**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



# LAB REPORT

**on**

COURSE TITLE

***Submitted by***

# Sreenath.M (1BM20CS217)

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

# COMPUTER SCIENCE AND ENGINEERING



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

# BENGALURU-560019

**October-2022 to Feb-2023**

# B. M. S. College of Engineering,

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “LAB COURSE **COMPUTER NETWORKS**” carried out by **SREENATH.M (1BM20CS217),** who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of a **Computer Networks - (20CS5PCCON)** work prescribed for the said degree.

**Dr. LOHITH J J Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

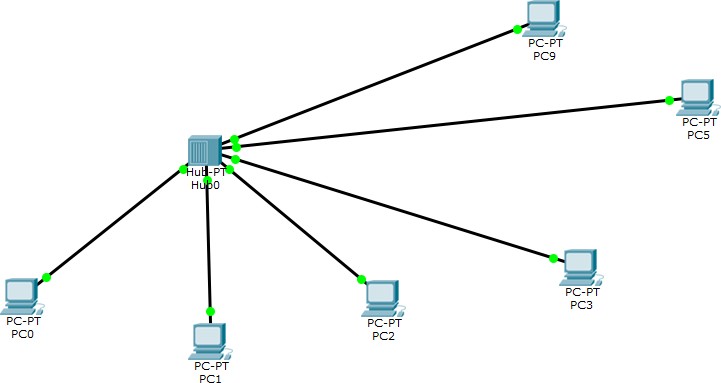
`

**Index**

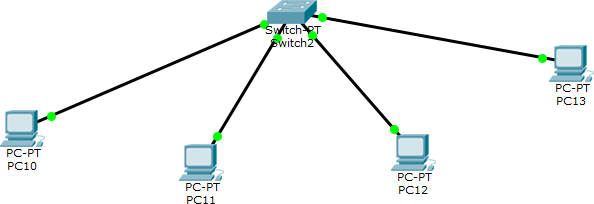
|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** |
| **01** | 07/11/2022 | Creating a topology and simulate sending a simple PDU from  source to destination using hub and switch as connecting devices. |
| **02** | 14/11/2022 | Configuring IP address to Routers in Packet Tracer. Explore the following messages: Ping Responses, Destination unreachable, Request timed out, Reply |
| **03** | 19/11/2022 | Configuring default route to the Router |
| **04** | 28/11/2022 | Configuring DHCP within a LAN in a packet Tracer |
| **05** | 05/12/2022 | Configuring RIP Routing Protocol in Routers |
| **06** | 12/12/2022 | Demonstration of WEB server and DNS using Packet Tracer |
| **07** | 19/12/2022 | Write a program for error detecting code using CRC-CCITT (16-bits). |
| **08** | 26/12/2022 | Write a program for distance vector algorithm to find suitable path  for transmission. |
| **09** | 02/01/2023 | Implement Dijkstra’s algorithm to compute the shortest path for a given topology. |
| **10** | 09/01/2023 | Write a program for congestion control using Leaky bucket  algorithm. |
| **11** | 16/01/2023 | 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. |
| **12** | 16/01/2023 | Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of  the requested file if present. |

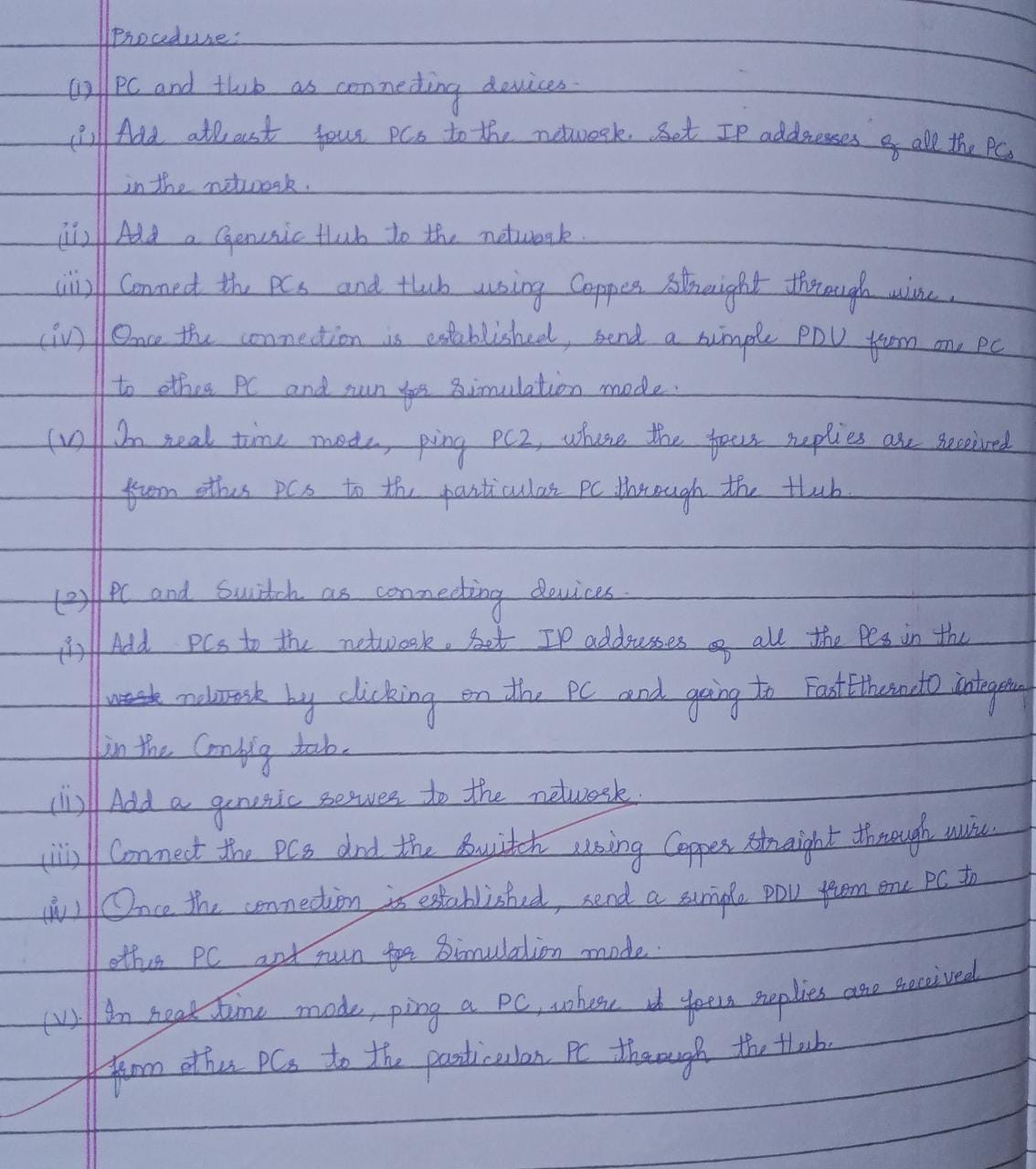
## LAB 01 : Creating a topology and simulate sending a simple PDU from source to destination using hub and switch as connecting devices.

Simple PDU using Hub

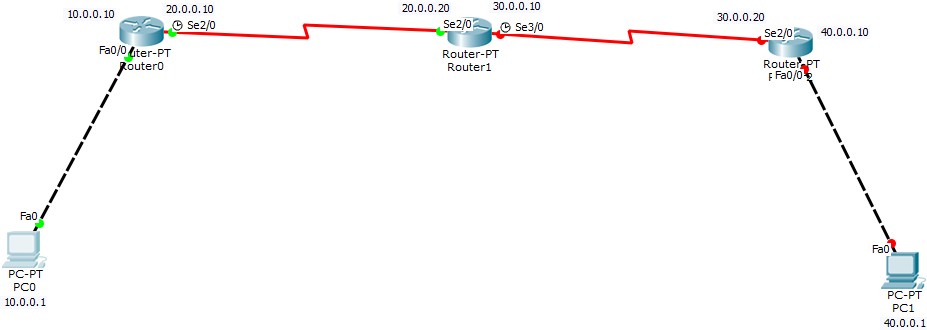


Simple PDU using Switch

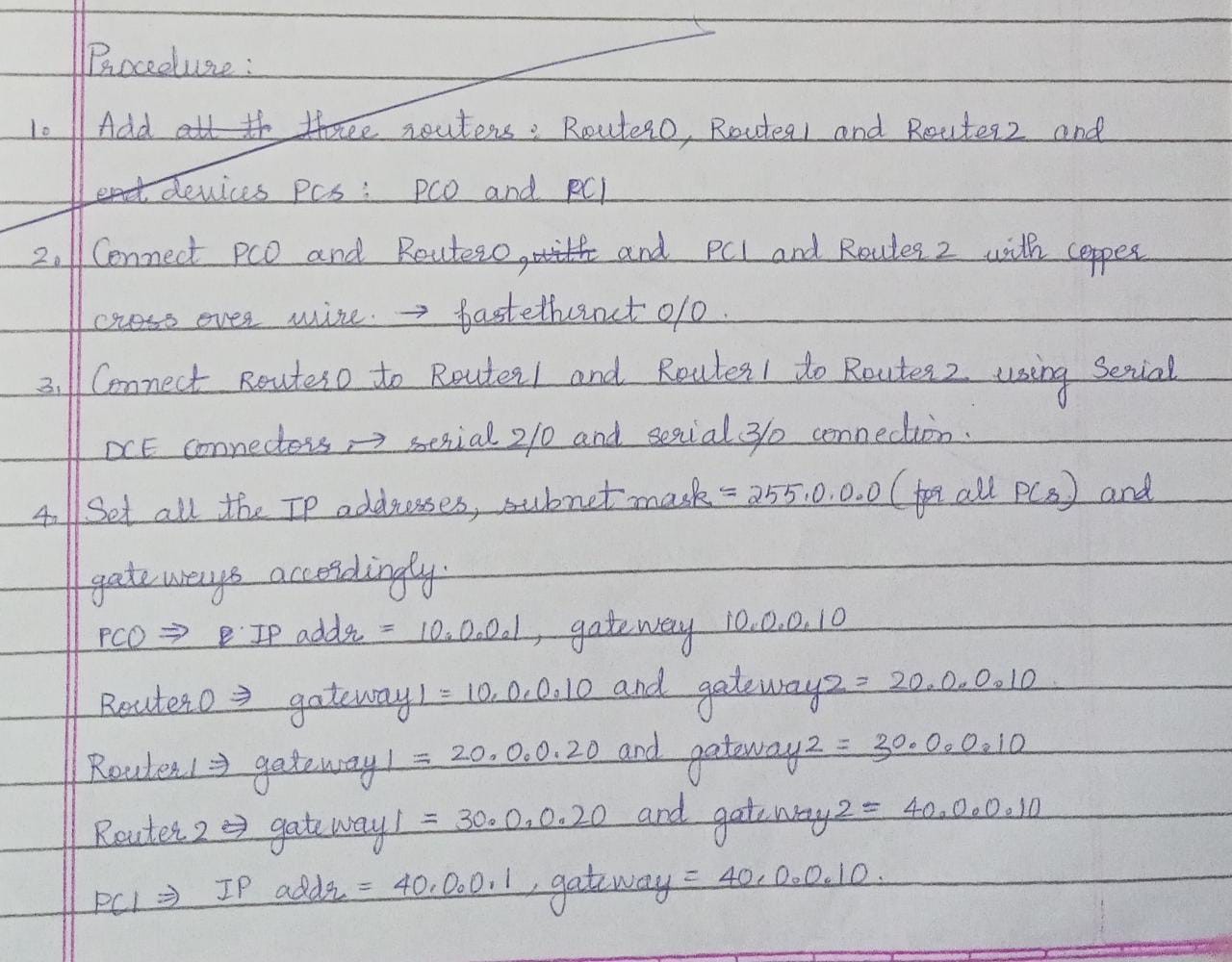




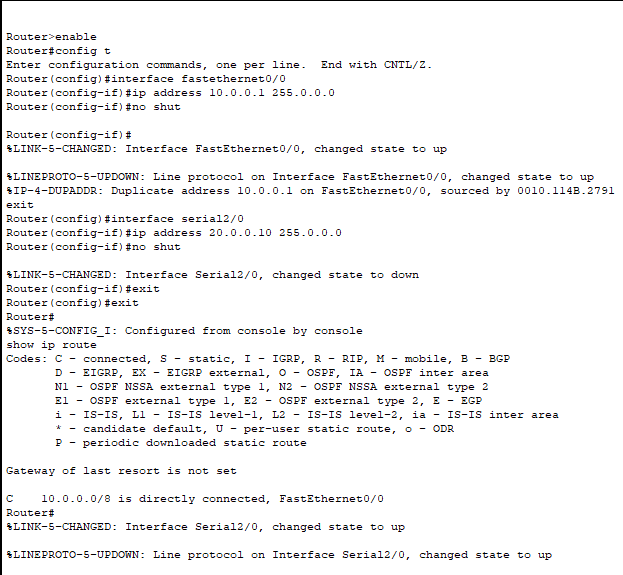
## LAB 02 : Configuring IP address to Routers in Packet Tracer. Explore the following messages: Ping Responses, Destination unreachable, Request timed out, Reply



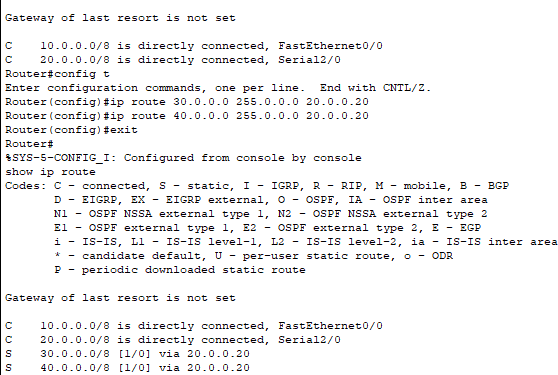
Procedure :



CLI commands for Router0 :



Teaching the Router0 about the 30.0.0.0 and 40.0.0.0 networks :



Similarly, this is done for Router1 for 10.0.0.0 and 40.0.0.0 networks, Router2 for

10.0.0.0 and 20.0.0.0 networks.

Pinging all the routers and PC1 from PC0

Packet Tracer PC Command Line 1.0 PC>ping 10.0.0.10

Pinging 10.0.0.10 with 32 bytes of data:

Reply from 10.0.0.10: bytes=32 time=1ms TTL=255 Reply from 10.0.0.10: bytes=32 time=0ms TTL=255 Reply from 10.0.0.10: bytes=32 time=0ms TTL=255 Reply from 10.0.0.10: bytes=32 time=0ms TTL=255

Ping statistics for 10.0.0.10:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

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

PC>ping 20.0.0.10

Pinging 20.0.0.10 with 32 bytes of data:

Reply from 20.0.0.10: bytes=32 time=0ms TTL=255 Reply from 20.0.0.10: bytes=32 time=0ms TTL=255 Reply from 20.0.0.10: bytes=32 time=0ms TTL=255 Reply from 20.0.0.10: bytes=32 time=0ms TTL=255

Ping statistics for 20.0.0.10:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

Minimum = 0ms, Maximum = 0ms, Average = 0ms PC>ping 20.0.0.20

Pinging 20.0.0.20 with 32 bytes of data:

Reply from 20.0.0.20: bytes=32 time=1ms TTL=254 Reply from 20.0.0.20: bytes=32 time=3ms TTL=254 Reply from 20.0.0.20: bytes=32 time=3ms TTL=254 Reply from 20.0.0.20: bytes=32 time=1ms TTL=254

Ping statistics for 20.0.0.20:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

Minimum = 1ms, Maximum = 3ms, Average = 2ms PC>ping 30.0.0.10

Pinging 30.0.0.10 with 32 bytes of data:

Reply from 30.0.0.10: bytes=32 time=22ms TTL=254 Reply from 30.0.0.10: bytes=32 time=3ms TTL=254 Reply from 30.0.0.10: bytes=32 time=1ms TTL=254 Reply from 30.0.0.10: bytes=32 time=13ms TTL=254

Ping statistics for 30.0.0.10:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

Minimum = 1ms, Maximum = 22ms, Average = 9ms PC>ping 30.0.0.20

Pinging 30.0.0.20 with 32 bytes of data:

Reply from 30.0.0.20: bytes=32 time=13ms TTL=253 Reply from 30.0.0.20: bytes=32 time=15ms TTL=253 Reply from 30.0.0.20: bytes=32 time=23ms TTL=253 Reply from 30.0.0.20: bytes=32 time=2ms TTL=253

Ping statistics for 30.0.0.20:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

Minimum = 2ms, Maximum = 23ms, Average = 13ms PC>ping 40.0.0.10

Pinging 40.0.0.10 with 32 bytes of data:

Reply from 40.0.0.10: bytes=32 time=29ms TTL=253 Reply from 40.0.0.10: bytes=32 time=19ms TTL=253 Reply from 40.0.0.10: bytes=32 time=14ms TTL=253 Reply from 40.0.0.10: bytes=32 time=21ms TTL=253

Ping statistics for 40.0.0.10:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

Minimum = 14ms, Maximum = 29ms, Average = 20ms PC>ping 40.0.0.1

Pinging 40.0.0.1 with 32 bytes of data:

Request timed out.

Reply from 40.0.0.1: bytes=32 time=28ms TTL=125 Reply from 40.0.0.1: bytes=32 time=15ms TTL=125

Reply from 40.0.0.1: bytes=32 time=2ms TTL=125

Ping statistics for 40.0.0.1:

Packets: Sent = 4, Received = 3, Lost = 1 (25% loss), Approximate round trip times in milli-seconds:

Minimum = 2ms, Maximum = 28ms, Average = 15ms PC>ping 40.0.0.1

Pinging 40.0.0.1 with 32 bytes of data:

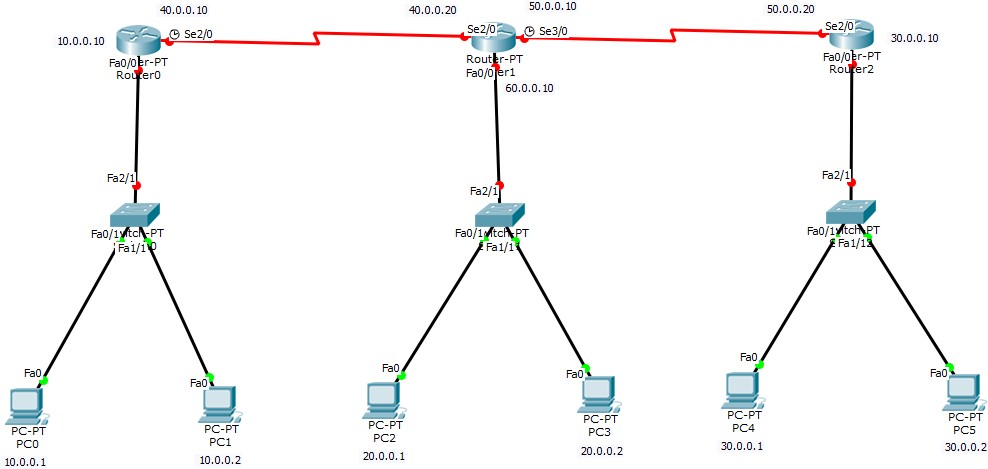
Reply from 40.0.0.1: bytes=32 time=3ms TTL=125 Reply from 40.0.0.1: bytes=32 time=2ms TTL=125 Reply from 40.0.0.1: bytes=32 time=4ms TTL=125 Reply from 40.0.0.1: bytes=32 time=2ms TTL=125

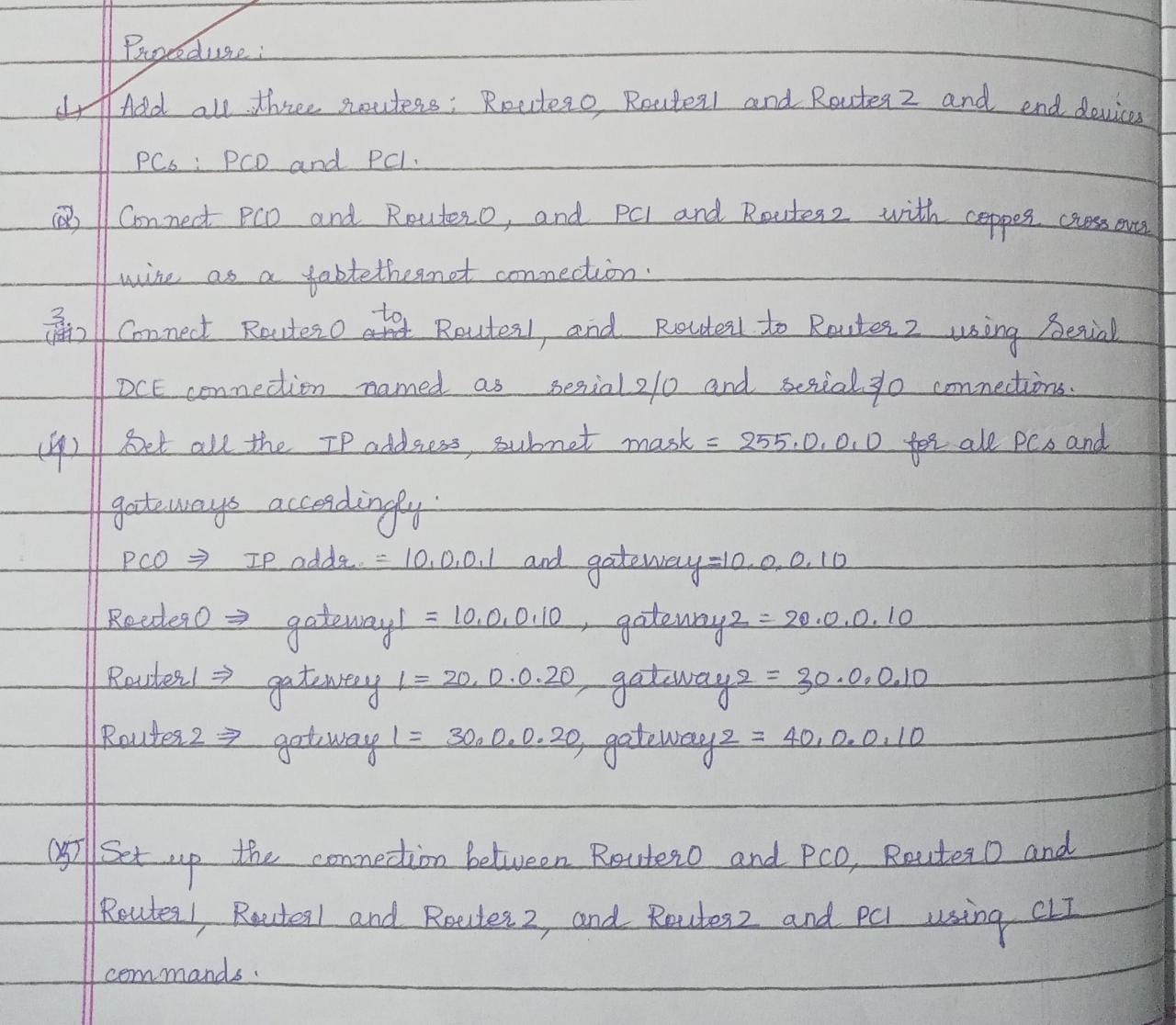
Ping statistics for 40.0.0.1:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

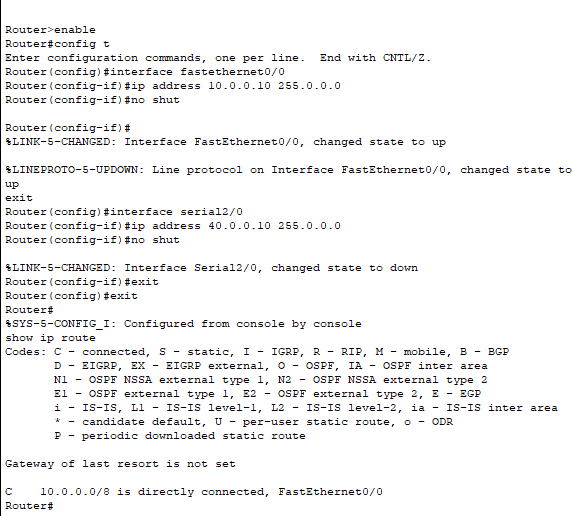
Minimum = 2ms, Maximum = 4ms, Average = 2ms

## LAB 03 : Configuring default route to the Router

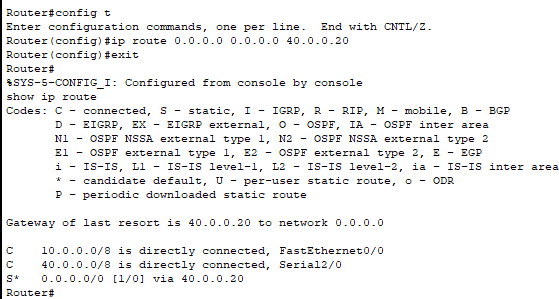




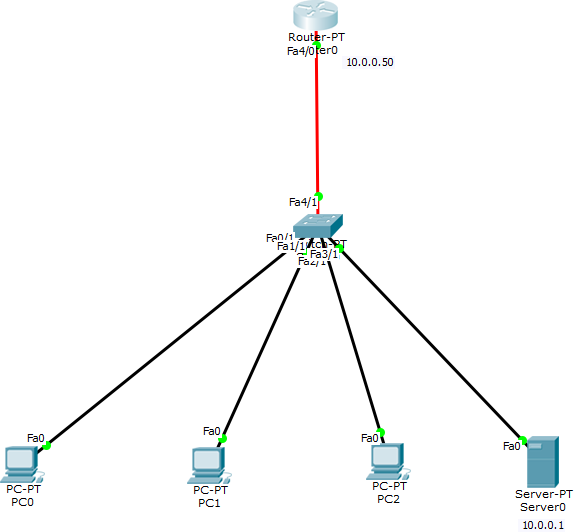
Router0 :

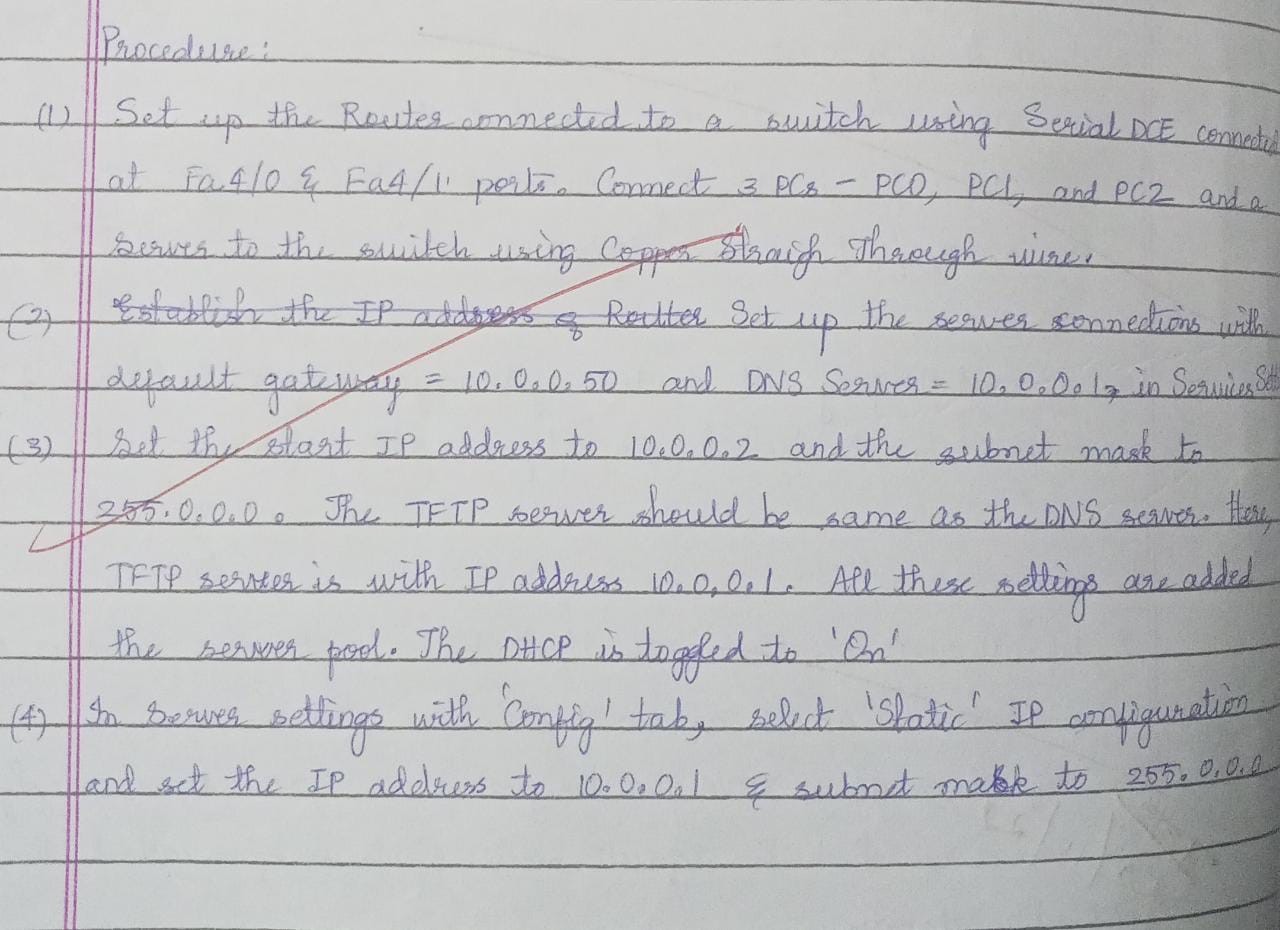


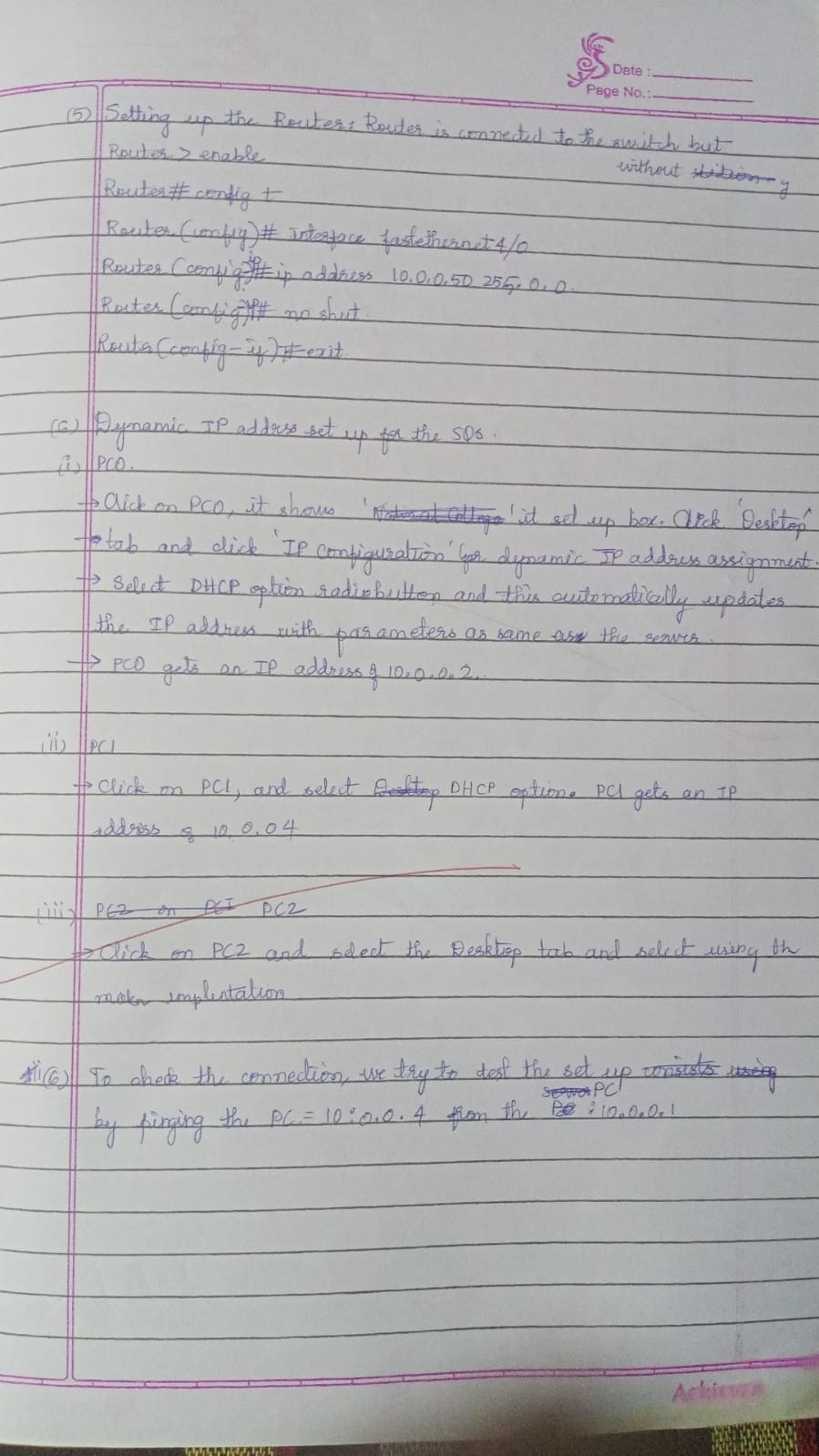
The above is done for Router1 and Router2. Teaching the router about other networks using Default Routing:



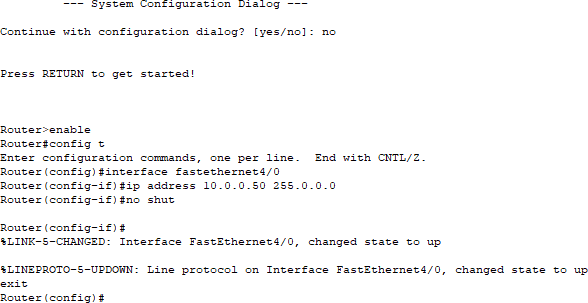
## LAB 04 : Configuring DHCP within a LAN in a packet Tracer



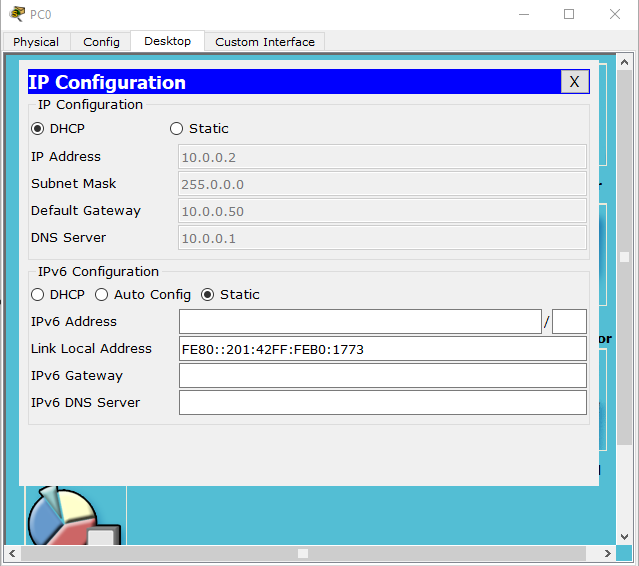




Commands for setting up the router:



Dynamic IP address set up for PCs



Pinging PC2 to PC0

Packet Tracer PC Command Line 1.0 PC>ping 10.0.0.4

Pinging 10.0.0.4 with 32 bytes of data:

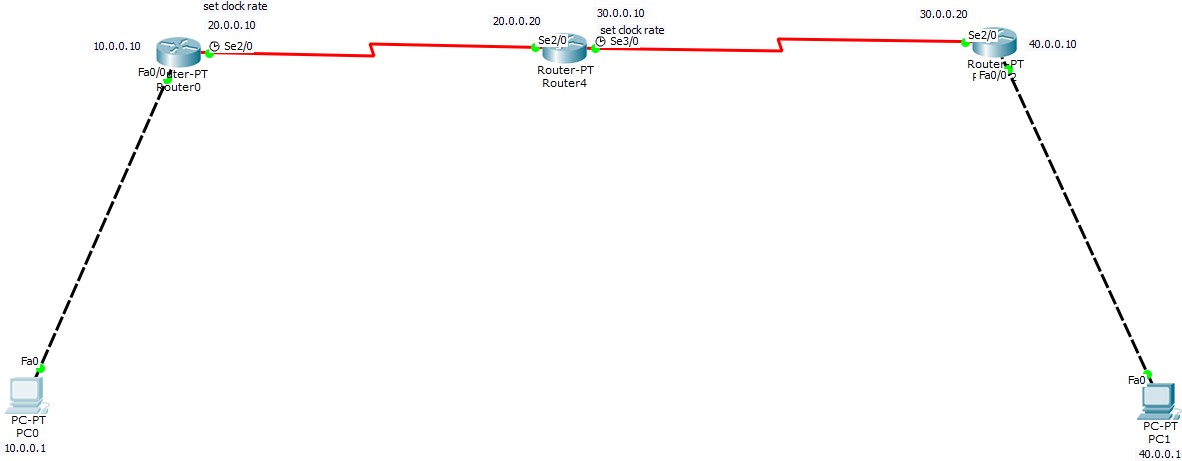
Reply from 10.0.0.4: bytes=32 time=0ms TTL=128 Reply from 10.0.0.4: bytes=32 time=0ms TTL=128 Reply from 10.0.0.4: bytes=32 time=0ms TTL=128 Reply from 10.0.0.4: bytes=32 time=0ms TTL=128

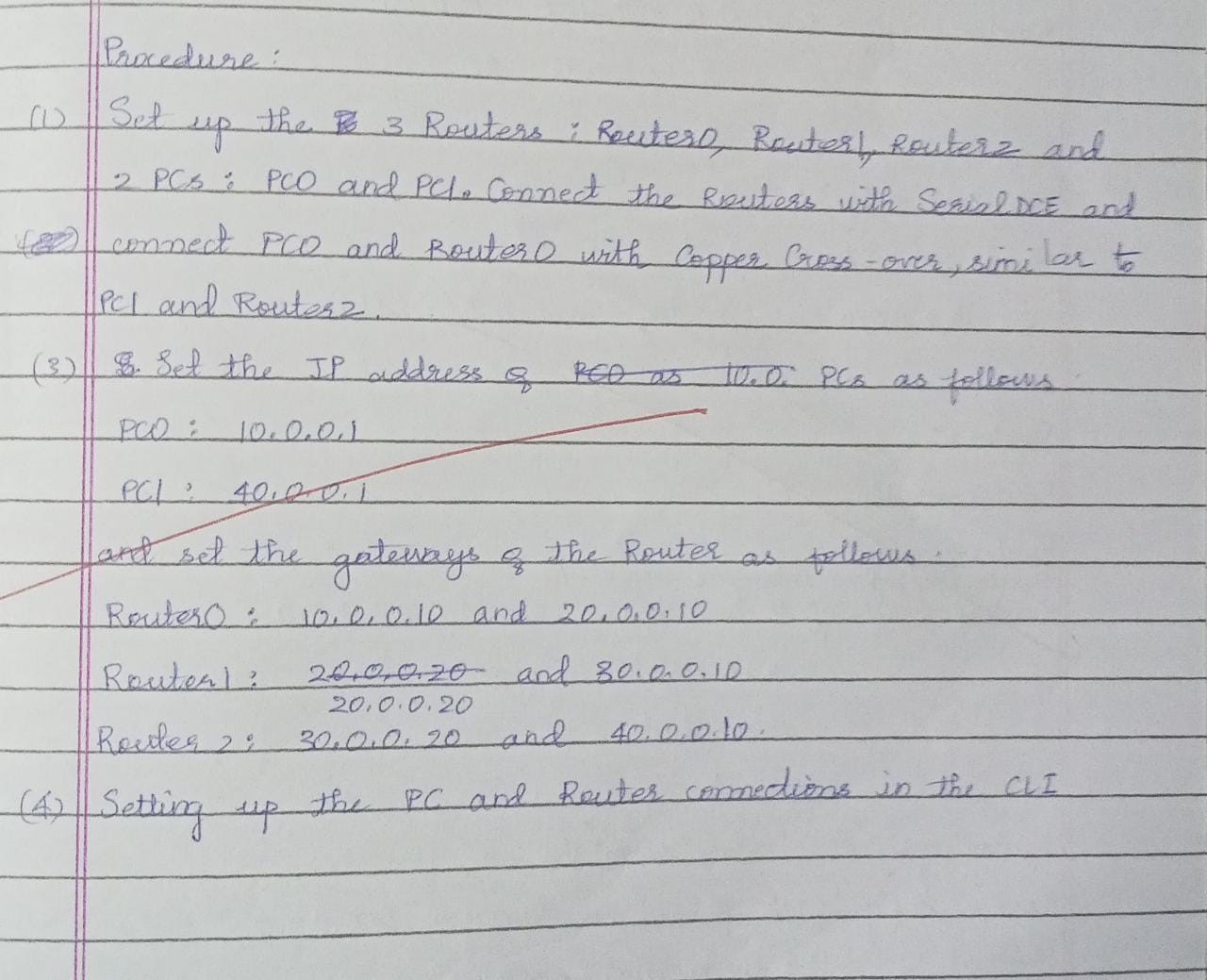
Ping statistics for 10.0.0.4:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

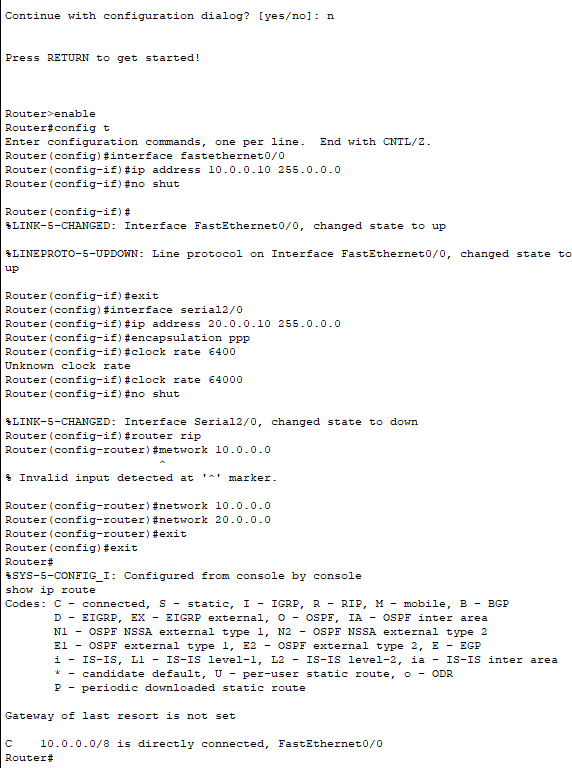
Minimum = 0ms, Maximum = 0ms, Average = 0ms

## LAB 05 : Configuring RIP Routing Protocol in Routers





Setting up the Router settings - Router 0



Similarly, the above commands are executed for Router1 and Router2

Pinging the PCs after all connections Packet Tracer PC Command Line 1.0 PC>ping 40.0.0.1

Pinging 40.0.0.1 with 32 bytes of data:

Request timed out.

Reply from 40.0.0.1: bytes=32 time=12ms TTL=125 Reply from 40.0.0.1: bytes=32 time=6ms TTL=125 Reply from 40.0.0.1: bytes=32 time=14ms TTL=125

Ping statistics for 40.0.0.1:

Packets: Sent = 4, Received = 3, Lost = 1 (25% loss), Approximate round trip times in milli-seconds:

Minimum = 6ms, Maximum = 14ms, Average = 10ms

PC>ping 40.0.0.1

Pinging 40.0.0.1 with 32 bytes of data:

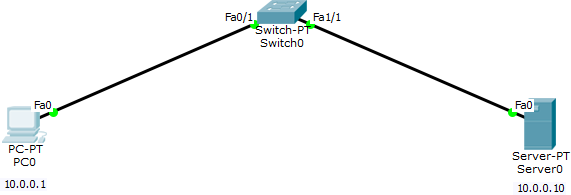
Reply from 40.0.0.1: bytes=32 time=17ms TTL=125 Reply from 40.0.0.1: bytes=32 time=10ms TTL=125 Reply from 40.0.0.1: bytes=32 time=17ms TTL=125 Reply from 40.0.0.1: bytes=32 time=7ms TTL=125

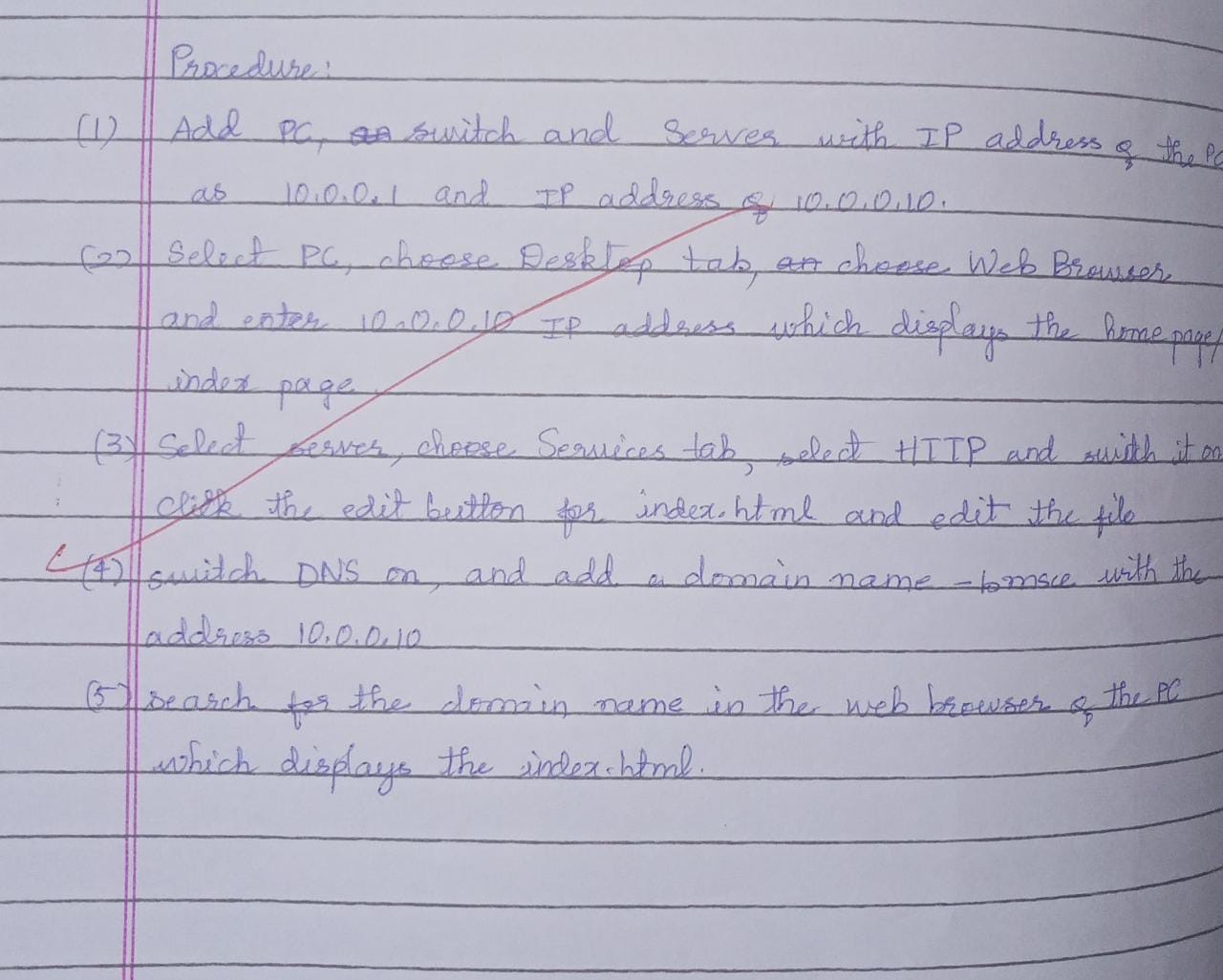
Ping statistics for 40.0.0.1:

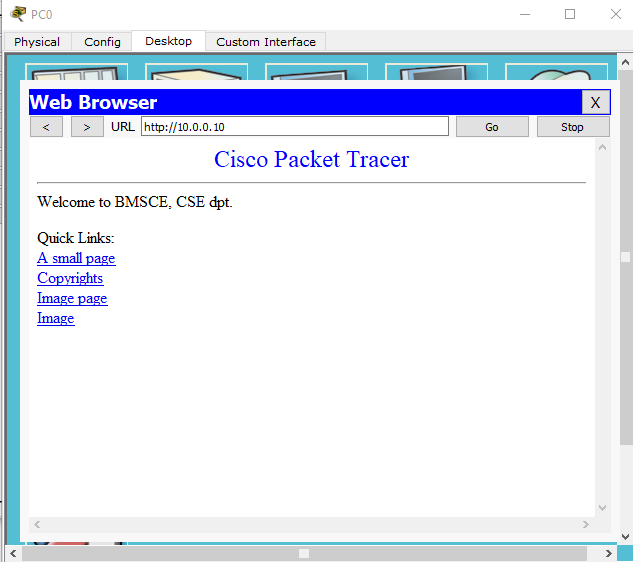
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss), Approximate round trip times in milli-seconds:

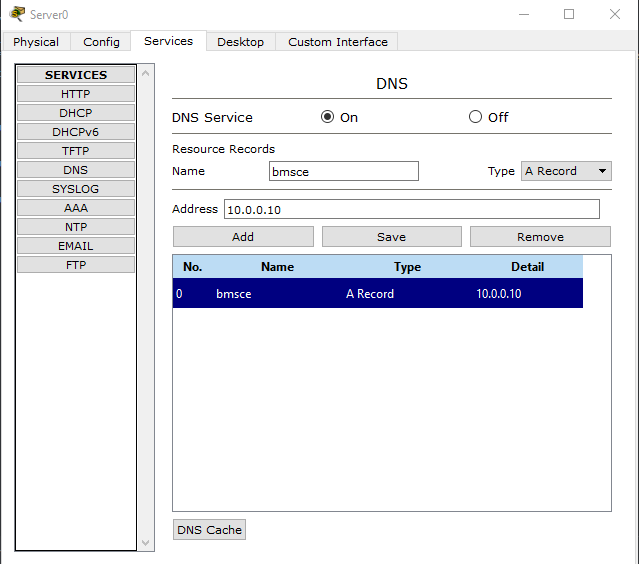
Minimum = 7ms, Maximum = 17ms, Average = 12ms

**LAB 06 : Demonstration of WEB server and DNS using Packet Tracer**









# CYCLE 2

## LAB 07 : Write a program for error detecting code using CRC-CCITT (16-bits).

#CRC at receiver and sender - binary division

def xor(a, b): result = []

for i in range(1, len(b)): if a[i] == b[i]:

result.append('0') else:

result.append('1') return ''.join(result)

def mod2div(dividend, divisor): pick = len(divisor)

tmp = dividend[0 : pick] while pick < len(dividend):

if tmp[0] == '1':

tmp = xor(divisor, tmp) + dividend[pick] else:

tmp = xor('0'\*pick, tmp) + dividend[pick] pick += 1

if tmp[0] == '1':

tmp = xor(divisor, tmp) else:

tmp = xor('0'\*pick, tmp) checkword = tmp

return checkword

def encodeData(data, key): l\_key = len(key)

appended\_data = data + '0'\*(l\_key-1) remainder = mod2div(appended\_data, key) codeword = data + remainder print("Remainder : ", remainder) print("Encoded Data (Data + Remainder) : ",

codeword)

data = "100100"

key = "10001000000100001"

encodeData(data, key)

#Output:

#remainder: 0110010011100110

#encoded data (dataword appended with remainder): 1001000110010011100110

## LAB 08 : Write a program for distance vector algorithm to find suitable path for transmission.

**/\***

**Distance Vector Routing in this program is implemented using Bellman Ford Algorithm:-**

**\*/ #include<stdio.h> struct node**

**{**

**unsigned dist[20]; unsigned from[20];**

**}rt[10];**

**int main() {**

**int costmat[20][20];**

**int nodes,i,j,k,count=0;**

**printf("\nEnter the number of nodes : "); scanf("%d",&nodes);//Enter the nodes printf("\nEnter the cost matrix :\n"); for(i=0;i<nodes;i++)**

**{**

**for(j=0;j<nodes;j++)**

**{**

**scanf("%d",&costmat[i][j]); costmat[i][i]=0;**

**rt[i].dist[j]=costmat[i][j];//initialise the distance equal to**

**cost matrix**

**rt[i].from[j]=j;**

**}**

**}**

**do**

**{**

**count=0;**

**for(i=0;i<nodes;i++)//We choose arbitary vertex k and we calculate the direct distance from the node i to k using the cost matrix**

**//and add the distance from k to node j for(j=0;j<nodes;j++) for(k=0;k<nodes;k++)**

**if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])**

**{//We calculate the minimum distance rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j]; rt[i].from[j]=k;**

**count++;**

**}**

**}while(count!=0); for(i=0;i<nodes;i++)**

**{**

**printf("\n\n For router %d\n",i+1); for(j=0;j<nodes;j++)**

**{**

**printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);**

**}**

**}**

**printf("\n\n");**

**//getch();**

**}**

## OUTPUT:

Enter the number of nodes : 3

Enter the cost matrix : 0 2 7

2 0 1

7 1 0

For router 1

node 1 via 1 Distance 0

node 2 via 2 Distance 2

node 3 via 2 Distance 3

For router 2

node 1 via 1 Distance 2

node 2 via 2 Distance 0

node 3 via 3 Distance 1

For router 3

node 1 via 2 Distance 3

node 2 via 2 Distance 1

node 3 via 3 Distance 0

## LAB 09 : Implement Dijkstra’s algorithm to compute the shortest path for a given topology.

#include<stdio.h>

void dijkstras();

int c[10][10], n, src;

void main() {

int i,j;

printf("\nEnter the num of vertices: \t"); scanf("%d", &n);

printf("\nEnter the cost matrix: \n"); for(i = 1; i <= n; i++) {

for(j = 1; j <= n; j++) { scanf("%d", &c[i][j]);

}

}

printf("\nEnter the source node: \t"); scanf("%d", &src);

dijkstras();

}

void dijkstras() {

int vis[10], dist[10], u, j, count, min; for(j = 1; j <= n; j++) {

dist[j] = c[src][j];

}

for(j = 1; j <= n; j++) { vis[j] = 0;

}

dist[src] = 0;

vis[src] = 1;

count = 1; while(count != n) {

min = 9999;

for(j = 1; j <= n; j++) {

if(dist[j] < min && vis[j] != 1) { min = dist[j];

u = j;

}

}

vis[u] = 1; count++;

for(j = 1; j <= n; j++) {

if(min + c[u][j] < dist[j] && vis[j] != 1) { dist[j] = min + c[u][j];

}

}

}

printf("\nThe shortest distance is: \n"); for(j = 1; j <= n; j++) {

printf("\n%d----->%d = %d", src, j, dist[j]);

|  |  |
| --- | --- |
| } |  |
| } |
| **OUTPUT:** |  |
| Enter the num of vertices: | 4 |
| Enter the cost matrix: |  |
| 0 9999 4 2 |  |
| 1 0 4 2 |  |
| 5 8 0 9999 |  |
| 2 9999 9999 0 |  |
| Enter the source node: | 2 |
| The shortest distance is: |  |
| 2----->1 = 1 |  |
| 2----->2 = 0 |  |
| 2----->3 = 4 |  |
| 2----->4 = 2 |  |

## LAB 10 : Write a program for congestion control using Leaky bucket algorithm.

import time

class Packet:

def init (self, id, size): self.id = id

self.size = size

def getSize(self): return self.size

def getId(self): return self.id

class LeakyBucket:

def init (self, leakRate, size): self.leakRate = leakRate self.bufferSizeLimit = size self.buffer = [] self.currBufferSize = 0

def addPacket(self, newPacket):

if self.currBufferSize + newPacket.getSize() > self.bufferSizeLimit:

print("Bucket is full. Packet rejected.") return

self.buffer.append(newPacket) self.currBufferSize += newPacket.getSize()

print("Packet with id = " + str(newPacket.getId()) + " added to bucket.")

def transmit(self):

if len(self.buffer) == 0:

print("No packets in the bucket.") return

n = self.leakRate

while len(self.buffer) > 0: topPacket = self.buffer[0]

topPacketSize = topPacket.getSize() if topPacketSize > n:

break

n = n - topPacketSize self.currBufferSize -= topPacketSize self.buffer.pop(0)

print("Packet with id = " + str(topPacket.getId()) + " transmitted.")

if name == ' main ':

bucket = LeakyBucket(1000, 10000) bucket.addPacket(Packet(1, 200))

bucket.addPacket(Packet(2, 500))

bucket.addPacket(Packet(3, 400))

bucket.addPacket(Packet(4, 500))

bucket.addPacket(Packet(5, 200)) while True:

bucket.transmit(); print("Waiting for next tick."); time.sleep(1)

## OUTPUT :

**Packet with id = 1 added to bucket. Packet with id = 2 added to bucket. Packet with id = 3 added to bucket. Packet with id = 4 added to bucket.**

**Packet with id = 5 added to bucket. Packet with id = 1 transmitted.**

**Packet with id = 2 transmitted. Waiting for next tick.**

**Packet with id = 3 transmitted. Packet with id = 4 transmitted. Waiting for next tick.**

**Packet with id = 5 transmitted. Waiting for next tick.**

**No packets in the bucket. Waiting for next tick.**

**No packets in the bucket.**

## LAB 11 : 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.

clienttcp.py

from socket import \*

serverName = "10.124.7.76"

serverPort = 12000

clientSocket = socket(AF\_INET, SOCK\_STREAM) clientSocket.connect((serverName, serverPort))

sentence = input("Enter file name: ")

clientSocket.send(sentence.encode()) filecontents = clientSocket.recv(1024).decode() print("From Server: ", filecontents)

clientSocket.close()

servertcp.py

from socket import \*

serverName = "10.124.7.76"

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_STREAM) serverSocket.bind((serverName, serverPort)) serverSocket.listen(1)

print("The server is ready to receive")

while 1:

connectionSocket, addr = serverSocket.accept() sentence = connectionSocket.recv(1024).decode()

file = open(sentence, "r") l = file.read(1024)

print("Recieved from client: ", l)

connectionSocket.send(l.encode()) file.close() connectionSocket.close()

a.txt

hello world

## OUTPUT :

Enter file name: a.txt From server:

The server is ready to receive Received from client: hello world

## LAB 12 : Using UDP sockets, write a client-server program to make client sending the file name and the server to send back the contents of the requested file if present.

udpClient.py

from socket import \*

serverName = "127.0.0.1"

serverPort = 12000

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

sentence = input("Enter file name: ") clientSocket.sendto(bytes(sentence, "utf-8"), (serverName, serverPort)) filecontents, serverAddress = clientSocket.recvfrom(2048)

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

clientSocket.close()

udpServer.py

from socket import \*

serverPort = 12000

serverSocket = socket(AF\_INET, SOCK\_DGRAM) serverSocket.bind(("127.0.0.1", serverPort))

print("The server is ready to receive")

while 1:

sentence, clientAddress = serverSocket.recvfrom(2048)

file = open(sentence, "r") l = file.read(2048)

serverSocket.sendto(bytes(l, "utf-8"), clientAddress) print("Sent back to client: ", l)

file.close()

b.txt

hello world

## OUTPUT :

Enter the file name: b.txt From server:

The server is ready to receive Sent back to client: hello world