# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB RECORD**

# Computer Network Lab (23CS5PCCON)

***Submitted by***

**Spurthi Reddy P (1BM23CS338)**

***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 **Spurthi Reddy P (1BM23CS338),** 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<br>Assistant Professor<br>Department of CSE, BMSCE | Dr. Kavitha Sooda<br>Professor & HOD<br>Department of CSE, BMSCE |

# Index



Github Link:
https://github.com/Spurthi-338/CN_Lab-338.git

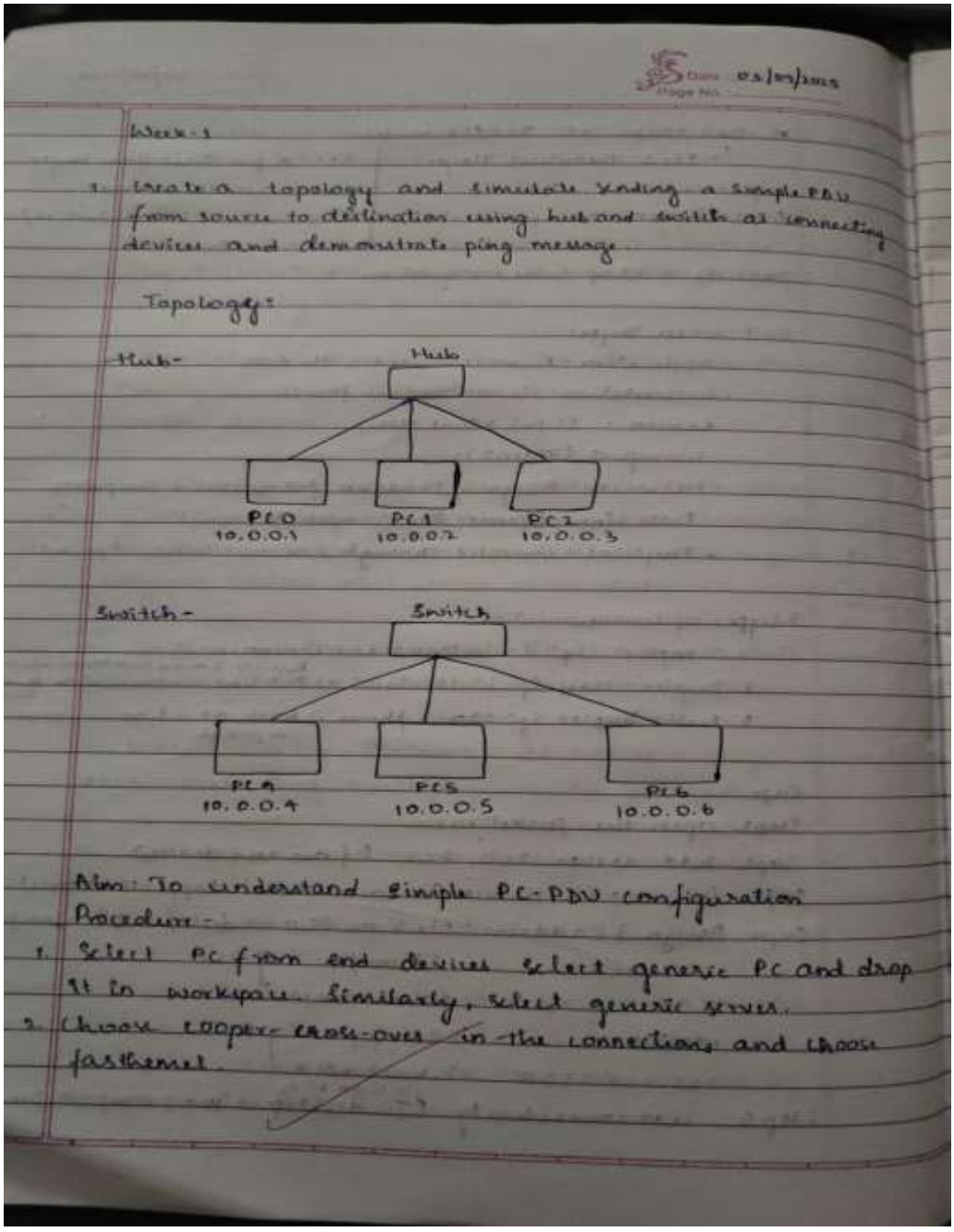
**CYCLE 1:**

**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.

Procedure and topology:



3. Click on server and choose fastethernet.
4. Click on PC and go to configuration tab. Set IP address to 10.0.0.1 and click on SubnetMask.
5. Repeat same step and set IP address for server.
6. In Simulation mode, in edit filters click only ICMP.
7. Add a simple PDU from PC to server.
8. Click on Autocapture/Play.

Hub-

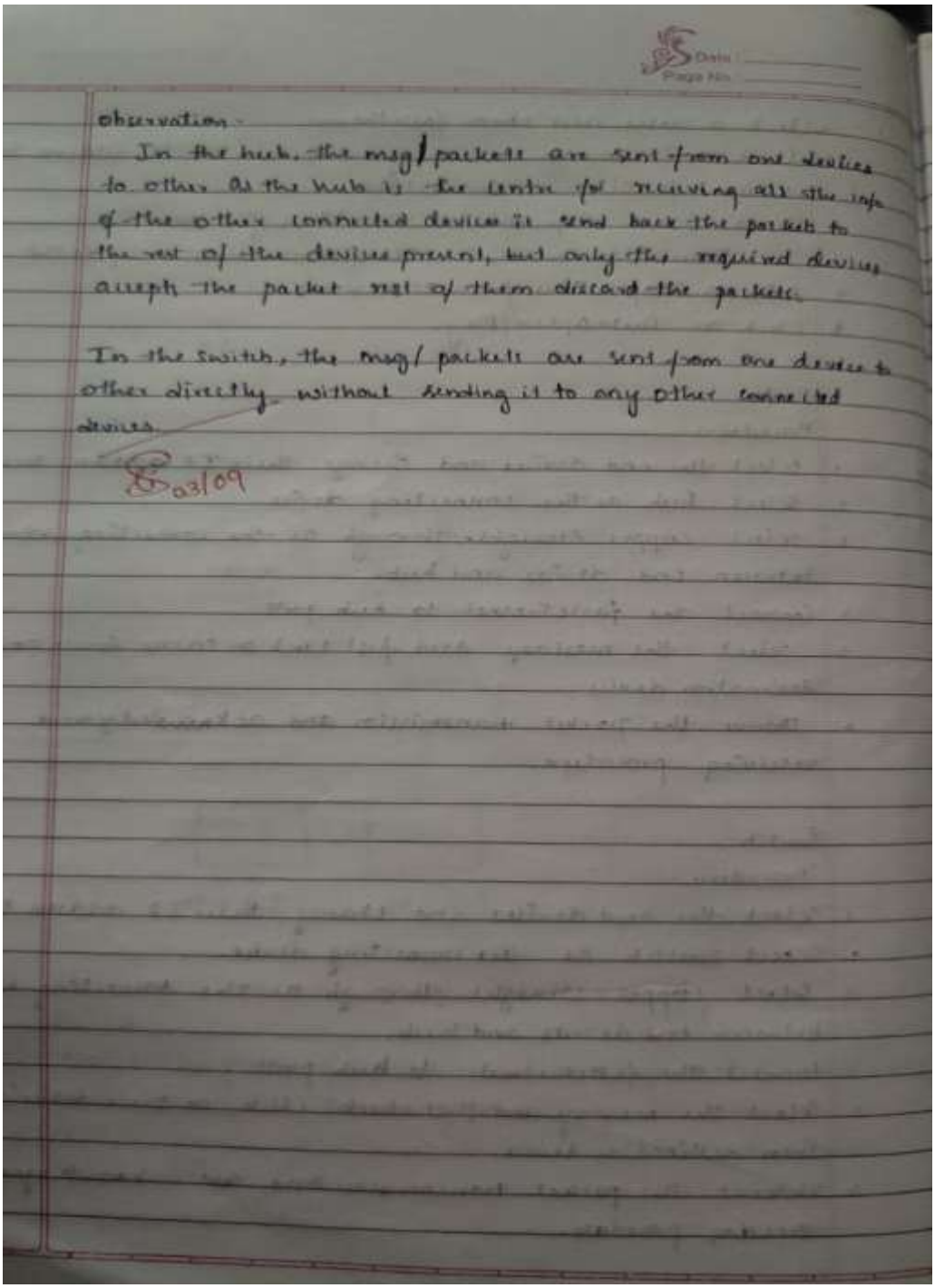
Procedure-

1. Select the end devices and change their IP address suitably.
2. Select hub as the connecting device.
3. Select copper straight-through as the connecting wire between end devices and hub.
4. Connect the fastethernet to hub ports.
5. Select the message and first click on source device and destination device.
6. Observe the packet transmission and acknowledgement receiving procedure.

Switch-

Procedure-

1. Select the end devices and change their IP address suitably.
2. Select switch as the connecting device.
3. Select copper straight-through as the connecting wire between end devices and hub.
4. Connect the fastethernet to hub ports.
5. Select the message and first ~~check~~ click on source device and then destination device.
6. Observe the packet transmission and see acknowledgement receiving procedure.



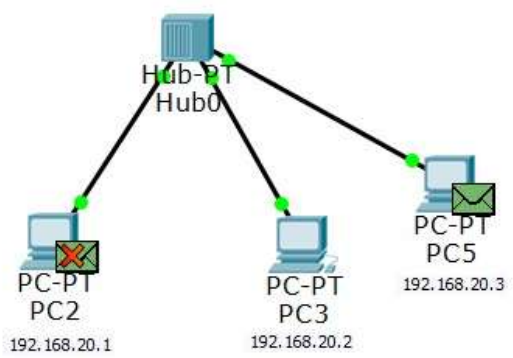
Observation:

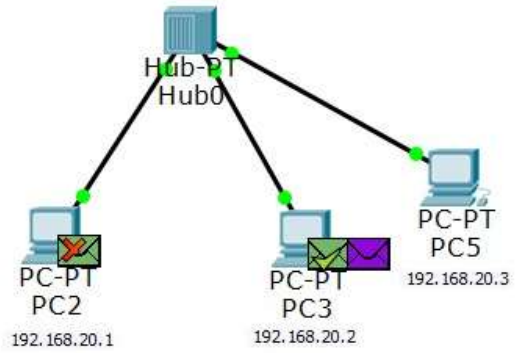
In the hub, the msg/ packets are sent from one devices to other. As the hub is the centre for receiving all the info of the other connected devices it send back the packets to the rest of the devices present, but only the required devices accept the packet rest of them discard the packets.

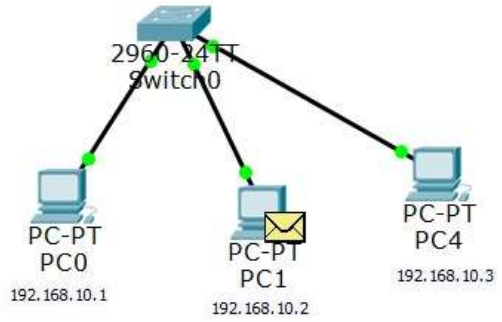
In the switch, the msg/ packets are sent from one devices to other directly without sending it to any other connected devices.

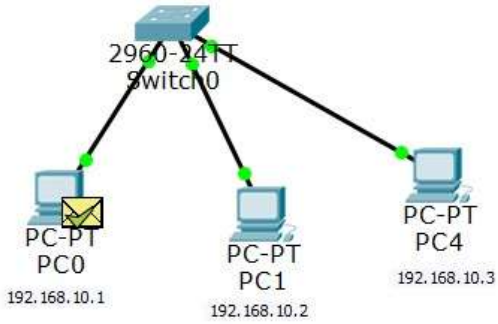
03/09

Screenshots/ Output:

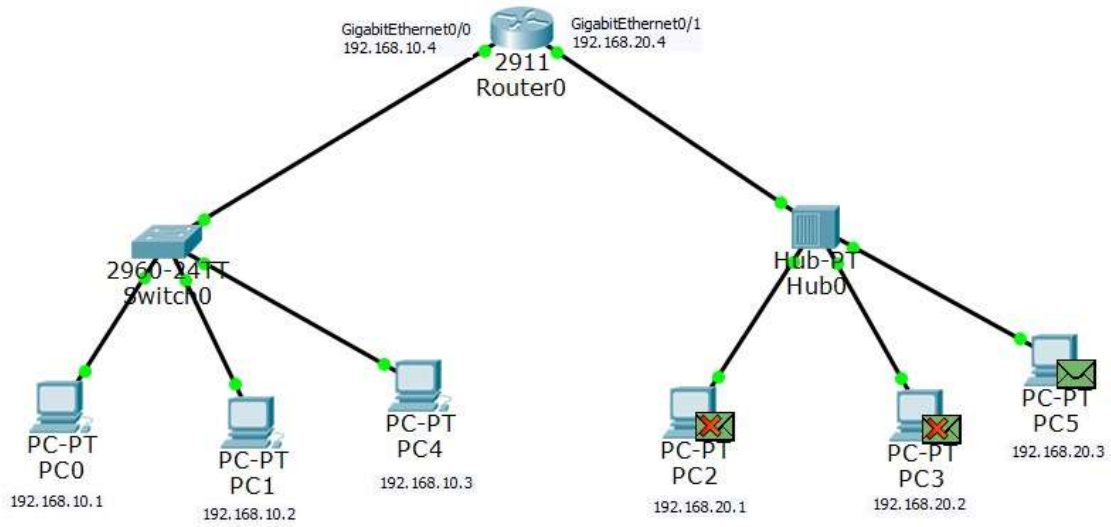


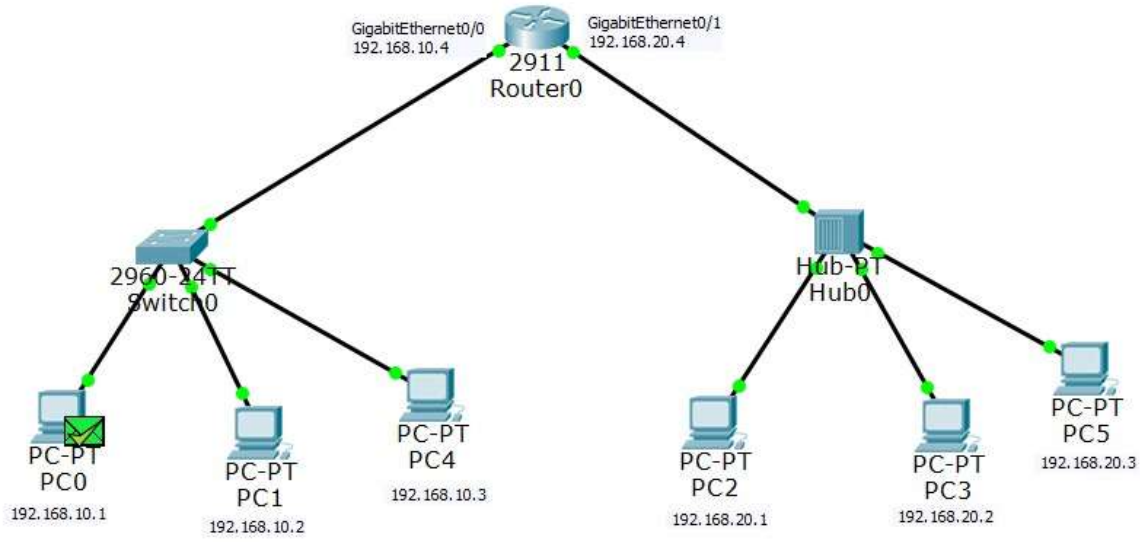




Updated topology






Observation:

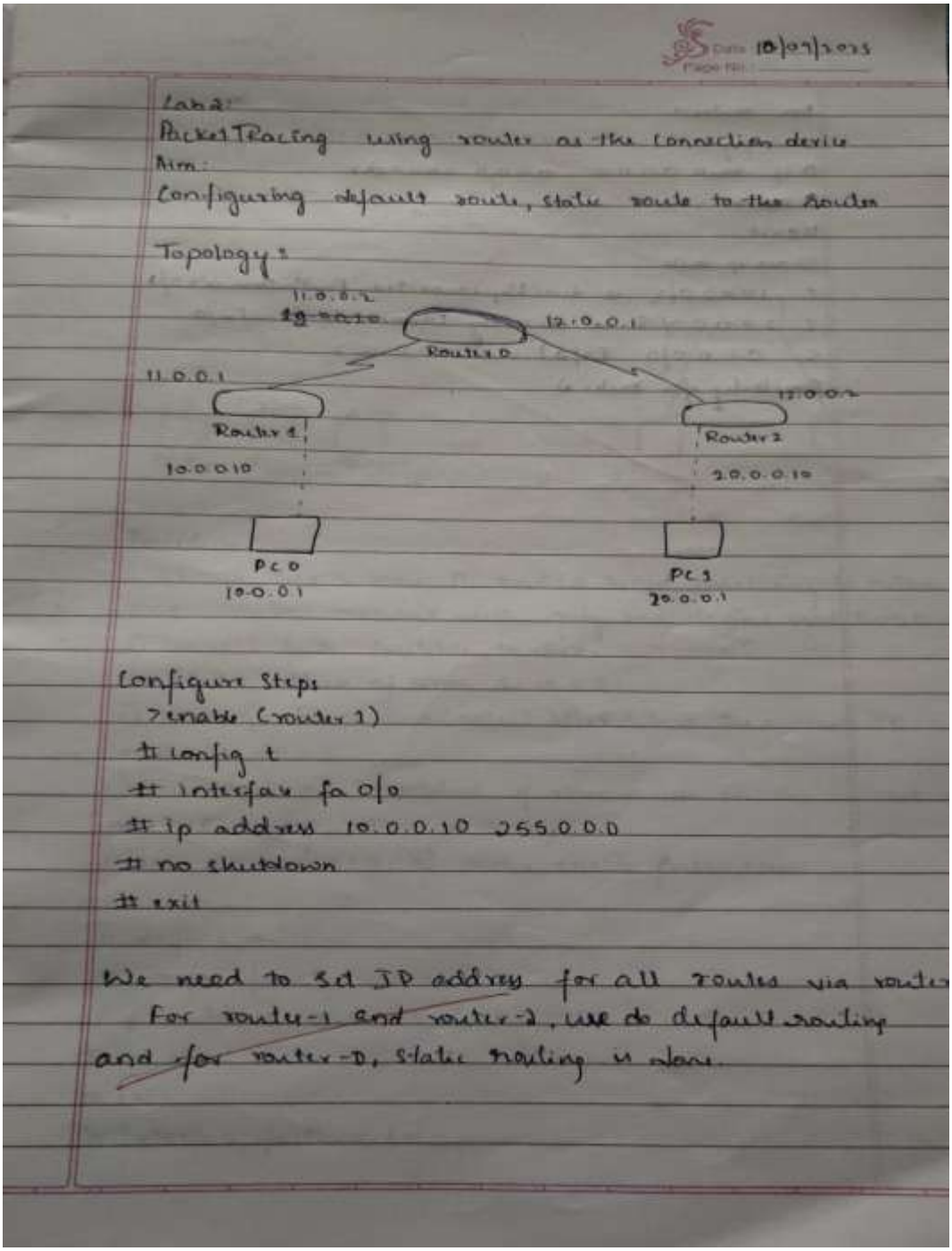
- In the hub-based topology, the PDU was broadcast to all ports, while the switch forwarded the PDU only to the destination MAC after learning addresses from incoming frames.
- Successful ICMP echo and echo-reply messages confirm that both devices enabled connectivity, with the switch demonstrating selective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

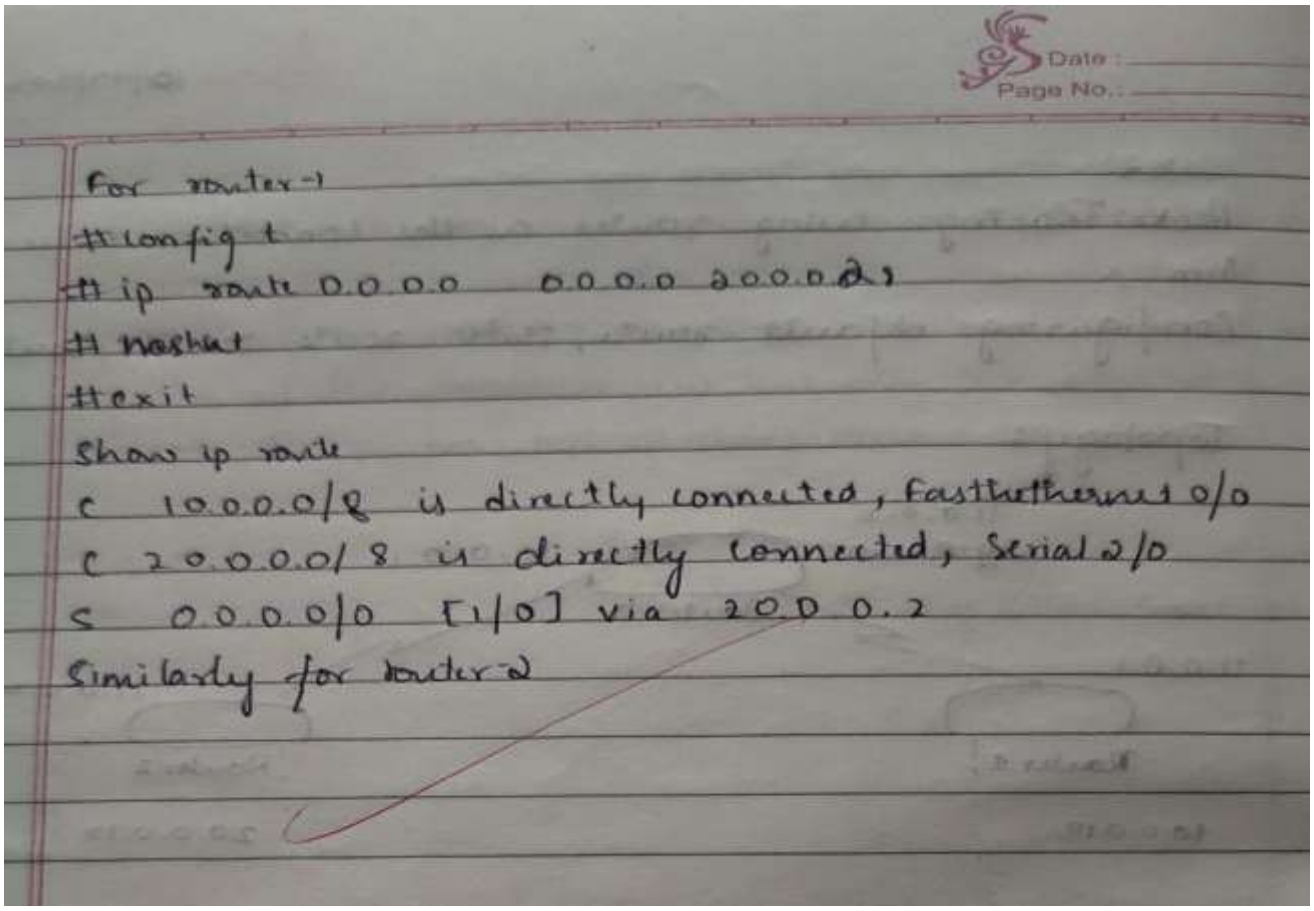
**Program 2**

Aim of the program:

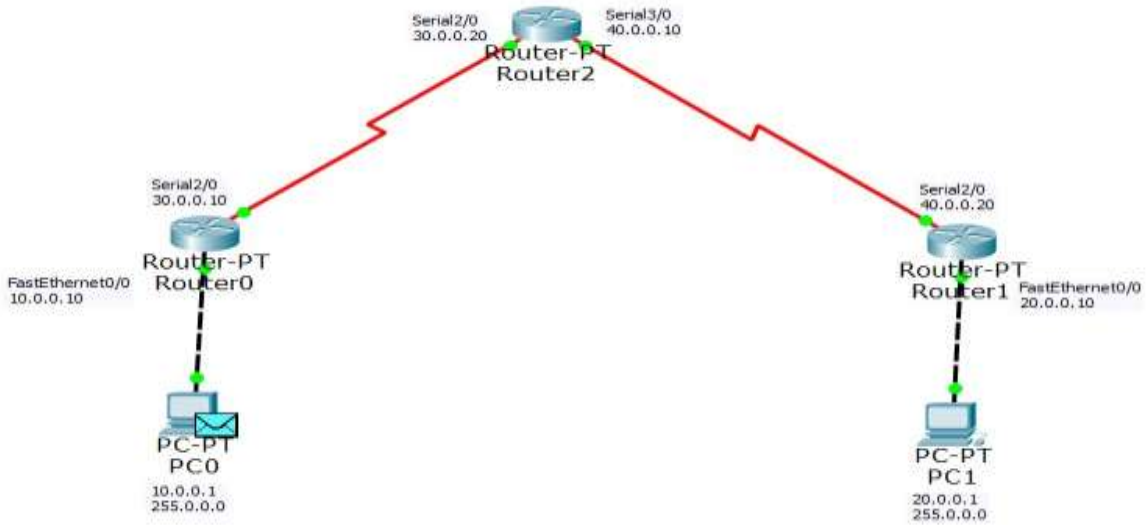
Configure default route, static route to the Router

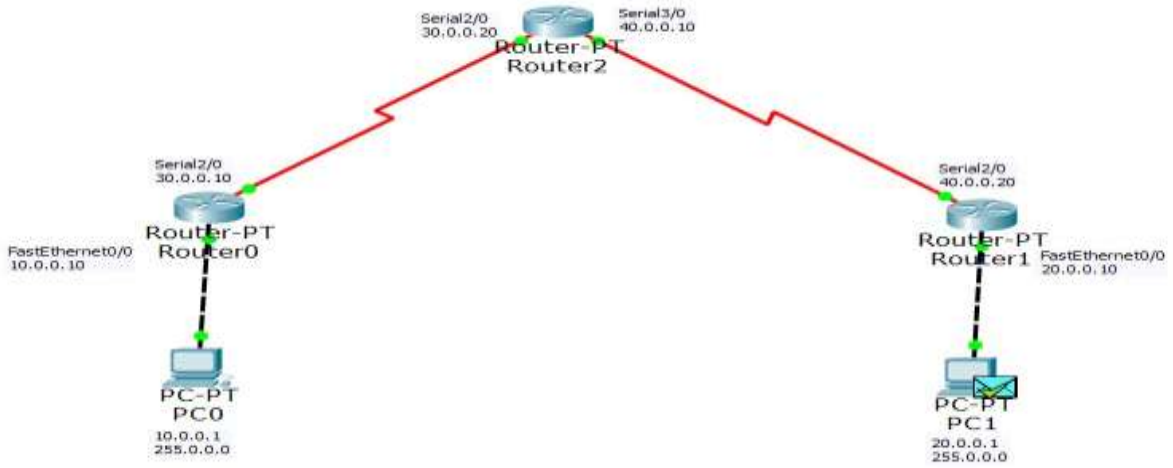
Procedure and topology:



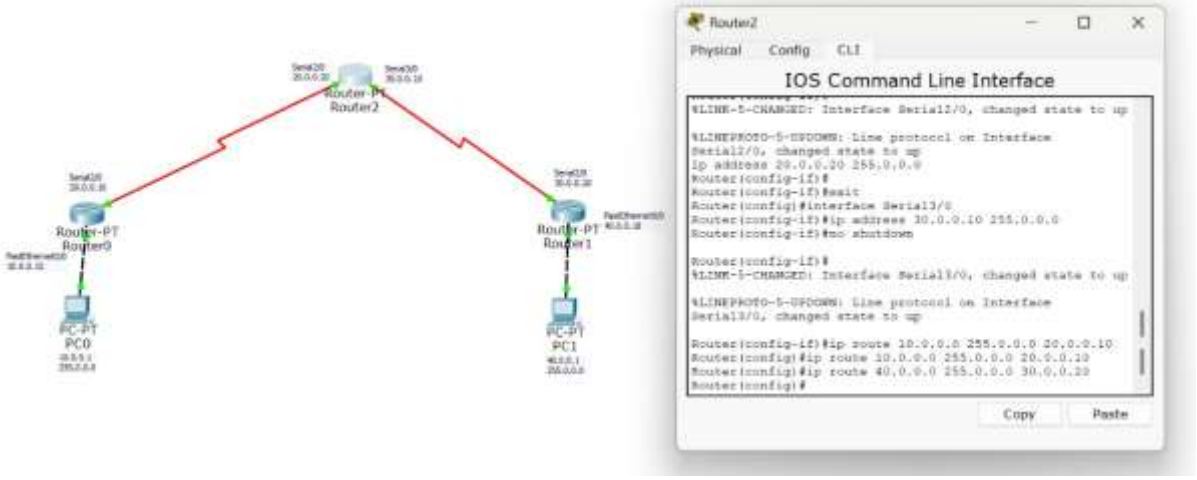



Screenshots/ Output:

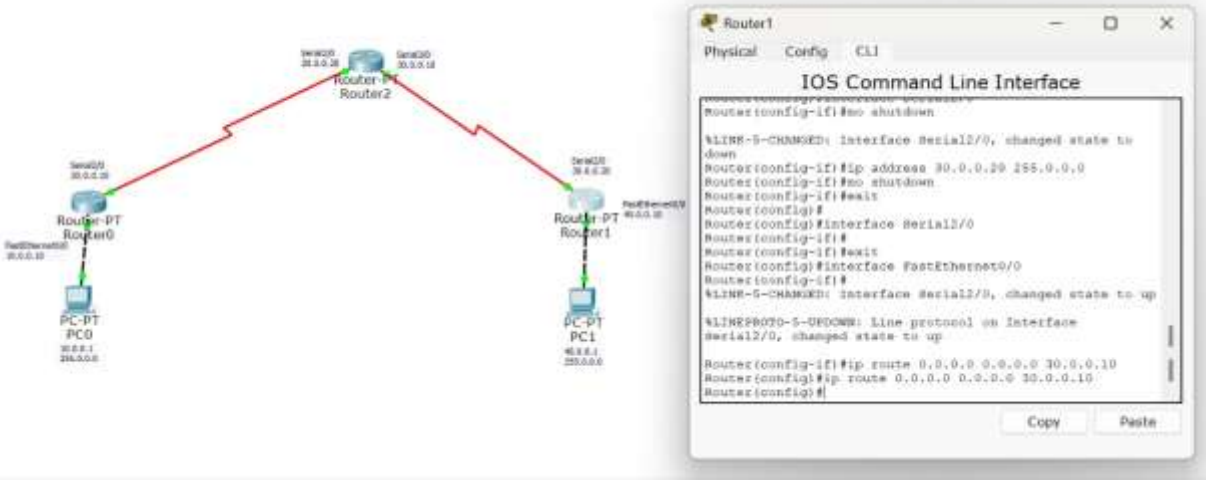





Static routing CLI commands:




Default routing CLI commands:



Observation:

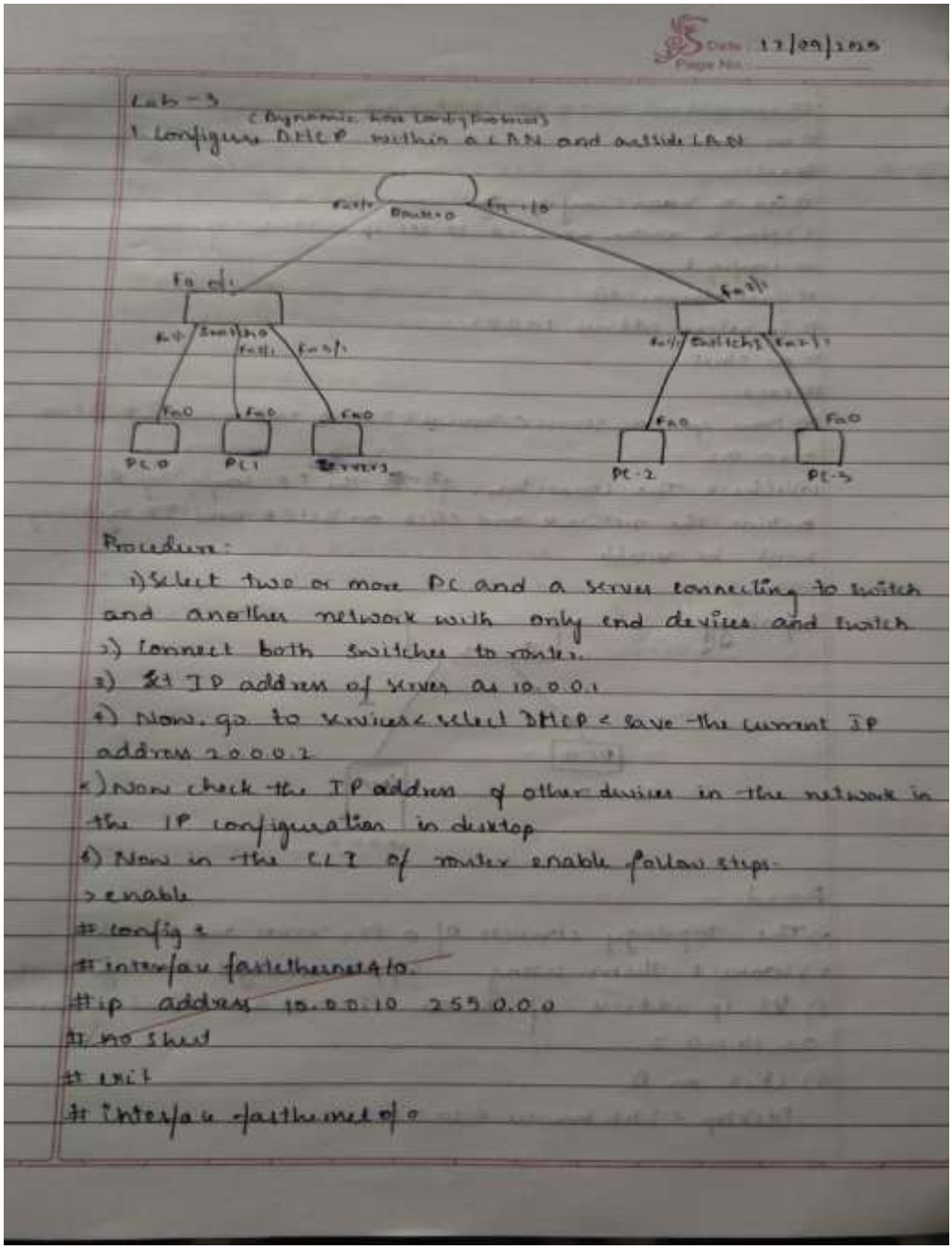
- The configured static and default routes correctly updated the router's routing table, enabling deterministic next-hop selection for remote networks.
- Successful ping tests verified that traffic was forwarded according to the static/default route entries, ensuring end-to-end reachability across different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

**Program 3**

Aim of the program:

Configure DHCP within a LAN and outside LAN.

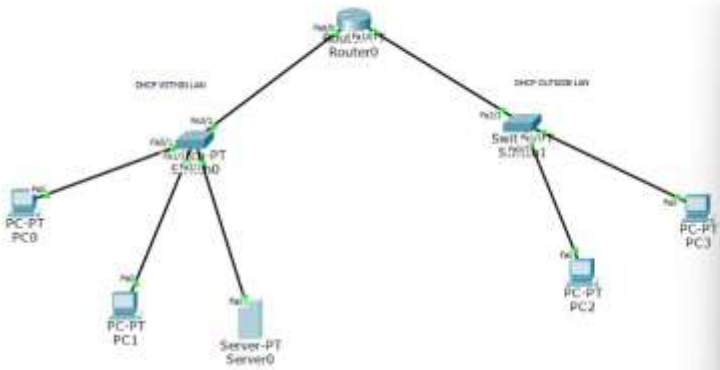
Procedure and topology:

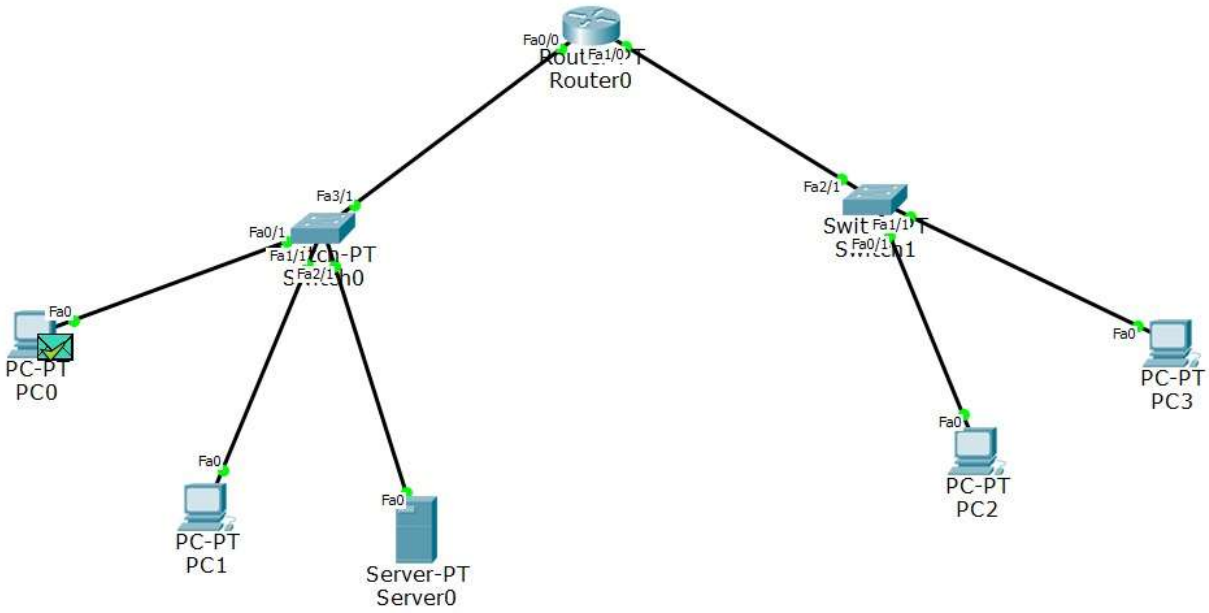


Date :
Page No.:

# ip address 20.0.0.20 255.0.0.0
# no shut
# exit.
7) Go to Server < config < gateway 10.0.0.20
8) Now in router, we need to set ip address of server.
# config t
# fast ethernet 0/0
# ip helper-address 10.0.0.1
# no shut
# exit
9) Now go to Server < Services < DHCP add new IP address 20.0.0.2
10) To check the connection, go to the IP config. of PC outside the network and click on DHCP and IP gateway will be visible.

Screenshots/ Output:




Observation:

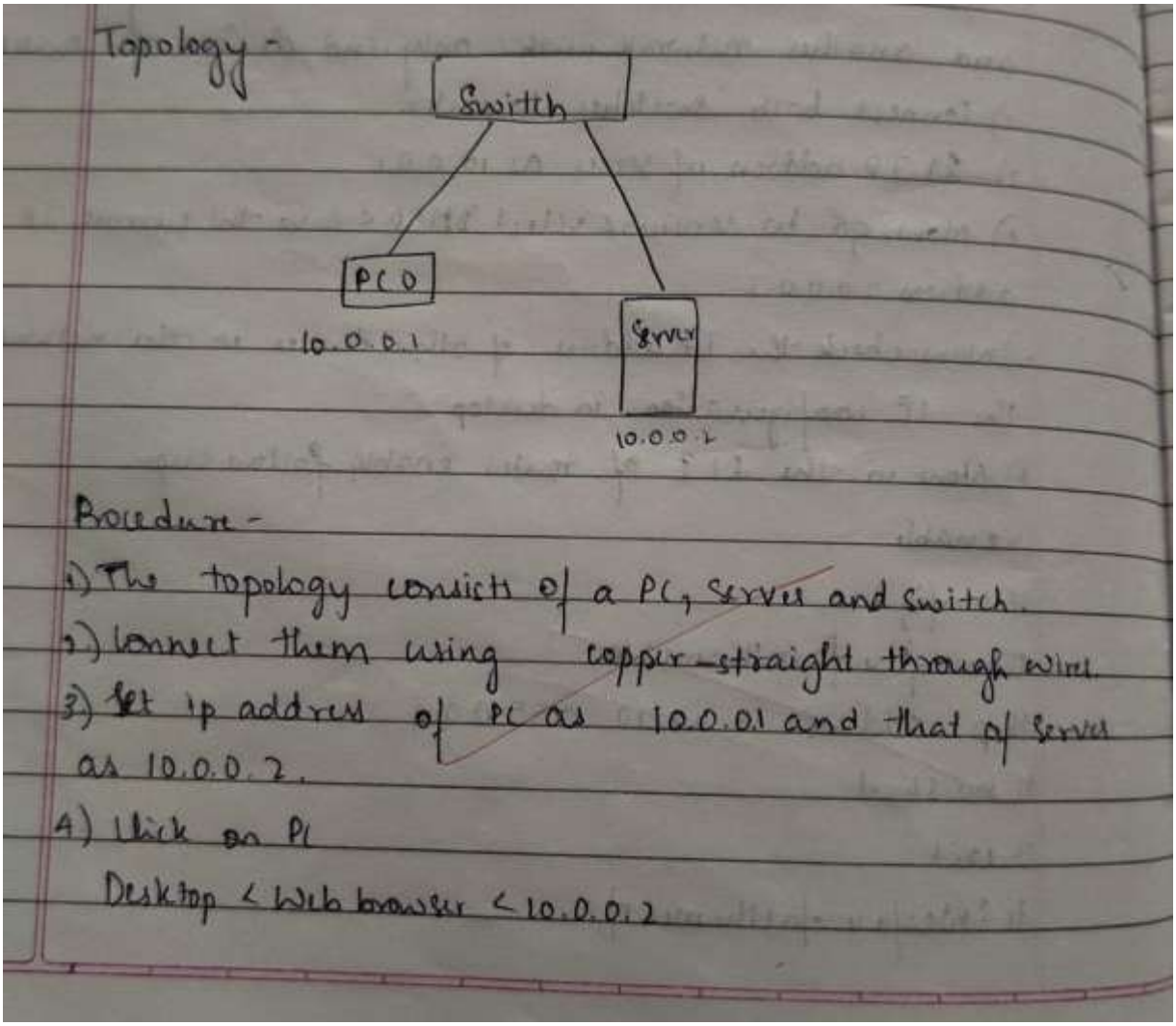
- The DHCP server successfully allocated IP addresses to clients within the LAN, confirming proper scope configuration and automatic distribution of network parameters.
- DHCP relay (IP Helper) enabled clients outside the LAN to obtain leases from the central DHCP server, demonstrating correct inter-network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

**Program 4**

Aim of the program:

Configure Web Server, DNS within a LAN.

Procedure and topology:



It displays the index.html page of the server

a) Click on server

Services < index.html < edit < change the subjects to BMS clg of Engineering CSE < save

b) Click on PC

Desktop < Web browser < 10.0.0.2

It changes the content of index.html page as set by us i.e BMC clg of Engineering CSE.

7) Click on server

Services < DNS < On <

give domain name as bmsece.c <

Set address as 10.0.0.2

< add

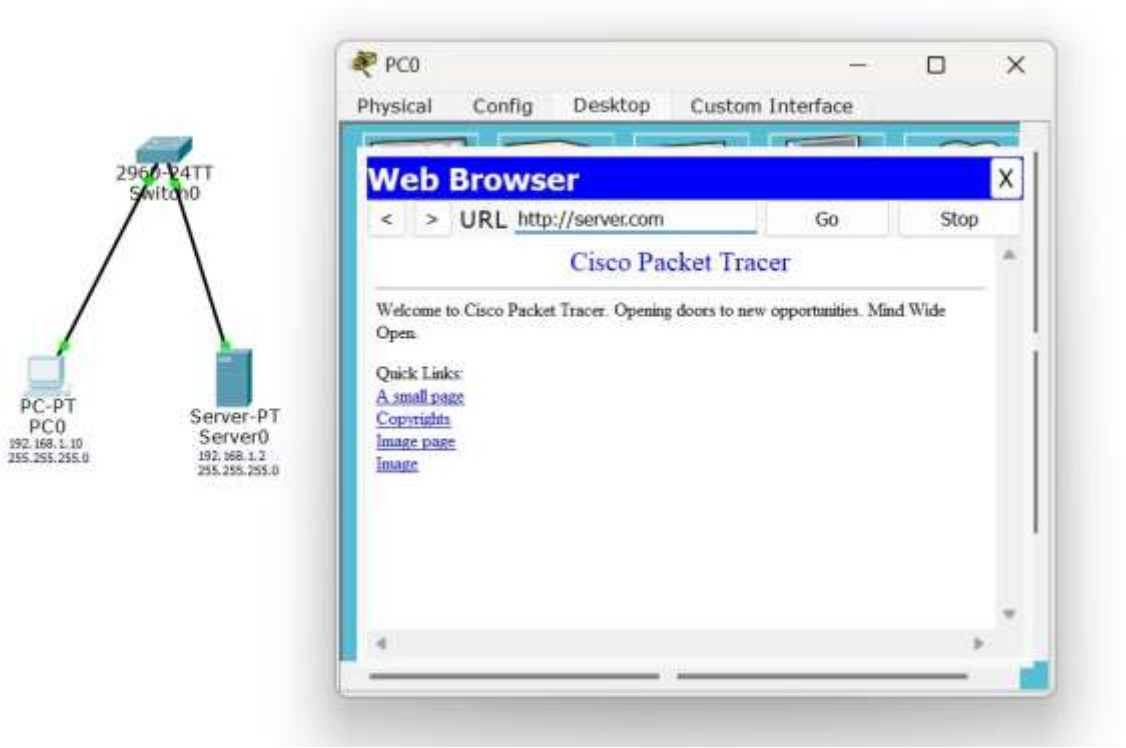
8) Click on PC

Desktop < Web browser < bmsece

It displays the same index.html of page of page of the server ( with ip address 10.0.0.2)

17.09

Screenshots/ Output:




Observation:

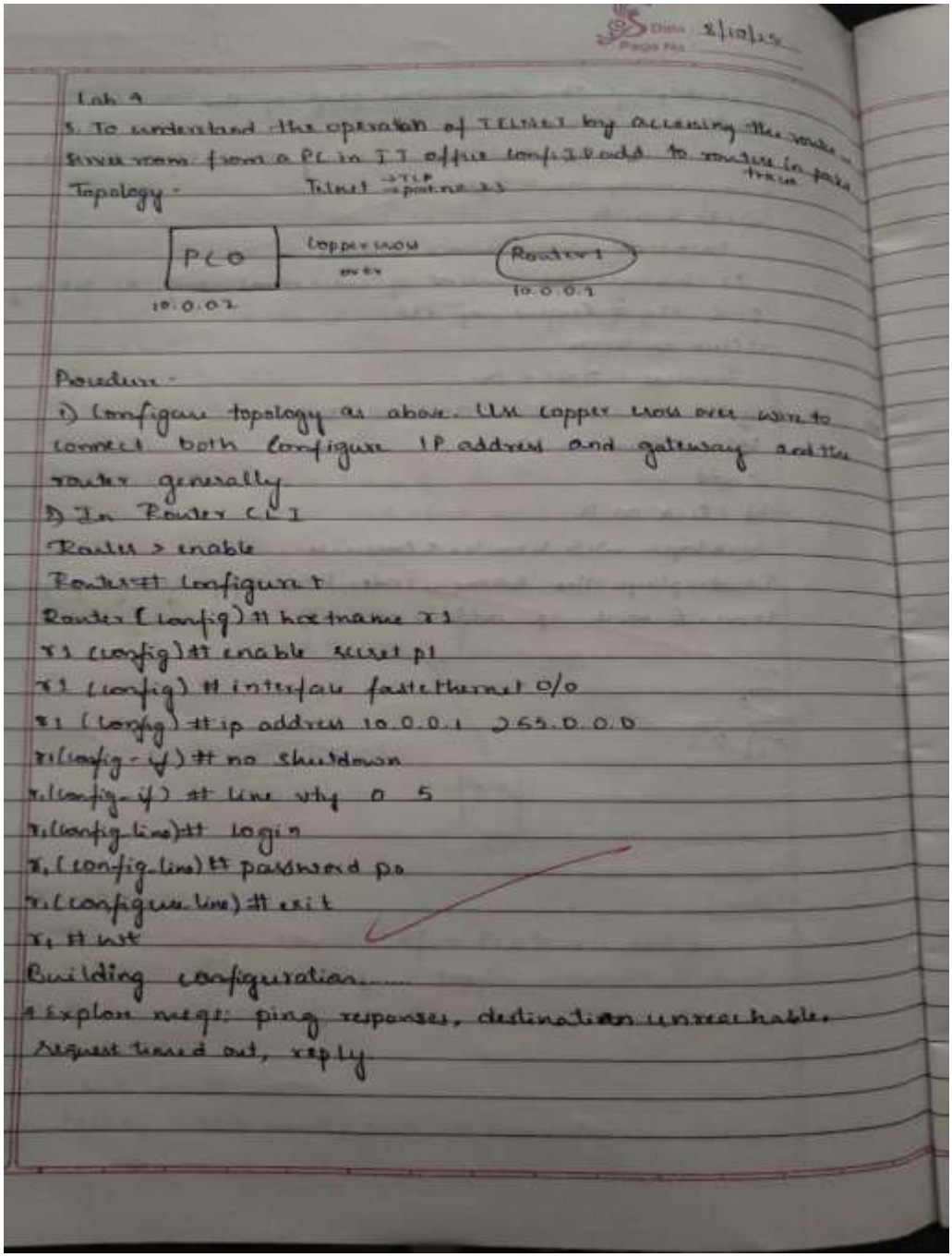
- The DNS server successfully resolved domain names to the corresponding web server's IP address, confirming proper hostname-to-IP mapping within the LAN.
- HTTP requests reached the web server using the DNS-resolved address, validating correct server configuration and internal LAN communication.

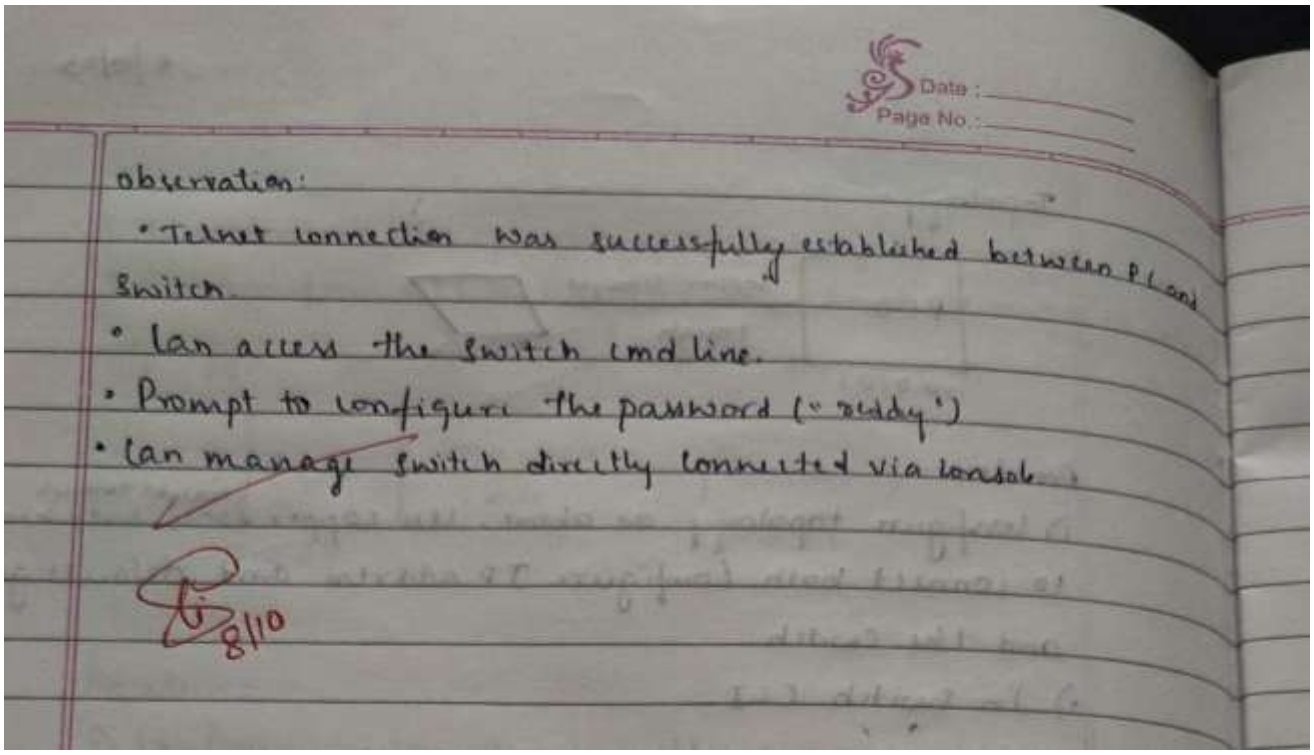
**Program 5**

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:



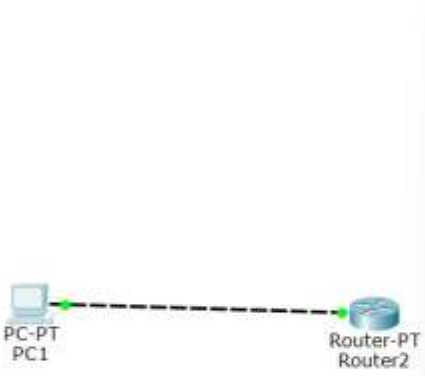


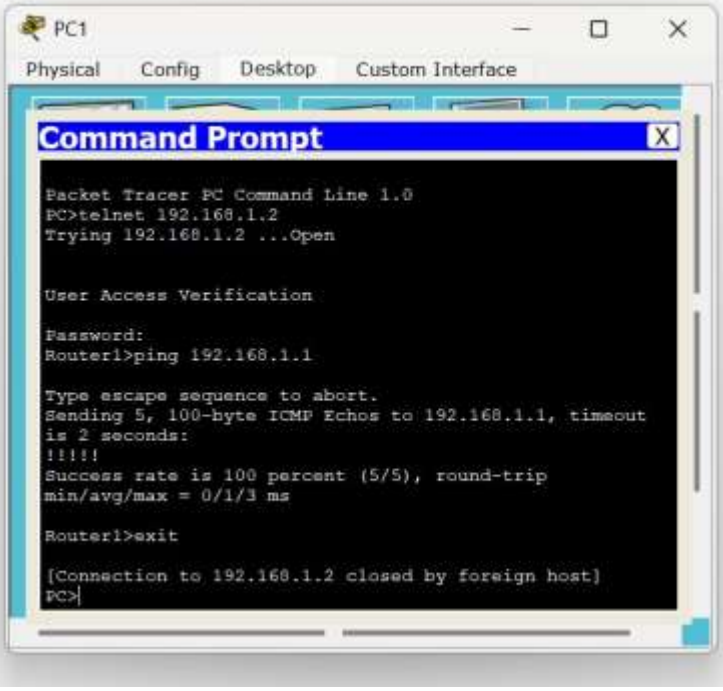
Observation:

- Telnet connection was successfully established between PC and Switch.
- Can access the switch cmd line.
- Prompt to configure the password ("reddy")
- Can manage switch directly connected via console.

8/10

Screenshots/ Output:




Observation:

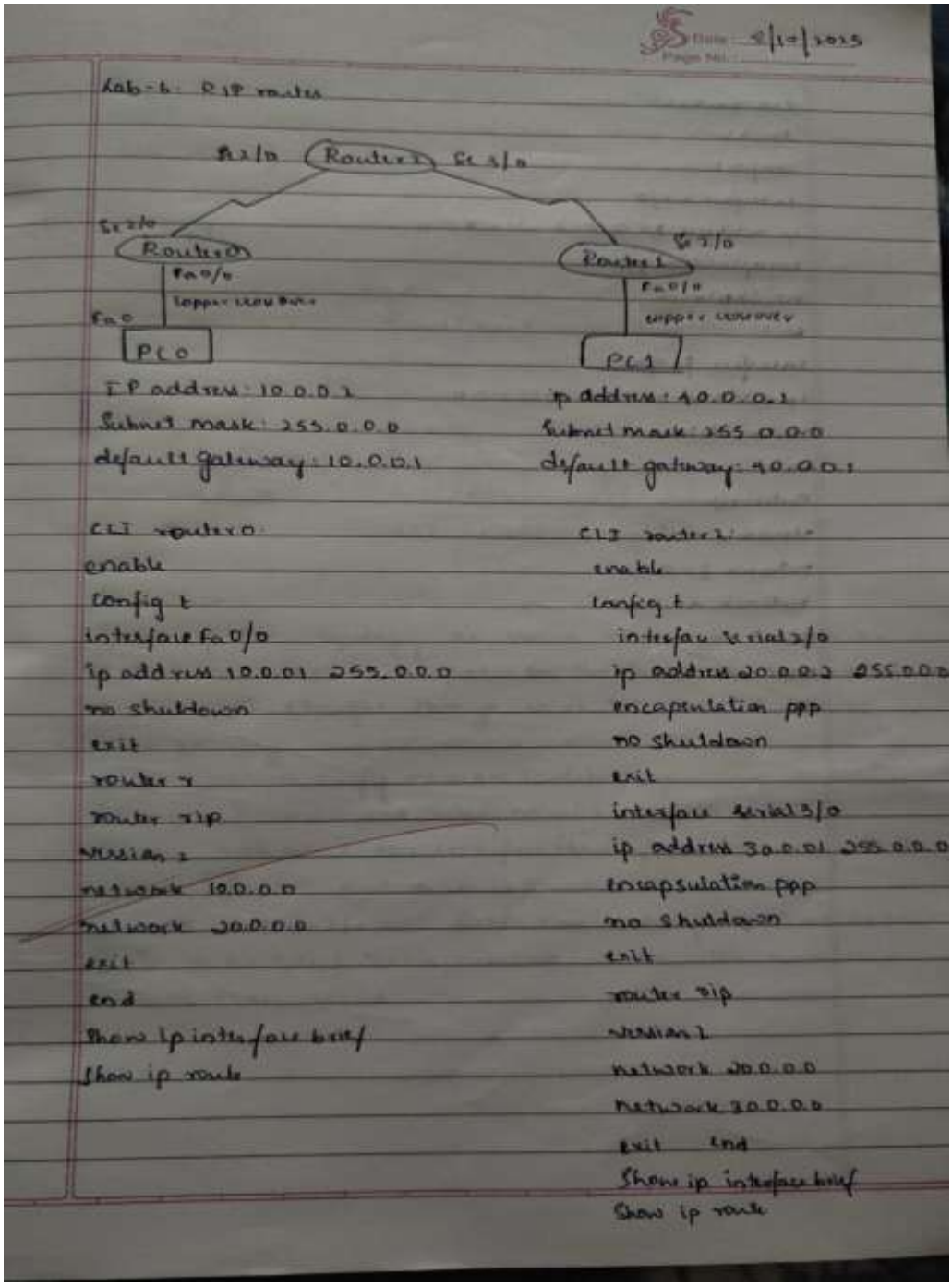
- The Telnet session successfully established a remote CLI connection to the router, confirming proper VTY line configuration and IP reachability between the IT office PC and the server room router.
- Command execution over the Telnet session demonstrated reliable remote device management.

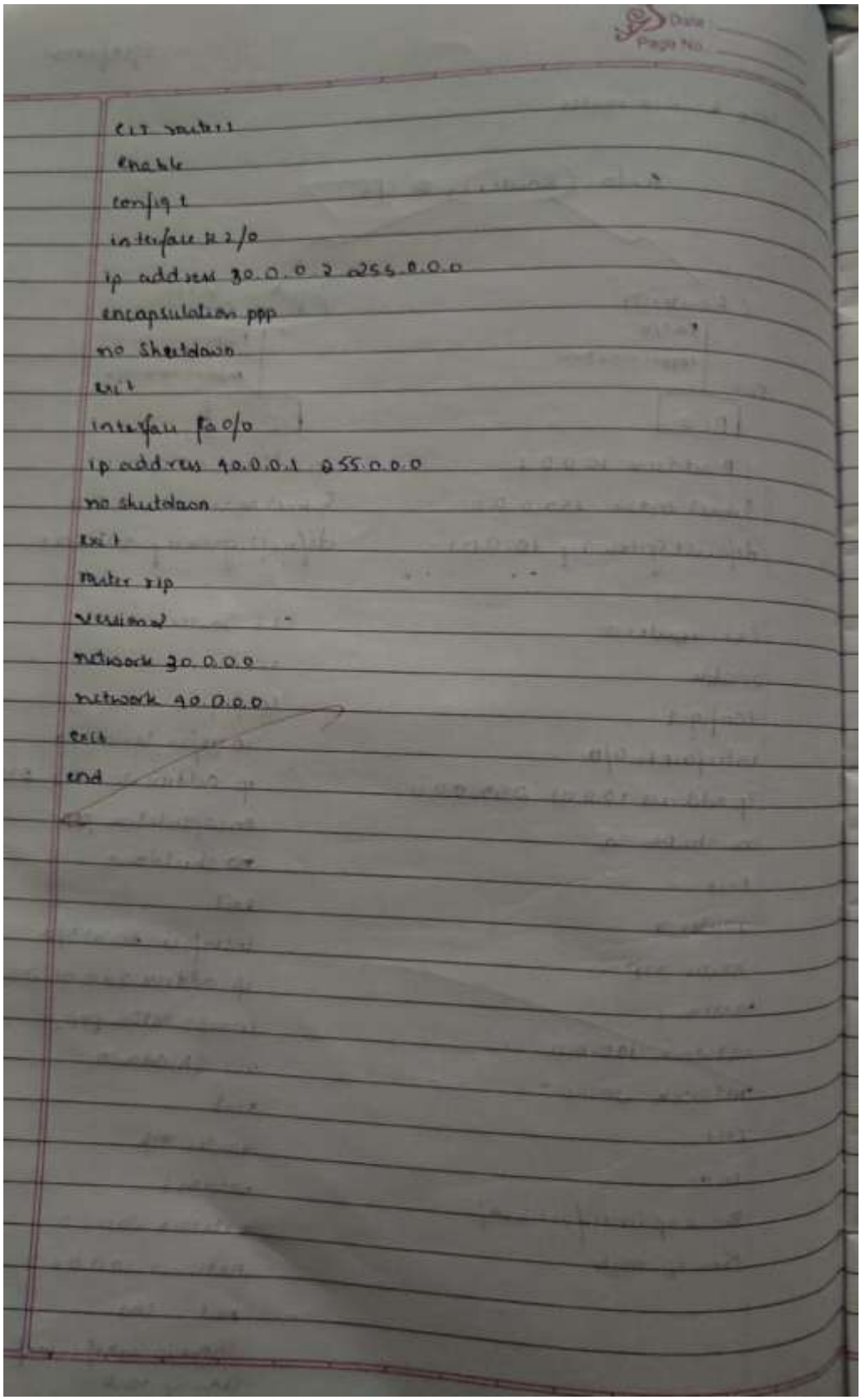
**Program 6**

Aim of the program:

Configure RIP routing Protocol in Routers

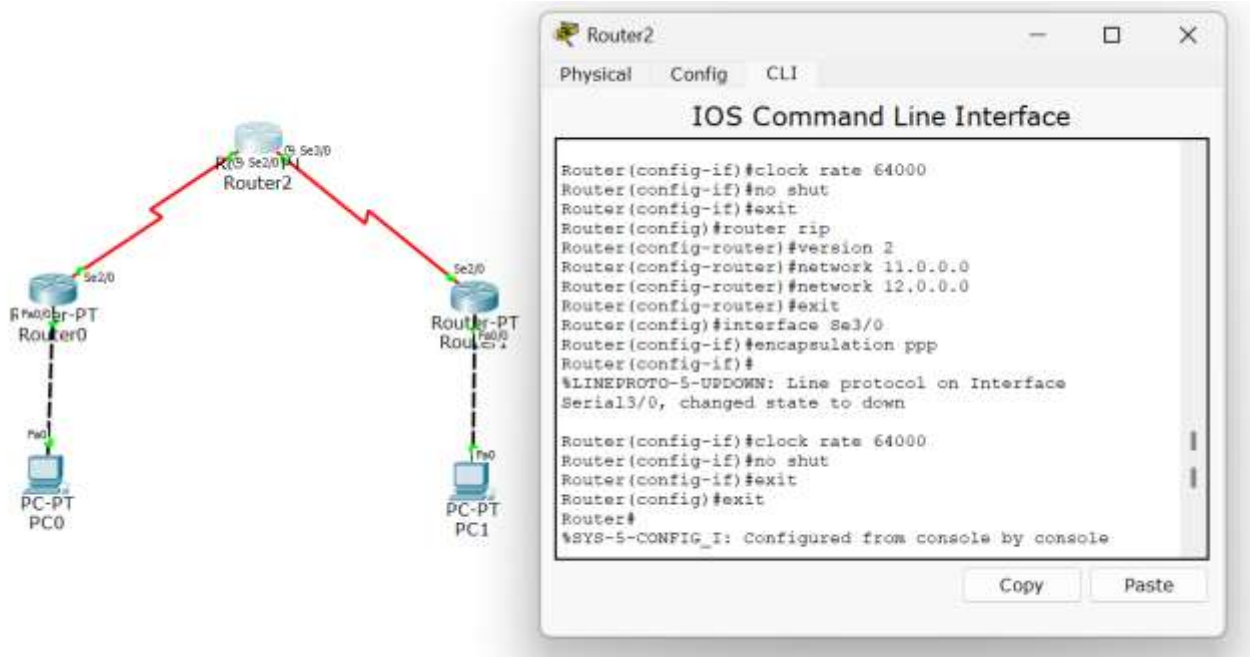
Procedure and topology:

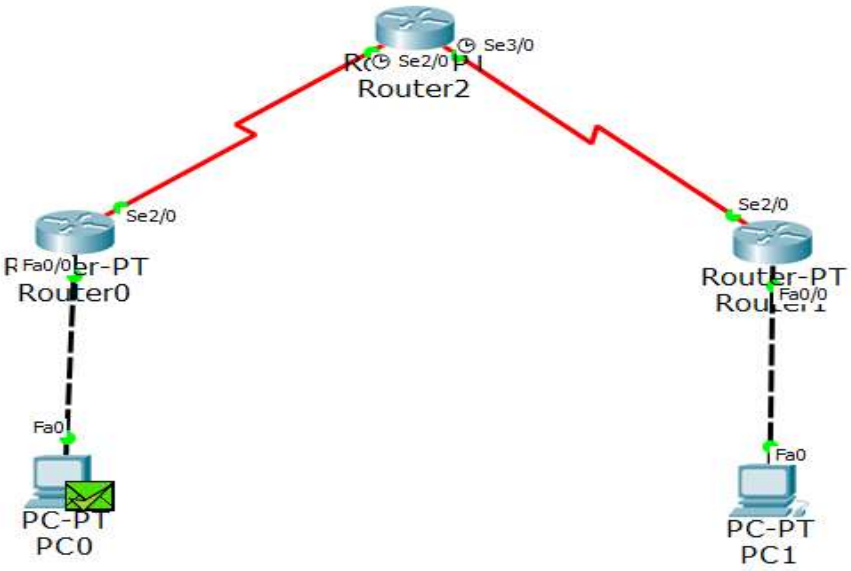




```
Router>
enable
config t
interface s2/0
ip address 30.0.0.2 255.0.0.0
encapsulation ppp
no shutdown
exit
interface fa0/0
ip address 40.0.0.1 255.0.0.0
no shutdown
exit
router rip
version 2
network 30.0.0.0
network 40.0.0.0
exit
end
```

Screenshots/ Output:




Observation:

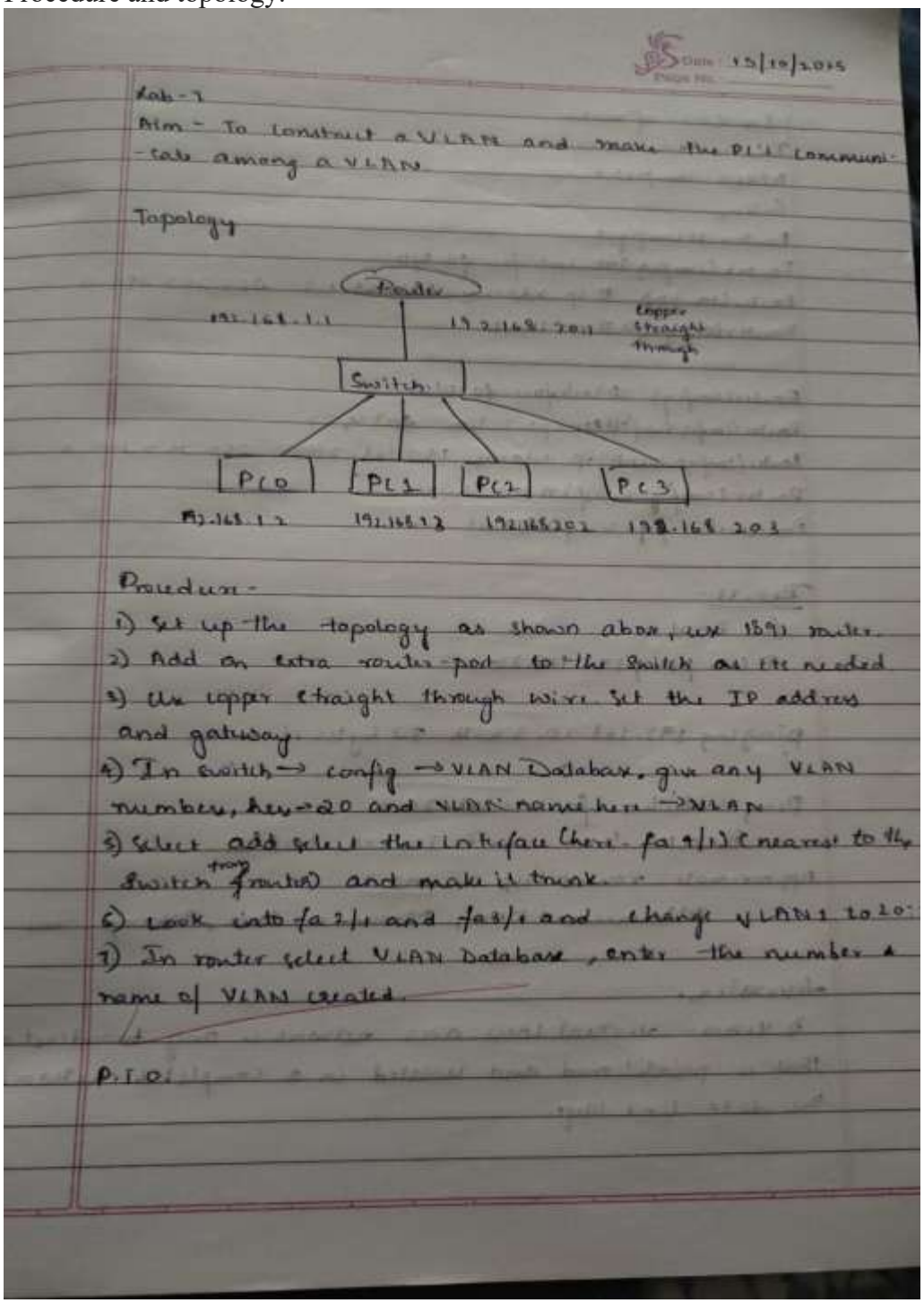
- RIP routing updates were successfully exchanged between routers, allowing each router to dynamically learn remote network routes through hop-count–based distance vector advertisements.
- The routing tables converged correctly, and successful ping tests confirmed end-to-end connectivity maintained by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

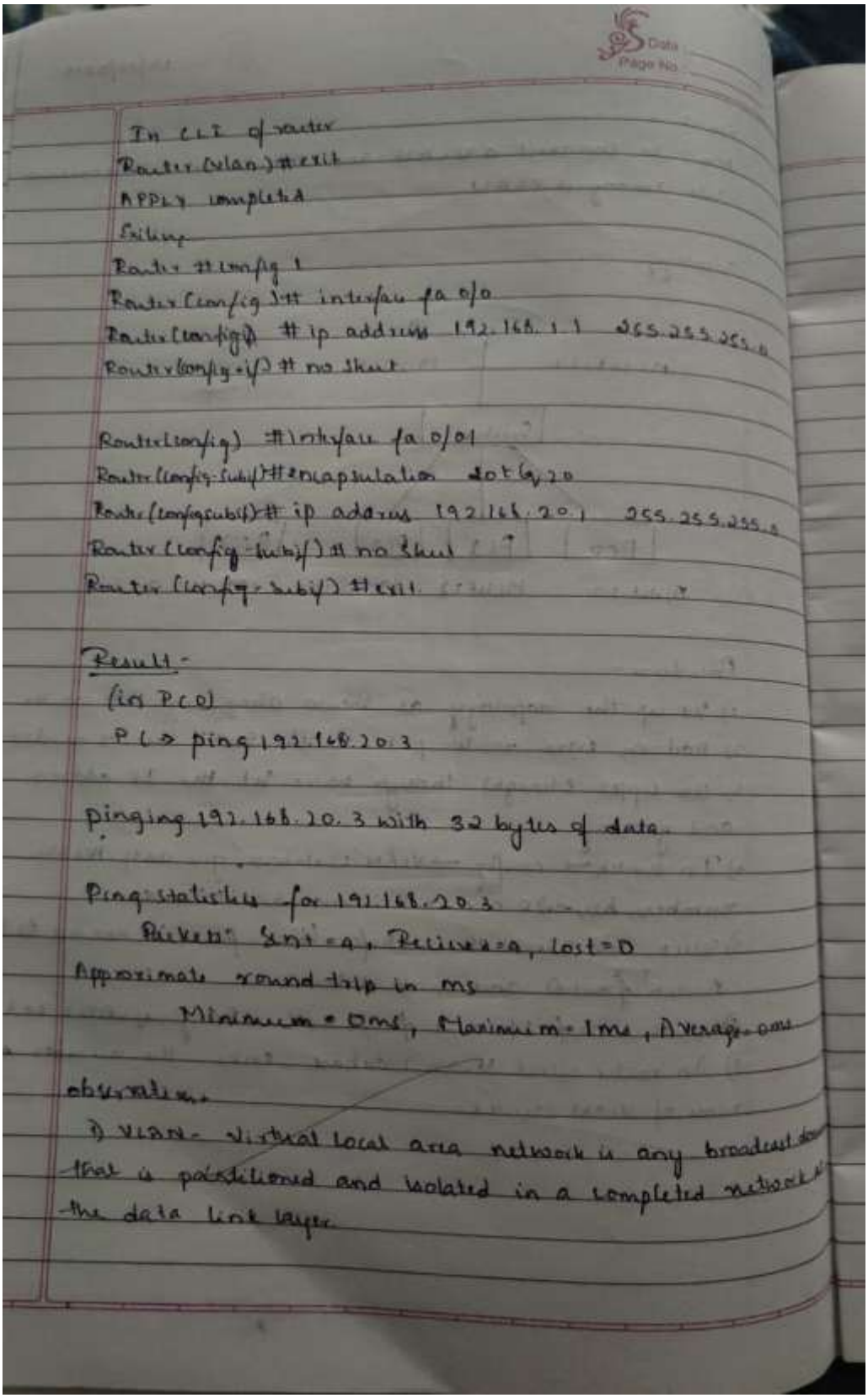
**Program 7**

Aim of the program:

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

Procedure and topology:





In CLI of router
Router(vlan)#exit
APPLY completed
Exiting
Router#config t
Router(config)#interface fa 0/0
Router(config)# ip address 192.168.1.1 255.255.255.0
Router(config-if)# no shut

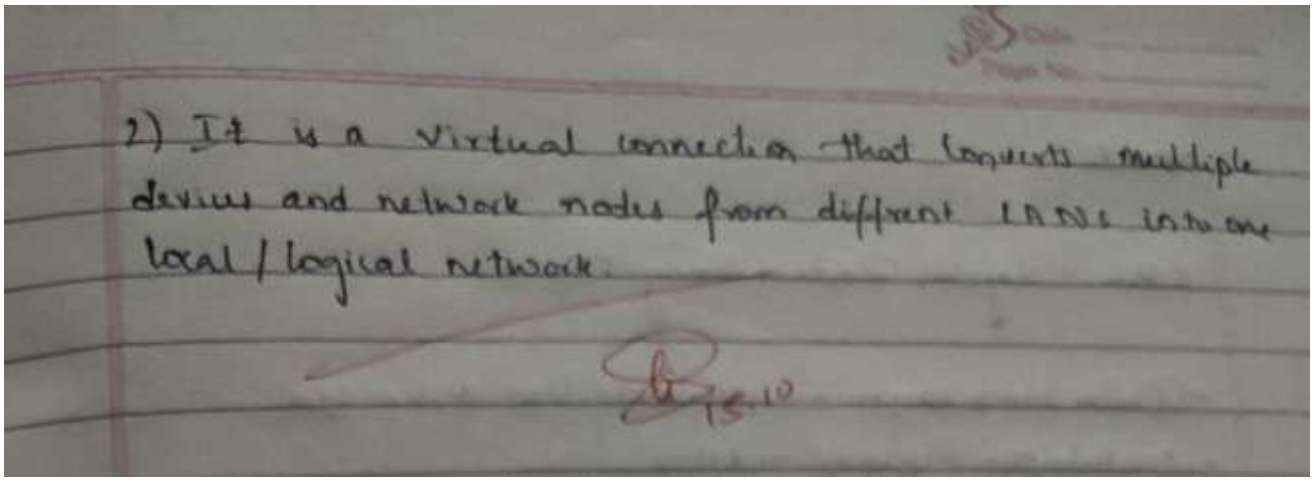
Router(config)#interface fa 0/0.1
Router(config-subif)#encapsulation dot1q 20
Router(config-subif)# ip address 192.168.20.1 255.255.255.0
Router(config-subif)# no shut
Router(config-subif)#exit

Result:-
(in PC0)
PC> ping 192.168.20.3

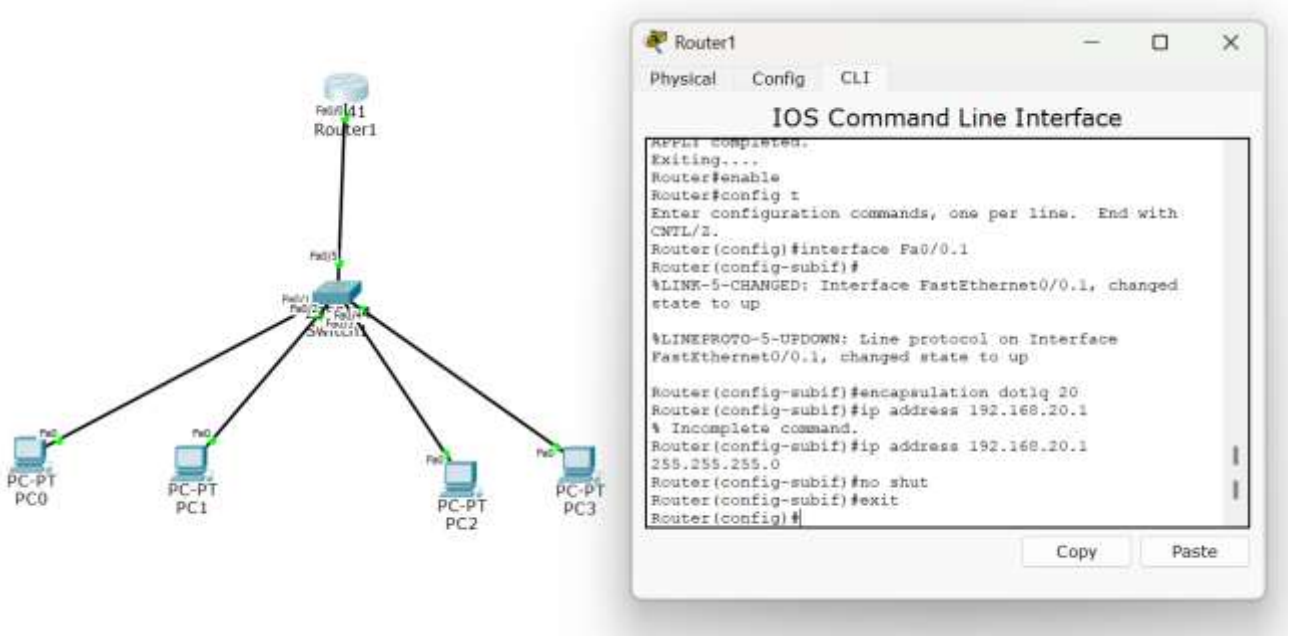
pinging 192.168.20.3 with 32 bytes of data.

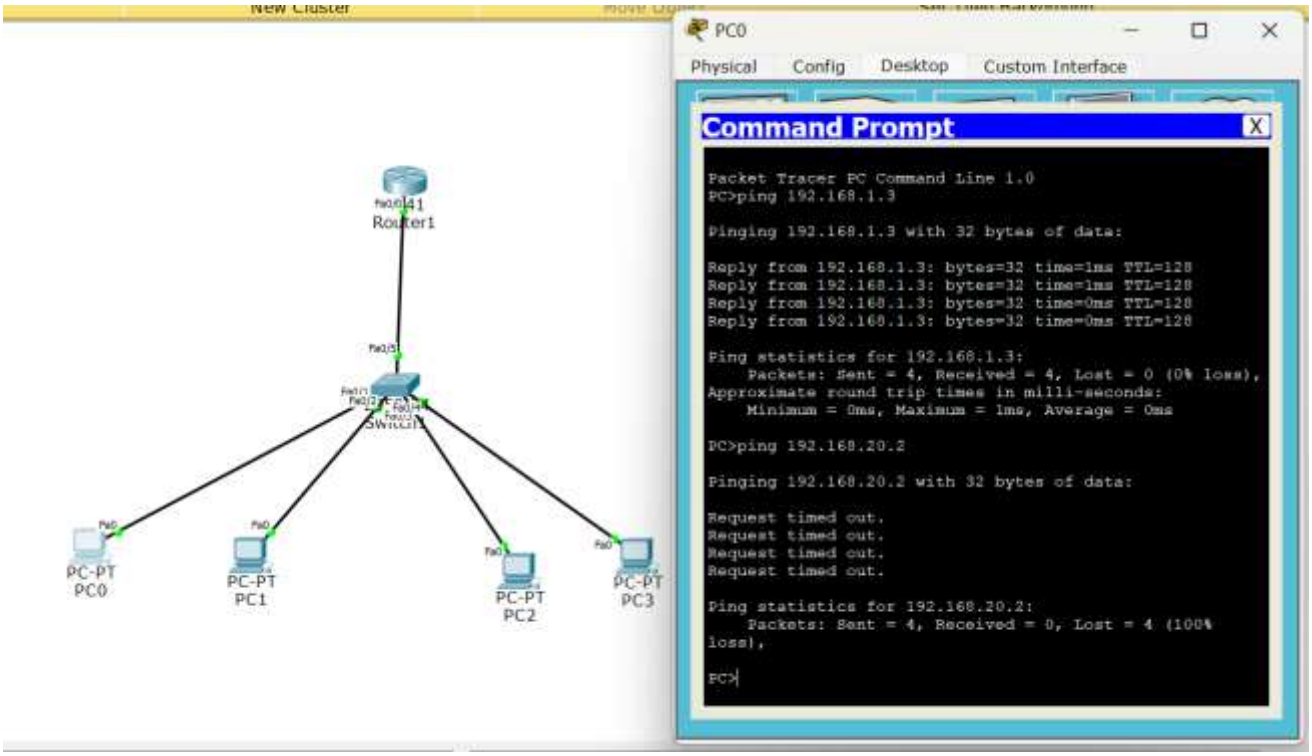
Ping statistics for 192.168.20.3
Packets: Sent = 4, Received = 4, Lost = 0
Approximate round trip in ms
Minimum = 0ms, Maximum = 1ms, Average = 0ms

observation:-
1) VLAN - Virtual local area network is any broadcast domain that is partitioned and isolated in a completed network at the data link layer.




Screenshots/ Output:






Observation:

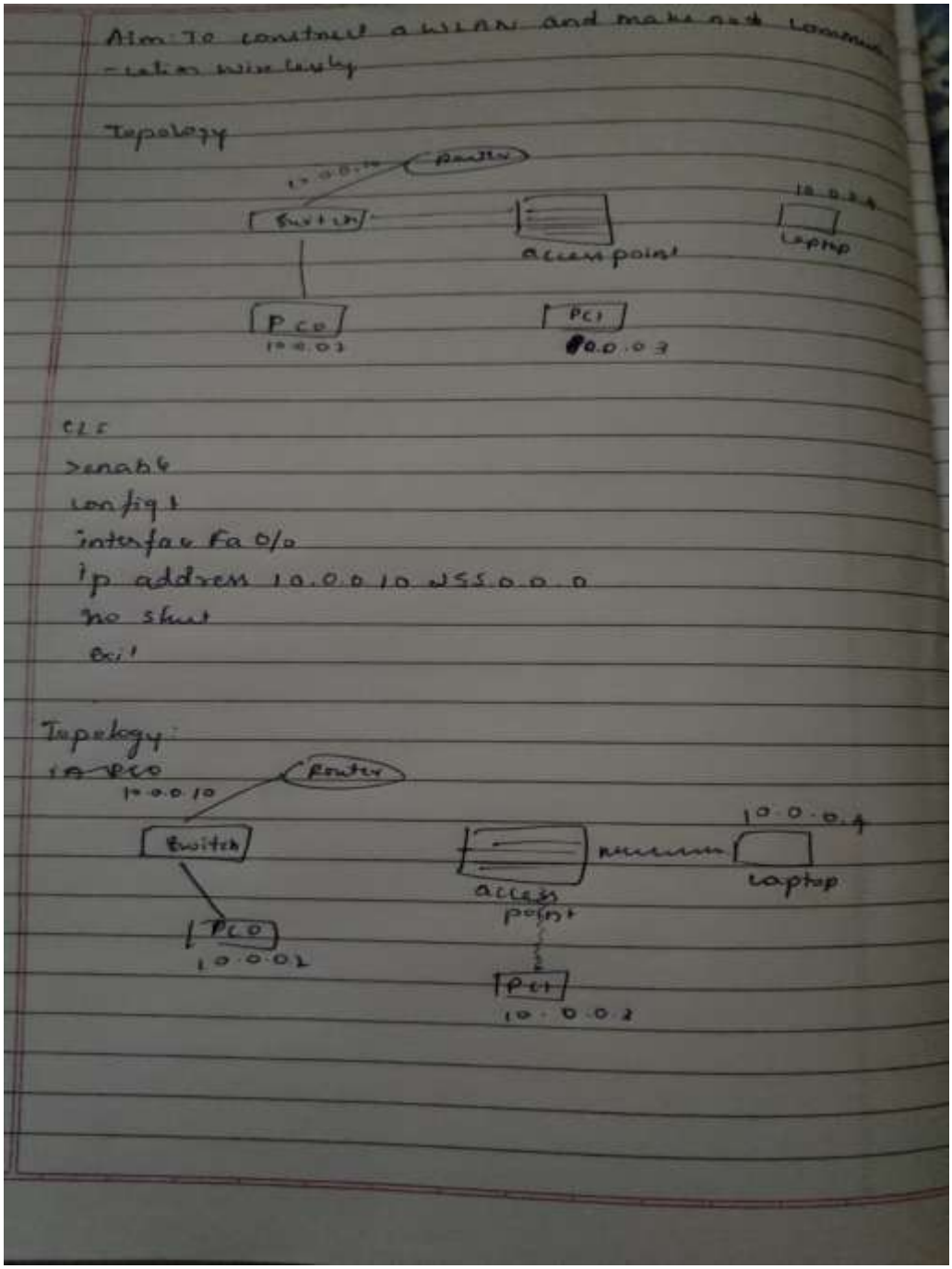
- VLAN segmentation successfully separated broadcast domains, and switch ports were correctly assigned to their respective VLAN IDs using access mode configuration.
- Inter-VLAN communication was achieved through the Layer-3 device, and successful ping tests confirmed proper VLAN membership, tagging,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

**Program 8**

Aim of the program:

To construct a WLAN and make the nodes communicate wirelessly

Procedure and topology:



Result-

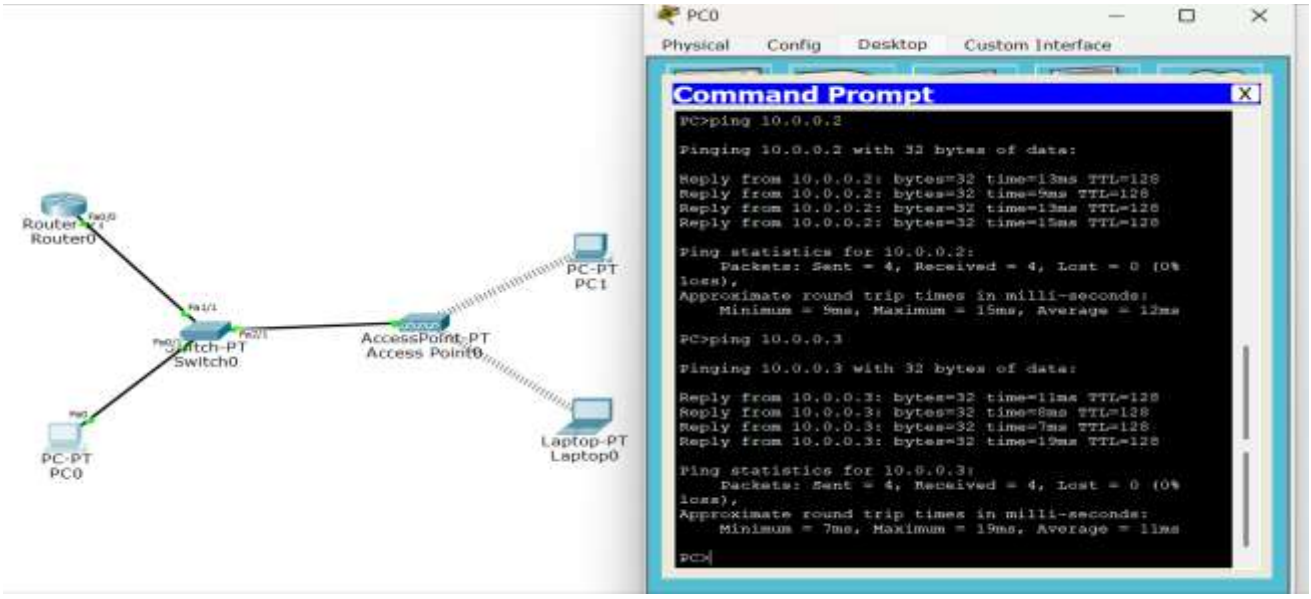
>ping 40.0.0.10

Pinging 40.0.0.10 with 32 bytes of data:

Reply from 40.0.0.10: bytes=32 time=9ms TTL=128
Reply from 40.0.0.10: bytes=32 time=9ms TTL=128
Reply from 40.0.0.10: bytes=32 time=9ms TTL=128
Reply from 40.0.0.10: bytes=32 time=9ms TTL=128

Ping statistics for 40.0.0.10:
Packets: Sent = 4, Received = 4, Lost = 0
Approximate round trip times in milliseconds:
Minimum = 2ms, Maximum = 12ms, Average = 8ms

Screenshots/ Output:



Observation:

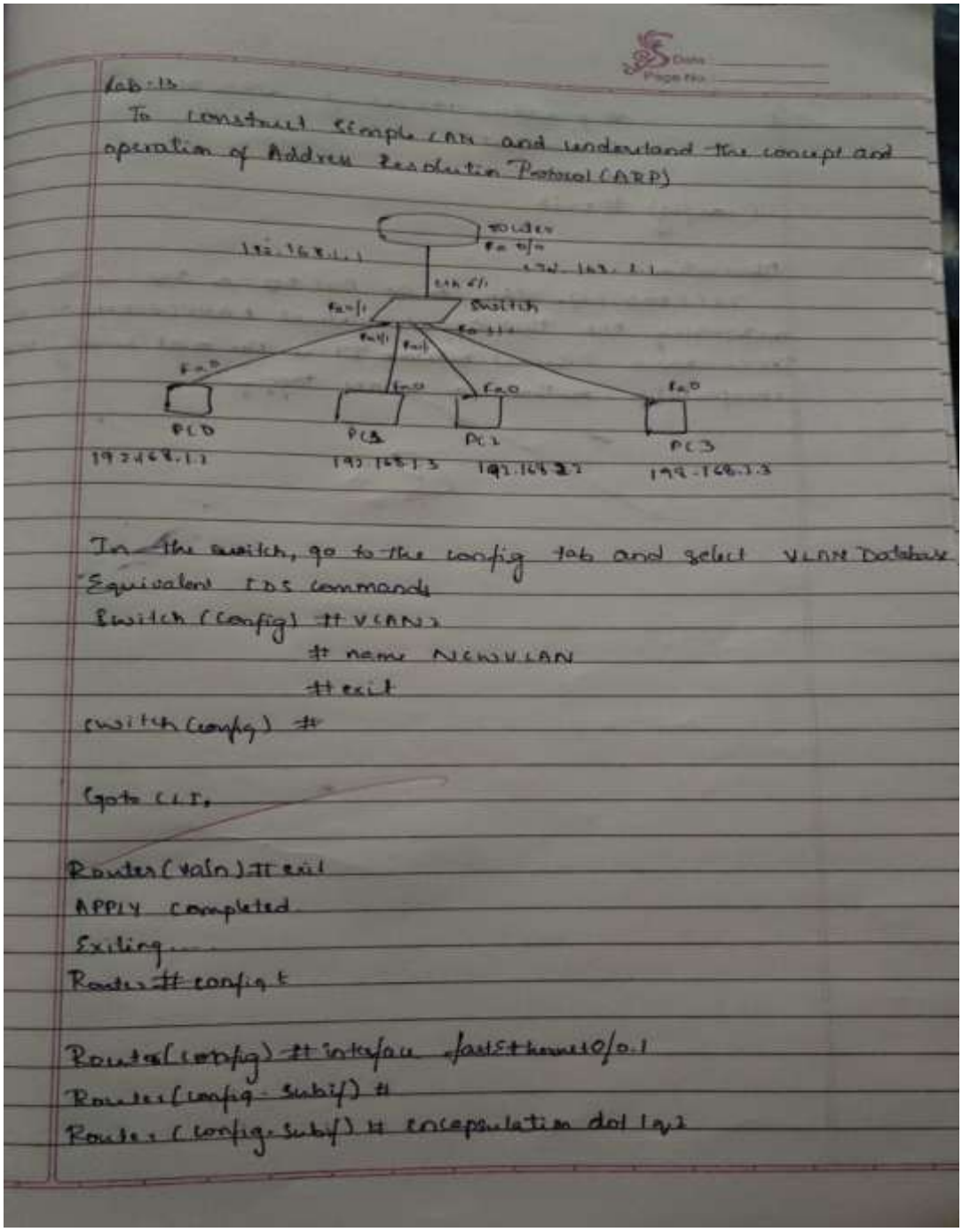
- The WLAN configuration enabled wireless nodes to associate with the access point using the configured SSID and security settings, confirming proper authentication and signal coverage.
- Successful ping communication between wireless devices verified stable wireless Connectivity.

**Program 9**

Aim of the program:

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

Procedure and topology:

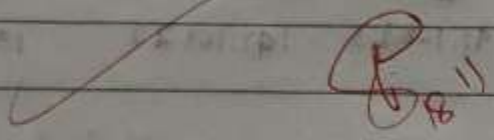


Date :
Page No. :

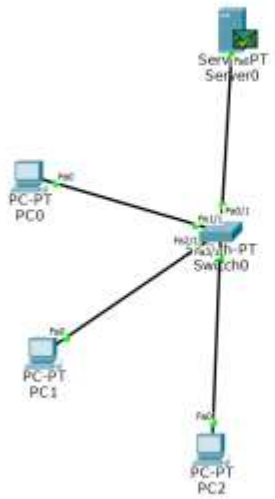
```
1       # ip address 192.168.2.1  255.255.255.0
        # no shut
        # exit
(config) # exit
```

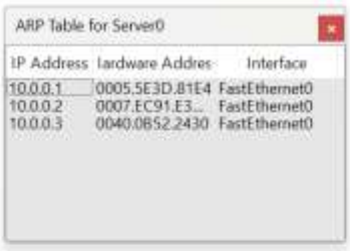
Observation:

IEEE802.1Q, o/w to as Dot1Q or 1Q, is the networking std. that supports virtual LANs (VLANs) on an IEEE 802.3 Ethernet network. It is the most widely used encapsulation method for VLAN tagging.



Screenshots/ Output:









Observation:

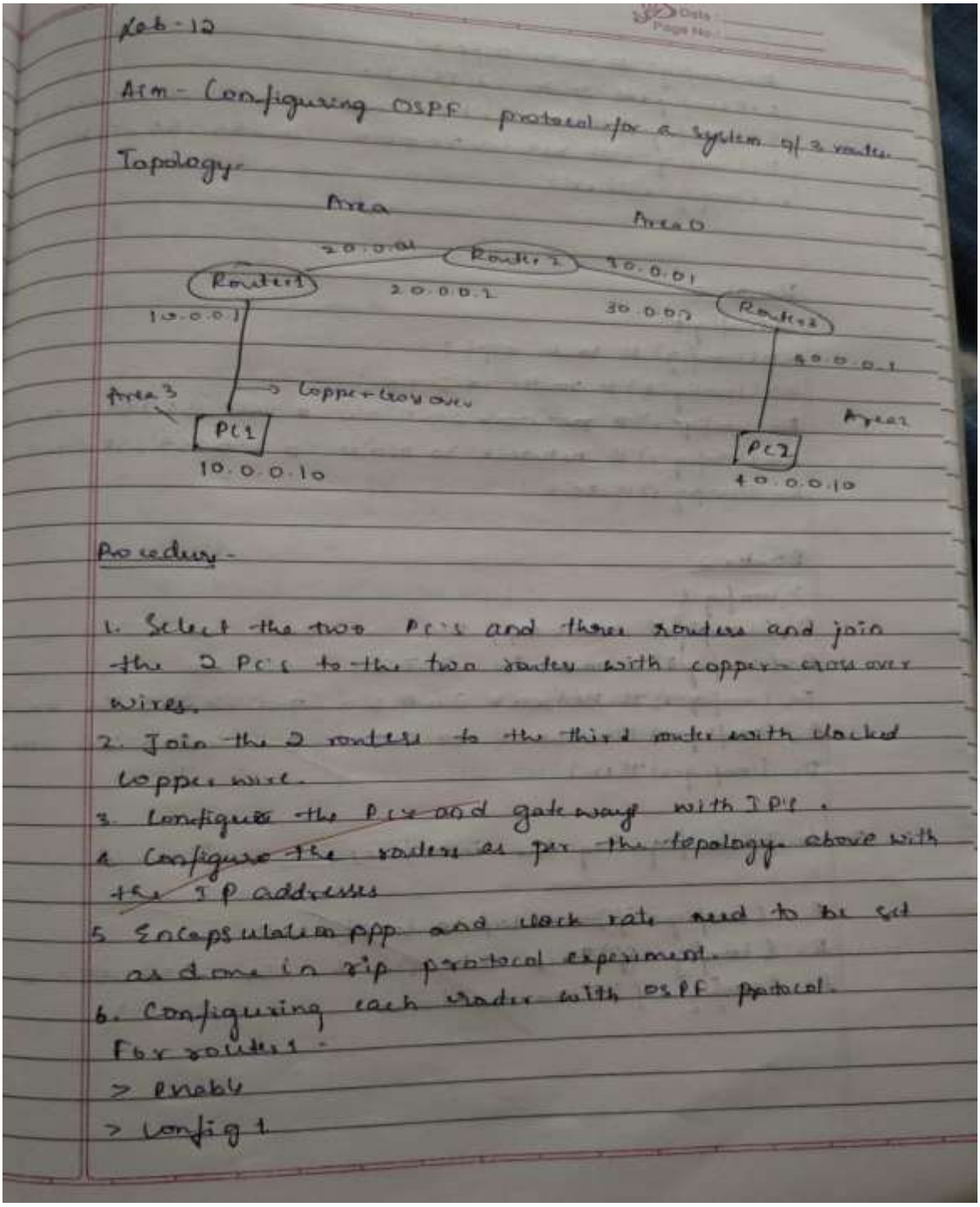
- ARP successfully resolved the destination IP address to its corresponding MAC address, as seen from ARP request and reply exchanges between LAN hosts.
- The populated ARP tables and successful ping communication confirmed correct layer-2 addressing, frame forwarding, and basic LAN operation.

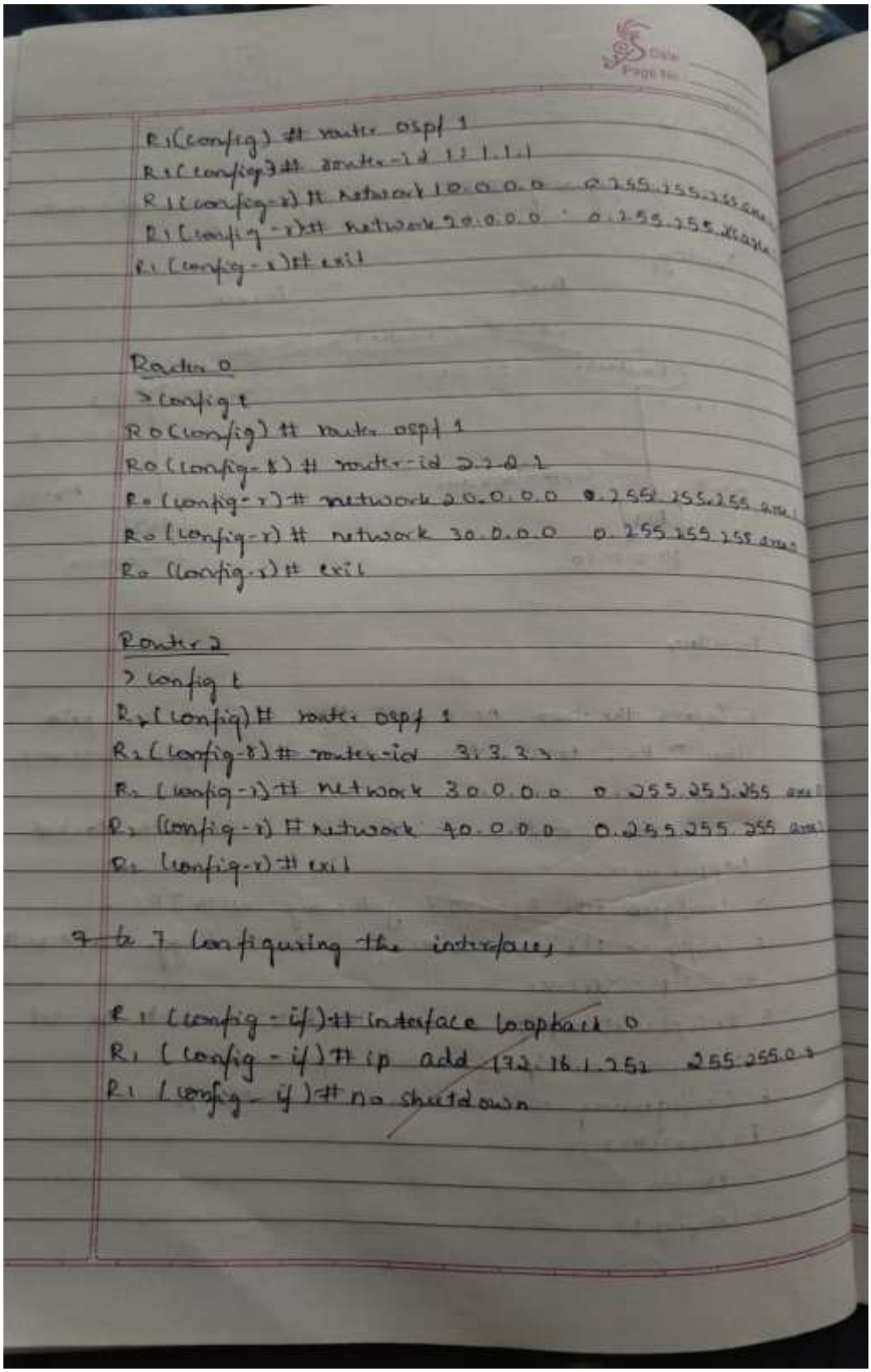
**Program 10**

Aim of the program:

Configure OSPF routing protocol

Procedure and topology:





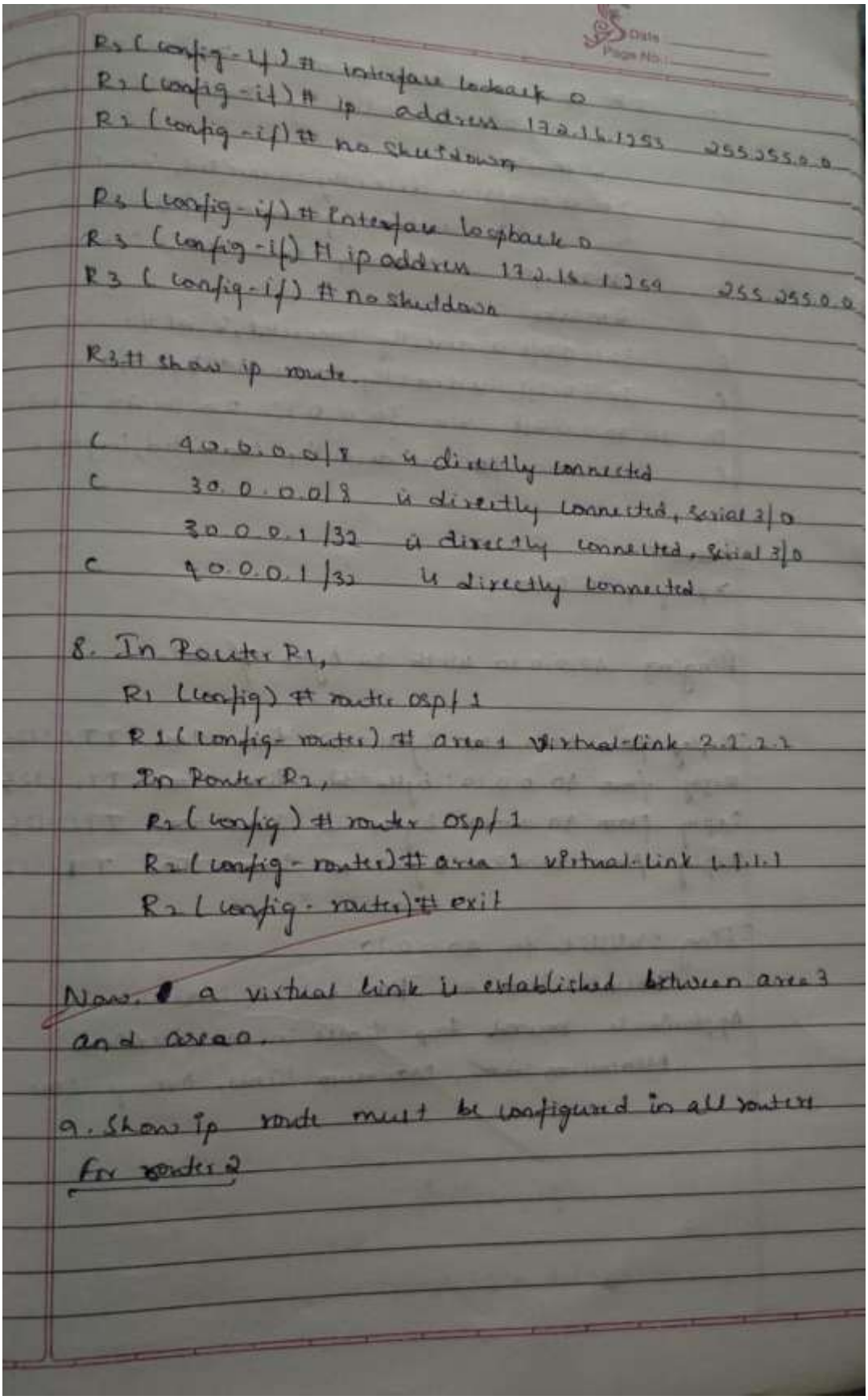
R1(config) # router ospf 1
R1(config)# router-id 1.1.1.1
R1(config-r)# network 10.0.0.0 0.255.255.255 area
R1(config-r)# network 20.0.0.0 0.255.255.255 area
R1(config-r)# exit

Router 0
> config t
R0(config) # router ospf 1
R0(config-r) # router-id 2.2.0.1
R0(config-r)# network 20.0.0.0 0.255.255.255 area
R0(config-r) # network 30.0.0.0 0.255.255.255 area
R0(config-r) # exit

Router 2
> config t
R2(config)# router ospf 1
R2(config-r)# router-id 3.3.3.3
R2(config-r)# network 30.0.0.0 0.255.255.255 area
R2(config-r)# network 40.0.0.0 0.255.255.255 area
R2(config-r)# exit

7 to 7 Configuring the interfaces

R1(config-if)# interface loopback 0
R1(config-if)# ip add 172.16.1.252 255.255.0.0
R1(config-if)# no shutdown



R2 (config-if) # interface loopback 0
R2 (config-if) # ip address 172.16.1.253 255.255.0.0
R2 (config-if) # no shutdown

R3 (config-if) # interface loopback 0
R3 (config-if) # ip address 172.16.1.254 255.255.0.0
R3 (config-if) # no shutdown

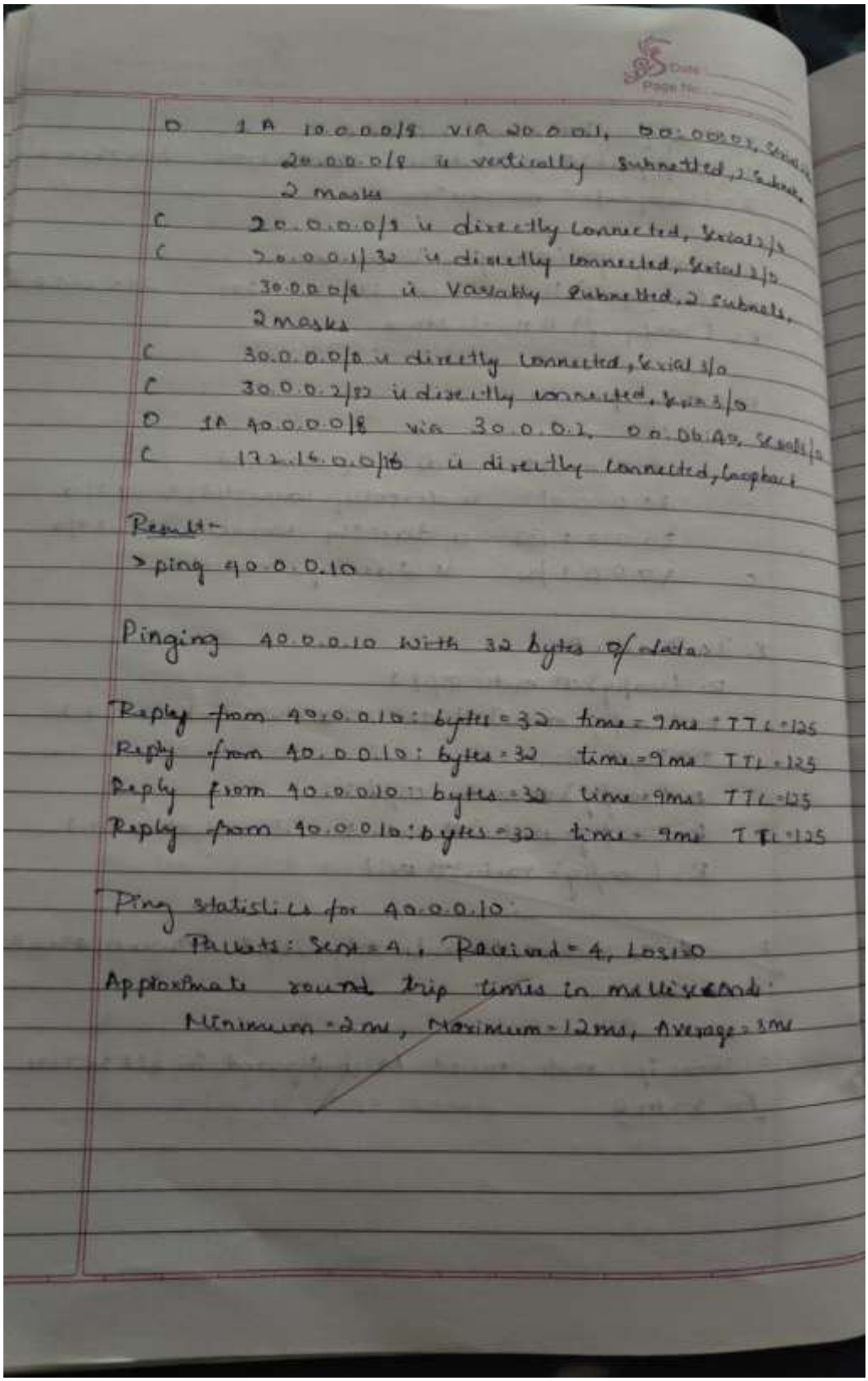
R3 # show ip route

C 90.0.0.0/8 is directly connected
C 30.0.0.0/8 is directly connected, Serial 3/0
30.0.0.1/32 is directly connected, Serial 3/0
C 90.0.0.1/32 is directly connected

8. In Router R1,
R1 (config) # router ospf 1
R1 (config-router) # area 1 virtual-link 2.2.2.2
In Router R2,
R2 (config) # router ospf 1
R2 (config-router) # area 1 virtual-link 1.1.1.1
R2 (config-router) # exit

Now, a virtual link is established between area 3 and area 0.

9. Show ip route must be configured in all routers
for router 2



O IA 10.0.0.0/8 via 20.0.0.1, 00:00:0x, Serial2/0
20.0.0.0/8 is vertically subnetted, 2 subnets, 2 masks
C 20.0.0.0/8 is directly connected, Serial2/0
C 20.0.0.1/32 is directly connected, Serial2/0
30.0.0.0/8 is vertically subnetted, 2 subnets, 2 masks
C 30.0.0.0/8 is directly connected, Serial3/0
C 30.0.0.2/32 is directly connected, Serial3/0
O IA 40.0.0.0/8 via 30.0.0.1, 00:06:40, Serial3/0
C 172.16.0.0/16 is directly connected, Loopback

Result-

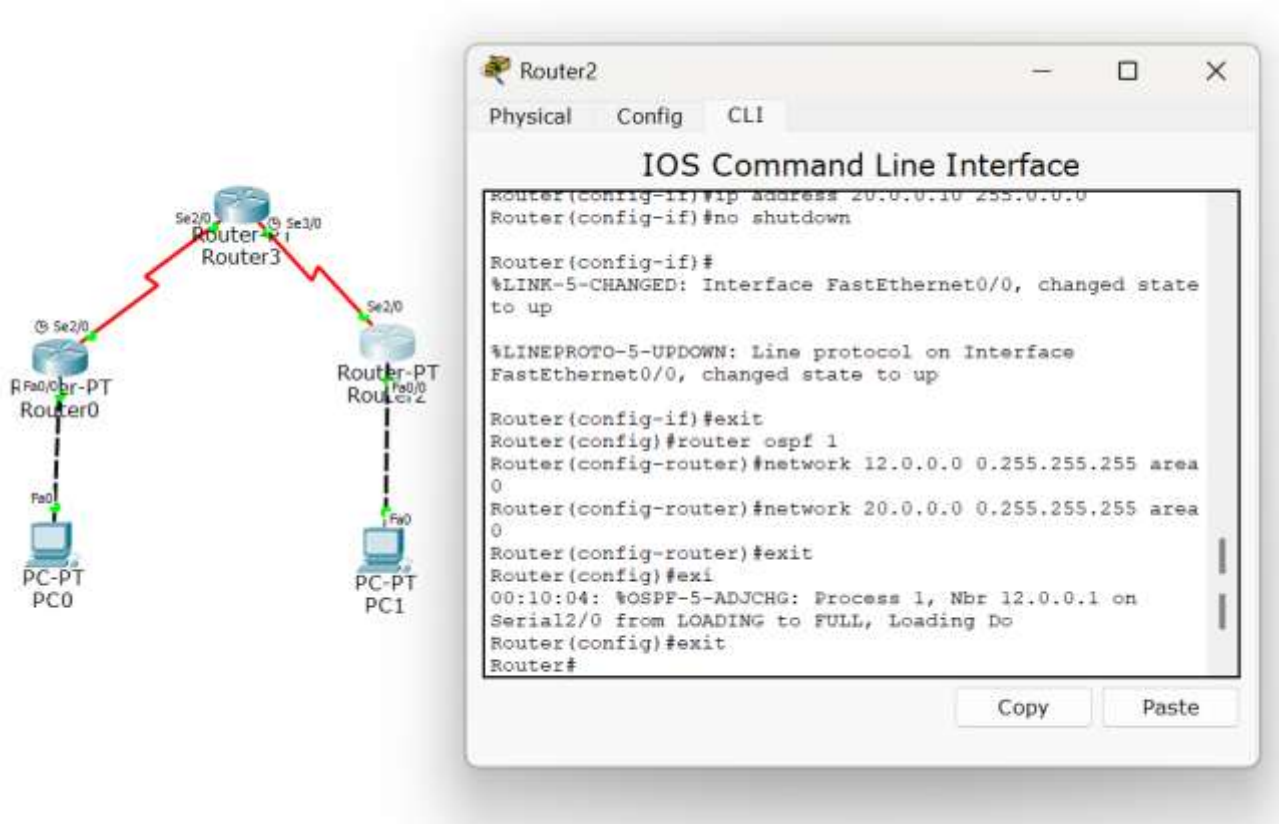
>ping 40.0.0.10

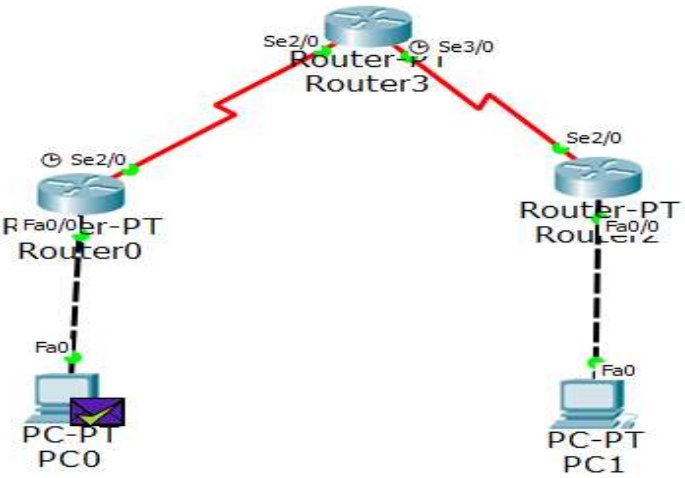
Pinging 40.0.0.10 with 32 bytes of data:

Reply from 40.0.0.10: bytes=32 time=9ms TTL=125
Reply from 40.0.0.10: bytes=32 time=9ms TTL=125
Reply from 40.0.0.10: bytes=32 time=9ms TTL=125
Reply from 40.0.0.10: bytes=32 time=9ms TTL=125

Ping statistics for 40.0.0.10:
Packets: Sent=4, Received=4, Lost=0
Approximate round trip times in milliseconds:
Minimum=2ms, Maximum=12ms, Average=3ms

Screenshots/ Output:




Observation:

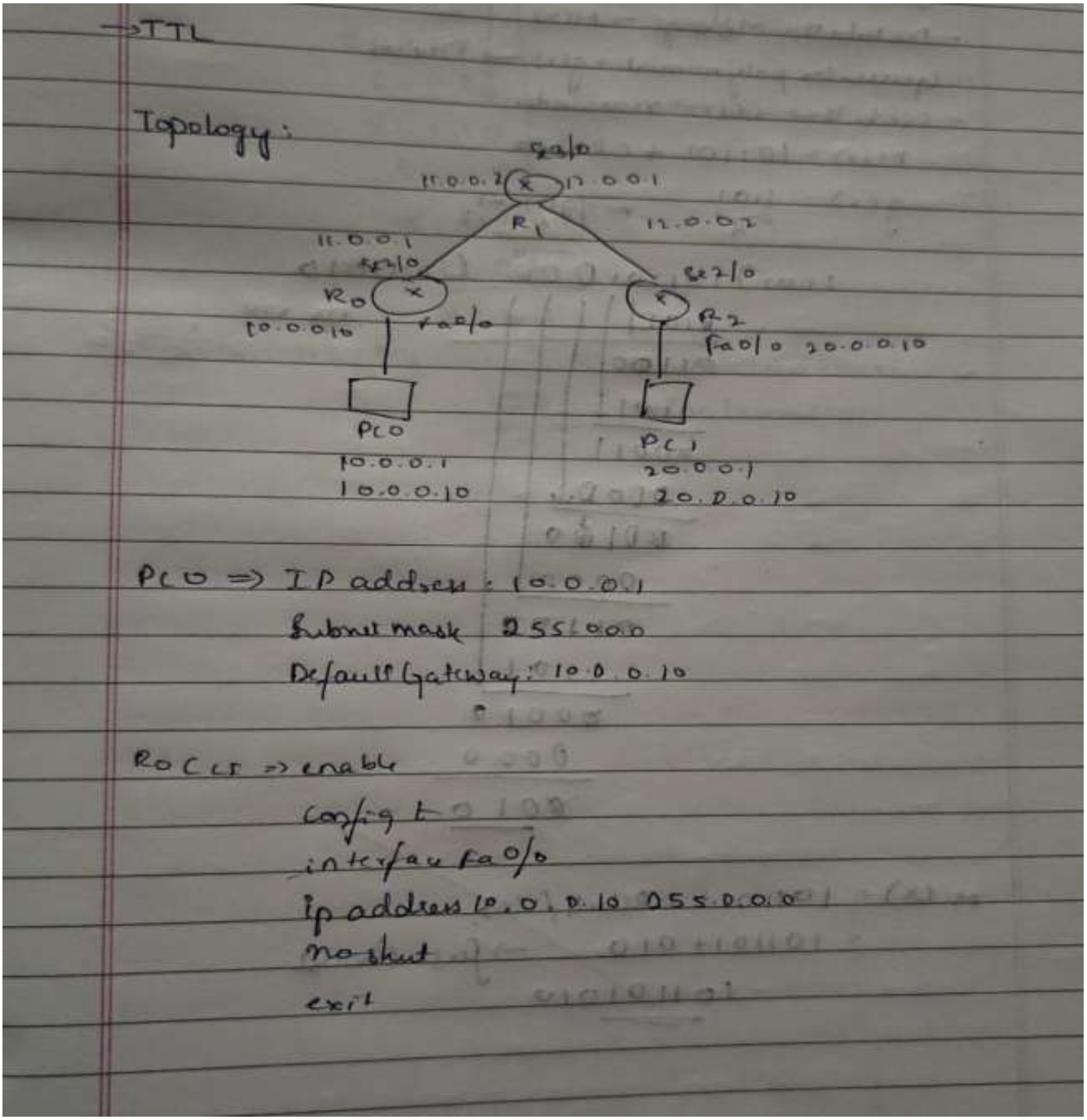
- OSPF successfully established neighbour adjacencies and exchanged LSAs, allowing routers to build a synchronized link-state database across the OSPF area.
- The routing tables converged using SPF calculations, and successful pings confirmed efficient path selection and dynamic route learning through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

**Program 11**

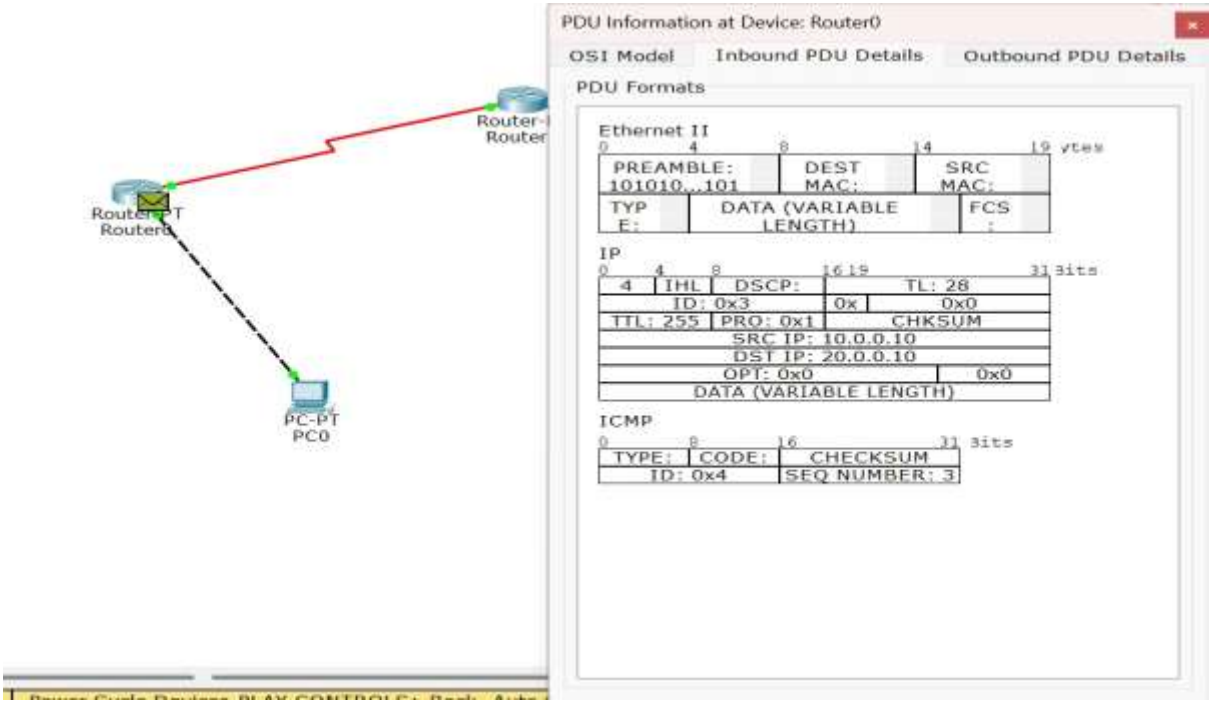
Aim of the program:

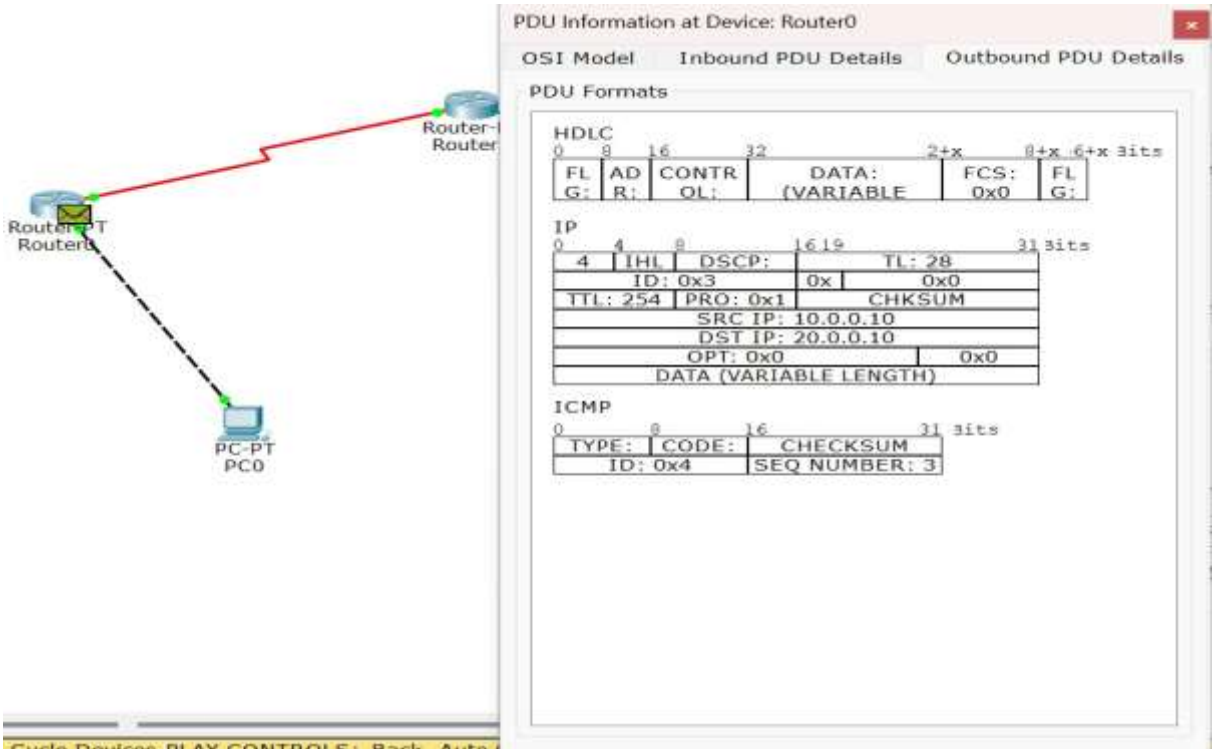
Demonstrate the TTL/ Life of a Packet

Procedure and topology:



Screenshots/ Output:




Observation:

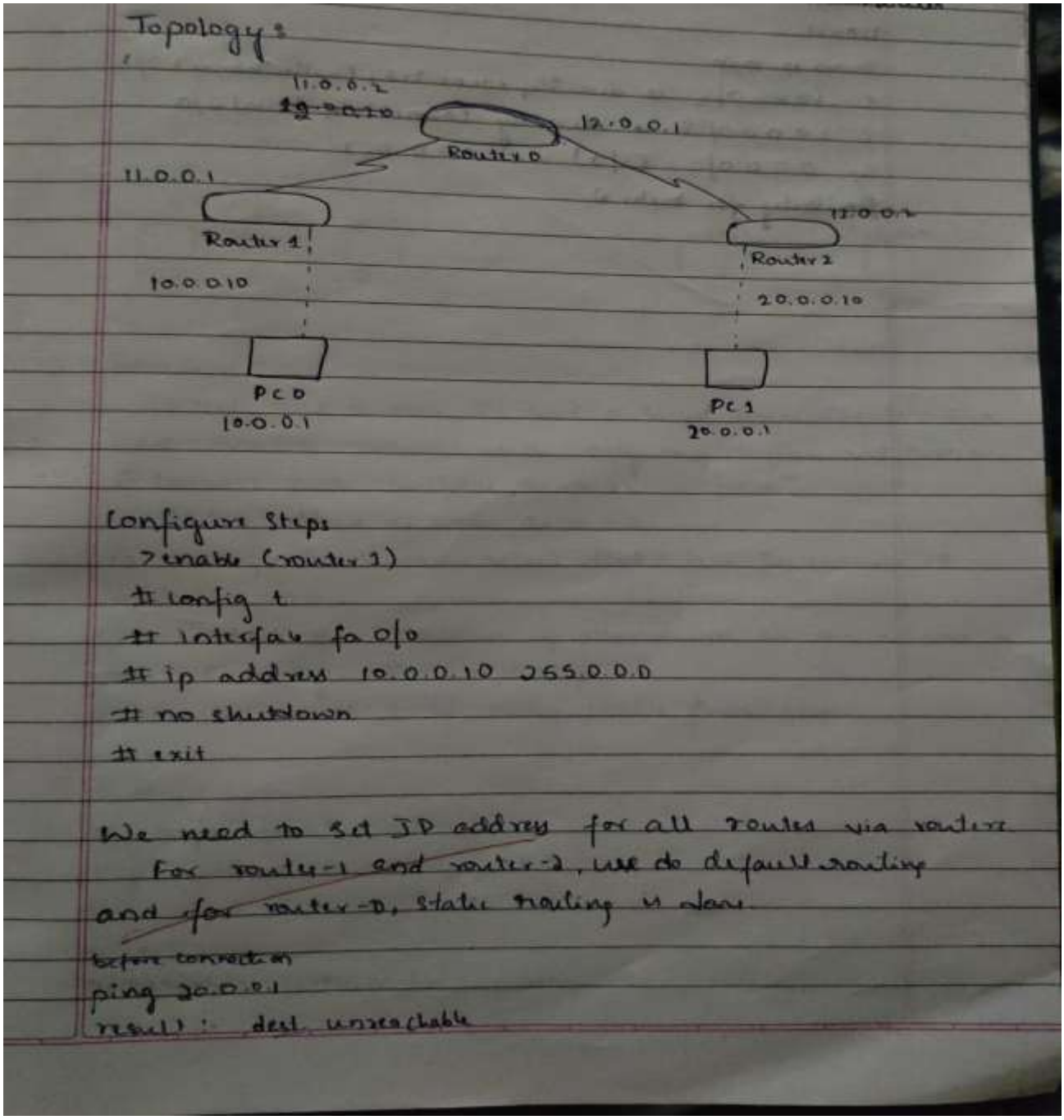
- The TTL field in the IP header decreased by one at each router hop, demonstrating its role in preventing packets from looping indefinitely in the network.

**Program 12**

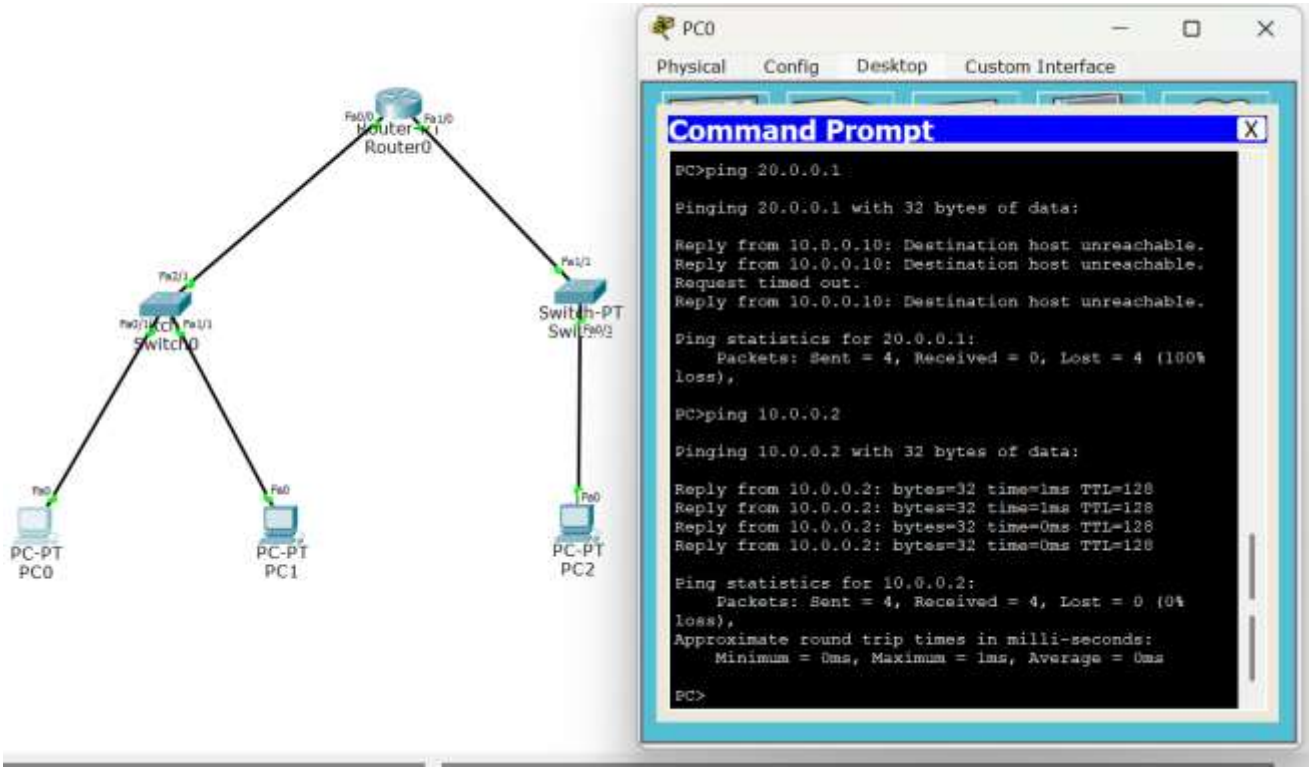
Aim of the program:

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

Procedure and topology:



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.

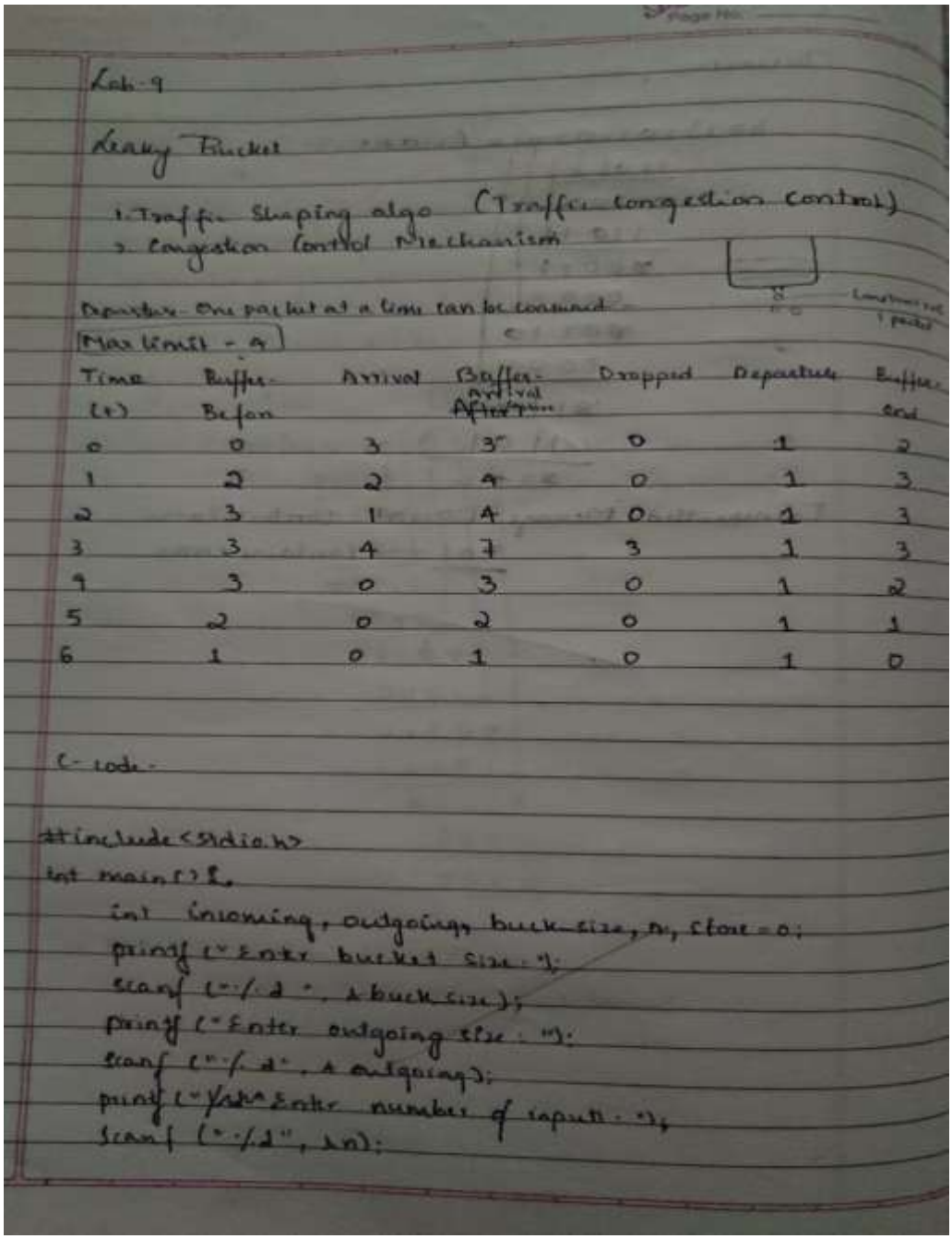
**CYCLE 2:**

**Program 13**

Aim of the program:

Write a program for congestion control using Leaky bucket algorithm

Procedure and topology:



Lab-9

Leaky Bucket

1. Traffic Shaping algo (Traffic congestion control)
2. Congestion control Mechanism

Operation: One packet at a time can be consumed

Max limit = 4

| Time (s) | Buffer Before | Arrival | Buffer Arrival After | Dropped | Departure | Buffer end |
|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 3 | 0 | 1 | 2 |
| 1 | 2 | 2 | 4 | 0 | 1 | 3 |
| 2 | 3 | 1 | 4 | 0 | 1 | 3 |
| 3 | 3 | 4 | 7 | 3 | 1 | 3 |
| 4 | 3 | 0 | 3 | 0 | 1 | 2 |
| 5 | 2 | 0 | 2 | 0 | 1 | 1 |
| 6 | 1 | 0 | 1 | 0 | 1 | 0 |

C-code:

```
#include<stdio.h>
int main(){
    int incoming, outgoing, buck_size, n, store = 0;
    printf("Enter bucket size: ");
    scanf("%d", &buck_size);
    printf("Enter outgoing size: ");
    scanf("%d", &outgoing);
    printf("\nEnter number of inputs: ");
    scanf("%d", &n);
```

```
while(n!=0) {
    printf("Enter the incoming bucket size: ");
    scanf("%d", &incoming);
    if (incoming <= (buck_size - store)) {
        store += incoming;
        printf("Bucket buffer size %d out of %d \n",
            store, buck_size);
    }
    else {
        printf("Dropped %d no of packets \n",
            incoming - (buck_size - store));
        printf("Bucket buffer size %d out of %d \n",
            store, buck_size)
    }
    store = store - outgoing;
    printf("After outgoing %d packets left out
    of %d in buffer \n", store, buck_size);
    n--;
} }
```

Output:

Enter bucket size: 5000
Enter outgoing rate: 2000
Enter number of input: 2
Enter the incoming packet size: 3000
Bucket buffer size 3000 out of 5000
After outgoing 1000 packets left out of 5000 in buffer.
Enter the incoming packet size: 1000
Bucket buffer size 2000 out of 5000
After outgoing 0 packets left out of 5000 in buffer.

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:

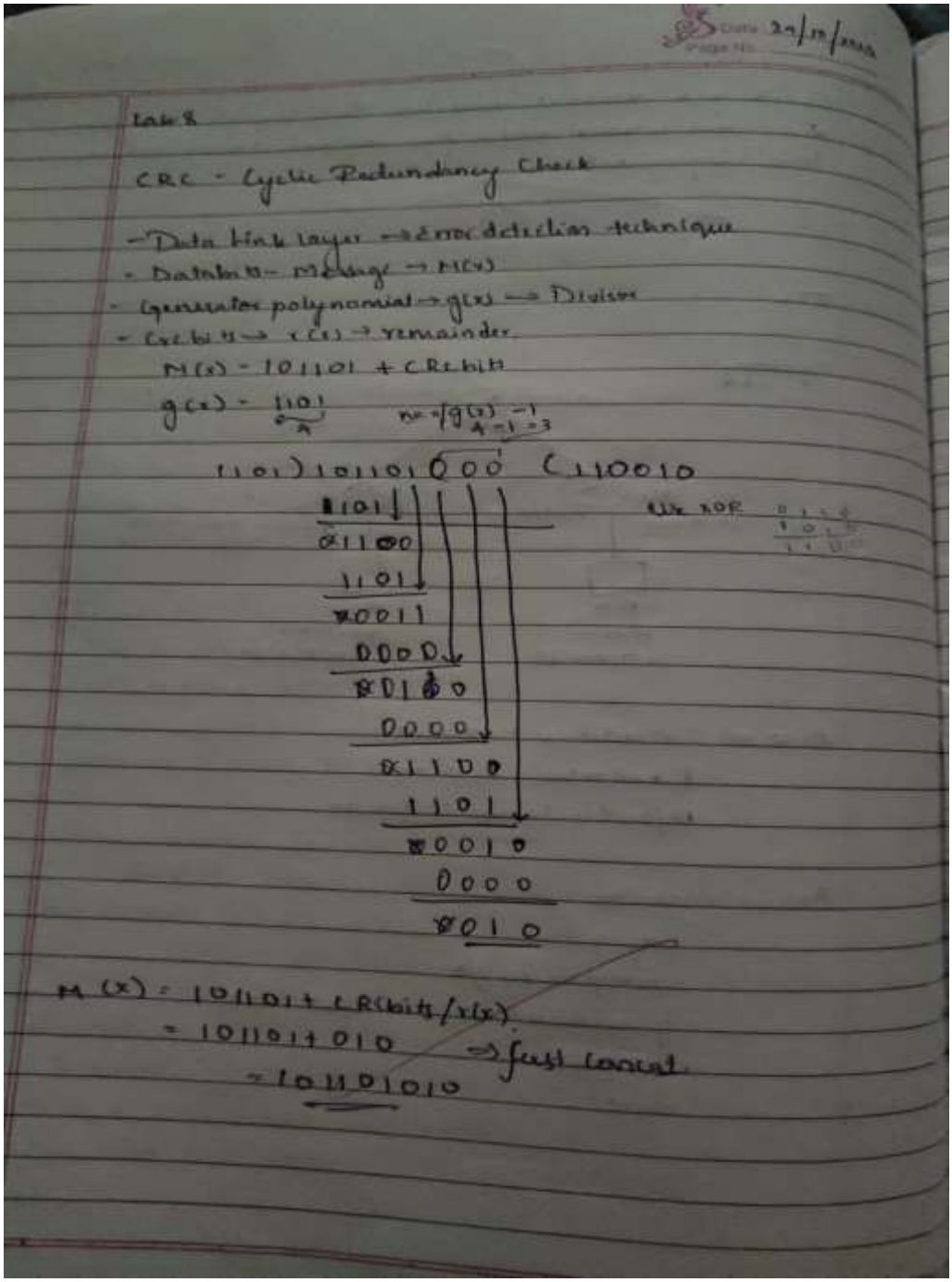
- The leaky bucket mechanism regulated the outgoing packet flow at a constant rate, preventing sudden traffic bursts from overwhelming the network.
- Packets exceeding the bucket capacity were dropped, demonstrating effective congestion control by smoothing traffic and enforcing rate-limiting.

**Program 14**

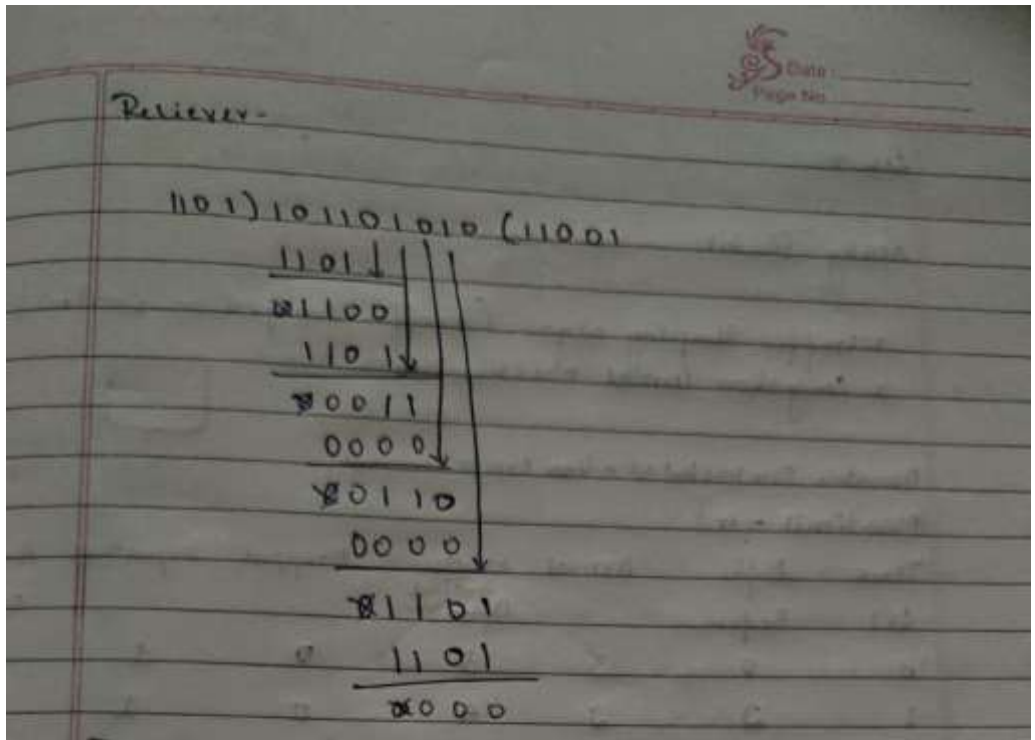
Aim of the program:

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

Procedure and topology:



Reciever-




Transmitted Message Doesn't contain Errors

001 → Contains errors.

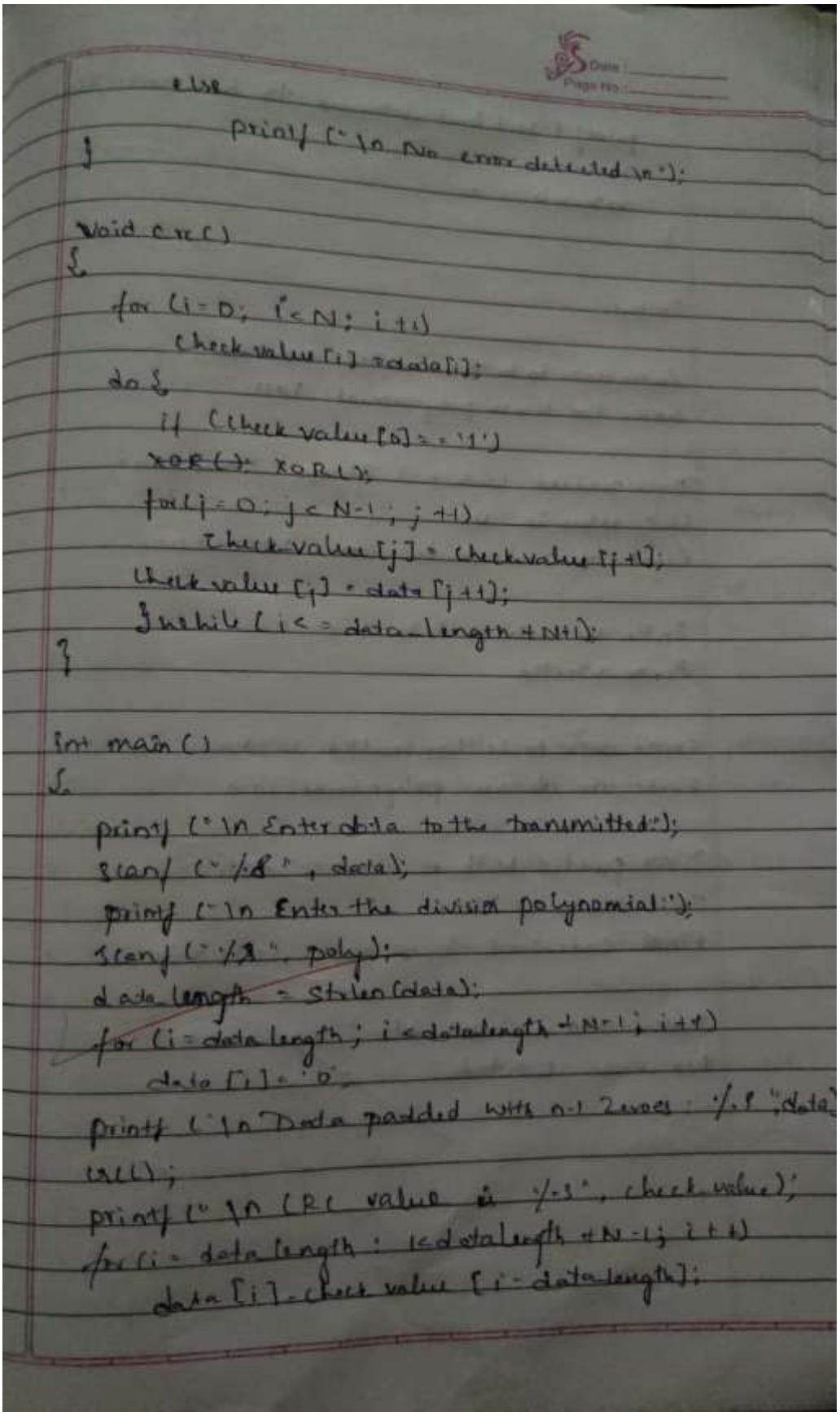
Lab - 8 continuation

CRC Implementation:

WAP for error detecting code using CRC-CCITT

C-code

```
#include<stdio.h>
#include<string.h>
#define N strlen(poly)

char data[30];
char check_value[30];
char poly[10];
int datalength,i,j;

void XOR
{
   for(j=1; j<N; j++)
   checkvalue[j] = ((check_value[j]==poly[j])?'0':'1');
}

Void receive()
{
   printf("Enter the received data:");
   scanf("%s", data);
   printf("Data received: %s", data);
   for(i=0; (i<N-1) && (checkvalue[i]!='1'); i++);
   if (i<N-1)
      printf("\n Error detected \n");
```



```
else
    printf("\n No error detected\n");
}

void crc()
{
  for(i=0; i<N; i++)
     checkvalue[i]=data[i];
  do{
     if(checkvalue[0]=='1')
     xor();
     for(j=0; j<N-1; j++)
        checkvalue[j]=checkvalue[j+1];
     checkvalue[j]=data[j+1];
  }while(i<=datalength+N+1);
}

int main()
{
  printf("\n Enter data to be transmitted:");
  scanf("%s", data);
  printf("\n Enter the division polynomial:");
  scanf("%s", poly);
  datalength = strlen(data);
  for(i=datalength; i<datalength+N-1; i++)
     data[i]='0';
  printf("\n Data padded with n-1 zeroes : %s", data);
  crc();
  printf("\n CRC value is %s", checkvalue);
  for(i=datalength; i<datalength+N-1; i++)
     data[i]=checkvalue[i-datalength];
```

printf("\n Final dataword to be sent: %s\n", received);
return 0;
}

Output:

Enter data to be transmitted: 101010
Enter the divisor polynomial: 1011

Data padded with n-1 zeroes: 101010000
CRC value is: 001
Final codeword to be sent: 101010001

Enter the received data: 100010000
Error detected

Enter data to be transmitted: 101100
Enter the divisor polynomial: 1001

Data padded with n-1 zeroes: 101100000
CRC value is: 001
Final codeword to be sent: 101100001

Enter the received data: 101100001
No error detected.

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:

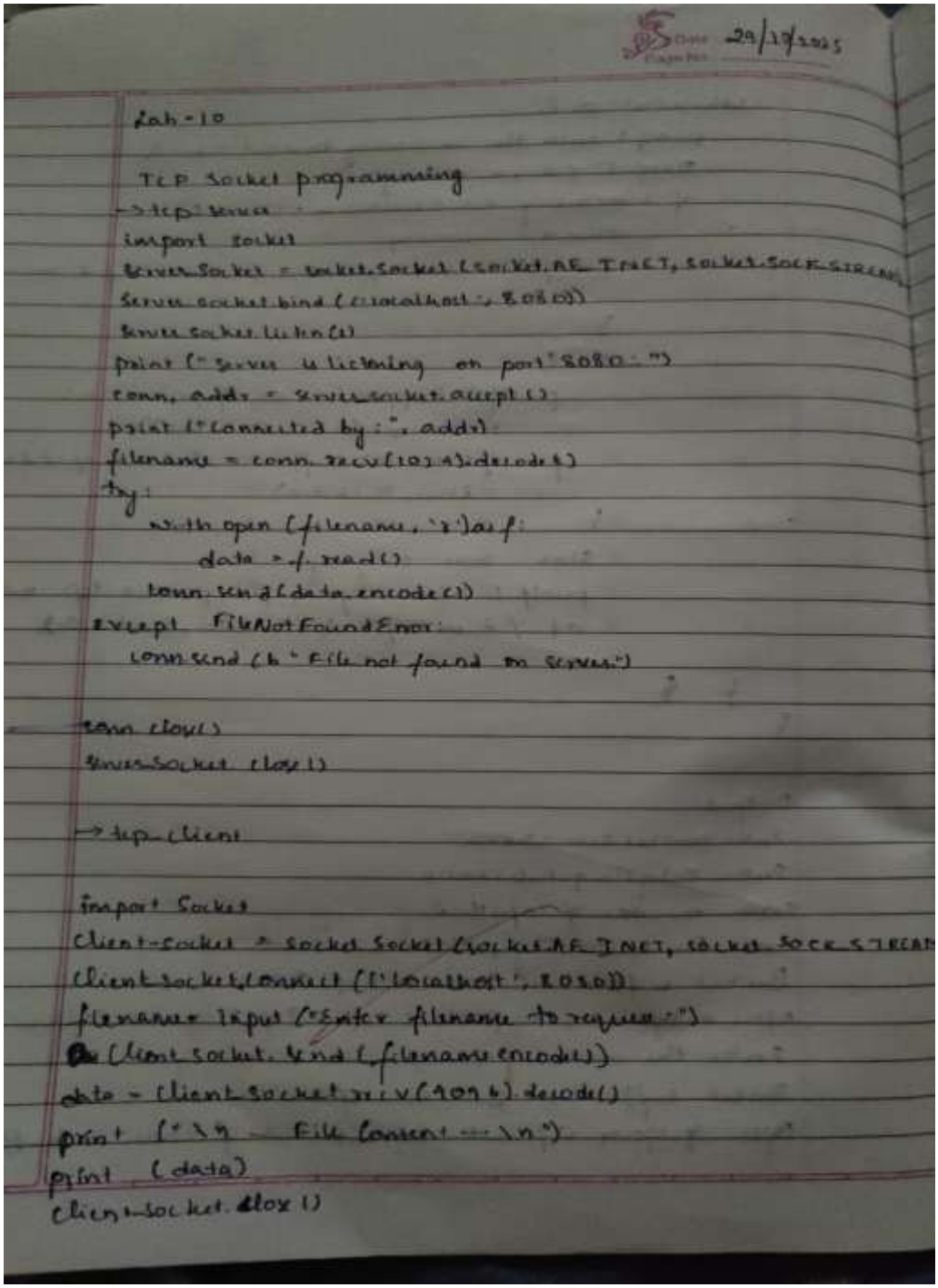
- The CRC-CCITT (16-bit) algorithm correctly generated a checksum for the transmitted data, ensuring reliable detection of single-bit and burst errors.
- Intentional error tests produced mismatched CRC values at the receiver, confirming accurate error detection through polynomial division.

**Program 15**

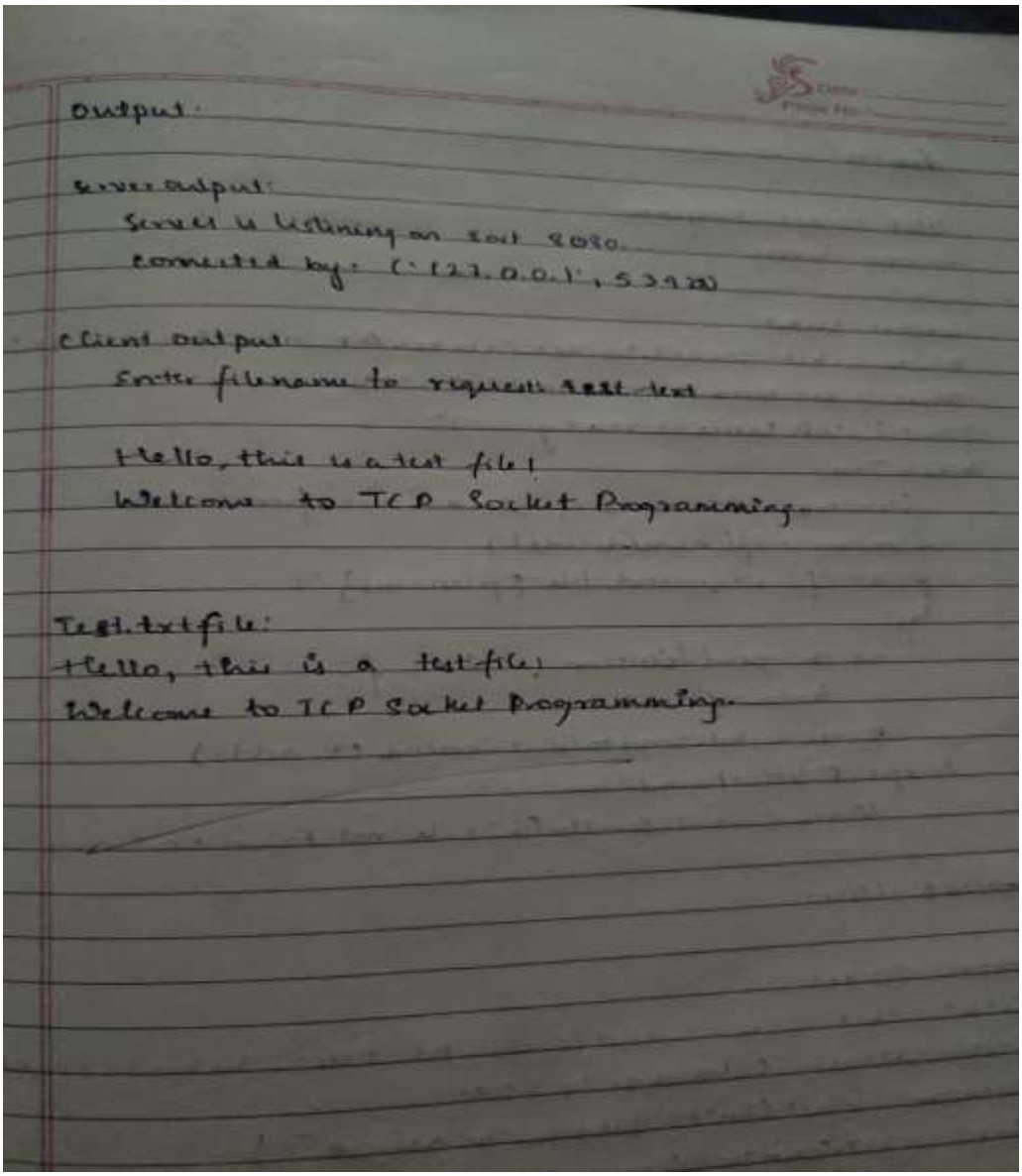
Aim of the program:

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:

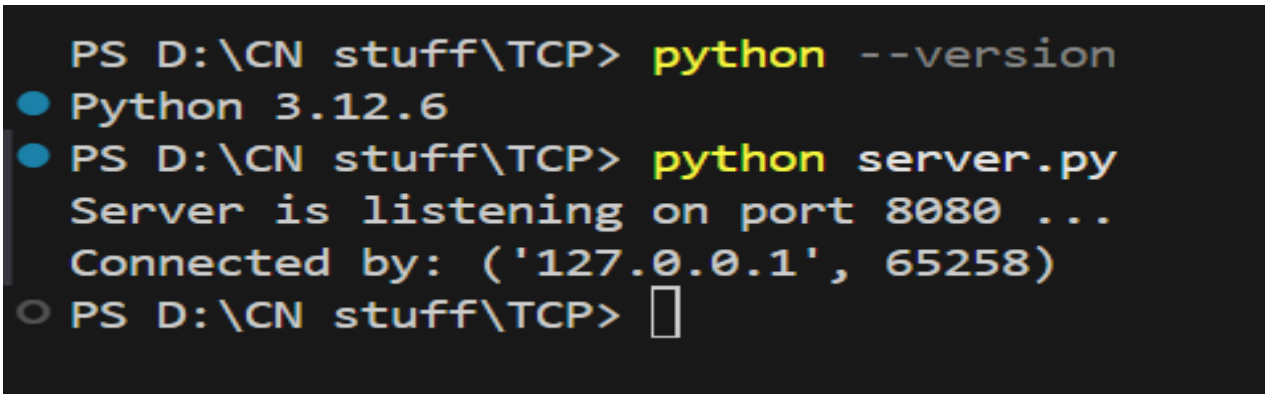


```
29/10/2025

Lab-10

TCP Socket programming
->tcp_server
import socket
ServerSocket = 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("File not found on server.")

conn.close()
ServerSocket.close()

->tcp_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()
```




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>
```




Observation:

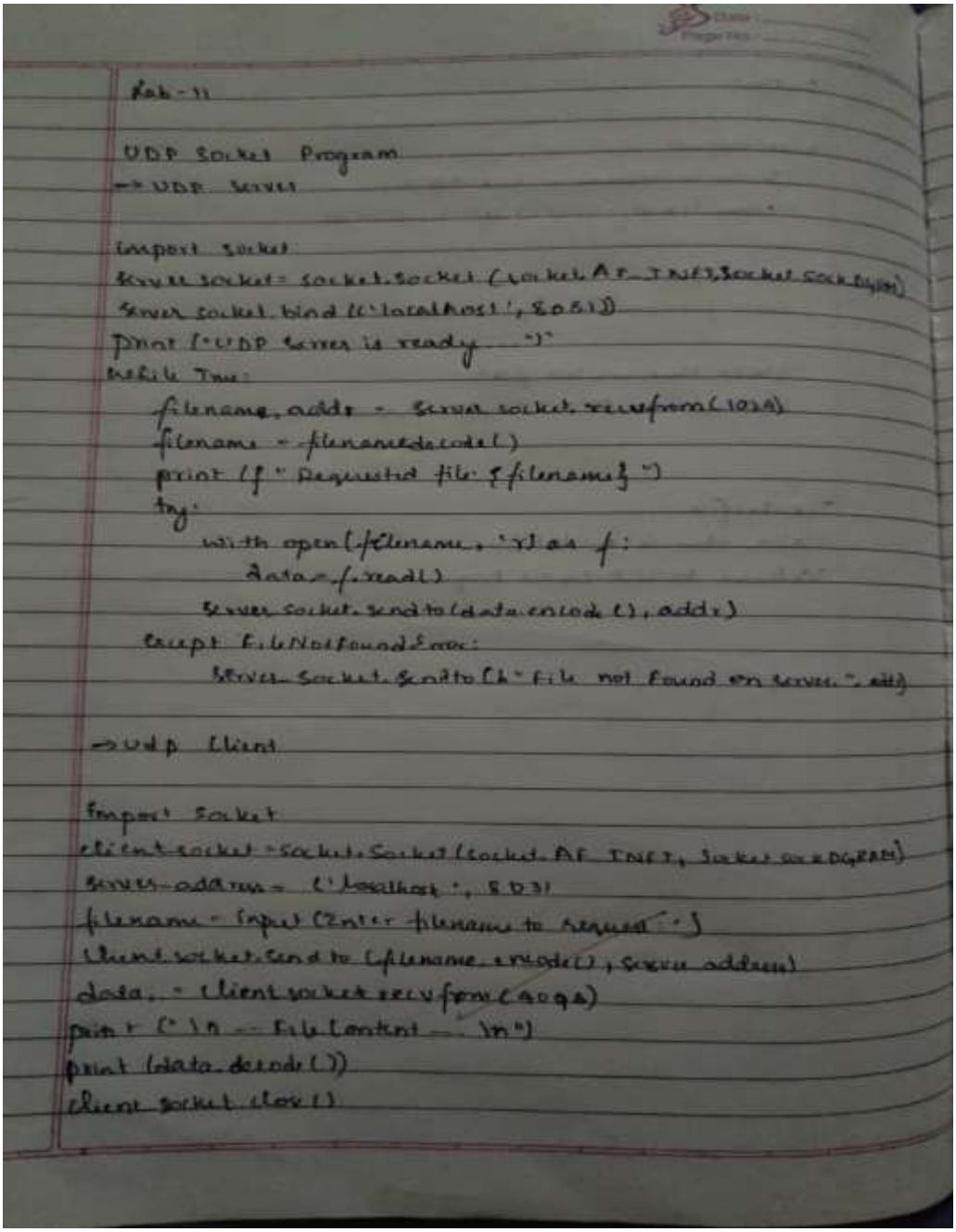
- The TCP client successfully established a reliable connection with the server and transmitted the requested filename using stream-oriented communication.
- The server correctly retrieved and returned the file contents over the same TCP session, demonstrating reliable data transfer, acknowledgment handling,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



Lab-11

UDP Socket Program

-> UDP Server

```
import socket
Server socket = socket.socket (socket.AF_INET, Socket.SOCK_DGRAM)
Server socket.bind (('localHost', 8031))
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)
```

-> udp Client

```
import Socket
client socket = Socket.Socket (socket.AF_INET, Socket.SOCK_DGRAM)
Server-address = ('localhost', 8031
filename = input ("Enter filename to request :")
client socket.sendto (filename.encode(), Server address)
data, = client socket.recvfrom (4096)
print ("\n -- File Content -- \n")
print (data.decode())
client socket.close()
```

Output:

info.txt file:

Hello, this is UDP file transfer.
UDP is connectionless but reliable for small data.

ex1:

Server Output:

UDP Server is ready...
Requested file: info.txt

Client output:

Enter filename to request: info.txt

Hello, this is UDP file transfer.
UDP is connectionless but reliable for small data.

ex2:

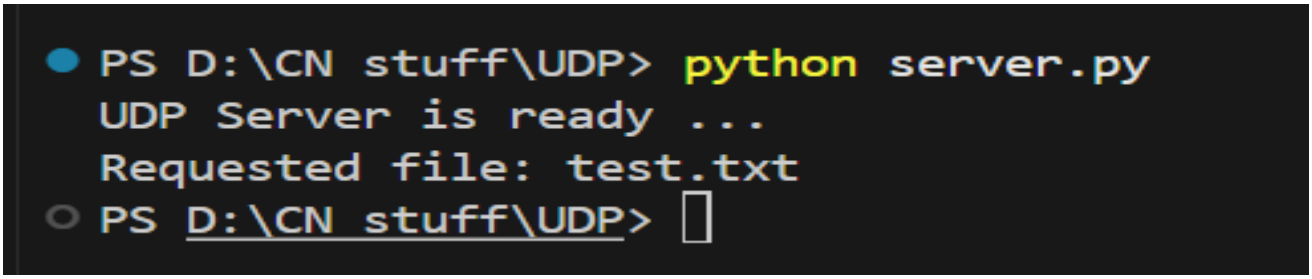
Server output:

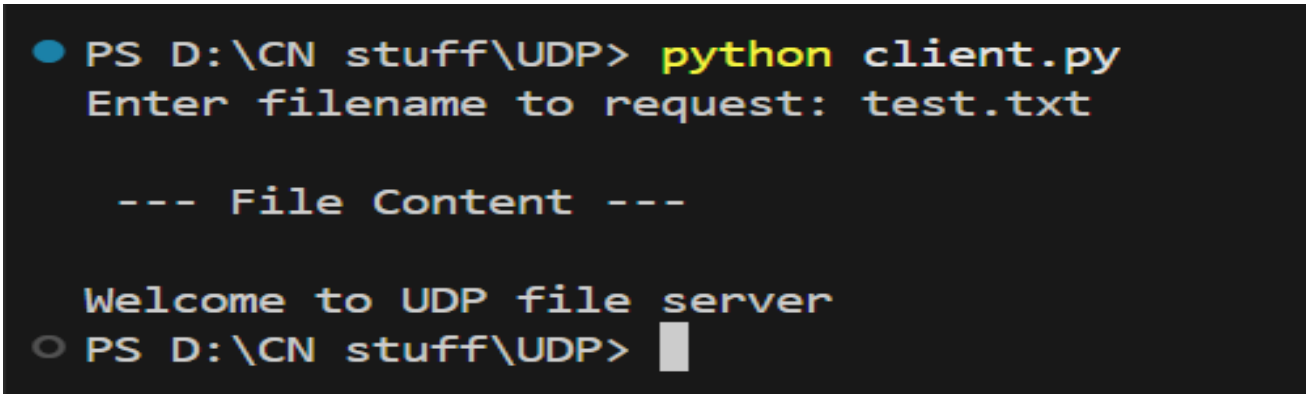
UDP Server is ready...
Requested file: nofile.txt

Client output:

Enter filename to request: nofile.txt
File not found on server.

Screenshots/ Output:




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.