# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

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

**LAB RECORD**

# Computer Network Lab (23CS5PCCON)

***Submitted by***

**Gayathri S (1BM24CS406)**

***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

## Academic Year 2024-25 (odd)

# B.M.S. College of Engineering

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Computer Science and Engineering

## <u>CERTIFICATE</u>

This is to certify that the Lab work entitled " Computer Network (23CS5PCCON)" carried out by **Gayathri S (1BM24CS406),** 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.

| | |
|---|---|
| Surabi S<br>Assistant Professor<br>Department of CSE, BMSCE | Dr. Selva Kumar<br>Professor & HOD<br>Department of CSE, BMSCE |

# Index


| Sl. No. | Date | Experiment Title | Page No. |
|---|---|---|---|
| 1 | 13-08-2025 | Simple PDU from source to destination using hub and switch as connecting devices. | 4-8 |
| 2 | 17-09-2025 | Default route and static route to the Router | 9-13 |
| 3 | 03-09-2025 | DHCP within a LAN and outside LAN | 14-15 |
| 4 | 10-09-2025 | Web Server, DNS within a LAN | 16-18 |
| 5 | 08-10-2025 | Operation of TELNET to access the router in server room from a PC in IT office | 19-21 |
| 6 | 19-11-2025 | RIP routing Protocol in Routers | 22-23 |
| 7 | 15-10-2025 | VLAN to make the PCs communicate among a VLAN | 24-25 |
| 8 | 15-10-2025 | WLAN to make the nodes communicate wirelessly | 26-27 |
| 9 | 19-11-2025 | Simple LAN to understand the concept and operation of ARP | 28-29 |
| 10 | 08-10-2025 | OSPF routing protocol | 30-31 |
| 11 | 15-10-2025 | TTL/ Life of a Packet | 32-33 |
| 12 | 15-10-2026 | Ping responses, destination unreachable, request timed out, reply | 34-36 |
| 13 | 29-10-2025 | Congestion control using Leaky bucket algorithm | 37-38 |
| 14 | 29-10-2025 | Error detecting code using CRC-CCITT | 39-40 |
| 15 | 12-11-2025 | TCP File Request–Response Using Client–Server Socket Program | 41-42 |
| 16 | 12-11-2025 | UDP File Request–Response Using Client–Server Socket Program | 43-44 |


GithubLink: https://github.com/GayathriS-CSE/CN-LAB

**CYCLE 1:**

**<u>Program 1</u>**

Aim of the program:

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

Procedure and topology:

Lab 1:- Simple Connection.

Aim:- Constructing a simple connection in cisco packet tracer.

Components:- 2 Switches → Swith-PT
3 Laptops → Laptop-PT
3 PC → PC-PT
Connections between all the compounts
→ Automatically choose connection type [Thunder.

Procedure:-

Step 1:- Select 3 laptop from End devices and drag and drop them on the logical screen.

Step 2:- Configure the laptops by setting the name as APC1. AP2 and APC3.
To change the name, click on laptop > config > Display name > edit.

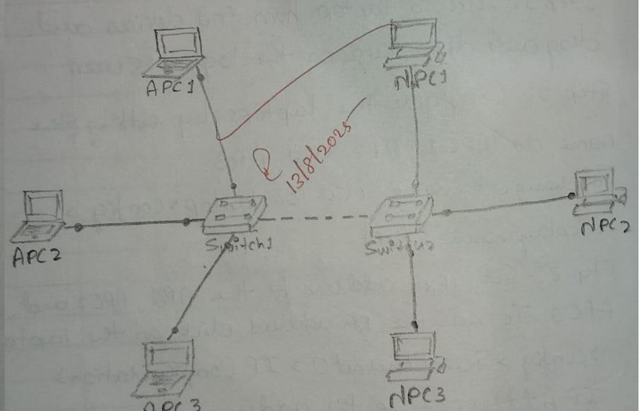
Step 3:- Set IPv4 address for the APC1, APC2, and APC3, To set the IP address, click on the laptop > config > FastEthernet0 > IP Configulation > IP Address, type the address.
Ex:- 192.162.10.1 for APC1. and similarly set ip address for remaining laptops
192.162.10.2 → APC2.
192.162.10.3 → APC3.

Step 4:- Drag and drop 3 more PC from End devices and follows the same steps 1- 3 for each ~~lapt~~

IP address.
NPC1 - 192.162.20.1
NPC2 → 192.162.20.2
NPC3 → 192.162.20.3

Step 5:- Drag and drop 2 Switches (generic) from Switches

Step 6:- Enable the connections between the components using thunder cable as in below diagram.




Step 7: Pass PDU from APC1 to APC2 &
APC3 to NPC3.

Observation:-
APC3 to NPC3 gets failed because two switches are two different networks

20.08.25

OSI Model 7 Application → Request content → Request content in required format.

6. Presentation layer → Encrypt Compress Traslate.

5. Session layer → Session of communication

4. Transport layer → Segmentation → Transport → Reassembly.

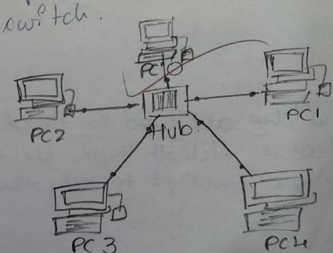
3. Network layer → Packet Creation → Routing → Packet assembly

2. Data link layer → Frame creation → local network Delivery → Transfer Frame between net

1. Physical layer.
↳ Cable. 0010100010

→ Create the topology and simulate simple PDU. From source to destination. Using hub and switch as connecting devices and demostrate the ping message.

→ Creating star topology. using hub and switch.




Star using Hub topology.

Observation:- The pinged message is tranfered to all the devices but acknowledged by Only the destination.

To ping a ~~ping~~ message.

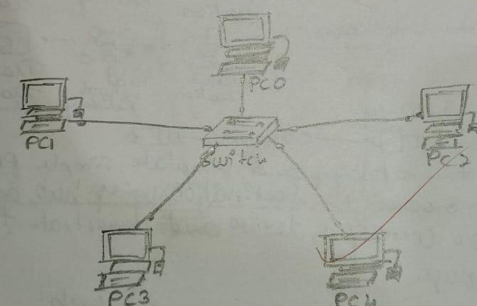
>device > Desktop > Command prompt

> ping <ip address of destination>.

To know about the ip address of own device.

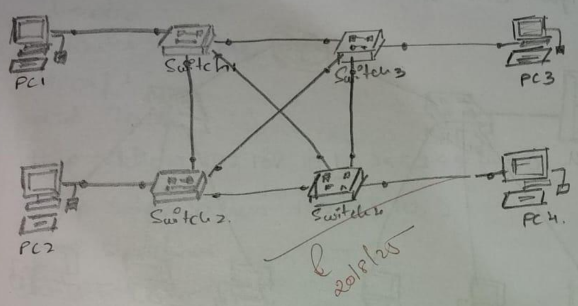
>device > Desktop > Command prompt

> ipconfig

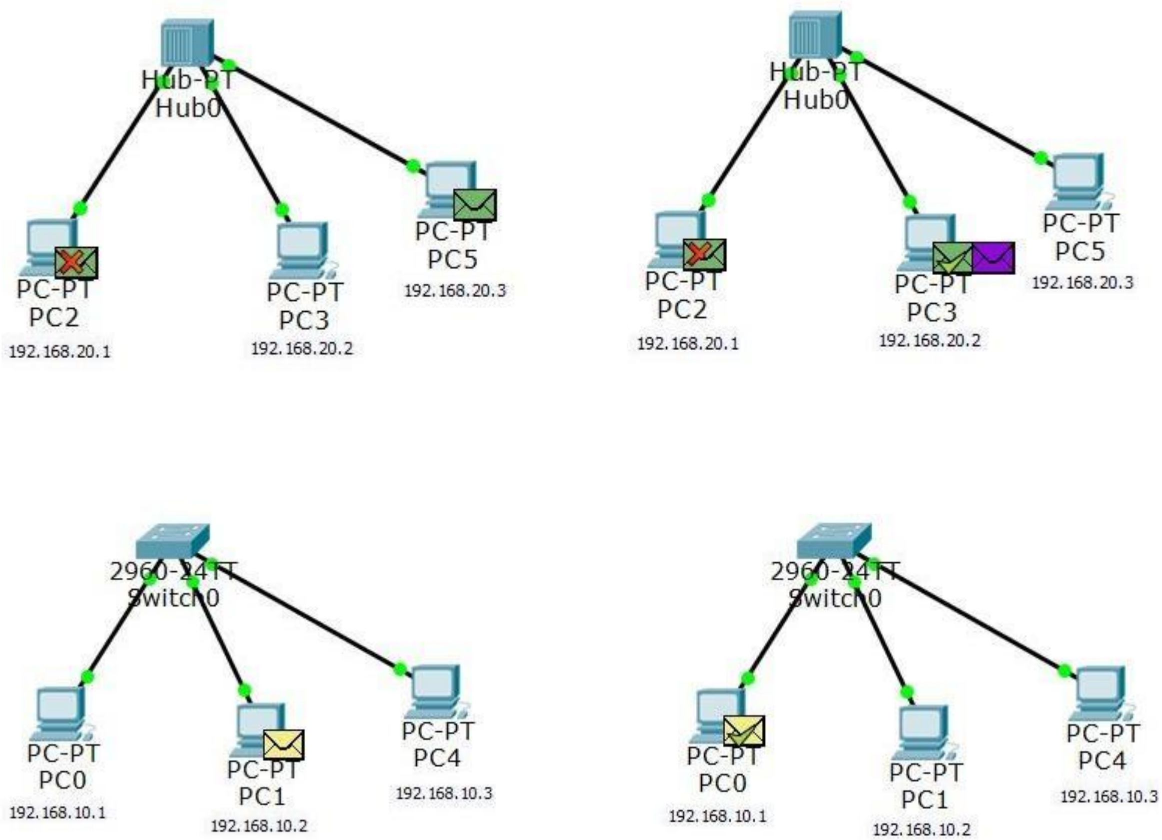



Observation: Switches are used in third layer of OSI model which is Network layer. Unlike the hub switches passes the message to only the destination device.

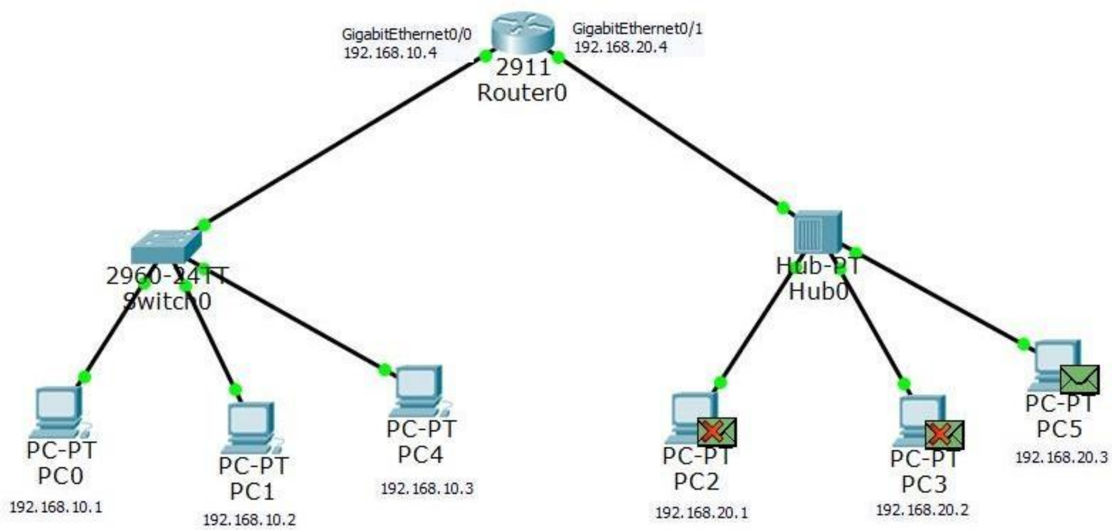
Mesh topology using switches.

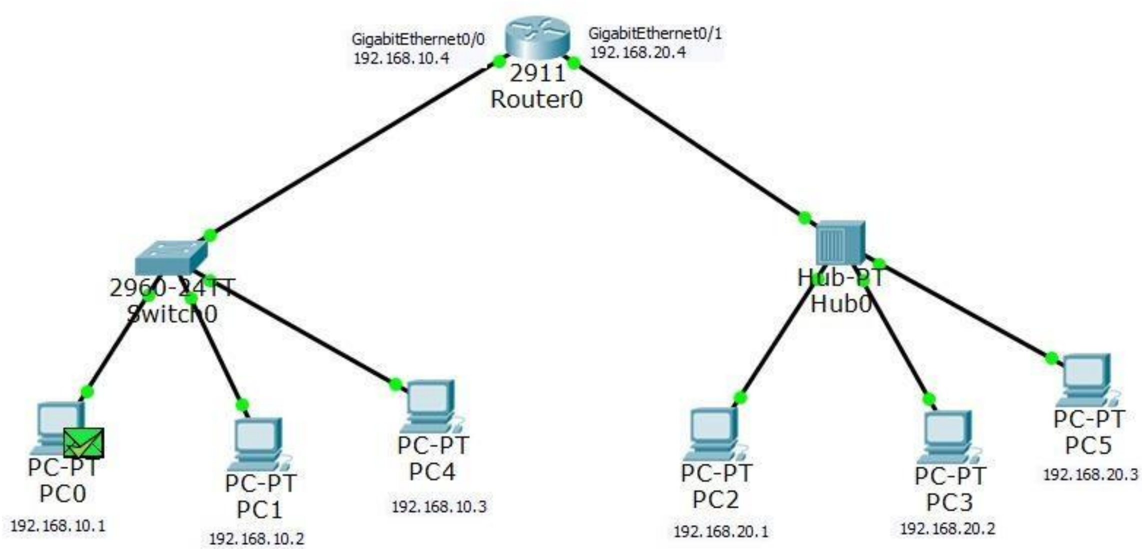


Screenshots/ Output:




Updated topology




Observation:

- In the hub-based topology, the PDU was broadcast to all ports, while the switch forwarded the PDU only to the destination MAC after learning addresses from incoming frames.
- Successful ICMP echo and echo-reply messages confirm that both devices enabled connectivity, with the switch demonstrating selective unicast forwarding and reduced unnecessary traffic.

**Program 2**

Aim of the program:

Configure default route, static route to the Router

Procedure and topology:

17.09.25.

=> Configure IPV4. Static and default router.

Router.1

—> enable

> conf t

> int Se0/1/0

> ip address 172.16.1.1 255.255.255.252

> no shut down

> exit

> int Fa0/0

> ip address 192.168.10.1 255.255.255.0

> no shutdown

Exit

> exit

write memory

10/9/25

Router 2

```
>enable
#conf t
(config)#int Se0/1/0
#ip address 172.16.1.2 255.255.255.252
#no shutdown
exit
#int Fa0/0
#ip address 192.168.20.1 255.255.255.0
#no shutdown
exit
#int Se0/1/1
#ip address 172.16.2.1 255.255.255.252
#no shutdown
#exit
#exit
write memory.
```

Router 3.

```
>enable
#conf t
#hostname R3.
#int Se0/1/0
#ip address 172.16.2.2 255.255.255.252
#no shutdown
exit
#int Fa0/1
#ip address 192.168.30.1 255.255.255.0
#no shutdown
exit
#exit
write memory.
```

PC0

```
>Desktop
>IP Configuration
IP address        192.168.10.10
Subnet mask       255.255.255.0
Default Gateway   192.168.10.1
```

PC1

```
>Desktop
>IP Configuration
IP address        192.168.20.10
Subnet mask       255.255.255.0
Default Gateway.  192.168.20.1
```

PC2.

```
>Desktop
>IP Configuration
IP address.       192.168.30.10
Subnet mask.      255.255.255.0
Default Gateway.  192.168.30.1
```

→ IP Routing

R1

```
>enable
#conf t.
#ip route 192.168.20.0 255.255.255.0 172.16.2.2
#ip route 172.16.2.0 255.255.255.0 172.16.1.2
#ip route 192.168.30.0 255.255.255.0 172.16.1.2
#exit
write memory
>enable
```

#show ip route.

R2

>enable

#conf t

#ip route 192.168.10.0 255.255.255.0 172.16.1.1

#ip route 192.168.30.0 255.255.255.0 172.16.2.2.

#exit

write memory

#show ip route.

R3 -> dynamic

>enable

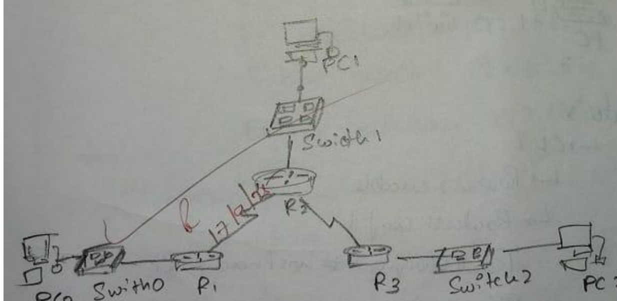
#conf t

#ip route 0.0.0.0 0.0.0.0 Se0/1/0.

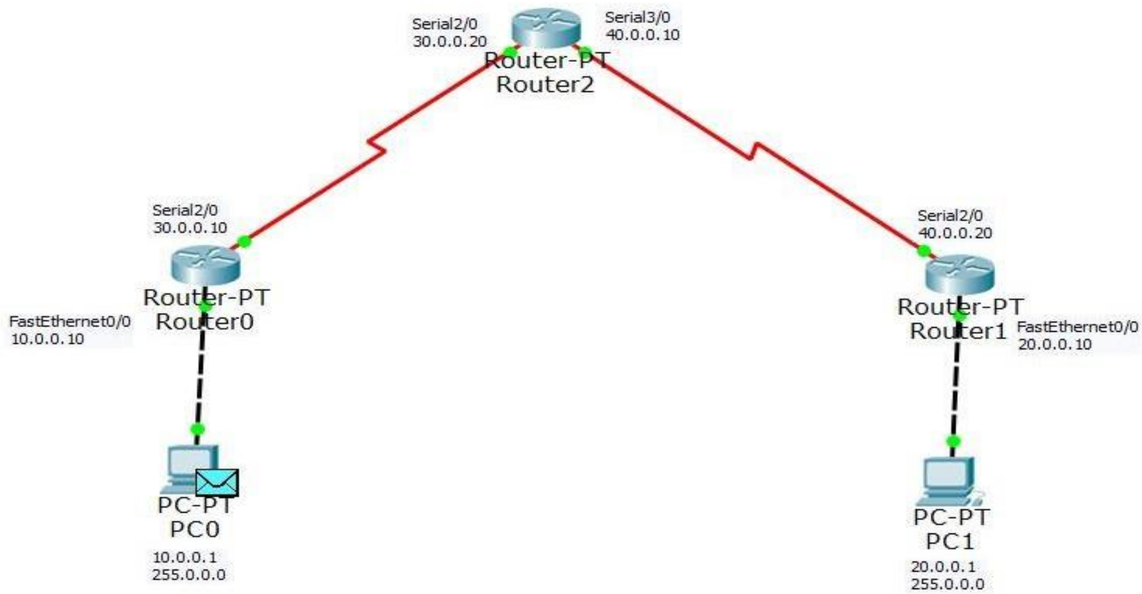
#exit

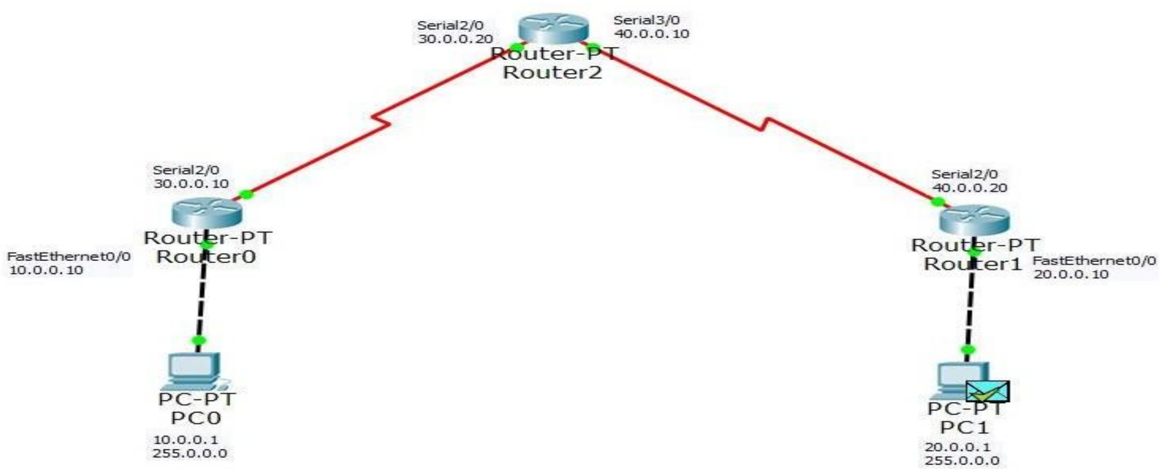
write memory

#show ip route.

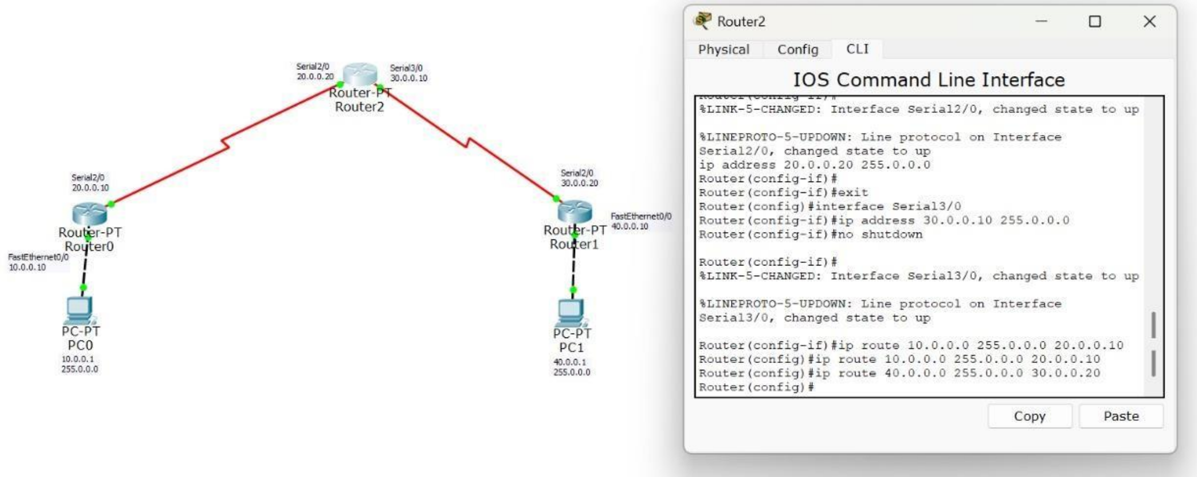


Screenshots/ Output:

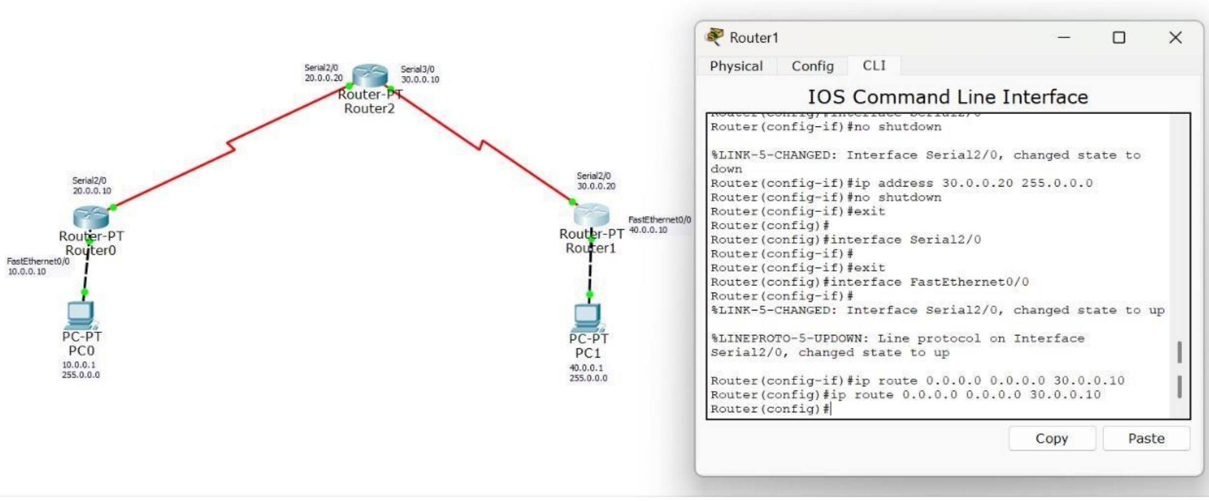



Static routing CLI commands:




Default routing CLI commands:




Observation:

- The configured static and default routes correctly updated the router's routing table, enabling deterministic next-hop selection for remote networks.
- Successful ping tests verified that traffic was forwarded according to the static/default route entries, ensuring end-to-end reachability across different network segments.

**Program 3**

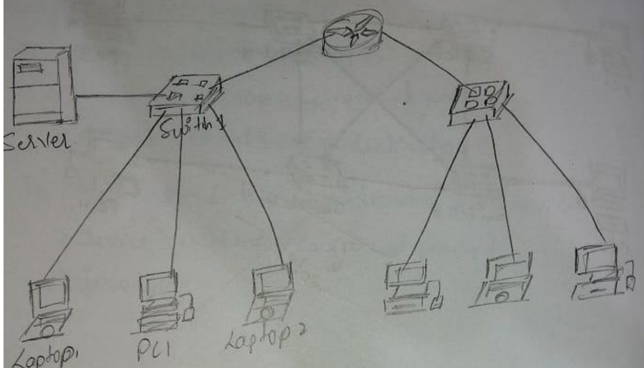
Aim of the program:

Configure DHCP within a LAN and outside LAN.

Procedure and topology:

.09.25

Configure DHCP within a lab and outside a lab.




Server

① DHCP
  ↳ Desktop IP Config
    ↳ static IP
      ↳ 192.168.10.2
    ↳ Gateway
      ↳ 192.168.10.1 ↳ IP of router

② Services.
  ↳ DHCP

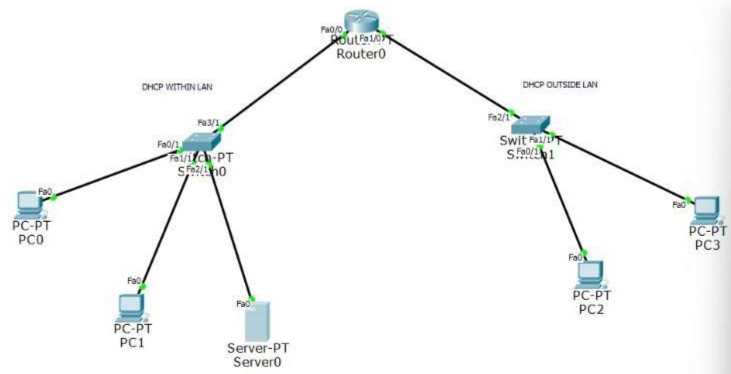
| | ON | ON |
|---|---|---|
| ↳ Pool Name | SwitchOne | SwitchTwo |
| Gateway | 192.168.10.1 | 192.168.20.1 |
| Static IP | 192.168.10.2 | 192.168.20.2 |
| Subnet mask | 255.255.255.0 | 255.255.255.0 |
| Max no. of user. | 20 | 20 |
| | ADD | ADD |

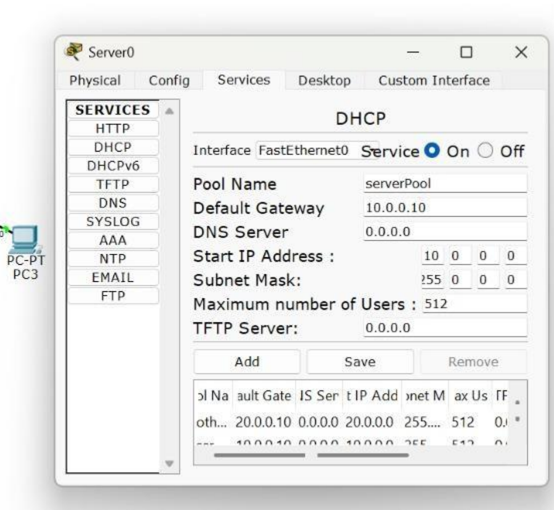
Router.
  ↳ CLI
    ↳ no
      Router > enable.
  #conf t
  #int fa0/0
  # ip address 192.168.10.1 255.255.255.0
  # ip helper-address 192.168.10.2
  # no shutdown
  do write memory
  # exit

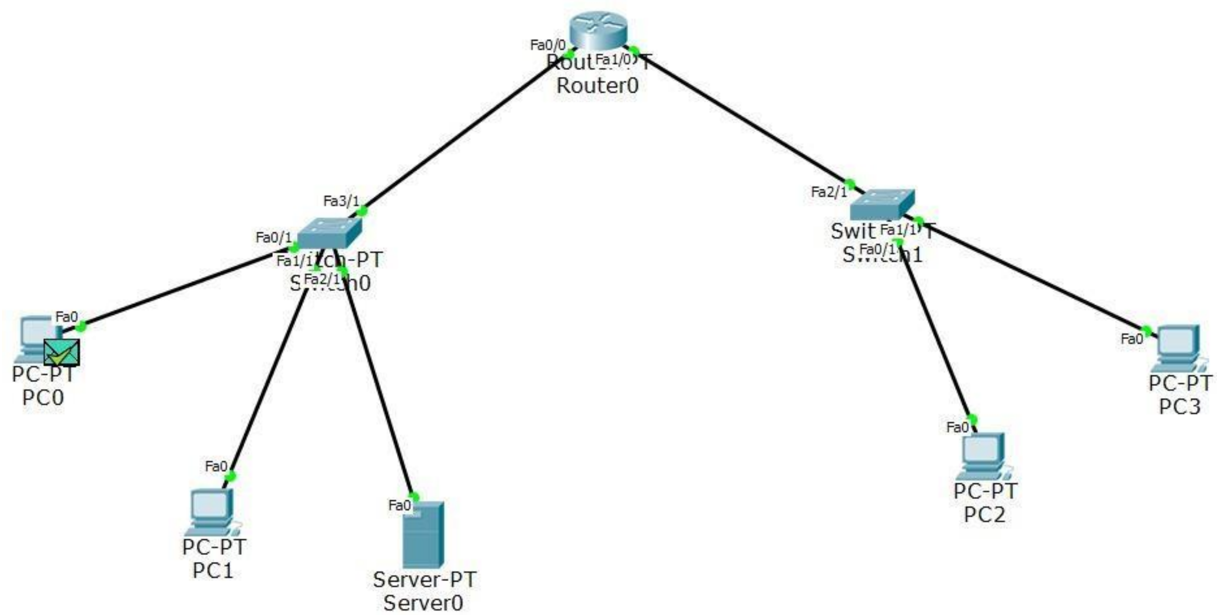
  #int fa0/1
  # ip address 192.168.20.1 255.255.255.0
  # ip helper-address 192.168.10.2
  # no shutdown
  do write memory
  # exit
  ~~# exit~~
  ~~write memory.~~

3/9/25

Screenshots/ Output:




Observation:

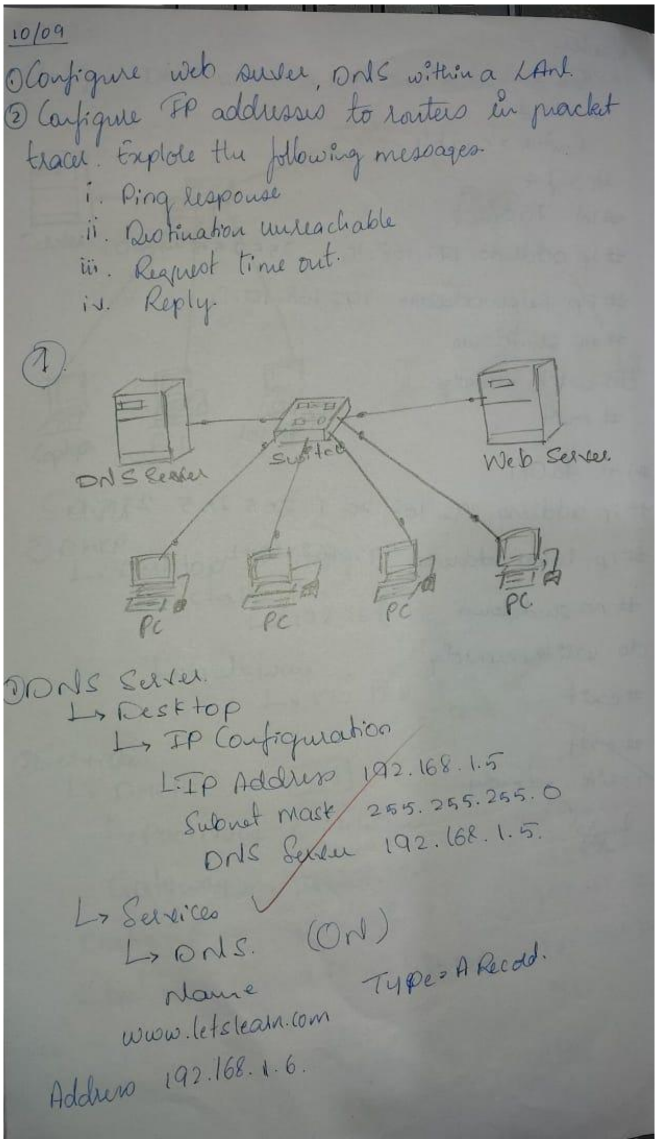
- The DHCP server successfully allocated IP addresses to clients within the LAN, confirming proper scope configuration and automatic distribution of network parameters.
- DHCP relay (IP Helper) enabled clients outside the LAN to obtain leases from the central DHCP server, demonstrating correct inter-network forwarding of DHCP Discover and Offer messages.

## **Program 4**

Aim of the program:

Configure Web Server, DNS within a LAN.

Procedure and topology:



10/09

① Configure web server, DNS within a LAN.

② Configure IP addresses to routers in packet tracer. Explore the following messages.

i. Ping response

ii. Destination unreachable

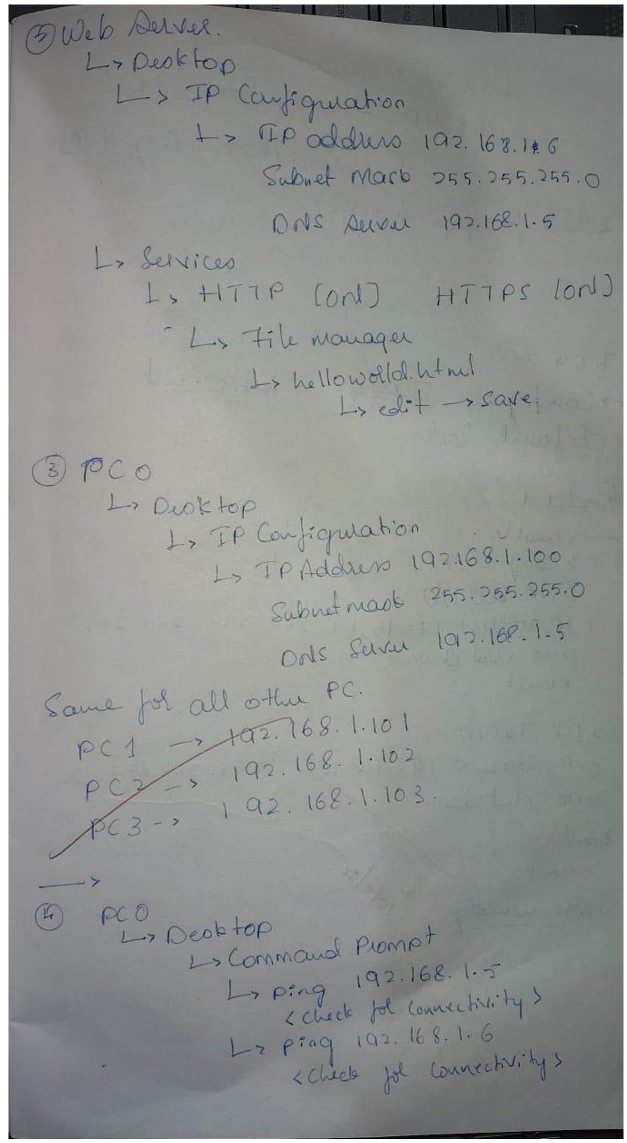
iii. Request time out.

iv. Reply.

①



② DNS Server.
↳ Desktop
↳ IP Configuration
↳ IP Address 192.168.1.5
Subnet mask 255.255.255.0
DNS Server 192.168.1.5.
↳ Services
↳ DNS. (On)
Name Type=A Record.
www.letslearn.com
Address 192.168.1.6.



⑤ Web Server.
↳ Desktop
↳ IP Configuration
↳ IP address 192.168.1.6
Subnet Mask 255.255.255.0
DNS Server 192.168.1.5
↳ Services
↳ HTTP (On) HTTPS (On)
↳ File manager
↳ helloworld.html
↳ edit → save.

③ PC0
↳ Desktop
↳ IP Configuration
↳ IP Address 192.168.1.100
Subnet mask 255.255.255.0
DNS Server 192.168.1.5

Same for all other PC.
PC1 → 192.168.1.101
PC2 → 192.168.1.102
PC3 → 192.168.1.103.

④ PC0
↳ Desktop
↳ Command Prompt
↳ ping 192.168.1.5
<check for connectivity>
↳ ping 192.168.1.6
<check for connectivity>

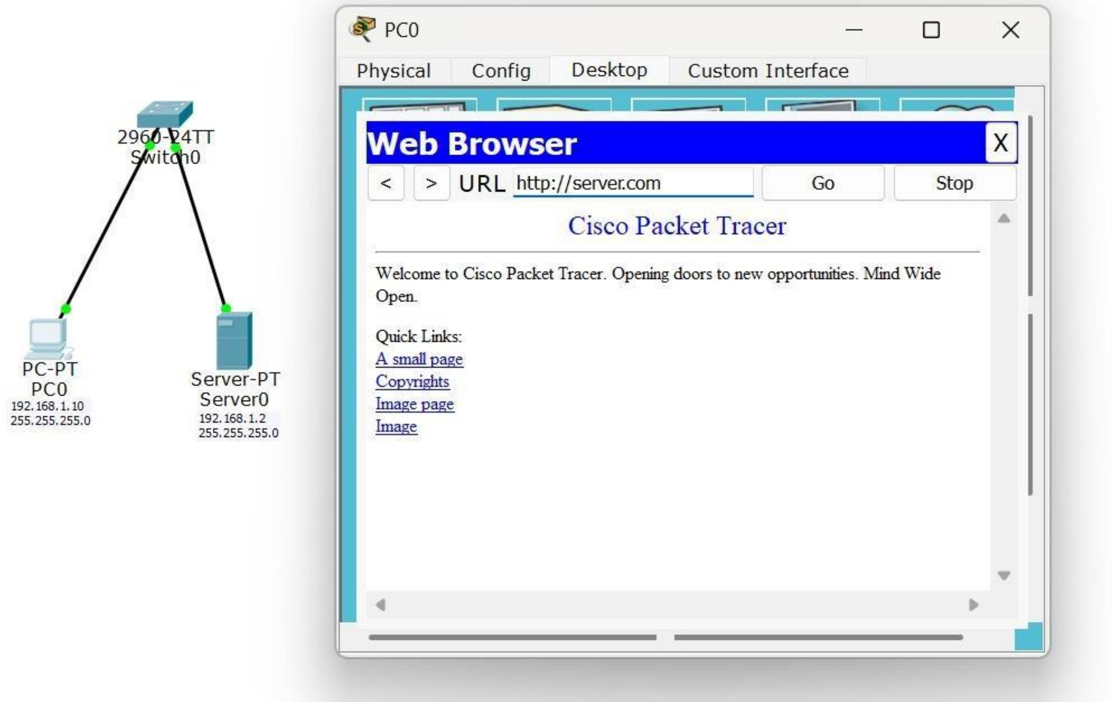
↳ Web Browser

↳ URL → www.letslearn.com

↳ QuickLinks → A small page

<check for your message>.

Screenshots/ Output:




Observation:

- The DNS server successfully resolved domain names to the corresponding web server's IP address, confirming proper hostname-to-IP mapping within the LAN.
- HTTP requests reached the web server using the DNS-resolved address, validating correct server configuration and internal LAN communication.

**Program 5**

Aim of the program:

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

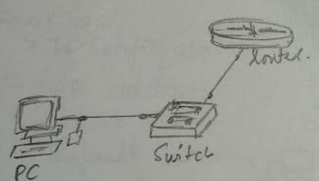
Procedure and topology:

8.10.2025:

Configure TELNET to access router from a remote location.

TELNET:

* It is used to access remote server.
* It is simple command line tool that runs on computer and it allows to send commands remotely to a server and administrator.
* TELNET is also used to manage other devices like, router, switch, to check if ports are open or close in a server.

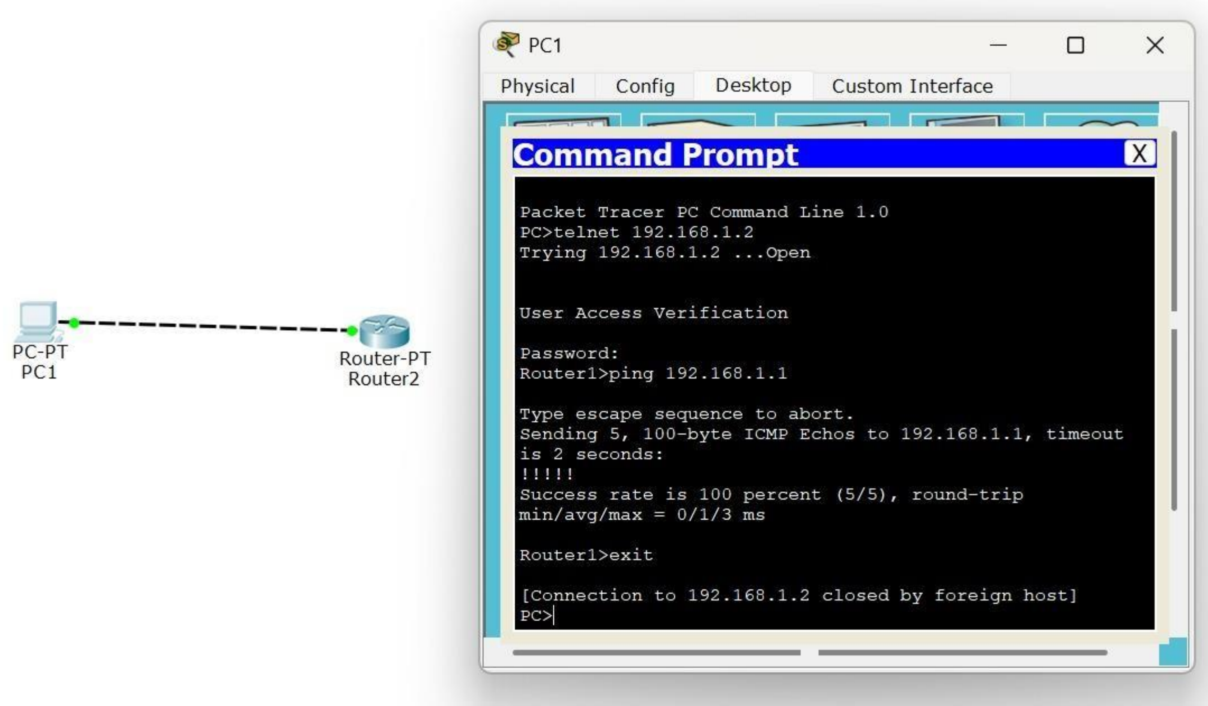



Router >
↳ CLI
↳ Router > enable
↳ Router# conf t
Router(config) #hostname R1
R1(config) #enable secret ep
R1(config) #int fa0/0
R1(config-if) # ip address 192.168.1.1 255.255.255.0
R1(config-if) # no shutdown
↳ enter.

R1(config-if) # line vty 0 5
R1(config-line)# login
—>
R1(config-line) # password tp.
R1(config-line) # exit
R1(config) # exit
wr.
R1# show ip interface brief
=>

PC >
↳ Desktop
↳ IP configuration.
↳ Static
IP address 192.168.1.2.
Subnet mask 255.255.255.0
Default Gateway 192.168.1.1
↳ Command prompt
↳ ping 192.168.1.1
=>
↳ PC > telnet 192.168.1.1.
Password: tp
R1 > enable
password: ep
R1# show ip interface brief.
R1 # conf t.
R1(config) # int fa1/0

Screenshots/ Output:




Observation:

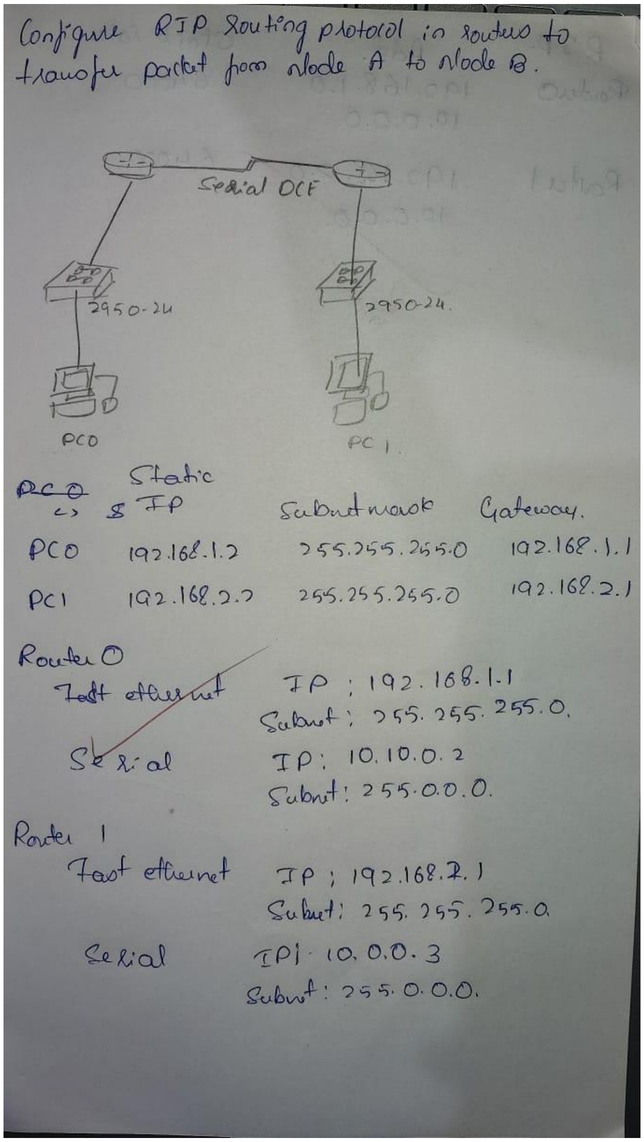
- The Telnet session successfully established a remote CLI connection to the router, confirming proper VTY line configuration and IP reachability between the IT office PC and the server room router.
- Command execution over the Telnet session demonstrated reliable remote device management.

**Program 6**

Aim of the program:

Configure RIP routing Protocol in Routers

Procedure and topology:


Configure RIP Routing protocol in routers to transfer packet from Node A to Node B.



| PC0 <> Static IP | | Subnetmask | Gateway. |
|---|---|---|---|
| PC0 | 192.168.1.2 | 255.255.255.0 | 192.168.1.1 |
| PC1 | 192.168.2.2 | 255.255.255.0 | 192.168.2.1 |

Router 0

Fast ethernet IP : 192.168.1.1
Subnet: 255.255.255.0.

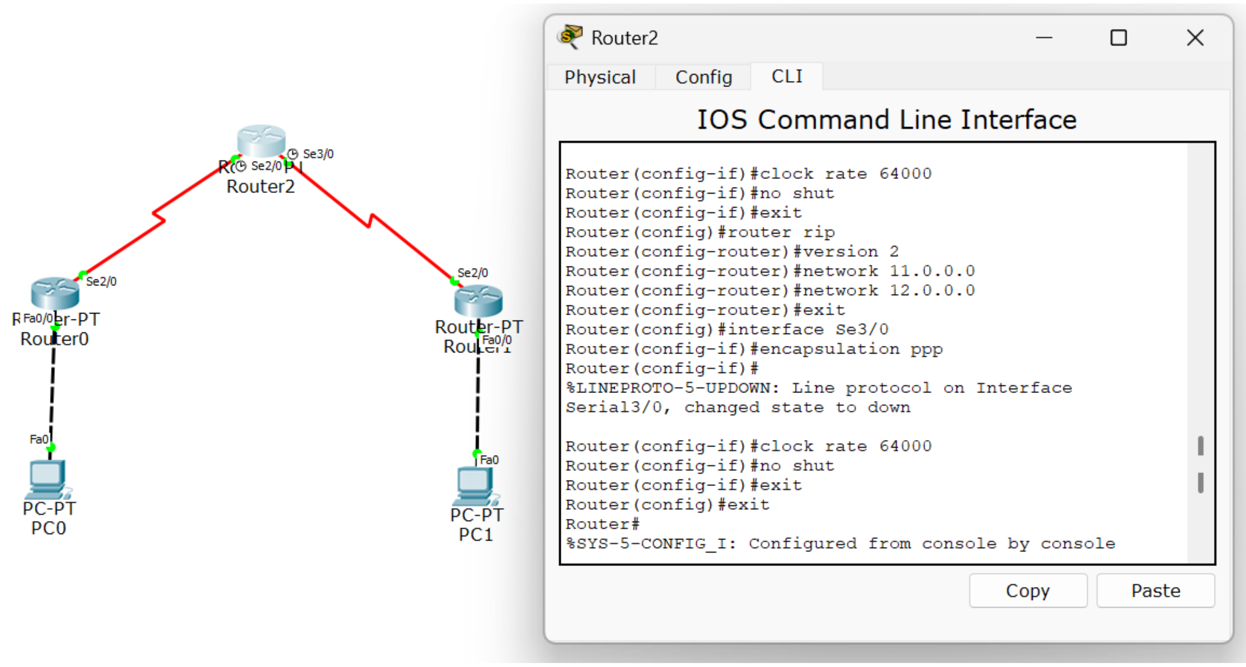
Serial IP: 10.10.0.2
Subnet: 255.0.0.0.

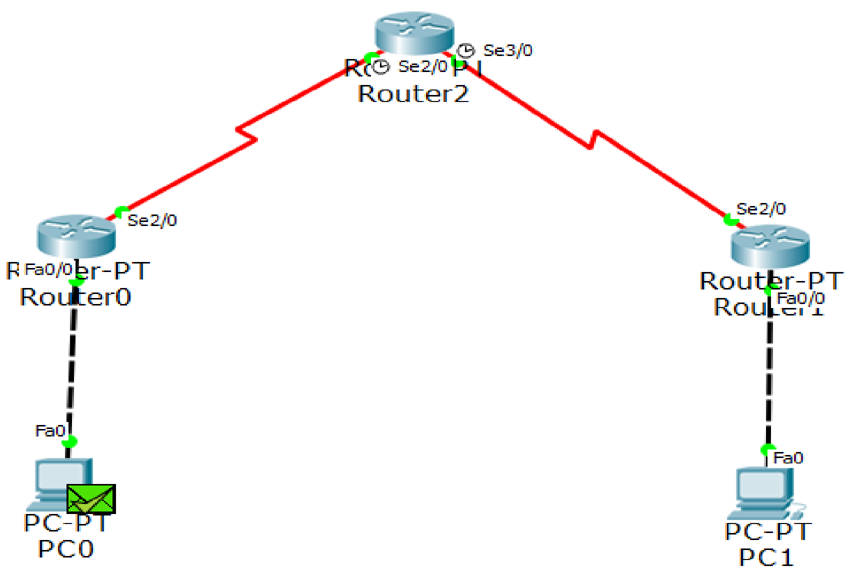
Router 1

Fast ethernet IP ; 192.168.2.1
Subnet: 255.255.255.0

Serial IP: 10.0.0.3
Subnet: 255.0.0.0.

Screenshots/ Output:




Observation:

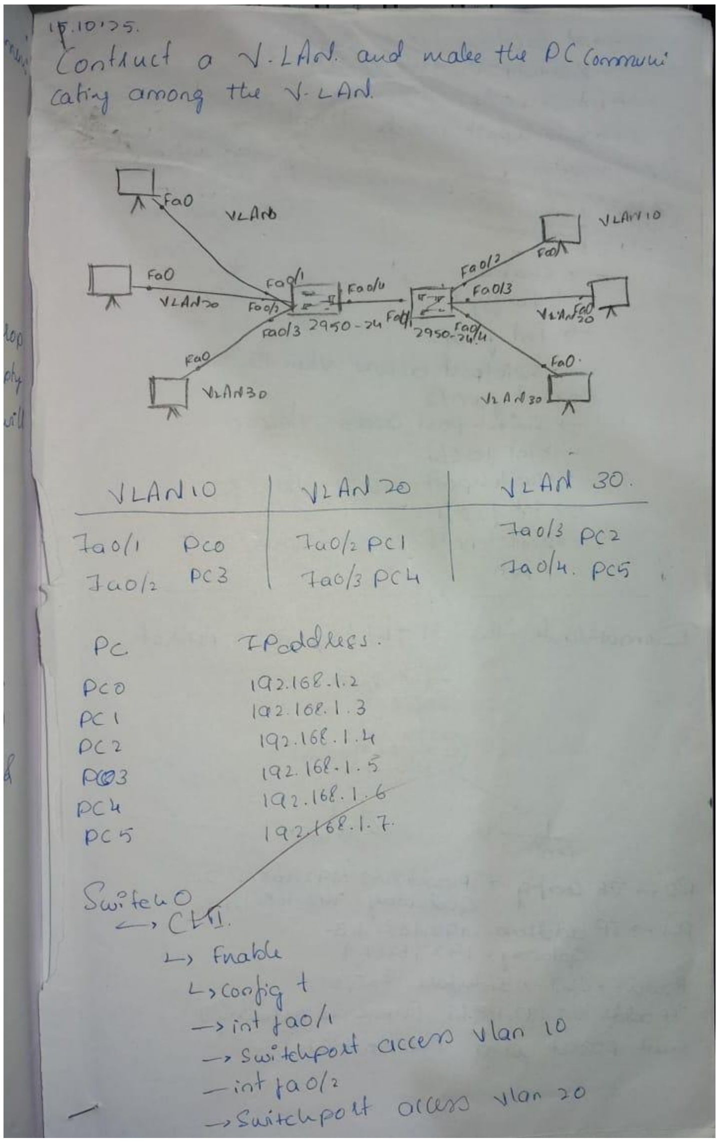
- RIP routing updates were successfully exchanged between routers, allowing each router to dynamically learn remote network routes through hop-count–based distance vector advertisements.
- The routing tables converged correctly, and successful ping tests confirmed end-to-end connectivity maintained by periodic RIP updates and route propagation.

**Program 7**

Aim of the program:

To construct a VLAN and make the PCs communicate among a VLAN

Procedure and topology:



15.10.25.

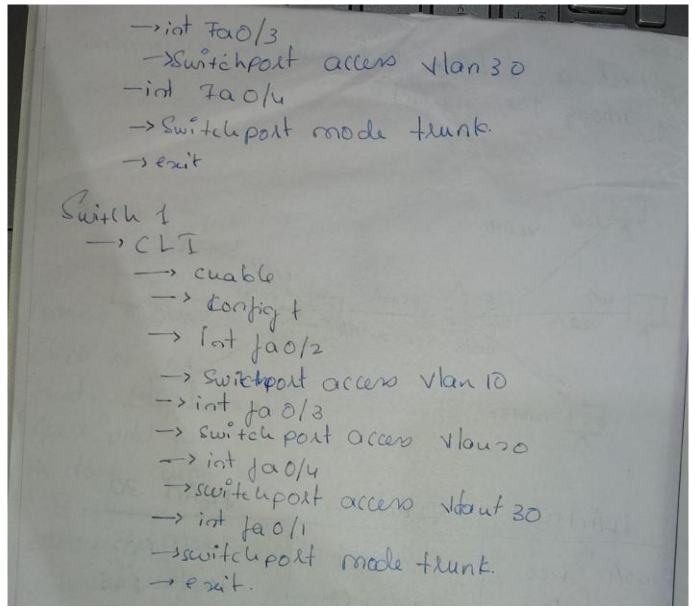
Contruct a V-LAN and make the PC communicating among the V-LAN.



| VLAN10 | VLAN20 | VLAN 30. |
|---|---|---|
| Fa0/1 PC0 | Fa0/2 PC1 | Fa0/3 PC2 |
| Fa0/2 PC3 | Fa0/3 PC4 | Fa0/4. PC5 |

| PC | IP address. |
|---|---|
| PC0 | 192.168.1.2 |
| PC1 | 192.168.1.3 |
| PC2 | 192.168.1.4 |
| PC3 | 192.168.1.5 |
| PC4 | 192.168.1.6 |
| PC5 | 192.168.1.7. |

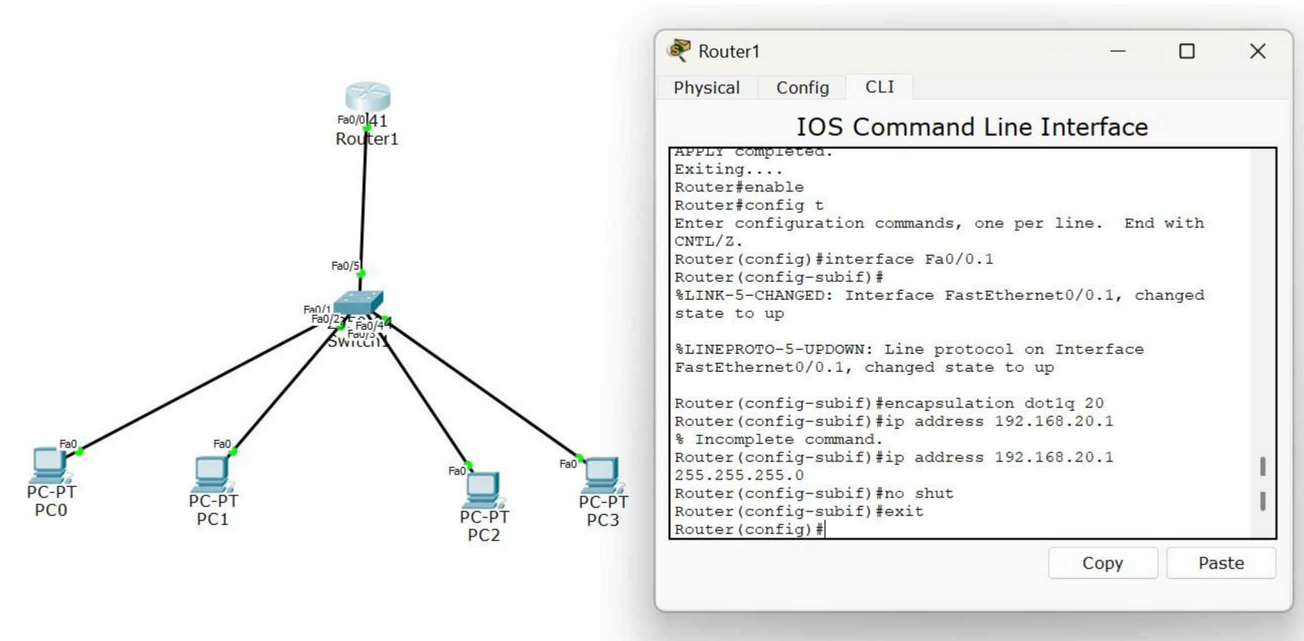
Switch 0
→ CLI.
→ Enable
→ config t
→ int fa0/1
→ switchport access vlan 10
— int fa0/2
→ switchport access vlan 20

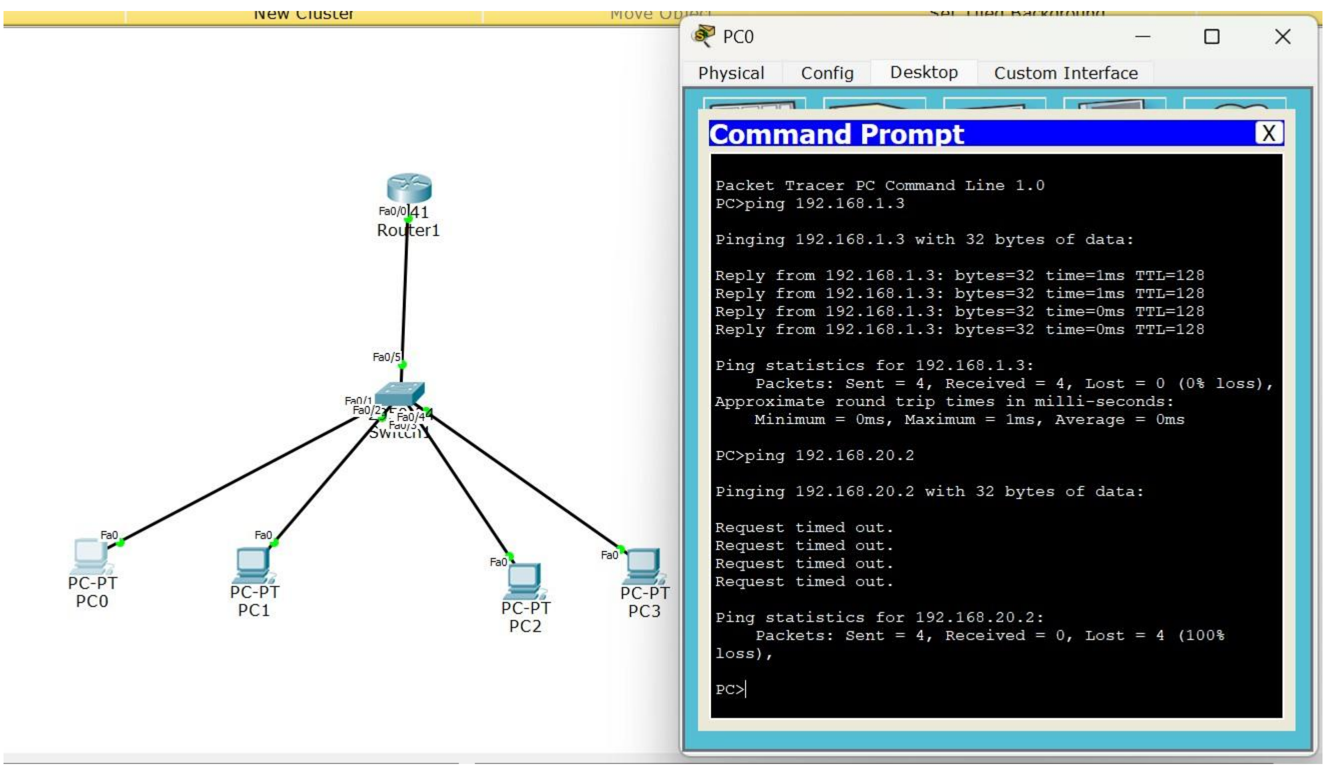


→ int fa0/3
→ switchport access vlan 30
— int fa0/4
→ switchport mode trunk.
→ exit

Switch 1
→ CLI
→ enable
→ config t
→ int fa0/2
→ switchport access vlan 10
→ int fa0/3
→ switchport access vlan 20
→ int fa0/4
→ switchport access vlan 30
→ int fa0/1
→ switchport mode trunk.
→ exit.

Screenshots/ Output:




Observation:

- VLAN segmentation successfully separated broadcast domains, and switch ports were correctly assigned to their respective VLAN IDs using access mode configuration.
- Inter-VLAN communication was achieved through the Layer-3 device, and successful ping tests confirmed proper VLAN membership, tagging, and routing functionality.

**Program 8**

Aim of the program:

To construct a WLAN and make the nodes communicate wirelessly

Procedure and topology:

15.10.25.

To Contruct a WLAN & make nodes communi cate wirelessly.


Connect Smartphone.

Click on PC → physical → turn off → drag & drop that LAN to modules, then to that empty space add new one then turn On → Connect will be done automatically.

[To change that default wired connection to wireless]. [IP add of all PC will be done automatically].

Router → config → wireless.

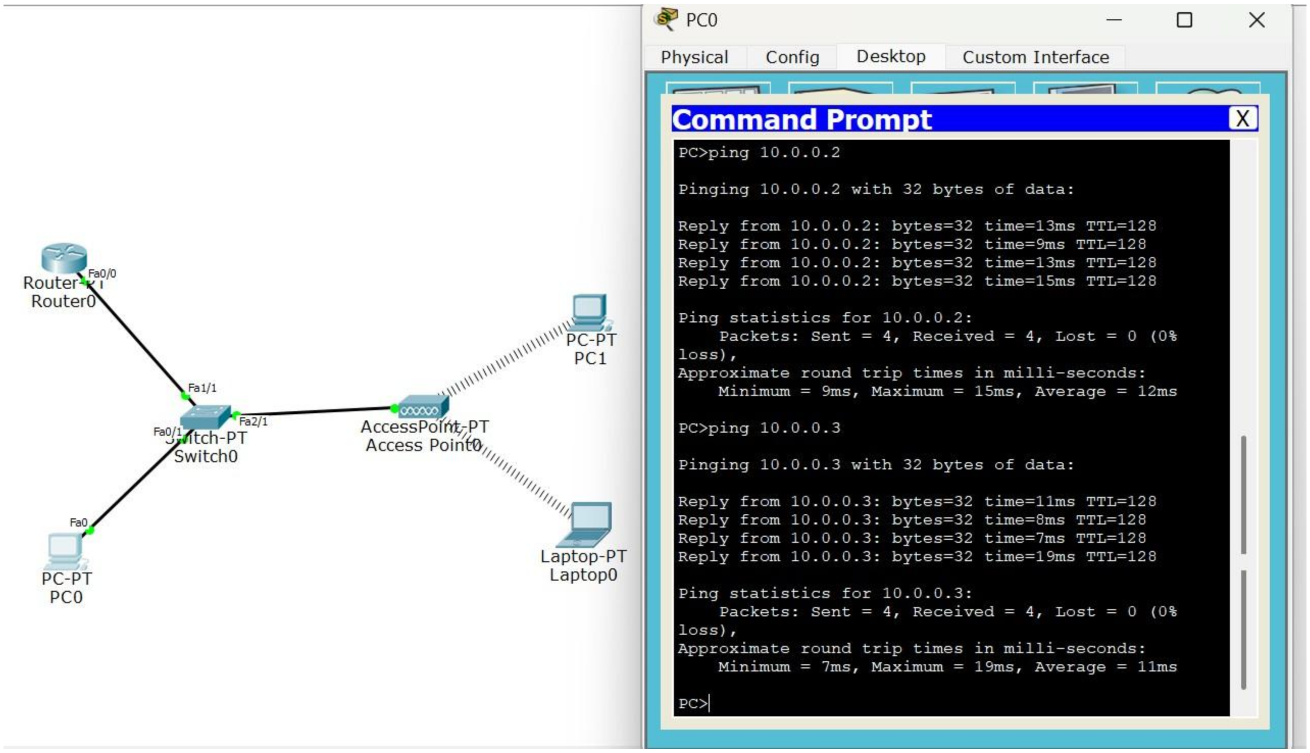
SSID: bmsce.

Authentication ; WAPA2-PSK phase : bmsce1234

[Now authomatic connection to PC will be lost]

PC - click - config → wireless → Add that SSID & connection will be done.

Screenshots/ Output:




Observation:

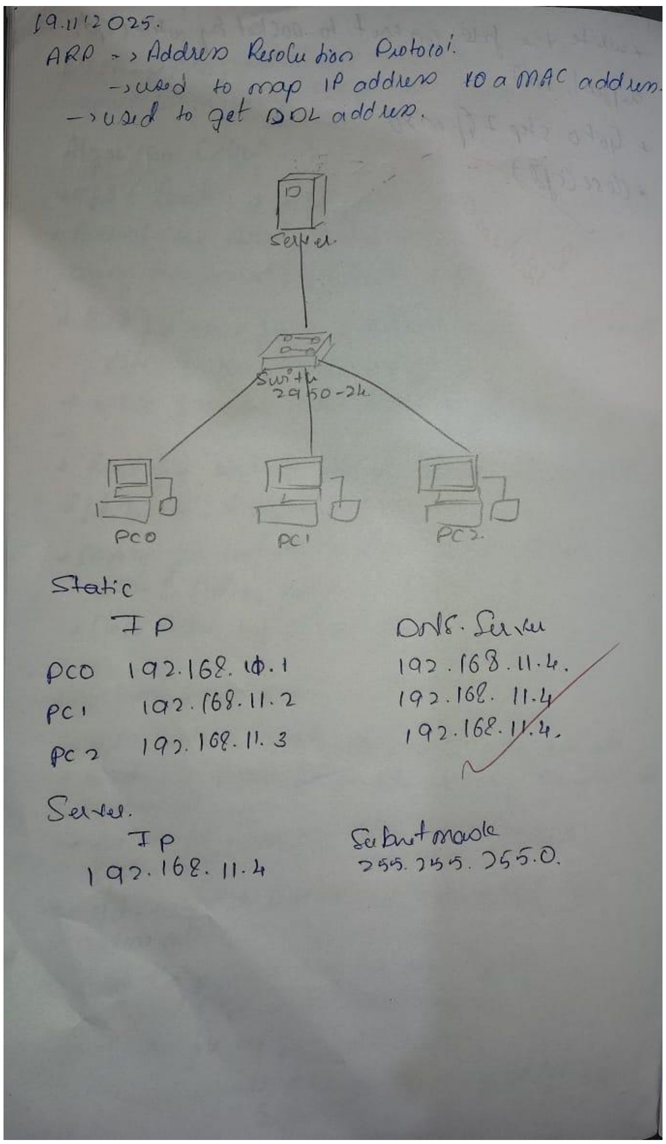
- The WLAN configuration enabled wireless nodes to associate with the access point using the configured SSID and security settings, confirming proper authentication and signal coverage.
- Successful ping communication between wireless devices verified stable wireless Connectivity.

## Program 9

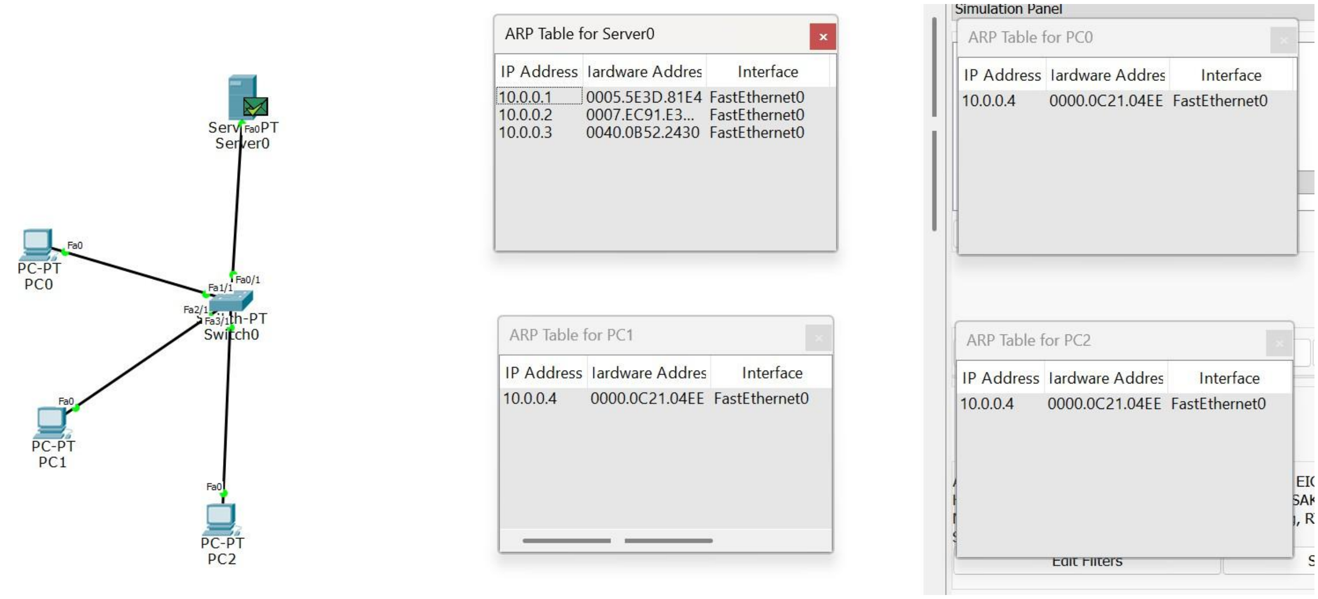
Aim of the program:

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

Procedure and topology:



Screenshots/ Output:




Observation:

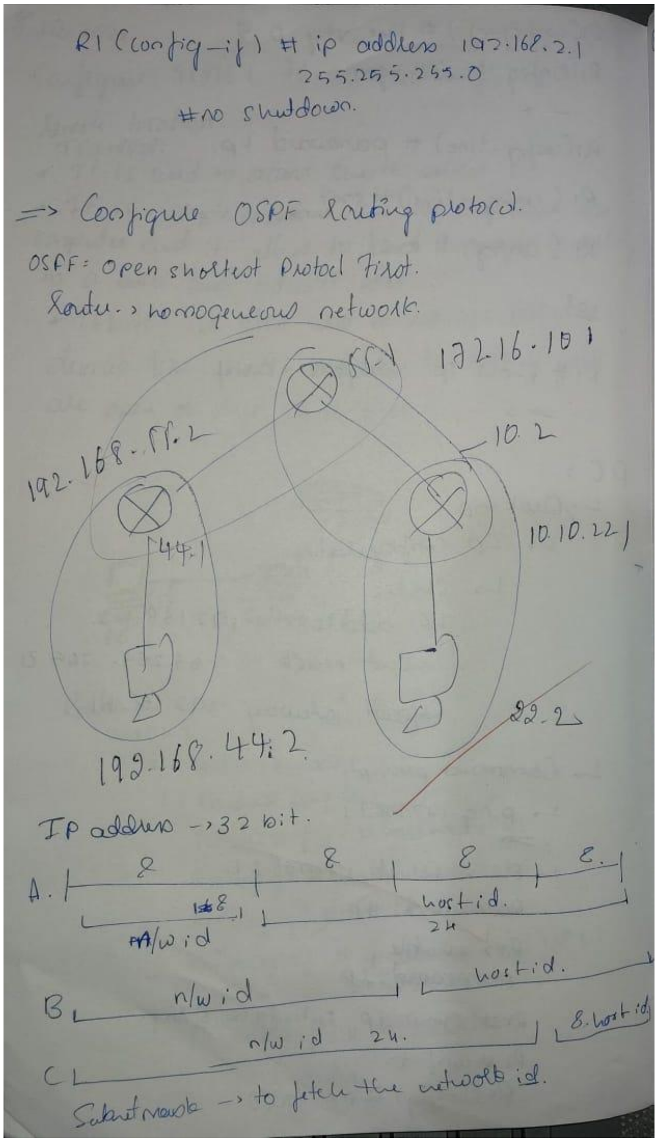
- ARP successfully resolved the destination IP address to its corresponding MAC address, as seen from ARP request and reply exchanges between LAN hosts.
- The populated ARP tables and successful ping communication confirmed correct layer-2 addressing, frame forwarding, and basic LAN operation.

**Program 10**

Aim of the program:

Configure OSPF routing protocol

Procedure and topology:



R1 (config-if) # ip address 192.168.2.1 255.255.255.0

#no shutdown.

=> Configure OSPF routing protocol.

OSPF = Open shortest Protocol First.

Router -> homogeneous network.



IP address -> 32 bit.

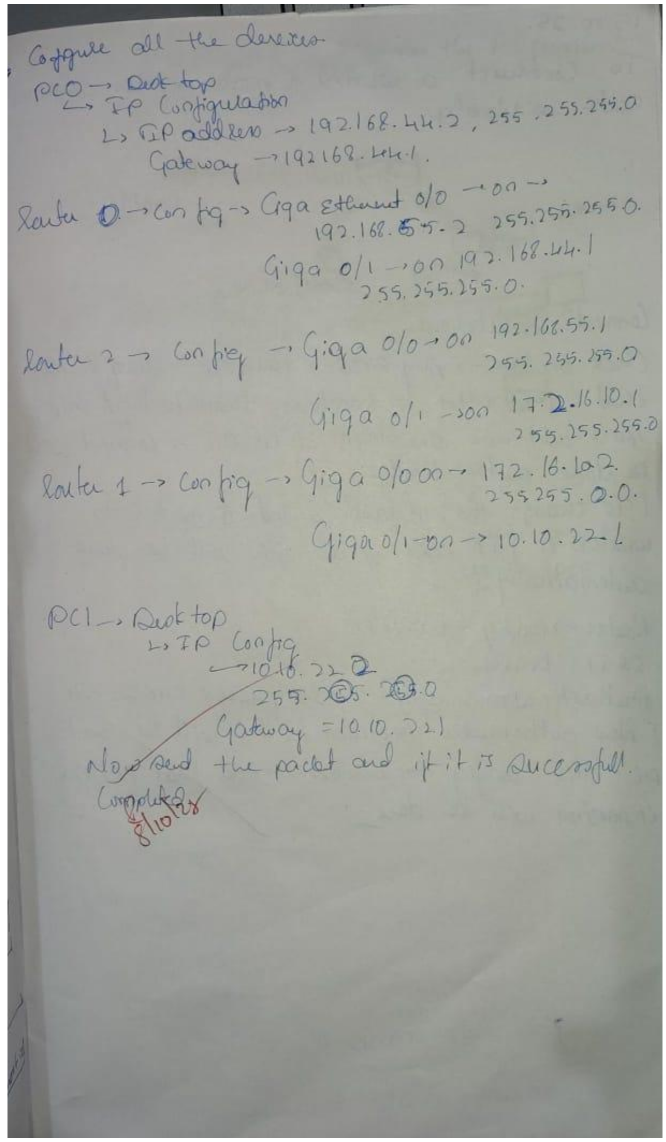
A. 8 | 8 | 8 | 8.

8 n/w id | host id. 24

B n/w id | host id.

C n/w id 24. | 8. host id

Subnet mask -> to fetch the network id.



Configure all the devices

PC0 -> Desk top

-> IP Configuration

-> IP address -> 192.168.44.2, 255.255.255.0

Gateway -> 192.168.44.1.

Router 0 -> Config -> Giga Ethernet 0/0 -> on -> 192.168.55.2 255.255.255.0.

Giga 0/1 -> on 192.168.44.1 255.255.255.0.

Router 2 -> Config -> Giga 0/0 -> on 192.168.55.1 255.255.255.0

Giga 0/1 -> on 172.16.10.1 255.255.255.0

Router 1 -> Config -> Giga 0/0 on -> 172.16.10.2. 255.255.0.0.

Giga 0/1 -> on -> 10.10.22.1

PC1 -> Desk top

-> IP Config

-> 10.10.22.2

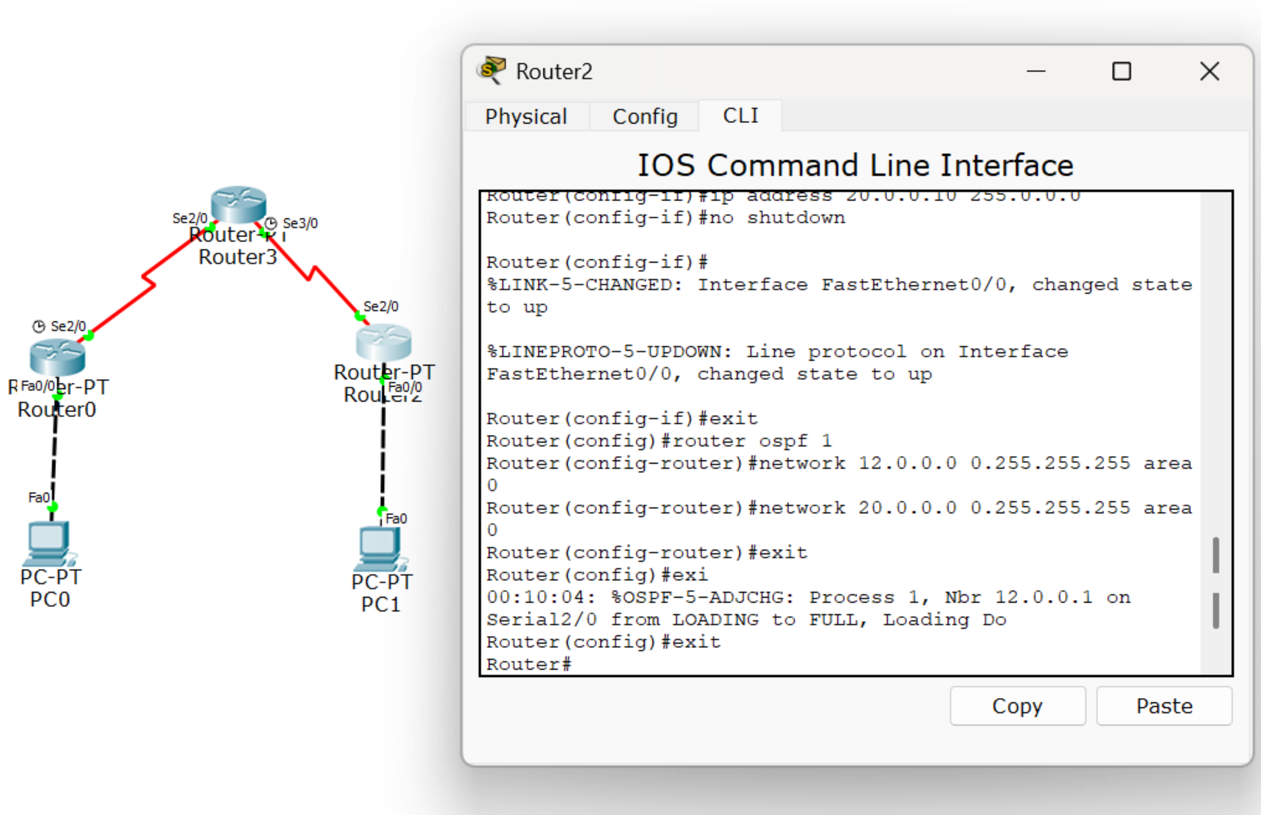
255.255.255.0

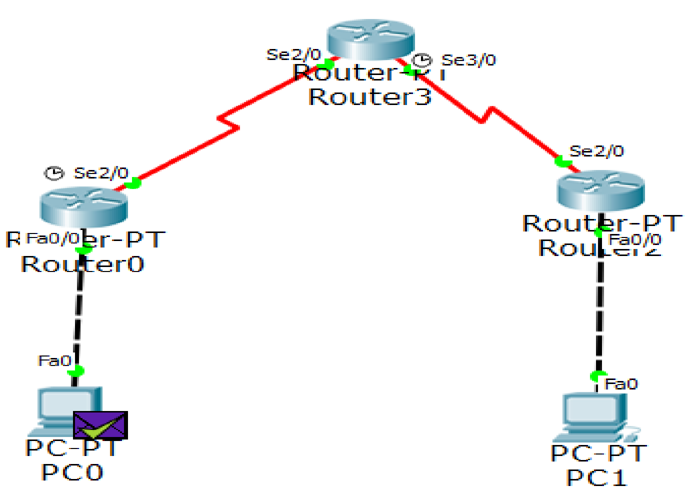
Gateway = 10.10.22.1

Now send the packet and if it is successfull.

Completed 8/10/28

Screenshots/ Output:




Observation:

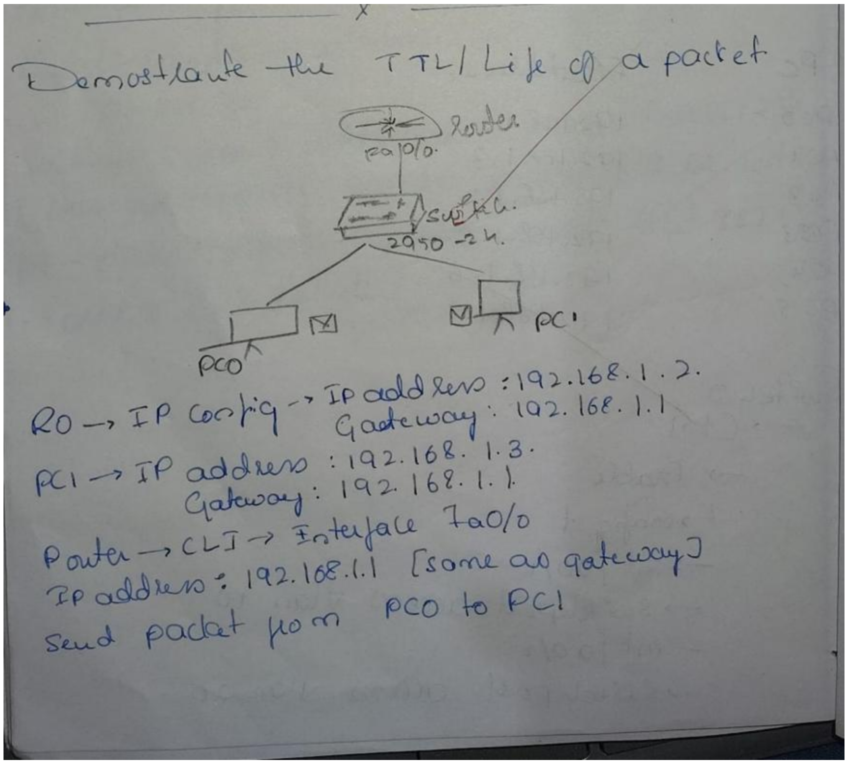
- OSPF successfully established neighbour adjacencies and exchanged LSAs, allowing routers to build a synchronized link-state database across the OSPF area.
- The routing tables converged using SPF calculations, and successful pings confirmed efficient path selection and dynamic route learning through OSPF.

**Program 11**

Aim of the program:

Demonstrate the TTL/ Life of a Packet

Procedure and topology:


Demonstrate the TTL/ Life of a packet

router

Fa0/0

switch

2950-24

PC0

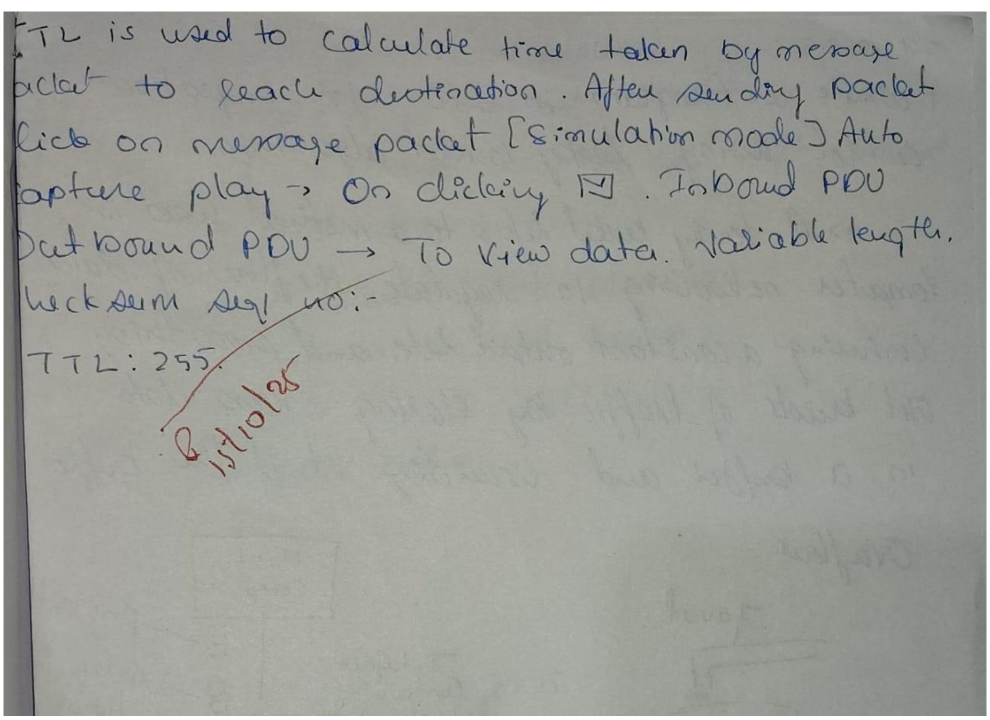
PC1

R0 → IP config → IP address : 192.168.1.2.
Gateway : 192.168.1.1

PC1 → IP address : 192.168.1.3.
Gateway : 192.168.1.1

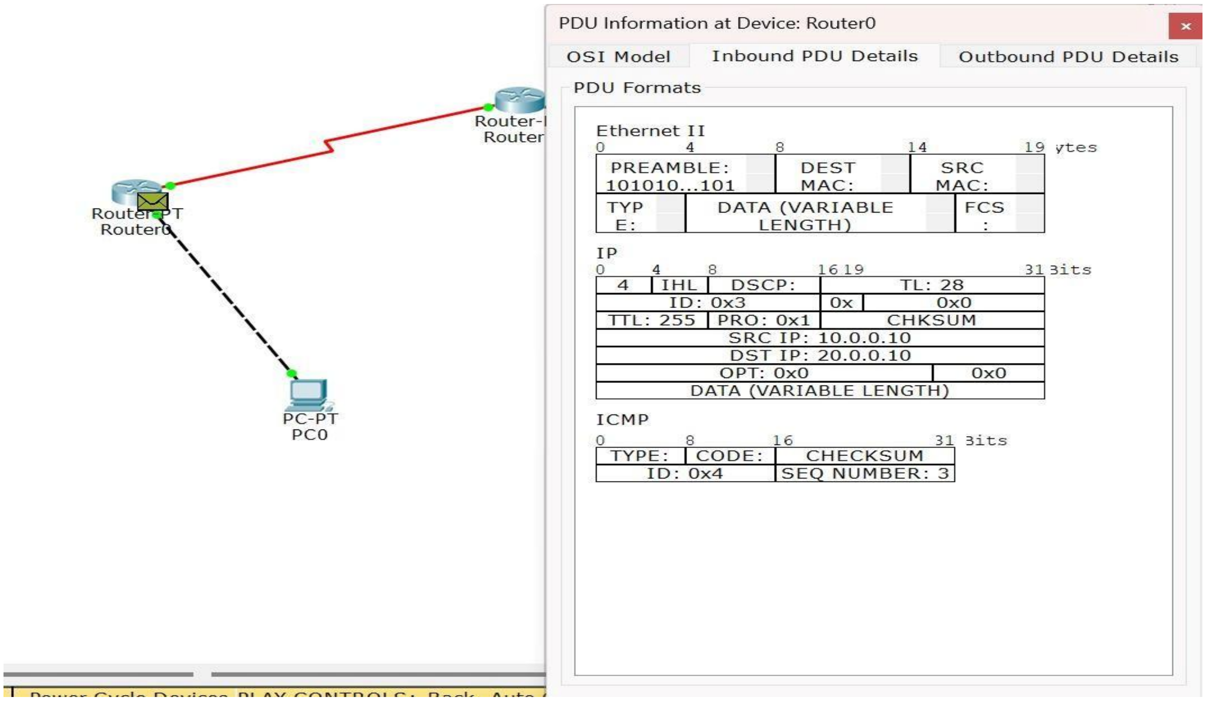
Router → CLI → Interface Fa0/0
IP address : 192.168.1.1 [same as gateway]

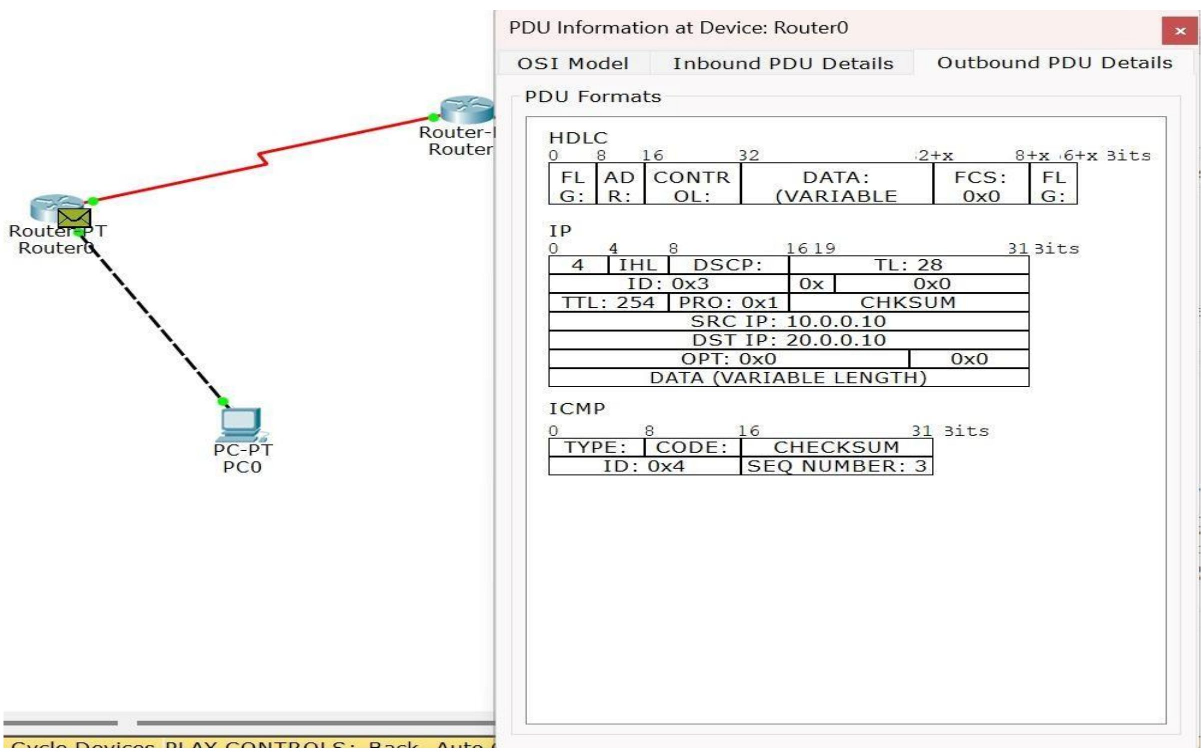
Send packet from PC0 to PC1


TTL is used to calculate time taken by message packet to reach destination. After sending packet click on message packet [simulation mode] Auto capture play → On clicking ✉. Inbound PDU Outbound PDU → To view data. Variable length, checksum, seq no:-

TTL : 255

Screenshots/ Output:




Observation:

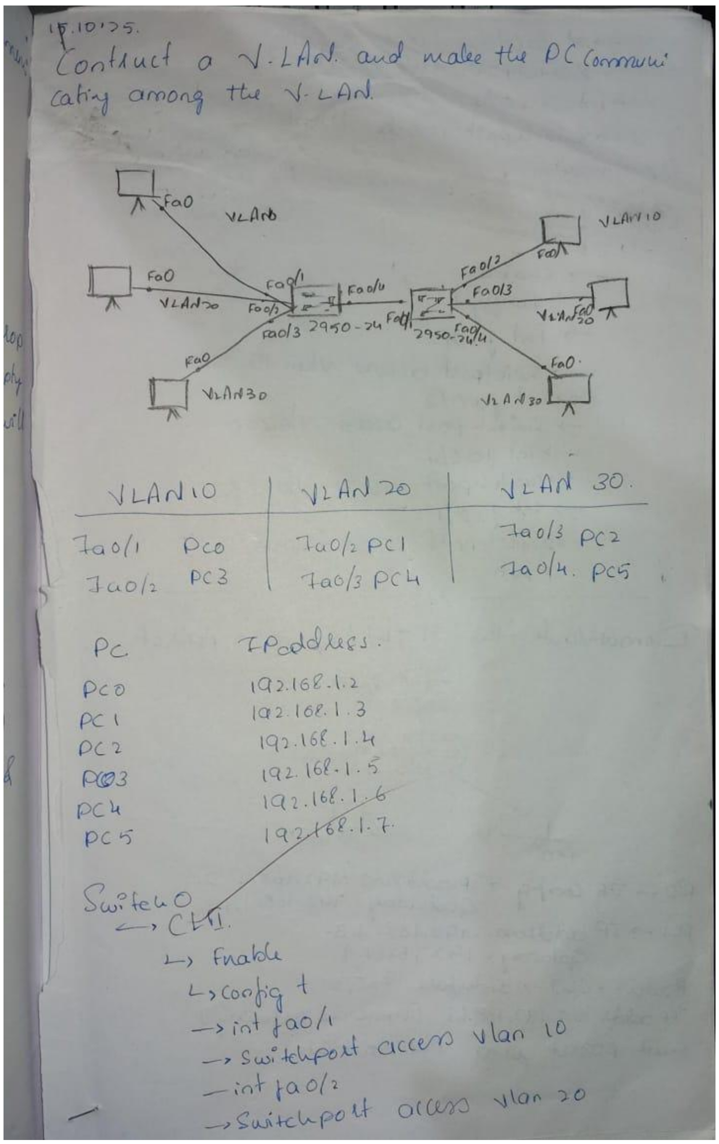
- The TTL field in the IP header decreased by one at each router hop, demonstrating its role in preventing packets from looping indefinitely in the network.

**Program 12**

Aim of the program:

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

Procedure and topology:



15.10.25.

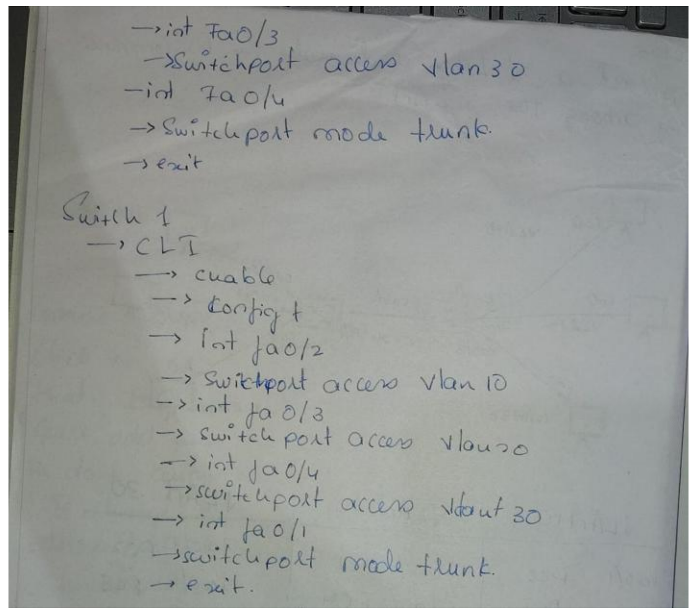
Contruct a V.LAN. and make the PC communicating among the V.LAN.

Fa0 VLAN10 Fa0 VLAN20 Fa0/1 Fa0/2 Fa0/3 Fa0/4 2950-24 Fa0/1 2950-24 Fa0/2 Fa0/3 Fa0/4 Fa0 VLAN30 VLAN10 VLAN20 VLAN30

| VLAN10 | | VLAN20 | | VLAN 30. | |
|---|---|---|---|---|---|
| Fa0/1 | PC0 | Fa0/2 | PC1 | Fa0/3 | PC2 |
| Fa0/2 | PC3 | Fa0/3 | PC4 | Fa0/4. | PC5 |

| PC | IPaddress. |
|---|---|
| PC0 | 192.168.1.2 |
| PC1 | 192.168.1.3 |
| PC2 | 192.168.1.4 |
| PC3 | 192.168.1.5 |
| PC4 | 192.168.1.6 |
| PC5 | 192.168.1.7. |

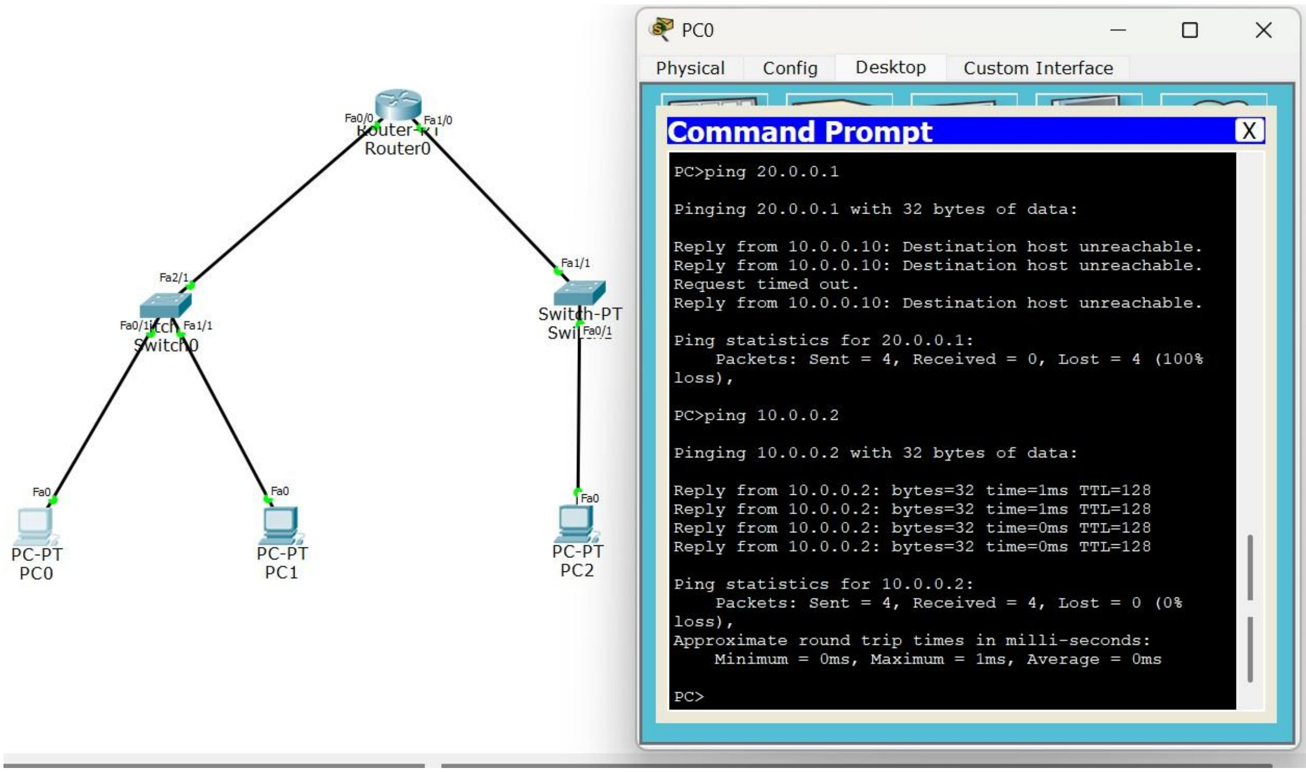
Switch0
→ CLI.
→ Enable
→ config t
→ int fa0/1
→ Switchport access vlan 10
— int fa0/2
→ Switchport access vlan 20



→ int Fa0/3
→ Switchport access vlan 30
— int Fa0/4
→ Switchport mode trunk.
→ exit

Switch 1
→ CLI
→ enable
→ config t
→ int fa0/2
→ Switchport access vlan 10
→ int fa0/3
→ switch port access vlan 20
→ int fa0/4
→ switchport access vlan 30
→ int fa0/1
→ switchport mode trunk.
→ exit.

Screenshots/ Output:



Observation:

- Proper IP addressing on router interfaces enabled successful ICMP echo and echo-reply communication, confirming correct Layer-3 configuration and reachability.
- “Destination Unreachable” and “Request Timed Out” responses occurred when routes or interfaces were misconfigured, demonstrating how routers handle missing paths and non-responsive hosts.

**CYCLE 2:**

**<u>Program 13</u>**

Aim of the program:

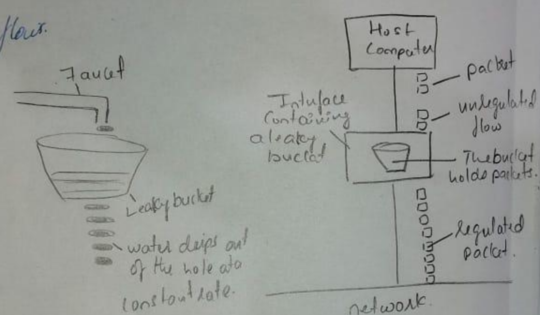
Write a program for congestion control using Leaky bucket algorithm

Procedure and topology:

29.10.25.

1. Write a Program in C/C++ for congestion control using Leaky Bucket algorithm.

=> A leaky bucket refers to a method used in computer networking to regulate the flow of data, ensuring a constant output rate and smoothing out bursts of traffic by storing excess data in a buffer and discarding it if the buffer overflows.




Algorithm:

1. start
2. Input:
   * Bucket capacity (cap)
   * Output rate (process)
   * ~~Packets~~ number of seconds (n sec)
   * Packets arriving each second (inp[i])
3. Initialize count=0, drop=0.
4. For each ~~sec~~ second:
   * count = count + inp[i]
   * if count > cap:
     -> drop = count - cap
     -> count = cap
   * sent = min(count, process)
   * count = count - sent
   * Print second, received, sent, left, dropped.
   * Reset drop=0.
5. After all seconds, while count > 0:
   * sent = min (count, process)
   * count = count - sent
   * print status
6. stop.

<u>Output</u>:

Enter the bucket size : 5
Enter the processing rate : 2
Enter the number of seconds you want to stimulate : 3.
Enter the size of packet entering at 1 sec : 5
Enter the size of packet entering at 2 sec : 4
Enter the size of packet entering at 3 sec : 3.
Enter the size of packet entering

| Second | Packet received | Packet sent | Packet left | Dro |
|---|---|---|---|---|
| 1 | 5 | 2 | 3 | 0 |
| 2 | 4 | 2 | 3 | 2 |
| 3 | 3 | 2 | 3 | 1 |
| 4 | 0 | 2 | 1 | 0 |
| 5 | 0 | 1 | 0 | 0 |

Screenshots/ Output:

```
Output
Enter bucket size: 4
Enter outgoing size: 1
Enter number of inputs: 7
Enter the incoming packet size: 3
Bucket buffer size 3 out of 4
After outgoing 2 packets left out of 4 in buffer
Enter the incoming packet size: 2
Bucket buffer size 4 out of 4
After outgoing 3 packets left out of 4 in buffer
Enter the incoming packet size: 1
Bucket buffer size 4 out of 4
After outgoing 3 packets left out of 4 in buffer
Enter the incoming packet size: 4
Dropped 3 packets
Bucket buffer size 4 out of 4
After outgoing 3 packets left out of 4 in buffer
Enter the incoming packet size: 0
Bucket buffer size 3 out of 4
After outgoing 2 packets left out of 4 in buffer
Enter the incoming packet size: 0
Bucket buffer size 2 out of 4
After outgoing 1 packets left out of 4 in buffer
Enter the incoming packet size: 0
Bucket buffer size 1 out of 4
After outgoing 0 packets left out of 4 in buffer
```

Observation:

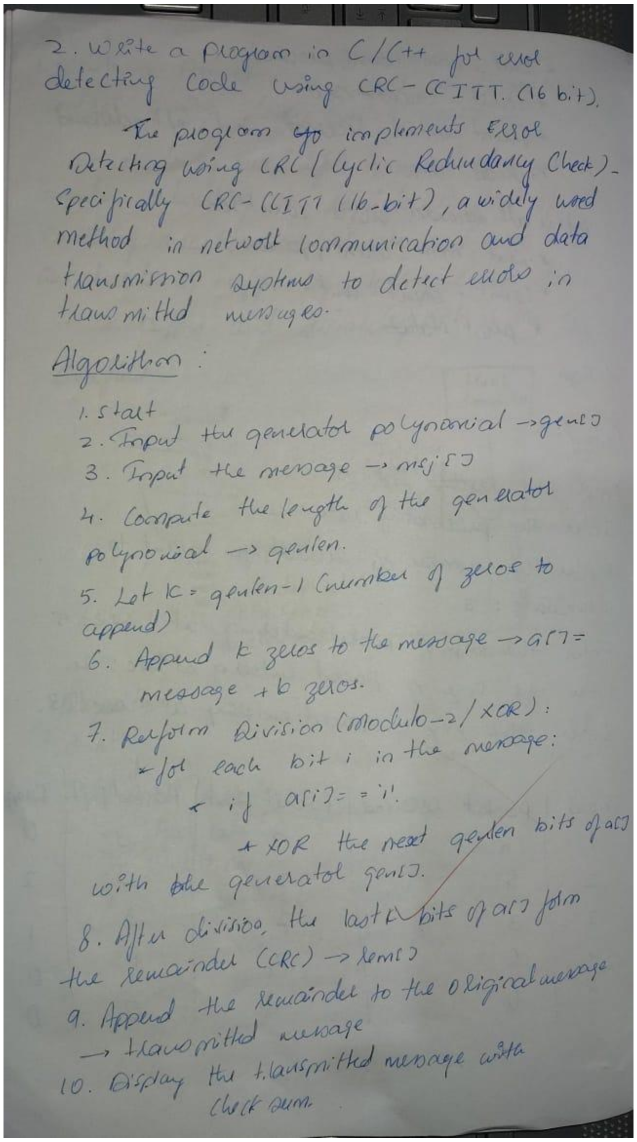
- The leaky bucket mechanism regulated the outgoing packet flow at a constant rate, preventing sudden traffic bursts from overwhelming the network.
- Packets exceeding the bucket capacity were dropped, demonstrating effective congestion control by smoothing traffic and enforcing rate-limiting.

**Program 14**

Aim of the program:

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

Procedure and topology:

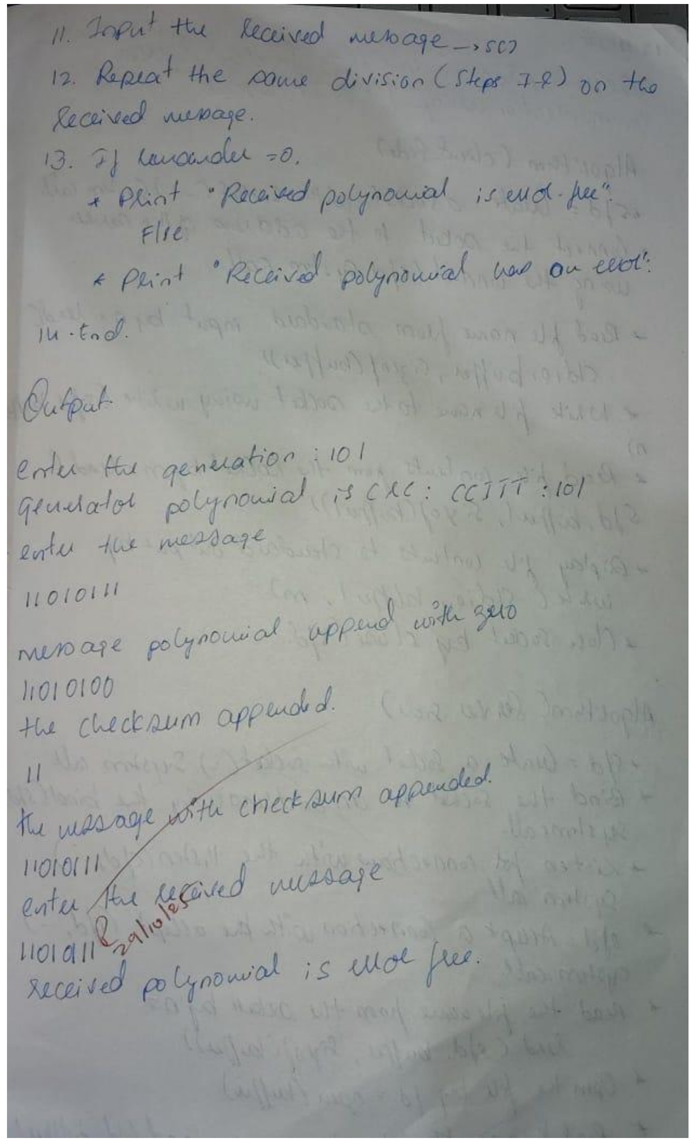


2. Write a program in C/C++ for error detecting code using CRC-CCITT (16 bit).

The program ~~yo~~ implements Error Detecting using CRC (Cyclic Redundancy Check) - specifically CRC-CCITT (16-bit), a widely used method in network communication and data transmission systems to detect errors in transmitted messages.

Algorithm:

1. Start
2. Input the generator polynomial → gen[ ]
3. Input the message → msg[ ]
4. Compute the length of the generator polynomial → genlen.
5. Let k = genlen-1 (number of zeros to append)
6. Append k zeros to the message → ar[ ] = message + k zeros.
7. Perform Division (Modulo-2 / XOR):
   - for each bit i in the message:
     - if ar[i] == '1'
       - XOR the next genlen bits of ar[ ] with the generator gen[ ].
8. After division, the last k bits of ar[ ] form the remainder (CRC) → rem[ ]
9. Append the remainder to the original message → transmitted message
10. Display the transmitted message with check sum.



11. Input the received message → s[ ]
12. Repeat the same division (Steps 7-8) on the received message.
13. If remainder = 0,
    - Print "Received polynomial is error-free".
    
    Else
    - Print "Received polynomial has an error".
14. End.

Output

enter the generation : 101
generator polynomial is CRC : CCITT : 101
enter the message

11010111

message polynomial append with zero

11010100
the checksum appended.

11
the message with checksum appended.
11010111
enter the received message
11010111 R 29/10/25
received polynomial is error free.

Screenshots/ Output:

```
Output

Enter message bits: 101100
Enter polynomial g(x): 1001

Padded data (Message + zeros): 101100000
CRC bits (remainder): 001
Transmitted message: 101100001

Enter received bits: 101100001

No Error detected. Message OK.


=== Code Execution Successful ===
```

Observation:

- The CRC-CCITT (16-bit) algorithm correctly generated a checksum for the transmitted data, ensuring reliable detection of single-bit and burst errors.
- Intentional error tests produced mismatched CRC values at the receiver, confirming accurate error detection through polynomial division.

**Program 15**

Aim of the program:

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.

Procedure and topology:

12.11.25

Write a program in C/C++ for client server communication using TCP or IP sockets

Algorithm (client side)

* sfd = Create a socket with socket(...) system call.
* Connect the socket to the address of the server using the connect(sfd, system call.
* Read file name from standard input by n = read( stdio, buffer, sizeof(buffer))
* Write file name to the socket using write (sfd, buffer, n)
* Read file contents from the socket by m = read ( sfd, buffer1, sizeof(buffer1))
* Display file contents to standard output by write ( stdio., buffer1, m)
* Close socket by close(sfd

Algorithm( server side)

* sfd = Create a socket with socket(..) system call
* Bind the socket to an address using the bind(sfd,..) system call.
* Listen for connections with the listen(sfd,..) system call
* sfd = Accept a connection with the accept (sfd, ..) system call.
* Read the filename from the socket by n = read ( sfd, buffer, sizeof(buffer))
* Open the file by fd = open (buffer)
* Read the contents of the file by m = read (fd, buffer1, sizeof( buffer1))
* write the file content to socket by write(sfd, buffer1, m)
* Goto step 7 if m>0
* close(sfd).

R 12/11/2025

Screenshots/ Output:

```
PS D:\CN stuff\TCP> python client.py
Enter filename to request: test.txt

 --- File Content ---

Hello from server side
PS D:\CN stuff\TCP> 
```

```
PS D:\CN stuff\TCP> python --version
Python 3.12.6
PS D:\CN stuff\TCP> python server.py
Server is listening on port 8080 ...
Connected by: ('127.0.0.1', 65258)
PS D:\CN stuff\TCP> 
```

Observation:

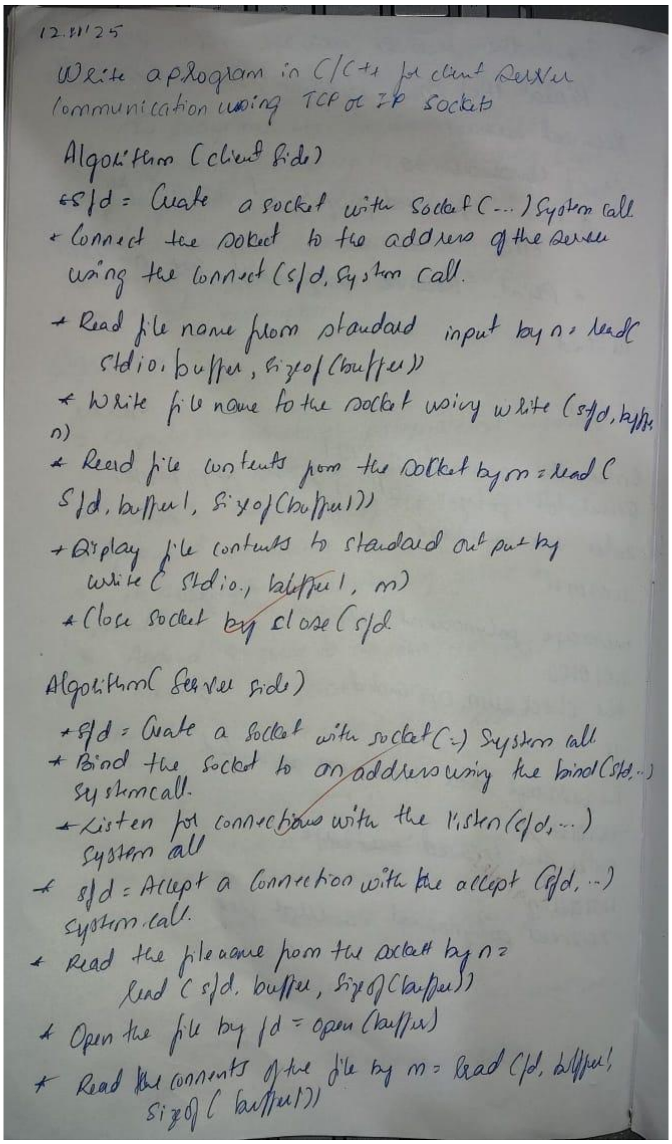
- The TCP client successfully established a reliable connection with the server and transmitted the requested filename using stream-oriented communication.
- The server correctly retrieved and returned the file contents over the same TCP session, demonstrating reliable data transfer, acknowledgment handling, and error-free delivery.

**Program 16**

Aim of the program:

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.

Procedure and topology:



12.11'25

Write a program in C/C++ for client server communication using TCP or IP sockets

Algorithm (client side)

* s/d = Create a socket with socket(...) system call
* Connect the socket to the address of the server using the connect(s/d, system call.
* Read file name from standard input by n = read( stdio, buffer, sizeof(buffer))
* Write file name to the socket using write(s/d, buffer, n)
* Read file contents from the socket by m = read( s/d, buffer1, sizeof(buffer1))
* Display file contents to standard output by write( stdio, buffer1, m)
* Close socket by close(s/d.

Algorithm( server side)

* s/d = Create a socket with socket(-) system call
* Bind the socket to an address using the bind(s/d,..) system call.
* Listen for connections with the listen(s/d,...) system call
* s/d = Accept a connection with the accept (s/d, ..) system call.
* Read the filename from the socket by n = read ( s/d. buffer, sizeof(buffer))
* Open the file by fd = open (buffer)
* Read the contents of the file by m = read (fd, buffer1, sizeof( buffer1))

Screenshots/ Output:

```
PS D:\CN stuff\UDP> python server.py
UDP Server is ready ...
Requested file: test.txt
PS D:\CN stuff\UDP>
```

```
PS D:\CN stuff\UDP> python client.py
Enter filename to request: test.txt

 --- File Content ---

Welcome to UDP file server
PS D:\CN stuff\UDP>
```

Observation:

- The UDP client successfully sent the filename as a connectionless datagram, demonstrating non-reliable, low-overhead message transfer without session establishment.
- The server responded with the file contents using UDP packets, and correct reception validated functional data exchange despite the absence of acknowledgment and retransmission mechanisms.