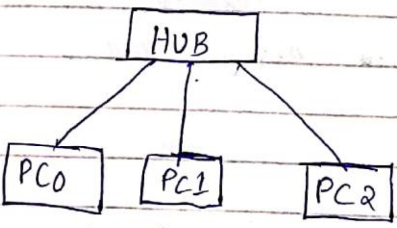
Name: Sai Sriram·V
USN: 1BM18CS140

# LAB-I

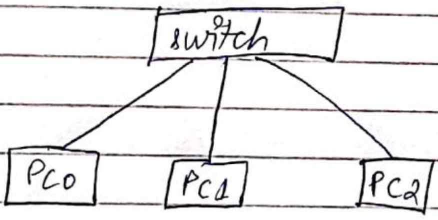
① Procedure :- Step 1) Take 3 PC's and connect them all to 1 hub.

Step 2) Set the IP address for the 3 end devices.
Step 3) Add PDU to each device and run simulation.



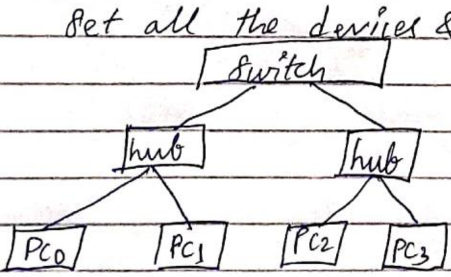
Outcome :- PC0 sends a message which is received by HUB & it is further sent to PC1 and PC2. These end devices may accept or reject the message.

② Procedure :- Same steps as hub topology except instead of hub, switch is used.



Outcome :- The end devices can communicate with each other without the interference of the switch.

③ Procedure :- Set all the devices & connect them to each other.



Outcome:- ( PC0 → PC2 )

PC0 sends message to hub and it inturn sends it to switch and PC1. PC1 rejects the message and switch sends it to hub which it inturn sends it to PC2 and PC3 simultaneously. PC3 rejects the message and PC2 sends an acknowledgement message to PC0.

Name: Sai Sriram.V
USN: 1BM18CS140

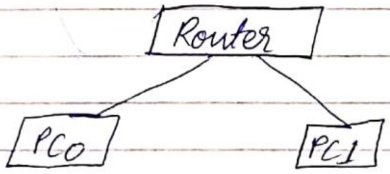
# LAB - II

**PROCEDURE:-** Step 1) Place the PC's and routers and make connections.

Step 2) For the PC's - PC0 and PC1, set IP address and gateway address.

Step 3) Configure the router's IP address same as the respective gateway devices of the desktop.

Step 4) Open desktop control panel and ping the IP address.



**Outcome:-** In total 4 packets sent and 4 packets are received.

So, PC0 replies from 10.0.0.20 every time.

The bytes = 32 times.

① 10ms
② 9ms
③ 7ms
④ 5ms.

DATE : □□ - □□ - □□□□

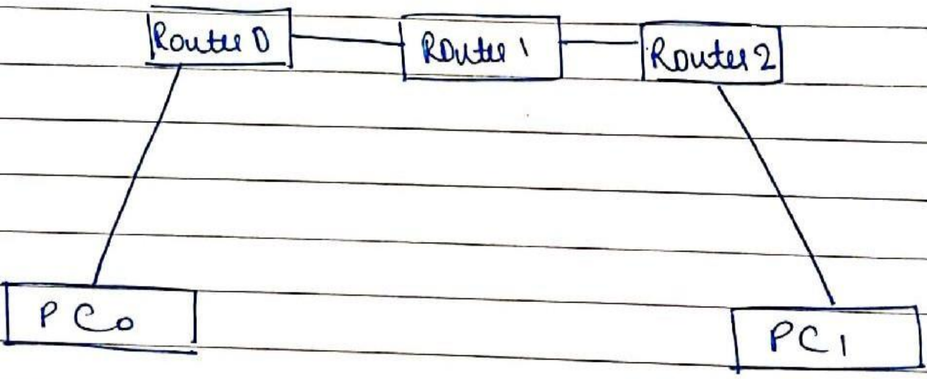
# LAB- 3

## PROCEDURE:

step1: Place the 2PC's and 3 routers and make the connections.

Step2: Configure the PC's and routers IP address and gateway address.

Step 3: Open comand promt & ping the IP address.



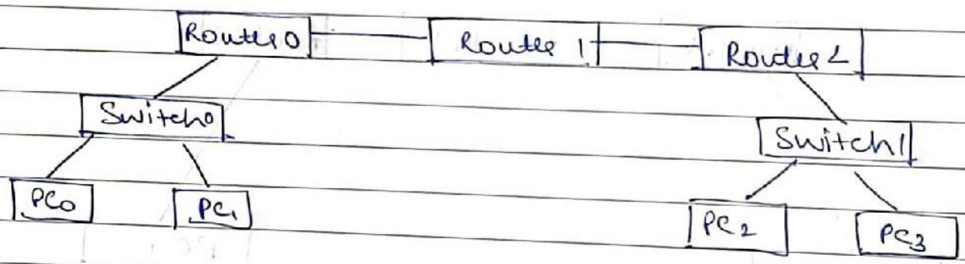
# LAB – 4

PROCEDURE :

Step1: Place the 3 routers, 4 PC'S and 2 switches. Connect 2 PC's to 1 switch each. The three routers must be connected to one another and to the switches.

Step2: Configure the PC's by gateway and IP address. Configure the routers

Step3: Open command prompt and ping the IP address.



OUTCOME:

Each router knows about its immediate neighbouring signal

Any signal can go through destination signal (R0 & R2).

Output before interfacing ⇒ Destination not reachable.

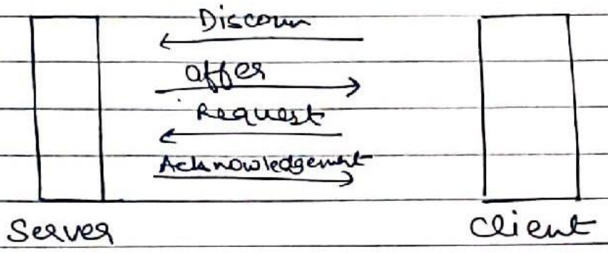
Output after setting ip route for R0 & R2 ⇒

```
Reply   from   40.0.0.21
Reply   from   40.0.0.21
Reply   from   40.0.0.21
Reply   from   40.0.0.21
```

LAB 5

DHCP → Dynamic Host configuration Protocol

It is used to control the network configuration of a host through a remote server. It is a client server model.



A server helps to configure the client automatically. It uses some pre-instructions.

PROGRAM : 6 — ERROR DETECTION USING CRC (16 bits)

CODE

```
import hashlib

def xor (a,b):
    result = []
    for i in range (1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else
            result.append('1')
    result''.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)
```

```
def print_routing_table (self, node, dist, next_ho
    print(f'Routingtable for {node}:')
    print('Dest  \t Cost \t Next Hop')
    for dest, cost in dist.items():
        print(f'{dest} \t {cost} \t
                {next_hop [dest]}')

def start (self):
    pass
```

PROGRAM 7B: DIJKSTRA'S ALGORITHM

CODE

```
import sys

class graph :

    def _init_ (self, vertices):
        self.V = vertices
        self.graph = [[0 for column in
         range(vertices)] for row in range(vertices)

    def printSolution(self, dist):
        print("Vertex \t Distance from source")
        for node in range(self.V):
            print (node, "\t", dist[node])

    def minDistance (self, dist, sptSet):
        min = sys.maxSize
        for v in range (self.V):
            if dist[v] < min and sptSet[v] = False
                min = dist [v]
            min_index = V
    return min_index

def digkstra (self, src) :
    dist = [sys . maxsize] * self . V
    dist [src] = 0
    sptset = [False) * self . V
    for cout in range (self . V) :
        u = self . minDistance (dist, sptset):
        sptset [u] = True
        for v in range (self . V) :
            if self . graph [u][v] > 0 and
              sptset [v] == False and
              dist [v] > dist [u] + self . graph[u][v]:
                  dist [v] = dist [u] + self . graph[u][v]
    self . printSolution (dist)
```

# PROGRAM 8 : LEAKY BUCKET ALGORITHM

CODE

```
import os
clear = lambda : os.system('clear')

class Client :
    def _init_(self, rate=int ,data=[]):
        self.rate =rate
        self.data =data

    def _str_(self):
        return str([str(self.rate),
                    str(self.data)])

class Buffer :

    def _init_(self, buffer_size =int, buffer=[]
        self.buffer-size = buffer_size
        self.buffer= buffer

    def checkstate (self) :
        if len (self.buffer) = = 0 :
            return True

    def _str_(self):
        return str([str (self.buffer_size),
                str(self.buffer)]);

basestate = True
sec = 1
```

```
buffer = Buffer(int(input(" enter buffer size"))
client = Client(int(input("enter client acceptance
                         rate in bps")))
data_to_send = str

while truestate:
    data_to_send = input("enter a string")
    count = 0
    if buffer.checkstate():
        for i in range(0, len(data_to_send)):
~~done~~    if i < client.rate:
                Client.data.append(data_to_send[i])
            else:
                if count < buffer.buffer_size:
                    buffer.buffer.append~~(data_to_send~~
                                    (data_to_send[i])
                    count = len(buffer.buffer)
                else:
                    print("Data loss "+data_to_send[i])

    else:
        j = 0
        for i in range(0, len(data_to_send) +
                        len(buffer.buffer)):
            if i < client.rate:
                if len(buffer.buffer):
                    client.data.append(buffer.buffer[0])
                    del buffer.buffer[0]
                else:
                    client.data.append(data_to_send[j])
                    j + = 1
            else:
                ~~if len(buffer.buffer) < buffer.buffer_size~~
```

```
if {len (buffer.buffer) <= buffer.buffer_size}
    if j< len(data_to_send):
        buffer.buffer.append (data_to_send[j]
        j+ = 1
else
    if j < len(data_to_send):
        print ("Data loss" + data_to_send[i]
        j+ = 1;
```

PROGRAM 9 : TCP/IP-CLIENT/SERVER

CODE

PART 1 — SERVER.PY

```
from socket import *
serverName = "127.0.0.1"
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind((serverName, serverPort))
serverSocket.listen(1)
print("Ready to recieve")
while 1:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    file.open(sentence,"r")
    l = file.read(1024)
    connectionSocket.send(l.encode())
    file.close()
connectionSocket.close()
```

CODE

PART 2 — CLIENT.PY

```
from socket import *

serverName = "127.0.0.1"
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()
```

PROGRAM 10 : UDP SOCKET , SERVER / CLIENT

CODE

PART 1 — UDP SERVER.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 recieve ")
while 1 :
    sentence, addr = serverSocket.recvfrom (2048)
    file = open (sentence, "r ")
    l = file. read (2048)
    serverSocket . sendto (bytes (l, "utf-8"), addr)
    print (" sent back to client", l )
file.close( )
```

CODE

PART 2 — UDP CLIENT.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 , addr = clientSocket.recvfrom(2048)
print ('From server :', filecontents )
clientSocket.close( )
```