NETWORK PROGRAMMING LAB MANUAL

Sample Program: To communicate between two systems making one as server and other as client.

SEVER:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/time.h>  // For gettimeofday()

int main() {
   int sfd, cfd; char buf[1024];
   struct sockaddr_in addr; socklen_t len = sizeof(addr);
   struct timeval start, end;

   sfd = socket(AF_INET, SOCK_STREAM, 0);
   addr.sin_family = AF_INET; addr.sin_addr.s_addr = INADDR_ANY; addr.sin_port = htons(8080);
   bind(sfd, (struct sockaddr *)&addr, sizeof(addr));
   listen(sfd, 3); printf("Listening...\n");

   cfd = accept(sfd, (struct sockaddr *)&addr, &len);

   gettimeofday(&start, NULL); // Start time before processing
   int n = read(cfd, buf, 1024); buf[n] = '\0';
   printf("Client: %s\n", buf);
   send(cfd, buf, n, 0);
   gettimeofday(&end, NULL); // End time after sending

   // Calculate processing time in microseconds
   long seconds = end.tv_sec - start.tv_sec;
   long microseconds = end.tv_usec - start.tv_usec;
```

```
    double elapsed = seconds + microseconds*1e-6;

    printf("Processing time: %.6f seconds\n", elapsed);

    close(cfd); close(sfd);
    return 0;
}
```

CLIENT:

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

int main() {
    int sock; char buf[1024];
    struct sockaddr_in serv;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv.sin_family = AF_INET; serv.sin_port = htons(8080);
    inet_pton(AF_INET, "127.0.0.1", &serv.sin_addr);
    connect(sock, (struct sockaddr *)&serv, sizeof(serv));

    printf("Enter message: ");
    fgets(buf, sizeof(buf), stdin);
    send(sock, buf, strlen(buf), 0);
    int n = read(sock, buf, sizeof(buf)); buf[n] = '\0';
    printf("Server: %s\n", buf);
    close(sock);
    return 0;
}
```

JAVA PROGRAM

SERVER:

```
import java.io.*;
import java.net.*;


public class Server {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(8080)) {
            System.out.println("Listening on port 8080...");


            Socket clientSocket = serverSocket.accept();
            long startTime = System.nanoTime(); // Start timing


            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);


            String clientMessage = in.readLine();
            System.out.println("Client: " + clientMessage);
            out.println(clientMessage);


            long endTime = System.nanoTime(); // End timing
            double elapsedTime = (endTime - startTime) / 1e6; // Convert to milliseconds
            System.out.printf("Processing time: %.3f ms\n", elapsedTime);


            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

CLIENT:

```
import java.io.*;
import java.net.*;


public class Client {
   public static void main(String[] args) {
      try (Socket socket = new Socket("127.0.0.1", 8080)) {
         BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);


         System.out.print("Enter message: ");
         String message = userInput.readLine();
         out.println(message);


         String serverMessage = in.readLine();
         System.out.println("Server: " + serverMessage);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

COMMAND FOR JAVA:

Install java in ubuntu using:

```
sudo apt update
sudo apt install default-jdk
```

To verify installation:

java -version

javac -version

Create java file:

gedit filname.java

compile:

javac filename.java

run:

java filename

Filename and class name should be same

To communicate between two systems making one as server and other as client:

Step1: Before starting ubuntu , go to settings in virtual manager.

Step2: Click on network tab.

Step3: In Attached To select Bridged Adapter from drop down, then click OK .

Step4: Start the ubuntu

[Do this changes in both the client and server systems]

Step5: In terminal type command hostname -I [to get IP address]

Step6: In the client system code change (127.0.0.1) to (server IP address) in line

[inet_pton(AF_INET, "127.0.0.1", &serv.sin_addr);]

Step7: then run server and then client as usual.

1.Write a program to implement daytime client/server program using TCP socket.

Server Side:

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

int main() {
    int sockfd, conntfd;
    struct sockaddr_in sa, cli;
    socklen_t len;
    char str[100];
    time_t tick;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) { perror("Socket failed"); exit(1); }
    printf("Socket opened\n");

    bzero(&sa, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(5600);
    sa.sin_addr.s_addr = htonl(INADDR_ANY);

    if (bind(sockfd, (struct sockaddr*)&sa, sizeof(sa)) < 0) {
        perror("Bind failed"); exit(1);
```

```
    }
    printf("Binded\n");

    listen(sockfd, 5);
    while (1) {
        len = sizeof(cli);
        conntfd = accept(sockfd, (struct sockaddr*)&cli, &len);
        if (conntfd < 0) { perror("Accept failed"); continue; }
        printf("Accepted\n");
        tick = time(NULL);
        snprintf(str, sizeof(str), "%s", ctime(&tick));
        write(conntfd, str, strlen(str));
        close(conntfd);
    }
    close(sockfd);
    return 0;
}
```

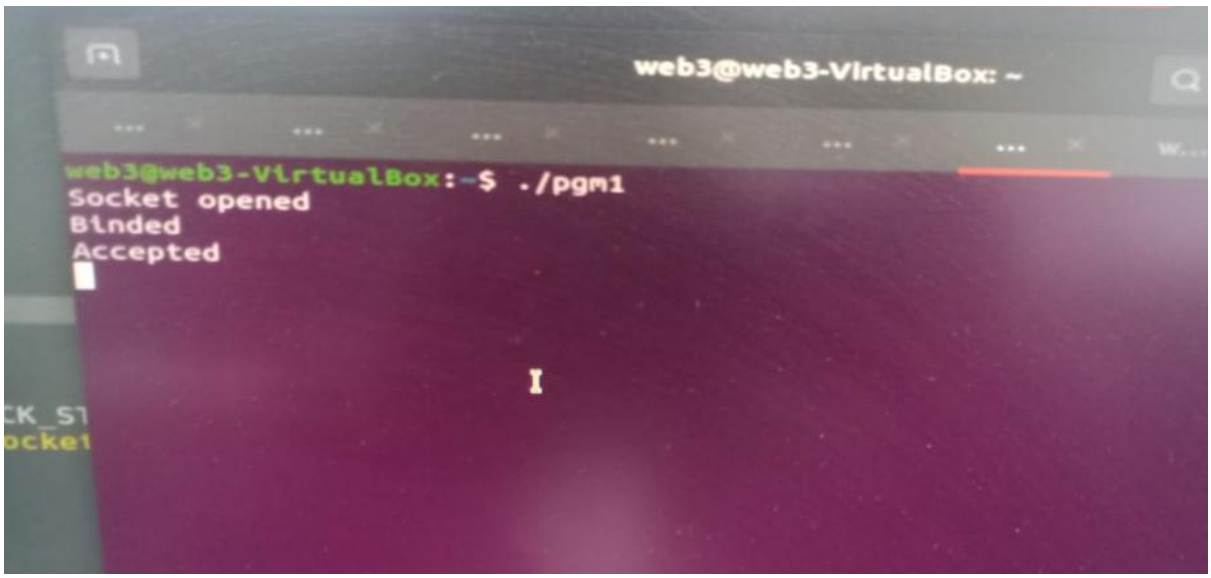
Client Side

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

int main() {
    int sockfd, n;
    struct sockaddr_in sa;
    char buff[100];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) { perror("Socket failed"); exit(1); }
    printf("Socket opened\n");
```

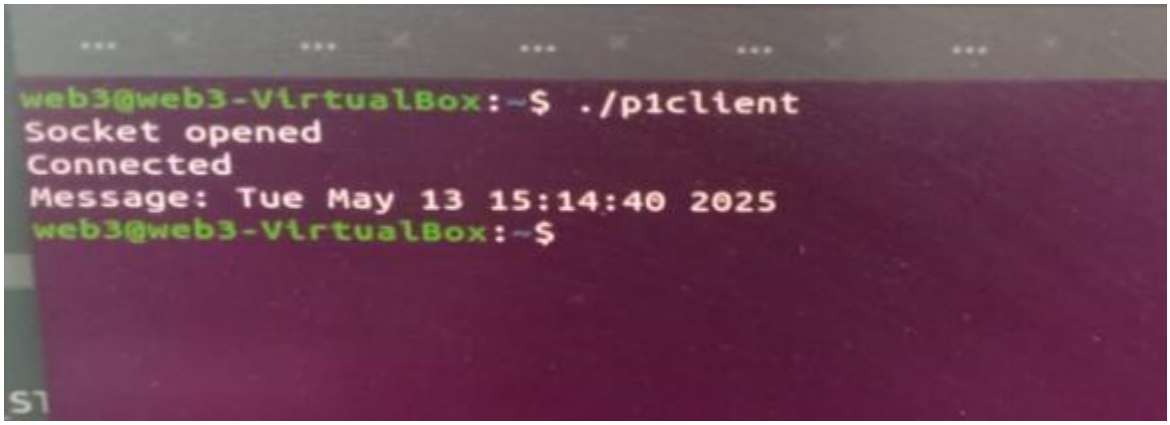
```
    bzero(&sa, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(5600);
    sa.sin_addr.s_addr = htonl(INADDR_LOOPBACK);

    if (connect(sockfd, (struct sockaddr*)&sa, sizeof(sa)) < 0) {
        perror("Connect failed"); exit(1);
    }
    printf("Connected\n");

    n = read(sockfd, buff, sizeof(buff) - 1);
    if (n < 0) { perror("Read error"); exit(1); }

    buff[n] = '\0';
    printf("Message: %s", buff);

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

OUTPUT:

Server side



Client Side


web3@web3-VirtualBox:~$ ./p1client
Socket opened
Connected
Message: Tue May 13 15:14:40 2025
web3@web3-VirtualBox:~$

Program 2: Write a TCP client/server program in which client sends three numbers to the server in a single message. Server returns sum, difference and product as a result single message. Client program should print the results appropriately.

Server Side:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>

#define PORT 8080

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    socklen_t addrlen = sizeof(address);

    int nums[3];      // to receive three numbers
    int results[3];   // sum, difference, product

    // Create socket
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Define address
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);
```

```
// Bind
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// Listen
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

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

// Accept
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, &addrlen)) < 0) {
    perror("Accept failed");
    exit(EXIT_FAILURE);
}

// Read 3 integers from client
int bytes_read = read(new_socket, nums, sizeof(nums));
if (bytes_read != sizeof(nums)) {
    perror("Failed to read all 3 integers");
    exit(EXIT_FAILURE);
}

// Calculate
results[0] = nums[0] + nums[1] + nums[2];
```

```
    results[1] = nums[0] - nums[1] - nums[2];
    results[2] = nums[0] * nums[1] * nums[2];

    // Send result back
    write(new_socket, results, sizeof(results));

    close(new_socket);
    close(server_fd);

    return 0;
}
```

Client Side:

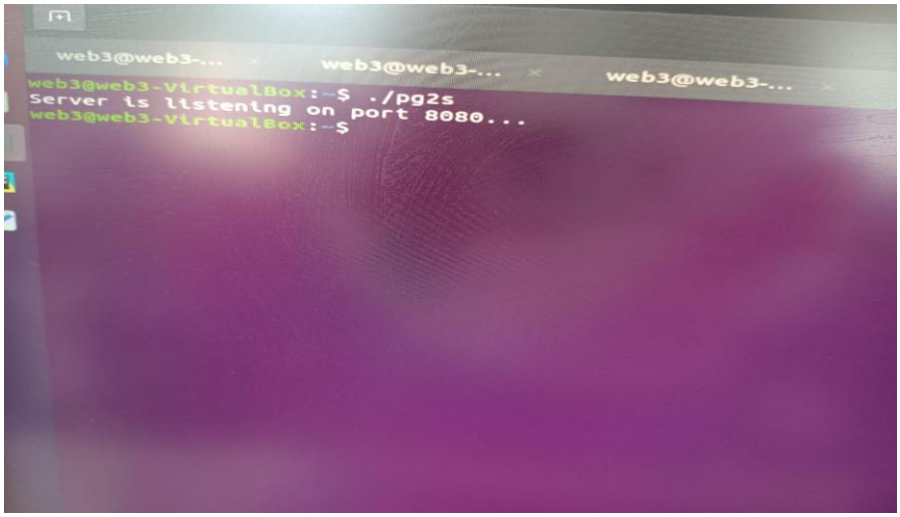
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#define PORT 8080

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    int nums[3];
    int results[3];

    printf("Enter three integers: ");
    scanf("%d %d %d", &nums[0], &nums[1], &nums[2]);
```

```
// Create socket
if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("Socket creation error");
    return -1;
}

serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(PORT);

// Convert IPv4 address from text to binary
if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
    perror("Invalid address / Address not supported");
    return -1;
}

// Connect to server
if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    perror("Connection Failed");
    return -1;
}

// Send 3 integers
send(sock, nums, sizeof(nums), 0);

// Receive result
read(sock, results, sizeof(results));

// Display result
printf("Sum: %d\n", results[0]);
printf("Difference: %d\n", results[1]);
```

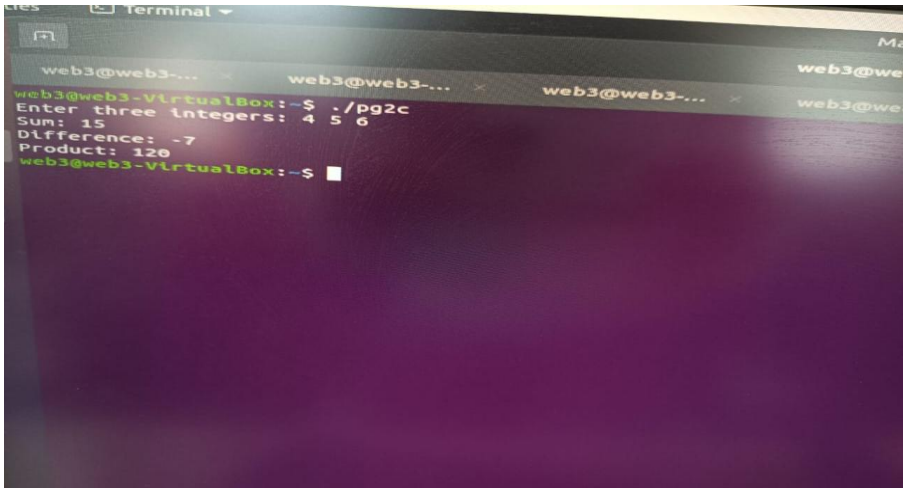
```
    printf("Product: %d\n", results[2]);

    close(sock);

    return 0;
}
```

OUTPUT:

Server Side



Client side



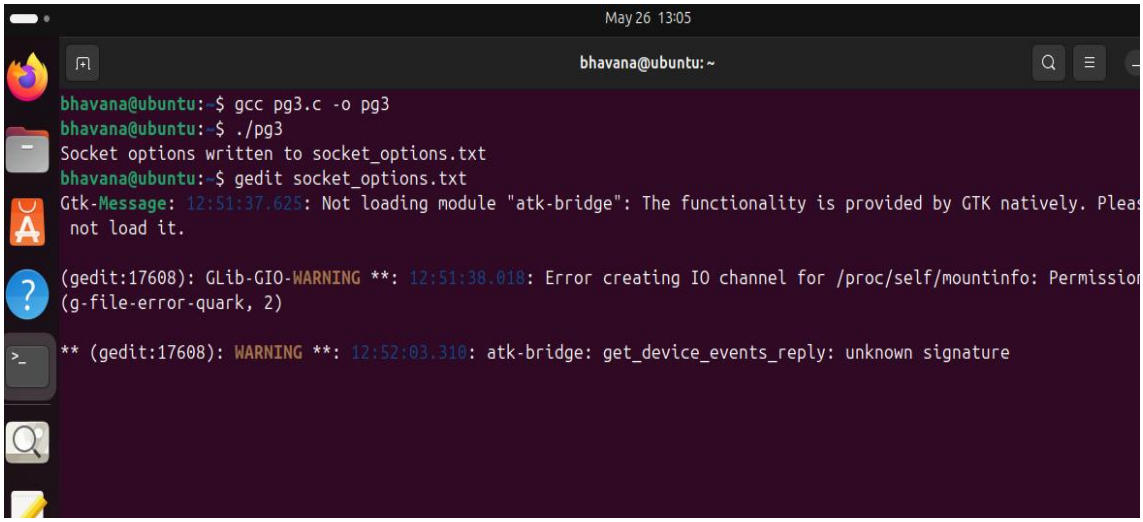
Program 3: Write a C program that prints the IP layer and TCP layer socket options in a separate file.

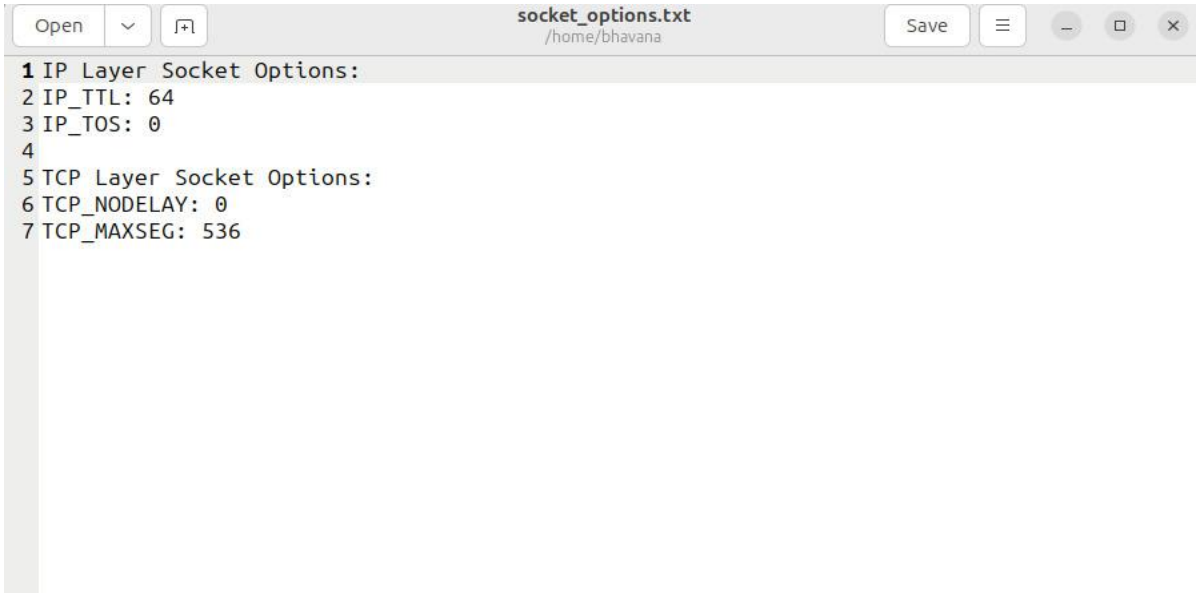
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <sys/socket.h>

int main() {
    int sockfd;
    FILE *file;
    int optval;
    socklen_t optlen = sizeof(optval);

    // Create a TCP socket
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    // Open file to write options
    file = fopen("socket_options.txt", "w");
    if (file == NULL) {
        perror("File open failed");
        close(sockfd);
```

```
    exit(EXIT_FAILURE);
}

// --- IP Layer Options ---
fprintf(file, "IP Layer Socket Options:\n");

if (getsockopt(sockfd, IPPROTO_IP, IP_TTL, &optval, &optlen) == 0)
    fprintf(file, "IP_TTL: %d\n", optval);
else
    perror("getsockopt IP_TTL");

if (getsockopt(sockfd, IPPROTO_IP, IP_TOS, &optval, &optlen) == 0)
    fprintf(file, "IP_TOS: %d\n", optval);
else
    perror("getsockopt IP_TOS");

// --- TCP Layer Options ---
fprintf(file, "\nTCP Layer Socket Options:\n");

if (getsockopt(sockfd, IPPROTO_TCP, TCP_NODELAY, &optval, &optlen) == 0)
    fprintf(file, "TCP_NODELAY: %d\n", optval);
else
    perror("getsockopt TCP_NODELAY");

if (getsockopt(sockfd, IPPROTO_TCP, TCP_MAXSEG, &optval, &optlen) == 0)
    fprintf(file, "TCP_MAXSEG: %d\n", optval);
else
    perror("getsockopt TCP_MAXSEG");

fclose(file);
```

```
    close(sockfd);
    printf("Socket options written to socket_options.txt\n");

    return 0;
}
```

OUTPUT:





Program 4: Exercises on Socket Programming using C and Java.

SEVER:

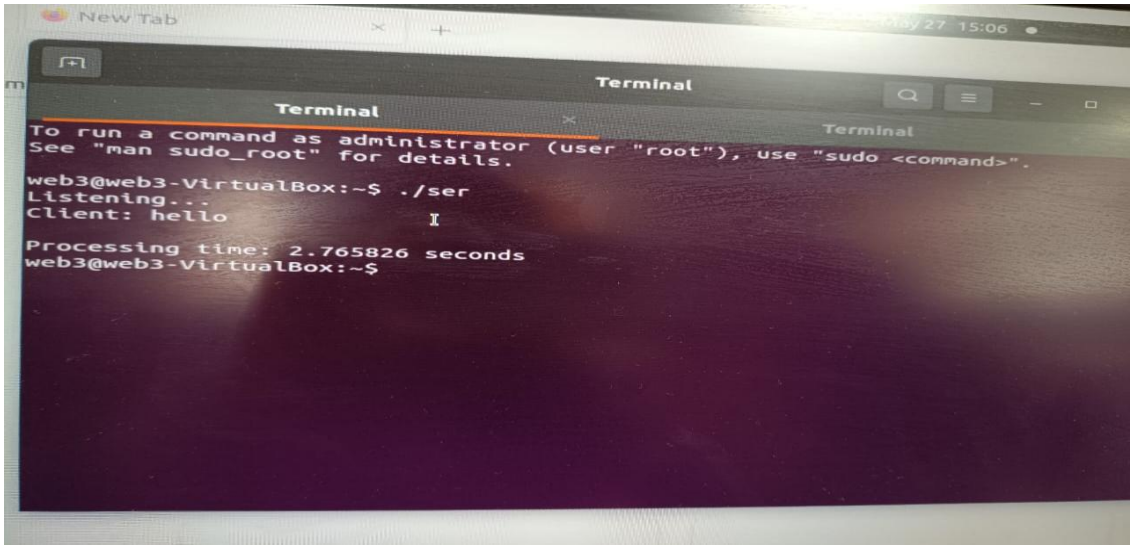
```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <netinet/in.h>
#include <sys/time.h>  // For gettimeofday()

int main() {
    int sfd, cfd; char buf[1024];
    struct sockaddr_in addr; socklen_t len = sizeof(addr);
    struct timeval start, end;

    sfd = socket(AF_INET, SOCK_STREAM, 0);
    addr.sin_family = AF_INET; addr.sin_addr.s_addr = INADDR_ANY; addr.sin_port = htons(8080);
    bind(sfd, (struct sockaddr *)&addr, sizeof(addr));
    listen(sfd, 3); printf("Listening...\n");

    cfd = accept(sfd, (struct sockaddr *)&addr, &len);

    gettimeofday(&start, NULL); // Start time before processing
    int n = read(cfd, buf, 1024); buf[n] = '\0';
    printf("Client: %s\n", buf);
    send(cfd, buf, n, 0);
    gettimeofday(&end, NULL); // End time after sending

    // Calculate processing time in microseconds
    long seconds = end.tv_sec - start.tv_sec;
    long microseconds = end.tv_usec - start.tv_usec;
    double elapsed = seconds + microseconds*1e-6;
```

```
    printf("Processing time: %.6f seconds\n", elapsed);

    close(cfd); close(sfd);
    return 0;
}
```

CLIENT:

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

int main() {
    int sock; char buf[1024];
    struct sockaddr_in serv;

    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv.sin_family = AF_INET; serv.sin_port = htons(8080);
    inet_pton(AF_INET, "127.0.0.1", &serv.sin_addr);
    connect(sock, (struct sockaddr *)&serv, sizeof(serv));

    printf("Enter message: ");
    fgets(buf, sizeof(buf), stdin);
    send(sock, buf, strlen(buf), 0);
    int n = read(sock, buf, sizeof(buf)); buf[n] = '\0';
    printf("Server: %s\n", buf);
    close(sock);
    return 0;
}
```

OUTPUT:



JAVA PROGRAM

SERVER:

```
import java.io.*;
import java.net.*;


public class Server {
    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(8080)) {
            System.out.println("Listening on port 8080...");


            Socket clientSocket = serverSocket.accept();
            long startTime = System.nanoTime(); // Start timing


            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);


            String clientMessage = in.readLine();
```

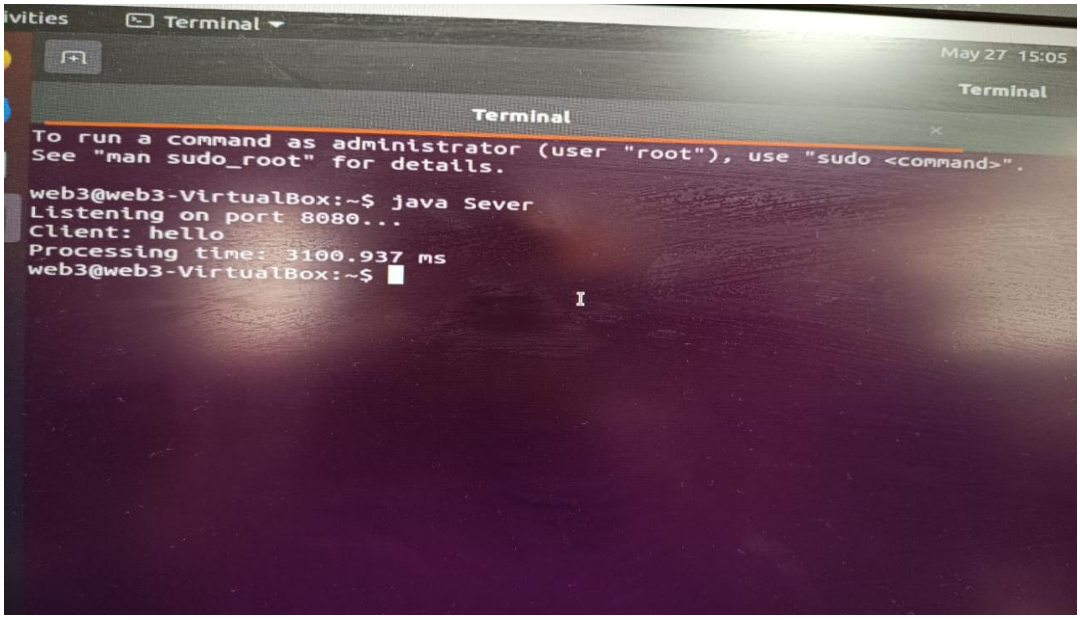
```
            System.out.println("Client: " + clientMessage);
            out.println(clientMessage);

            long endTime = System.nanoTime(); // End timing
            double elapsedTime = (endTime - startTime) / 1e6; // Convert to milliseconds
            System.out.printf("Processing time: %.3f ms\n", elapsedTime);

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

CLIENT:

```
import java.io.*;
import java.net.*;

public class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("127.0.0.1", 8080)) {
            BufferedReader userInput = new BufferedReader(new
InputStreamReader(System.in));
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            System.out.print("Enter message: ");
            String message = userInput.readLine();
            out.println(message);

            String serverMessage = in.readLine();
```

```
            System.out.println("Server: " + serverMessage);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:



Program5: Exercises using Cisco packet tracer

a. **Setting up of various network topologies**

b. **Implementation of various MAC protocols**

c. **Analysis of TCP/IP protocol under various mechanisms**

5)a) Setting up of various network topologies

1. **Bus**
2. **Star**
3. **Ring**
4. **Mesh**

Software/Tools Required

Cisco Packet Tracer (version 7.x or later)

PC/Laptop with minimum 4 GB RAM

Theory

Network Topology refers to the physical or logical arrangement of nodes in a network. Different topologies offer various benefits and drawbacks related to performance, scalability, and fault tolerance.

Topology	Description	Central Device	Collision Domain
Bus	Shared backbone	None	High
Star	Devices connected via central switch/hub	switch/hub	Low (with switch)
Ring	Each device connects to two others	None	Medium
Mesh	Every node connected to every other	None	Low

1. Star Topology

Steps:

1. Open Packet Tracer → Add 1 Switch and 5 PCs.

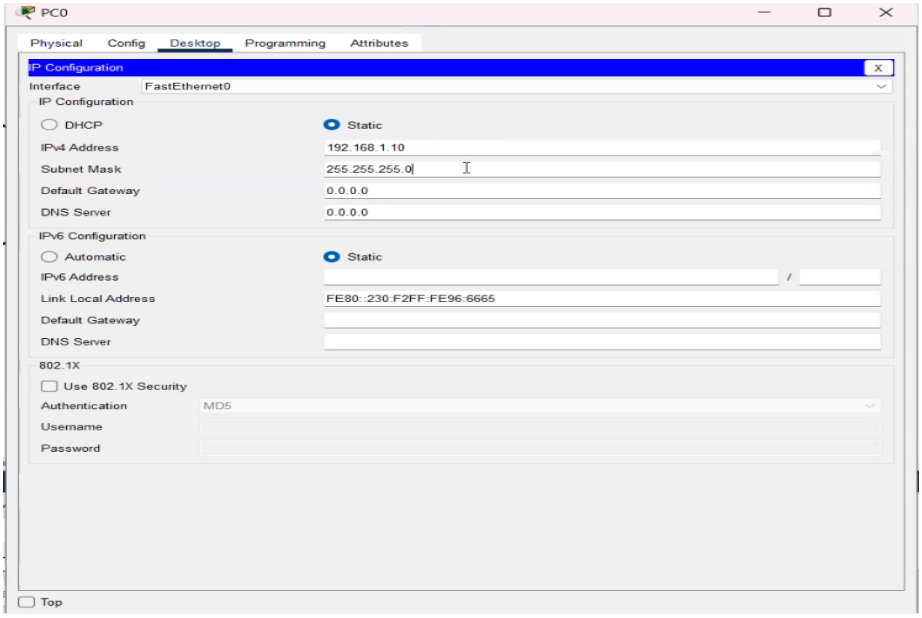
2. Connect each PC to the switch using **Copper Straight-Through** cable.

3. Assign IP addresses:

PC0: 192.168.1.1 /24

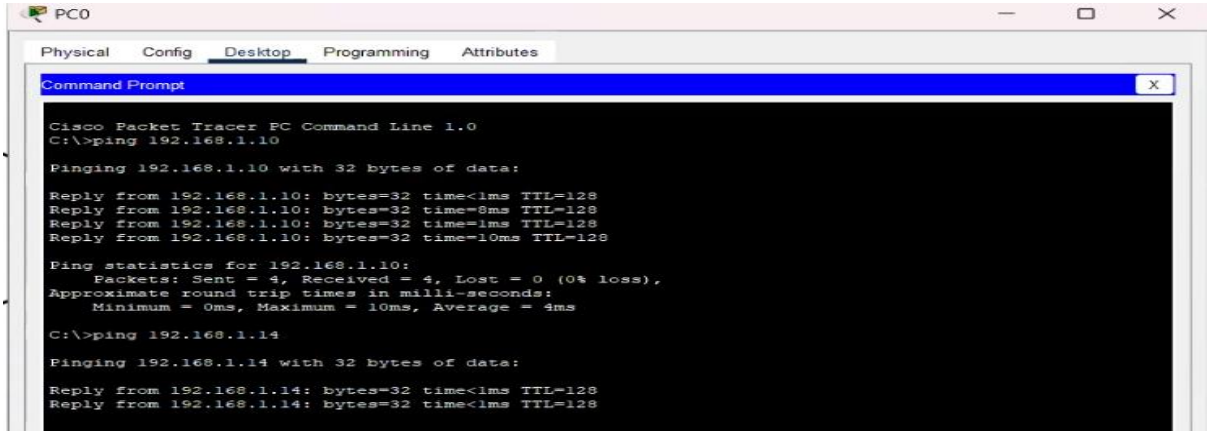
PC1: 192.168.1.2 /24

PC2: 192.168.1.3 /24

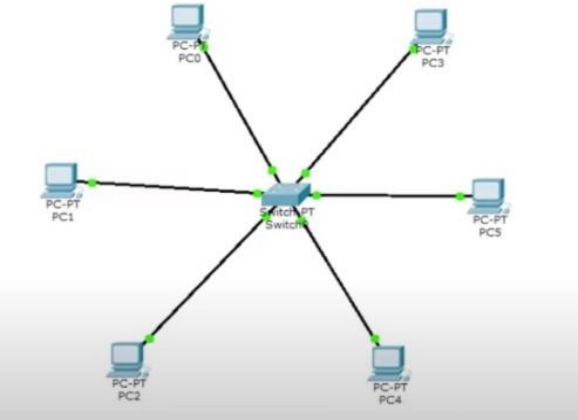


4.Use Command Prompt to test:

ping 192.168.1.2 from PC0



OUTPUT:



2.Bus Topology

Steps:

1. Add 4 switch and 4 PCs/Laptops.

2. Connect all PCs to the hub using **Copper Straight-Through** cable.

3. Assign IPs:

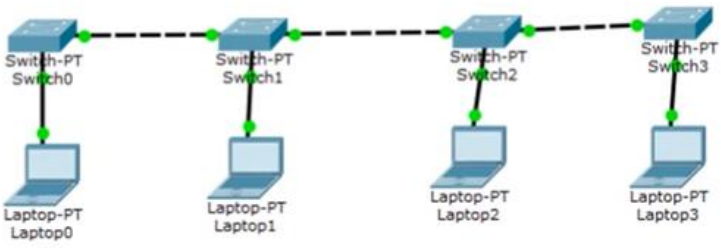
PC0: 192.168.1.10 /24

PC1: 192.168.1.11 /24

PC2: 192.168.1.12 /24

4. Switch to Simulation Mod, send pings from all PCs simultaneously → observe collisions (Bus behavior).

OUTPUT:



3.Ring Topology

Steps:

1. Add 4 switch and 4 PCs/Laptops.

2. Connect:

   Sw0 ↔ PC0

   sw1 ↔ PC1

   sw2 ↔ PC2

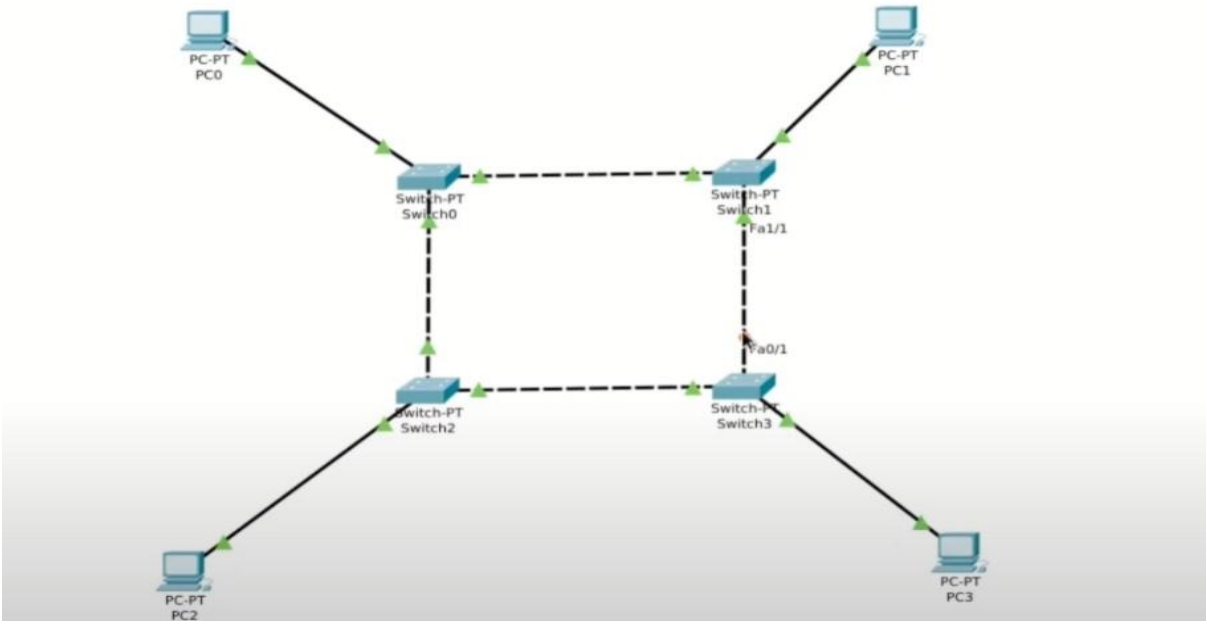
   sw3 ↔ PC3

   Use **Cross-Over Cables**

3. Assign IPs from same network.

4. Test connectivity using `ping` from any PC to others.

OUTPUT:



4.Mesh Topology

Steps:

1. Add 4 Switch 4 PCs/Laptops.

2. Connect each PC to all others using **Cross-Over Cables**

3. Assign IPs:

   PC0: 192.168.1.1

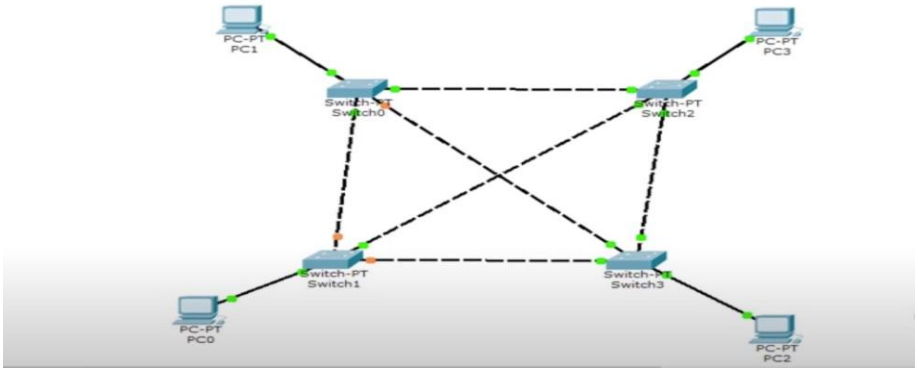
   PC1: 192.168.1.2

   PC2: 192.168.1.3

   PC3: 192.168.1.4

4. Ping between every pair to demonstrate full connectivity.

OUTPUT:



5)b). Implementation of various MAC protocol

Introduction to MAC Protocols

Medium Access Control (MAC) protocols are responsible for coordinating how data is transmitted over a shared medium, preventing collisions, and ensuring efficient use of bandwidth.

Common Types of MAC Protocols

Protocol Type	Description
CSMA/CD (Carrier Sense Multiple Access/Collision Detection)	Used in wired Ethernet; devices sense the medium before transmitting.
CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance)	Used in wireless networks (e.g., Wi-Fi); devices avoid collisions by using acknowledgment and RTS/CTS mechanisms.
TDMA (Time Division Multiple Access)	Allocates fixed time slots to each node (not supported directly in Packet Tracer).
FDMA/CDMA	Used in telecom; not directly implementable in Packet Tracer.

Cisco Packet Tracer supports CSMA/CD and CSMA/CA behaviors based on Ethernet and Wireless configurations.

Tools and Requirements

- Cisco Packet Tracer (version 7.3 or higher recommended)
- Basic understanding of network topologies
- Devices: PCs, Switches, Routers, Wireless Routers, Access Points

---

Steps to Implement MAC Protocols in Packet Tracer

1. Implementing CSMA/CD (Wired Ethernet)

   Devices Required:

   - 2+ PCs
   - Ethernet Switch
   - Copper Straight-Through Cables

Steps:

1. Place Devices:
   - Drag and drop at least two PCs and one switch onto the workspace.
2. Connect Devices:
   - Use Copper Straight-Through Cable to connect each PC to the switch.
3. Assign IP Addresses:
   - Click on each PC → Desktop → IP Configuration.
   - Assign static IPs (e.g., PC1: 192.168.1.2, PC2: 192.168.1.3, Subnet: 255.255.255.0).
4. Test Communication:
   - Use the ping command to verify connectivity.
   - Simultaneously start pings from both PCs to simulate potential collisions.
5. Simulate Collision Scenario:
   - Switch to Simulation Mode (bottom-right).
   - Send pings simultaneously and observe the Ethernet frame flow.

Note: Packet Tracer doesn't visibly show collision and backoff, but CSMA/CD is implemented under the hood for wired Ethernet.

---

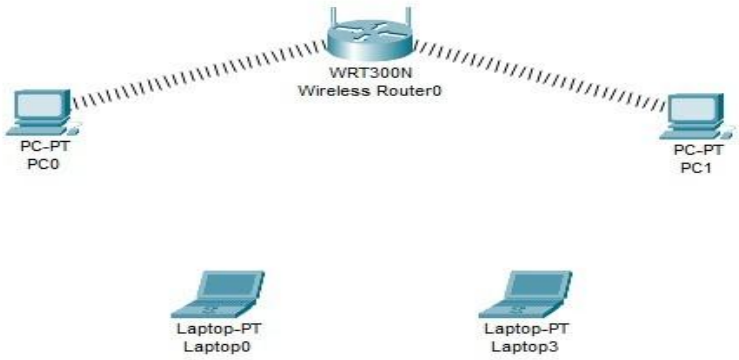
2. Implementing CSMA/CA (Wireless) Devices Required:

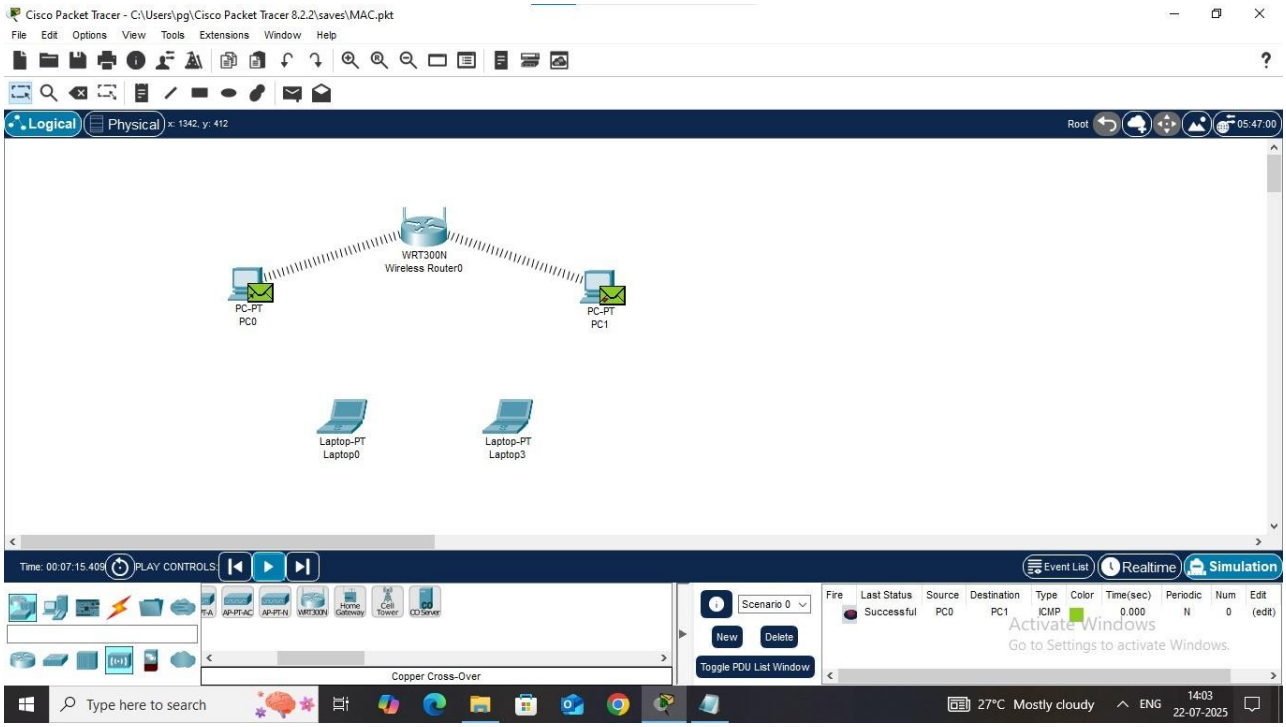
   - 2+ Laptops or Wireless PCs
   - Wireless Router or Access Point

Steps:

1. Place Devices:
    - Add a Wireless Router and 2+ wireless-enabled PCs or laptops.
2. Configure Wireless Router:
    - Click the router → Wireless tab → Wireless MAC Filter → Set the MAC Address (Paste the PC MAC Address). o Set security if needed (e.g., WPA2 PSK).
3. Configure Clients:
    - Click the PC change the wireless icon → Config → Wireless Tab (copy the MAC Address).
4. Test Connectivity:
    - Ping between devices.
    - Observe wireless frames in Simulation Mode.
5. Observe MAC Layer Behaviour:
        - In Simulation Mode, you'll notice:
        - RTS (Request to Send) / CTS (Clear to Send) messages if security and MAC filtering are enabled.
        - Acknowledgement and retry packets if collisions or interference occur.

OUTPUT:


WRT300N
Wireless Router0
PC-PT
PC0
PC-PT
PC1
Laptop-PT
Laptop0
Laptop-PT
Laptop3


Cisco Packet Tracer - C:\Users\pg\Cisco Packet Tracer 8.2.2\saves\MAC.pkt
File Edit Options View Tools Extensions Window Help
Logical
Physical
Root
WRT300N
Wireless Router0
PC-PT
PC0
PC-PT
PC1
Laptop-PT
Laptop0
Laptop-PT
Laptop3
Time: 00:07:15.409
PLAY CONTROLS:
Event List
Realtime
Simulation
Copper Cross-Over
Scenario 0
New
Delete
Toggle PDU List Window
Fire
Last Status
Source
Destination
Type
Color
Time(sec)
Periodic
Num
Edit
Successful
PC0
PC1
ICMP
0.000
N
0
(edit)
Activate Windows
Go to Settings to activate Windows.
Type here to search
27°C Mostly cloudy
ENG
14:03
22-07-2025

5) c) Analysis of TCP/IP protocol under various mechanisms

OBJECTIVE:

Analyze how the TCP/IP protocol stack operates under:

1. Normal operation (TCP handshake)
2. ICMP diagnostics (ping, traceroute)
3. FTP and HTTP simulation

Build the Basic Network Topology

Devices You Need:

- **1 Router**: Cisco 1941
- **2 PCs**: PC0 and PC1
- **Cables**: Copper straight-through cables

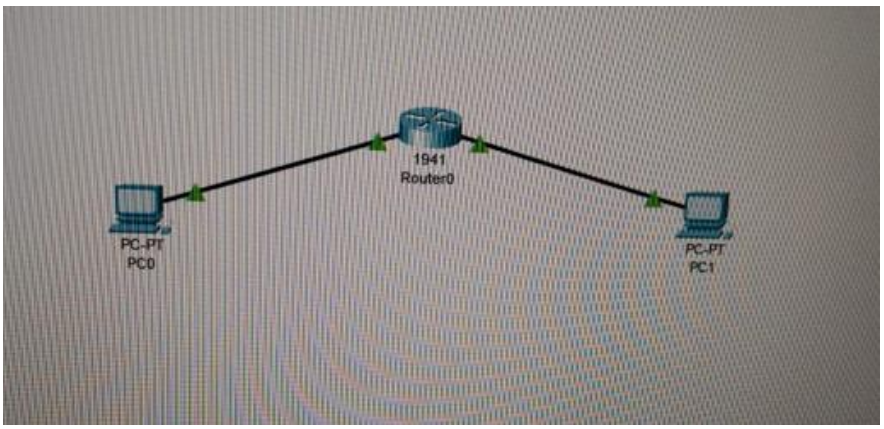
STEP-BY-STEP CONNECTIONS:

1. Connect PC0 to Router (Gig0/0)

- Click the **"Lightning Bolt" icon** (for cables) → choose **"Copper Straight-Through"**
- Click on **PC0**, then select **FastEthernet0**
- Click on the **Router**, then select **GigabitEthernet0/0**

2. Connect PC1 to Router (Gig0/1)

- Use another **"Copper Straight-Through"** cable
- Click on **PC1**, then select **FastEthernet0**
- Click on the **Router**, then select **GigabitEthernet0/1**



IP CONFIGURATION

To allow communication between the two PCs through the router (and observe TCP/IP behavior), assign **IP addresses and default gateways**:

On PC0:

1. Click PC0 → Desktop → IP Configuration
2. Set:
    - IP Address: 192.168.1.2
    - Subnet Mask: 255.255.255.0
    - Default Gateway: 192.168.1.1

On PC1:

1. Click PC1 → Desktop → IP Configuration
2. Set:
    - IP Address: 192.168.2.2
    - Subnet Mask: 255.255.255.0
    - Default Gateway: 192.168.2.1

On Router 1941:

Configure Router Interfaces

1. Click on the Router → Go to CLI tab

2. Enter Configuration Mode:

Router> enable

Router# configure terminal

Configure Interface Gig0/0 (connected to PC0):

Router(config)# interface gigabitEthernet 0/0

Router(config-if)# ip address 192.168.1.1 255.255.255.0

Router(config-if)# no shutdown

Router(config-if)# exit

Configure Interface Gig0/1 (connected to PC1):

Router(config)# interface gigabitEthernet 0/1

Router(config-if)# ip address 192.168.2.1 255.255.255.0

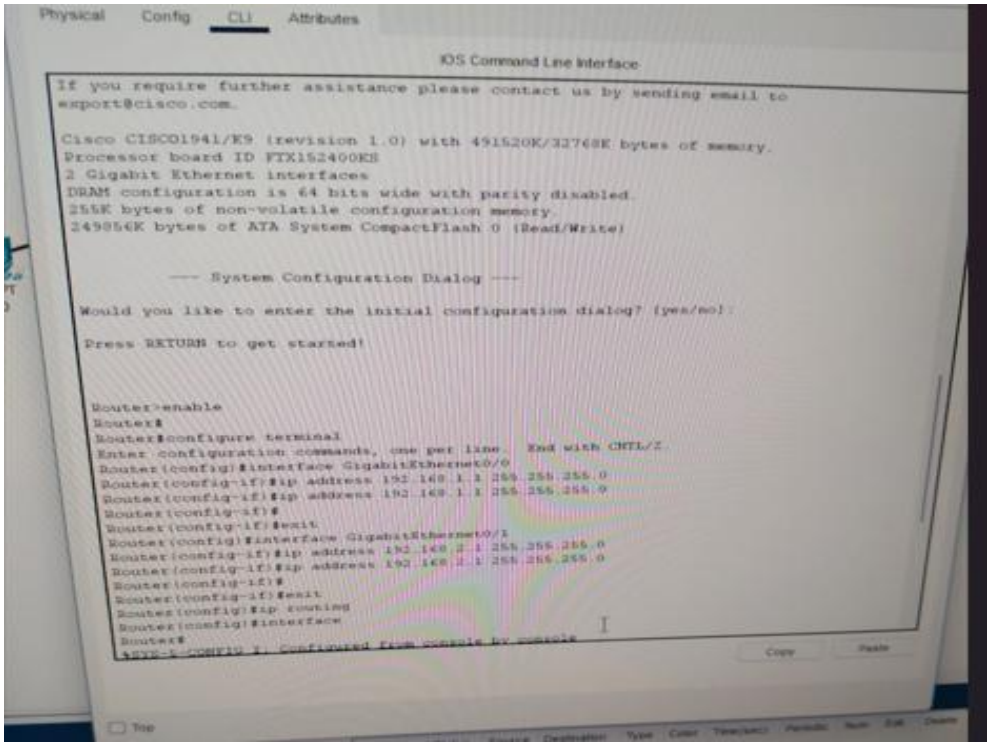
Router(config-if)# no shutdown

Router(config-if)# exit

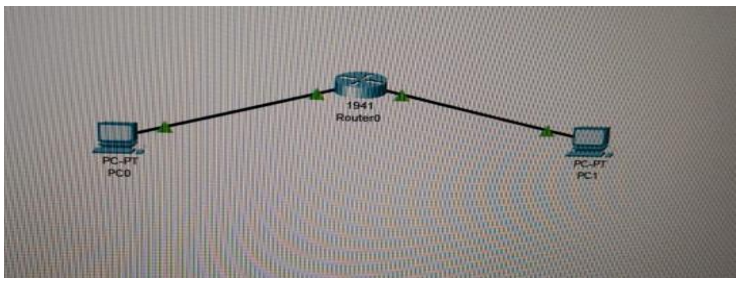
Exit Configuration Mode:

Router(config)# exit

Router#



You'll see interface lights turn green, indicating successful connections.



Test Connectivity (Ping)

From PC0:

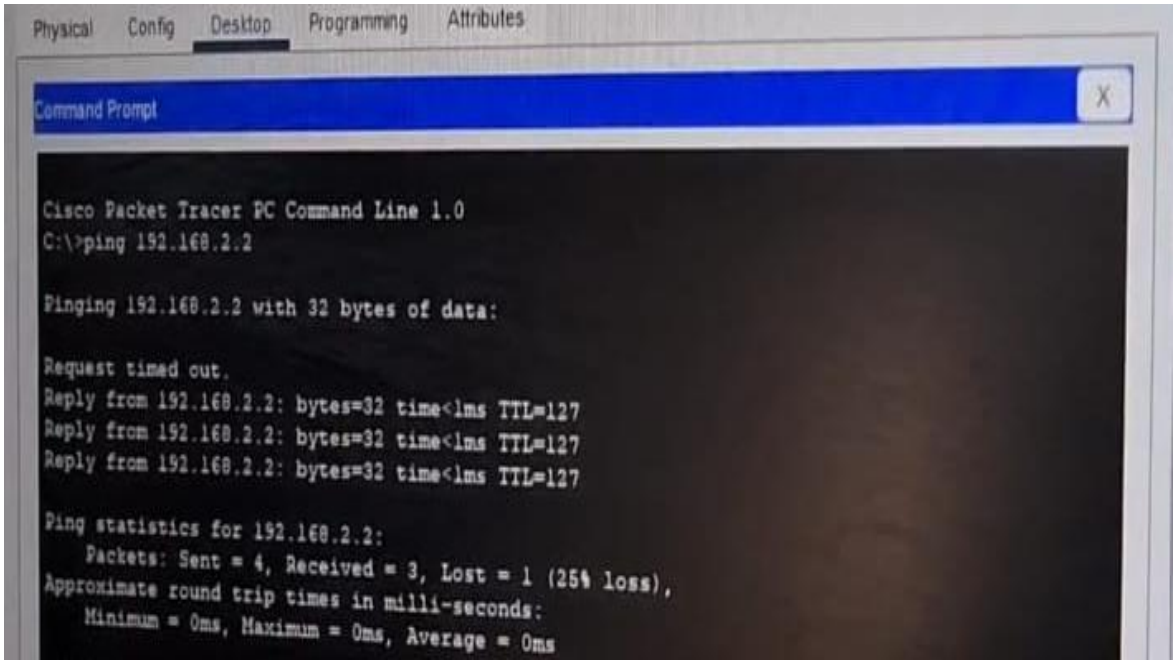
1. Go to Desktop → Command Prompt

2. Type: **ping 192.168.2.2**

SCENARIO 1: TCP 3-Way Handshake

open Command Prompt in PC0: ping 192.168.2.2

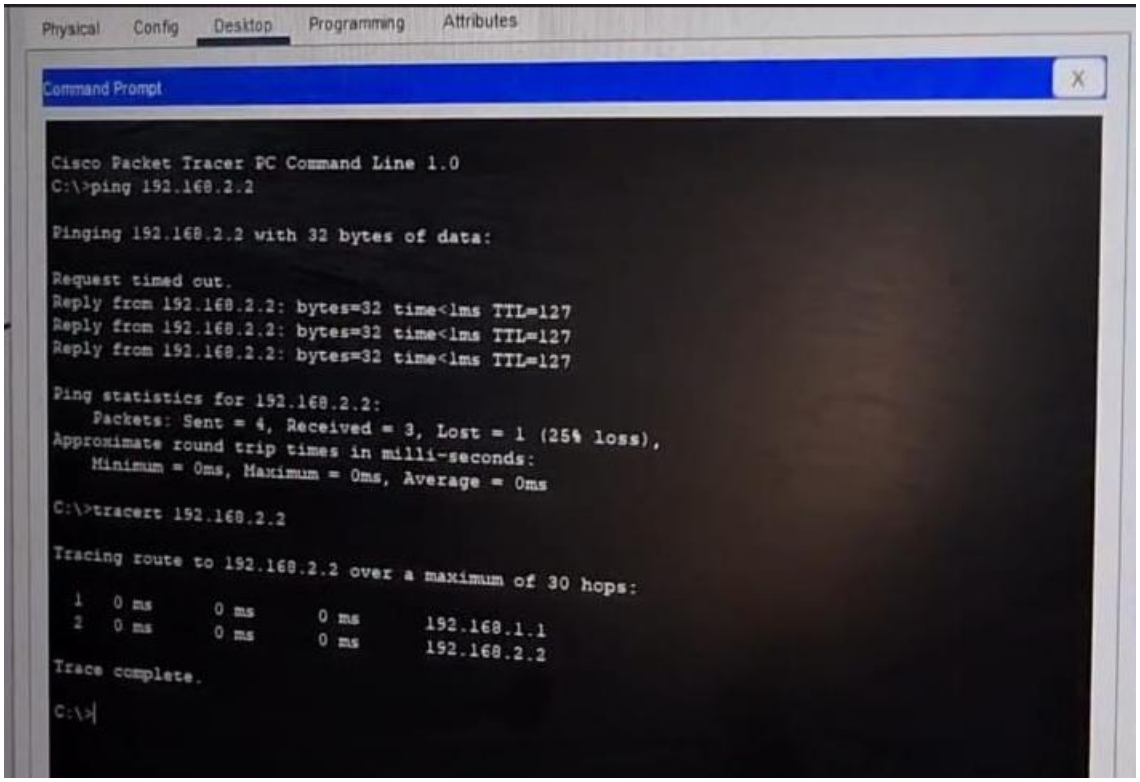
You should see successful replies if everything is correctly configured.



SCENARIO 2: Traceroute or ICMP TTL Expiry

open Command Prompt in PC0: tracert 192.168.2.2

- Simulate tracert from PC0 to PC1 across multiple routers
- Observe ICMP Time Exceeded replies when TTL expires
- View step-by-step packet flow using Simulation Mode



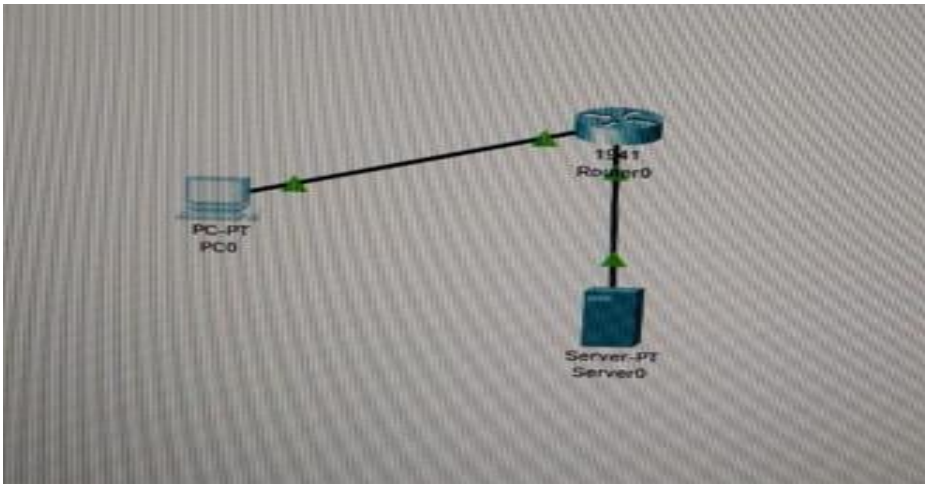
SCENARIO 3: Application-Level Protocol (HTTP and FTP)

- **Add a Server (Server0).**
- **Set IP: 192.168.2.10, enable HTTP and FTP services**

Connection Requirements:

Assuming you're using a simple router-based topology:

PC0 ↔ Router ↔ Server



Example IP Scheme:

Device	Interface	IP Address
PC0	FastEthernet0	192.168.1.2
Router	Gig0/0	192.168.1.1
Router	Gig0/1	192.168.2.1
Server	FastEthernet0	192.168.2.10

✅ Connect:

- **PC0 to Router Gig0/0**
- **Server to Router Gig0/1**

Note: also set the default gateway address as 192.168.2.1

- Connect **PC0 to Router Gig0/0** using straight-through cable
- Connect **Server to Router Gig0/1** using another straight-through cable
- Configure IP addresses and gateways (as in the table above)
- On the **Server**, go to **Services tab** and turn on:
    - **HTTP**
    - **FTP**

Verify FTP Service is ON

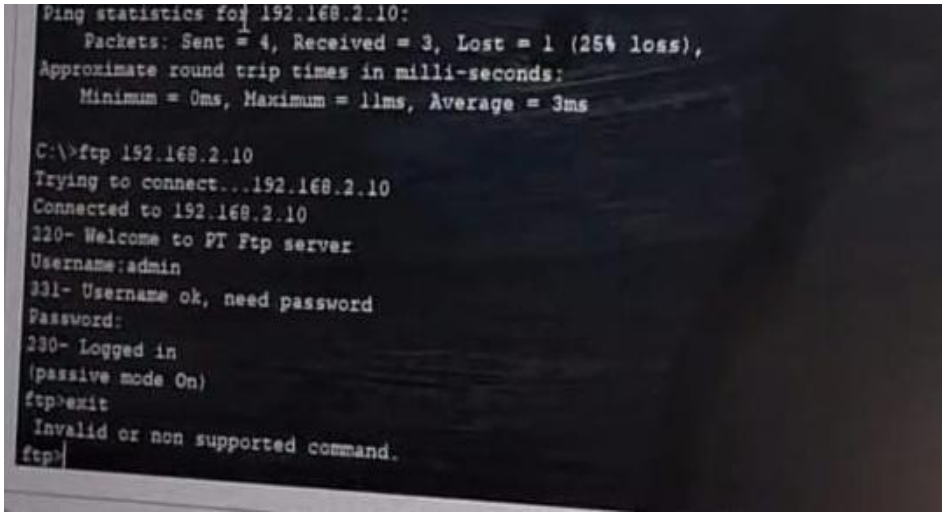
- On **Server0**, go to **Services tab**
- Click on **FTP**
    - Make sure **FTP is ON**
    - There should be at least one **username/password** defined (e.g., admin / admin)

If needed, add a user:

- Username: admin
- Password: admin
- Click **Add**
- On **PC0**, open command prompt:

ftp 192.168.2.10

Enter username and password when prompted.

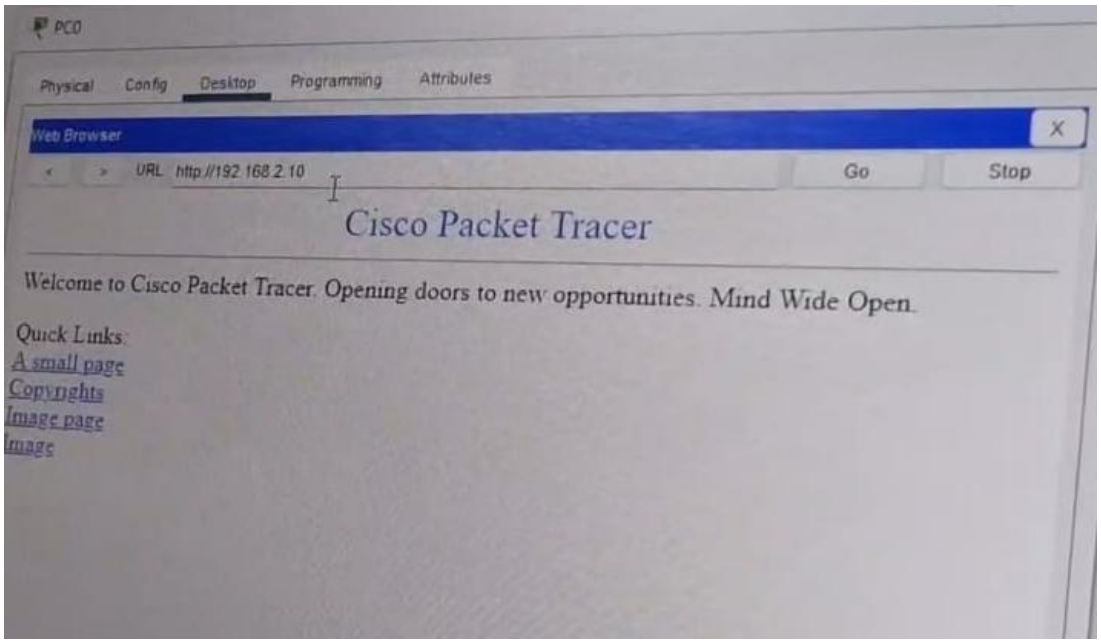


Test HTTP

From PC0 → Desktop → Web Browser

In URL bar, type: http://192.168.2.10

You should see the default web page from the server.



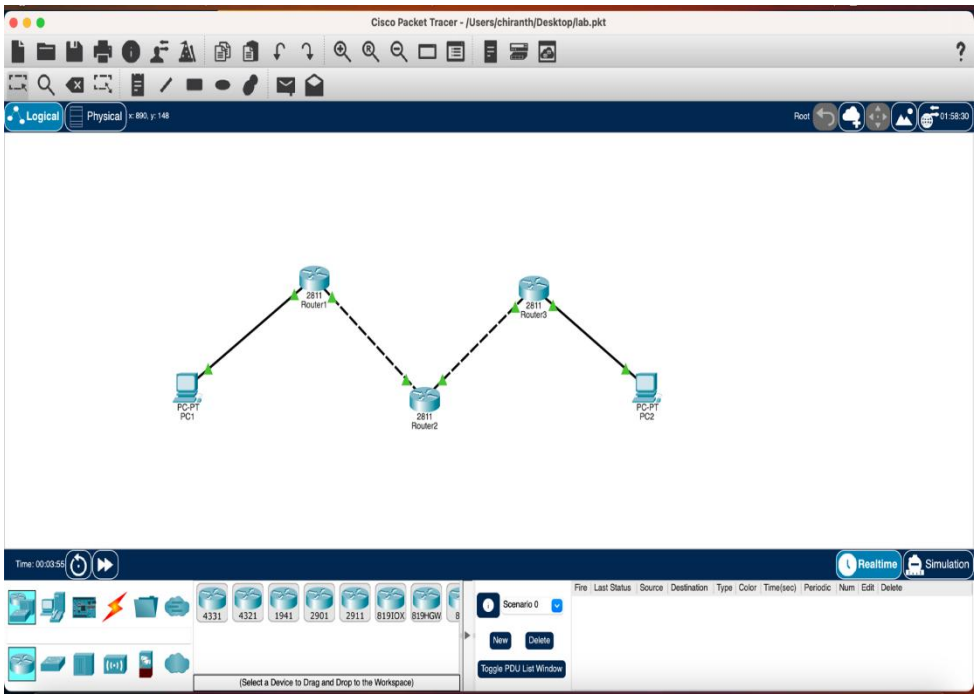
Program 6: **Exercises using Cisco packet tracer Setting up of network that carries various application routing protocols and analyzing their performances.**

To demonstrate the performance of two routing protocols **(**RIP and EIGRP**)** based on parameters like convergence time using Cisco Packet Tracer with 2 PCs and 3 routers**.**

STEP 1: Setup the Network in Packet Tracer

Devices:

- 2 PCs
- 3 Routers (e.g., 2811)
- Cables: Copper Straight-Through (for PC to Router), Copper Cross-over cable between routers



Step 2: Basic Device Configuration

🖥 PC1

- IP: 192.168.1.10
- Subnet: 255.255.255.0
- Default Gateway: 192.168.1.1

🖥 PC2

- IP: 192.168.3.10
- Subnet: 255.255.255.0
- Default Gateway: 192.168.3.1

(Router configuration to be done in router CLI)

R1 Configuration

```
enable
config t
hostname R1

interface f0/0
ip address 192.168.1.1 255.255.255.0
no shutdown

interface f0/1
ip address 10.0.12.1 255.255.255.0
no shutdown
exit
```

R2 Configuration

```
enable
config t
hostname R2
```

interface f0/0
ip address 10.0.12.2 255.255.255.0
no shutdown

interface f0/1
ip address 10.0.23.1 255.255.255.0
no shutdown
exit

R3 Configuration

enable
conf t
hostname R3

interface f0/0
ip address 192.168.3.1 255.255.255.0
no shutdown

interface f0/1
ip address 10.0.23.2 255.255.255.0
no shutdown
exit

Device	Interface	IP Address	Subnet Mask
PC1	F0	192.168.1.10	255.255.255.0
R1 F0/0		192.168.1.1	255.255.255.0
R1 F0/0		10.0.12.1	255.255.255.0
R2 F0/0		10.0.12.2	255.255.255.0
R2 F0/1		10.0.23.1	255.255.255.0
R3 F0/0		192.168.3.1	255.255.255.0
PC3	F0	192.168.3.10	255.255.255.0

Step 3: Configure RIP on all 3 routers

On R1:
config t
router rip
version 2
no auto-summary
network 192.168.1.0
network 10.0.0.0
exit

On R2:
config t
router rip
version 2
no auto-summary
network 10.0.0.0
exit

On R3:
conf t
router rip
version 2
no auto-summary
network 192.168.3.0
network 10.0.0.0
exit

Once routing protocol is configured on all routers, ping from P1 to P2 should be successful

Step 4: Measure RIP convergence time

What is Convergence Time?
Time taken for all routers to learn about the full network after a change (e.g., link down/up).

Switch to Simulation Mode

- At the **bottom right**, click **Simulation Mode** -You'll now see protocol packets (e.g., RIP, ICMP) in the event list
- Click **Edit Filters** -Tick only: RIP and ICMP
- Open PC1 → Desktop → Command Prompt - **ping** from PC1 to PC3,
- Start simulation to notice ICMP packets in simulation time line
- Stop simulation
- Click the **Delete tool (red X)**
- Click the **cable between R2 and R3 -** This simulates a **link failure , start simulation again**
- **Observe the event list- ICMP packets start to fail**
- RIPv2 updates are exchanged between R1 ↔ R2
- Once RIP updates finish and **routes are relearned**, the **ICMP pings will succeed again**
- Find the **last successful ICMP Echo Reply** (just before failure)
- Find the **first successful ICMP Echo Reply after the link break and RIP recovery Subtract the timestamps** → That's your convergence time!

Event Timestamp

First ping failure 0:14.000

RIP updates occur 0:44.000

First successful ping again 0:47.000

➤ **Convergence Time** ~33 secs

(convergence time is for RIP is usually between 30-180 seconds)

Step 5: Remove RIP configuration

On **each router**, go to CLI and enter **config mode**

config t
no router rip

Do this on **Router1, Router2, and Router3**

Step 6: Configure EIGRP on all 3 routers

Router 1

```
Router1> enable
Router1# configure terminal
Router1(config)# router eigrp 100
Router1(config-router)# network 192.168.1.0 0.0.0.255
Router1(config-router)# network 10.0.12.0 0.0.0.255
Router1(config-router)# no auto-summary
Router1(config-router)# exit
```

Router 2

```
Router2> enable
Router2# configure terminal
Router2(config)# router eigrp 100
Router2(config-router)# network 10.0.12.0 0.0.0.255
Router2(config-router)# network 10.0.23.0 0.0.0.255
Router2(config-router)# no auto-summary
Router2(config-router)# exit
```

Router 3

```
Router3> enable
Router3# configure terminal
Router3(config)# router eigrp 100
Router3(config-router)# network 192.168.3.0 0.0.0.255
Router3(config-router)# network 10.0.23.0 0.0.0.255
Router3(config-router)# no auto-summary
Router3(config-router)# exit
```

Step 7: Measure EIGRP convergence time

- Start “Simulation Mode”
- Start continuous ping: On PC1 → ping 192.168.3.1
- Observe ICMP (black = request, red = reply)
- Break the Link -Remove the link between Router2 and Router3
- Watch: ICMP packets fail (black only) **,**EIGRP (D) packets start appearing**,** Eventually, ICMP
- From the **Event List**, note:

Event	Time (sec)
Last ICMP reply before failure	e.g., 24.500
First ICMP reply after EIGRP recovery	e.g., 27.500

Convergence Time = 27.5 - 24.5 = 3 seconds

Parameter	RIP	EIGRP
Convergence Time	30–180 sec	<10 sec
Protocol Type	Distance-vector	Hybrid
Metric	Hop Count	Bandwidth + Delay
Classless	Yes (RIPv2)	Yes

Program 7: Comparison of TCP/IP socket , pipes. Analyse which is best Pipes

1) **Unidirectional communication (Parent writes, Child reads)**

2) **Unidirectional communication (Child writes, Parent reads)**

3) **Bidirectional communication (Both processes write and read using two pipes)**

CASE 1: Parent Writes, Child Reads

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd[2];
    pipe(fd);  // create pipe

    pid_t pid = fork();

    if (pid > 0) {  // Parent
        close(fd[0]);  // Close read end
        char msg[] = "Hello from parent!";
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    } else if (pid == 0) {  // Child
        close(fd[1]);  // Close write end
        char buffer[100];
        read(fd[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(fd[0]);
    }

    return 0;
```

```
}
```

Compile and Run

gcc filename.c -o filename

./filename

CASE 2: Child Writes, Parent Reads

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int fd[2];
    pipe(fd);

    pid_t pid = fork();

    if (pid == 0) {  // Child
        close(fd[0]);
        char msg[] = "Hello from child!";
        write(fd[1], msg, strlen(msg) + 1);
        close(fd[1]);
    } else {  // Parent
        close(fd[1]);
        char buffer[100];
        read(fd[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);
        close(fd[0]);
    }
```

```
    return 0;
}
```

Compile and Run

gcc filename.c -o filename

./filename

CASE 3: Bidirectional Communication Using Two Pipes

You need two pipes, since pipe() is unidirectional.

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int pipe1[2], pipe2[2];
    pipe(pipe1); // Parent -> Child
    pipe(pipe2); // Child -> Parent

    pid_t pid = fork();

    if (pid > 0) {  // Parent
        close(pipe1[0]);  // Close read end of pipe1
        close(pipe2[1]);  // Close write end of pipe2

        char msg[] = "Hello from parent!";
        write(pipe1[1], msg, strlen(msg) + 1);
        close(pipe1[1]);

        char buffer[100];
        read(pipe2[0], buffer, sizeof(buffer));
        printf("Parent received: %s\n", buffer);
```

```
        close(pipe2[0]);
    } else {  // Child
        close(pipe1[1]);  // Close write end of pipe1
        close(pipe2[0]);  // Close read end of pipe2

        char buffer[100];
        read(pipe1[0], buffer, sizeof(buffer));
        printf("Child received: %s\n", buffer);
        close(pipe1[0]);

        char reply[] = "Hi Parent, message received!";
        write(pipe2[1], reply, strlen(reply) + 1);
        close(pipe2[1]);
    }

    return 0;
}
```

Compile and Run

gcc filename.c -o filename

./filename

CASE 4: Both Try to Write to the Same Pipe

This is **technically valid**, but can cause race conditions or garbled output unless managed with synchronization.

Use synchronization (e.g., wait(), semaphores)

To handle the race condition in Case 4 (where both parent and child write to the same pipe), we can use synchronization. A common approach is to use wait() to ensure that one process finishes writing before the other begins.

Case 4: Both parent and child write to the same pipe, but synchronized using wait() and sleep() to avoid race conditions.

Both messages are also read and printed in order from the pipe. Case 4, where both parent and child write to the same pipe in a synchronized manner using wait(). Then the parent reads back the combined message.

Case4

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <sys/wait.h>

int main() {
    int fd[2];
    char buffer[200];
    pipe(fd);

    pid_t pid = fork();

    if (pid == 0) {
        // Child: write after parent
        sleep(1);
        close(fd[0]);  // Close read end
        write(fd[1], "Child message.\n", 15);
        close(fd[1]);
    } else {
        // Parent
        close(fd[0]);  // Close read end
        write(fd[1], "Parent message.\n", 16);
        wait(NULL);    // Wait for child to finish
        close(fd[1]);

        // Reopen pipe to read (simulate shared read buffer)
```

```
        pipe(fd); // new pipe to read from
        pid_t reader = fork();
        if (reader == 0) {
            // Reader
            close(fd[1]);
            int n = read(fd[0], buffer, sizeof(buffer)-1);
            buffer[n] = '\0';
            printf("Combined messages:\n%s", buffer);
            close(fd[0]);
            exit(0);
        } else {
            close(fd[0]);
            close(fd[1]);
            wait(NULL);
        }
    }

    return 0;
}
```

Compile and Run

gcc filename.c -o filename

./filename

<u>Note:</u> For this Problem statement refer the sample program also which is in the starting of this report.

Comparison: TCP/IP Socket vs Pipe

Feature / Aspect	TCP/IP Sockets	Pipes (Named & Unnamed)
Definition	Communication endpoint over network (TCP/IP)	A method for inter-process communication (IPC)
Communication Type	Network communication (local or remote)	Local inter-process communication (IPC only)
Directionality	Full-duplex (two-way communication)	Unnamed: Half-duplexNamed: Full-duplex possible
Process Scope	Can connect processes on **different machines** or same	Only for **processes on the same machine**
Speed / Latency	**Slower** due to network stack overhead	**Faster** for local IPC (no networking layer)
Setup Complexity	More complex: requires socket creation, binding, etc.	Very simple setup, especially unnamed pipes
Reliability	Reliable (TCP ensures delivery/order)	Reliable (if handled properly in local memory)
Protocol Dependency	Depends on TCP/IP stack	No external protocol needed
Security	Requires authentication/encryption (e.g. SSL)	Safer in local systems; access controlled by OS
Data Format	Byte stream; programmer handles serialization	Byte stream (raw)
Programming Effort	More boilerplate code needed	Simpler code (especially for unnamed pipes)
Use in Distributed Systems	Ideal (across networks)	Not possible
Buffering	Kernel buffers for send/recv	OS kernel buffers (usually small)
Example Languages	Python: socket moduleC: socket.h	Python: os.pipe() or multiprocessing.Pipe()C: pipe() or mkfifo()

Which is Best?

✅ **Use Pipes If:**

- You need **simple, fast communication** between **related processes** on the **same machine**
- You want **low-latency IPC** with minimal setup
- You're building local tools like **shell pipelines**, **parallel processes**, etc.

✅ Best for: local-only applications, fast intra-process communication

✅ **Use TCP/IP Sockets If:**

- You need to communicate between **different machines** or across a **network**

- You want a **scalable** client-server model (e.g., web servers, APIs)
- Your app must run in **distributed** or **cloud environments**

✅ Best for: distributed systems, internet apps, real-time services

Conclusion

Best for Local IPC ➤**Pipes (faster, simpler)**

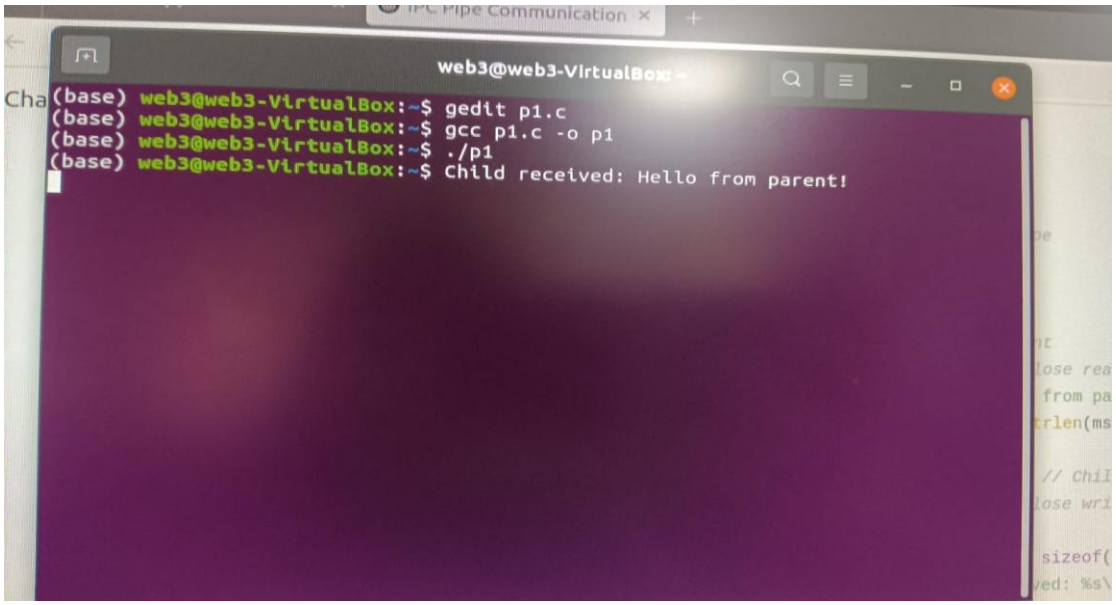
Best for Network Communication ➤ **TCP/IP Sockets** (scalable, network-capable)

No one is universally better — it depends on your **requirements**:

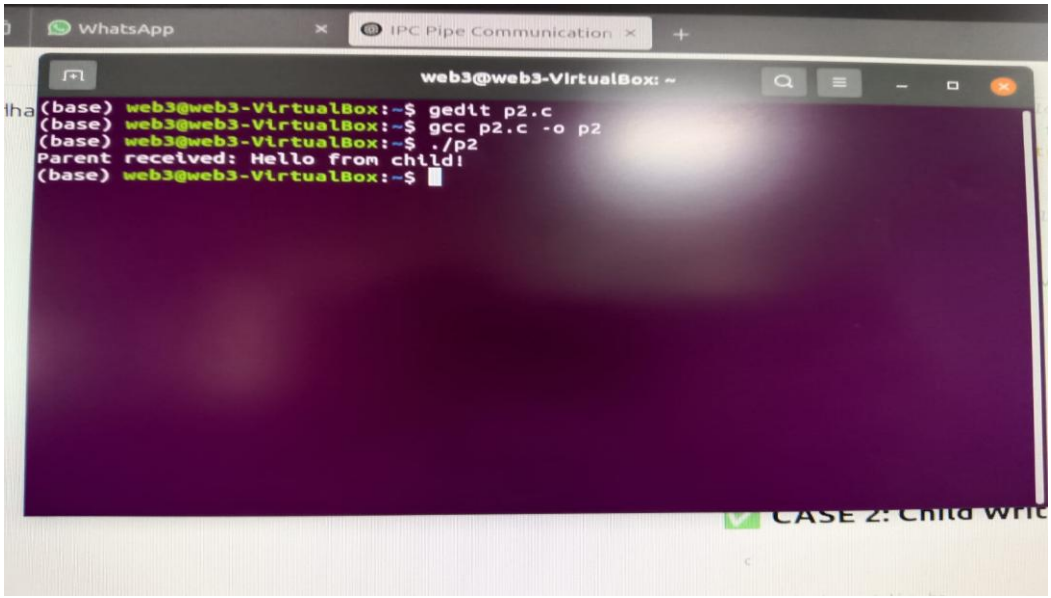
- Use **pipes** for performance-critical, simple **local IPC**
- Use **TCP/IP sockets** for **networked** or **multi-host** applications

OUTPUT:

For Case 1:



For Case 2:



For Case 3: