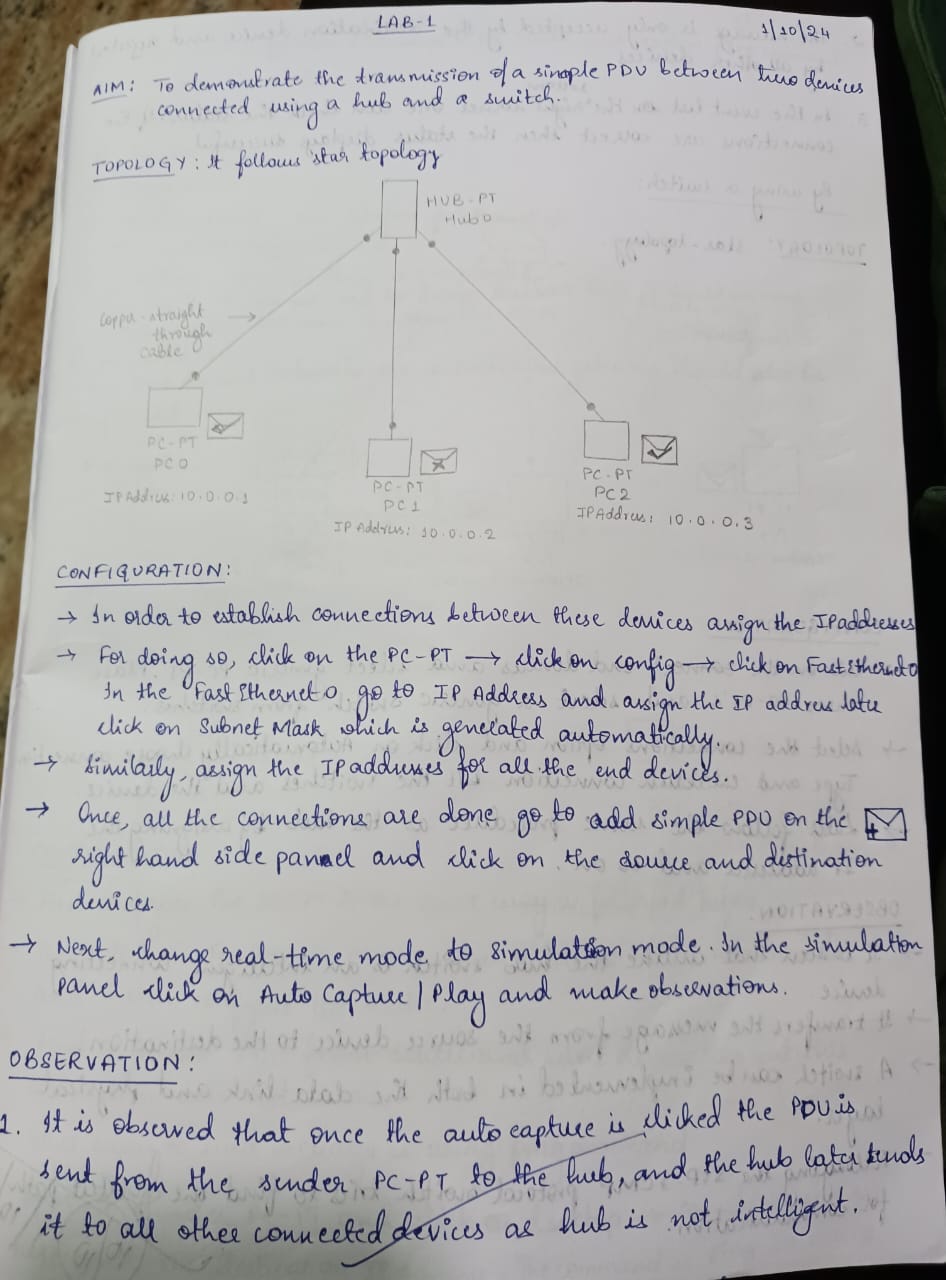
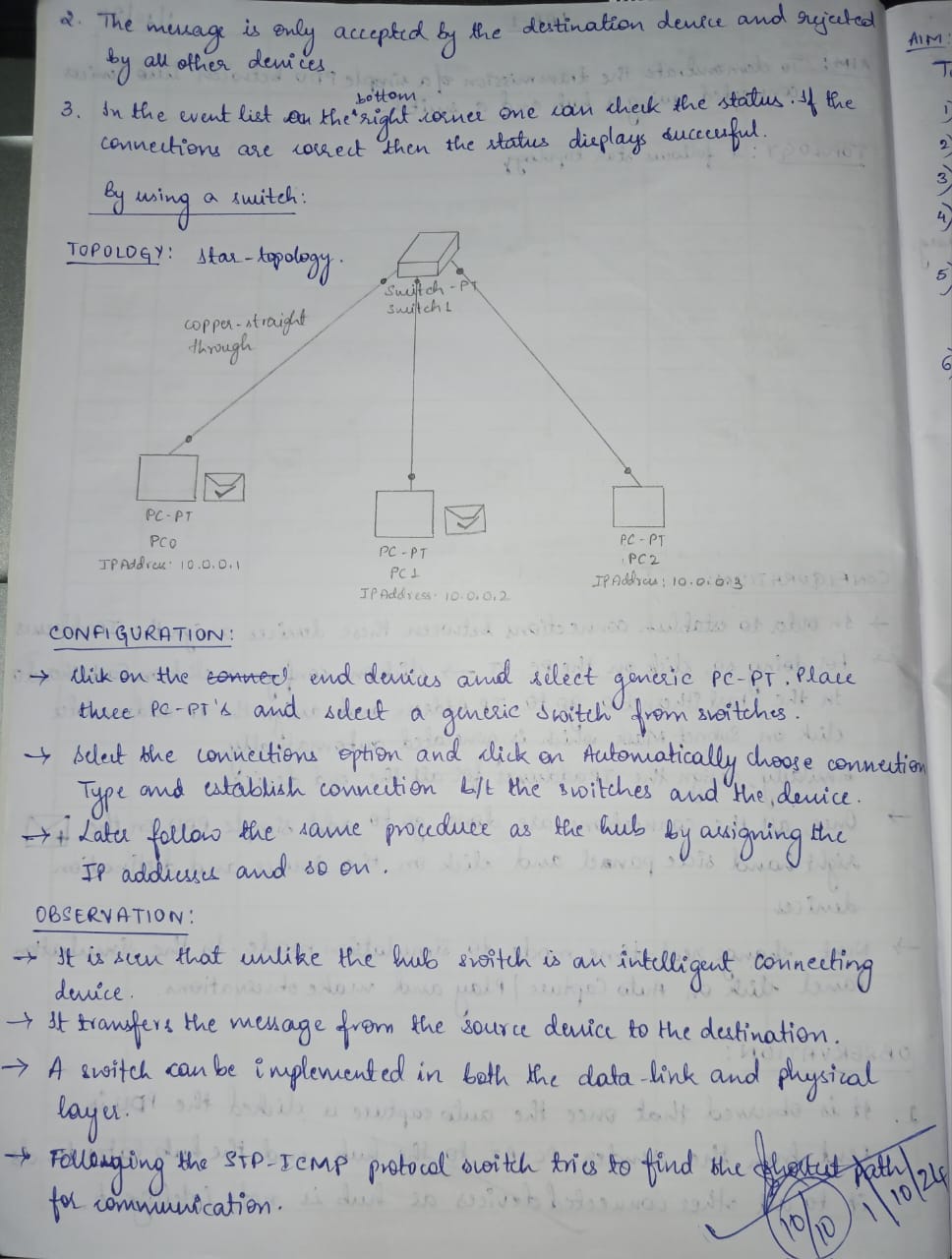
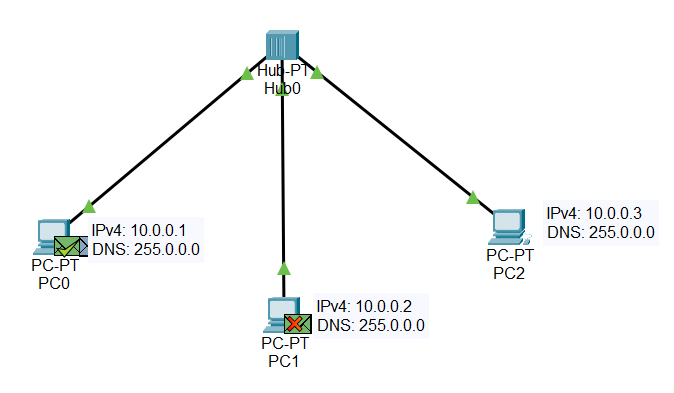
**Computer Networks**

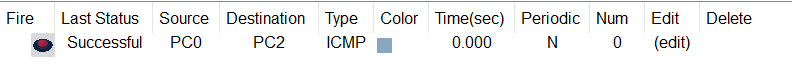
**LABORATORY PROGRAM – 1**

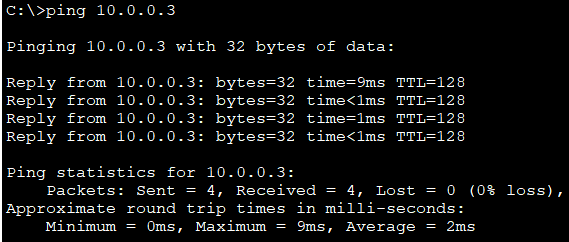
Create a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices and demonstrate ping message.







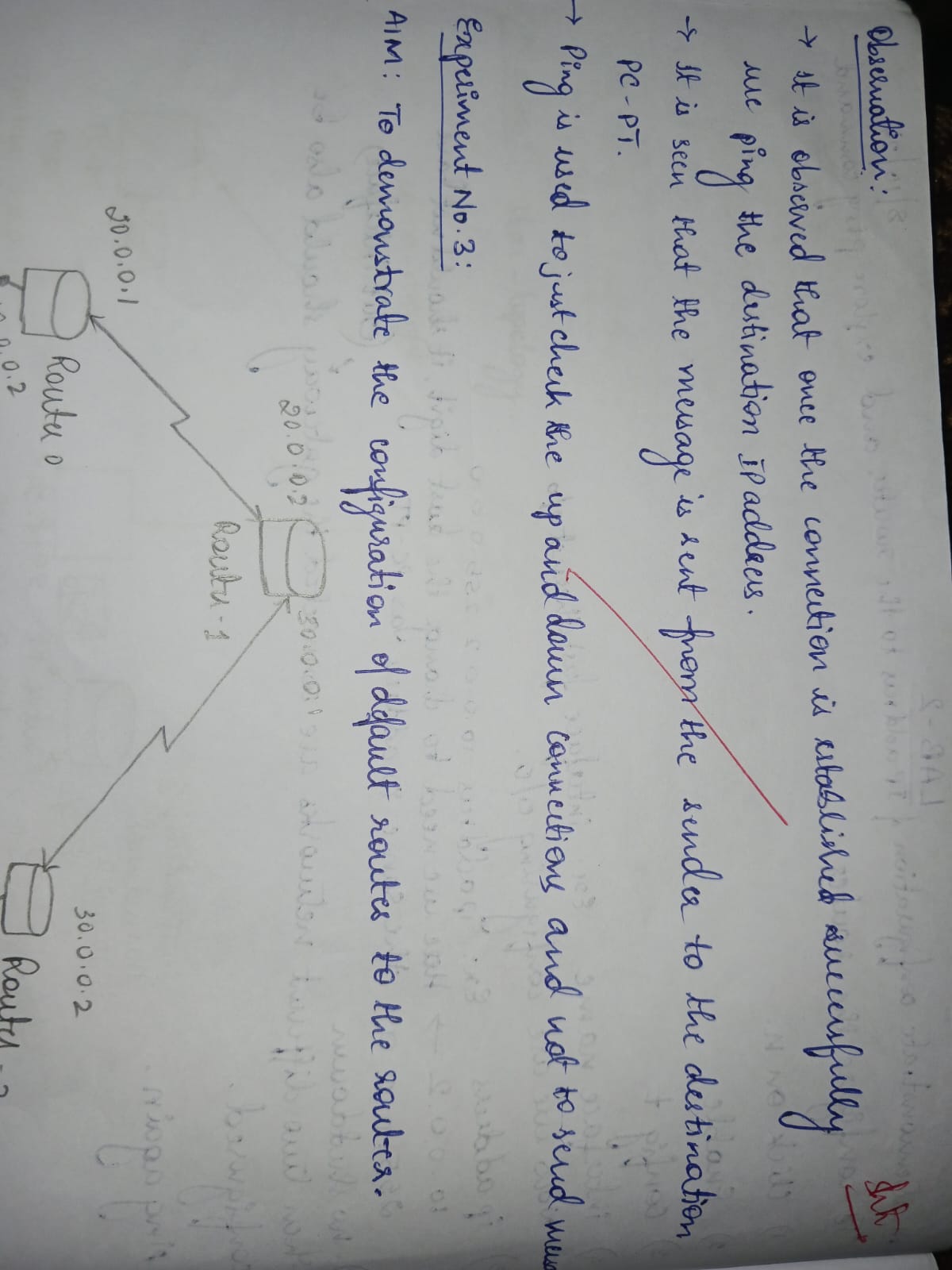


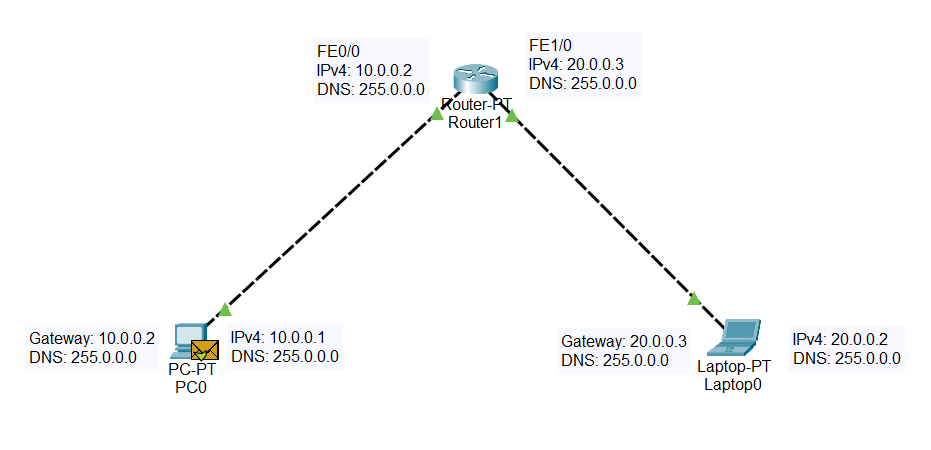
****

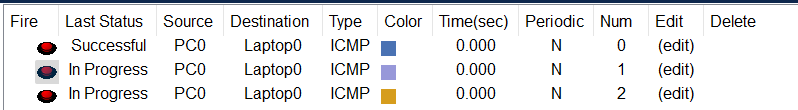
**LABORATORY PROGRAM – 2**

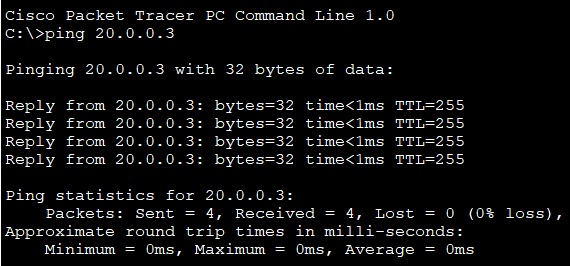
Configure IP address to routers in packet tracer. Explore the following messages: ping responses, destination unreachable, request timed out, reply.









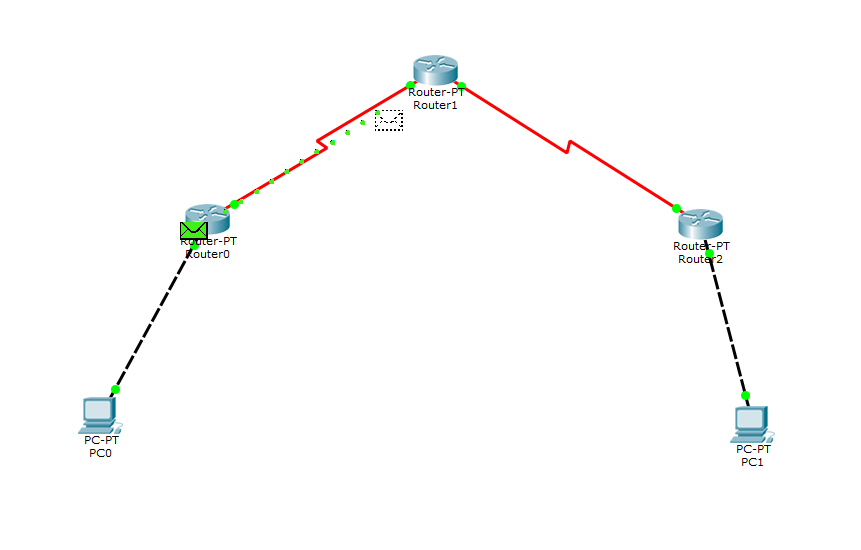
****

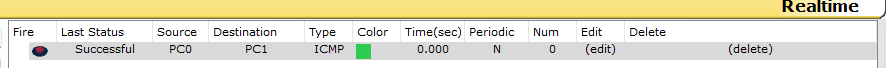
**LABORATORY PROGRAM – 3**

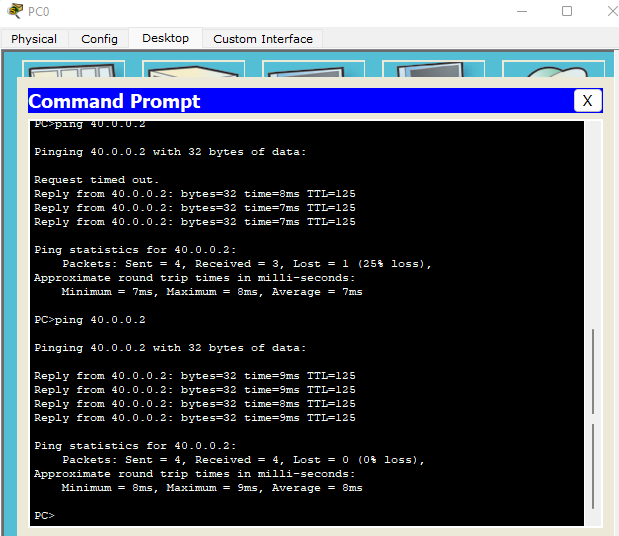
Configure default route, static route to the Router.





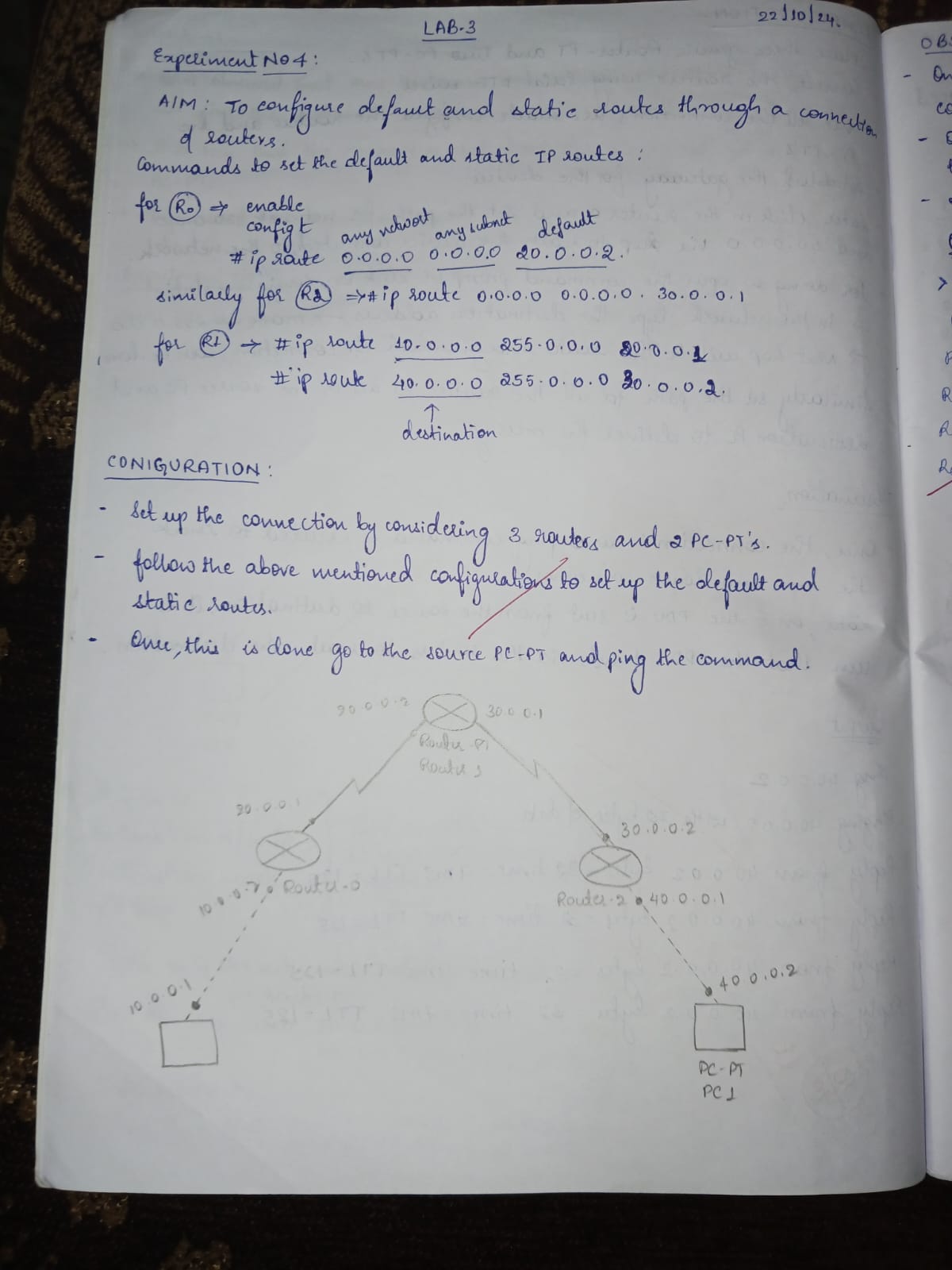
****

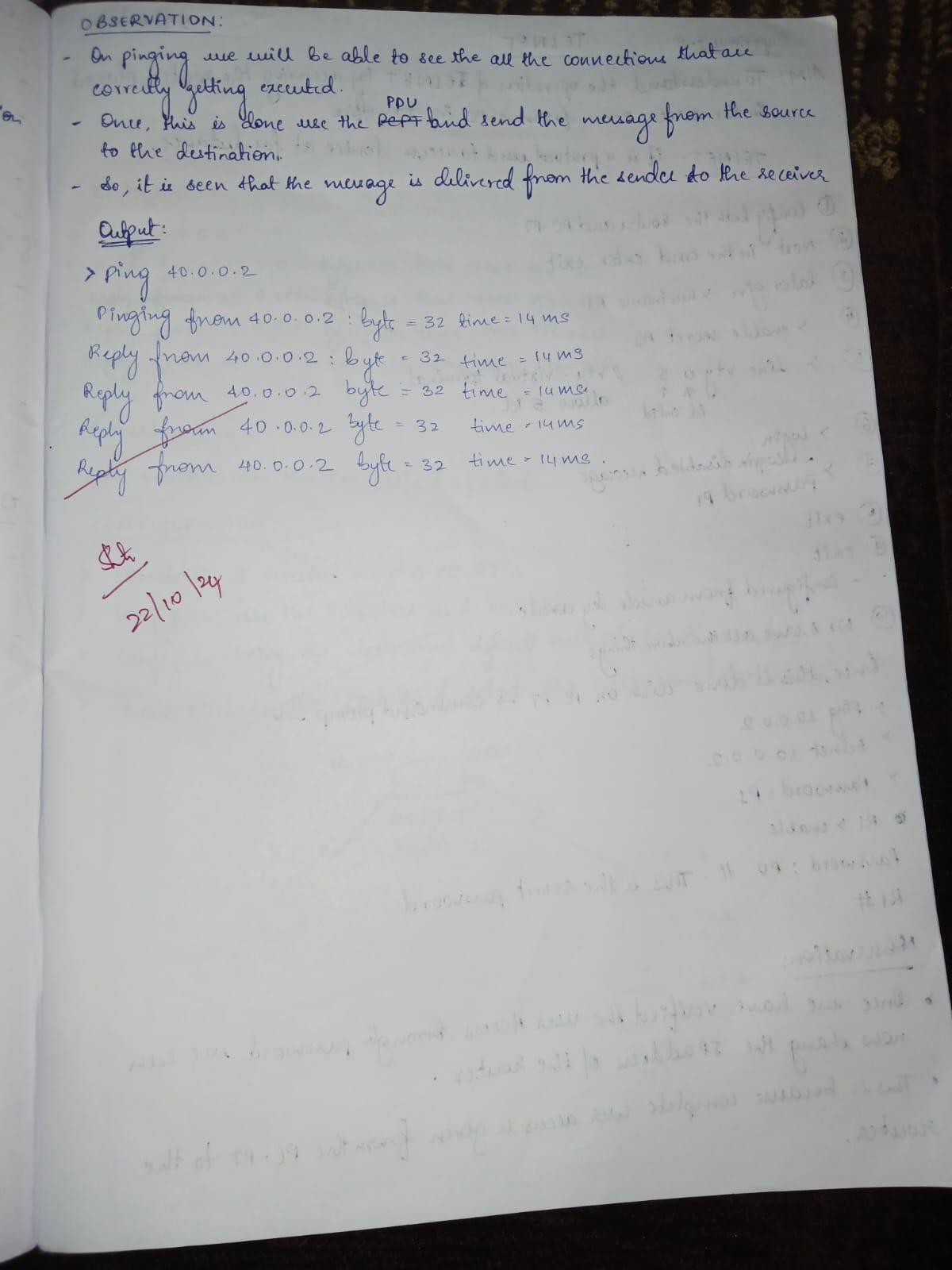
****

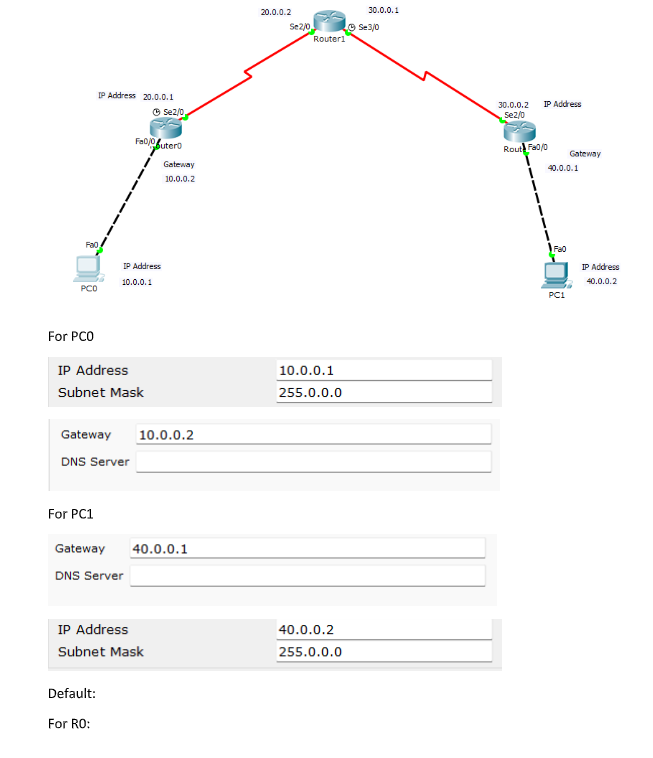
****

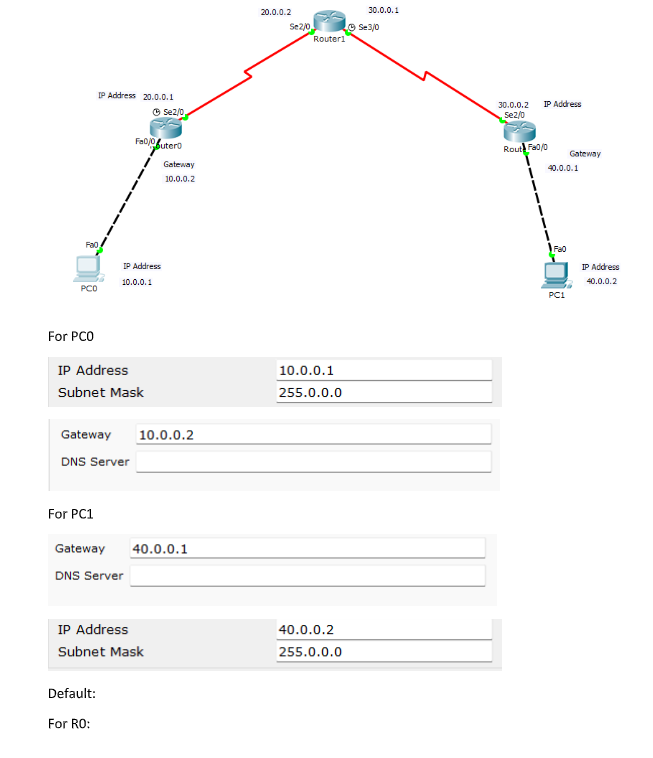
**LABORATORY PROGRAM-4**

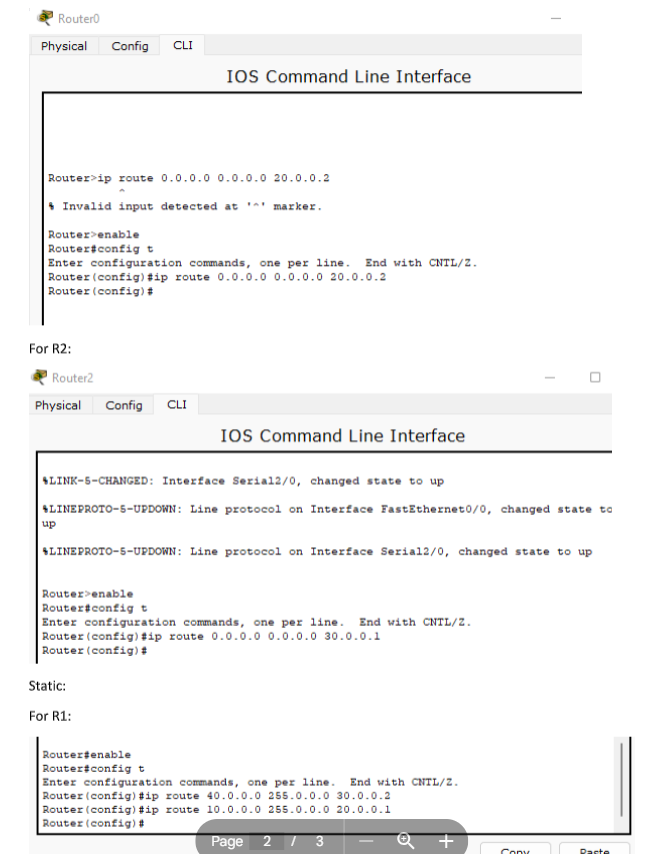
To configure default and static routers.

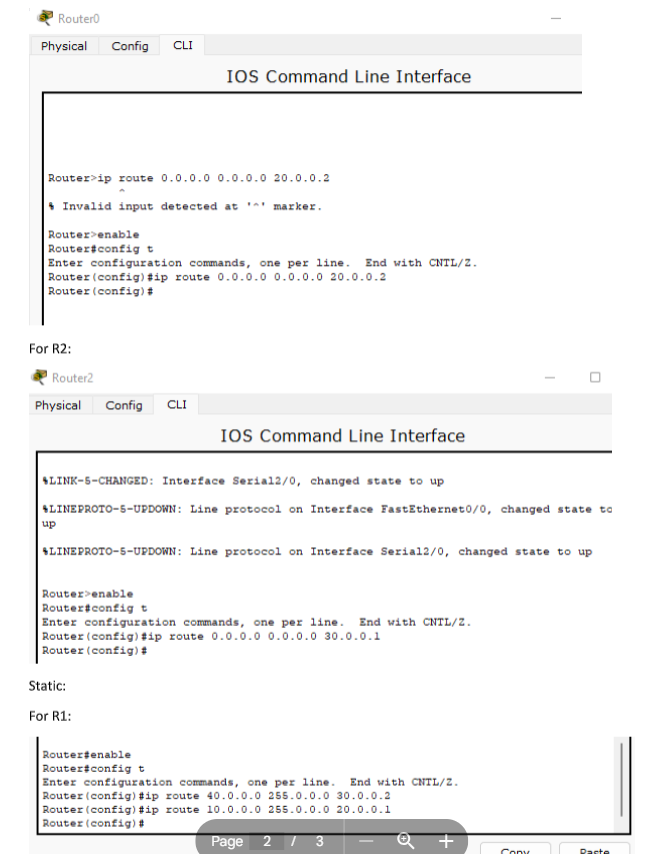


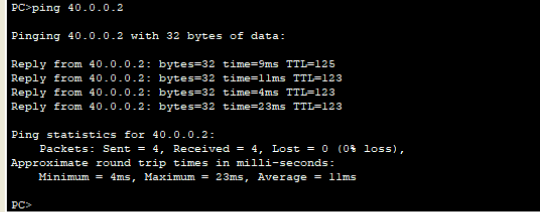






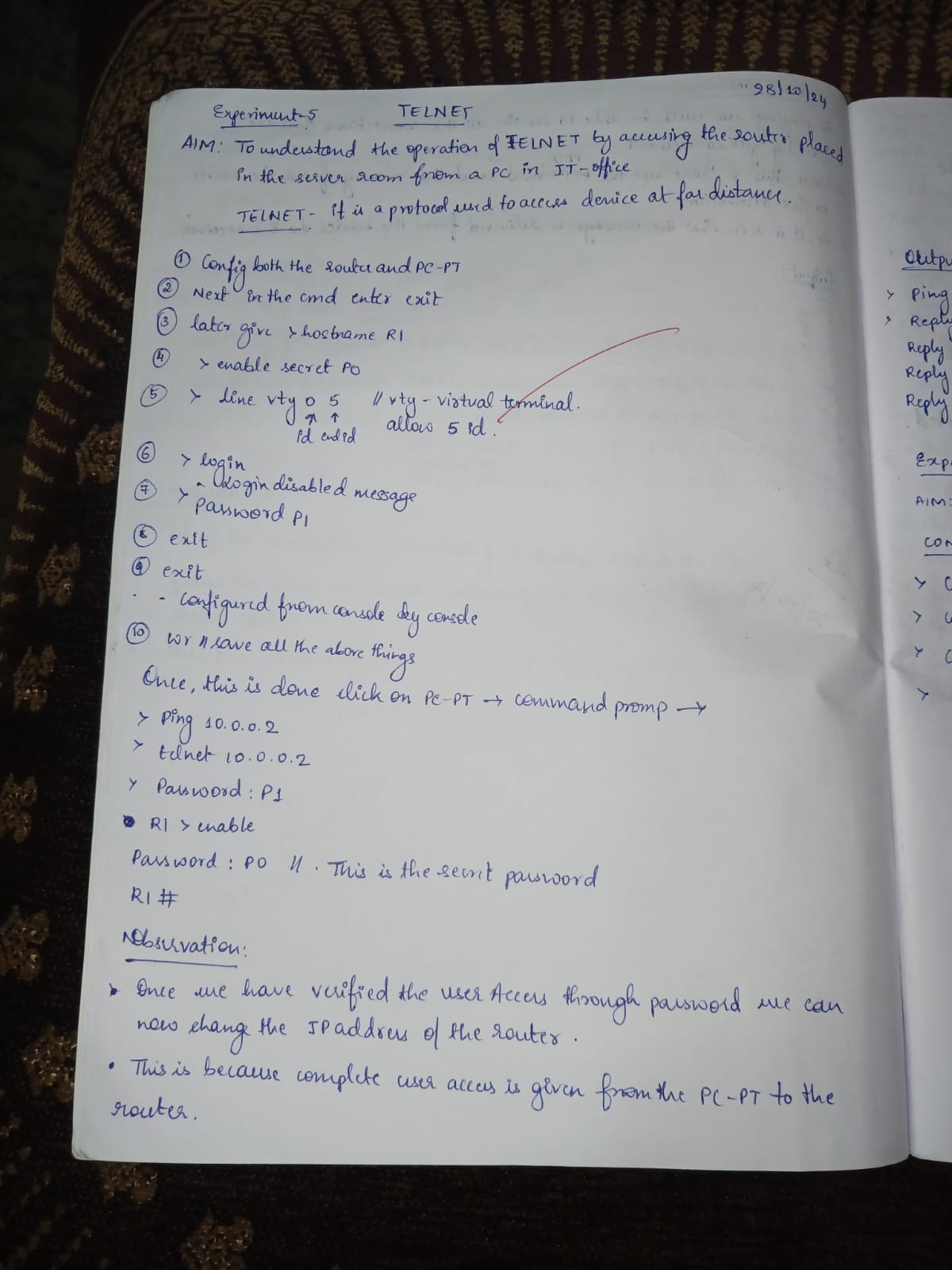


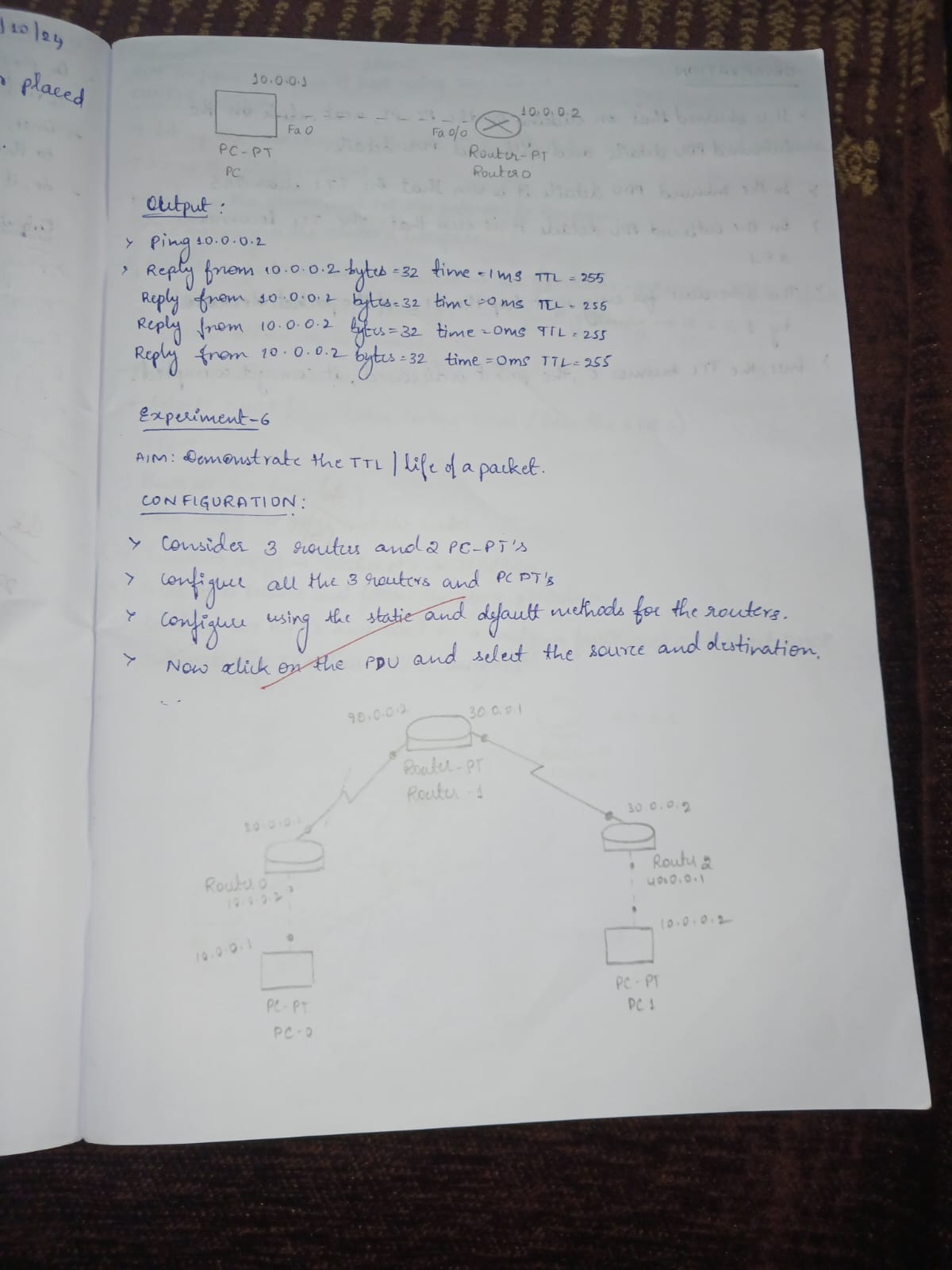


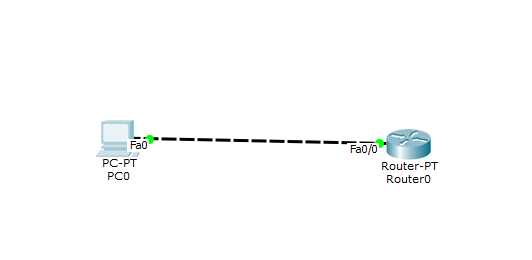


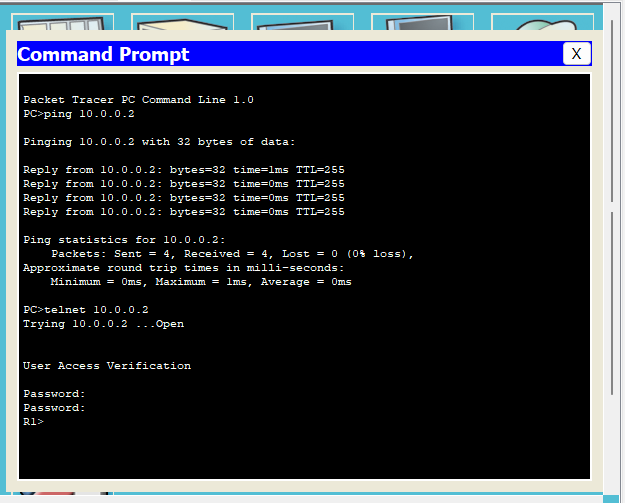
**LABORATORY PROGRAM-5**

To understand the operations of TELNET



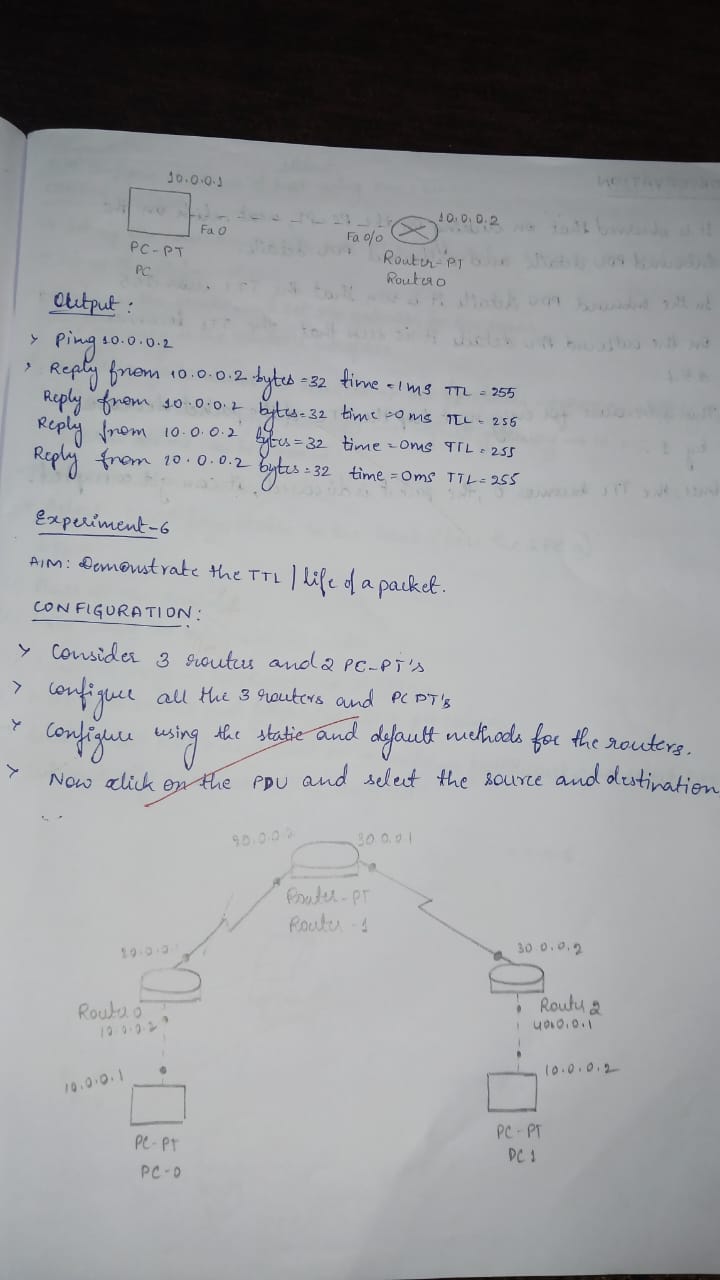


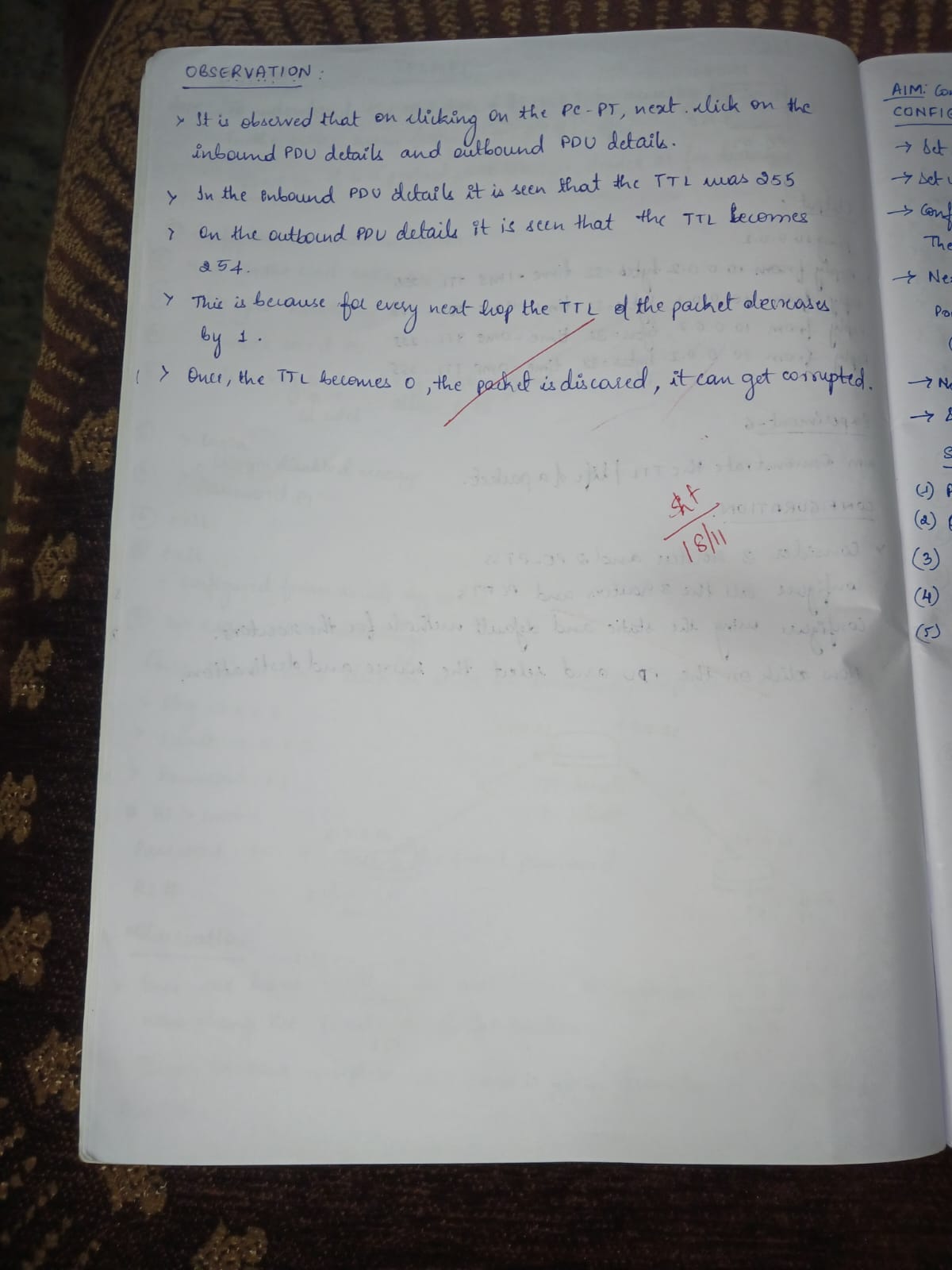
****

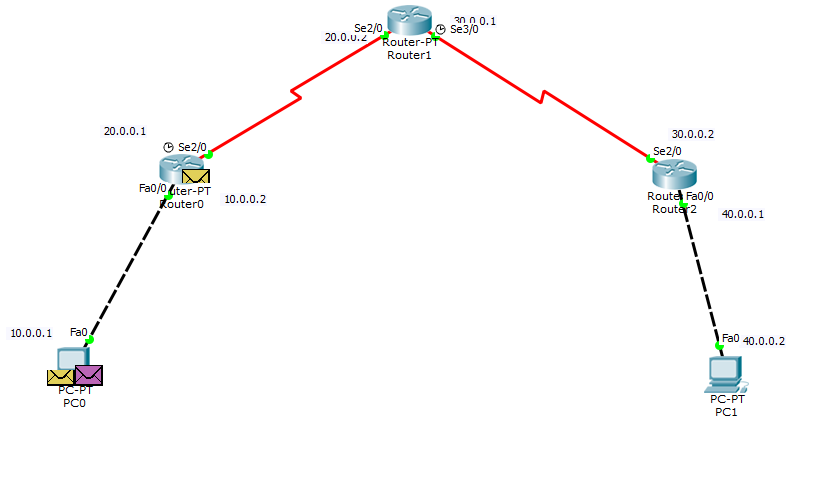
****

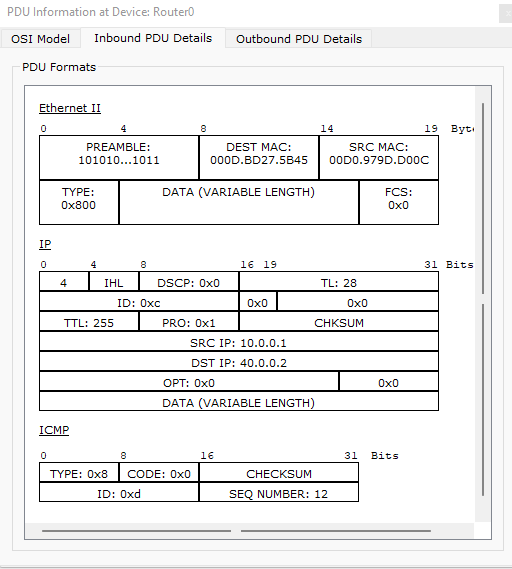
**LABORATORY PROGRAM-6**

Demonstrate the TTL/Life of a packet.

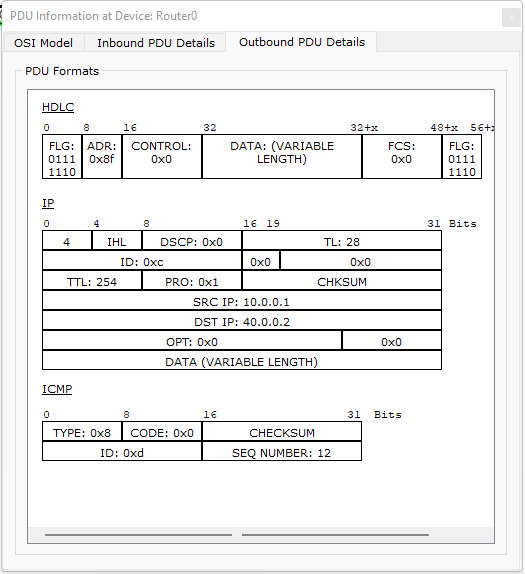






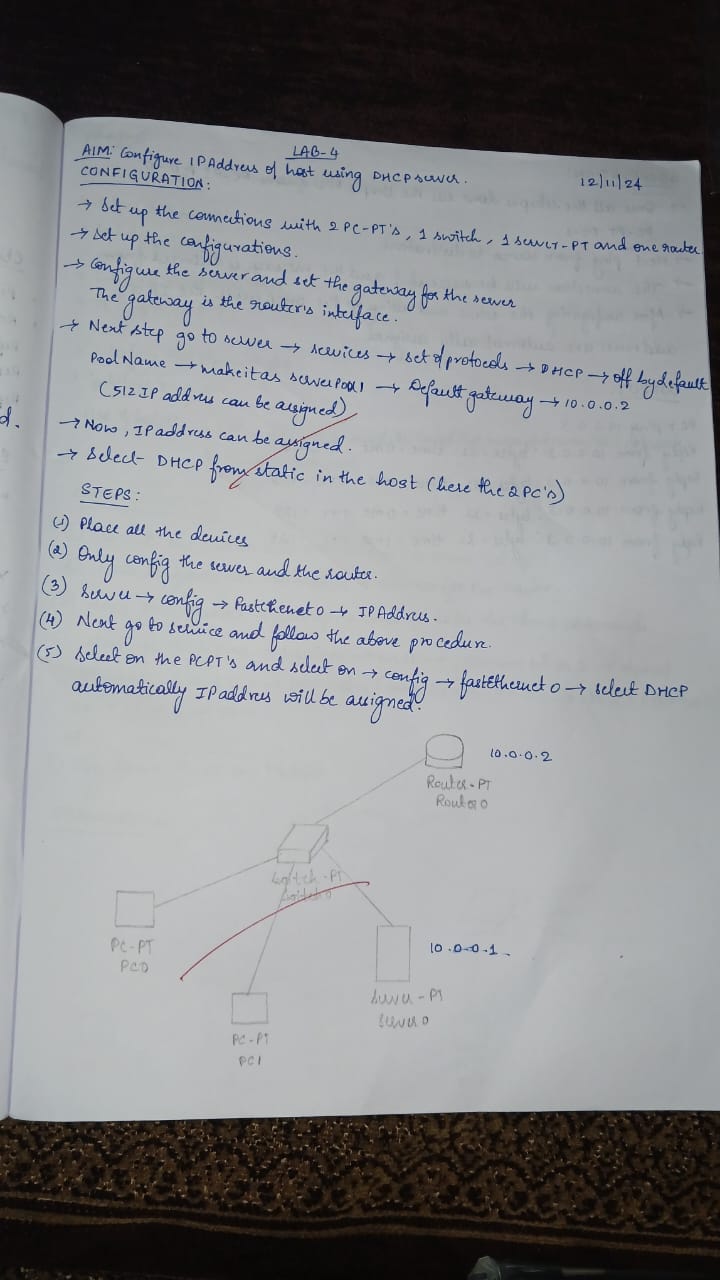


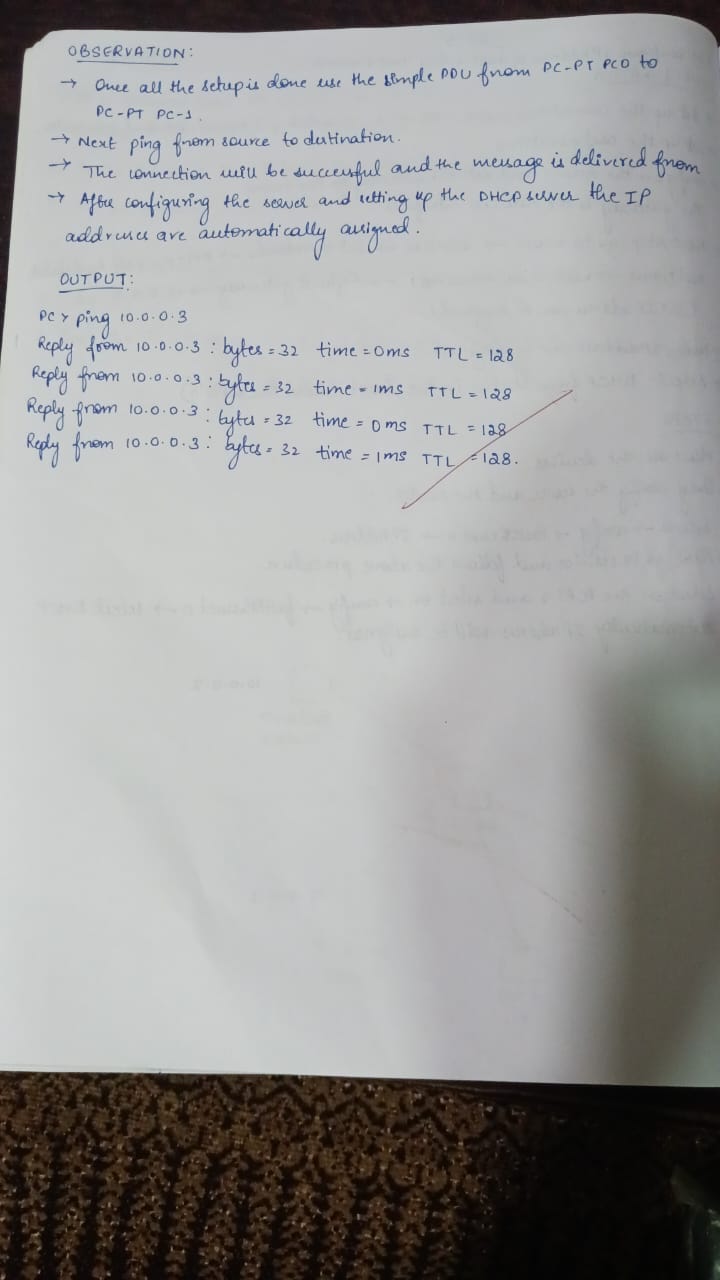


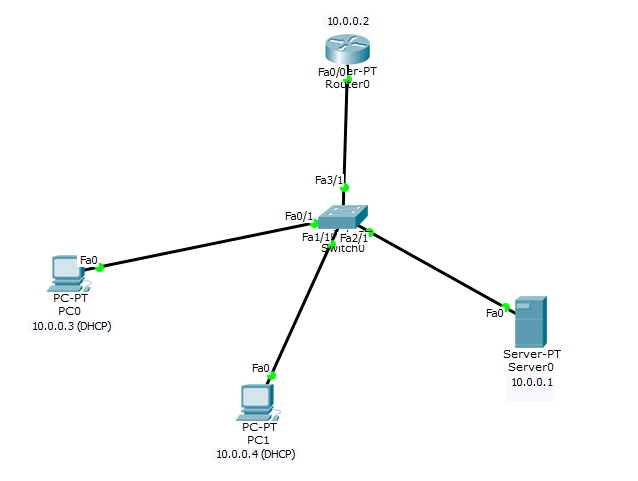


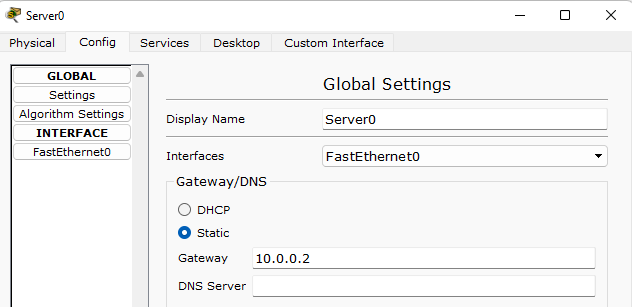
**LABORATORY PROGRAM-7**

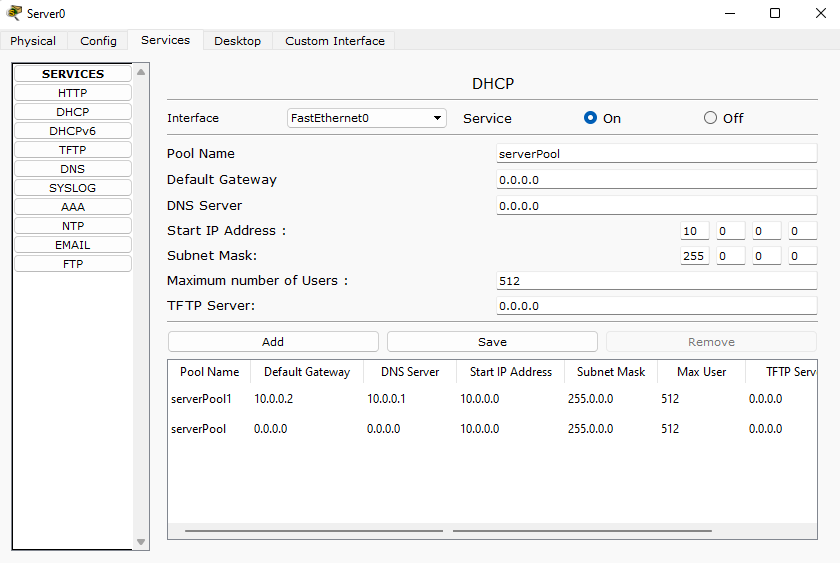
To configure IP Addresses of the host using DHCP server present within the LAN.

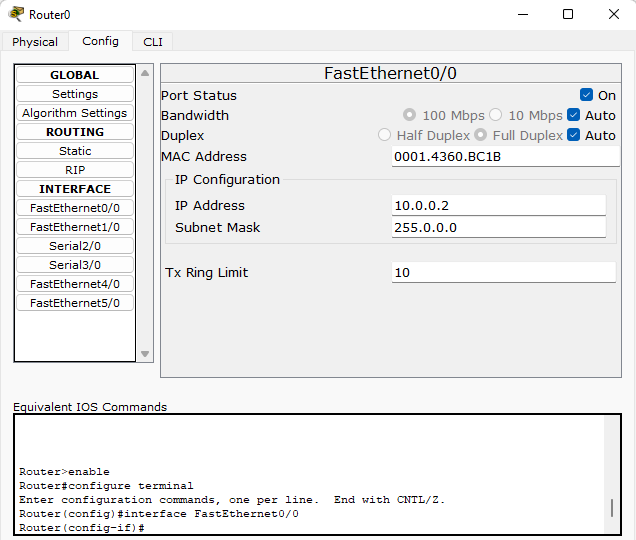




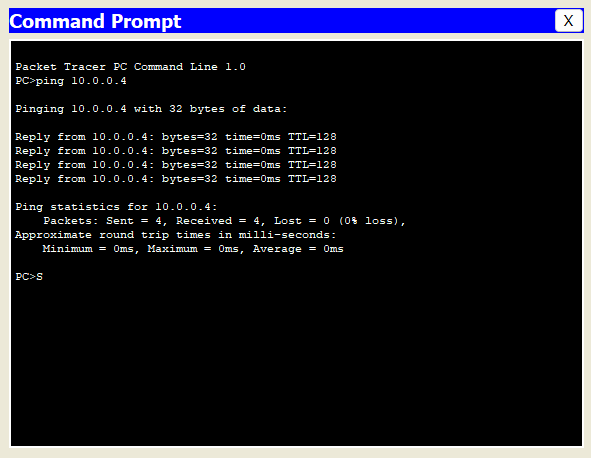






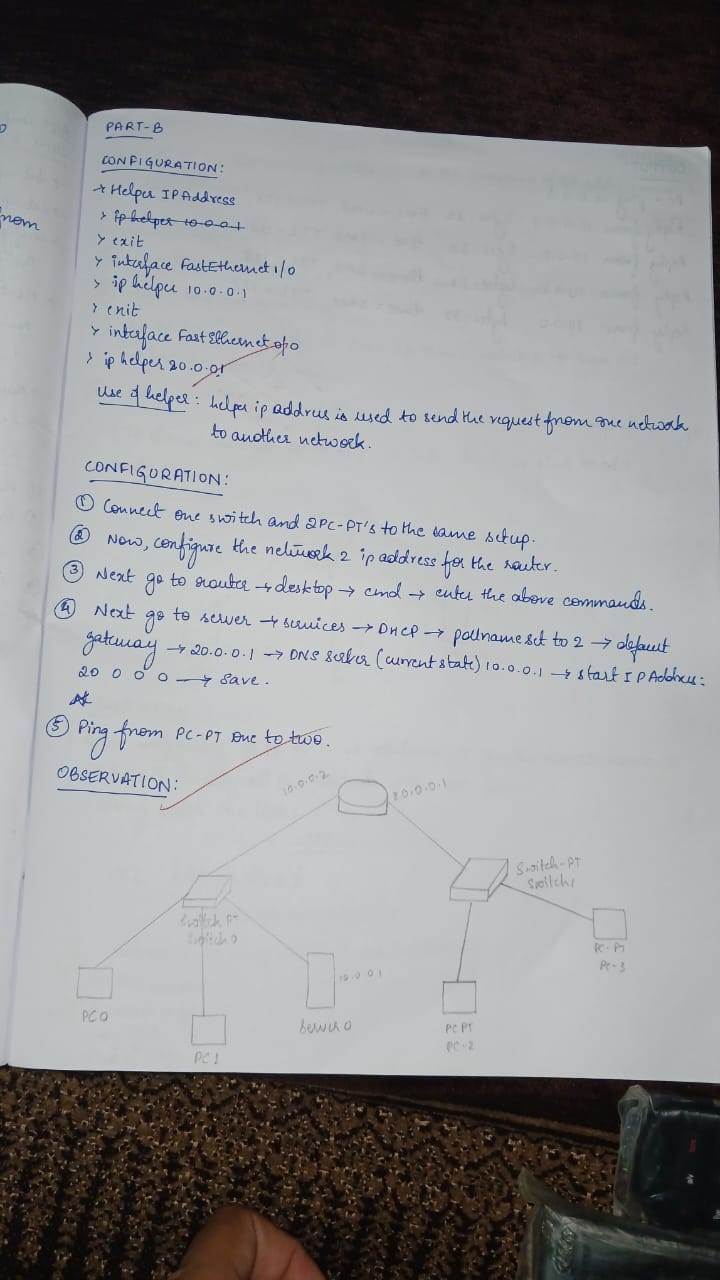


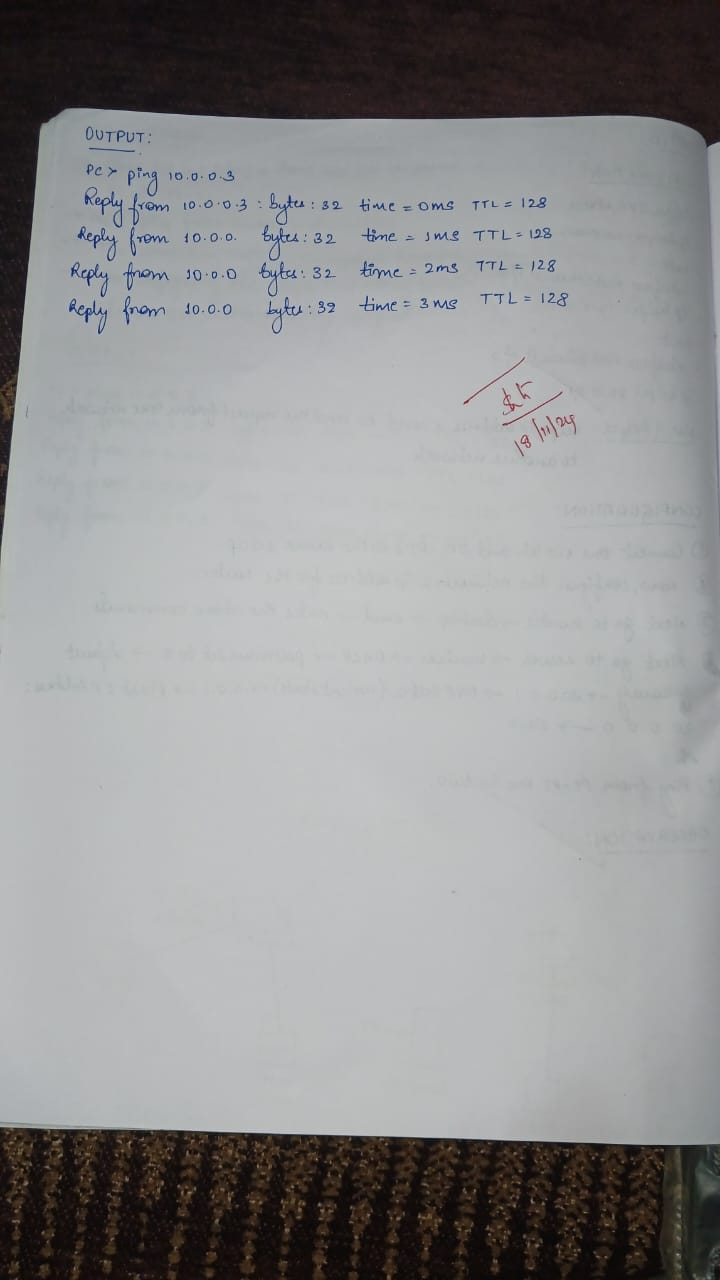


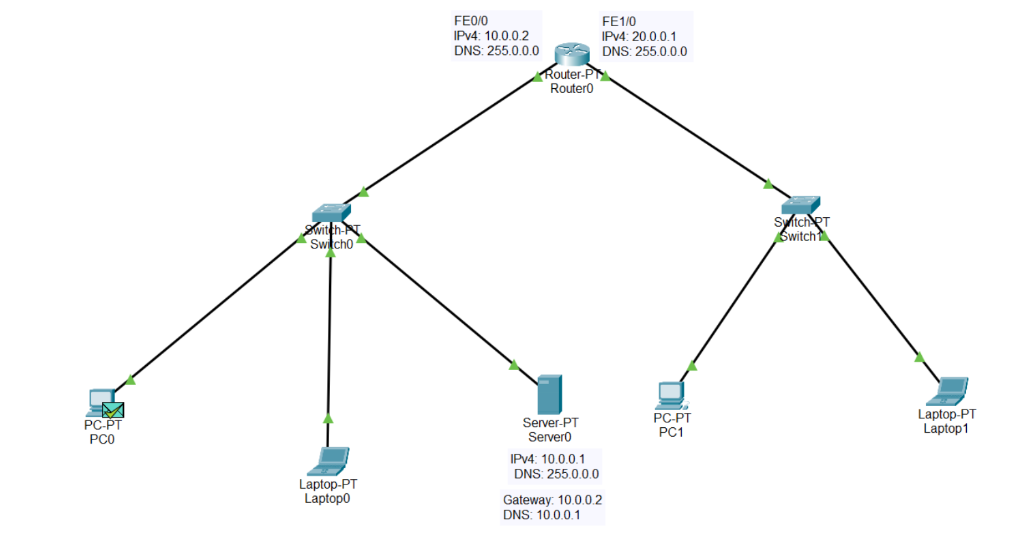


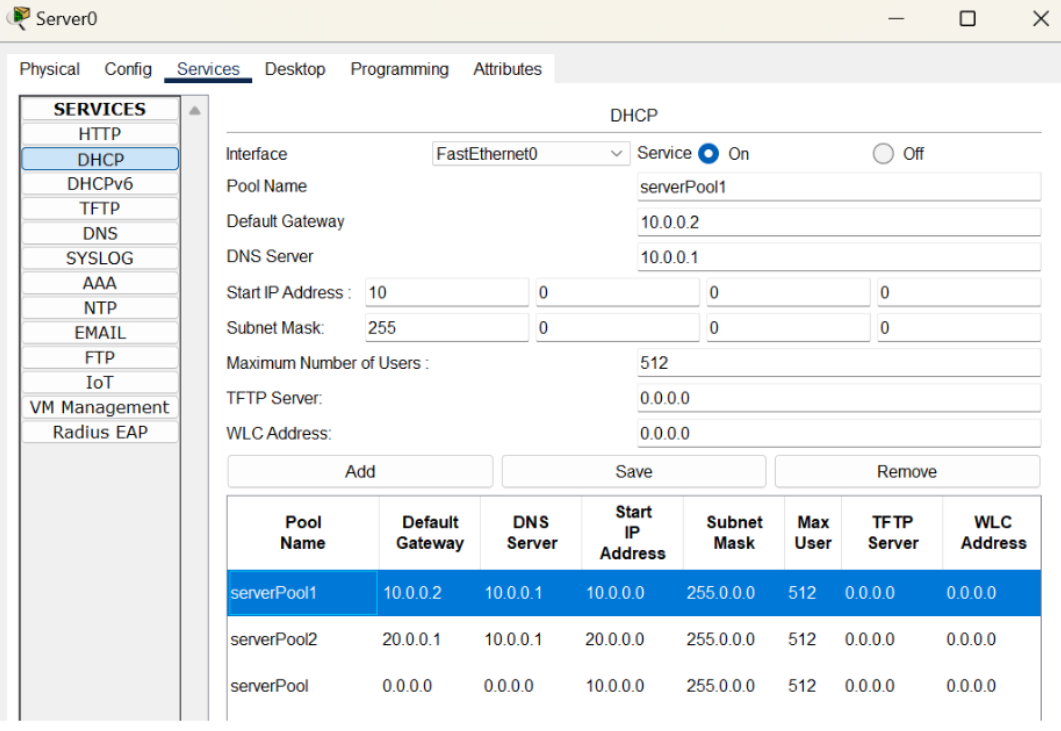
**LABORATORY PROGRAM-7**

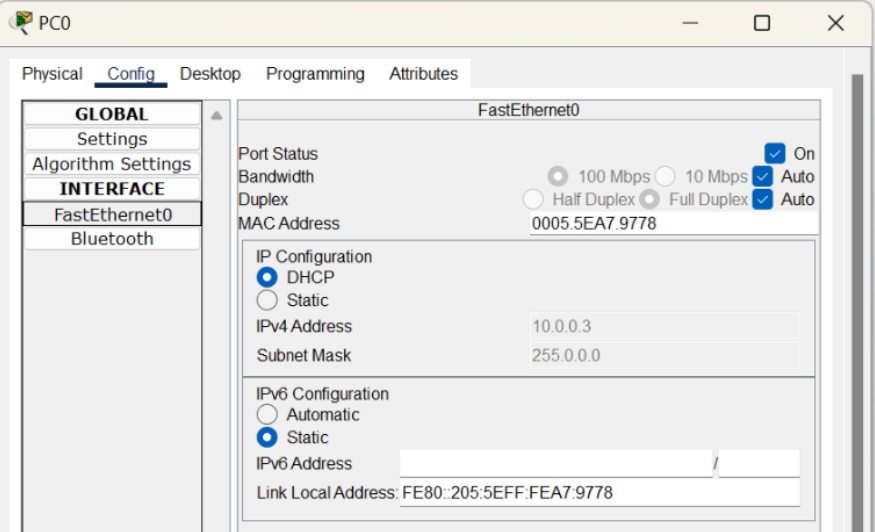
To configure IP addresses of the host using DHCP server outside a LAN.

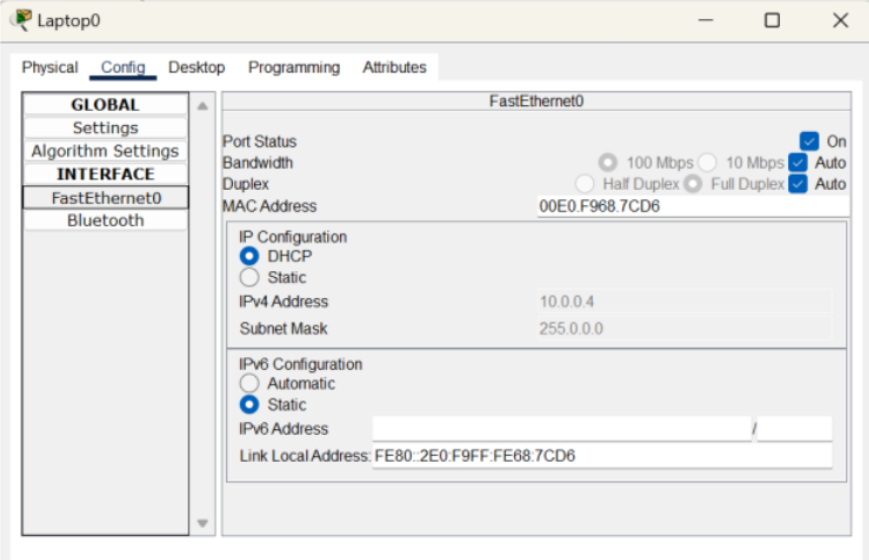


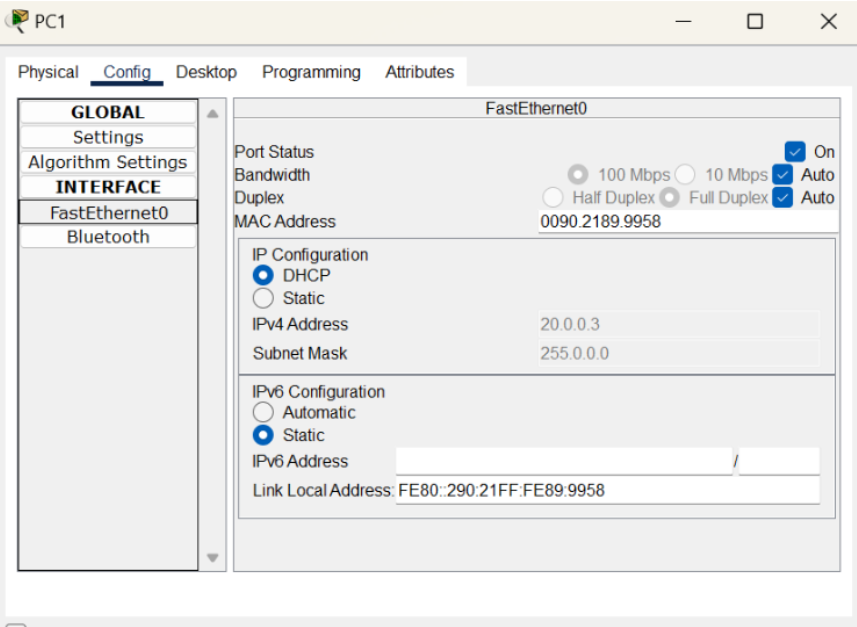


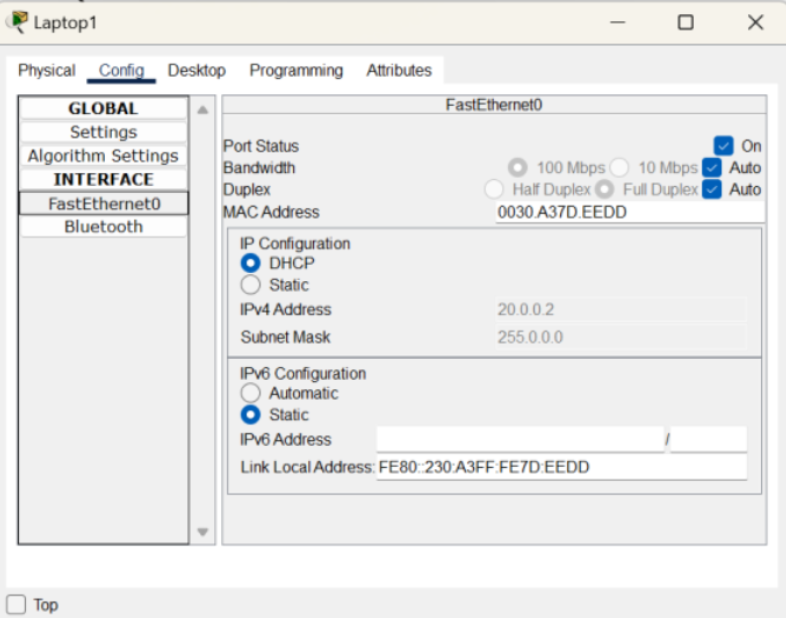
****

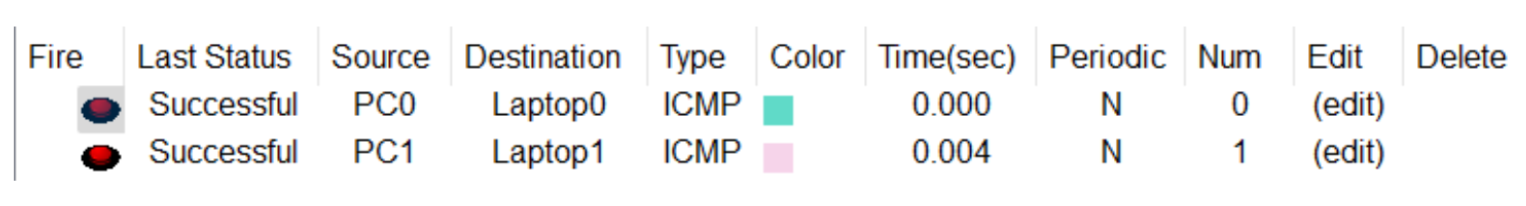
****

****

****

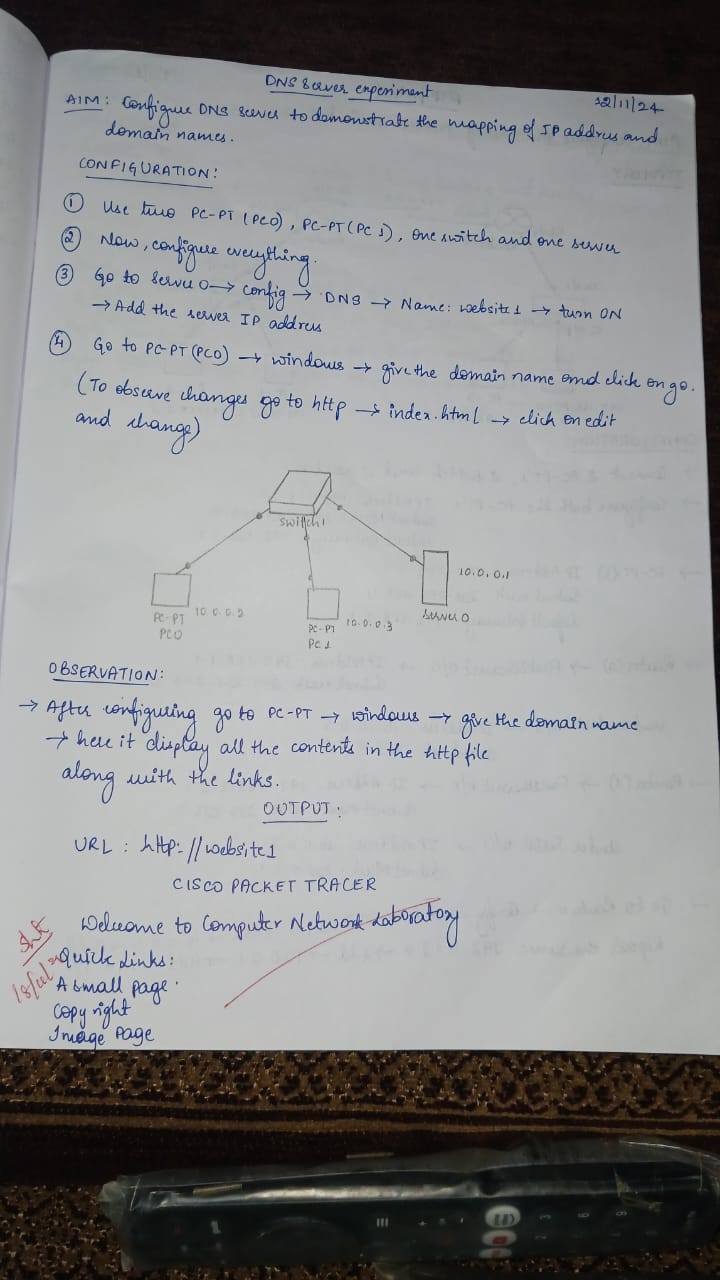
****

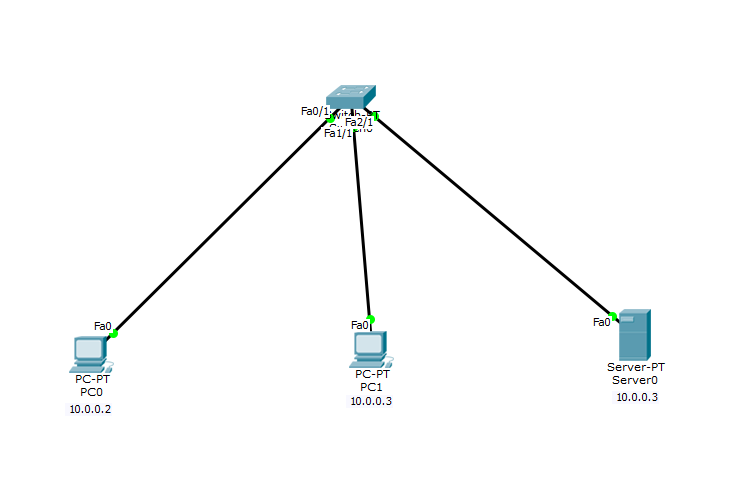
****

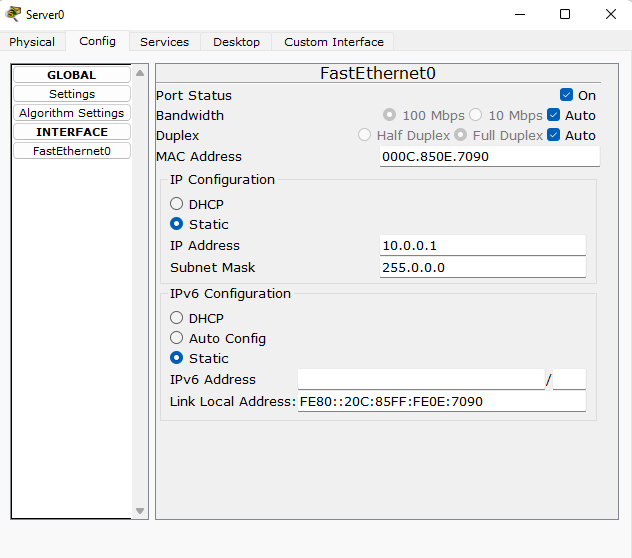
****

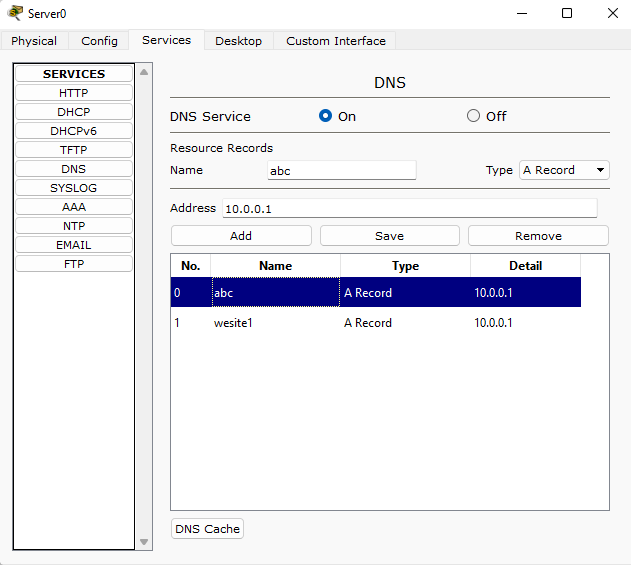
**LABORATORY PROGRAM-8**

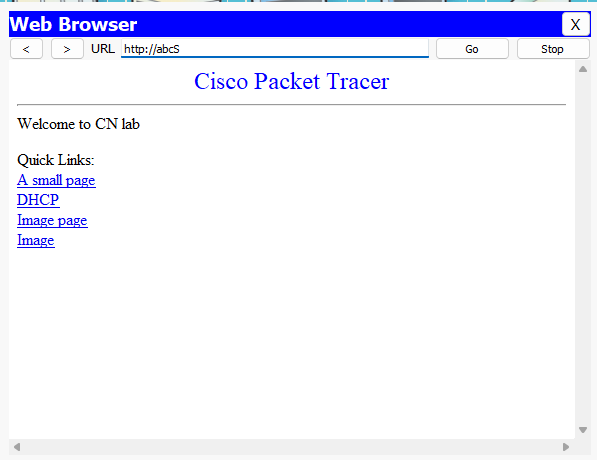
To configure DNS server to demonstrate the mapping of IP address and Domain names.





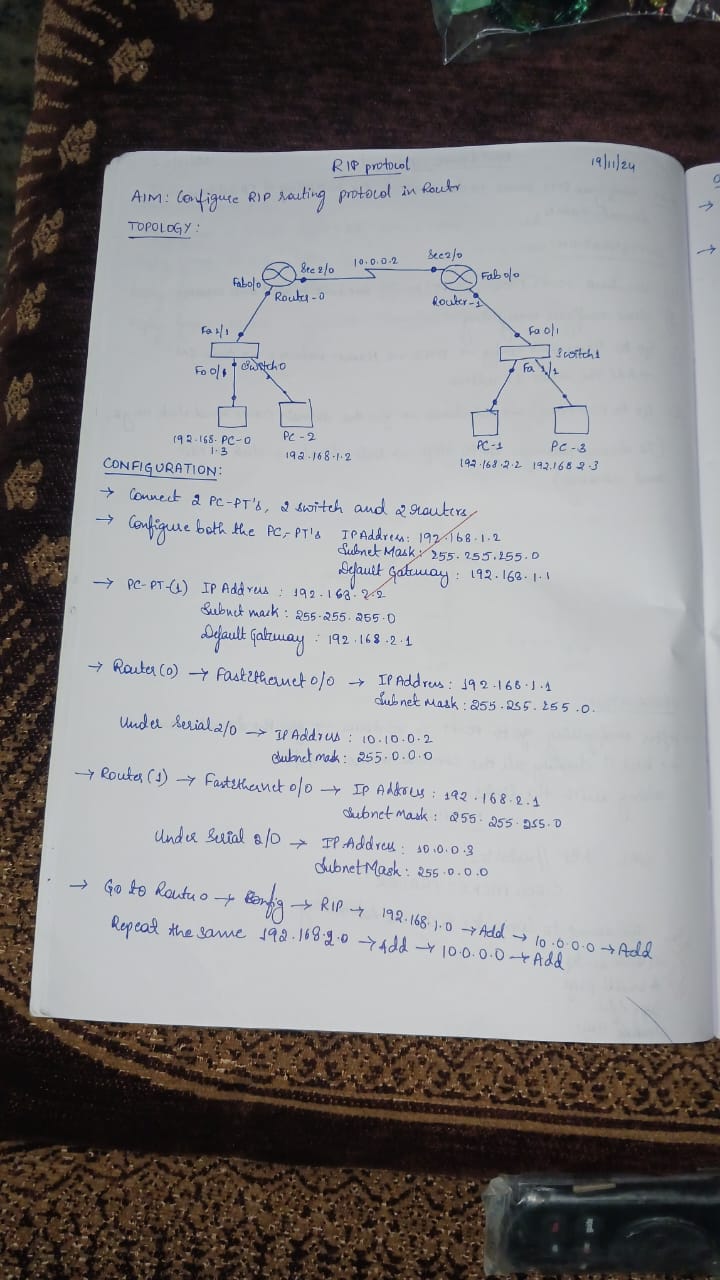


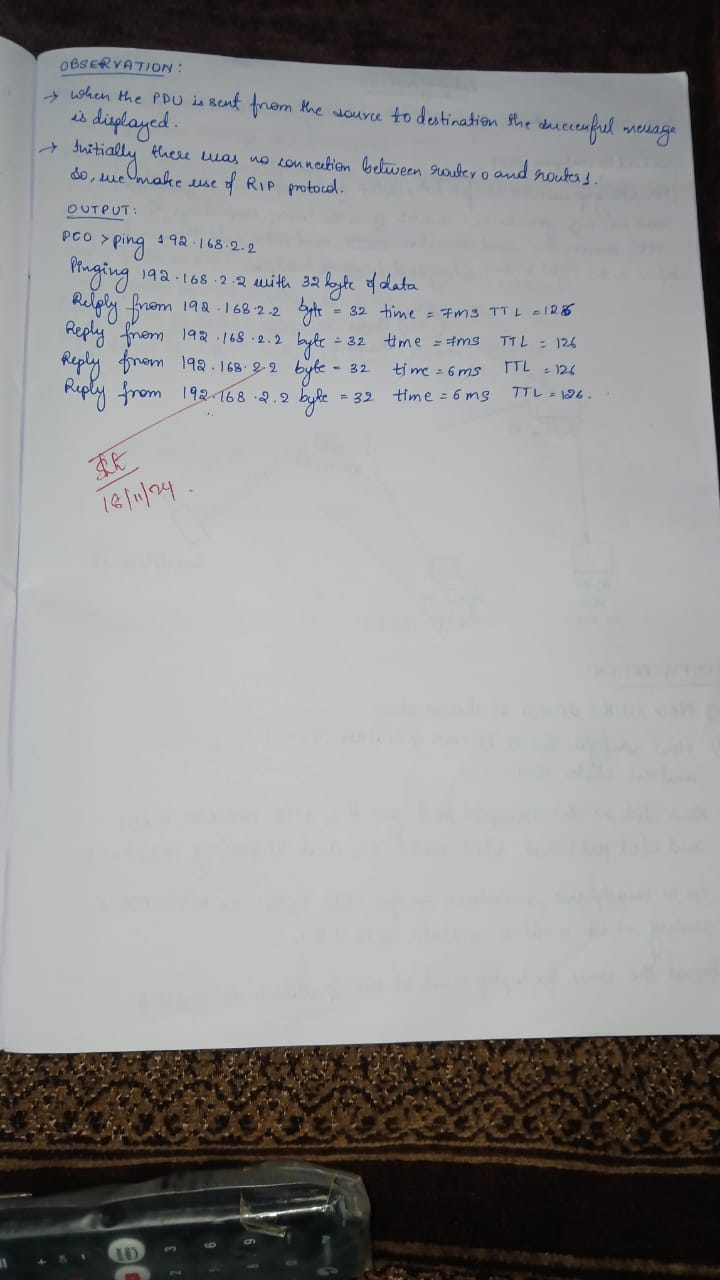


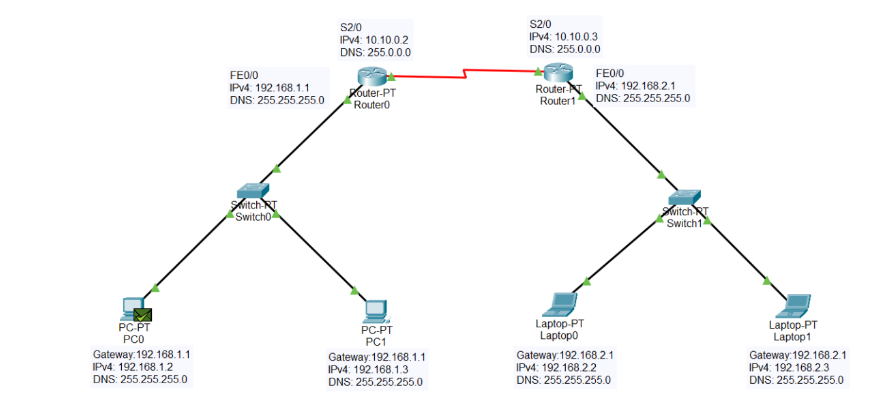


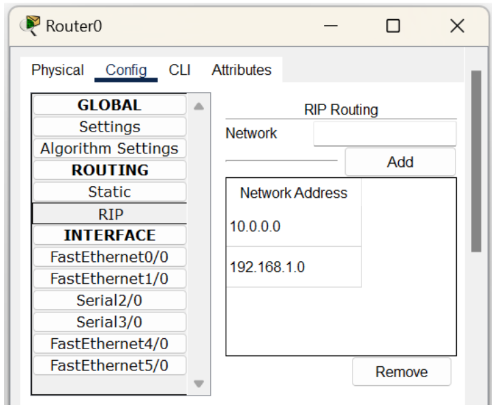
**LABORATORY PROGRAM-9**

Configure RIP routing protocol in router.

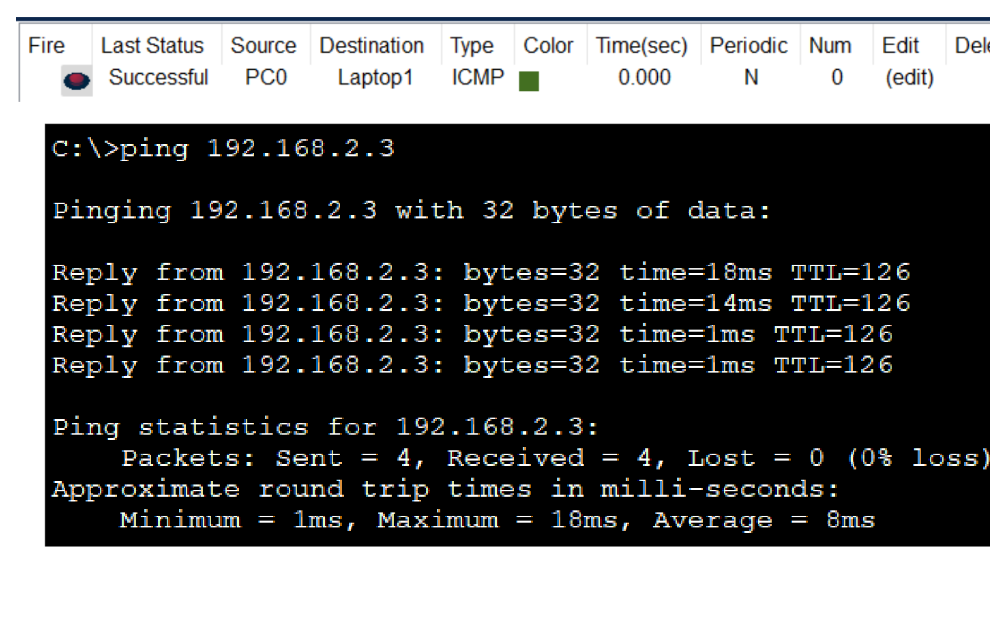






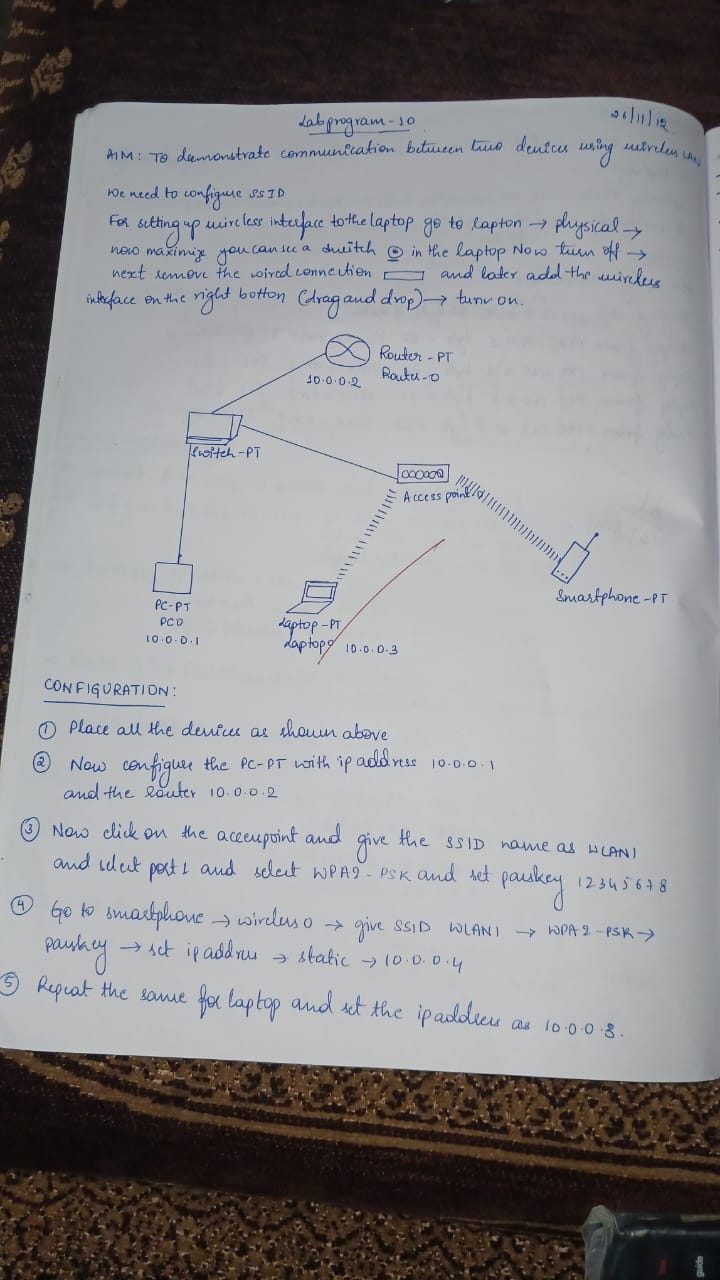


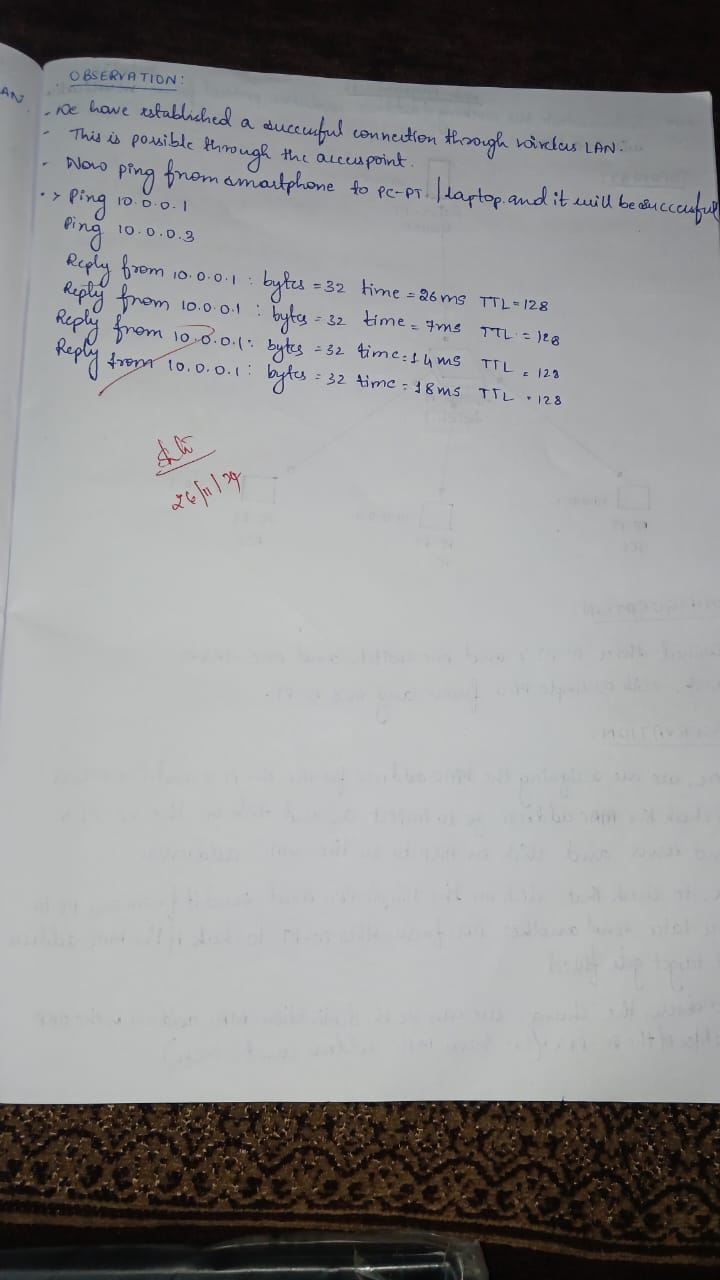


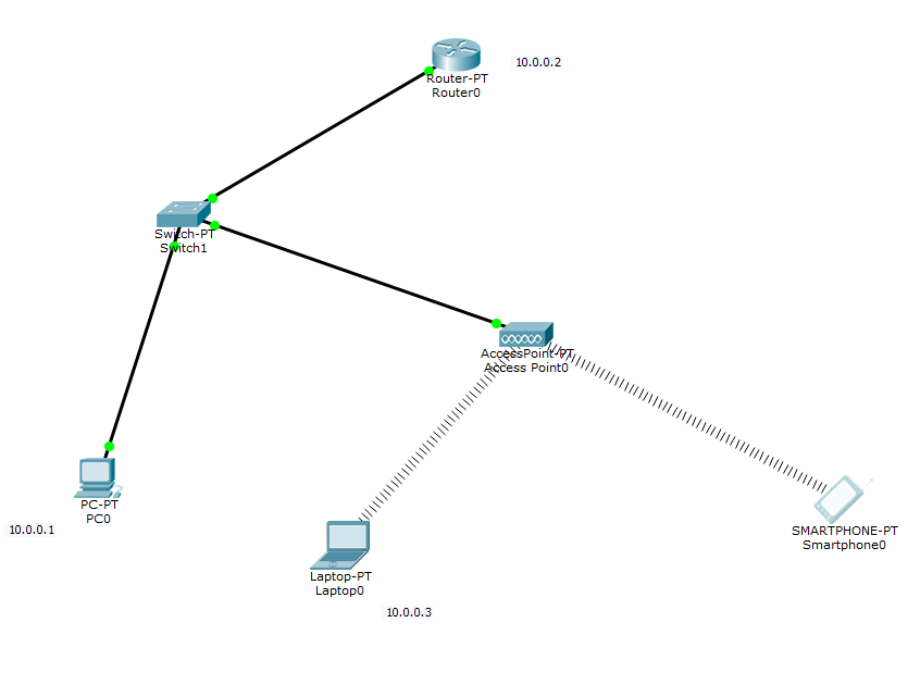


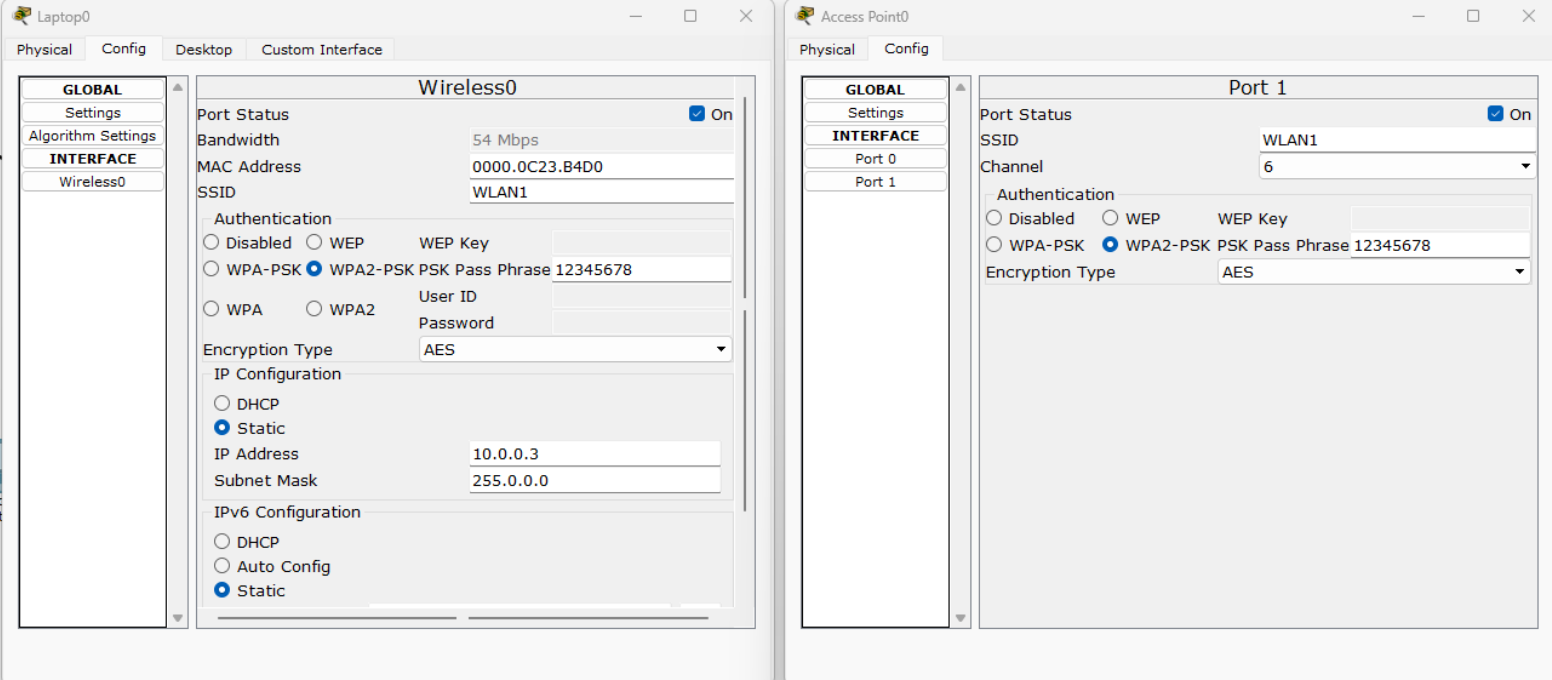
**LABORATORY PROGRAM-10**

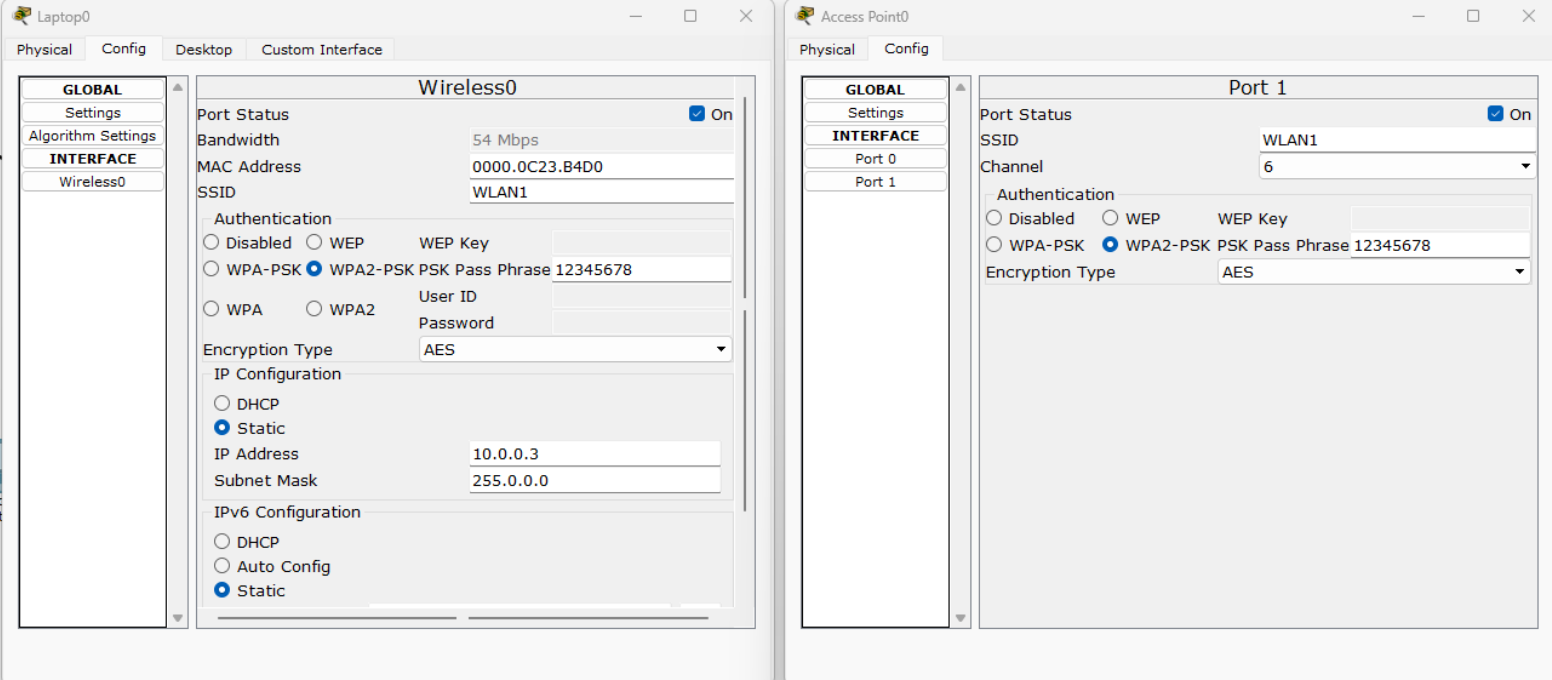
To demonstrate the communication between two devices using wireless LAN.

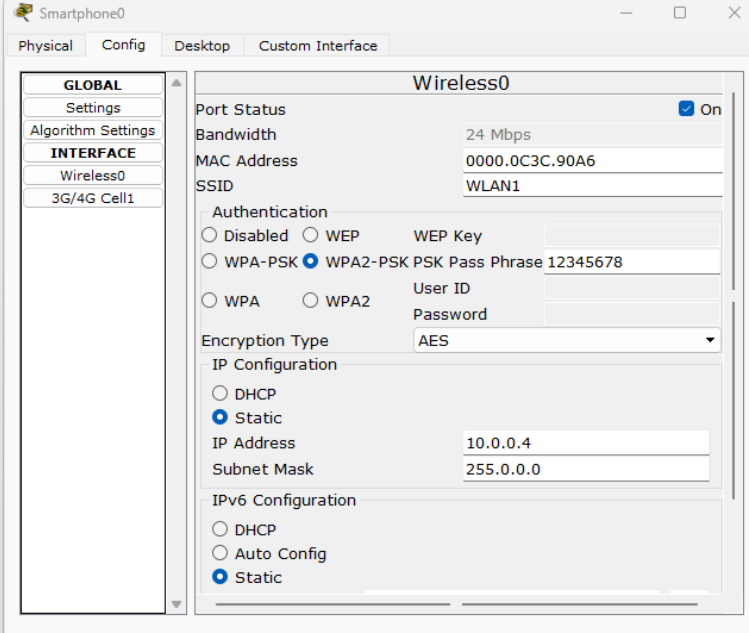




****

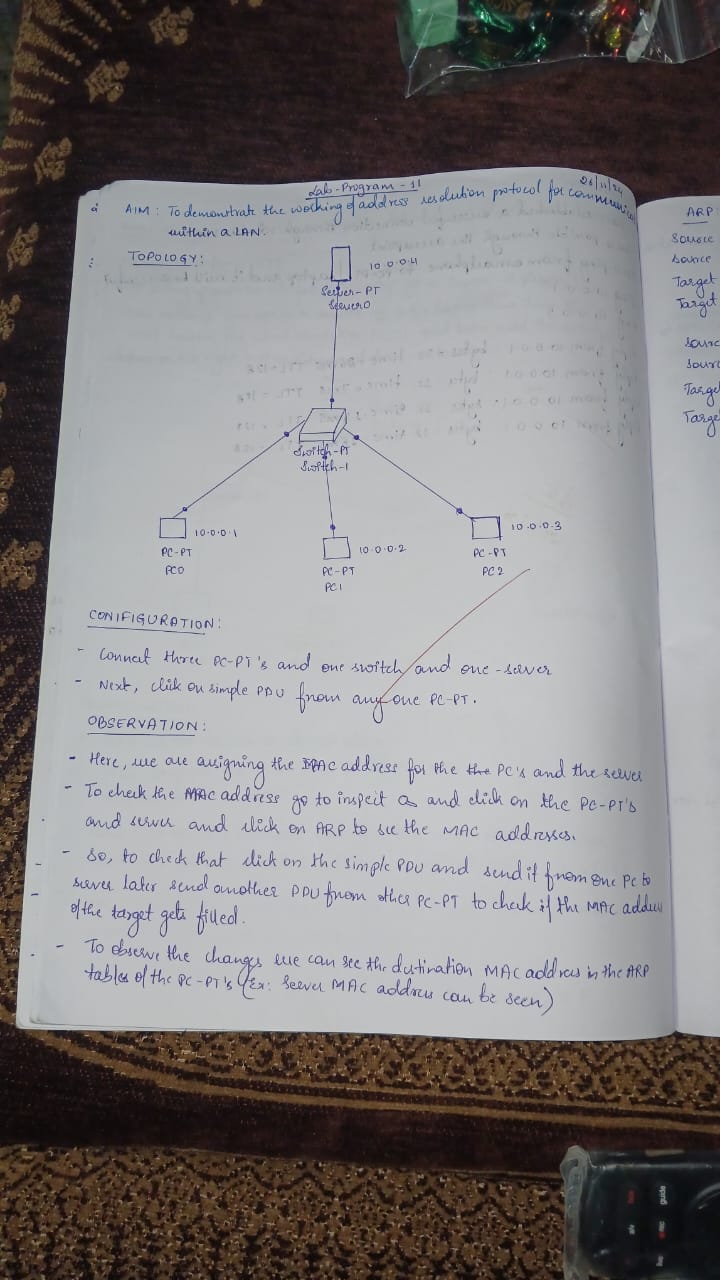
****

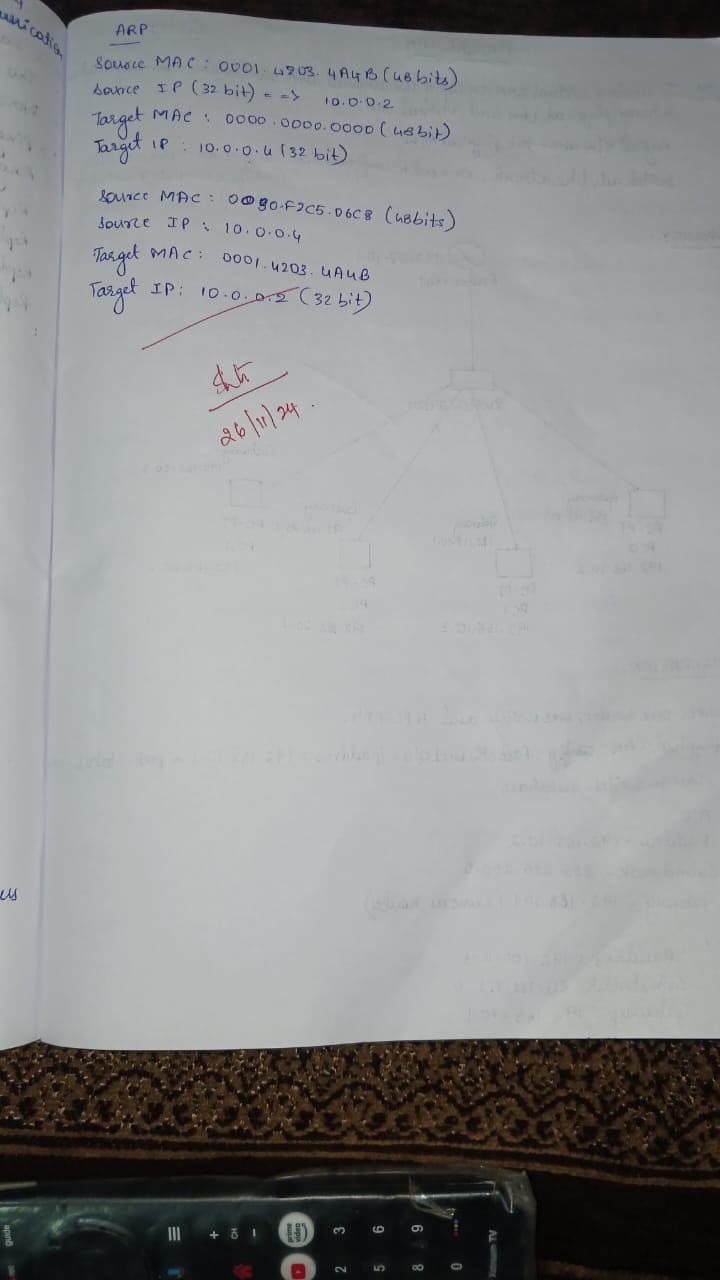
****

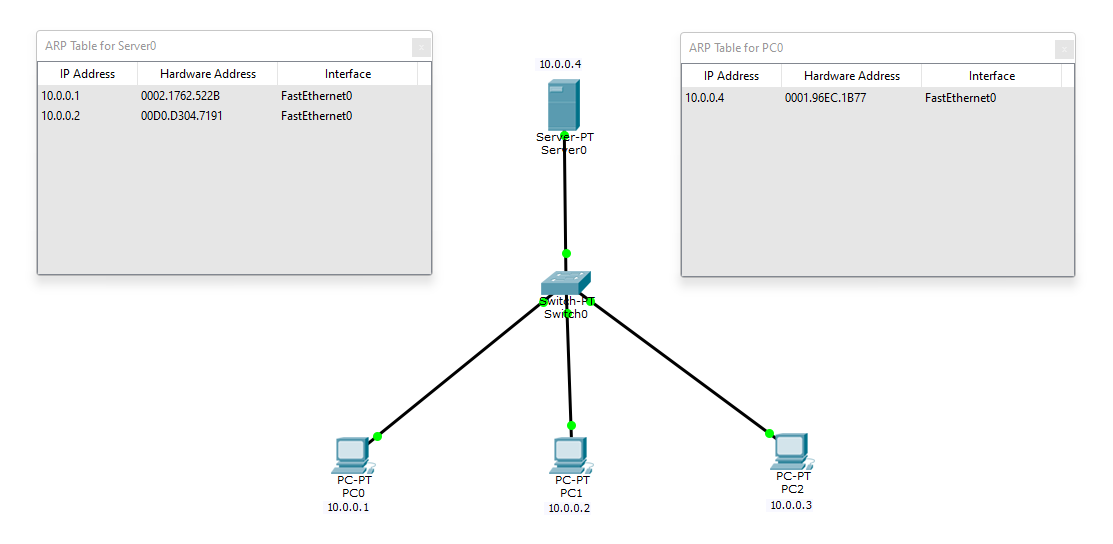


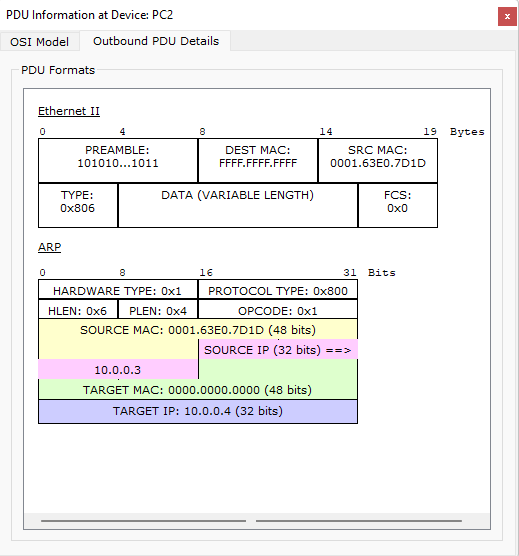
**LABORATORY PROGRAM-11**

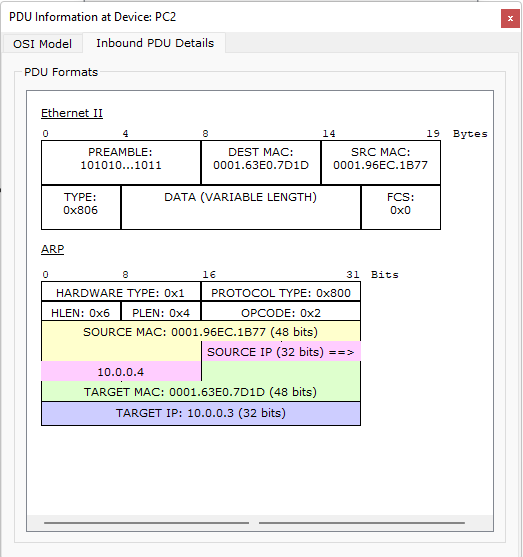
To demonstrate the working of ARP for communication within a LAN.



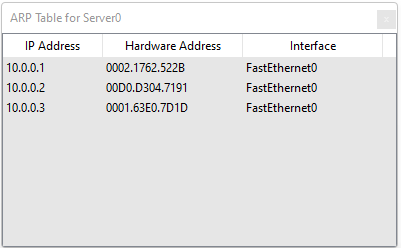






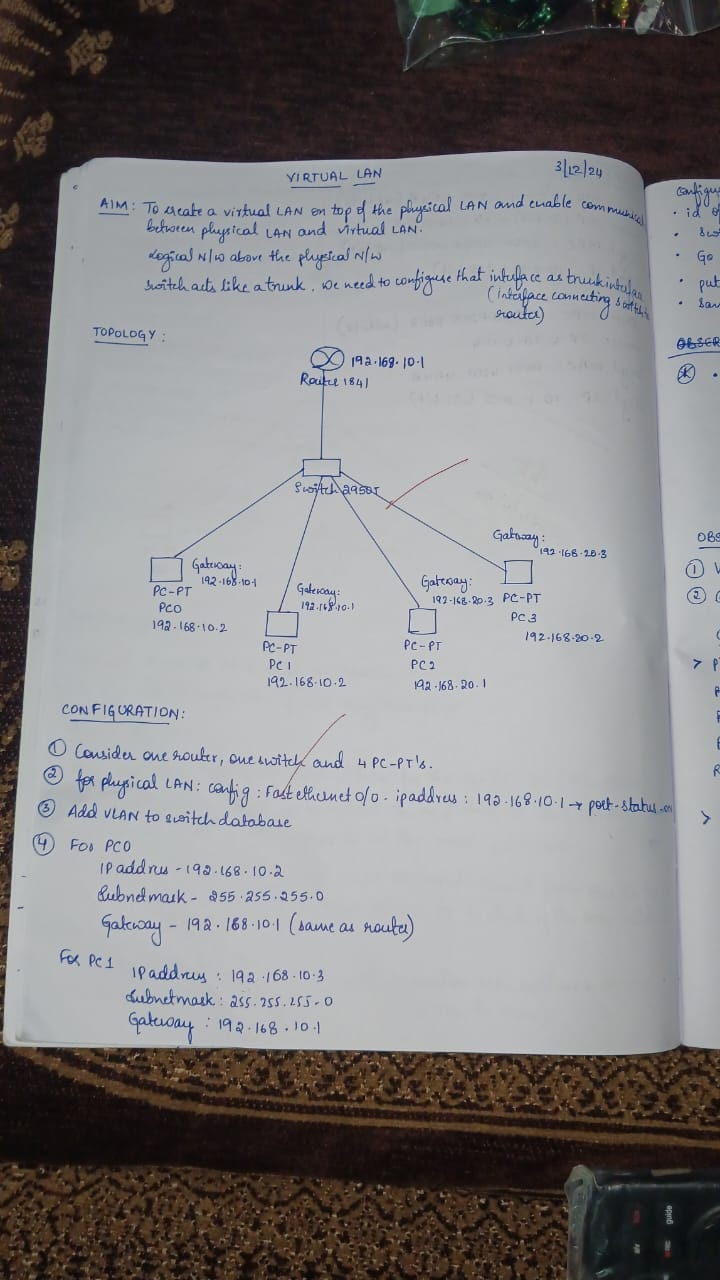


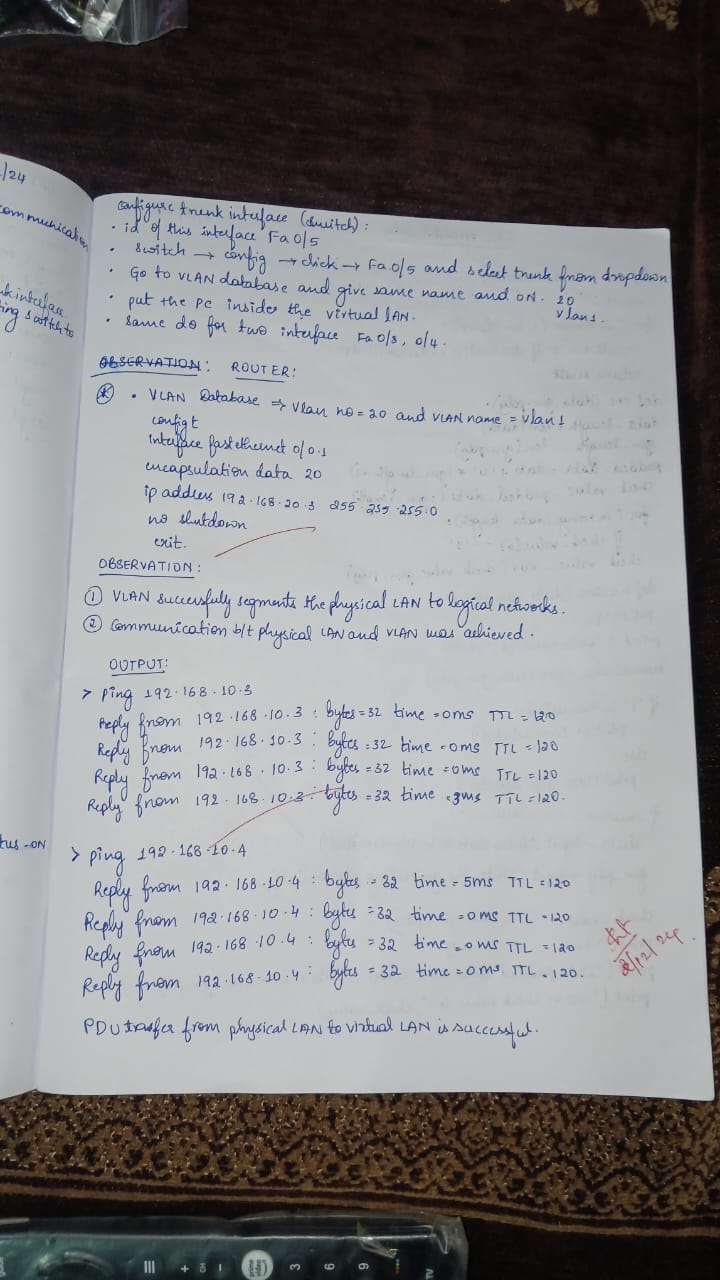


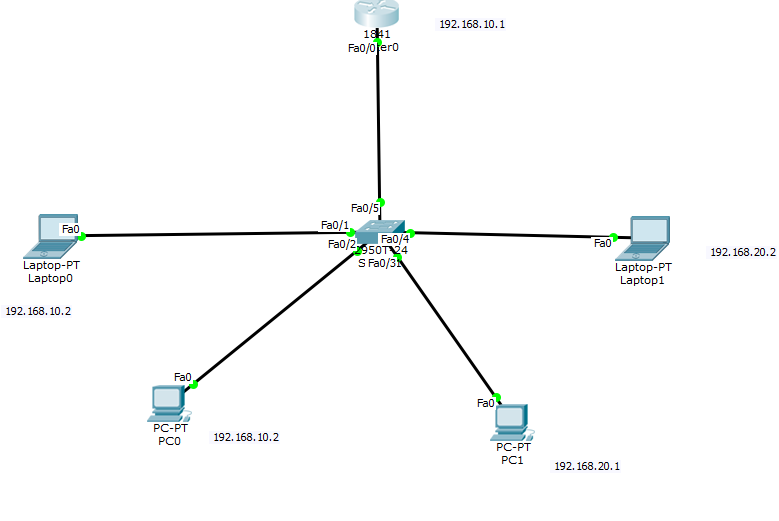


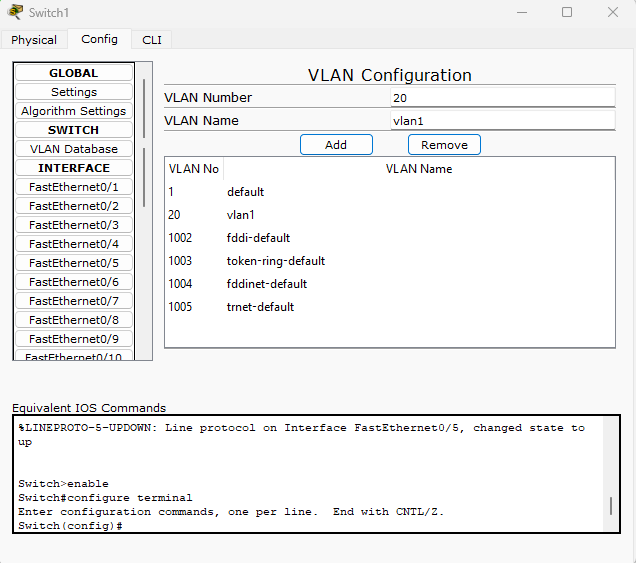
**LABORATORY PROGRAM-12**

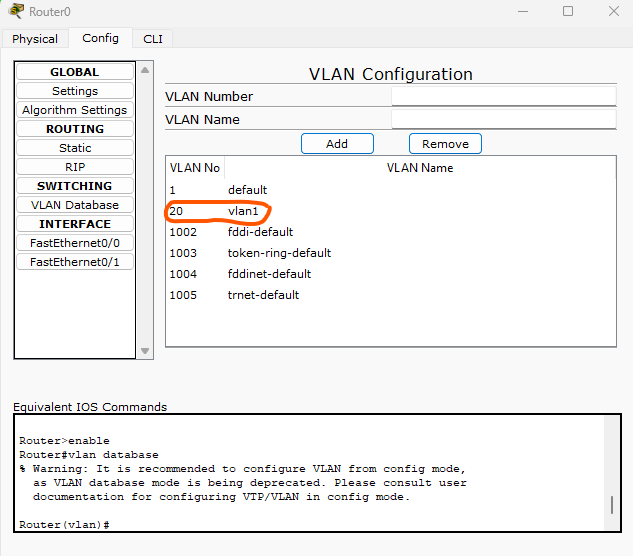
To create a virtual LAN on top of the physical LAN and enable communication between physical LAN and virtual LAN.

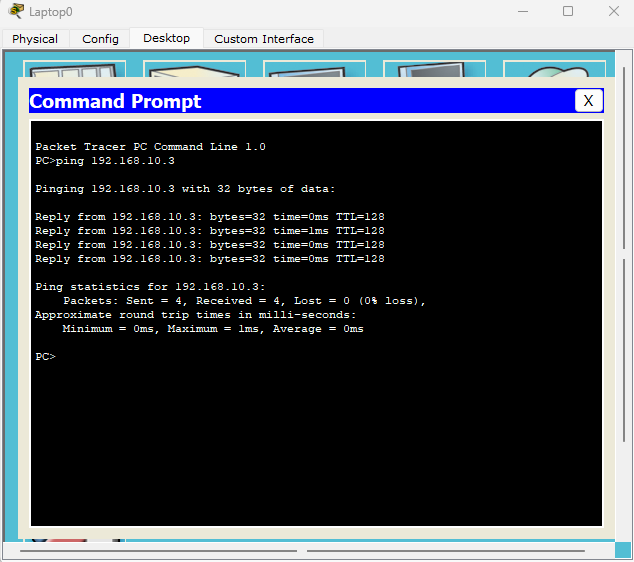






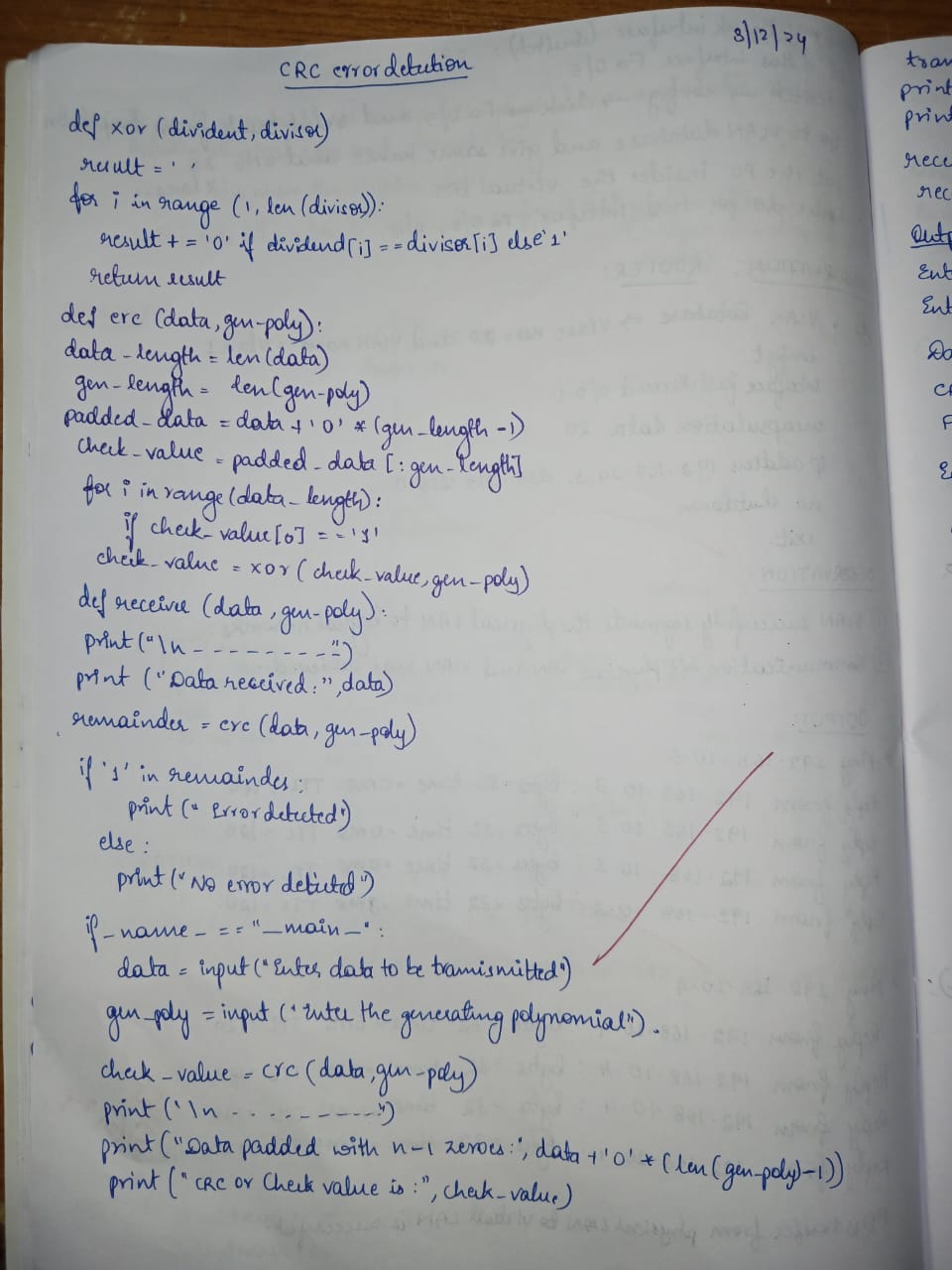


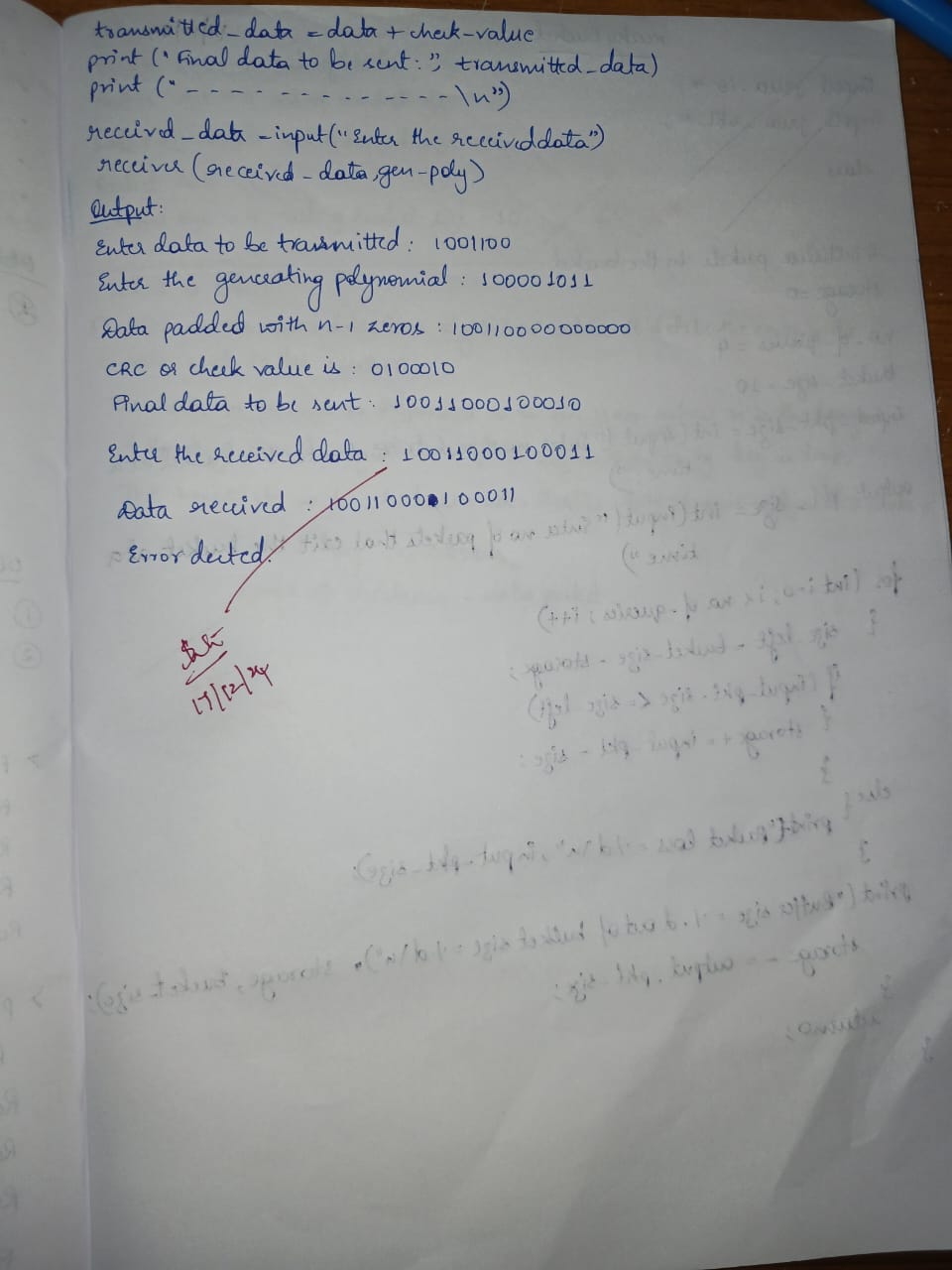




**LABORATORY PROGRAM-13**

Write a program for error detecting code using CRC-CCITT (8-bits).





**CODE:**

def xor(dividend, divisor):

    """Perform XOR operation between dividend and divisor."""

    result = ''

    for i in range(1, len(divisor)):

        result += '0' if dividend[i] == divisor[i] else '1'

    return result

def crc(data, gen\_poly):

    """Compute the CRC check value using CRC-CCITT (8-bit)."""

    data\_length = len(data)

    gen\_length = len(gen\_poly)

    # Append n-1 zeros to the data

    padded\_data = data + '0' \* (gen\_length - 1)

    check\_value = padded\_data[:gen\_length]

    for i in range(data\_length):

        if check\_value[0] == '1':

            # XOR operation if the first bit is 1

            check\_value = xor(check\_value, gen\_poly)

        else:

            # Retain original check value if first bit is 0

            check\_value = check\_value[1:]

        # Shift left and add the next data bit

        if i + gen\_length < len(padded\_data):

            check\_value += padded\_data[i + gen\_length]

    return check\_value[1:]  # Remove the leading bit

def receiver(data, gen\_poly):

    """Simulate the receiver side to check for errors."""

    print("\n-----------------------------")

    print("Data received:", data)

    # Perform CRC computation on received data

    remainder = crc(data, gen\_poly)

    # Check if the remainder is all zeros

    if '1' in remainder:

        print("Error detected")

    else:

        print("No error detected")

if \_\_name\_\_ == "\_\_main\_\_":

    # Input data and generator polynomial

    data = input("Enter data to be transmitted: ")

    gen\_poly = input("Enter the Generating polynomial: ")

    # Compute CRC check value

    check\_value = crc(data, gen\_poly)

    print("\n----------------------------------------")

    print("Data padded with n-1 zeros:", data + '0' \* (len(gen\_poly) - 1))

    print("CRC or Check value is:", check\_value)

    # Append check value to data for transmission

    transmitted\_data = data + check\_value

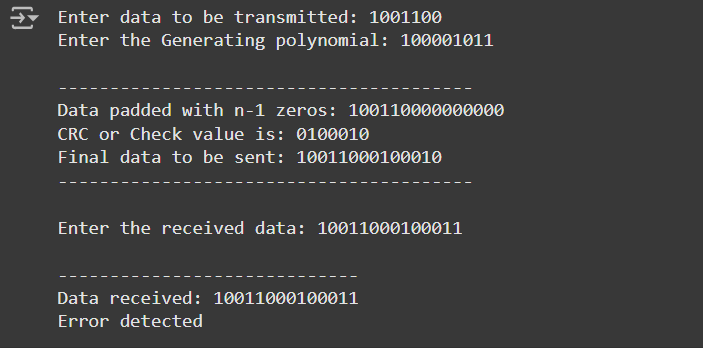
    print("Final data to be sent:", transmitted\_data)

    print("----------------------------------------\n")

    # Simulate the receiver side

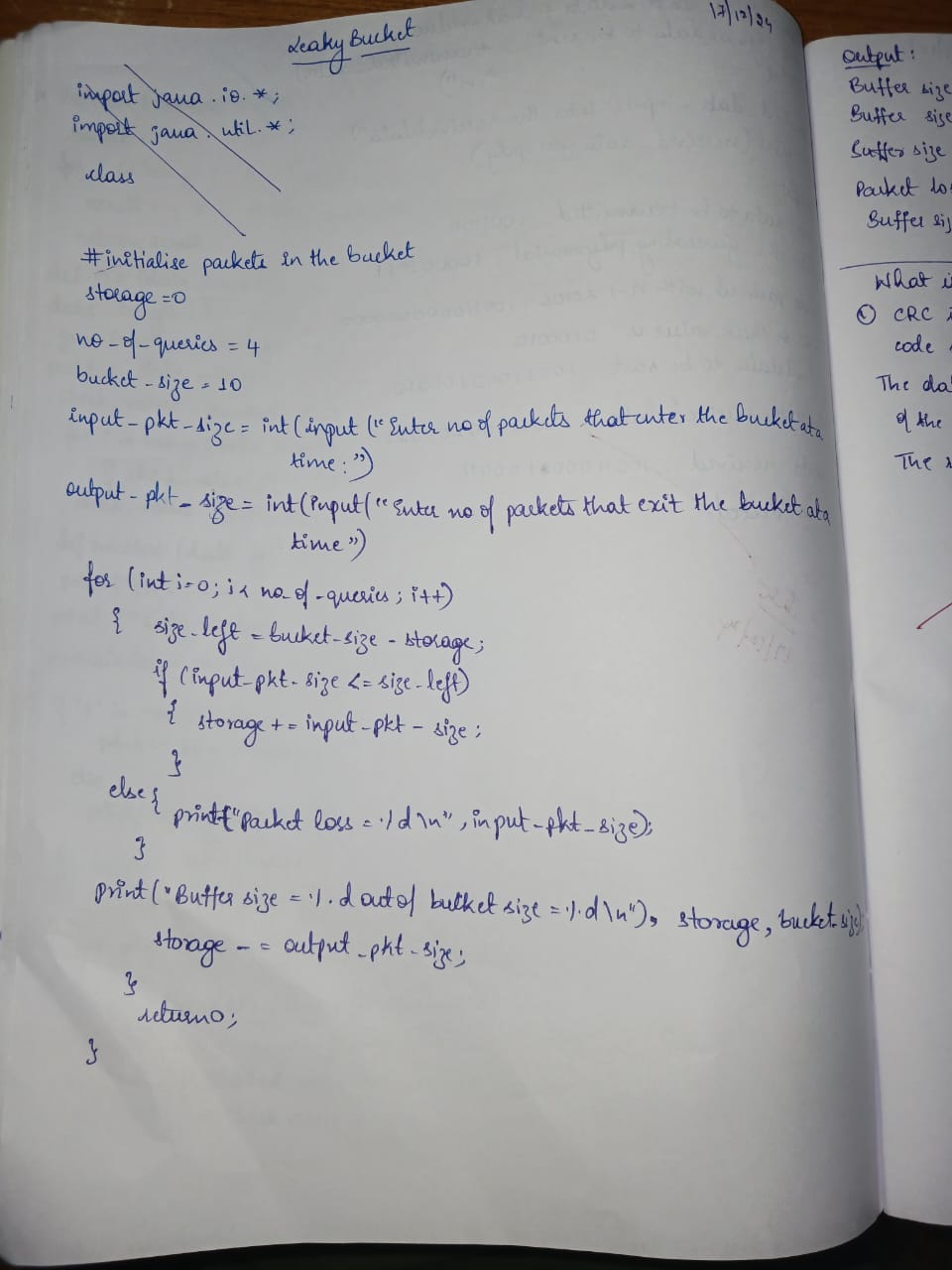
    received\_data = input("Enter the received data: ")

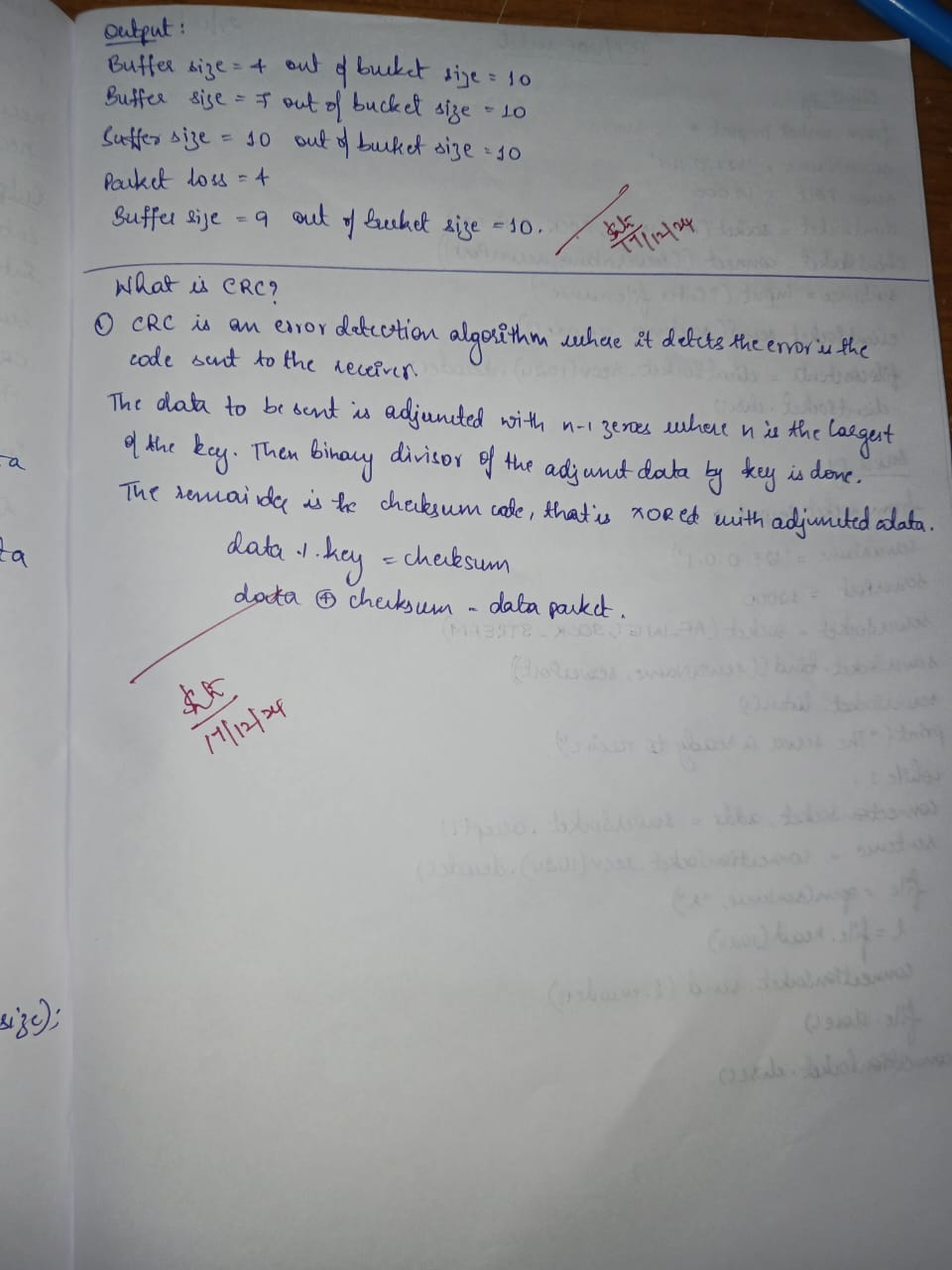
    receiver(received\_data, gen\_poly)

**OUTPUT:  
**

**LABORATORY PROGRAM-14**

Write a program for congestion control using Leaky bucket algorithm.





**CODE:**

storage = int(input("Enter initial packets in the bucket: "))

no\_of\_queries = int(input("Enter total no. of times bucket content is checked: "))

bucket\_size = int(input("Enter total no. of packets that can be accommodated in the bucket: "))

input\_pkt\_size = int(input("Enter no. of packets that enters the bucket at a time: "))

output\_pkt\_size = int(input("Enter no. of packets that exits the bucket at a time: "))

for i in range(no\_of\_queries):  # space left

    size\_left = bucket\_size - storage

    if input\_pkt\_size <= size\_left:

        # update storage

        storage += input\_pkt\_size

    else:

        print("Packet loss =", input\_pkt\_size)

    print(f"Buffer size = {storage} out of bucket size = {bucket\_size}")

    # as packets are sent out into the network, the size of the storage decreases

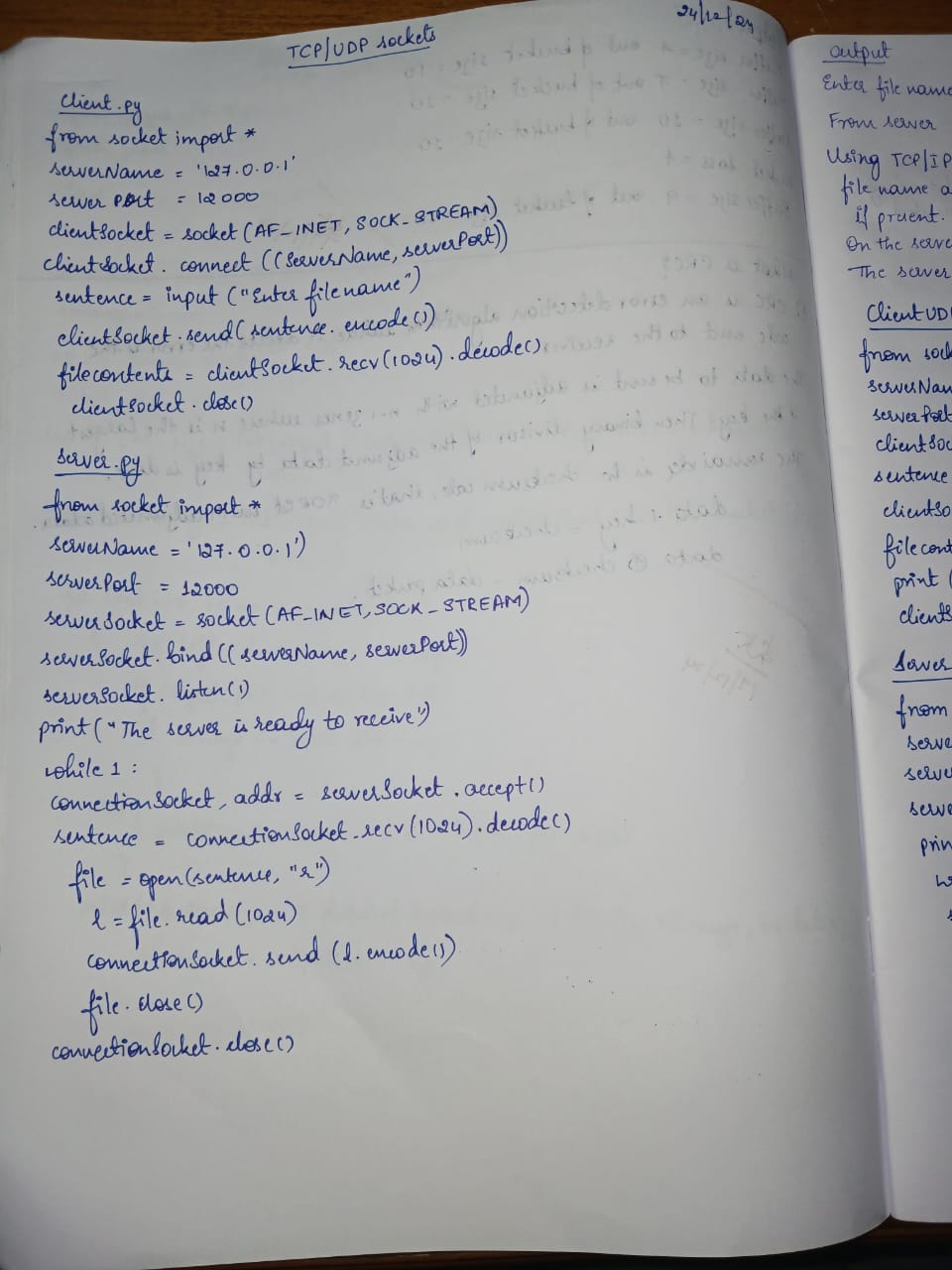
    storage -= output\_pkt\_size

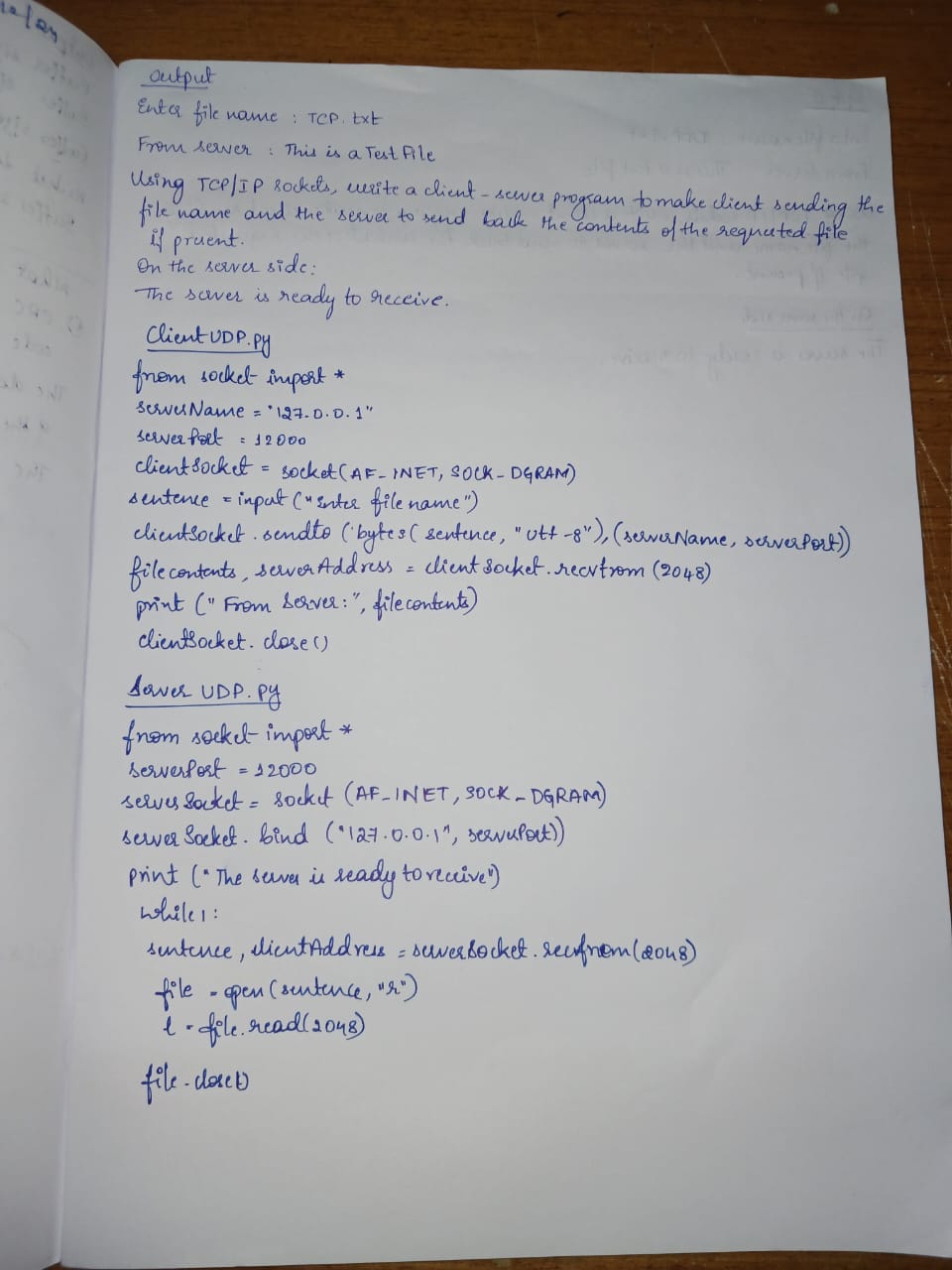
**OUTPUT:**

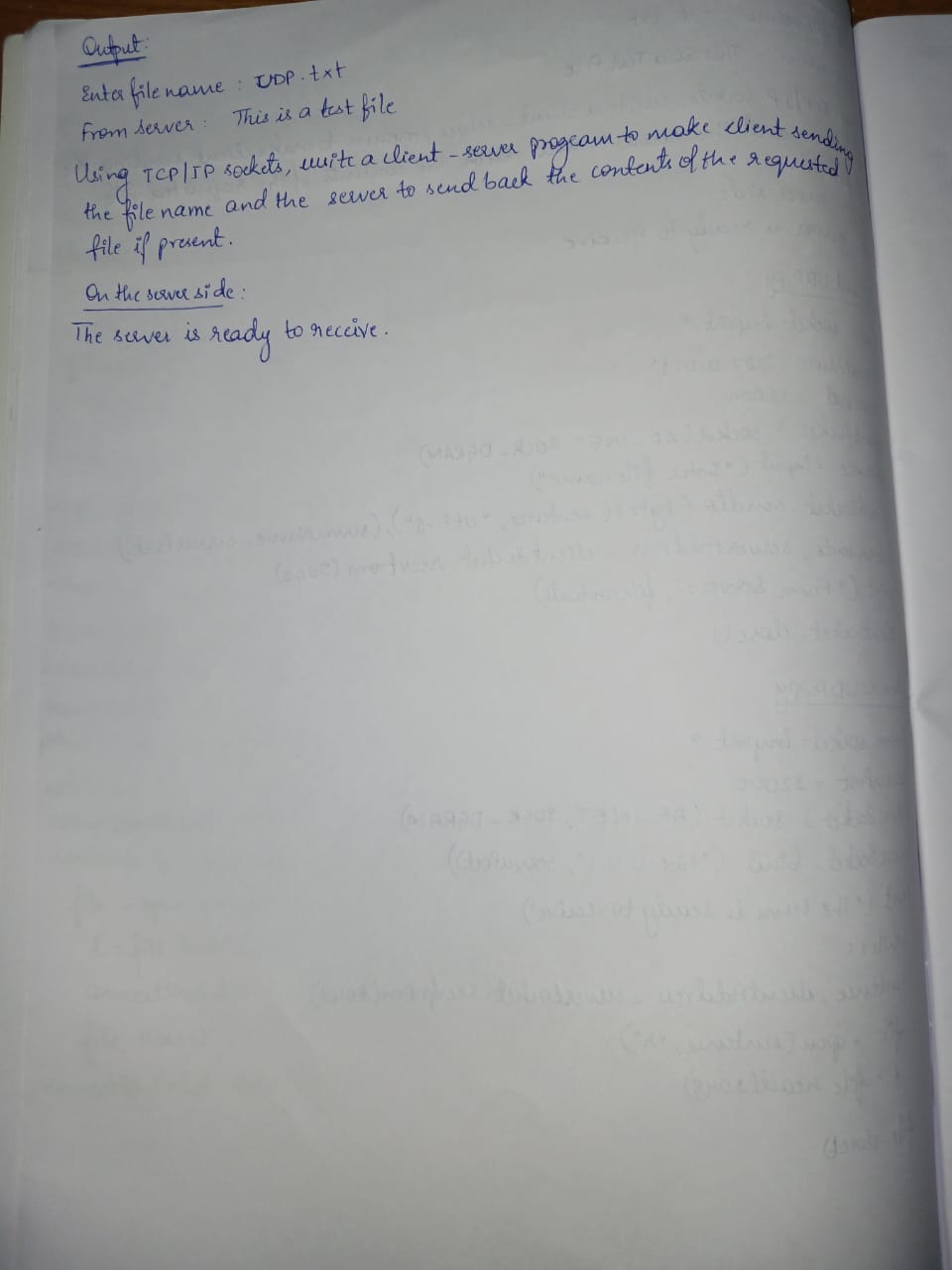


**LABORATORY PROGRAM-15**

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







**CODE:**

**Code: Client.py**

from socket import \*

serverName = "127.0.0.1" # Server address (localhost)

serverPort = 12000 # Port number where the server listens

# Create TCP socket

clientSocket = socket(AF\_INET, SOCK\_STREAM)

clientSocket.connect((serverName, serverPort)) # Connect to server

# Ask user for file name to request

sentence = input("Enter file name: ")

# Send file name to server

clientSocket.send(sentence.encode())

# Receive file contents from server

filecontents = clientSocket.recv(1024).decode()

print('From Server:', filecontents)

# Close the connection

clientSocket.close()

**Code: Server.py**

from socket import \*

serverName = "127.0.0.1" # Server address (localhost)

serverPort = 12000 # Port number to listen on

# Create TCP socket

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.bind((serverName, serverPort)) # Bind socket to the address and port

serverSocket.listen(1) # Listen for 1 connection

print("The server is ready to receive")

while True:

# Accept a connection

connectionSocket, addr = serverSocket.accept()

# Receive the file name from the client

sentence = connectionSocket.recv(1024).decode()

# Try opening the file

try:

file = open(sentence, "r") # Open file in read mode

fileContents = file.read(1024) # Read file content (up to 1024 bytes)

connectionSocket.send(fileContents.encode()) # Send file contents to client

file.close()

except FileNotFoundError:

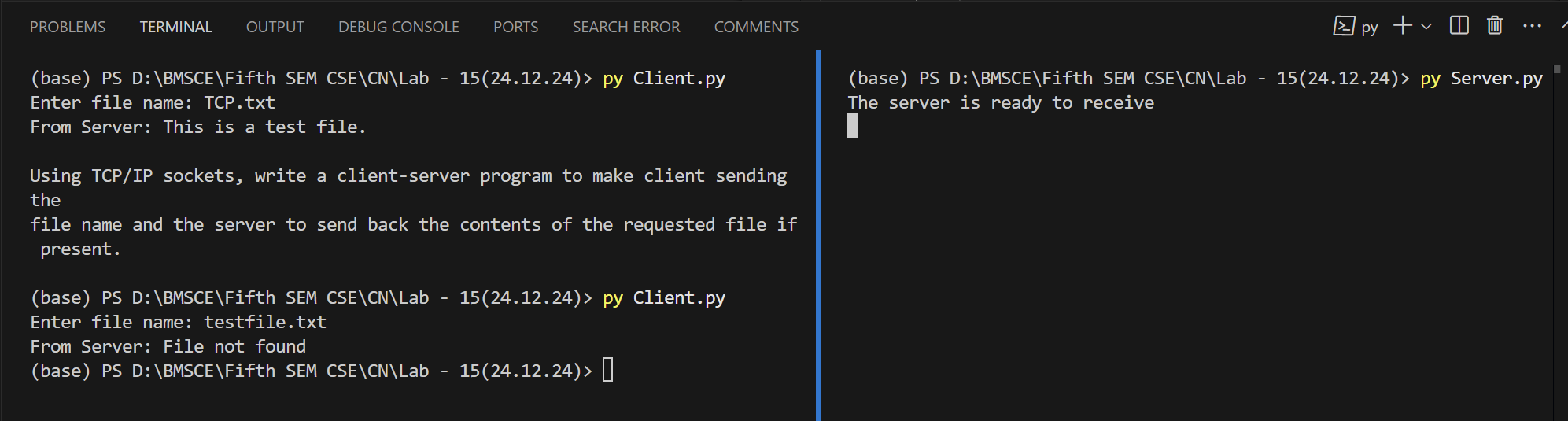
# Send error message if file not found

connectionSocket.send("File not found".encode())

# Close the connection

connectionSocket.close()

**OUTPUT:**



**Code: ClientUDP.py**

from socket import \*

serverName = "127.0.0.1" # Server address (localhost)

serverPort = 12000 # Port number where the server listens

# Create UDP socket

clientSocket = socket(AF\_INET, SOCK\_DGRAM)

# Ask user for file name to request

sentence = input("Enter file name: ")

# Send the file name to the server using UDP

clientSocket.sendto(sentence.encode("utf-8"), (serverName, serverPort))

# Receive file contents from the server

fileContents, serverAddress = clientSocket.recvfrom(2048)

# Print the file contents received from the server

print("From Server:", fileContents.decode())

# Close the UDP socket

clientSocket.close()

**Code: ServerUDP.py**

from socket import \*

serverPort = 12000 # Port number to listen on

serverSocket = socket(AF\_INET, SOCK\_DGRAM)

serverSocket.bind(("127.0.0.1", serverPort)) # Bind the socket to the server address and port

print("The server is ready to receive")

while True:

# Receive file name from the client

sentence, clientAddress = serverSocket.recvfrom(2048)

# Try opening the file

try:

file = open(sentence.decode(), "r") # Open file in read mode

fileContents = file.read(2048) # Read file content (up to 2048 bytes)

serverSocket.sendto(fileContents.encode("utf-8"), clientAddress) # Send file contents to client

file.close()

except FileNotFoundError:

# Send error message if file not found

serverSocket.sendto("File not found".encode("utf-8"), clientAddress)

**Output:**

