

B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU



Lab Record

Computer Networks – 23CS5PCCON

Submitted in partial fulfillment for the 5th Semester Laboratory

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

DHARUNYA BALAVELAVAN

(1BM23CS090)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
August 2025-December 2025

B.M.S. COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING



CERTIFICATE

This is to certify that the Computer Networks (23CS5PCCON) laboratory has been carried out by Dharunya Balavelavan (1BM23CS090) during the 5th Semester August 2025-December 2025.

Signature of the Faculty Incharge:

Sarala D V
Assistant Professor

Department of Computer Science and Engineering

Table of Contents

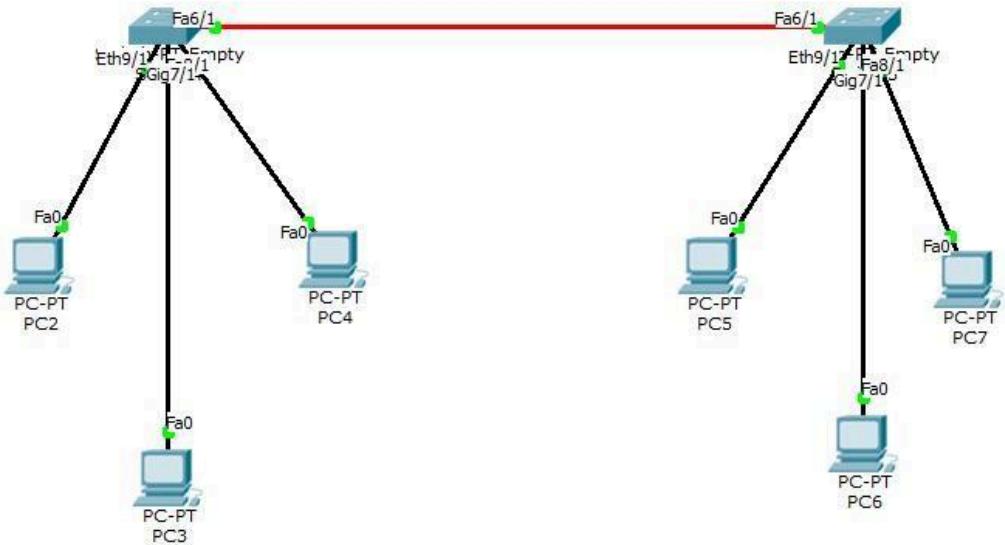
PART - A	
Serial No.	Name of Experiment
1.	Create a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices and demonstrate ping messages.
2.	Configure DHCP within a LAN and outside LAN.
3.	Configure Web Server, DNS within a LAN.
4.	Configure IP address to routers in packet tracer. Explore the following messages: ping responses, destination unreachable, request timed out, reply.
5.	Configure default route, static route to the Router.
6.	Configure RIP routing Protocol in Routers.
7.	Configure OSPF routing protocol.
8.	To construct a VLAN and make the PC's communicate among a VLAN.
9.	To construct a WLAN and make the nodes communicate wirelessly.
10.	Demonstrate the TTL/ Life of a Packet.
11.	To understand the operation of TELNET by accessing the router in the server room from a PC in the IT office.
12.	To construct a simple LAN and understand the concept and operation of Address Resolution Protocol (ARP).

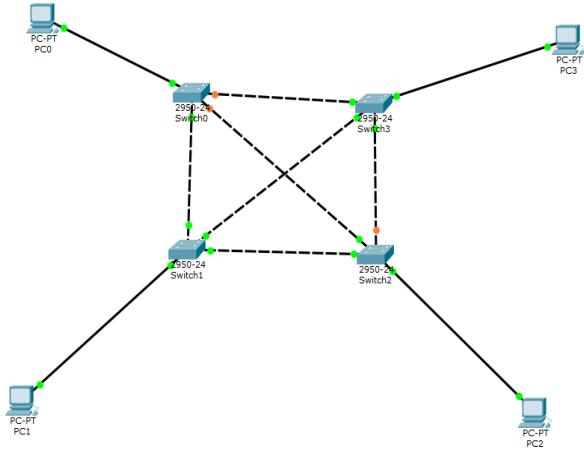
PART – B	
Serial No.	Name of Experiment
1.	Write a program for congestion control using Leaky bucket algorithm.
2.	Using TCP/IP sockets, write a client-server program to make the client send the file name and the server to send back the contents of the requested file if present.
3.	Using UDP sockets, write a client-server program to make the client send the file name and the server to send back the contents of the requested file if present.
4.	Write a program for error detecting code using CRC-CCITT (16-bits).

PART - A

Program 1: Create a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices and demonstrate ping messages.

Network diagram:





Configuration:

20/8/25

Lab 2

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

→ 1 Generic switch connected to 4 laptops

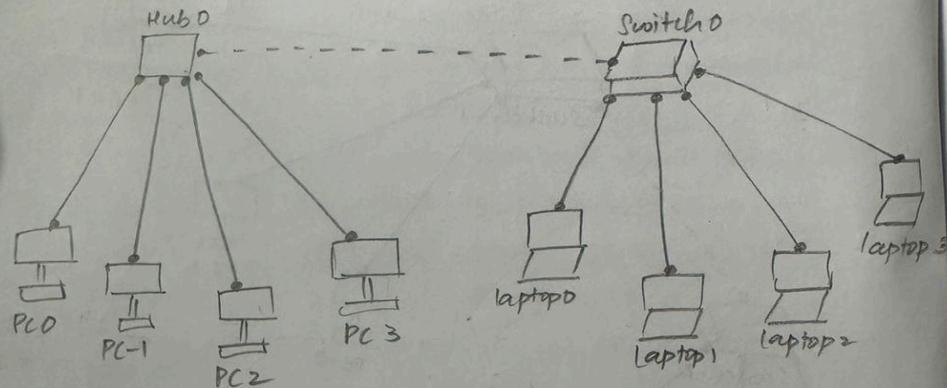
1 Generic Hub connected to 4 PC's

IP Addresses

for PC's 192.162.10.1 2 3 4

for laptop 192.162.20.1 2 3 4

1 PDU from PLC - PC1



→ 4 Generic switches

1 ~~switch~~ switch connected to 1 PC's.

IP Address

for PC's 192.162.10.1 2 3 4

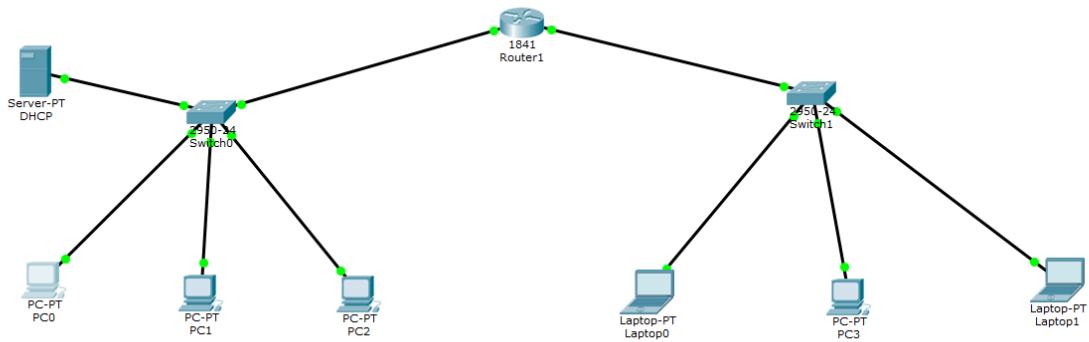
1 PDU from PLC - PC1

1 PDU from PC2 - PC3

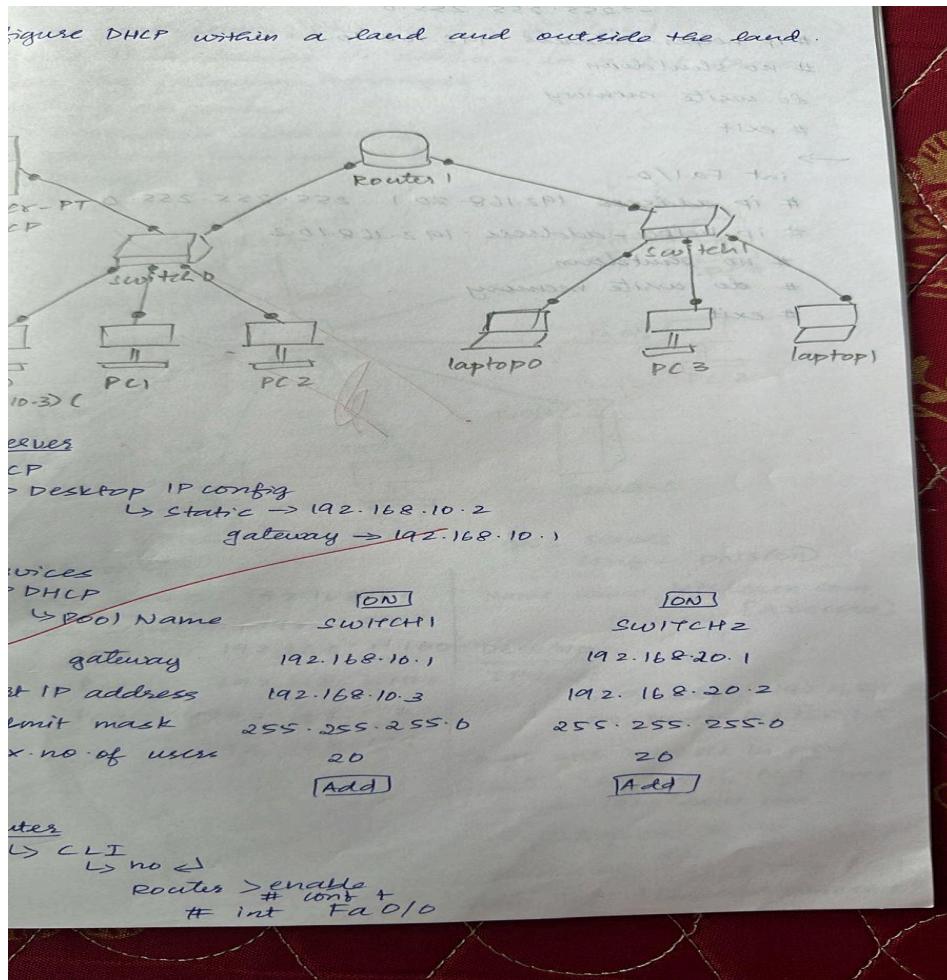
⇒

Program 2: Configure DHCP within a LAN and outside LAN.

Network diagram:



Configuration:



```

# ip address 192.168.10.10
      -> 255.255.255.0
# ip helpers - address 192.168.10.2
# no shutdown
# do write memory
# exit.

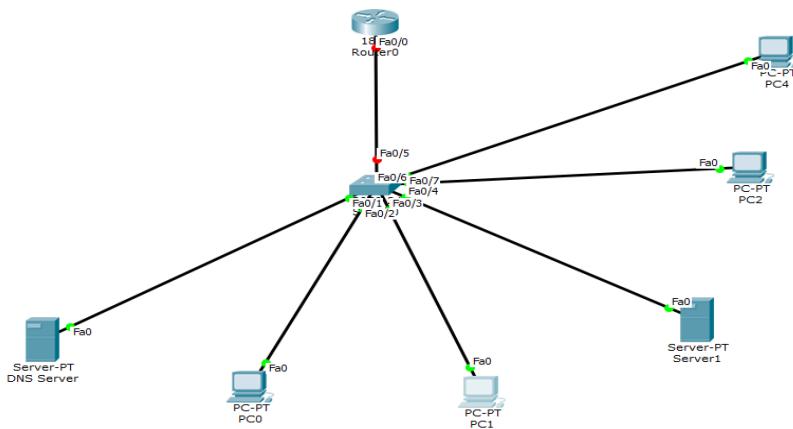
→
int Fa1/0
# ip address 192.168.20.1 255.255.255.0
# ip helpers - address 192.168.10.2
# no shutdown
# do write memory
# exit

```

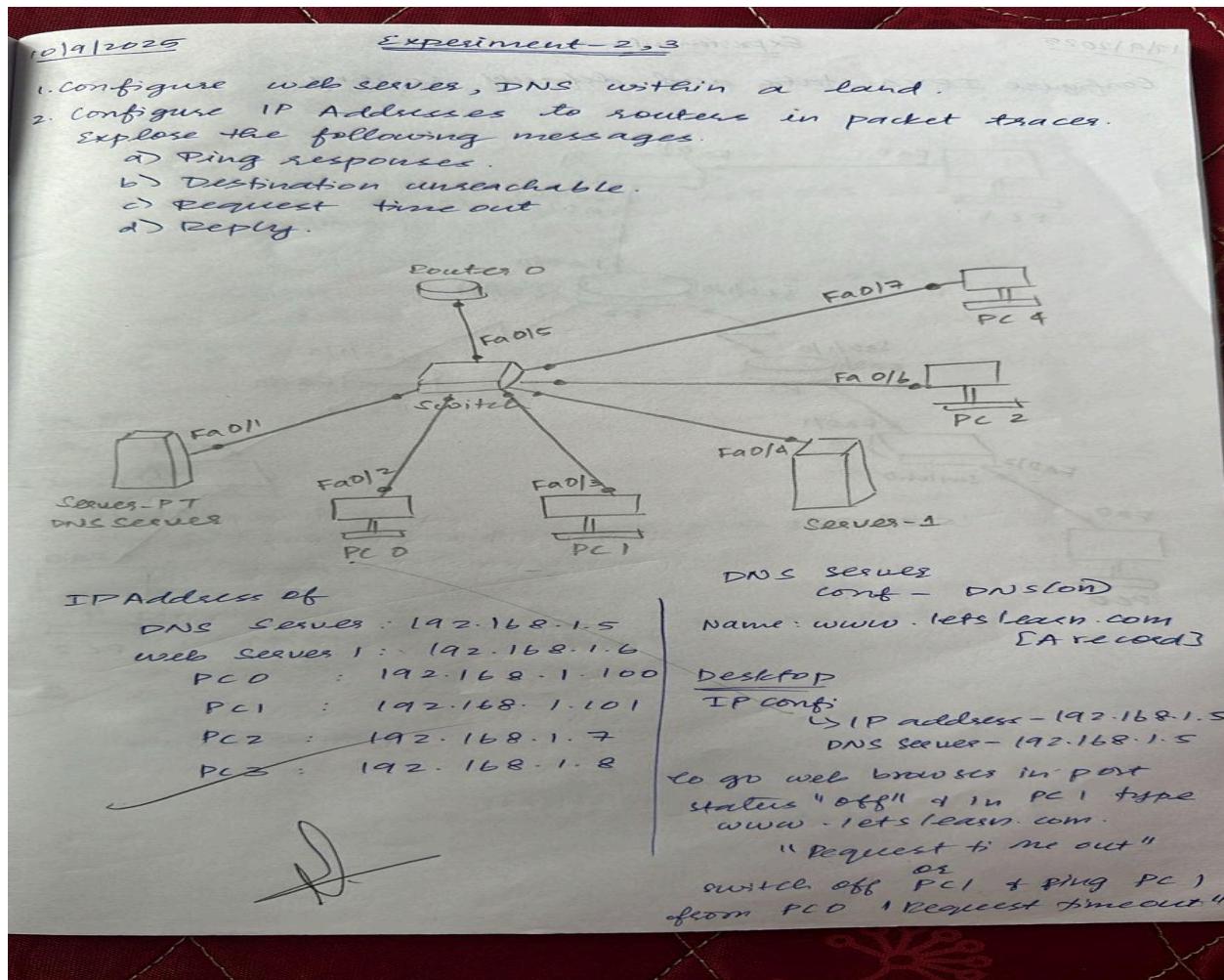
↙

Program 3: Configure Web Server, DNS within a LAN.

Network diagram:

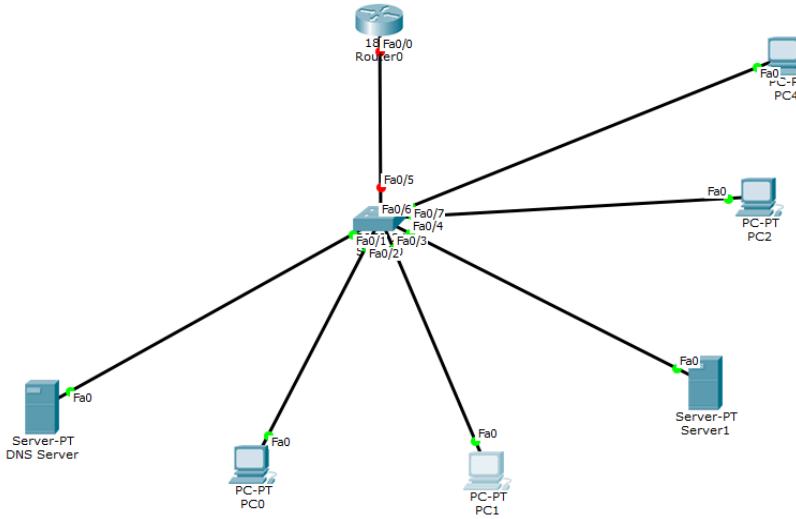


Configuration:



Program 4: Configure IP address to routers in packet tracer. Explore the following messages: ping responses, destination unreachable, request timed out, reply.

Network diagram:

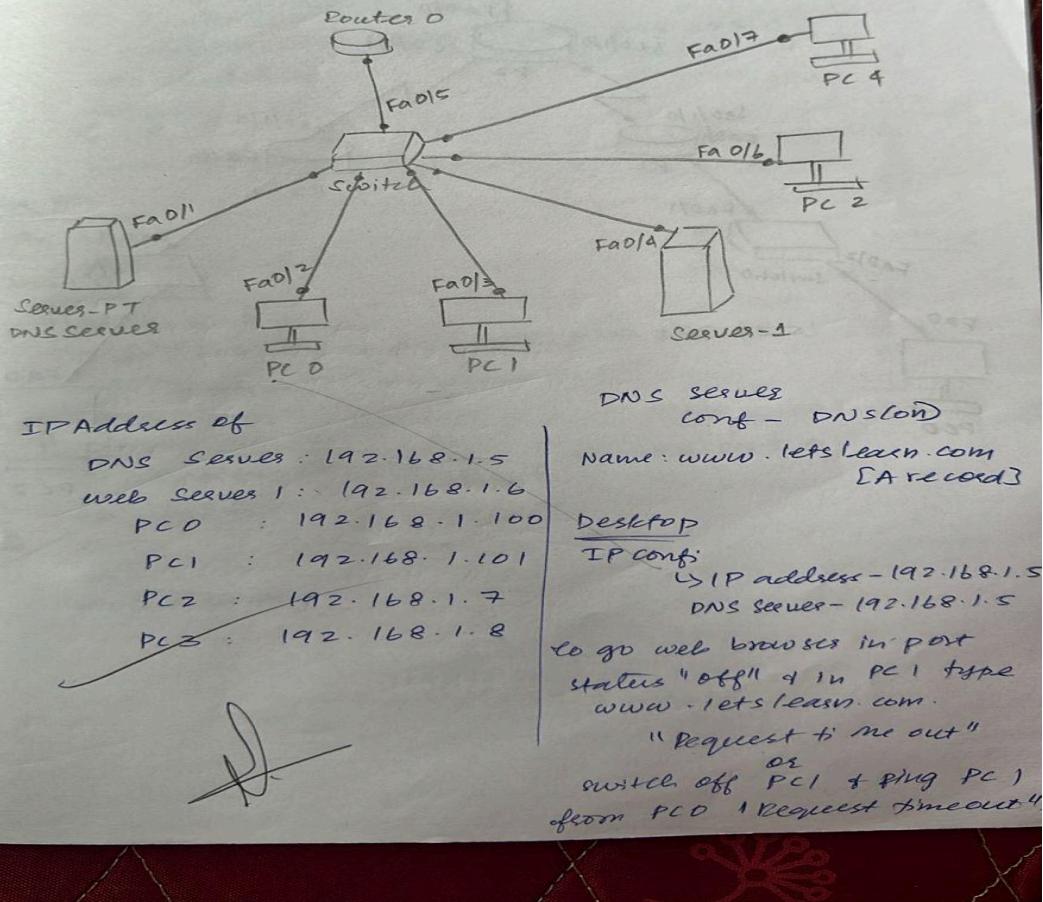


Configuration:

10/9/2025

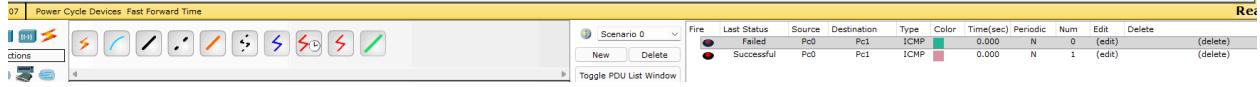
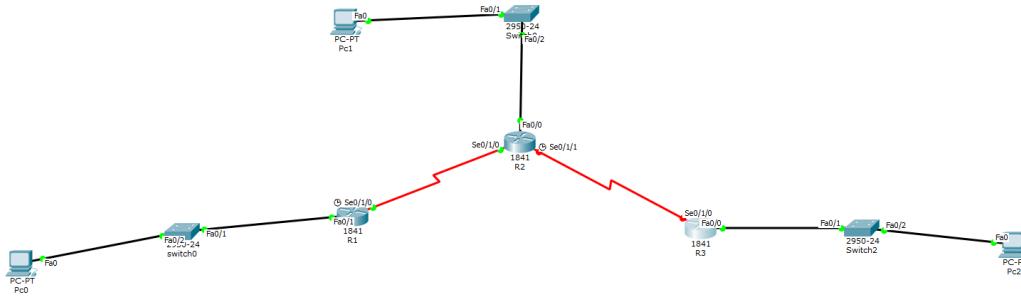
Experiment - 2, 3

1. Configure web servers, DNS within a lab.
2. Configure IP Addresses to routers in packet traces.
Explore the following messages.
 - a) Ping responses.
 - b) Destination unreachable.
 - c) Request time out
 - d) Reply.

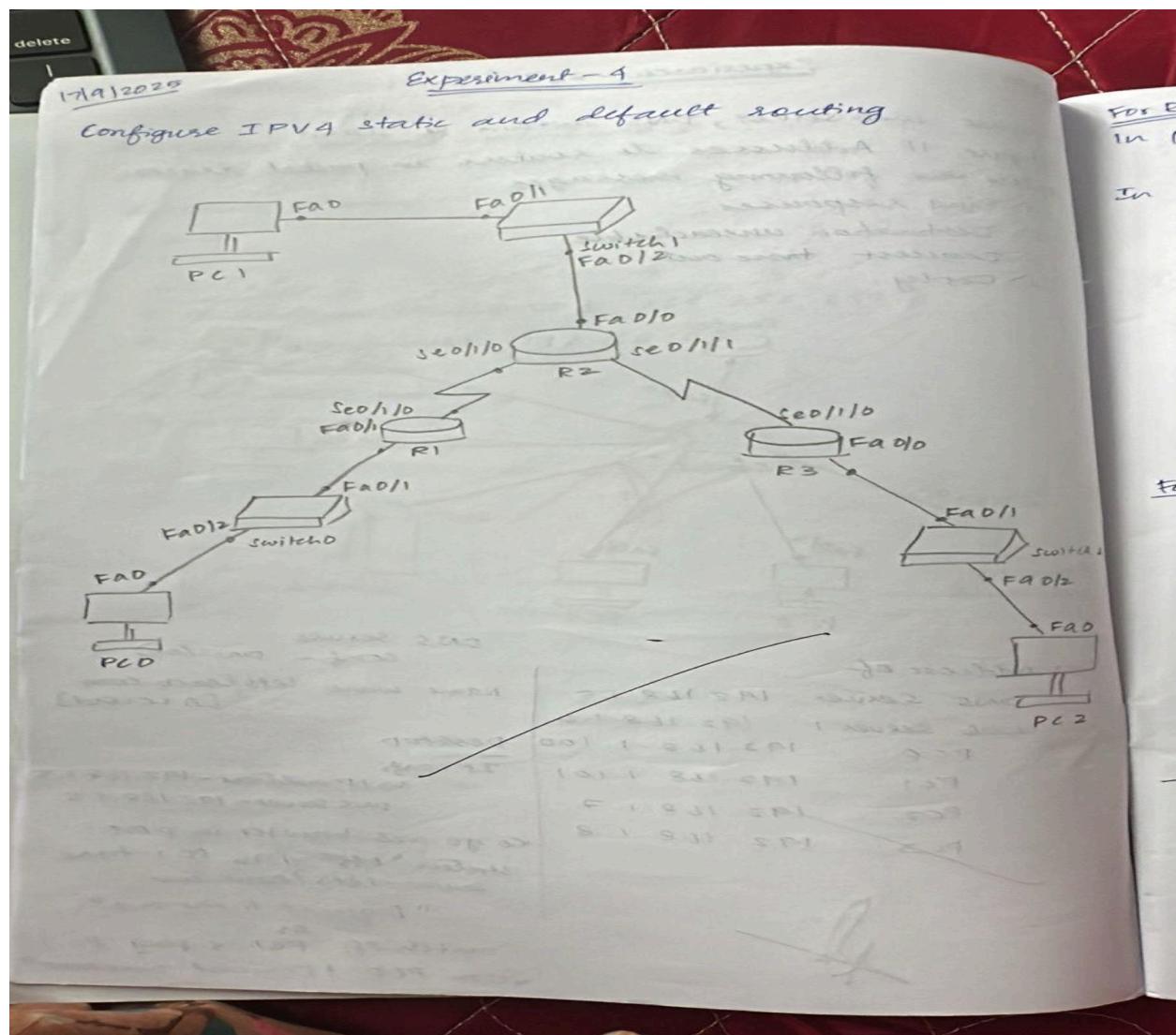


Program 5: Configure default route, static route to the Router.

Network diagram:



Configuration:



For R1

In physical → off switch
drag & drop HWIC-2+

In CLI,

enable

conf t

int Se0/1/0

ip address 172.16.1.1 255.255.255.252

no shutdown

exit

int Fa0/1

ip address 192.168.10.1 255.255.255.0

no shutdown

exit

exit

write memory

For R2

In CLI,

enable

conf t

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

exit

write memory

In CLI

```

enable
conf t
int S0/1/0
ip address 192.168.2.2 255.255.255.252
no shutdown
exit
int Fa 0/0
ip address 192.168.30.1 255.255.255.0
no shutdown
exit
exit
write memory

```

In PC0

desktop → static

- ↳ IP address : 192.168.10.10
- ↳ default gateway : 192.168.10.1

In PC1

desktop → static

- ↳ IP : 192.168.20.10
- ↳ default gateway : 192.168.20.1

In PC2

desktop → static

- ↳ IP : 192.168.30.10
- ↳ default gateway : 192.168.30.1

In R1

CLI

```

enable
conf t
ip route 192.168.20.0 255.255.255.0 172.16.1.2
ip route 192.168.20.0 255.255.255.252 172.16.1.2
ip route 192.168.30.0 255.255.255.0 172.16.1.2

```

In 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

```

In R3

CLI

```

enable
conf t
ip route 0.0.0.0 0.0.0.0 S0/1/0
exit
write memory

```

In R1

show ip route

In PC0

command prompt ⇒ ~~ping 192.168.10.1~~

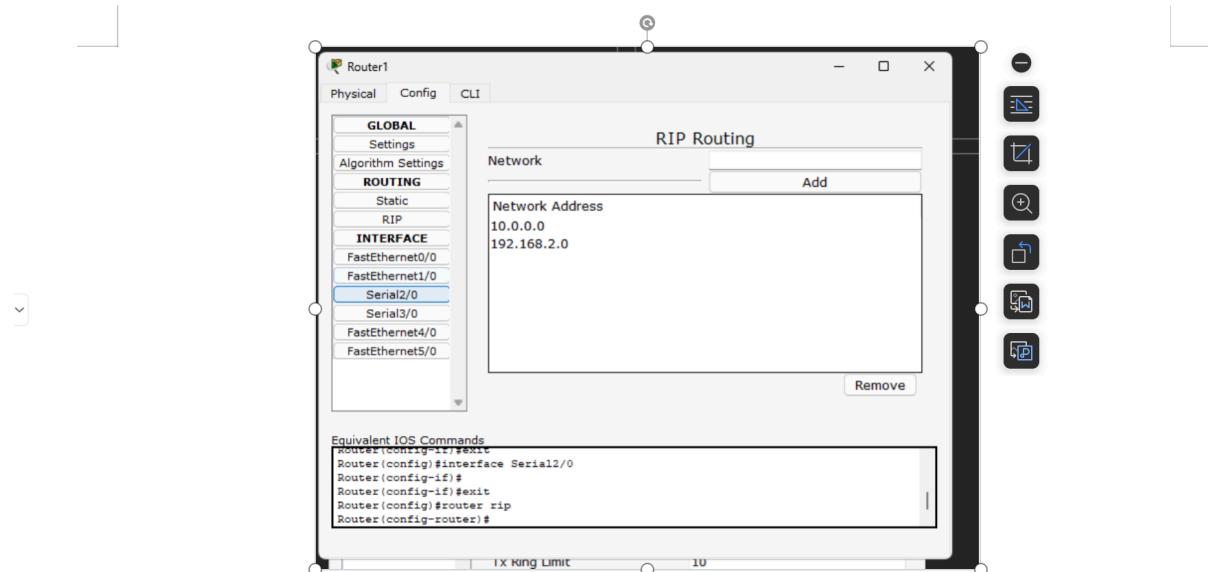
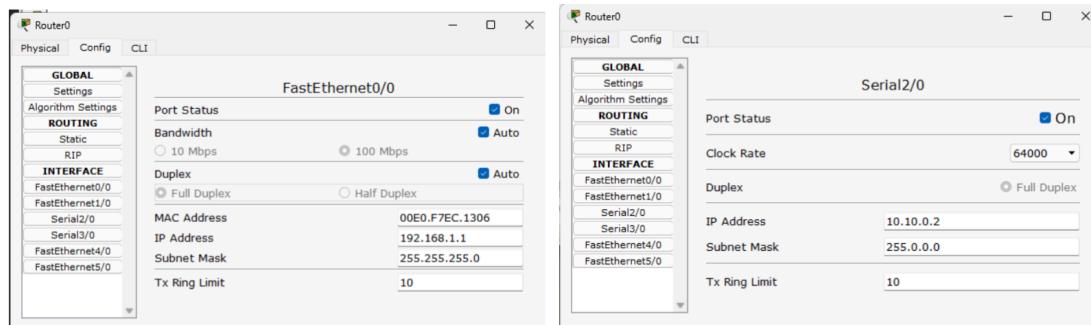
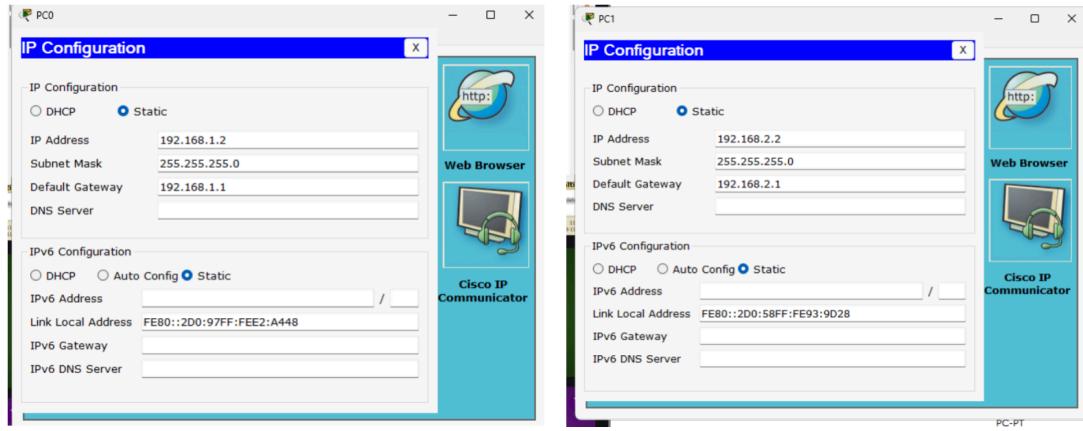
In PC1

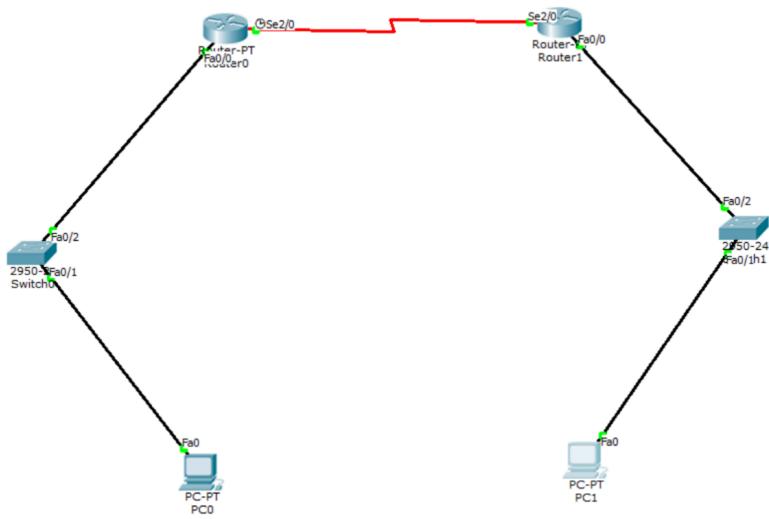
CMD ⇒ ping 192.168.20.10

Message from PC0 → PC1 ⇒ successful

Program 6: Configure RIP routing Protocol in Routers.

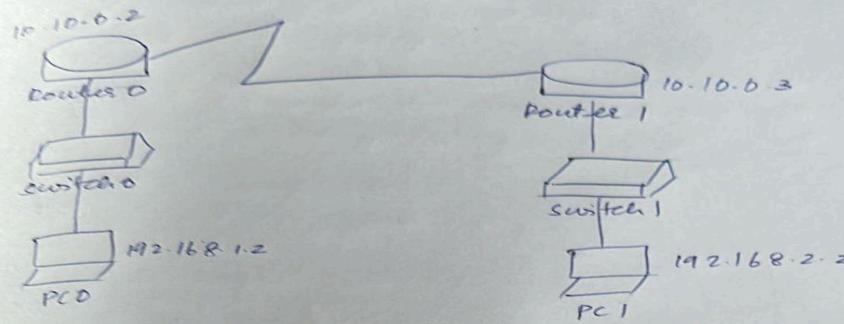
Network diagram:





Configuration:

Configure RIP routing protocol in Router 0 to transfer packet from Node A to Node B.



Gateway : 192.168.1.1

Gateway : 192.168.2.1

Configure 2 PC with IP address.

Router 0 - config - Fast Ethernet Fa0/0
switch port status ON.
IP : 192.168.1.1

serial 0/2 : port status : ON
clock rate : 64000
IP : 10.10.0.2

same in router 1 :

Fa0/0 : 192.168.2.1
serial 0/2 : 10.10.0.3

config Router

Router 0 - config - RIP

In network : 192.168.1.0 → add
10.10.0.0 → add.

settings ↳ NVRAM - same.

Router 1

(as)
CLI network : 192.168.2.0
netmask : 10.0.0.0

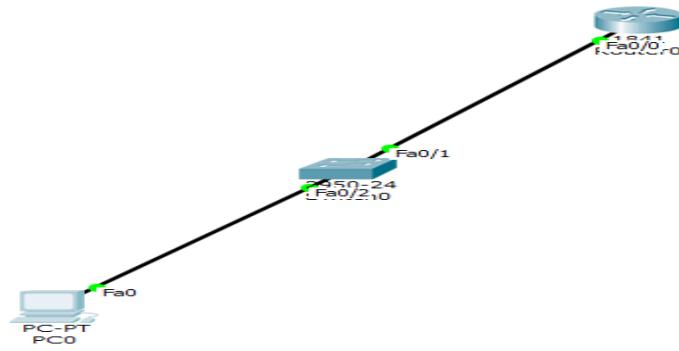
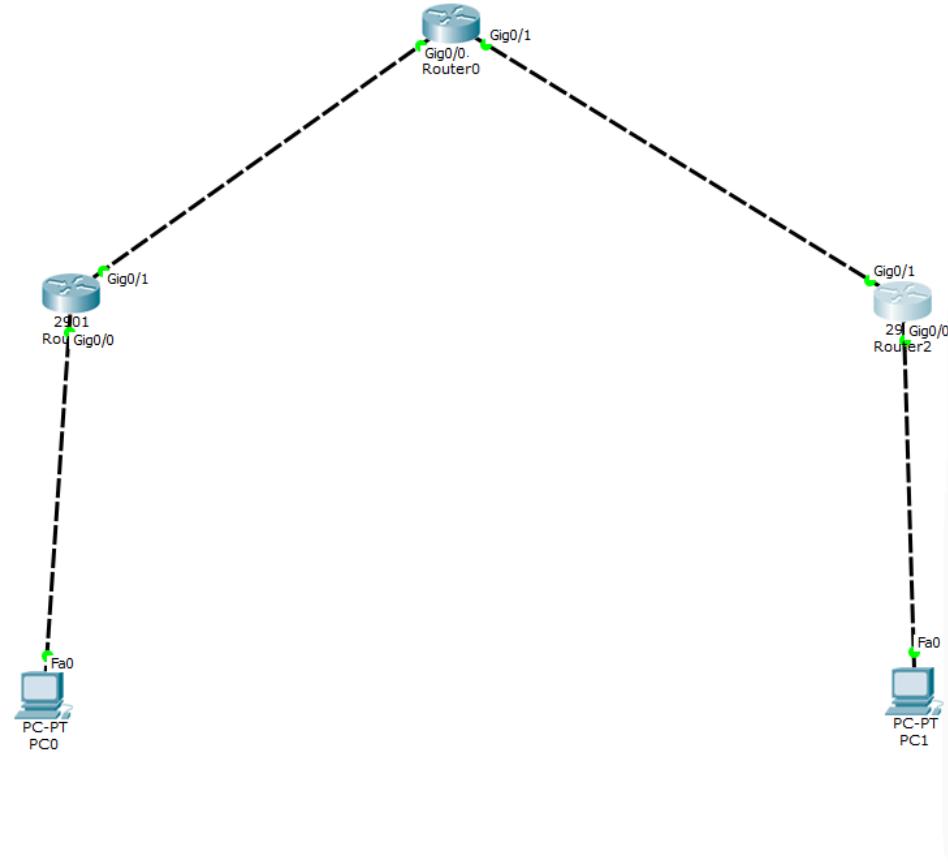
exit

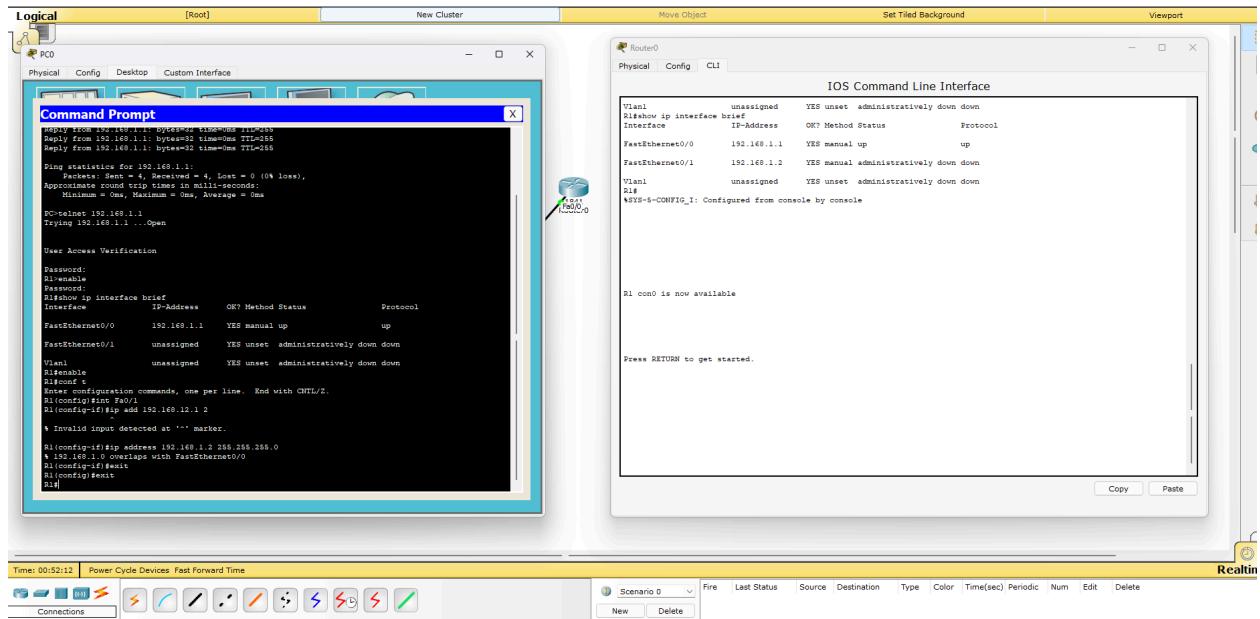
exit

setting NVRAM - same.

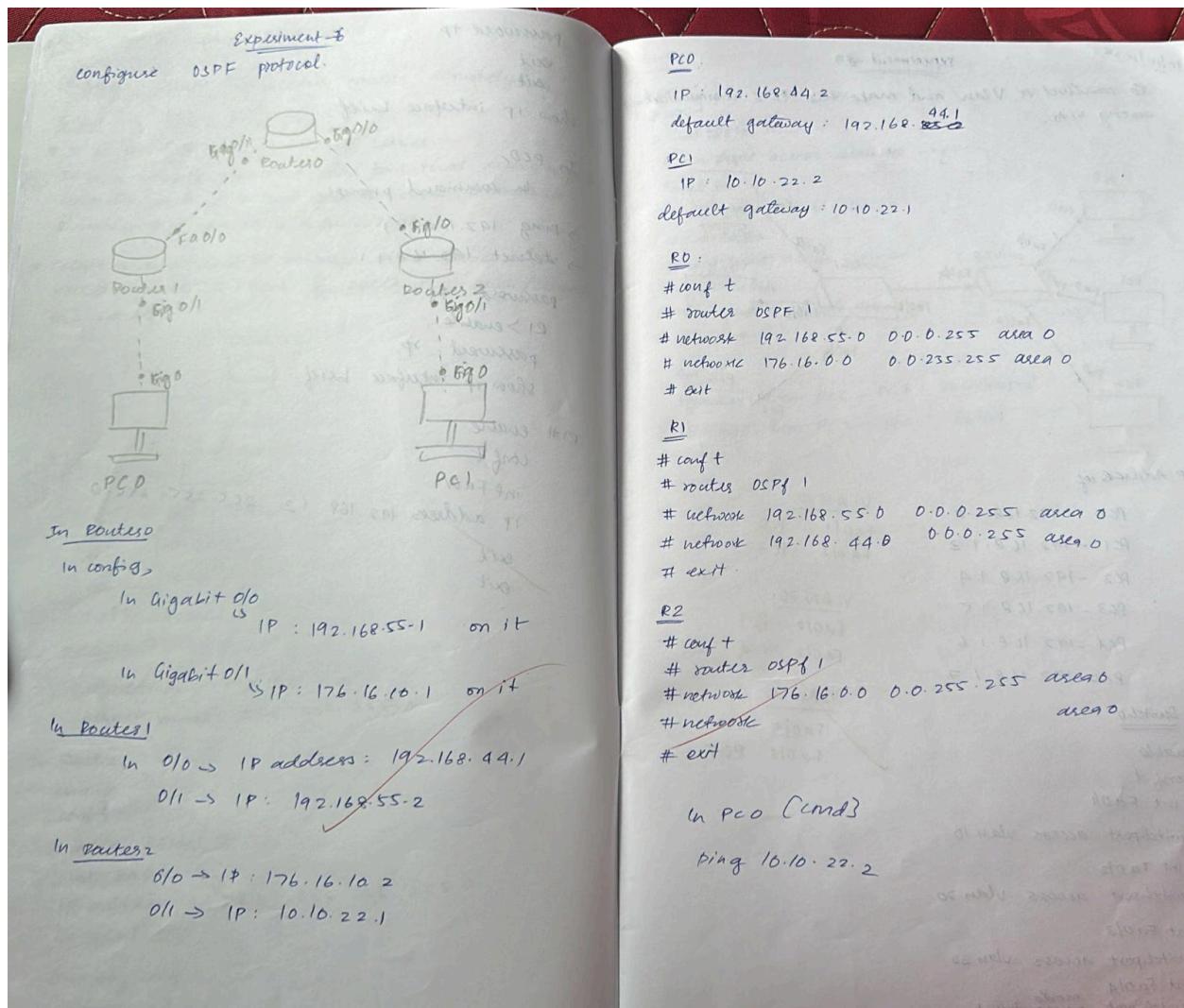
Program 7: Configure OSPF routing protocol.

Network diagram:



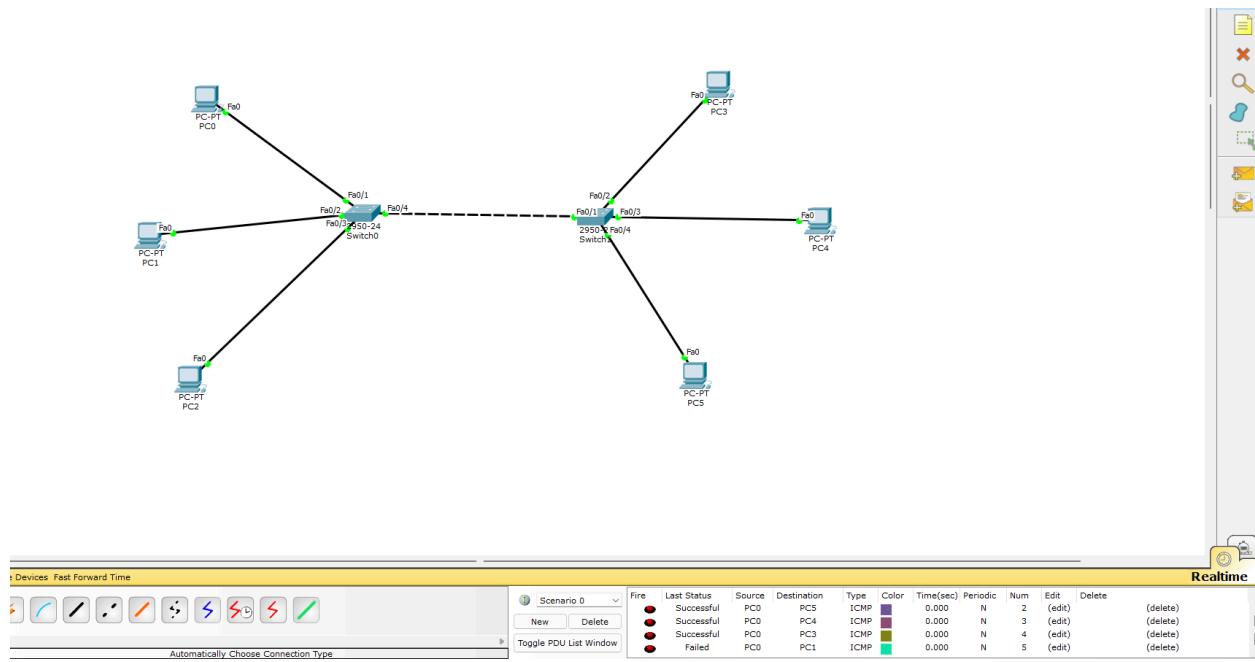


Configuration:



Program 8: To construct a VLAN and make the PC's communicate among a VLAN.

Network diagram:

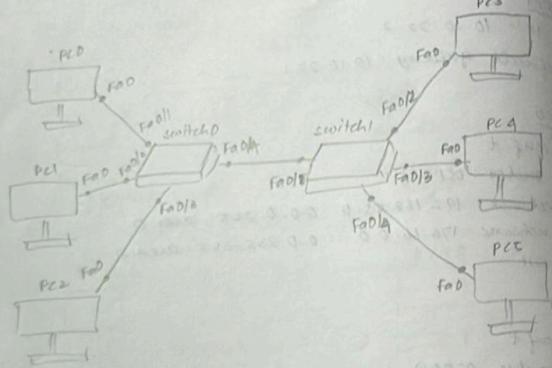


Configuration:

17/10/2025

Experiment - 87

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



IP Address of :

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

VLAN 10

Fa0/1 PC0
Fa0/2 PC3

VLAN 20

Fa0/2 PC1
Fa0/3 PC4

VLAN 30

Fa0/3 PC2
Fa0/4 PC5

switch1

>enable

config t

int Fa0/1

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

Output

Message from PC0 - PC3 successful

Message from PC2 - PC4 failed

Switch2

>enable

config t

int Fa0/1

switchport access vlan 10

int Fa0/2

switchport access vlan 20

int Fa0/3

switchport access vlan 30

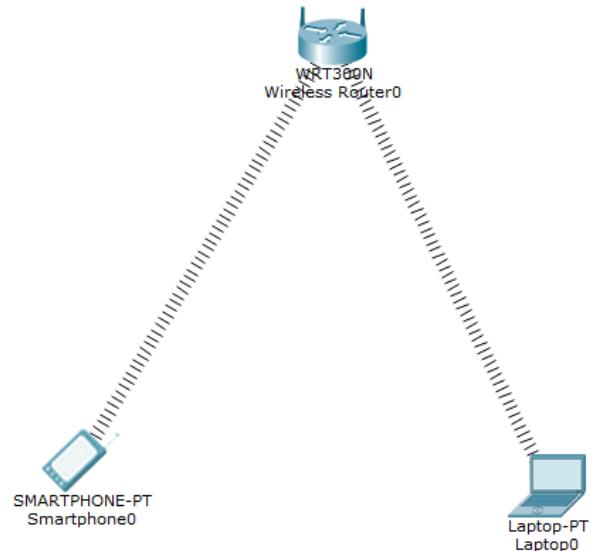
int Fa0/4

switchport mode trunk

exit

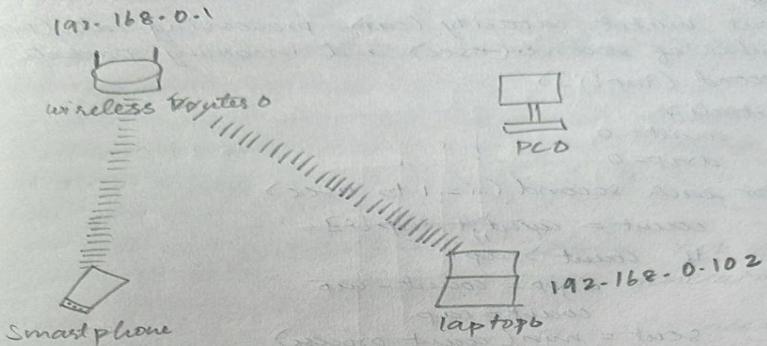
Program 9: To construct a WLAN and make the nodes communicate wirelessly.

Network diagram:



Configuration:

^{Experiment - 9}
to construct a WLAN and make the nodes communicate wirelessly.



for laptop,
turn it off, take out the port & drop it.
again drag + drop wireless port.

wireless Router

↳ Config → wireless, wireless taken

SSID - BMSCE

authentication - WPA2 - PSK PSK

password: bmsce1234

in smartphone

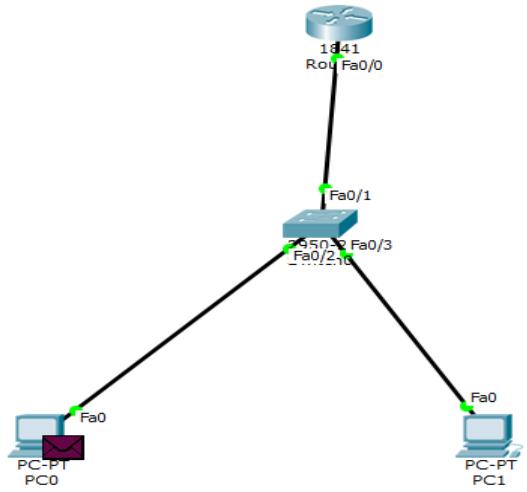
↳ config - wireless

↳ SSID - BMSCE

↳ authentication : bmsce1234

Program 10: Demonstrate the TTL/ Life of a Packet.

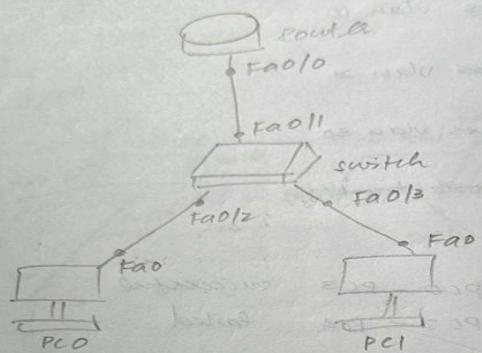
Network diagram:



Configuration:

Experiment-8

Demonstrate the TTL/life of a packet.



IP address of

PC0 - 192.168.1.2

PC1 - 192.168.1.3

Router - 192.168.1.1

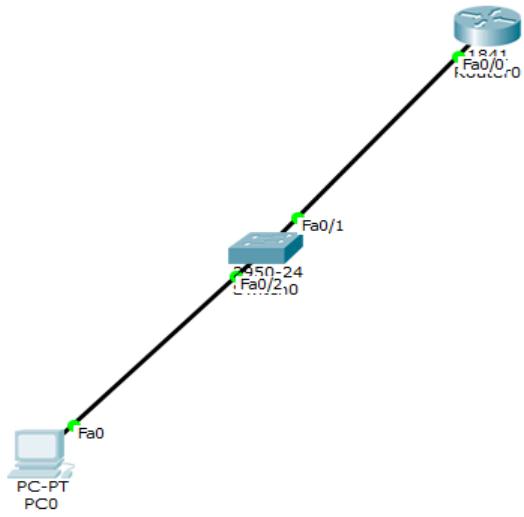
Default gateway
for PC0 & PC1 is 192.168.1.1

Message from PC0 - PC1 - successful

TTL 255, 128

Program 11: To understand the operation of TELNET by accessing the router in the server room from a PC in the IT office.

Network diagram:



Configuration:

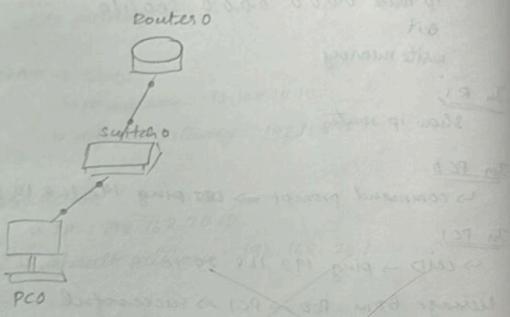
8/10/2025

Experiment-5

Configure Telnet to access router remotely.

Telnet:

- * It is used to access remote servers.
- It is a simple command-line tool that runs on your computer and it allows you to send commands remotely to a server or administrator.
- * Telnet is also used to manage other devices like routers, switches to check if ports are open/closed or the status.



In PCO,

IP address : 192.168.1.2
Subnet mask : 255.255.255.0

In Router,

```
enable
conf t
enable secret rp
int Fa0/0
ip address 192.168.1.1 255.255.255.0
no shutdown
line vty 0 5
login
```

password tp

exit

exit

show ip interface brief.

In PCO,

In command prompt,

→ Ping 192.168.1.1

→ telnet 192.168.1.1

password : tp

R1>enable

password : tp

show ip interface brief.

R1# enable

conf t

int Fa0/1

ip address 192.168.1.2 255.255.255.0

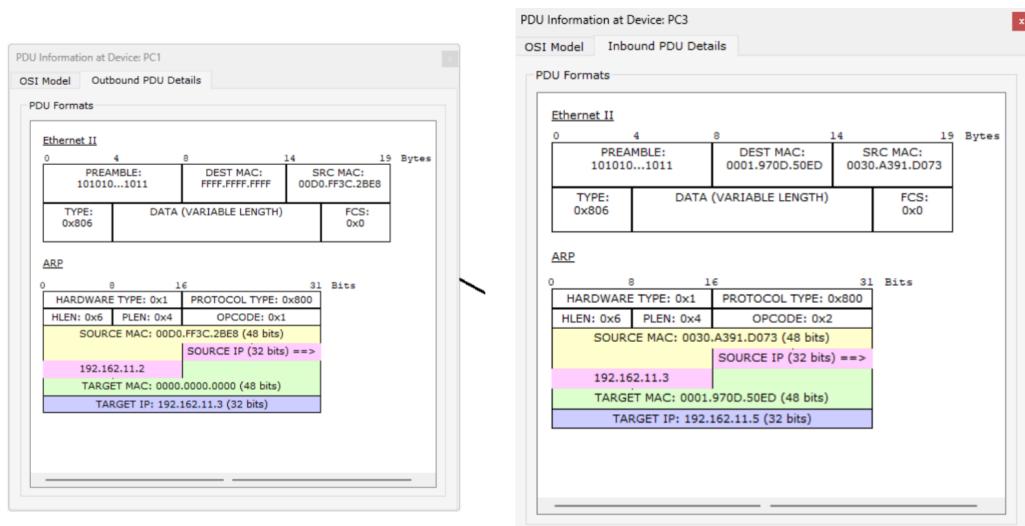
exit

exit

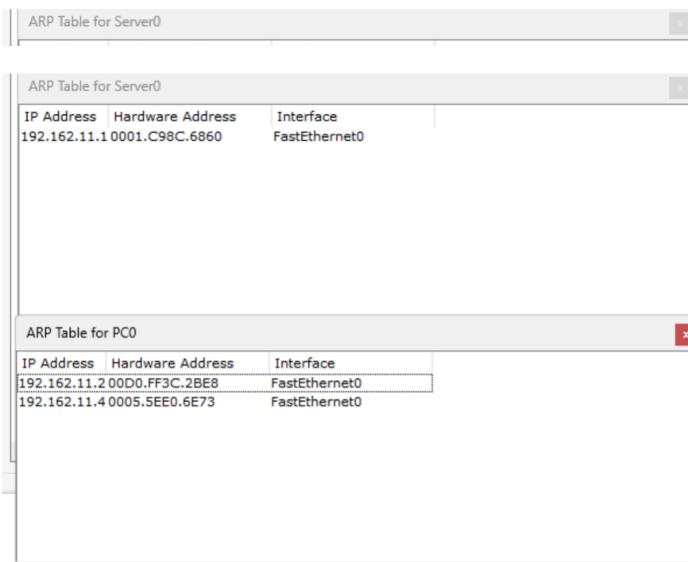
Program 12: To construct simple LAN and understand the concept and operation of Address Resolution Protocol (ARP).

Network diagram:

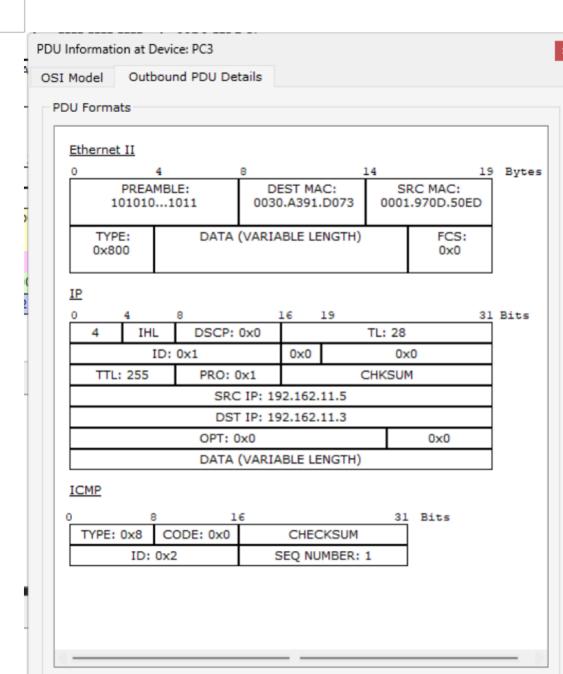
ARP



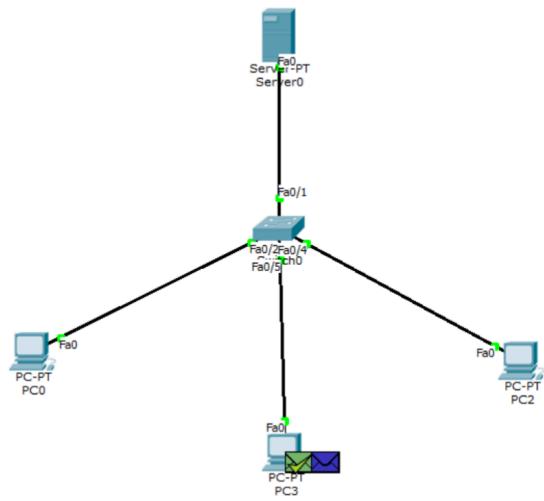
THE MAC ADDRESS before and after encrypting and sending



ARP table of pc0 and server after sending a simple PDU



The address of sent one



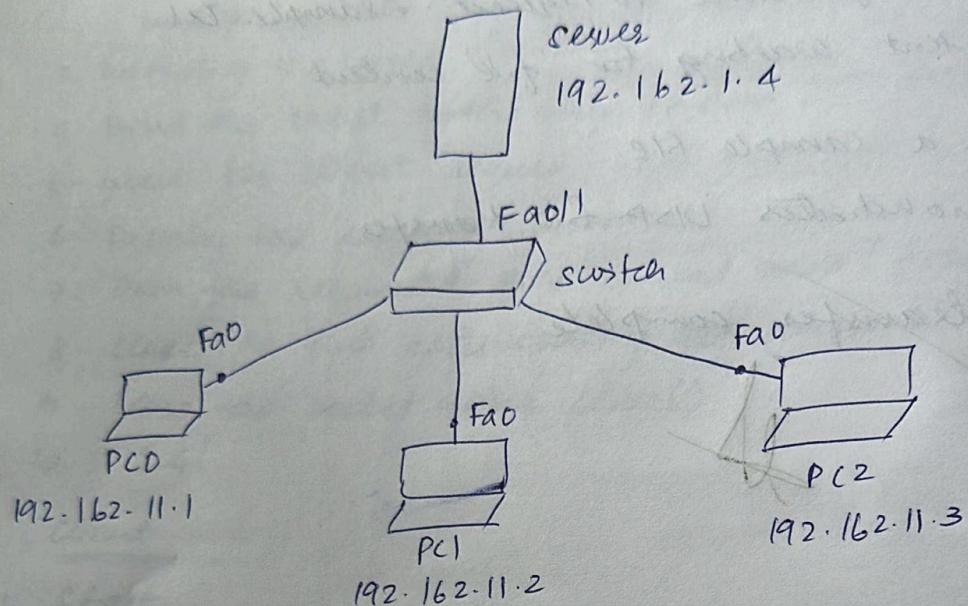
topology

Configuration:

19/11/2025

ARP

- ARP stands for Address Resolution Protocol.
- ARP is used to map an IP address to MAC address.
- ARP is used to get the data link layer address, MAC address with the help of IP address.



Server 192.168.11.4 → DNS Server

Gateway - when using router.

PART - B

Program 1: Write a program for congestion control using Leaky bucket algorithm.

Code:

```
#include <stdio.h>
int min(int x, int y)
{
    return (x < y) ? x : y;
}
int main()
```

```

int drop = 0, mini, nsec, cap, count = 0, i, inp[25], process;
printf("Enter the bucket size:\n");
scanf("%d", &cap);
printf("Enter the processing rate:\n");
scanf("%d", &process);
printf("Enter the number of seconds you want to simulate:\n");
scanf("%d", &nsec);
for (i = 0; i < nsec; i++)
{
    printf("Enter the size of the packet entering at %d sec:\n", i + 1);
    scanf("%d", &inp[i]);
}
printf("\n Second | Packet received | Packet sent | Packet left | Dropped\n");
printf("-----\n");
for (i = 0; i < nsec; i++)
{
    printf("Enter the size of the packet entering at %d sec:\n", i + 1);
    scanf("%d", &inp[i]);
}

printf("\n Second | Packet received | Packet sent | Packet left | Dropped\n");
printf("-----\n");

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

    if (count > cap)
    {
        drop = count - cap;
        count = cap;
    }

    printf("%6d | %15d |", i + 1, inp[i]);

    mini = min(count, process);
    printf(" %11d |", mini);

    count -= mini;
    printf(" %12d | %7d\n", count, drop);

    drop = 0;
}
// Process remaining packets after all seconds
for (; count != 0; i++)
{
    if (count > cap)
    {

```

```
drop = count - cap;
count = cap;
}

printf("%6d | %15d |", i + 1, 0);

mini = min(count, process);
printf(" %11d |", mini);

count -= mini;
printf(" %12d | %7d\n", count, drop);
}

return 0;
}
```

Output:

29/10/2025

Part-B

I. Leaky Bucket Algorithm

1. Start
2. Input bucket capacity (cap), processing rate (process), number of seconds (nsec) and incoming packets per second (inp[i])
3. Initialize
count = 0
drop = 0
4. For each second ($i = 1$ to nsec)
 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 packets.
 reset drop = 0
5. After all seconds:
 while count > 0:
 sent = min(count, process)
 count = count - sent
 Print remaining packets.
6. End.

Output:

Enter the bucket size: 5

Enter the processing rate: 2

Enter the no. of seconds you want to simulate: 3

Enter the size of the packet entering at 1 sec: 5

Enter the size of the packet entering at 2 sec: 4

Enter the size of the packet entering at 3 sec: 3

Second	Packet received	Packet sent	Packet left	Dropped
1	5			
2	4	2	3	0
3	3	2	3	2
4	0	2	3	1
5	0	1	1	0

Program 2: Using TCP/IP sockets, write a client-server program to make the client send the file name and the server to send back the contents of the requested file if present.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv;
    struct hostent *server;
    char buffer[256];
    char c[20000];

    if (argc < 3) {
        printf("Error: insufficient arguments.\n");
        printf("Usage: %s <hostname> <port>\nExample: %s 127.0.0.1 7777\n",
               argv[0], argv[0]);
        exit(1);
    }

    portno = atoi(argv[2]);

    // Create socket
```

```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
if (sockfd < 0) {
    perror("Error opening socket");
    exit(1);
}

// Get server by name/IP
server = gethostbyname(argv[1]);
if (server == NULL) {
    fprintf(stderr, "Error: no such host.\n");
    exit(1);
}

// Zero out the structure
bzero((char *)&serv, sizeof(serv));
serv.sin_family = AF_INET;
bcopy((char *)server->h_addr, (char *)&serv.sin_addr.s_addr, server->h_length);
serv.sin_port = htons(portno);

// Connect to server
if (connect(sockfd, (struct sockaddr *)&serv, sizeof(serv)) < 0) {
    perror("Error connecting");
    exit(1);
}

printf("Enter the file path (complete path): ");
scanf("%s", buffer);
```

```
// Send filename to server
n = write(sockfd, buffer, strlen(buffer));
if (n < 0) {
    perror("Error writing to socket");
    exit(1);
}

bzero(c, sizeof(c));
printf("Reading file contents from server..\n");

// Read file contents
n = read(sockfd, c, sizeof(c) - 1);
if (n < 0) {
    perror("Error reading from socket");
    exit(1);
}

printf("\nClient: Display content of %s\n-----\n",
buffer);
fputs(c, stdout);
printf("\n-----\n");

close(sockfd);
return 0;
}

#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
    int sockfd, newsockfd, portno, len, n;
    char buffer[256], c[2000], cc[20000];
    struct sockaddr_in serv, cli;
    FILE *fd;

    if (argc < 2) {
        printf("Error: no port number provided.\n");
        printf("Usage: %s <port>\nExample: %s 7777\n", argv[0], argv[0]);
        exit(1);
    }

    // Create socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Error opening socket");
        exit(1);
    }
```

```
// Initialize server address structure
bzero((char *)&serv, sizeof(serv));
portno = atoi(argv[1]);
serv.sin_family = AF_INET;
serv.sin_addr.s_addr = INADDR_ANY;
serv.sin_port = htons(portno);

// Bind socket
if (bind(sockfd, (struct sockaddr *)&serv, sizeof(serv)) < 0) {
    perror("Error on binding");
    close(sockfd);
    exit(1);
}

// Listen for incoming connections
listen(sockfd, 5);
len = sizeof(cli);
printf("Server: waiting for connection on port %d...\n", portno);

// Accept a client
newsockfd = accept(sockfd, (struct sockaddr *)&cli, (socklen_t *)&len);
if (newsockfd < 0) {
    perror("Error on accept");
    close(sockfd);
    exit(1);
}
```

```
// Read filename from client
bzero(buffer, 255);
n = read(newsockfd, buffer, 255);
if (n < 0) {
    perror("Error reading from socket");
    close(newsockfd);
    close(sockfd);
    exit(1);
}

printf("Server received filename: %s\n", buffer);

// Try to open the file
fd = fopen(buffer, "r");
if (fd != NULL) {
    printf("Server: file '%s' found, reading and sending...\n", buffer);
    bzzero(cc, sizeof(cc));
    while (fgets(c, sizeof(c), fd) != NULL) {
        strcat(cc, c);
    }
    fclose(fd);
}

// Send file content to client
n = write(newsockfd, cc, strlen(cc));
if (n < 0)
    perror("Error writing to socket");
else
```

```
    printf("File transfer complete.\n");
} else {
    printf("Server: file not found.\n");
    n = write(newsockfd, "Error: file not found.\n", 24);
    if (n < 0)
        perror("Error writing to socket");
}

close(newsockfd);
close(sockfd);
return 0;
}
```

Output:

12/11/2025

III. Client Server communication using TCP/IP sockets.

Algorithm (Client side):

1. sockfd = create a socket with the socket system call.
2. Connect the socket to the address of the server using the connect(sockfd,...) system call. The IP address of the server machine and port number of the server service need to be provided.
3. Read file name from standard input by n = read(stdin, buffer, sizeof(buffer))
4. Write file name to the socket using write.
5. Read file contents from the socket by m = read(sockfd, buffer1, sizeof(buffer1))
6. Display file contents to standard output by write (stdout, buffer1, m)
7. Go to step 5 if m > 0
8. Close socket by close(sockfd)

Algorithm (server side):

1. sockfd = Create a socket with the socket system call.
2. Bind the socket to an address using the bind(sockfd,...) system call. If not sure of machine IP address, keep the structure member sockaddr_SA set to INADDR_ANY. Assign a port number between 3000-5000 to htons(port).
3. Listen for connection with the listen(sockfd,...) system call. This call typically blocks until a client connects with the server.
4. sockfd = Accept a connection with the accept system call. This call typically blocks until a client connects with the server.
5. Read the filename from the socket by n = read(sockfd, buffer, sizeof(buffer))
6. Open the file by fd = open(buffer)
7. Read the contents of the file by m = read(fd, buffer1, sizeof(buffer1))

8. Write the file content by socket to write(sockfd, buffer1, m)

9. Go to step 7 if m > 0

10. Close (sockfd)

Output:

\$ cc socketserver.c

./a.out 1025

server:

waiting for connection

server received : /home/aps/cse.txt

server : /home/aps/cse.txt found

opening and reading ...

reading ...

... reading complete

transfer complete

\$ cc socketclient.c

./a.out 1025

Enter the file with complete path

/home/aps/cse.txt

Reading ...

client : displays contents of /home/aps/cse.txt

....

Welcome to CSE department

Program 3: Using UDP sockets, write a client-server program to make the client send the file name and the server to send back the contents of the requested file if present.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
```

```
#define PORT 8080
#define MAXLINE 1024

int main() {
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr, cliaddr;
    socklen_t len;
    ssize_t n;

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Server info
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    // Bind socket to the port
    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("bind failed");
```

```

        close(sockfd);
        exit(EXIT_FAILURE);
    }

printf("UDP Server is running on port %d...\n", PORT);

len = sizeof(cliaddr);

// Receive filename from client
n = recvfrom(sockfd, (char *)buffer, MAXLINE, 0, (struct sockaddr *)&cliaddr,
&len);
buffer[n] = '\0';
printf("Client requested file: %s\n", buffer);

FILE *fp = fopen(buffer, "r");
if (fp == NULL) {
    char *msg = "File not found!";
    sendto(sockfd, msg, strlen(msg), 0, (struct sockaddr *)&cliaddr, len);
    printf("File not found, message sent to client.\n");
} else {
    // Read and send file content
    while (fgets(buffer, MAXLINE, fp) != NULL) {
        sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&cliaddr, len);
    }
    fclose(fp);
    printf("File sent successfully.\n");
}

```

```
close(sockfd);
return 0;
}

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8080
#define MAXLINE 1024

int main() {
    int sockfd;
    char buffer[MAXLINE];
    struct sockaddr_in servaddr;
    socklen_t len;

    // Create UDP socket
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
```

```
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);
servaddr.sin_addr.s_addr = INADDR_ANY;

printf("Enter the filename to request: ");
fgets(buffer, MAXLINE, stdin);
buffer[strcspn(buffer, "\n")] = '\0'; // remove newline

// Send filename to server
sendto(sockfd, buffer, strlen(buffer), 0, (const struct sockaddr *)&servaddr,
sizeof(servaddr));

printf("Request sent. Waiting for file content...\n\n");

len = sizeof(servaddr);

// Receive file contents
ssize_t n;
while ((n = recvfrom(sockfd, buffer, MAXLINE, 0, (struct sockaddr *)
*)&servaddr, &len)) > 0) {
    buffer[n] = '\0';
    printf("%s", buffer);
    if (n < MAXLINE - 1) break; // assume end of file
}

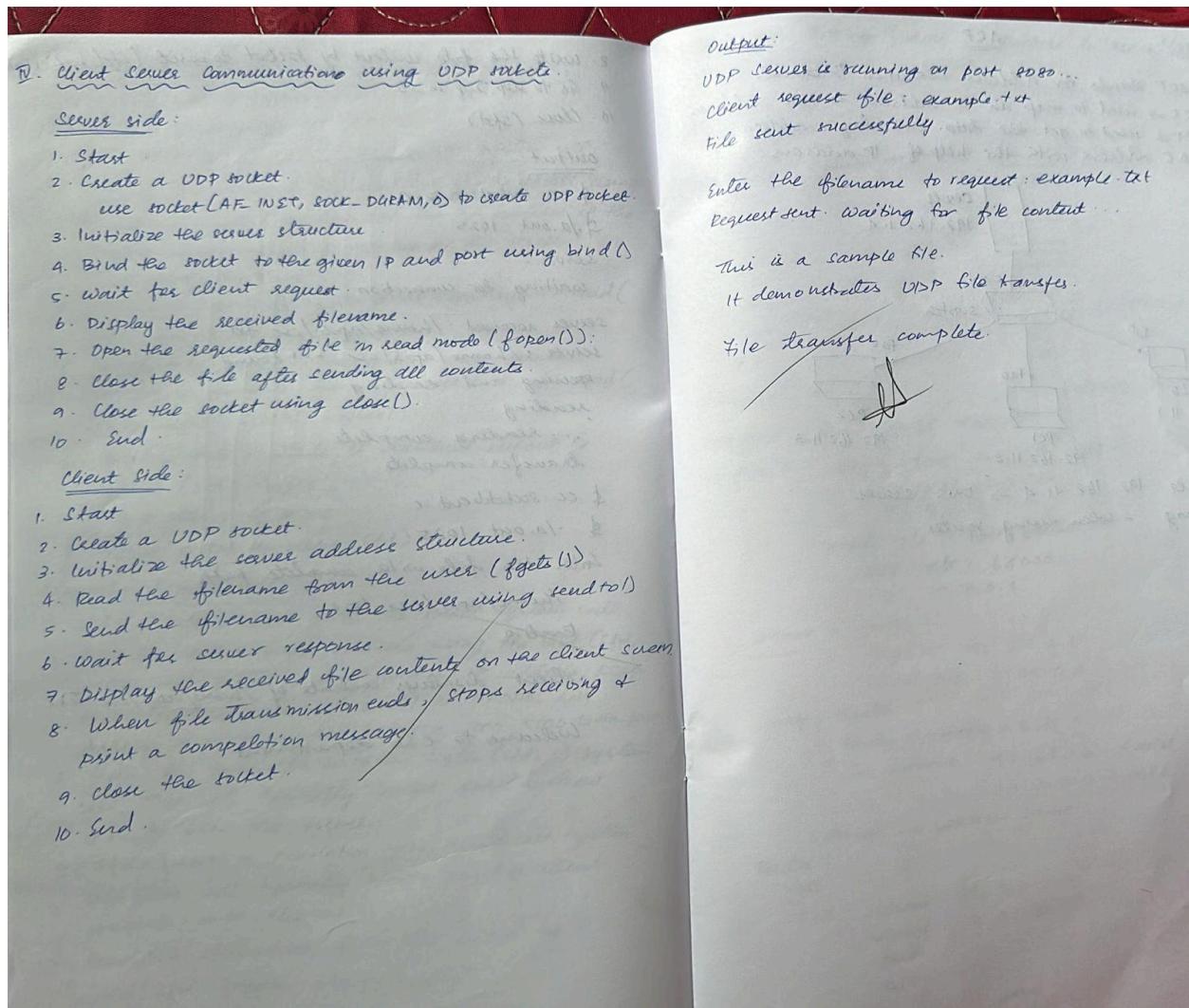
printf("\n\nFile transfer complete.\n");
```

```

        close(sockfd);
        return 0;
    }
}

```

Output:



Program 4: Write a program for error detecting code using CRC-CCITT (16-bits).

Code:

```

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

```

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

    printf("Enter the generator polynomial:\n");
    fgets(gen, sizeof(gen), stdin);
    gen[strcspn(gen, "\n")] = '\0'; // remove newline if present
    printf("Generator polynomial (CRC-CCITT): %s\n", gen);

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

    printf("Enter the message:\n");
    fgets(msj, sizeof(msj), stdin);
    msj[strcspn(msj, "\n")] = '\0'; // remove newline

    n = strlen(msj);

    // Append k zeros to the message
    for (i = 0; i < n; i++)
        a[i] = msj[i];
    for (i = 0; i < k; i++)
        a[n + i] = '0';
    a[n + k] = '\0';
```

```

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

// Division (XOR)
for (i = 0; i < n; i++)
{
    if (a[i] == '1')
    {
        t = i;
        for (j = 0; j <= k; j++)
        {
            if (a[t] == gen[j])
                a[t] = '0';
            else
                a[t] = '1';
            t++;
        }
    }
}

// Get remainder
for (i = 0; i < k; i++)
{
    rem[i] = a[n + i];
    rem[k] = '\0';
}
printf("\nChecksum (Remainder):\n");
puts(rem);

// Append checksum to message
printf("\nTransmitted message (with checksum):\n");

```

```

for (i = 0; i < n; i++)
    a[i] = msj[i];
for (i = 0; i < k; i++)
    a[n + i] = rem[i];
a[n + k] = '\0';
puts(a);
// Receiver side
printf("\nEnter the received message:\n");
fgets(s, sizeof(s), stdin);
s[strcspn(s, "\n")] = '\0'; // remove newline
n = strlen(s);
// Division on received message
for (i = 0; i < n - k; i++)
{
    if (s[i] == '1')
    {
        t = i;
        for (j = 0; j <= k; j++, t++)
        {
            if (s[t] == gen[j])
                s[t] = '0';
            else
                s[t] = '1';
        }
    }
    for (i = 0; i < k; i++)

```

```
rem[i] = s[n - k + i];
rem[k] = '\0';
// Check for error
flag = 0;
for (i = 0; i < k; i++)
{
    if (rem[i] == '1')
        flag = 1;
}
if (flag == 0)
    printf("\nReceived message is error-free ✓\n");
else
    printf("\nReceived message contains errors ✗\n");
return 0;
}
```

Output:

Output

```
b Enter the generator polynomial:  
101  
Generator polynomial (CRC-CCITT): 101  
Enter the message:  
110101  
  
Message polynomial appended with zeros:  
11010100  
  
Checksum (Remainder):  
11  
  
Transmitted message (with checksum):  
11010111  
  
Enter the received message:  
11010111  
  
Received message is error-free 
```

==== Code Execution Successful ===

IV Error detecting code using CRC-CCITT (16-bit)

Algorithm:

1. Start
2. Input data
read the generator polynomial, message bits.
3. Append zeros.
degree of generator = $k = (\text{length of generator} - 1)$
append k zeros to the end of the message.
4. Perform Binary Division (XOR)
for each bit of the message
 - * If the current bit is 1, perform XOR of that part of the message with the generator polynomial.
 - * If the current bit is 0, skip the next bit.The result after the division gives the remainder.
5. form transmitted Message
append the remainder to the original message.
6. Receives side.
Input the received message.
Perform the same XOR using the same generator polynomial
check the remainder obtained.
 - If all bits in the remainders are 0 → No Error.
 - If any bits in the remainders is 1 → Error.
7. End.

Output:

Enter the generator polynomial : 101
Generator polynomial (CRC-CCITT) : 101
Enter the message : 110101
Message polynomial appended with zeros: 11010100
checksum (remainder) : 11
Transmitted message (with checksum): 11010111
Enter the received message : 11010111
Received message is error-free