VIT®

Vellore Institute of Technology

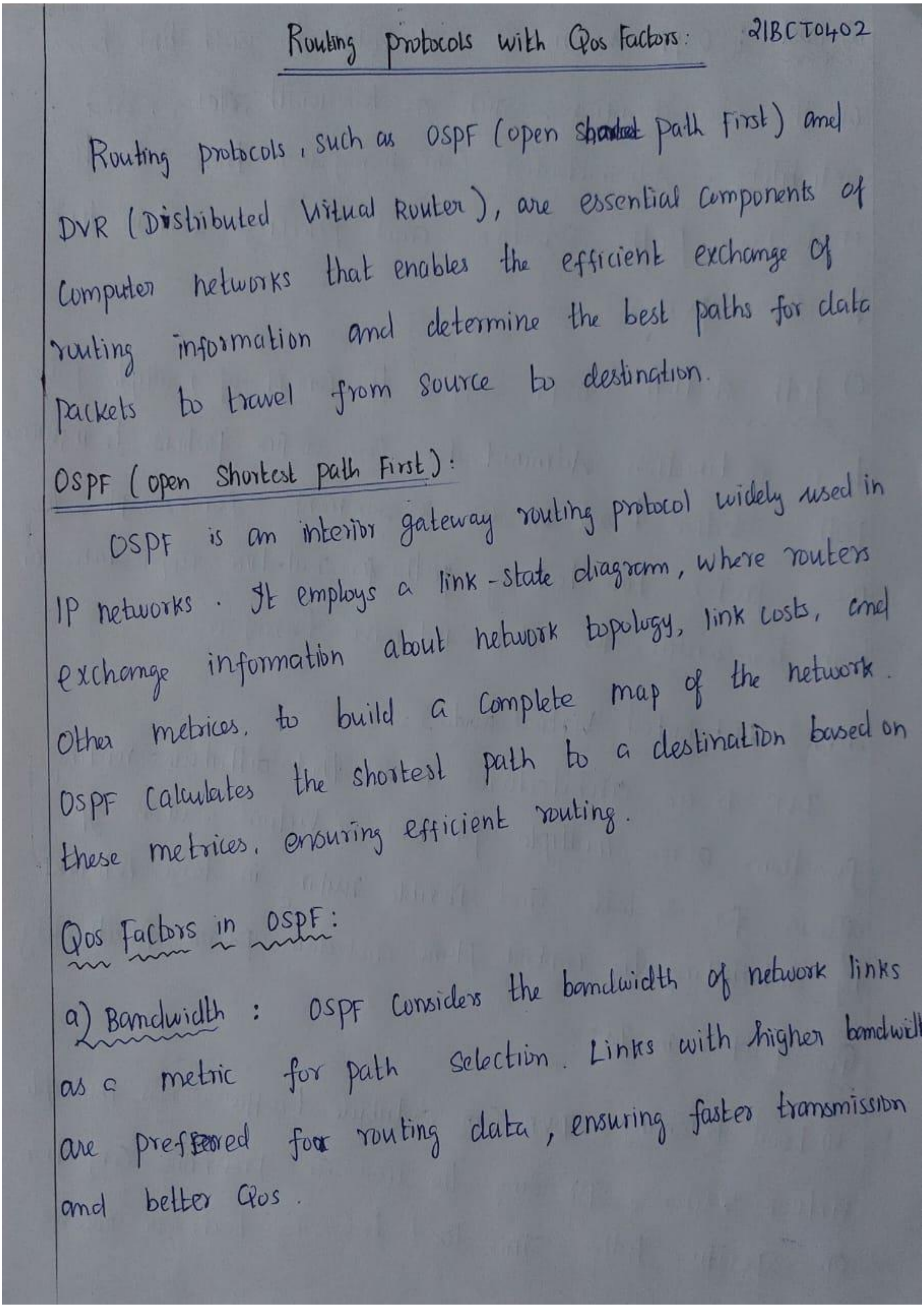
(Deemed to be University under section 3 of UGC Act, 1956)

COMPUTER NETWORKS LAB

Task 3: Exercise5 & Exercise6

Name	: Madasamy S
Reg no	: 21BCT0402
Course Name	: Computer Networks
Course code	: BCSE308P
Lab slot	:L25+L26

EXERCISE 5: Routing protocal (OSPF and DVR) with QoS factors.



Routing protocols with Qos Factors:

21BCT0402

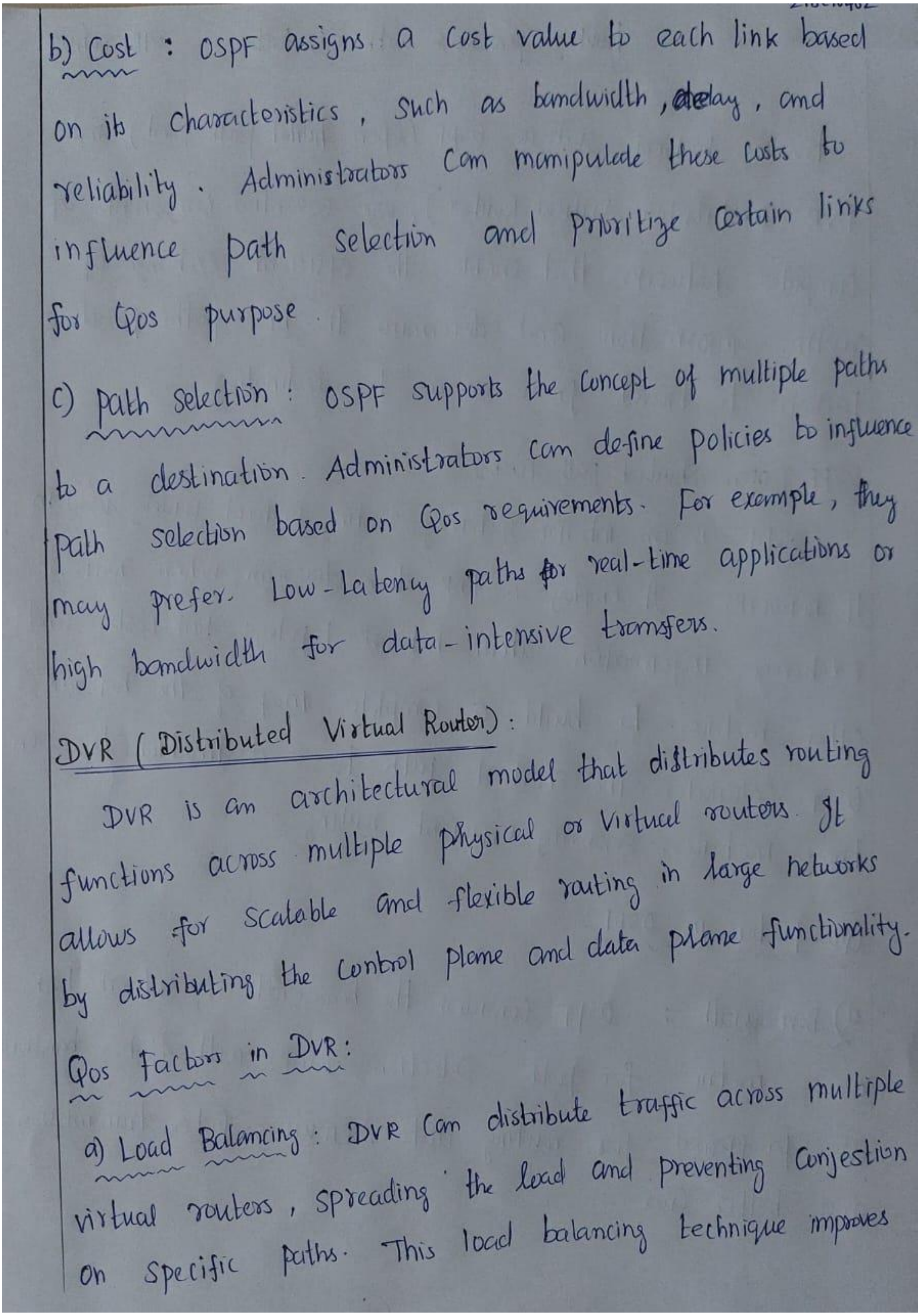
Routing protocols, such as OSPF (open Shortest Path First) and DVR (Distributed Virtual Router), are essential components of computer networks that enables the efficient exchange of routing information and determine the best paths for data packets to travel from source to destination.

OSPF (open Shortest path First):

OSPF is an interior gateway routing protocol widely used in IP networks. It employs a link-state diagram, where routers exchange information about network topology, link costs, and other metrices, to build a complete map of the network. OSPF calculates the shortest path to a destination based on these metrices, ensuring efficient routing.

Qos Factors in OSPF:

a) Bandwidth : OSPF considers the bandwidth of network links as a metric for path selection. Links with higher bandwid are preferred for routing data, ensuring faster transmission and better Qos.



b) Cost : OSPF assigns a cost value to each link based on its characteristics, such as bandwidth, delay, and reliability. Administrators can manipulate these costs to influence path selection and prioritize certain links for Qos purpose.

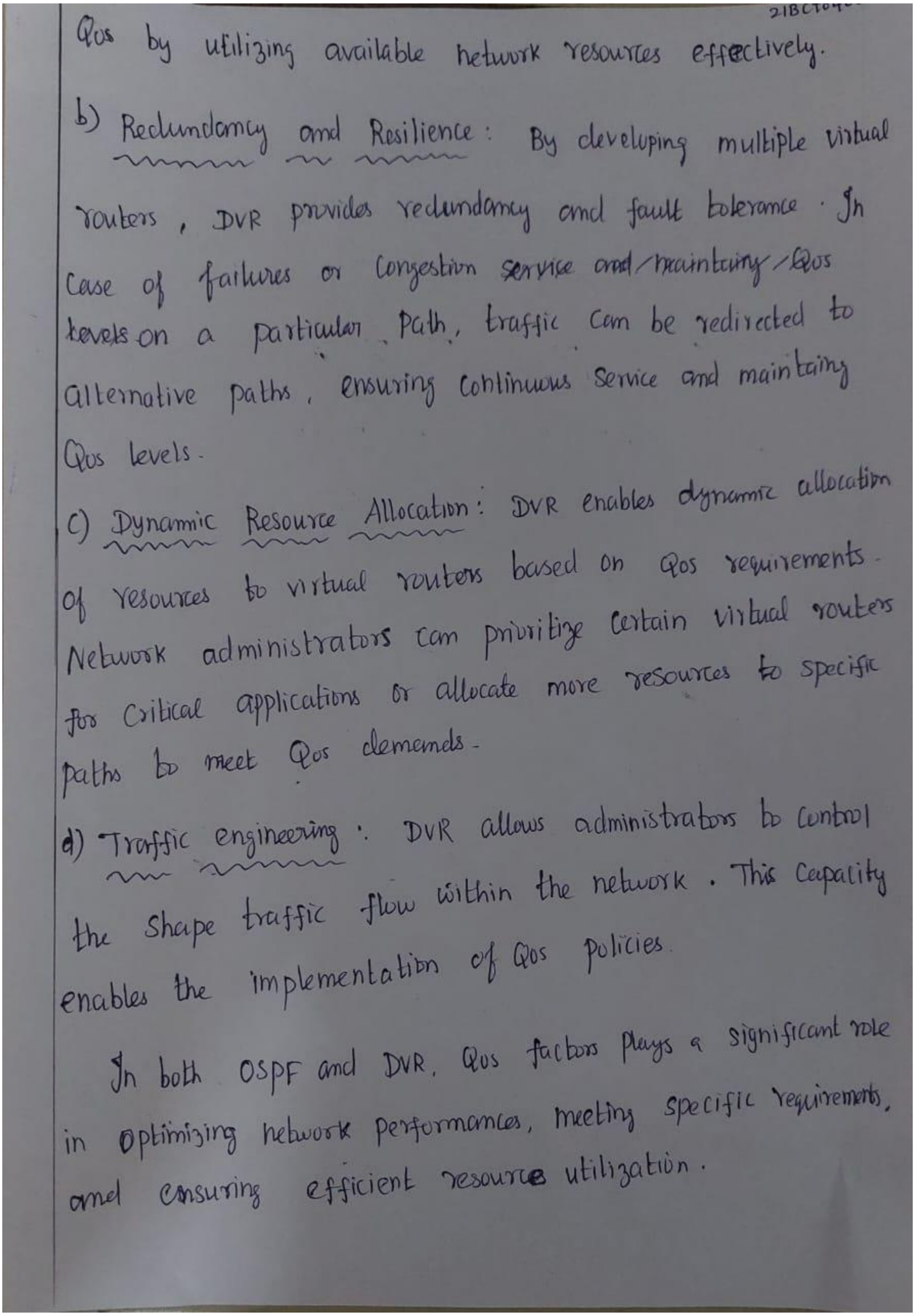
c) Path selection : OSPF supports the concept of multiple paths to a destination. Administrators can define policies to influence path selection based on Qos requirements. For example, they may prefer Low-Latency paths for real-time applications or high bandwidth for data-intensive transfers.

DVR (Distributed Virtual Router) :

DVR is an architectural model that distributes routing functions across multiple physical or virtual routers. It allows for scalable and flexible routing in large networks by distributing the control plane and data plane functionality.

Qos Factors in DVR:

a) Load Balancing : DVR can distribute traffic across multiple virtual routers, spreading the load and preventing conjestion on specific paths. This load balancing technique improves



Qos by utilizing available network resources effectively.

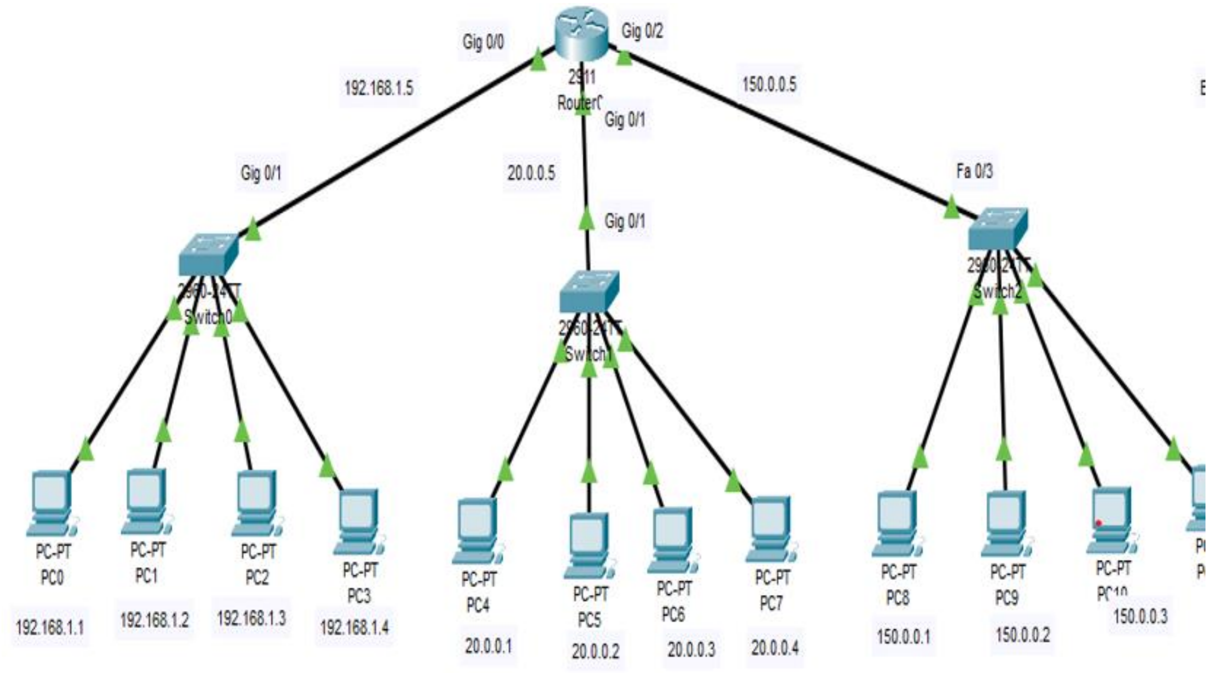
b) Redundancy and Resilience: By developing multiple virtual routers, DVR provides redundancy and fault tolerance. In case of failures or congestion service and/maintaing/Qos levels on a particular Path, traffic can be redirected to alternative paths, ensuring continuous Service and maintaing Qos levels.

c) Dynamic Resource Allocation: DVR enables dynamic allocation of resources to virtual routers based on Qos requirements. Network administrators can prioritize certain virtual routers for critical applications or allocate more resources to specific paths to meet Qos demands.

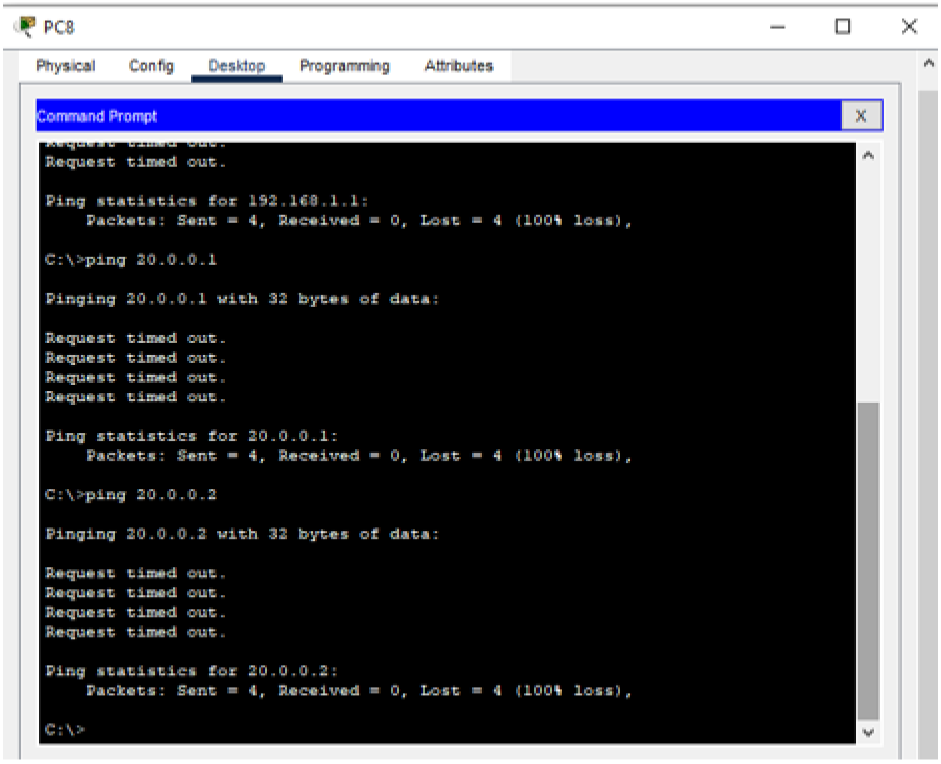
d) Traffic engineering: DVR allows administrators to control the shape traffic flow within the network. This capacity enables the implementation of Qos policies.

In both OSPF and DVR, Qos factors plays a significant role in optimizing network performances, meeting specific requirements, and ensuring efficient resource utilization.

Design:

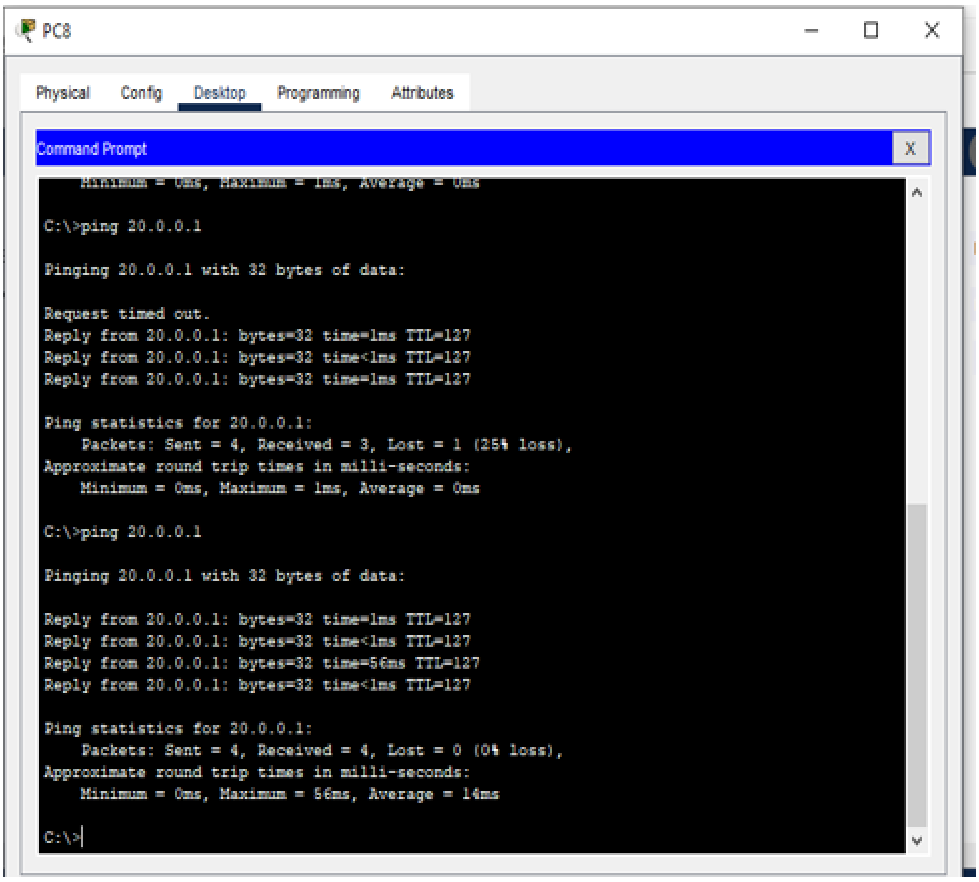


Before Configuration of the router:



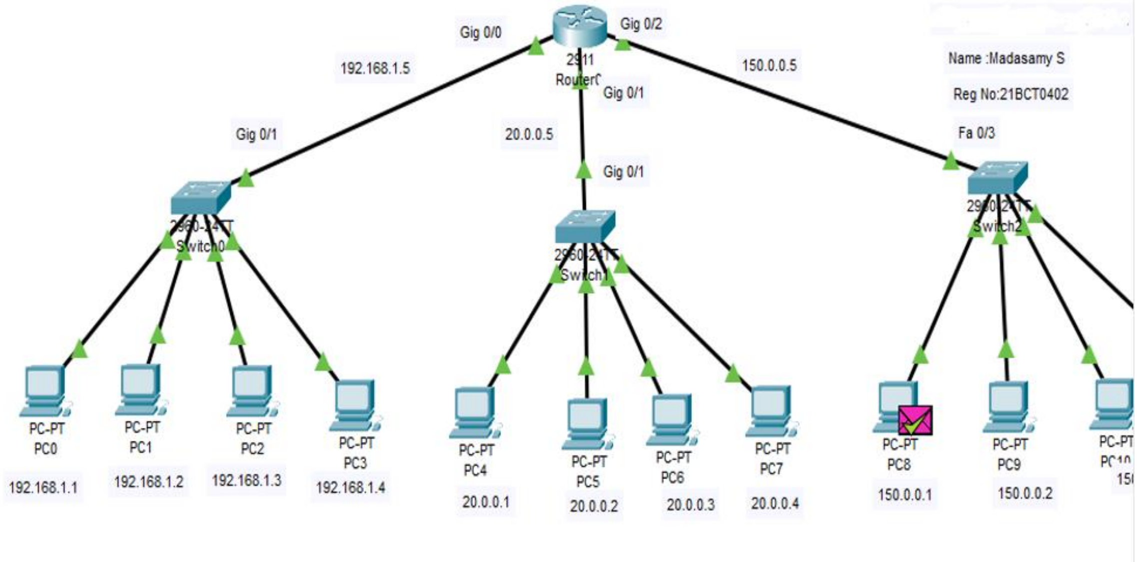
```
Request timed out.
Request timed out.

Ping statistics for 192.168.1.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\>ping 20.0.0.1

Pinging 20.0.0.1 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 20.0.0.1:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\>ping 20.0.0.2

Pinging 20.0.0.2 with 32 bytes of data:

Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 20.0.0.2:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\>
```

After Configuration of the router:

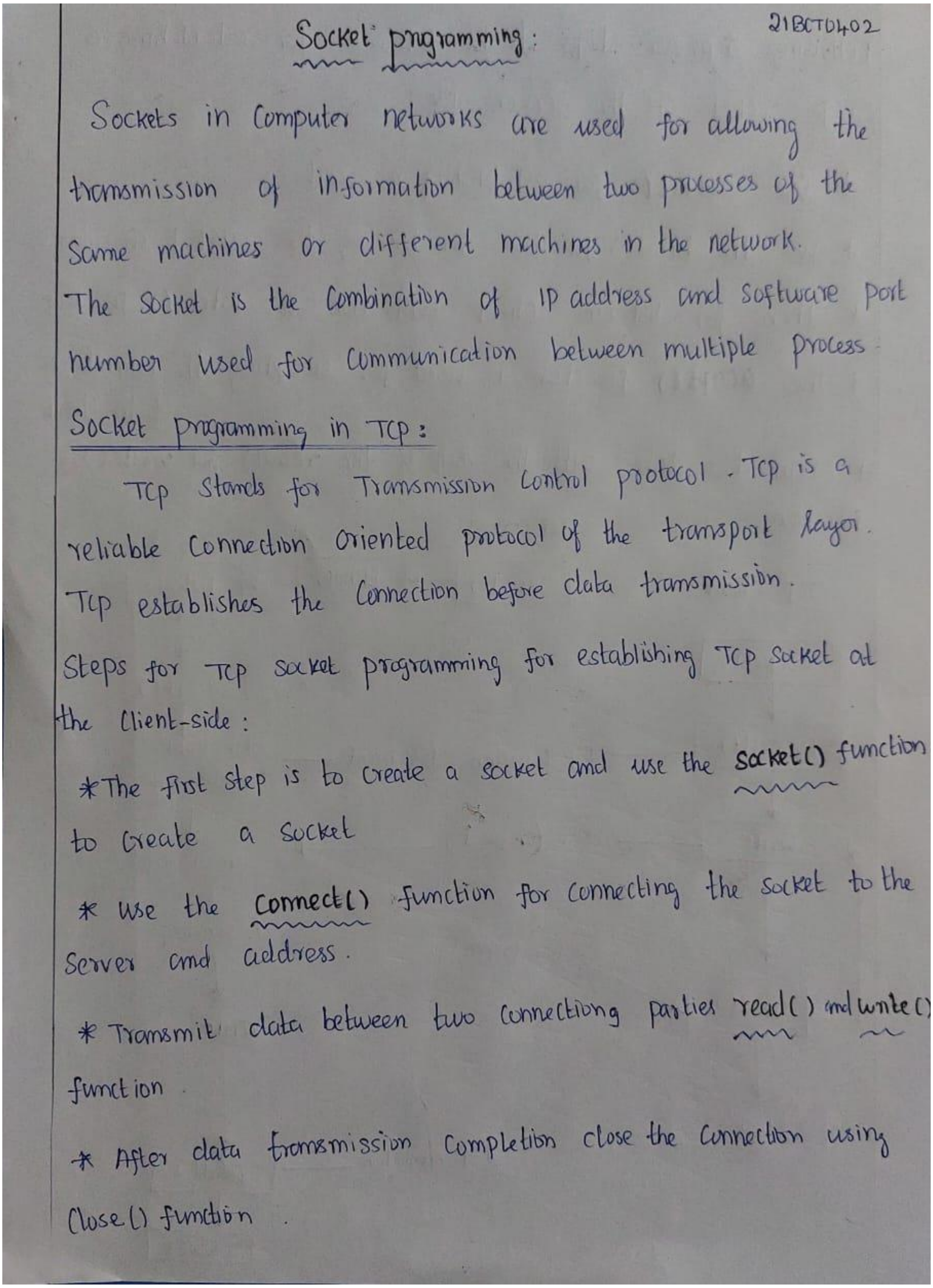


```
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\>ping 20.0.0.1

Pinging 20.0.0.1 with 32 bytes of data:

Request timed out.
Reply from 20.0.0.1: bytes=32 time=1ms TTL=127
Reply from 20.0.0.1: bytes=32 time<1ms TTL=127
Reply from 20.0.0.1: bytes=32 time=1ms TTL=127

Ping statistics for 20.0.0.1:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\>ping 20.0.0.1

Pinging 20.0.0.1 with 32 bytes of data:

Reply from 20.0.0.1: bytes=32 time=1ms TTL=127
Reply from 20.0.0.1: bytes=32 time<1ms TTL=127
Reply from 20.0.0.1: bytes=32 time=56ms TTL=127
Reply from 20.0.0.1: bytes=32 time<1ms TTL=127

Ping statistics for 20.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 56ms, Average = 14ms

C:\>
```

Simulation:


Gig 0/0
Gig 0/2
192.168.1.5
2911
Router(
Gig 0/1
150.0.0.5
Name :Madasamy S
Reg No:21BCT0402
Gig 0/1
20.0.0.5
Fa 0/3
Gig 0/1
2960-24TT
Switch0
2960-24TT
Switch1
2960-24TT
Switch2
PC-PT
PC0
192.168.1.1
PC-PT
PC1
192.168.1.2
PC-PT
PC2
192.168.1.3
PC-PT
PC3
192.168.1.4
PC-PT
PC4
20.0.0.1
PC-PT
PC5
20.0.0.2
PC-PT
PC6
20.0.0.3
PC-PT
PC7
20.0.0.4
PC-PT
PC8
150.0.0.1
PC-PT
PC9
150.0.0.2
PC-PT
PC10
15

EXERCISE 6: Socket programming



21BCT0402

Socket programming :

Sockets in Computer networks are used for allowing the transmission of information between two processes of the Same machines or different machines in the network.

The Socket is the Combination of IP address and Software port number used for communication between multiple process

Socket programming in TCP :

TCP stands for Transmission control protocol. TCP is a reliable Connection oriented protocol of the transport layer. Tcp establishes the Connection before data transmission.

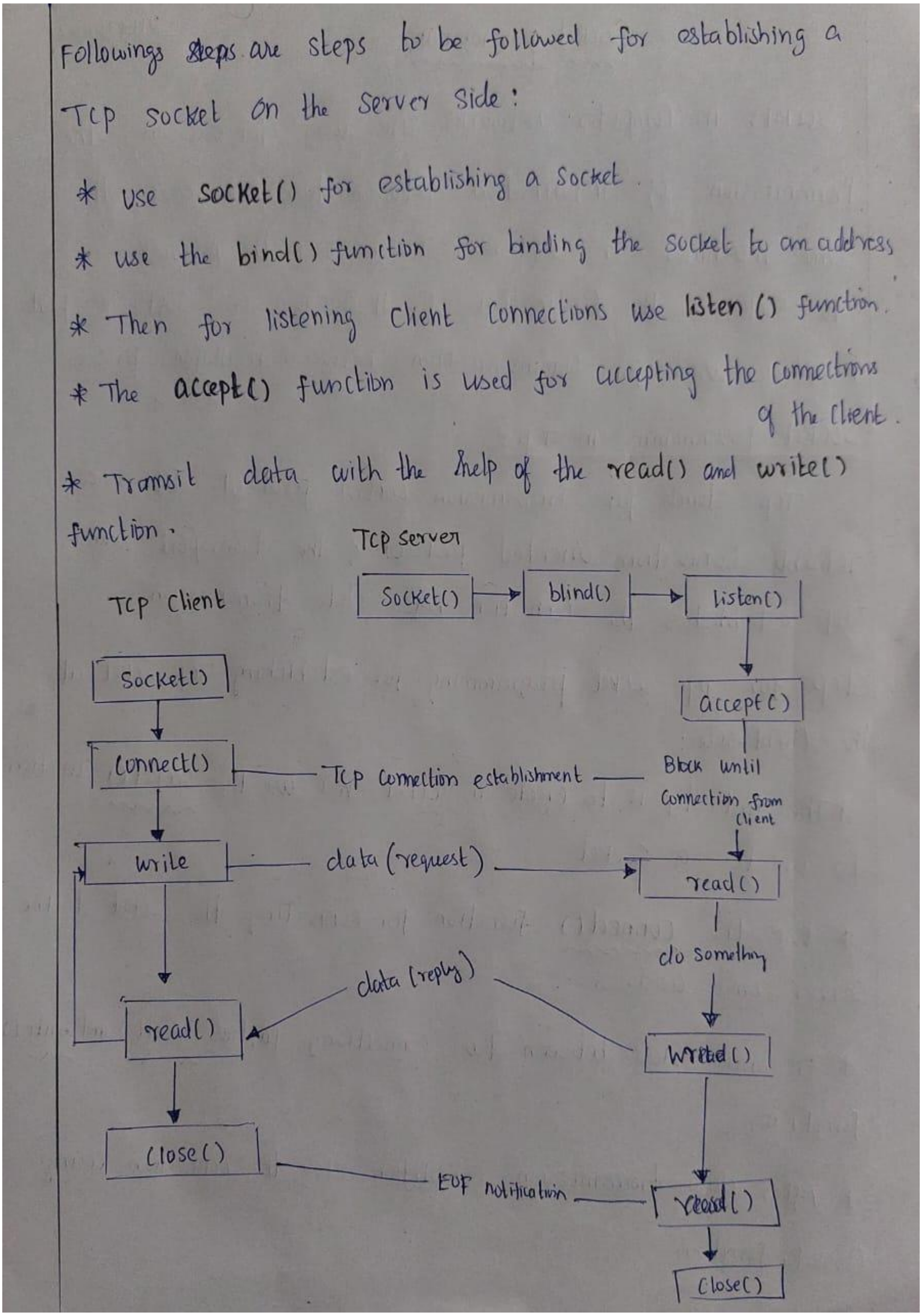
Steps for TCP socket programming for establishing TCP socket at the Client-side :

* The first step is to create a socket and use the socket() function to Create a Socket

* Use the connect() function for connecting the socket to the Server and address.

* Transmit data between two connecting parties read() and write() function.

* After data transmission Completion close the Connection using close() function.



Followings steps are steps to be followed for establishing a TCP socket on the Server Side :

* Use socket() for establishing a Socket.

* use the bind() function for binding the socket to an address

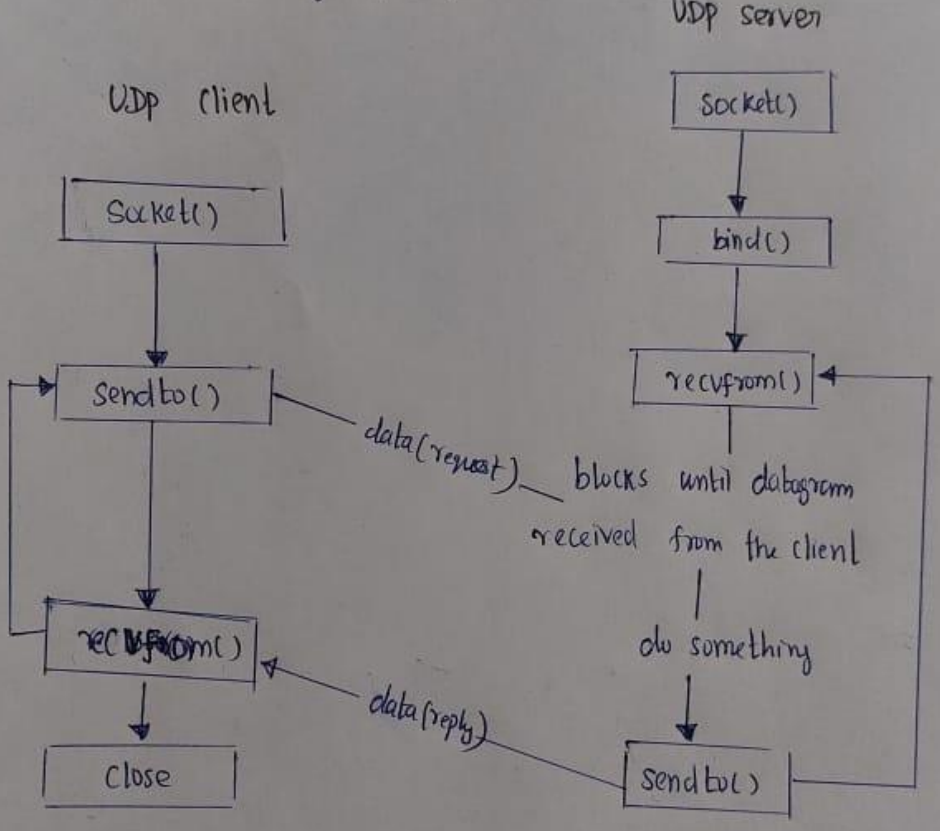
* Then for listening client connections use listen () function.

* The accept() function is used for accepting the connections of the client.

* Tramsit data with the help of the read() and write() function.

Socket programming UDP:

UDP stands for User Datagram protocol. UDP is a connection-less and ureliable protocol of tramport layer. UDP doesnot establishes a connection between two communicating parties before tramsmitting the data.

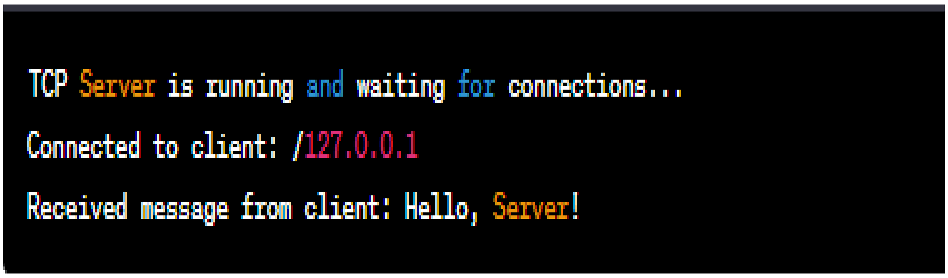


TCP Server:

Code:

```
import java.io.*;

import java.net.*;

public class TCPServer {

    public static void main(String[] args) {

        try {

            ServerSocket serverSocket = new ServerSocket(1234);

            System.out.println("TCP Server is running and waiting for connections...");


            Socket clientSocket = serverSocket.accept();

            System.out.println("Connected to client: " + clientSocket.getInetAddress());


            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);


            String message = in.readLine();

            System.out.println("Received message from client: " + message);


            out.println("Server says: Hi, Client!");


            in.close();

            out.close();

            clientSocket.close();

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

Output:



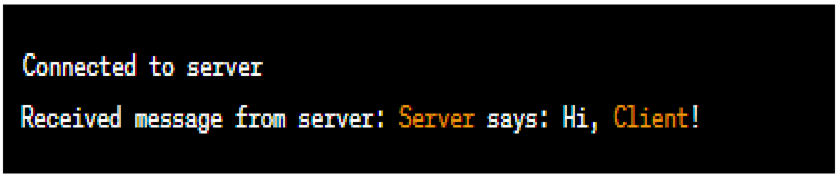
```
TCP Server is running and waiting for connections...
Connected to client: /127.0.0.1
Received message from client: Hello, Server!
```

TCP Client:

Code:

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

public class TCPClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 1234);
            System.out.println("Connected to server");

            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            out.println("Hello, Server!");


            String message = in.readLine();

            System.out.println("Received message from server: " + message);


            in.close();

            out.close();

            socket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

Output:



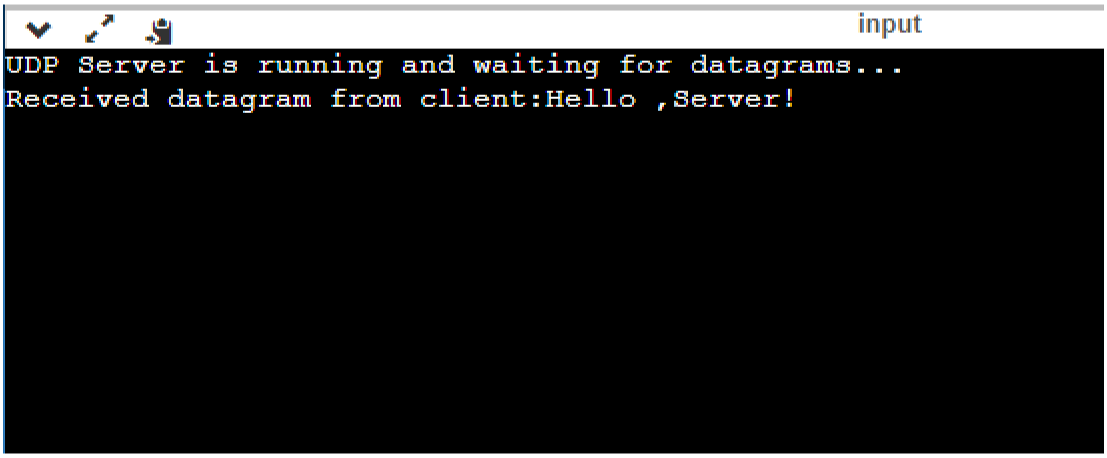
```
Connected to server
Received message from server: Server says: Hi, Client!
```

UDP Server:

Code:

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

public class UDPServer {
    public static void main(String[] args) {
        try {
            DatagramSocket serverSocket = new DatagramSocket(1234);
            System.out.println("UDP Server is running and waiting for datagrams...");

            byte[] receiveData = new byte[1024];
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);

            serverSocket.receive(receivePacket);
            System.out.println("Received datagram from client: " + new String(receivePacket.getData()));

            InetAddress clientAddress = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();
            String response = "Server says: Hi, Client!";
            byte[] sendData = response.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
clientAddress, clientPort);

            serverSocket.send(sendPacket);


            serverSocket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

Output:



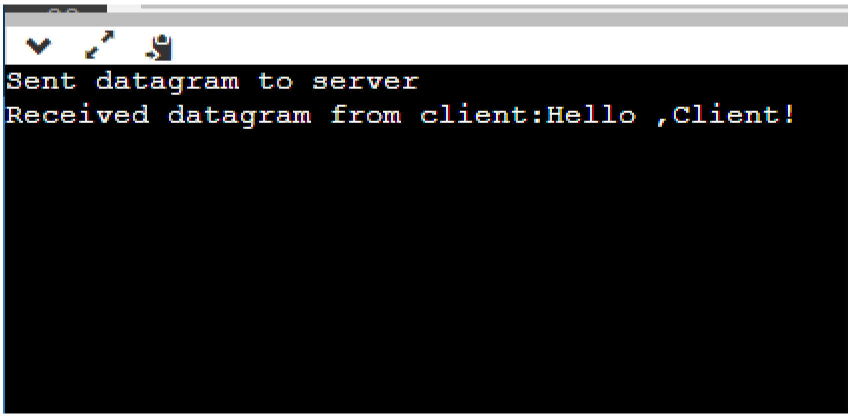
```
UDP Server is running and waiting for datagrams...
Received datagram from client:Hello ,Server!
```

UDP Client:

Code:

```
import java.io.*;

import java.net.*;


public class UDPClient {

    public static void main(String[] args) {

        try {

            DatagramSocket socket = new DatagramSocket();


            InetAddress serverAddress = InetAddress.getByName("localhost");

            int serverPort = 1234;


            String message = "Hello, Server!";

            byte[] sendData = message.getBytes();


            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
serverAddress, serverPort);

            socket.send(sendPacket);

            System.out.println("Sent datagram to server");
            byte[] receiveData = new byte[1024];

            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);


            socket.receive(receivePacket);

            System.out.println("Received datagram from server: " + new
String(receivePacket.getData()));


            socket.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

Output:



```
Sent datagram to server
Received datagram from client:Hello ,Client!
```