VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB RECORD

Computer Network Lab (23CS5PCCON)

Submitted by

Arpith Gowda H S (1BM23CS053)

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

CERTIFICATE

This is to certify that the Lab work entitled “Computer Network (23CS5PCCON)” carried out by **Arpith Gowda H S (1BM23CS053),** 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.

Surabhi S	Dr. Kavitha Sooda
Assistant Professor	Professor & HOD
Department of CSE, BMSCE	Department of CSE, BMSC

Index

Github Link:

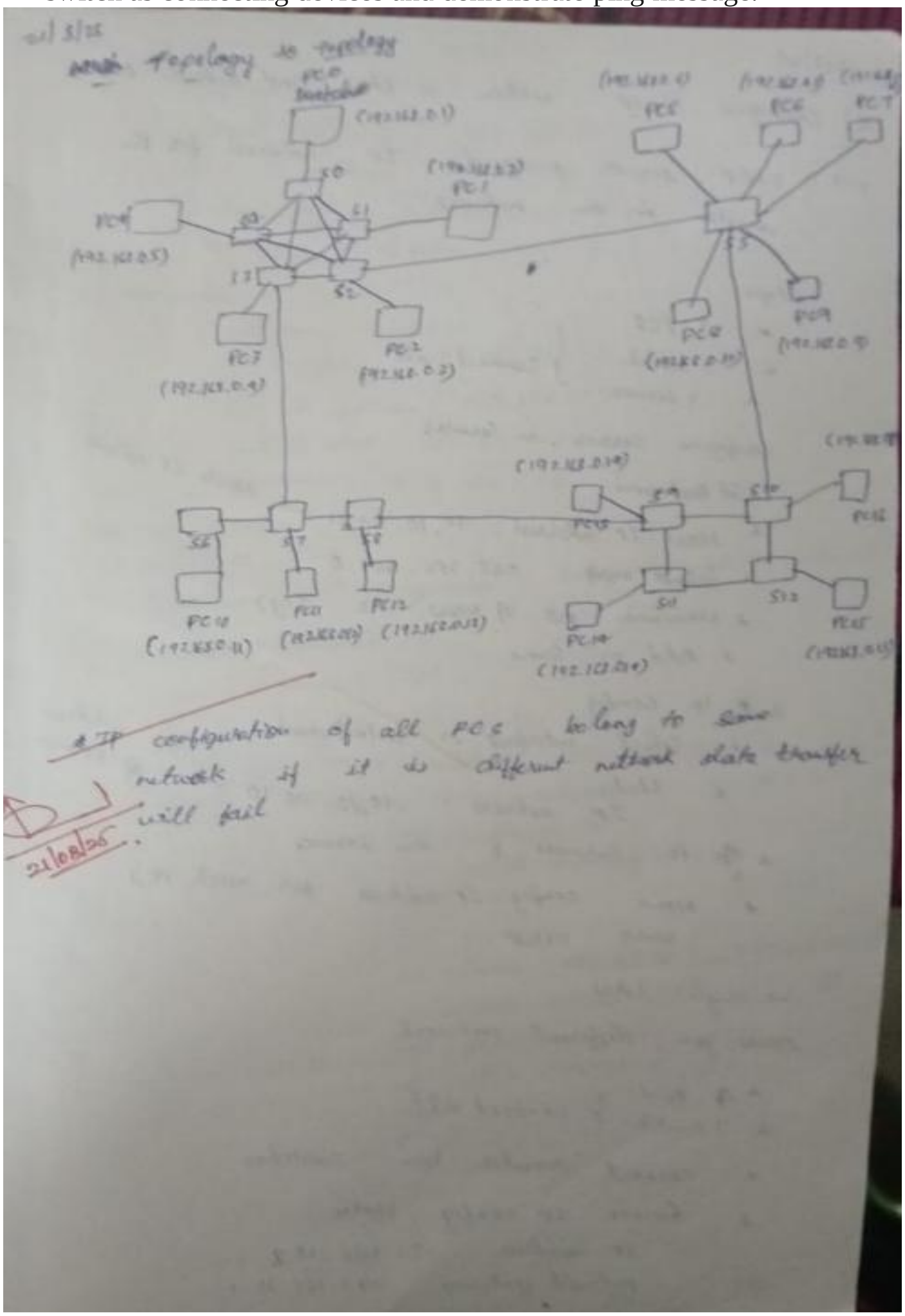
https://github.com/Arpith261/COMPUTER-NETWORKS

PART-A

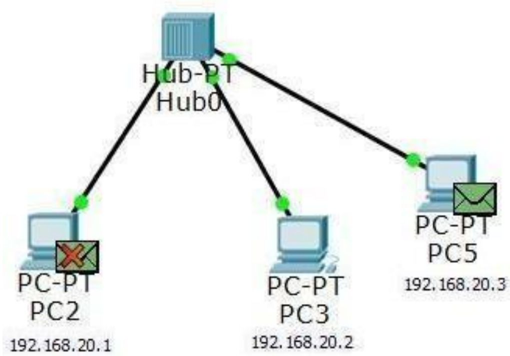
Program 1

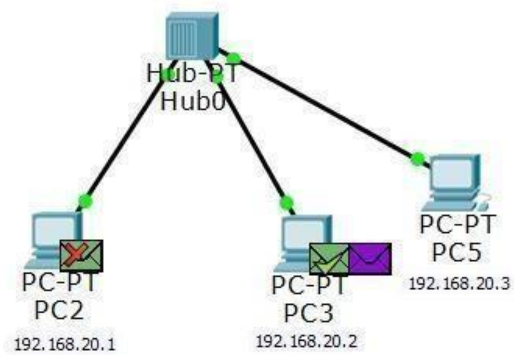
Aim of the program:

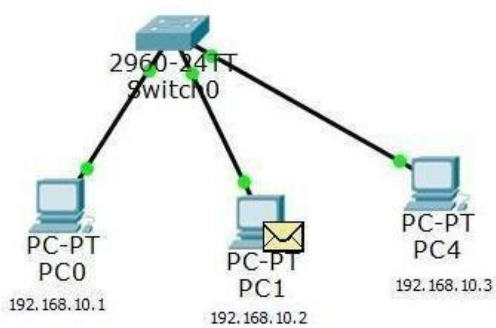
Create a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices and demonstrate ping message.

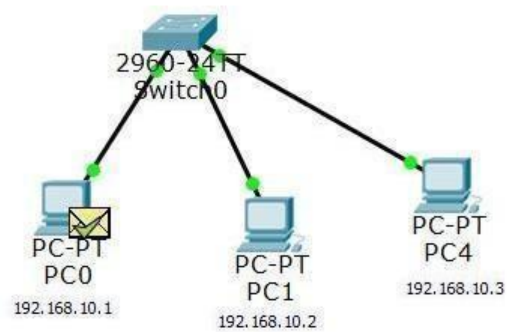


Screenshots/ Output:

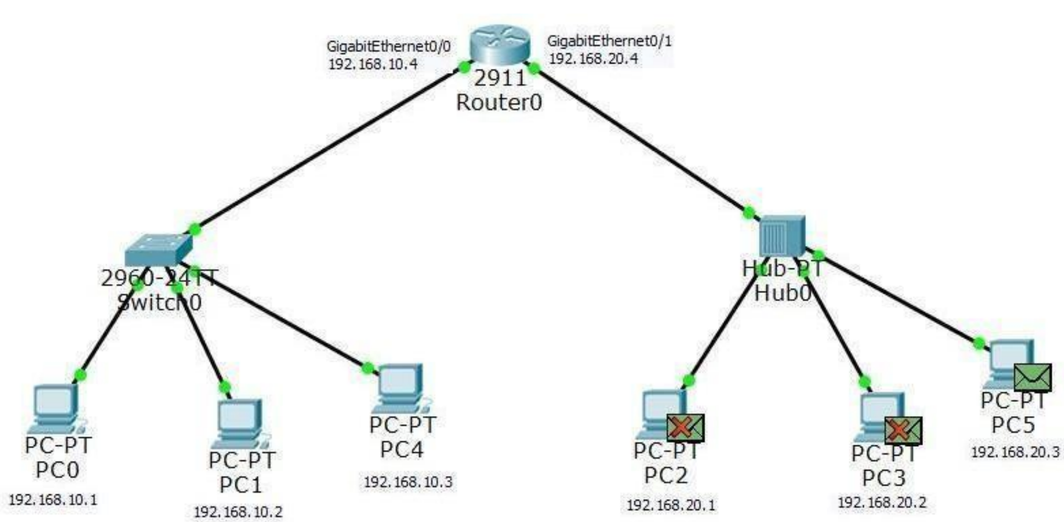


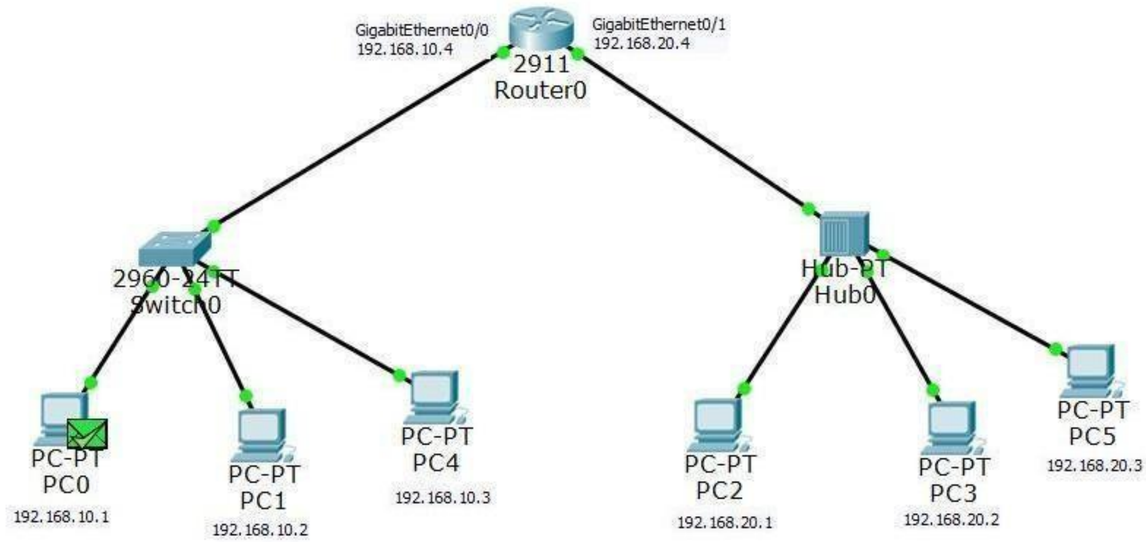






Updated topology





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 router and the Router

Procedure and topology:

11/9/18

2) Configure default route, static route to the router

create the topology
→ 3 routers
→ 3 switches
→ 3 PCs

check serial interface
Router > config → no serial interface
Router > Physical > HWIC-2T > } for all 3 routers
Switch on & off

* use serial interface to connect routers (HWIC-2T)
→ select serial DCE (in wires)

Router1 > cli

```
no ↵
enable ↵
conf t ↵
int Se0/1/0 ↵
ip address 172.16.1.1 255.255.255.252 ↵
no shutdown ↵
exit ↵
int Fa0/0
ip address 192.168.10.1 255.255.255.0 ↵
no shutdown ↵
write memory ↵
exit ↵
write memory ↵
exit ↵
wr ↵
```

Router 2 > cli

```
no ↵ ↵
enable ↵
conf t ↵
hostname R2 ↵
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 ↵
ctrl+z ↵
write memory ↵
```

Router 3 > cli

```
no ↵ ↵
enable ↵
conf t ↵
hostname R3 ↵
int Se0/1/0 ↵
ip address 172.16.2.2 255.255.255.252 ↵
no shutdown ↵
exit ↵
int Fa0/0 ↵
ip address 192.168.30.1 255.255.255.0 ↵
no shutdown ↵
exit ↵
ctrl+z ↵
wr ↵
```

PC0 > desktop > ip config > static >
ip add: 192.168.10.10
Default gateway: 192.168.10.1

PC1 > —"—
ip add: 192.168.20.10
DG: 192.168.20.1

PC2 > —"—
ip add: 192.168.30.10
DG: 192.168.30.1

Router1 > cli

```
↵
enable ↵
conf t ↵
hostname R1 ↵
```

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 ↵
ctrl+z ↵
write memory ↵

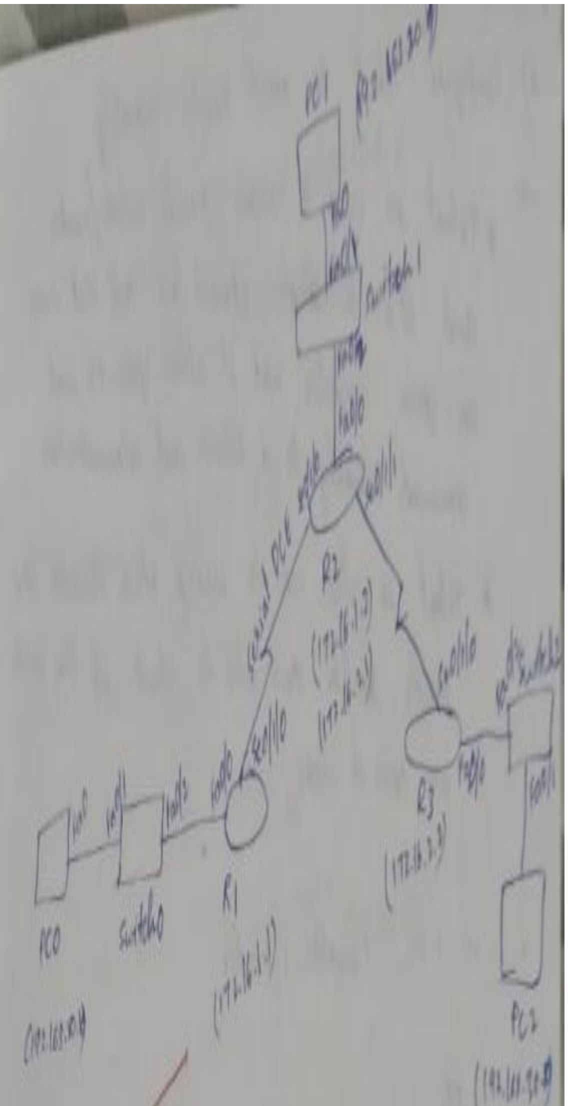
router3 > cli
no ↵
enable ↵
conf t ↵
hostname R3 ↵
int Se0/1/0 ↵
ip address 172.16.2.2 255.255.255.252 ↵
no shutdown ↵
exit ↵
int fa0/0 ↵
ip address 192.168.30.1 255.255.255.0 ↵
no shutdown ↵
exit ↵
ctrl+z ↵
wr ↵

PC0 > desktop > ip config > static >
ip add: 192.168.10.10
Default gateway: 192.168.10.1

PC1 > —"—
ip add: 192.168.20.10
DG: 192.168.20.1

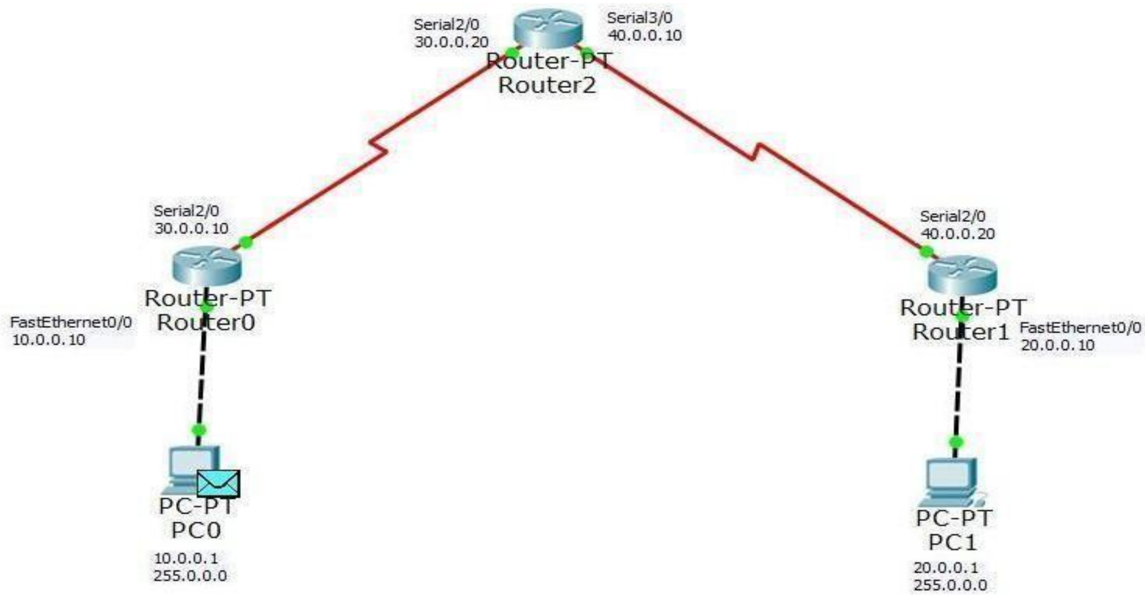
PC2 > —"—
ip add: 192.168.30.10
DG: 192.168.30.1

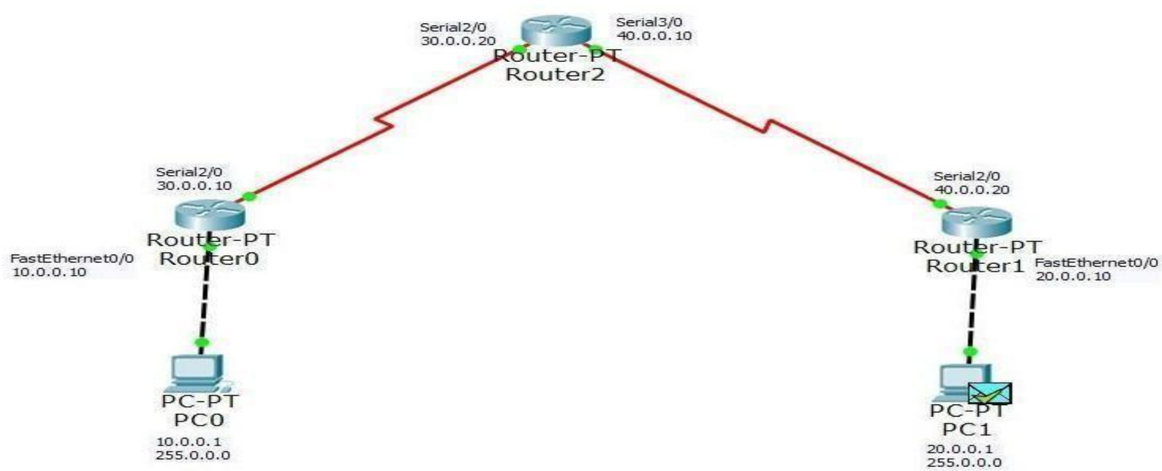
router1 > cli
↵
enable ↵
conf t ↵
hostname R1 ↵



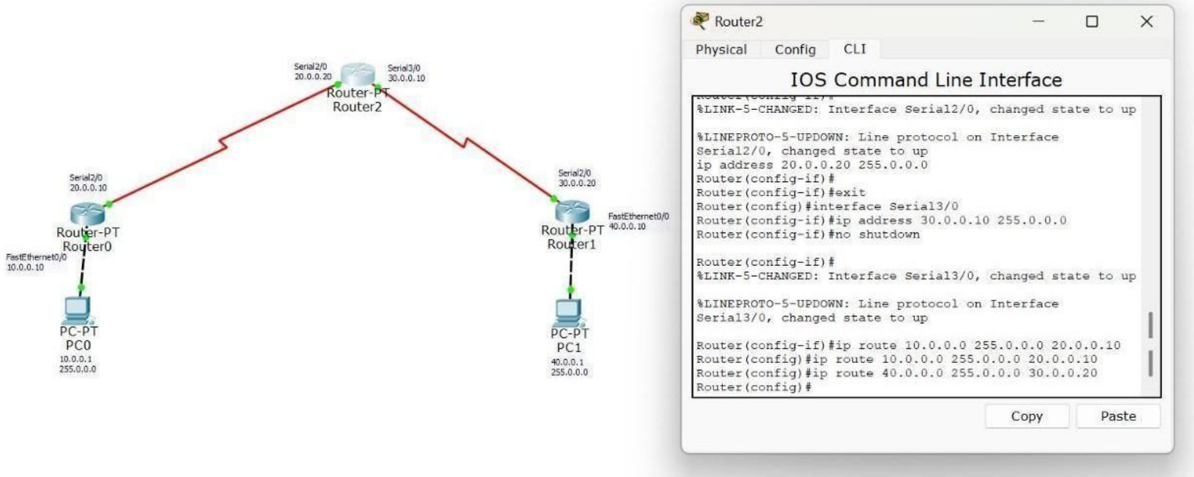
11/09/25

Sreenshots/ Output:

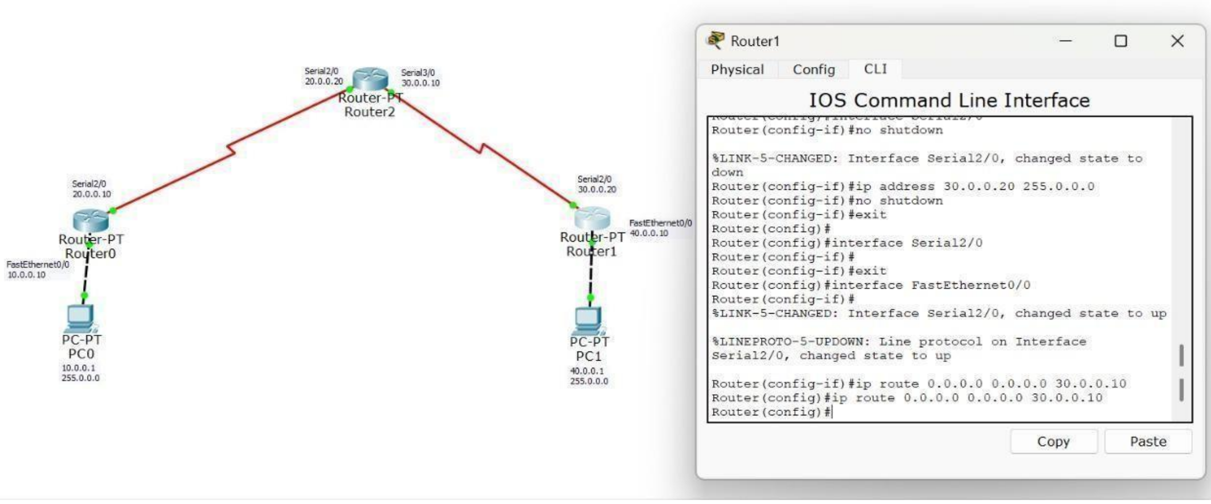




Static routing CLI commands:



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 the LAN.

Procedure and topology:

28/1/25

8) Configure DHCP within a LAN and outside LAN

→ DHCP server provide IP address for the system in a network

Steps

- 4 PCs
- 1 switch
- 1 server

} Connect all

configure Server in Services

- Set Pool name
- Start IP address : 10.10.20.11 → starts IP address
- Subnet mask : 255.255.255.0
- Maximum no. of users : 12 (eg)
- Add or Save

Go To config

- Select interface → FastEthernet
- Static
  IP adress : 10.10.20.10 → IP address of Server
- Go to Services & on Service
- Now config IP address for each PC's using DHCP

↳ single LAN

now for different network

- 4 PCs
- 1 switch

} connect all

- Connect Router b/w switches
- Server IP config static
  IP address : 192.168.10.2
  Default gateway : 192.168.10.1

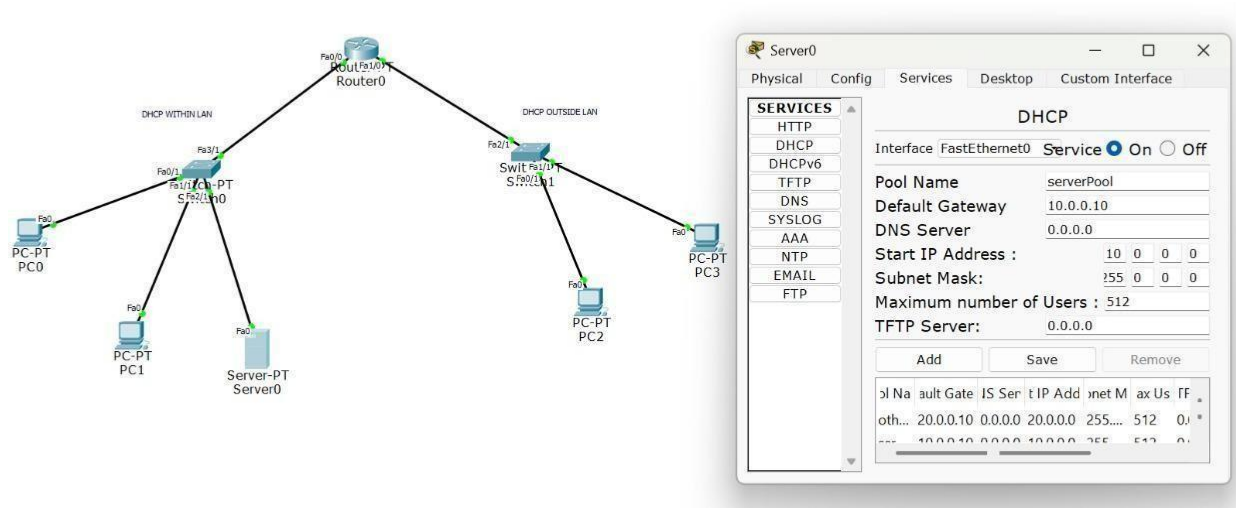
- In Services (DHCP) of Server
  Pool name : Switch1
  Default gateway : 192.168.10.1
  Start IP address : 192.168.10.3
  Subnet mask : 255.255.255.0
  Maximum no. of users : 20

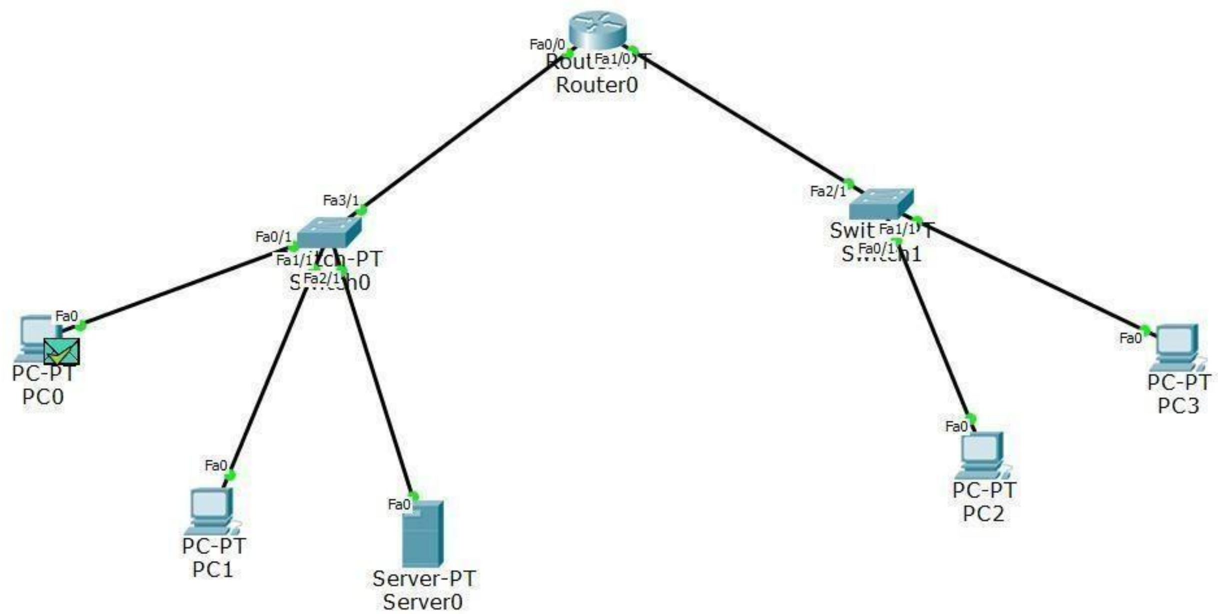
→ Again
  Pool name : Switch2
  Default gateway : 192.168.20.1
  Start IP address : 192.168.20.2
  Subnet mask : 255.255.255.0
  Maximum no. of users : 20

- In Router
  - Go to CLI
  - Continue with configuration dialog (yes/no) : no ↵

```
Router > enable ↵
Router # conf t ↵
Router (config)# int fa0/0 ↵
Router (config)# ip address 192.168.10.1 255.255.255.0 ↵
Router (config)# ip helper-address 192.168.10.2 ↵
                 no shutdown ↵
                 do write memory ↵
                 exit ↵
                 int fa 0/1
                 ip address 192.168.20.1 255.255.255.0 ↵
                 ip helper-address 192.168.10.2 ↵
                 no shutdown ↵
                 do write memory ↵
                 exit
```

Screenshots/Output:





Observation:

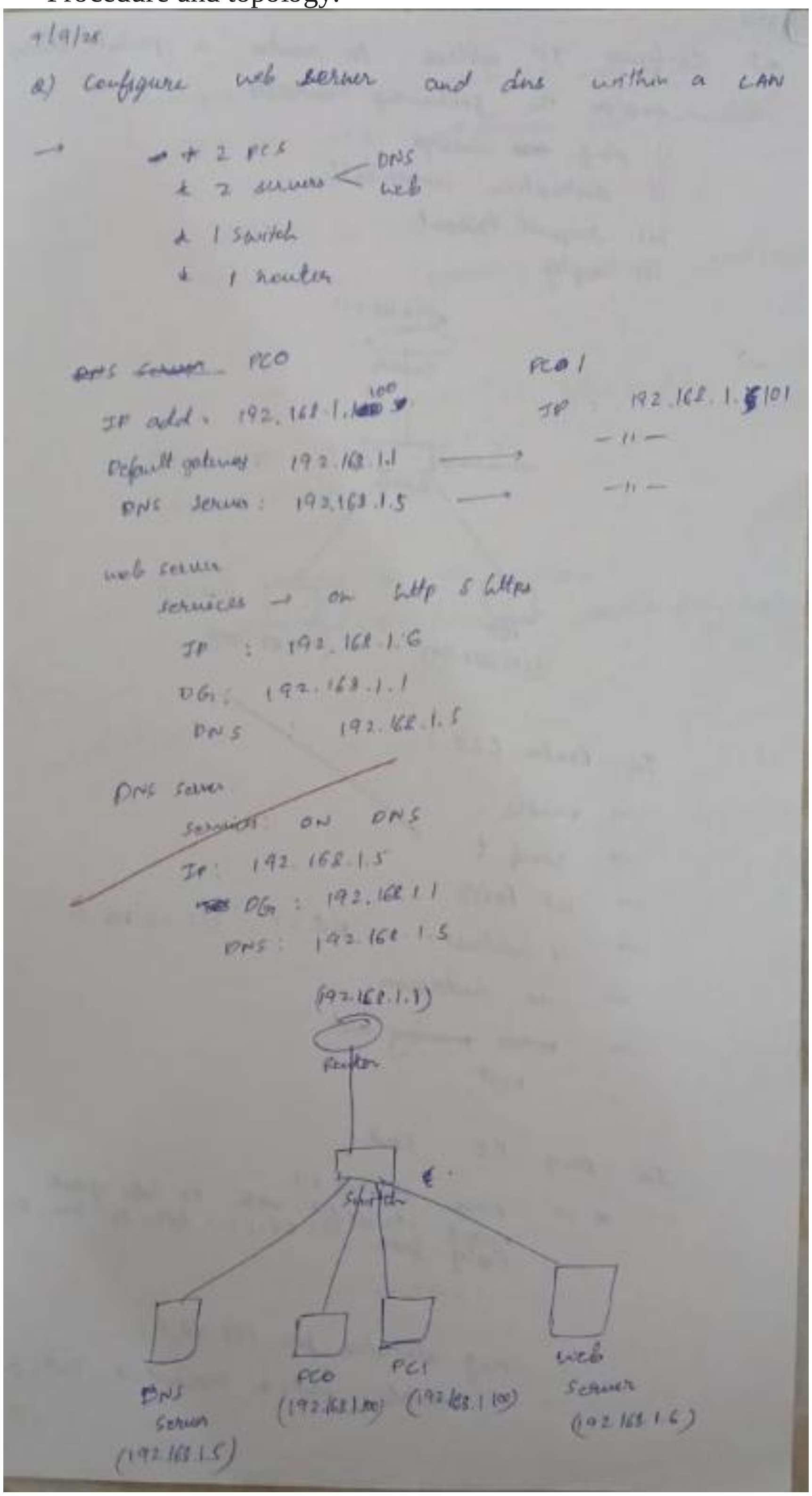
- The DHCP server successfully allocated IP addresses to clients within the LAN, confirming proper scope configuration and automatic distribution of network parameters.
- DHCP relay (IP Helper) enabled clients outside the LAN to obtain leases from the central DHCP server, demonstrating correct inter-network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

Program 4

Aim of the program:

Configure Web Server, DNS within a LAN.

Procedure and topology:


7/9/25
Q) Configure web server and dns within a LAN

→ * 2 PCs
* 2 servers < DNS / web
* 1 Switch
* 1 router

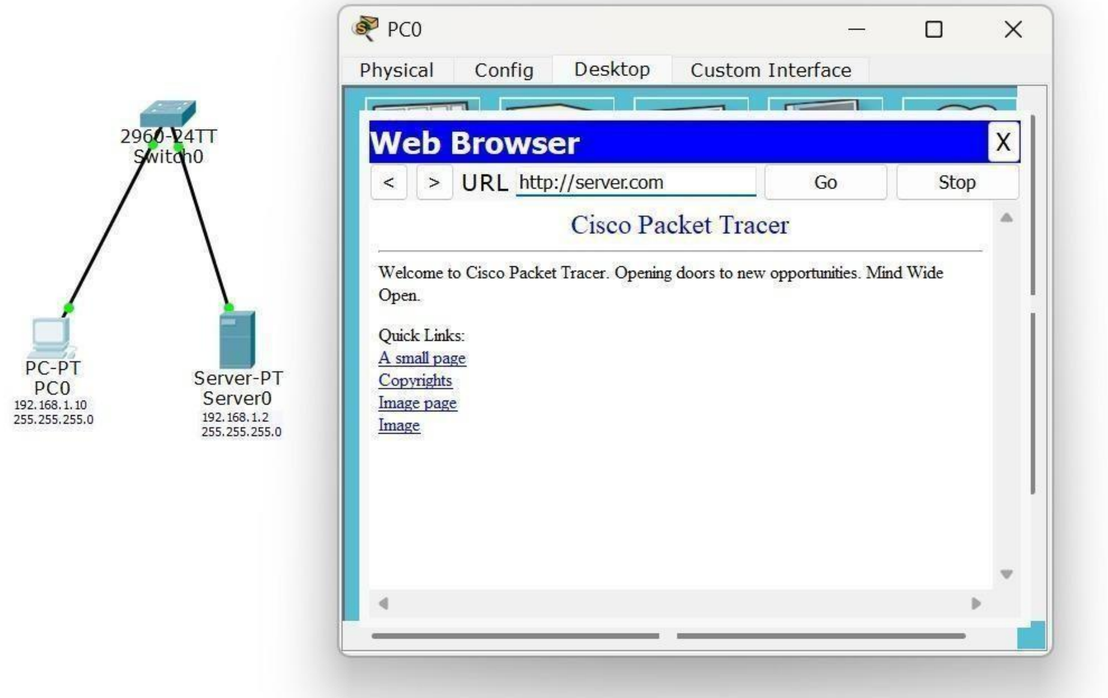
~~DNS Server~~ PC0
IP add.: 192.168.1.100
Default gateway: 192.168.1.1 →
DNS Server: 192.168.1.5 →

PC1
IP 192.168.1.101
—"—
—"—

web server
services → on http & https
IP : 192.168.1.6
DG : 192.168.1.1
DNS 192.168.1.5

DNS Server
Services ON DNS
IP: 192.168.1.5
DG : 192.168.1.1
DNS : 192.168.1.5

Screenshots/Output:



- 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:

Operation of TELNET to access the router in server room from a PC in IT office.

Procedure and topology:

9/11/25

8) Configure Telnet to access router remotely

→ * Telnet is used to access remote server/router and it's a simple command line tool that runs on your computer and it allow you to send command remotely to a server and administrator

* Telnet is also used to manage other devices like router, switch and also to check if the ports are open or close

→ 1 PC, 1 switch, 1 router

PC
IP add: 192.168.1.2
Default gateway: 192.168.1.1

Router > cli
no ↵ ↵
enable ↵
conf t ↵
hostname R1 ↵
enable secret R1 ↵ → acts as password variable name

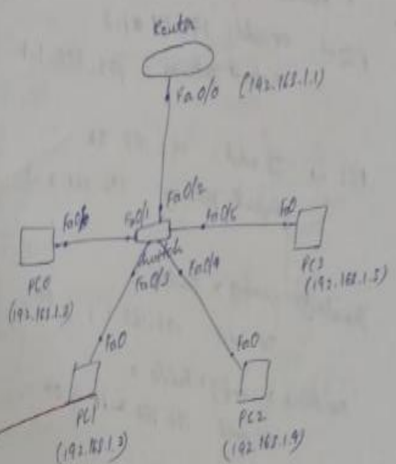
int Fa0/0 ↵
ip address 192.168.1.1 255.255.255.0 ↵
no shutdown ↵ ↵
line vty 0 5 ↵ → bandwidth range vty → virtual interface

login ↵
password tp ↵
exit ↵
exit ↵
wr ↵
show ip interface brief ↵ (shows summary of interface)

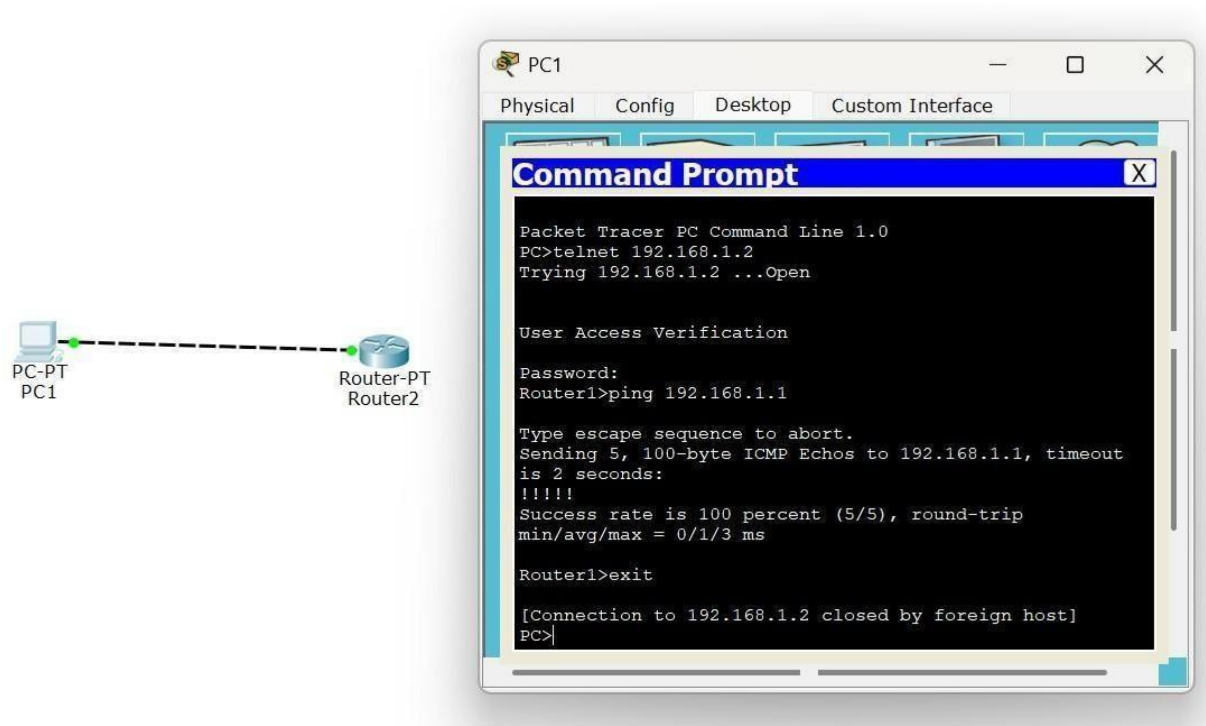
PC > cmd
ping 192.168.1.1 ↵
telnet 192.168.1.1 ↵
password: tr ↵
enable ↵
password: sp ↵

conf t ↵
int Fa0/1 ↵
ip address 192.168.1.4 255.255.255.0 ↵
do show ip interface brief ↵

do same for 3 more PCs



Screenshots/Output:


PC-PT
PC1
Router-PT
Router2
PC1
Physical
Config
Desktop
Custom Interface
Command Prompt
X
Packet Tracer PC Command Line 1.0
PC>telnet 192.168.1.2
Trying 192.168.1.2 ...Open
User Access Verification
Password:
Router1>ping 192.168.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 0/1/3 ms
Router1>exit
[Connection to 192.168.1.2 closed by foreign host]
PC>


Router2
Physical
Config
CLI
IOS Command Line Interface
Router(config-if)#npno shutdown
Router(config-if)#
%LINK-5-CHANGED: Interface FastEthernet0/0, changed state to up
%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0, changed state to up
Router(config-if)#exit
Router(config)#hostname Router1
Router1(config)#enable secret p1
Router1(config)#line vty 0 4
Router1(config-line)#login
% Login disabled on line 132, until 'password' is set
% Login disabled on line 133, until 'password' is set
% Login disabled on line 134, until 'password' is set
% Login disabled on line 135, until 'password' is set
% Login disabled on line 136, until 'password' is set
Router1(config-line)#password cisco
Router1(config-line)#exit
Copy
Paste

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:

RIP routing Protocol in Routers.

Procedure and topology:

16/10/25

Q) Configure RIP routing protocol in router

→ 2 routers, 2 switch, 2 PC

switch to PC & switch to router → Automatically choose connection type

router to router → Serial DCE

* Routers are generic to insert port (ISS8)

PC0 → IP add: 192.168.1.2
Default gateway: 192.168.1.1

PC1 → IP add: 192.168.2.2
Default gateway: 192.168.2.1

router0 > config > Fa0/0 >
IP add: 192.168.1.1 > on

router1 > config > Fa0/0 >
IP add: 192.168.2.1 > on

router0 > config > Se 2/0 >
clock rate: 64000
IP add: 10.10.0.2 > on

router1 > config > Se2/0
clock rate: 64000
IP add: 10.10.0.3 > on

Packet from PC0 to Router0 → Successfull
PC1 to Router1 → Successful
PC0 to PC1 → failed

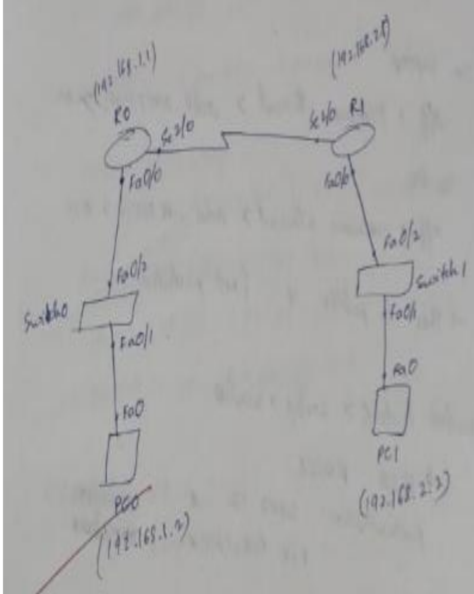
router0 > config > RIP >
network: 192.168.1.0 → add
10.0.0.0 → add

settings > save

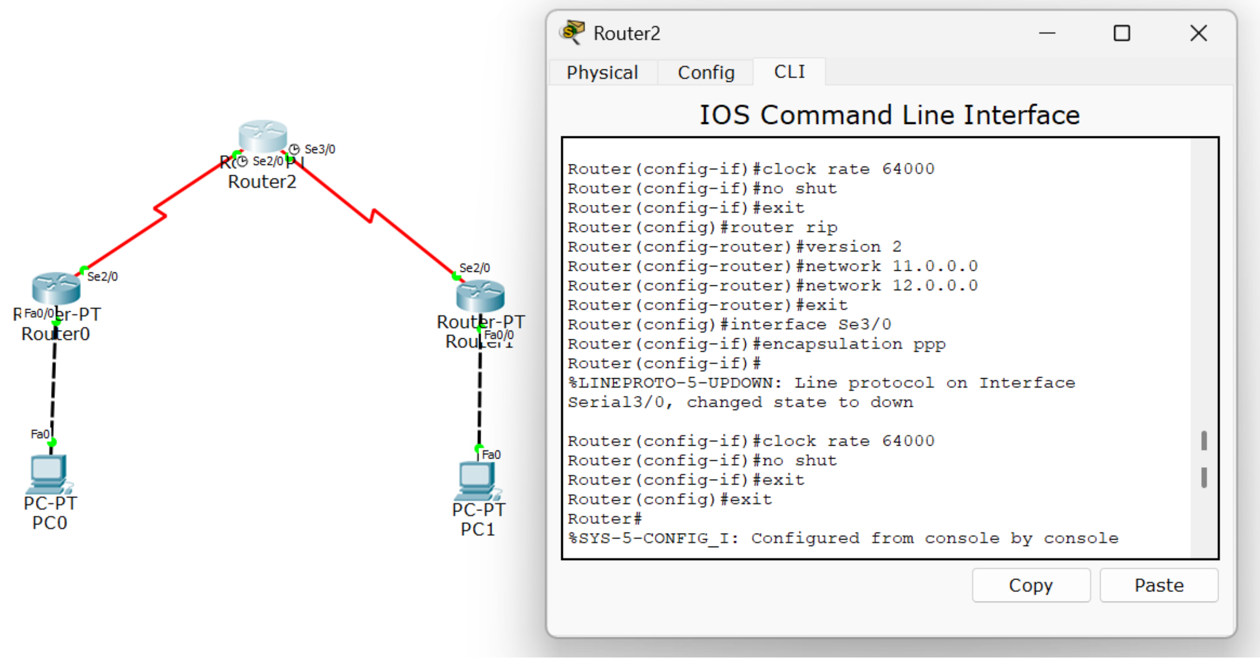
router1 > config > RIP >
network: 192.168.2.0 → add
10.0.0.0 → add

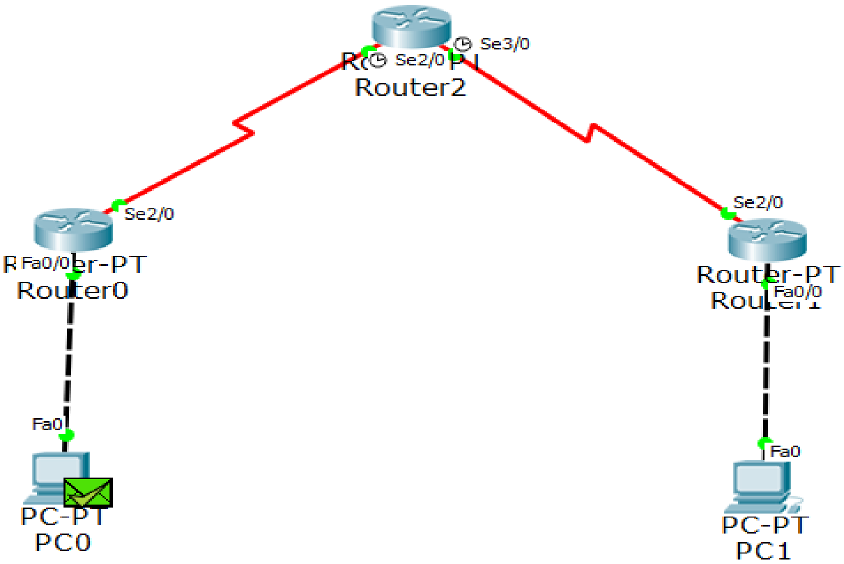
settings > save

Packet from PC0 to PC1



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:

VLAN to make the PCs communicate among a VLAN.

Procedure and topology:

20/10/25

Q) To conduct a VLAN and make PC's communicate among VLAN

→ 4 PC's, 1 switch
connects through copper straight through

1 router
connect switch & router through copper straight through

Router > config >
FastEthernet 0/0 >
IP add: 10.0.0.1 > on

PC0 > config > global settings
gateway: 10.0.0.1
Fa0/0 ⇒ IP add: 10.0.0.2

PC1 > config > global setting
gateway: 10.0.0.1
Fa0/0 ⇒ IP add: 10.0.0.3

PDU from
PC0 to PC1 ⇒ Successful
PC0 to Router ⇒ Successful
Router to PC0 ⇒ ~~Failed~~ Successful (Double click on Fire button)

PC2 > config > global setting
gateway: 20.0.0.1
Fa0/0 ⇒ IP add: 20.0.0.3

PC3 > config > global setting
gateway: 20.0.0.1
Fa0/0 ⇒ IP add: 20.0.0.2

switch > config > VLAN database
VLAN number: 2
VLAN name: VLAN > add

switch > Fa0/5
Access ✗ Trunk ✓

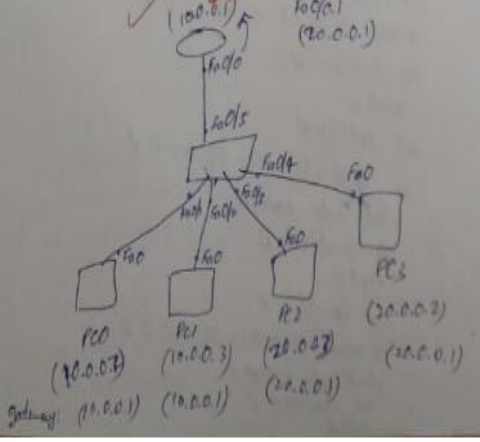
switch > config > Fa0/3 & Fa0/4
Access ✓
VLAN → 2: vlan

Router > cli > ↵

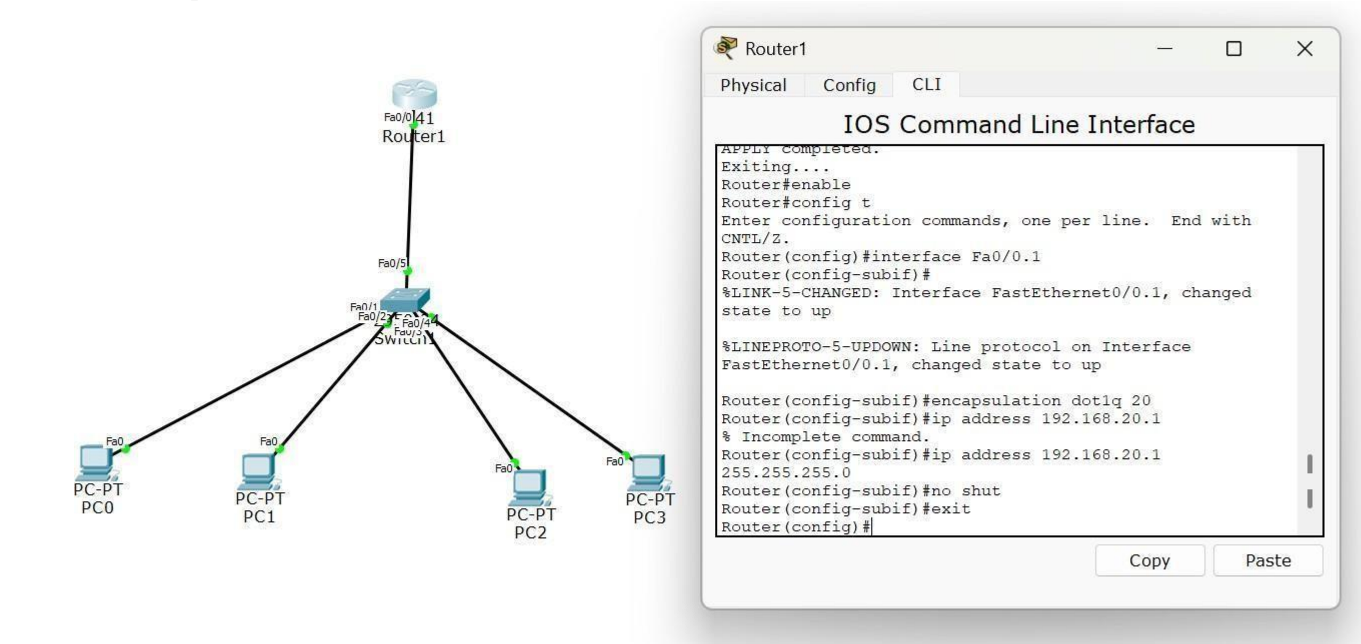
```
enable ↵
conf t ↵
int fa0/0 ↵
ip add 10.0.0.1 255.0.0.0 ↵
no shutdown ↵
exit ↵
int fa0/0.1 ↵ ↵
encapsulation dot1q 2 ↵
ip add 20.0.0.1 255.0.0.0 ↵
no shutdown ↵
exit ↵
```

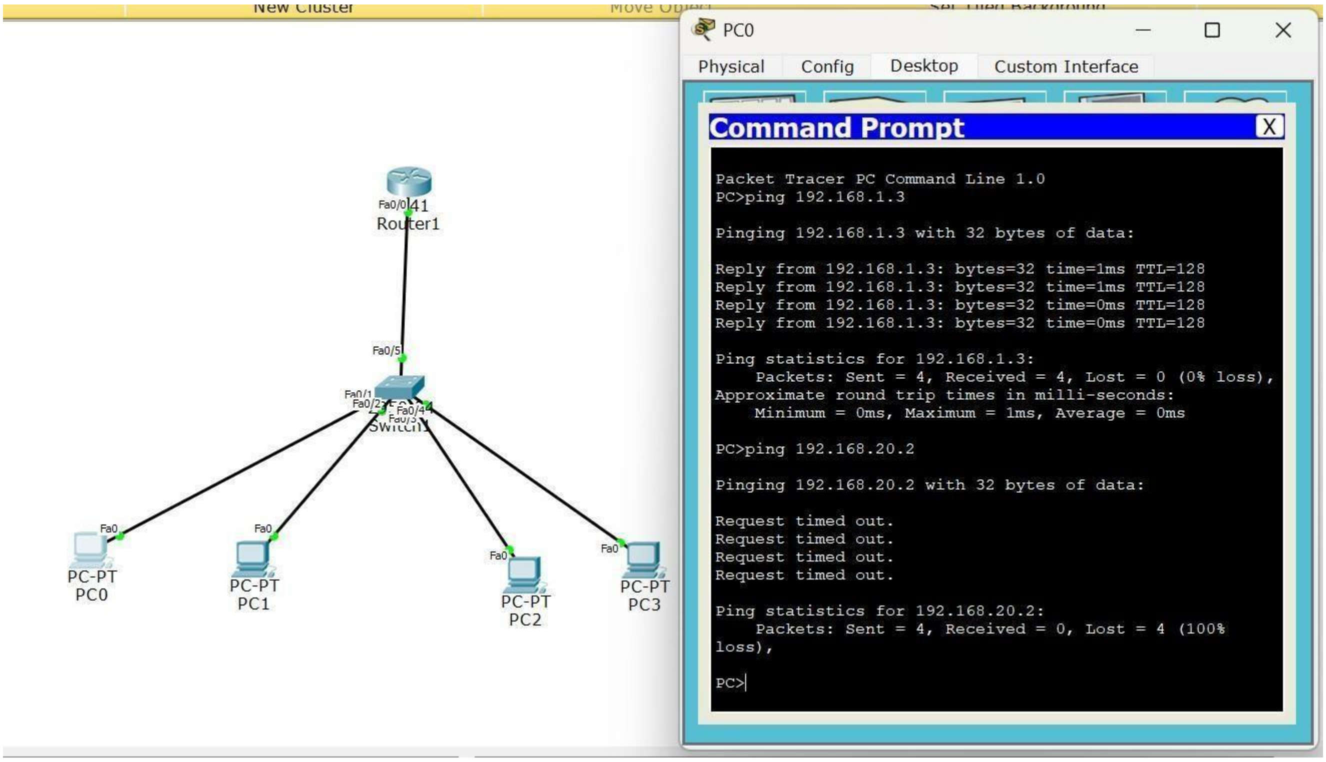
PDU from
PC0 to PC3 → Successful
PC0 to PC2 → Successful
PC1 to PC2 → Successful
PC1 to PC3 → Successful

& 20/10/2025



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:

WLAN to make the nodes communicate wirelessly.

Procedure and topology:

Q) To construct a WLAN & make the nodes communicate wirelessly

→ wirelessdevice → WRT300N
1 smartphone, 1 laptop, 1 PC

In laptop
off > remove ethernet > add WPC300N > on

In PC
off > remove ethernet > add WPC300N > on

→ This is public ip (not protected

wireless Router0 > config > wireless

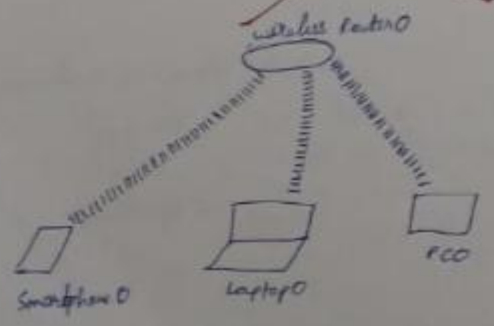
SSID: BMSCE
Authentication: WPA2-PSK
PSK Pass Phrase: Akshay123 ~~[illegible]~~

Smartphone > config > wireless
SSID:
WPA2-PSK:

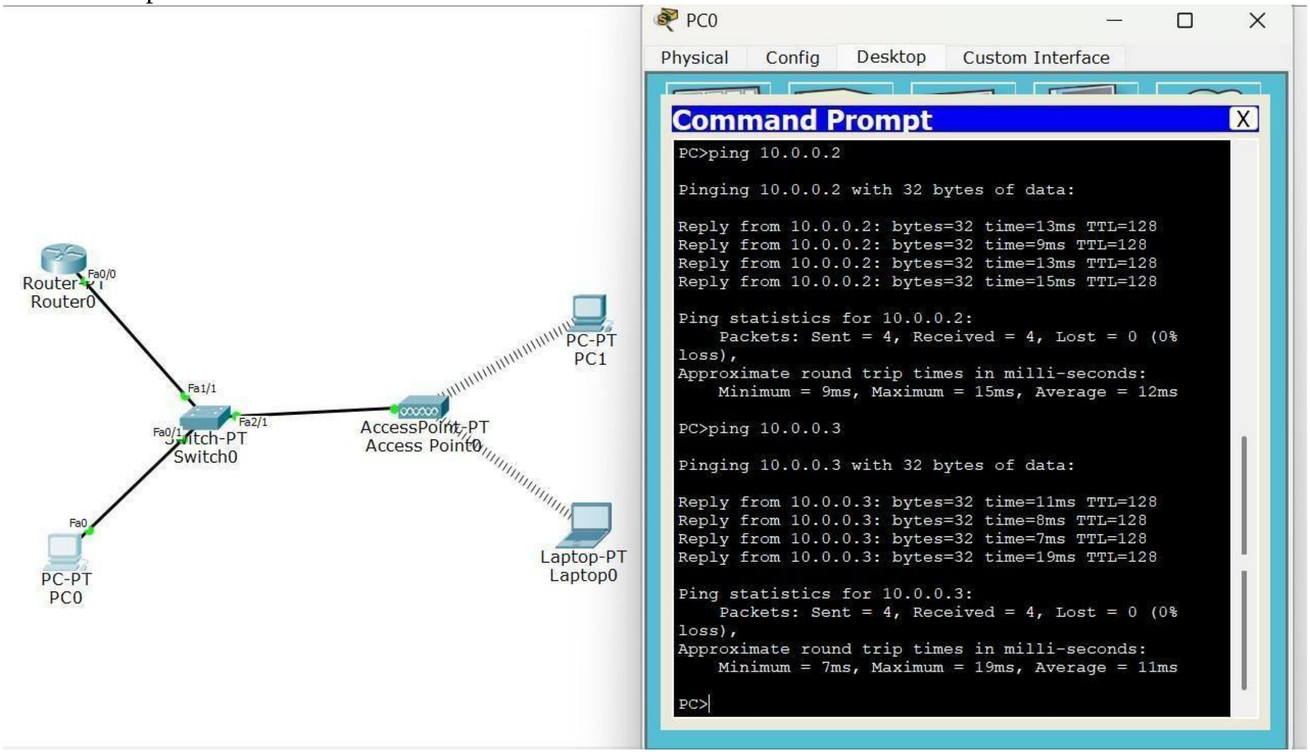
PC > config > wireless
SSID:
WPA2-PSK:

Laptop > config > wireless
SSID:
WPA2-PSK:

Enter same SSID & Pass key



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:

Simple LAN to understand the concept and operation of ARP.

Procedure and topology:

Q) To construct simple LAN & understand the concept ~~of~~ and operation of ARP

→ ARP - address resolution protocol

* It is used map an IP address to a MAC address
* It is used to get data link layer address, MAC address with the help of IP address

3PC, 1 switch, 1 server

PC0 > IP config
IP add: 192.168.11.1

PC1: IP add: 192.168.11.2

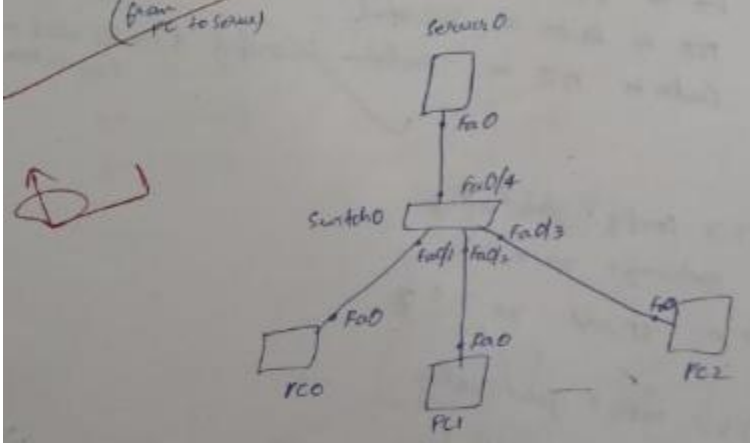
PC2: IP add: 192.168.11.3

Server0: IP add: 192.168.11.4

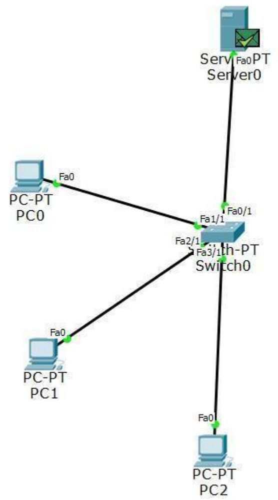
PC0 > cmd > ARP -A ↵
ping 192.168.11.4 ↵

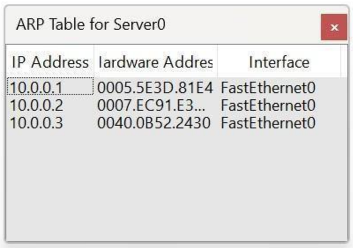
Q → ARP table of PC & server

PDU → outbound PDU details
(from PC to server)

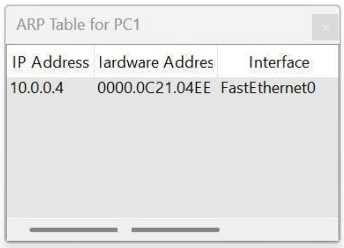


Screenshots/Output:

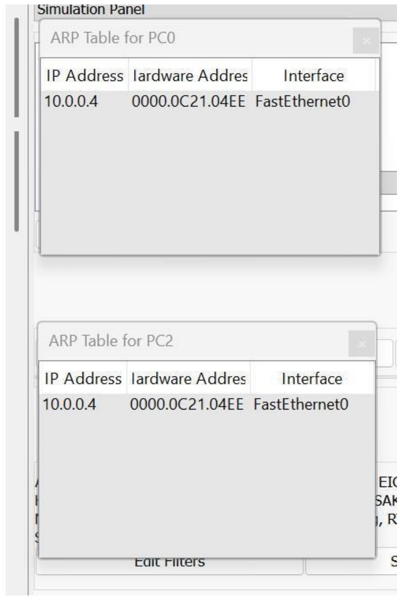



ARP Table for Server0

IP Address	Iardware Addres	Interface
10.0.0.1	0005.5E3D.81E4	FastEthernet0
10.0.0.2	0007.EC91.E3...	FastEthernet0
10.0.0.3	0040.0B52.2430	FastEthernet0


ARP Table for PC1

IP Address	Iardware Addres	Interface
10.0.0.4	0000.0C21.04EE	FastEthernet0


Simulation Panel

ARP Table for PC0

IP Address	Iardware Addres	Interface
10.0.0.4	0000.0C21.04EE	FastEthernet0

ARP Table for PC2

IP Address	Iardware Addres	Interface
10.0.0.4	0000.0C21.04EE	FastEthernet0

Edit Filters

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:

OSPF routing protocol.

Procedure and topology:

-/11/25

Q) Configure OSPF and demonstrate how packet is transferred from one node to other node

→ OSPF - Open short path first

3 routers, 3 switch, 2 PC for each

PC0
ip: 192.168.1.10
default gateway: 192.168.1.1

PC1
ip: 192.168.1.11
dg: 192.168.1.1

PC2:
ip: 192.168.2.12
dg: 192.168.2.1

PC3
ip: 192.168.2.13
dg: 192.168.2.1

PC4:
ip: 192.168.3.14
dg: 192.168.3.1

PC5
ip: 192.168.3.15
dg: 192.168.3.1

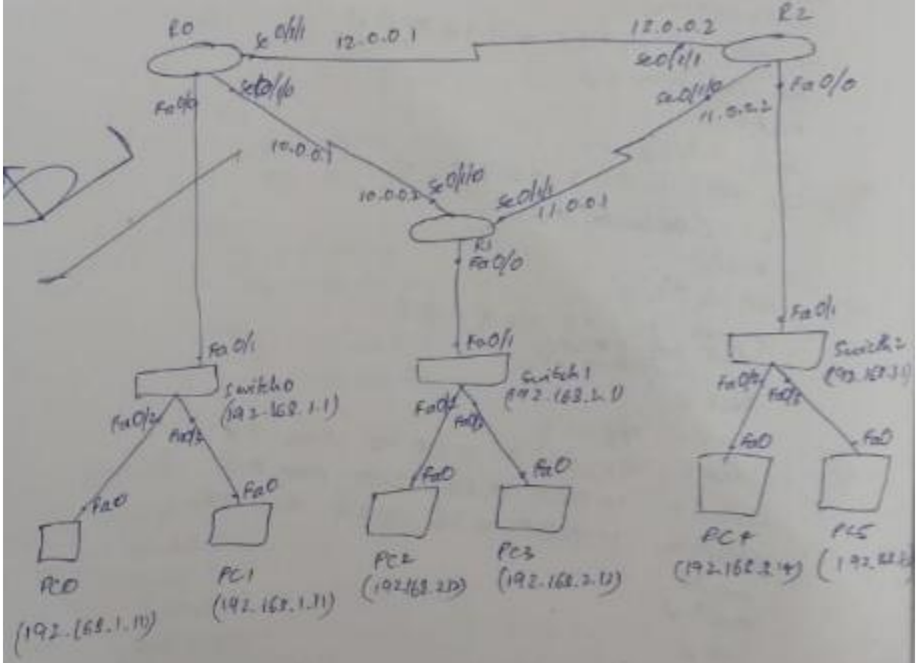
R0 > cli
no ↵
enable ↵
conf t ↵
hostname R0 ↵
int Fa0/0 ↵
ip add 192.168.1.1 255.255.255.0 ↵
no shutdown ↵ ↵
exit ↵
int Se0/1/0 ↵
ip add 10.0.0.1 255.0.0.0 ↵
clock rate 64000 ↵
no shutdown ↵
exit ↵
int Se 0/1/1 ↵
ip add 12.0.0.1 255.0.0.0 ↵
clock rate 64000 ↵
no shutdown ↵
exit ↵
router ospf 1 ↵
network 192.168.1.0 0.0.0.255 area 0 ↵
network ~~192.168~~ 10.0.0.0 0.255.255.255 area 0 ↵
network 12.0.0.0 0.255.255.255 area 0 ↵
exit ↵
exit ↵
wr ↵

Same for R1 & R2
show ip route ↵ (you should see 0(?))

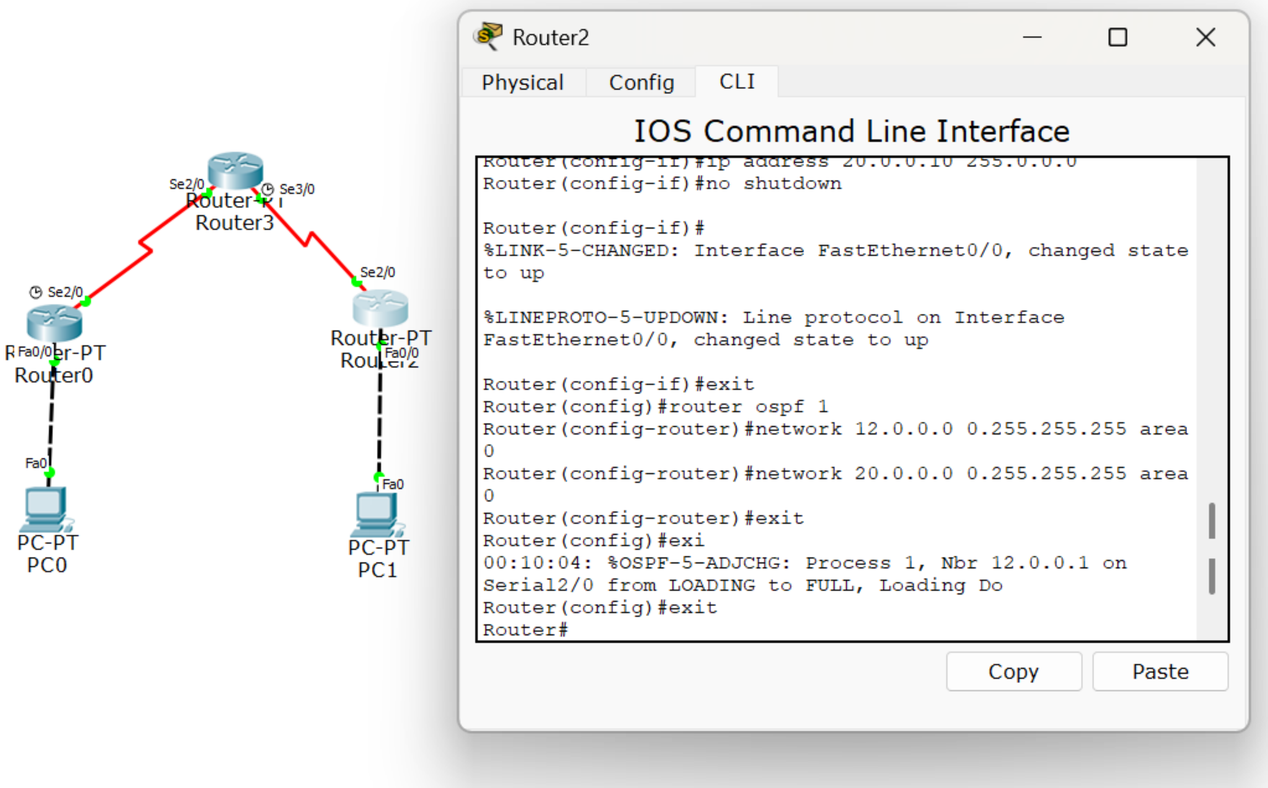
R1 > cli >
no ↵
en ↵
conf t ↵
int Fa0/0 ↵
ip add 192.168.2.1 255.255.255.0 ↵
no shutdown ↵ ↵
exit ↵
int Se0/1/0 ↵
ip add 10.0.0.2 255.0.0.0 ↵
no shutdown ↵ ↵
exit ↵
int Se0/1/1 ↵
ip add 11.0.0.1 255.0.0.0 ↵
clock rate 64000 ↵
no shutdown ↵
exit ↵
router ospf 1 ↵
network 192.168.2.0 0.0.0.255 area 0 ↵
network 10.0.0.0 0.255.255.255 area 0 ↵
network 11.0.0.0 0.255.255.255 area 0 ↵
exit ↵
exit ↵
wr ↵

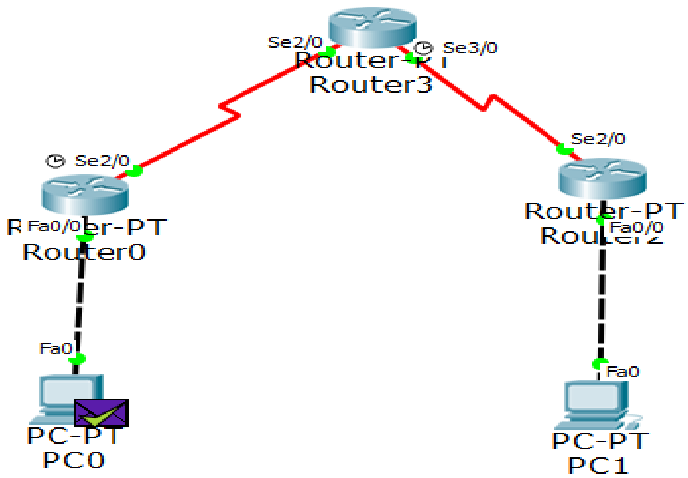
R2 > cli >
no ↵
en ↵
conf t ↵
int Fa0/0 ↵
ip add 192.168.3.1 255.255.255.0 ↵
no shutdown ↵ ↵
exit ↵
int Se0/1/0 ↵
ip add 11.0.0.2 255.0.0.0 ↵
no shutdown ↵
exit ↵

int se 0/1/1 ↵
ip add 12.0.0.2 255.0.0.0 ↵
no shutdown ↵
exit ↵
router ospf 1 ↵
network 192.168.3.0 0.0.0.255 area 0 ↵
network 11.0.0.0 0.255.255.255 area 0 ↵
network 12.0.0.0 0.255.255.255 area 0 ↵
exit ↵
exit ↵
wr ↵



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:

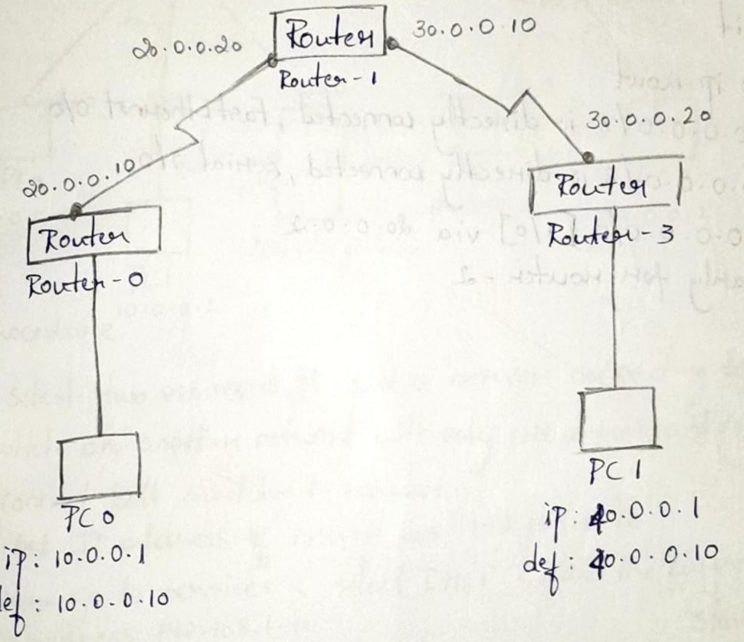
TTL/ Life of a Packet.

Procedure and topology:

Title : Packet tracing using router as the connection device

Aim: configuring default route, static route to the router

→ Static route : if we are not in same router to pass the message to the next hop



Config - steps -

>enable (router 1)

#config t

#interface fastethernet 0/0

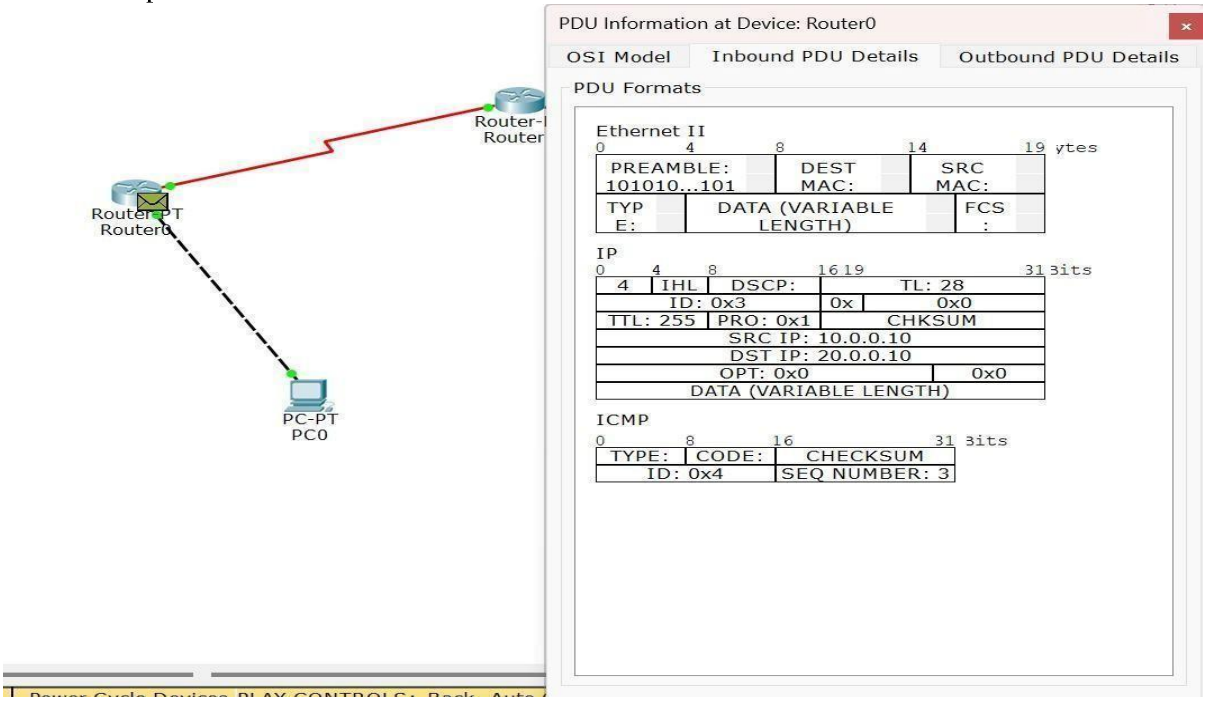
#ip address 10.0.0.10 255.0.0.0

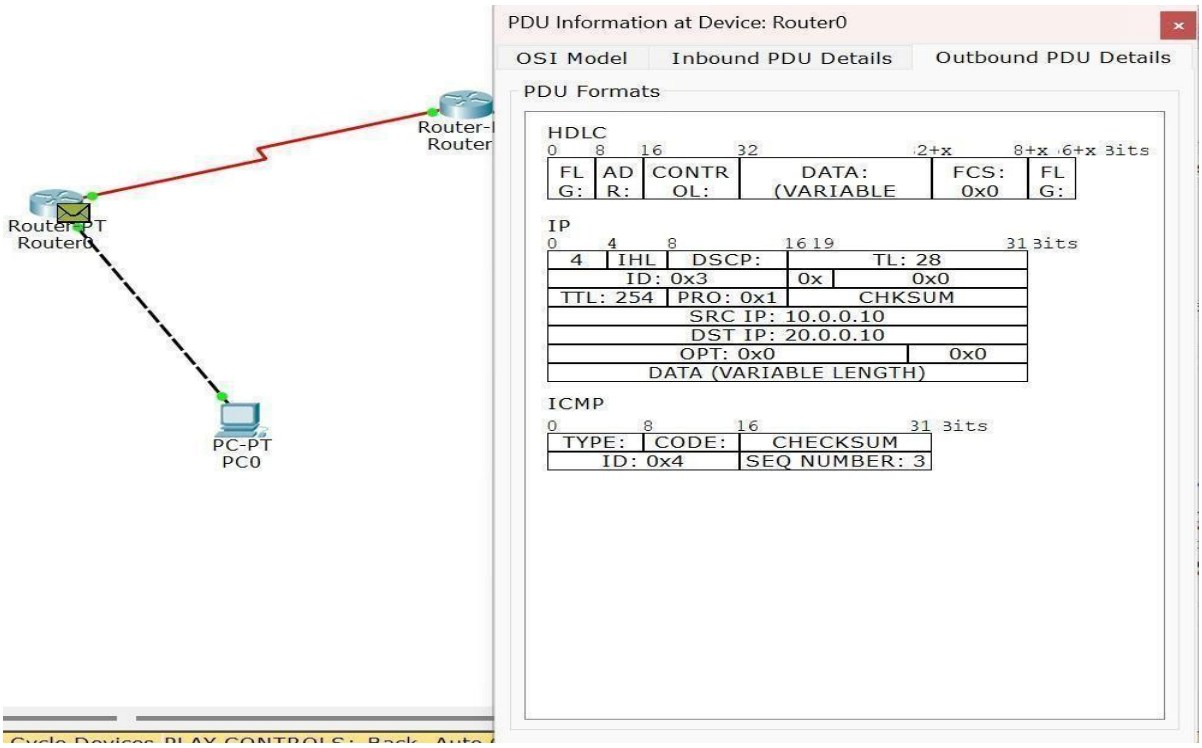
no shut

exit

Proceed with same steps for other routers

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

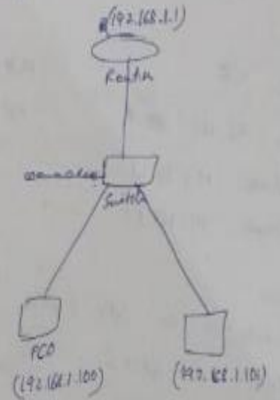
Aim of the program:

Ping responses, destination unreachable, request timed out, reply.

Procedure and topology:

Q) Configure IP address to router in packet tracer, explore the following messages

i) ping ~~ ~~ message
ii) destination unreachable
iii) request timeout
iv) reply



In Router CLI

→ enable
→ conf t
→ int Fa0/0
→ ip address 192.168.1.1 255.255.255.0
→ no shutdown
→ ~~write memory~~ exit
→ exit

In any PC cmd

→ ping 192.168.1.1
Pinging 192.168.1.1 with 32 bytes of data:
Reply from 192.168.1.1 bytes=32 time=0ms TTL=255

—||—

Ping statistics for 192.168.1.1
Packets: Sent=4, Received=4, Lost=0 (0% loss)

→ ping 192.167.1.1
Pinging 192.167.1.1 with 32 bytes of data.
Reply from 192.168.1.1 Destination host unreachable

—||—

Ping statistics for 192.167.1.1
Packets: Sent=4, Received=0 Lost=4 (100% Loss)

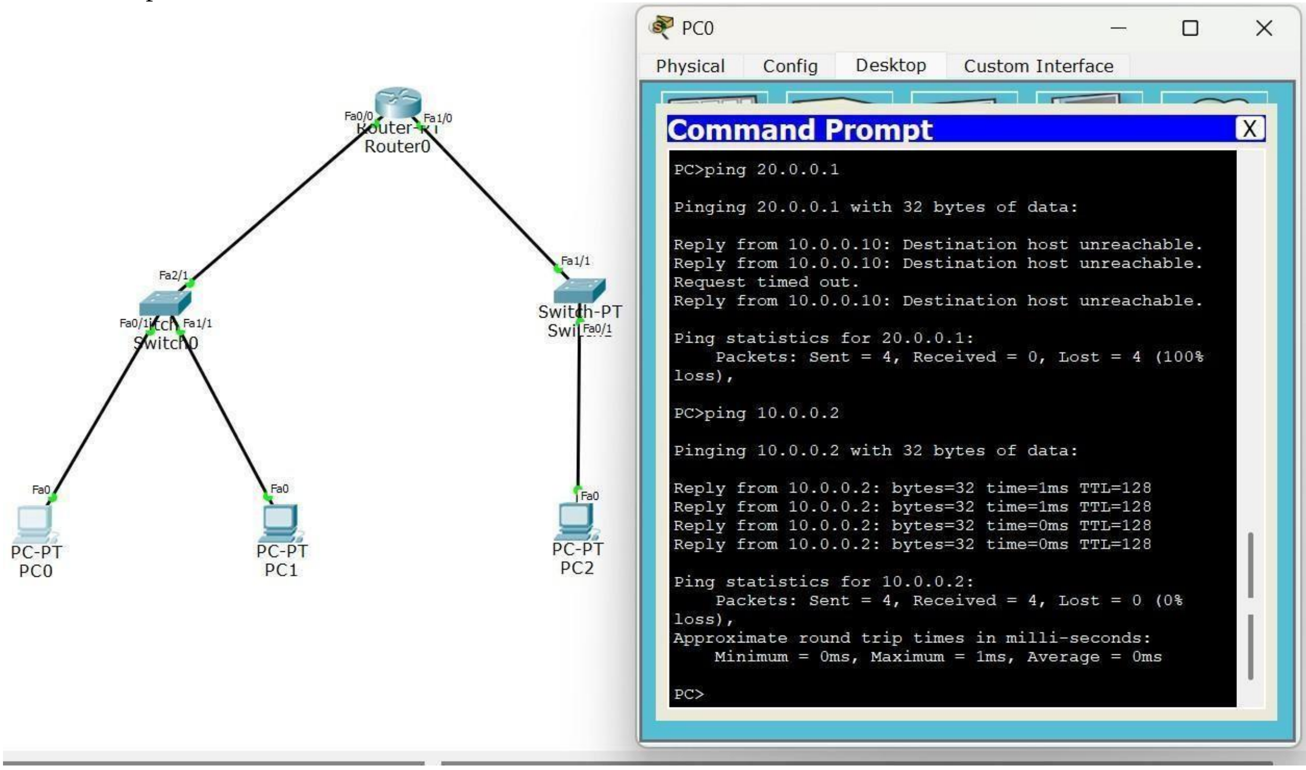
→ ping 192.168.1.200
Pinging 192.168.1.200 with 32 bytes of data:
~~Reply from 192.168.1.1~~
Request timed out

—||—

Pinging statistics for 192.168.1.200
Packets: Sent=4 Received=0, Lost=4 (100% Loss)

outlogfor

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.

PART-B

Program 13

Aim of the program:

Congestion control using Leaky bucket algorithm.

Algorithm:

13/11/25

PART - B

Q) write a program for congestion control using leaky bucket algorithm

→ Bucket size = 10 packets
output rate : 1 packet / sec
packet arrives as follows

Time (s)	packets arriving
0	6
1	4
2	8
3	1
4	0

Pseudocode :

```
Initialize bucket_size B
initialize leak_rate r
bucket 0
while (true):
    if new_packet_arrives (size):
        if bucket + size <= B:
            bucket += size
        else:
            drop(packet)
    bucket = max( 0, bucket - r * time_interval)
```

Time	packets arriving	packet sent	packets left in bucket	packets dropped
0	6	1	5	0
1	4	1	8	0
2	8	1	10	5 (dropped)
3	1	1	10	0
4	0	1	9	0
5	0	1	8	0
			⋮	
13	0	1	0	0

Code:

```
#include <iostream>
using namespace std;

int main() {
    int bucketSize, incoming, outgoing, stored = 0;

    cout << "Enter bucket size: ";
    cin >> bucketSize;

    cout << "Enter outgoing rate: ";
    cin >> outgoing;

    cout << "Enter number of incoming packets: ";
    cin >> incoming;

    int packets;

    while (incoming--) {
        cout << "\nEnter incoming packet size: ";
        cin >> packets;

        // If packet + stored exceeds bucket capacity → drop
        if (packets + stored > bucketSize) {
            cout << "Bucket overflow! " << (packets - (bucketSize - stored))
                 << " packet(s) dropped!\n";
            stored = bucketSize;
        } else {
            stored += packets;
            cout << "Packet stored. Current bucket content: " << stored << endl;
        }

        // leak outgoing packets
        if (stored > outgoing) {
            stored -= outgoing;
        } else {
            stored = 0;
        }

        cout << "Packets left in bucket after leaking: " << stored << endl;
    }

    return 0;
}
```

Packets left in bucket after leaking: 0

Output:

```
Enter bucket size (in KB): 10
Enter output rate (in KB/s): 1
Enter number of incoming packets: 5
Enter sizes of 5 incoming packets (in KB):
6 4 8 1 0

Time    Packet Size Bucket Status   Action
---------------------------------------------
1   6       6       Packet accepted
2   4       9       Packet accepted
3   8       8       Packet dropped (overflow)
4   1       8       Packet accepted
5   0       7       Packet accepted
6   -       6       Leaking...
7   -       5       Leaking...
8   -       4       Leaking...
9   -       3       Leaking...
10  -       2       Leaking...
11  -       1       Leaking...
---------------------------------------------
Bucket empty. Transmission complete.
```

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:

Error detecting code using CRC-CCITT.

Algorithm:

Q)

Sender

Data

CRC generator

CRC bits:

CRC generator = n bits

CRC bits = (n-1) bits

CRC bits = Data + (n-1) '0's / CRC generator

Data + CRC
↳ Receiver

Receiver

Eg:

Sender

Data : 10110 1

CRC generator : 1101

CRC bits - 3

```
1101) 101101000 (
      1101
      01100
       1101
       0000 1100
            1101
            00010
```

Remainder: 010

Pseudocode:

Input: message_bits, generator_bits

Append (len(generator_bits) - 1) zeros to message_bits

dividend = message_bits + zeros

while (enough bits left):

if leftmost bit == 1:

XOR dividend segment with generator

shift one bit to right

remainder = last (len(generator_bits) - 1) bits

transmitted_frame = original_message + remainder

Code:

```
#include <iostream>
#include <string>
using namespace std;

string xor1(string a, string b) {
    string result = "";
    for (int i = 1; i < b.length(); i++)
        result += (a[i] == b[i]) ? '0' : '1';
    return result;
}

string mod2div(string dividend, string divisor) {
    int pick = divisor.length();
    string tmp = dividend.substr(0, pick);

    int n = dividend.length();
    while (pick < n) {
        if (tmp[0] == '1')
            tmp = xor1(divisor, tmp) + dividend[pick];
        else
            tmp = xor1(string(pick, '0'), tmp) + dividend[pick];
        pick += 1;
    }

    if (tmp[0] == '1')
        tmp = xor1(divisor, tmp);
    else
        tmp = xor1(string(pick, '0'), tmp);

    return tmp.substr(tmp.length() - (divisor.length() - 1));
}

int main() {
    string data = "101101";
    string divisor = "1101";

    int l_key = divisor.length();
    string appended_data = data;
    for (int i = 1; i < l_key; i++) appended_data += '0';

    string remainder = mod2div(appended_data, divisor);
    cout << "Remainder: " << remainder << endl;

    string codeword = data + remainder;
    cout << "Transmitted Frame: " << codeword << endl;
}
```

Output:

```
Remainder: 010
Transmitted Frame: 101101010
```

Observation:

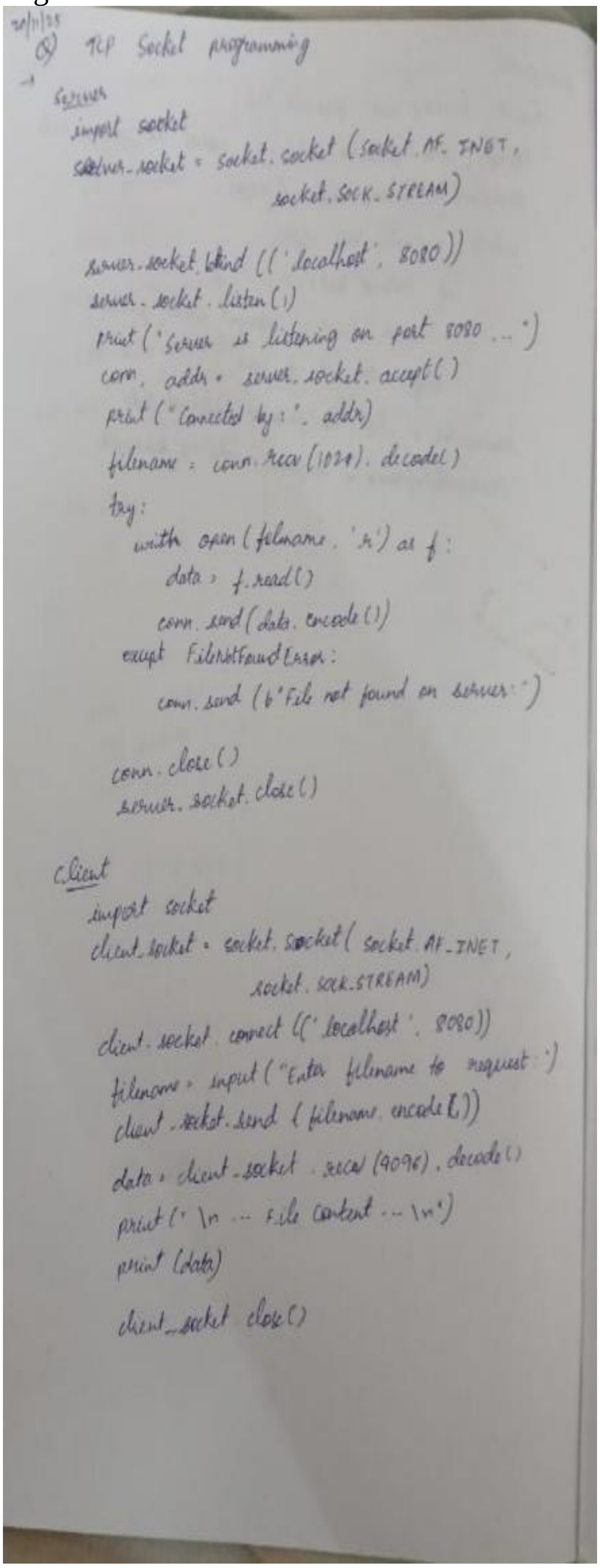
- The CRC-CCITT (16-bit) algorithm correctly generated a checksum for the transmitted data, ensuring reliable detection of single-bit and burst errors.
- Intentional error tests produced mismatched CRC values at the receiver, confirming accurate error detection through polynomial division.

Program 15

Aim of the program:

TCP File Request–Response Using Client–Server Socket Program.

Algorithm:



20/11/25

Q) TCP Socket programming

Server

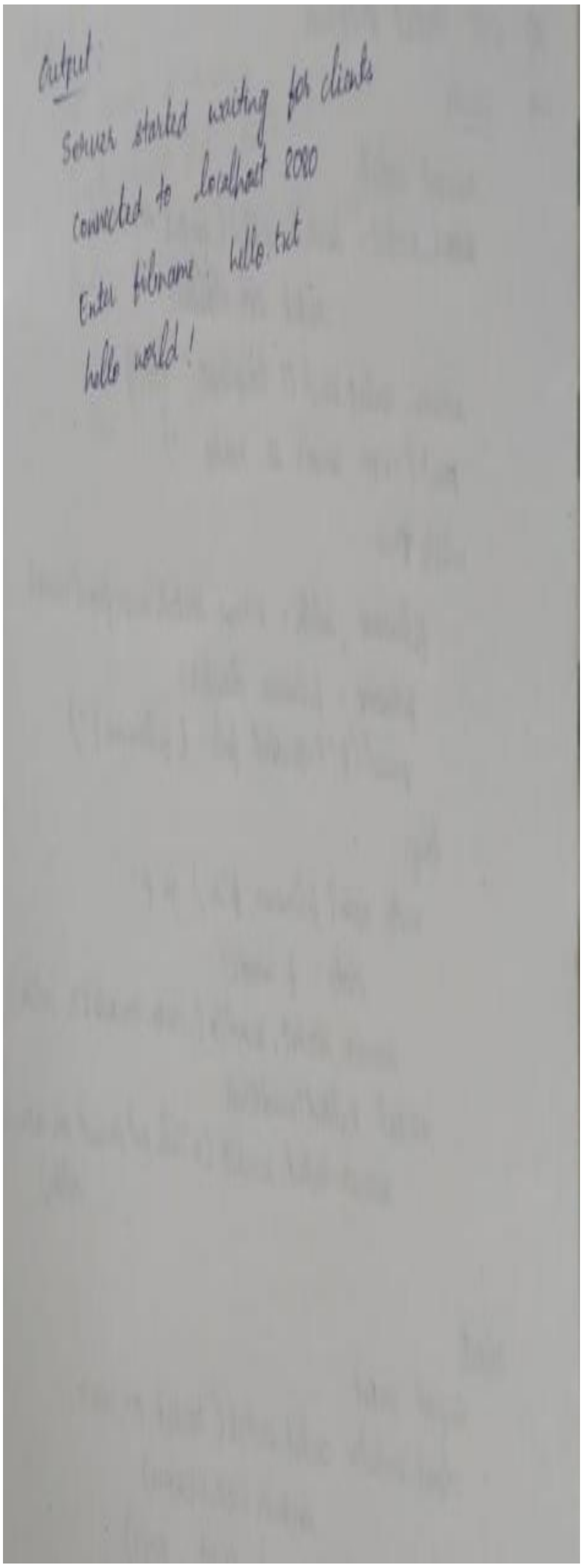
```
import socket
server_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_STREAM)

server_socket.bind(('localhost', 8080))
server_socket.listen(1)
print('Server is listening on port 8080...')
conn, addr = server_socket.accept()
print("Connected by:", addr)
filename = conn.recv(1024).decode()
try:
    with open(filename, 'r') as f:
        data = f.read()
        conn.send(data.encode())
    except FileNotFoundError:
        conn.send(b"File not found on server.")

conn.close()
server_socket.close()
```

Client

```
import socket
client_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_STREAM)
client_socket.connect(('localhost', 8080))
filename = input("Enter filename to request: ")
client_socket.send(filename.encode())
data = client_socket.recv(4096).decode()
print('\n--- File content ---\n')
print(data)
client_socket.close()
```



Output

```
Server started waiting for clients
connected to localhost 8080
Enter filename: hello.txt
hello world!
```

Code:

Client:

```
# tcp_client.py
import socket

# Step 1: Create a TCP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Step 2: Connect to the 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 content
data = client_socket.recv(4096).decode()
print("\n--- File Content ---\n")
print(data)

# Step 5: Close connection
client_socket.close()
```

Server:

```
import socket

# Step 1: Create a TCP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Step 2: Bind it to an address and port
server_socket.bind(('localhost', 8080))

# Step 3: Listen for client connection
server_socket.listen(1)
print("Server is listening on port 8080...")

# Step 4: Accept client connection
conn, addr = server_socket.accept()
print("Connected by:", addr)

# Step 5: Receive file name from client
filename = conn.recv(1024).decode()

try:
    # Step 6: Open file and read contents
    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:

```
PS D:\python> python tcp_server.py
Server is listening on port 8080...
Connected by: ('127.0.0.1', 61136)
PS D:\python>
```

```
PS D:\python> python tcp_client.py
Enter filename to request: hello.txt

--- File Content ---

Hello World!
PS D:\python>
```

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:

UDP File Request–Response Using Client–Server Socket Program.

Algorithm:

20/11/25

8) UDP Socket program

→ Server

```
import socket
server_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_DGRAM)

server_socket.bind(('localhost', 8081))
print("UDP server is ready...")
while True:
    filename, addr = server_socket.recvfrom(1024)
    filename = filename.decode()
    print(f"Requested file: {filename}")

try:
    with open(filename, 'r') as f:
        data = f.read()
    server_socket.sendto(data.encode(), addr)
except FileNotFoundError:
    server_socket.sendto(b'File not found on server',
                                  addr)
```

client

```
import socket
client_socket = socket.socket(socket.AF_INET,
                    socket.SOCK_DGRAM)
server_address = ('localhost', 8081)
filename = input("Enter filename to request:")
client_socket.sendto(filename.encode(), server_addr)
data, _ = client_socket.recvfrom(4096)
print('\n -- File content -\n')
print(data.decode())
client_socket.close()
```

output

```
UDP server is running
client request file: hello.txt

Enter filename to request: hello.txt

Hello world!
This is a test file
```

Code:

Client:

```
# 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.encode(), server_address)

# Step 3: Receive response
data, _ = client_socket.recvfrom(4096)
print("\n--- File Content ---\n")
print(data.decode())

# Step 4: Close socket
client_socket.close()
```

Server:

```
# udp_server.py
import socket

# Step 1: Create UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Step 2: Bind it to an address
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()
    print(f"Requested file: {filename}")

    try:
        # Step 4: Read file and send data
        with open(filename, 'r') as f:
            data = f.read()
            server_socket.sendto(data.encode(), addr)
    except FileNotFoundError:
        server_socket.sendto(b"File not found on server.", addr)
```

Output:

```
PS D:\python> python udp_server.py
UDP Server is ready...
Requested file: hello.txt
```

```
PS D:\python> python udp_client.py
Enter filename to request: hello.txt

--- File Content ---

Hello World!
PS D:\python>
```

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.