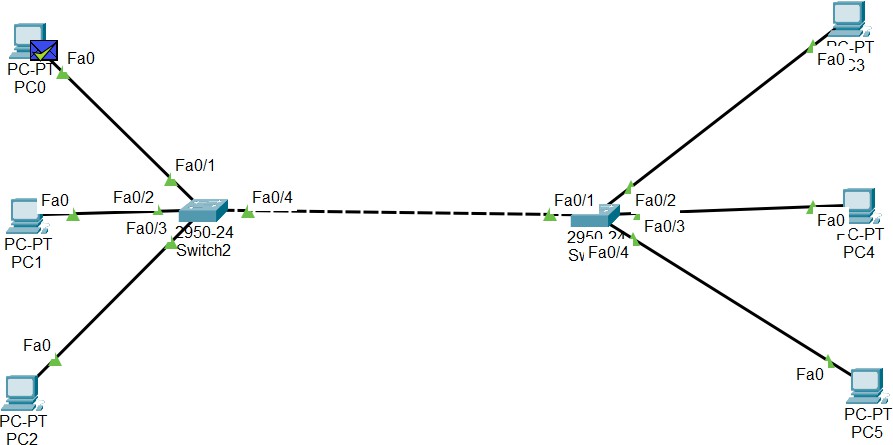
****

Fig 1. Sending PDU message from PC0 to PC5

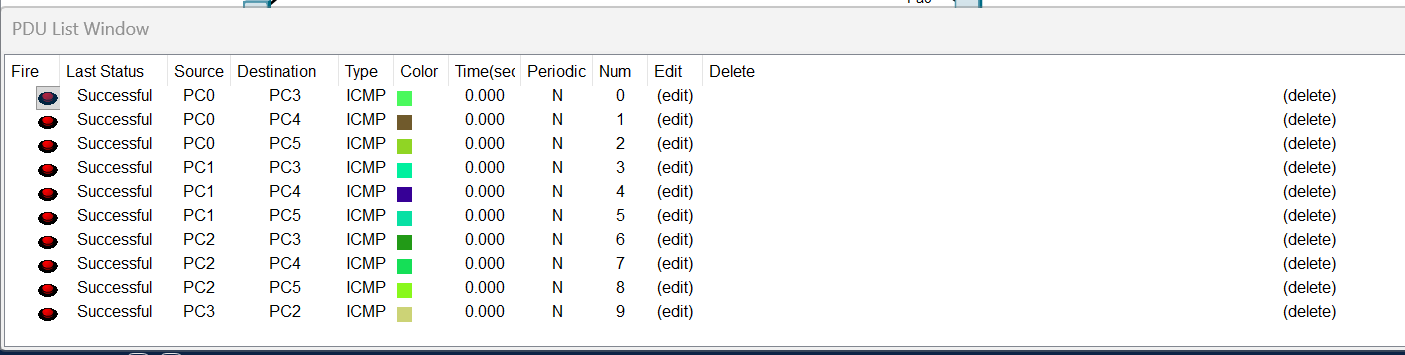
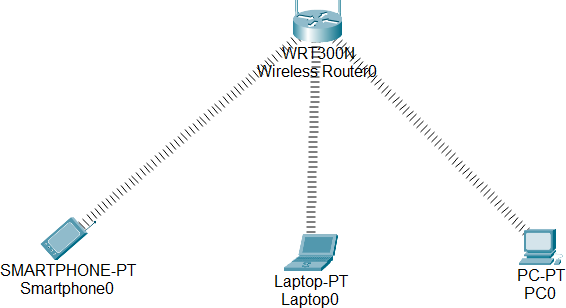


Fig 2. Checking PDU messages

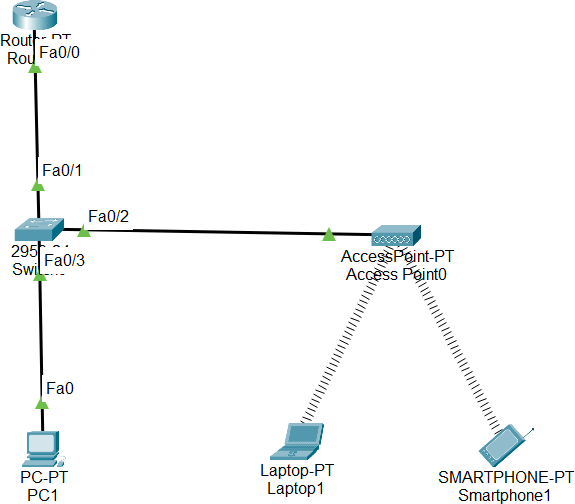
**Program 9:**

**Aim:** To construct a WLAN and make the nodes communicate wirelessly.

**Network diagram:**



Configuration 1



Configuration 2

**Configuration:**

|  |  |
| --- | --- |
|  |  |

**Output:**

Do Physical Connections In:

* Laptop
* PC

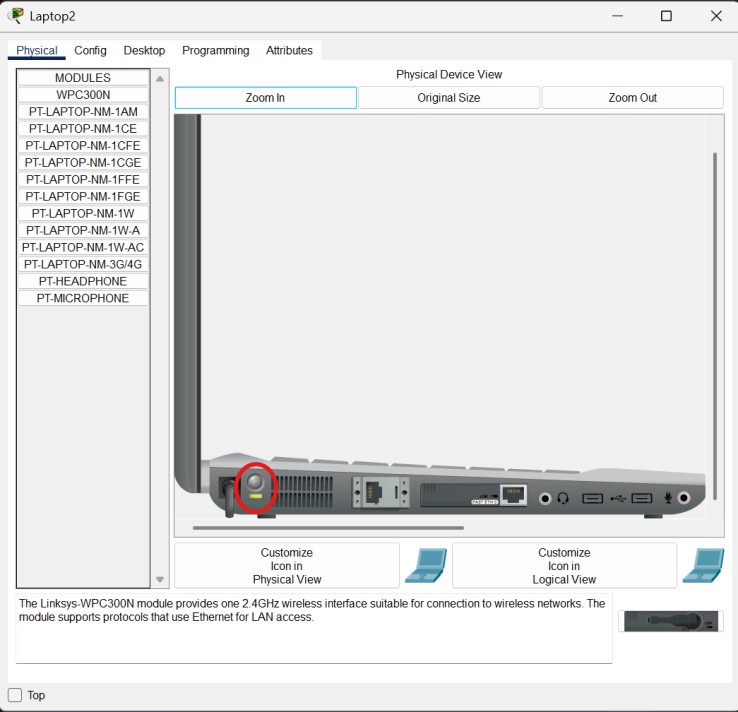


Fig 1.1 Step1: Turn off light / Power off laptop

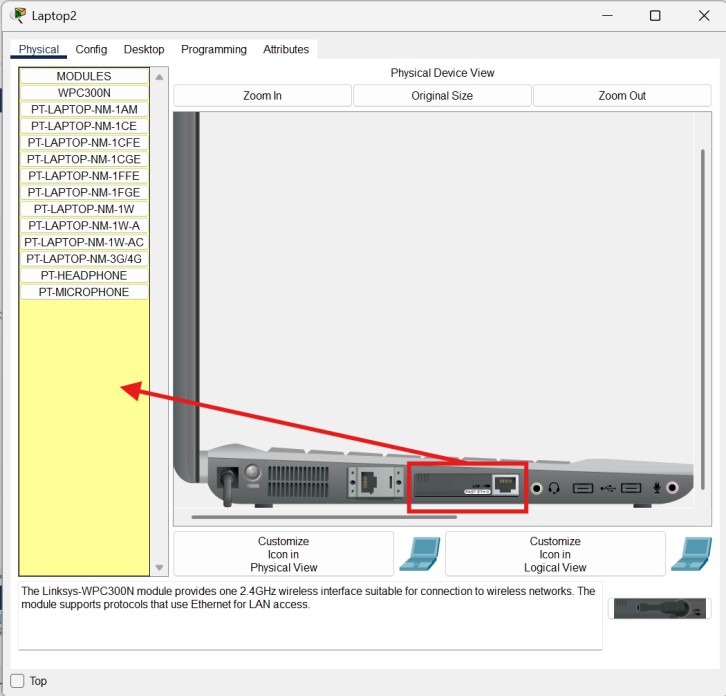


Fig 1.2 Step2: Drag and Drop the Ethernet into pointed location

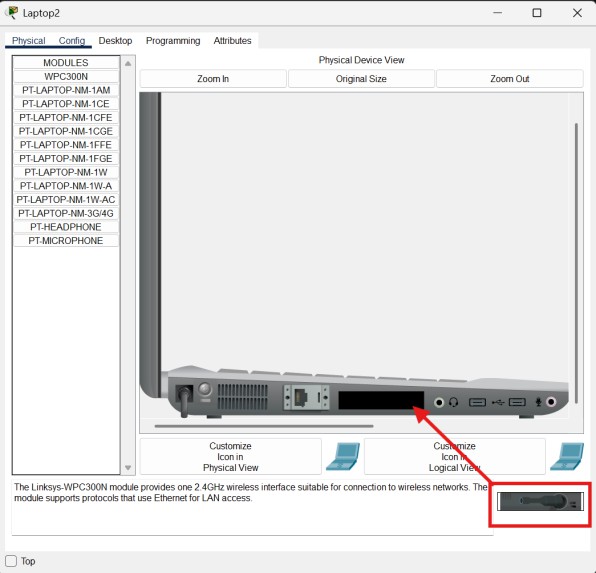


Fig 1.3 Step3: Drag and Drop the device into pointed location and Turn on light/Laptop

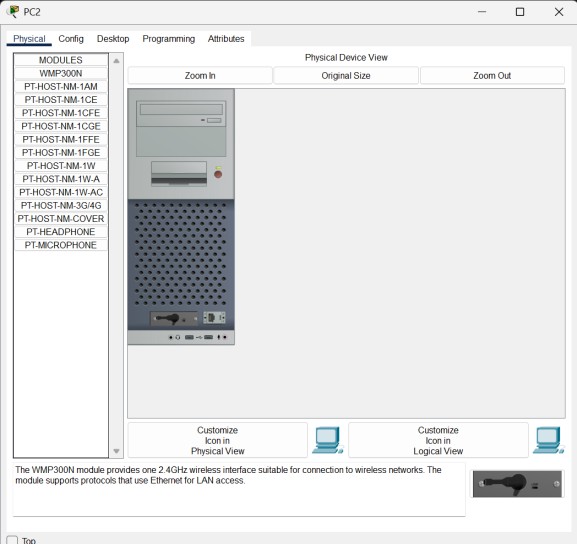


Fig 2. PC physical connection (combined 3 steps)

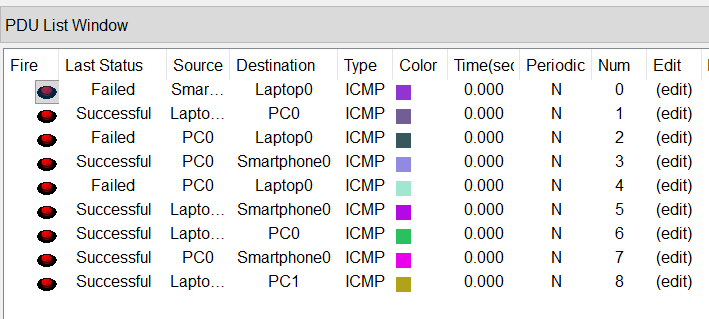


Fig 3. Checking PDU messages

**Program 10:**

**Aim:** Demonstrate the TTL/ Life of a Packet.

**Network diagram:**



**Configuration:**

|  |  |
| --- | --- |
|  |  |

**Output:**

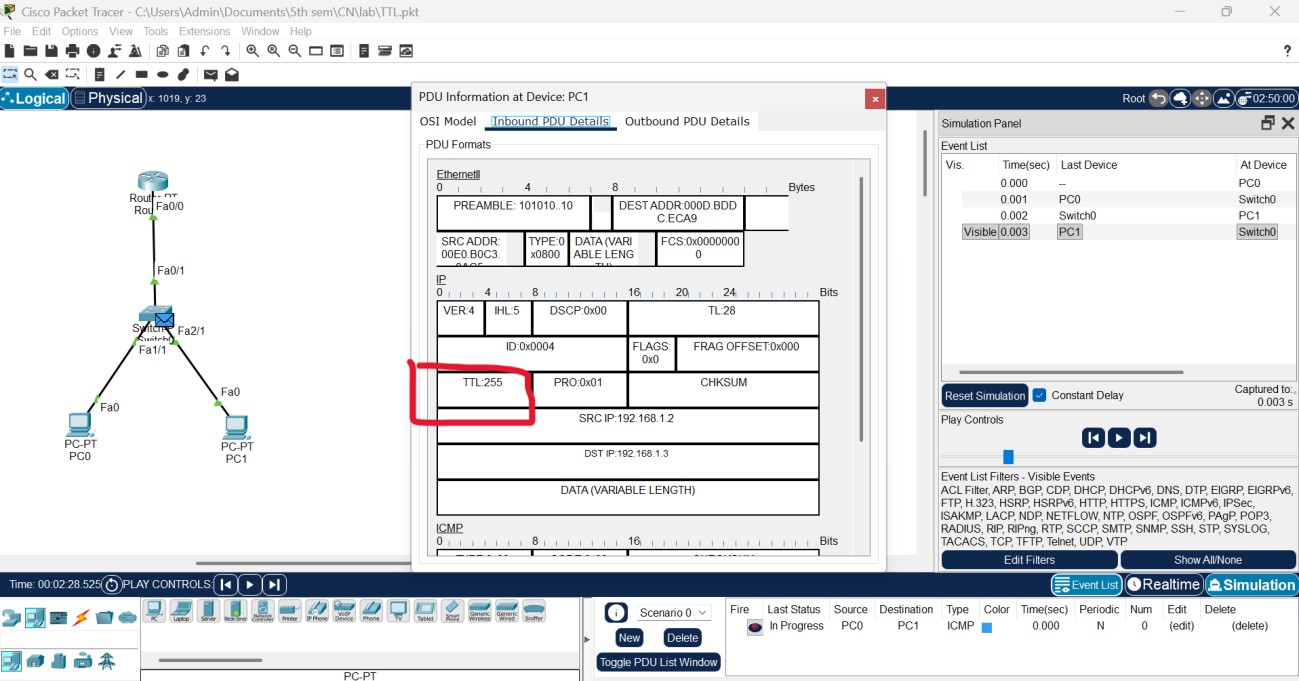
****

Fig 1. Inbound PDU Details at Device PC1

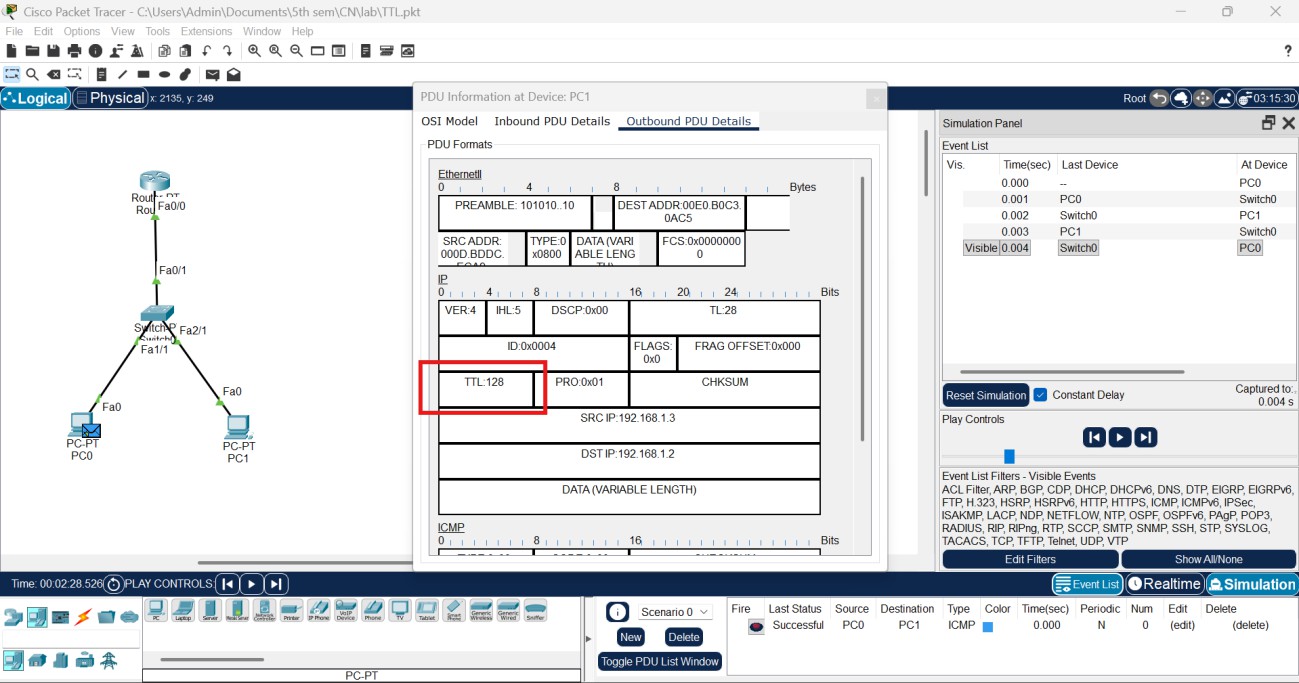
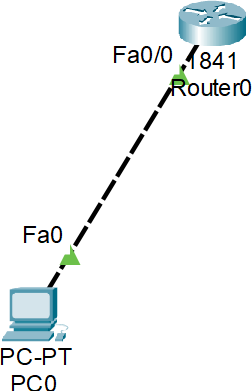


Fig 1. Outbound PDU Details at Device PC1

**Program 11:**

**Aim:** To understand the operation of TELNET by accessing the router in server room from a PC in IT office.

**Network diagram:**



**Configuration:**

|  |  |
| --- | --- |
|  |  |

**Output:**

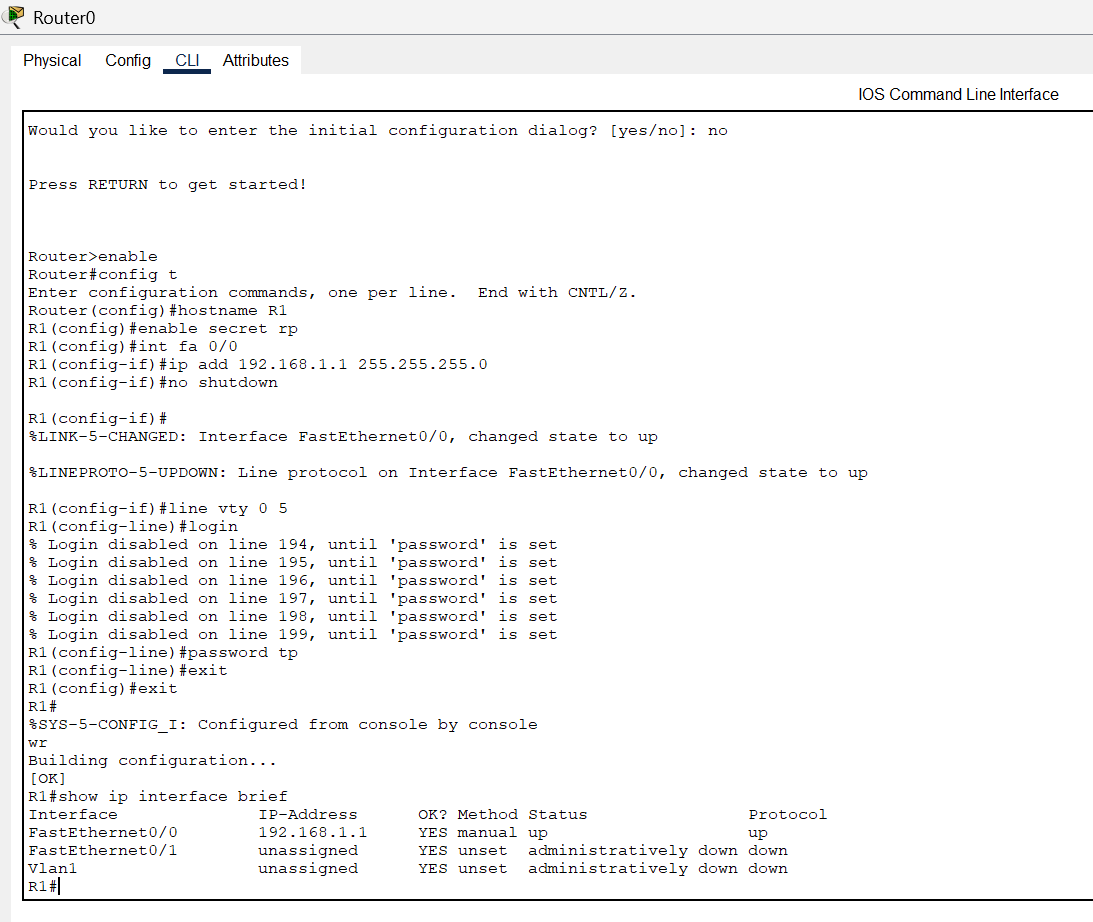
****

Fig 1. Router0 – CLI commands

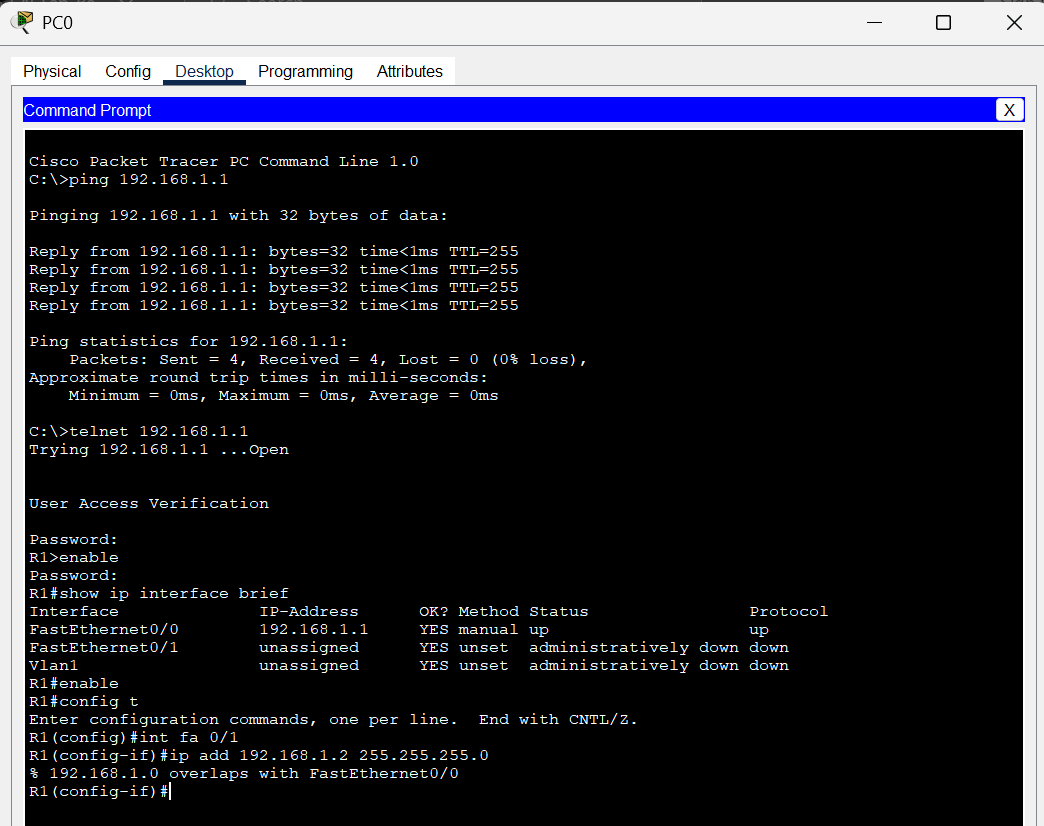


Fig2. PC command line prompt

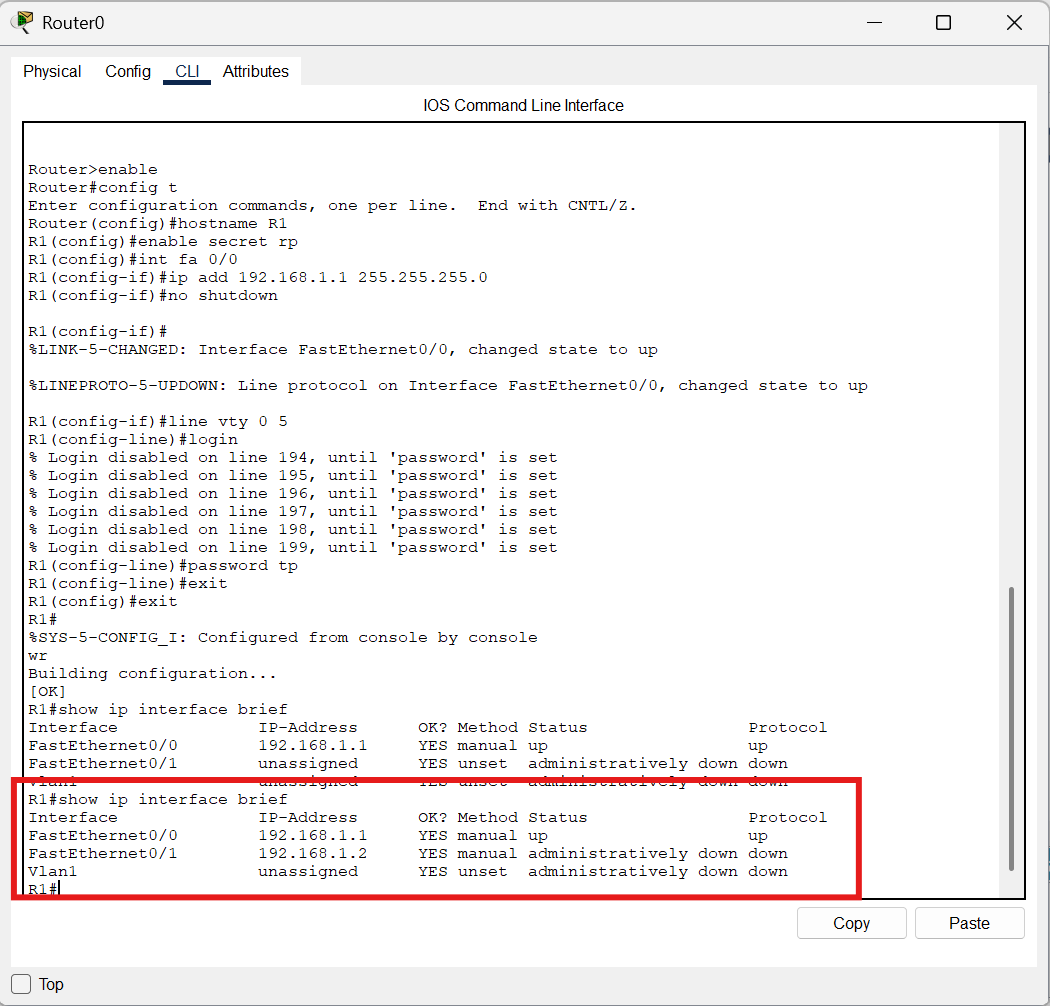


Fig 3. Updated the changes into Router0

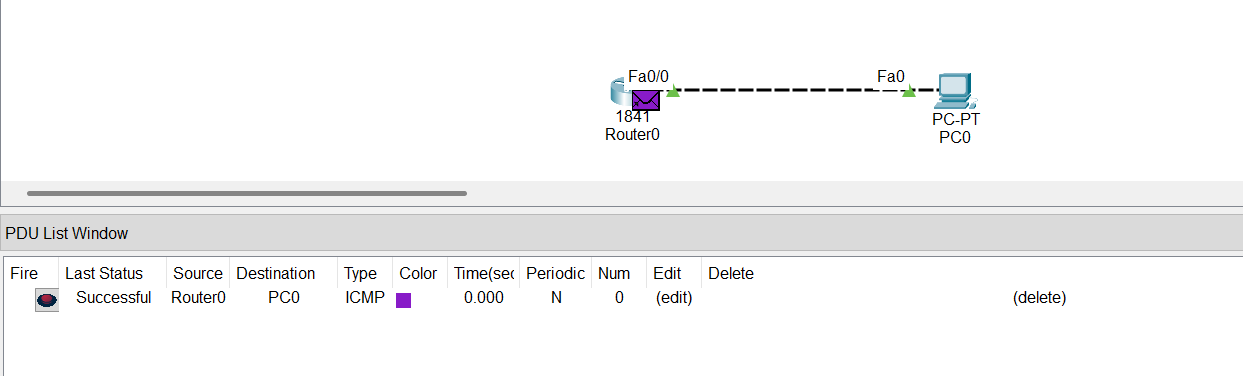
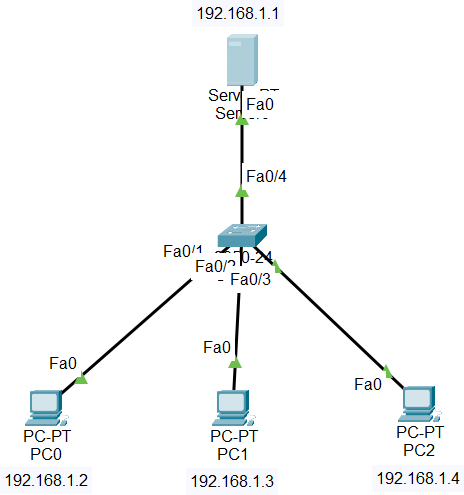


Fig 4. PDU message Successful

**Program 12:**

**Aim:** To construct simple LAN and understand the concept and operation of Address Resolution Protocol (ARP).

**Network diagram:**



**Configuration:**

|  |
| --- |
|  |

**Output:**

|  |  |
| --- | --- |
| Fig 1.1 ARP table at Server0 | Fig 1.2 Command Prompt at Server0 |
| Fig 2.1 ARP table at PC0 | Fig 2.2 Command Prompt at PC0 |
| Fig 3.1 ARP table at PC1 | Fig 3.2 Command Prompt at PC1 |

|  |  |
| --- | --- |
| Fig 4.1 ARP table at PC2 | Fig 4.2 Command Prompt at PC2 |

# PART - B

**Program 1:**

**Aim:** Write a program for congestion control using Leaky bucket algorithm.

**Code:**

#include <stdio.h>

int min(int x, int y) { if (x < y)

return x; else

return y;

}

int main() {

int drop = 0, mini, nsec, cap, count = 0, i, inp[25], process;

printf("Enter the bucket size:\n"); scanf("%d", &cap);

printf("Enter the processing rate:\n"); scanf("%d", &process);

printf("Enter the number of seconds you want to simulate:\n");

scanf("%d", &nsec);

for (i = 0; i < nsec; i++) {

printf("Enter the size of the packet entering at %d sec:\n", i + 1);

scanf("%d", &inp[i]);

}

printf("\nSecond | Packet Received | Packet Sent | Packet Left | Dropped\n");

printf("

\n");

for (i = 0; i < nsec; i++) { count += inp[i];

if (count > cap) {

drop = count - cap; count = cap;

}

printf("%d\t %d\t\t", i + 1, inp[i]);

mini = min(count, process); printf("%d\t\t", mini);

count = count - mini; printf("%d\t\t %d\n", count, drop);

drop = 0;

}

// Remaining packets after time ends for (; count != 0; i++) {

if (count > cap) {

drop = count - cap; count = cap;

}

printf("%d\t 0\t\t", i + 1);

mini = min(count, process); printf("%d\t\t", mini);

count = count - mini; printf("%d\t\t %d\n", count, drop);

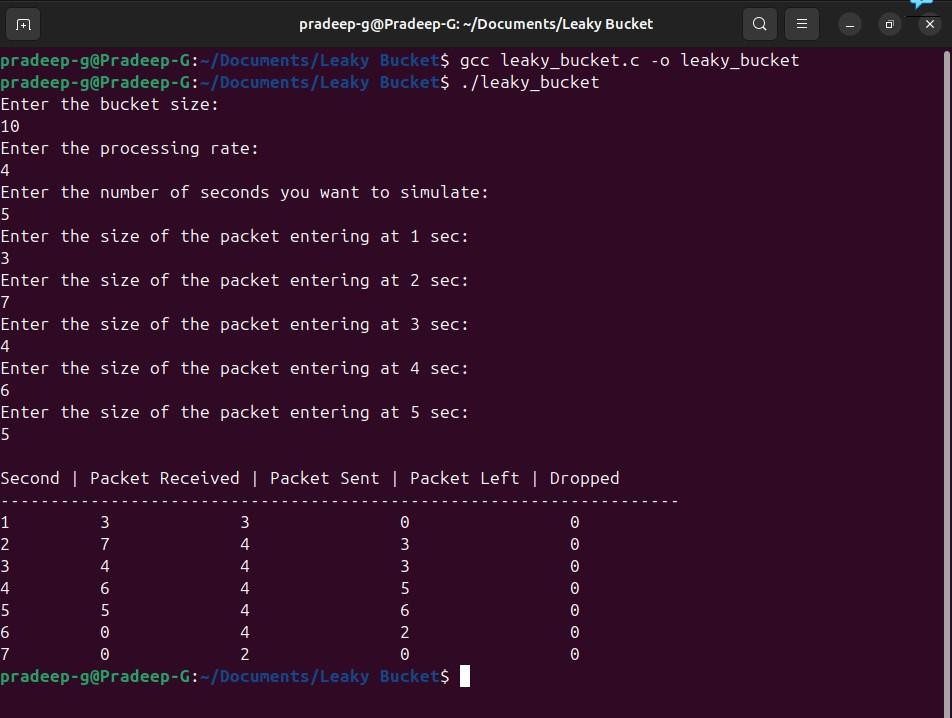
drop = 0;

}

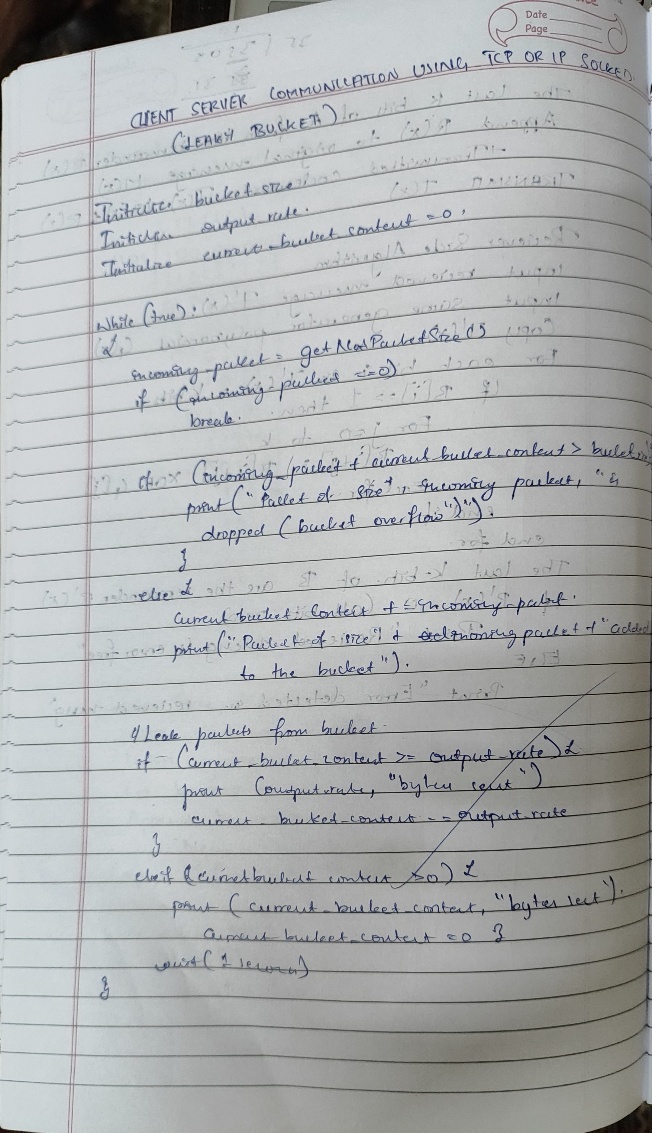
return 0;

}

**Output:**

****

**Observation:**

****

**Program 2:**

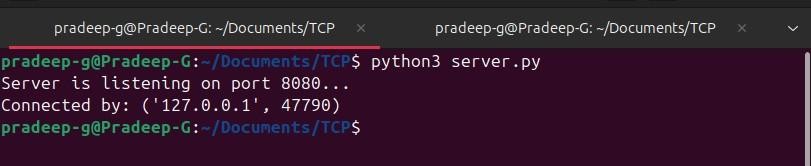
**Aim:** Using TCP/IP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

**Code:**

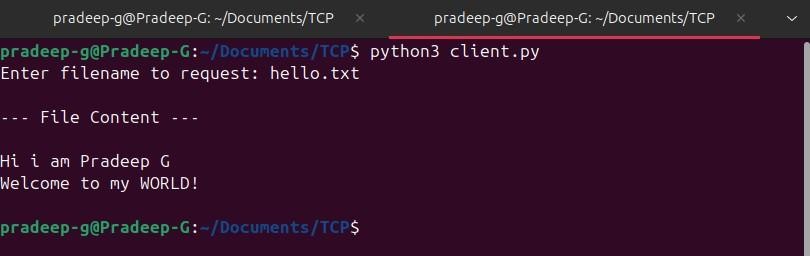
|  |  |
| --- | --- |
| # tcp\_client.py  import socket  # Step 1: Create TCP socket  client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # Step 2: Connect to server  client\_socket.connect(('localhost', 8080))  # Step 3: Send filename  filename = input("Enter filename to request: ")  client\_socket.send(filename.encode())  # Step 4: Receive file contents  data = client\_socket.recv(4096).decode()  print("\n--- File Content ---\n") print(data)  # Step 5: Close connection client\_socket.close() | # tcp\_server.py import socket  # Step 1: Create a TCP socket server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # Step 2: Bind to address and port server\_socket.bind(('localhost', 8080))  # Step 3: Listen for client connections server\_socket.listen(1) print("Server is listening on port 8080...")  # Step 4: Accept connection  conn, addr = server\_socket.accept() print("Connected by:", addr)  # Step 5: Receive file name filename = conn.recv(1024).decode().strip()  try:  # Step 6: Open and read file with open(filename, 'r') as f:  data = f.read()  conn.send(data.encode()) # Send file contents  except FileNotFoundError: conn.send(b"File not found on  server.")  # Step 7: Close connection conn.close() server\_socket.close() |

**Output:**

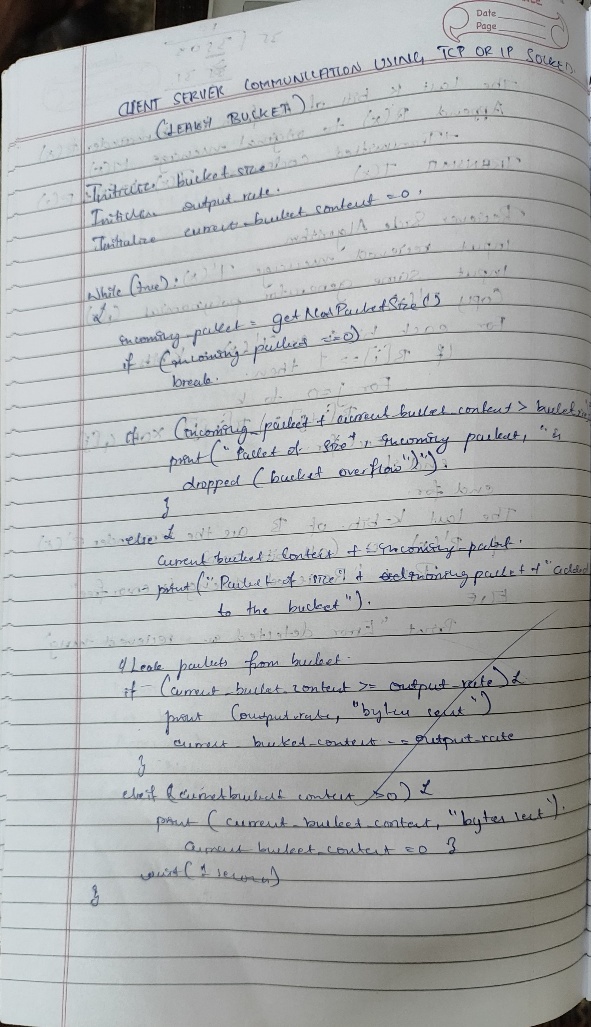
Server side Terminal:



Client side Terminal:



**Observation:**

****

**Program 3**:

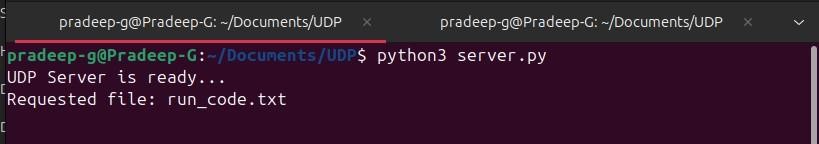
**Aim:** Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

**Code:**

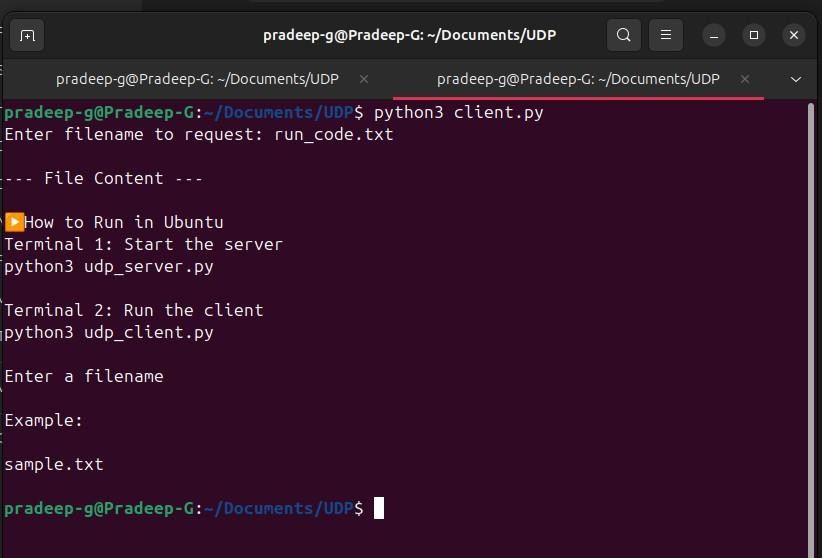
|  |  |
| --- | --- |
| # udp\_client.py  import socket  # Step 1: Create UDP socket  client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  server\_address = ('localhost', 8081)  filename = input("Enter filename to request: ")  # Step 2: Send filename to server  client\_socket.sendto(filename.en code(), server\_address)  # Step 3: Receive response  data, addr = client\_socket.recvfrom(4096)  print("\n--- File Content ---  \n") print(data.decode())  # Step 4: Close socket client\_socket.close() | # udp\_server.py import socket  # Step 1: Create UDP socket server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  # Step 2: Bind to address and port server\_socket.bind(('localhost', 8081))  print("UDP Server is ready...") while True:  # Step 3: Receive filename  from client  filename, addr = server\_socket.recvfrom(1024)  filename = filename.decode().strip()  print(f"Requested file:  {filename}")  try:  # Step 4: Open file and send content  with open(filename, 'r')  as f:  data = f.read()  server\_socket.sendto(data. encode(), addr)  except FileNotFoundError: server\_socket.sendto(b"Fil  e not found on server.", addr) |

**Output:**

Server side Terminal:



Client side Terminal:



**Observation:**

****

**Program 4:**

**Aim:** Write a program for error detecting code using CRC-CCITT (16-bits).

**Code:**

#include <stdio.h> #include <string.h> #include <stdlib.h>

int main() {

char rem[50], a[50], s[50], c, msj[50], gen[30]; int i, genlen, t, j, flag = 0, k, n;

printf("Enter the generation polynomial:\n"); gets(gen);

printf("Generator polynomial is CRC-CCITT: %s\n", gen);

genlen = strlen(gen); k = genlen - 1;

printf("Enter the message:\n"); n = 0;

while ((c = getchar()) != '\n') { msj[n] = c;

n++;

}

msj[n] = '\0';

for (i = 0; i < n; i++) a[i] = msj[i];

for (i = 0; i < k; i++) a[n + i] = '0';

a[n + k] = '\0';

printf("\nMessage polynomial appended with zeros:\n"); puts(a);

for (i = 0; i < n; i++) { if (a[i] == '1') {

t = i;

for (j = 0; j <= k; j++) { if (a[t] == gen[j])

a[t] = '0';

else

t++;

}

}

}

a[t] = '1';

for (i = 0; i < k; i++) rem[i] = a[n + i];

rem[k] = '\0';

printf("Checksum (remainder):\n"); puts(rem);

printf("\nMessage with checksum appended:\n"); for (i = 0; i < n; i++)

a[i] = msj[i];

for (i = 0; i < k; i++) a[n + i] = rem[i];

a[n + k] = '\0'; puts(a);

n = 0;

printf("Enter the received message:\n"); while ((c = getchar()) != '\n') {

s[n] = c; n++;

}

s[n] = '\0';

for (i = 0; i < n; i++) { if (s[i] == '1') {

t = i;

for (j = 0; j <= k; j++, t++) { if (s[t] == gen[j])

s[t] = '0';

else

}

}

}

s[t] = '1';

for (i = 0; i < k; i++) rem[i] = s[n + i];

rem[k] = '\0';

for (i = 0; i < k; i++) {

if (rem[i] == '1') flag = 1;

}

if (flag == 0)

printf("Received polynomial is error-free \n"); else

printf("Received polynomial contains error \n");

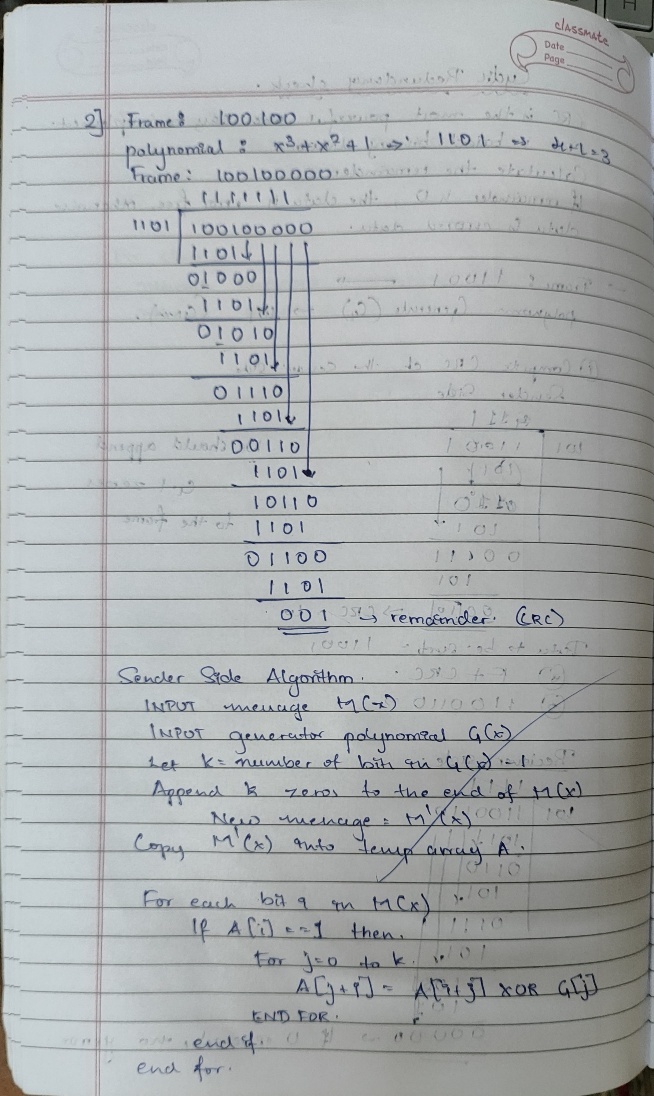
return 0;

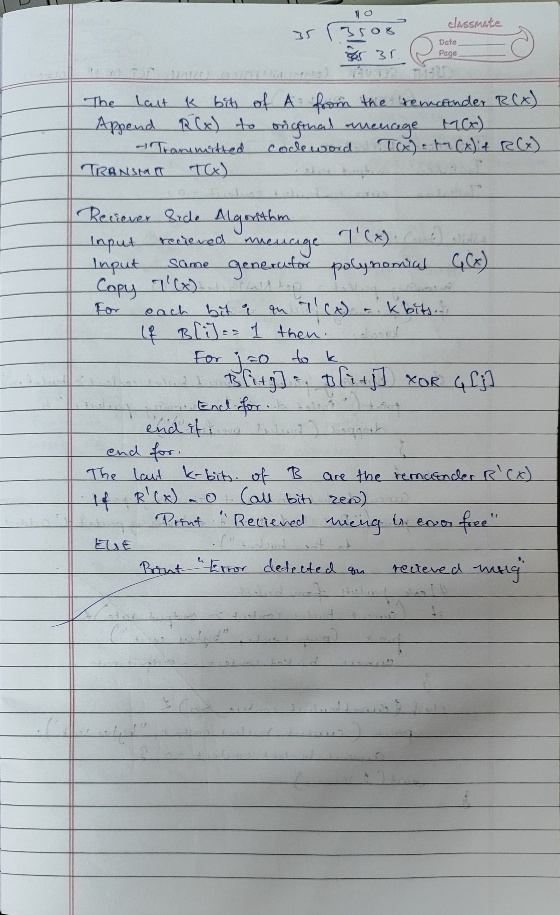
}

**Output:**

****

**Observation:**

****

****