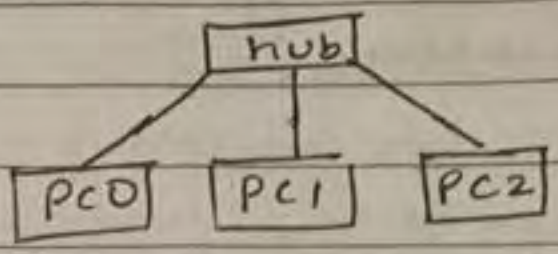
Lab-1

1) Observation :



Step 1 : Place 3 end devices & one hub

Step 2 : connect them using cables

Step 3 :- set their IP address

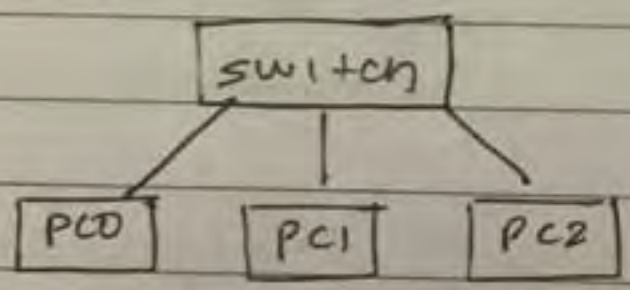
Step 4 : Add simple PDU to each device

Step 5 : run Simulation

OUTCOME

hub receives message from PC0 & sends it to other devices PC1 & PC2 simultaneously & these devices accepts or rejects the message

2)

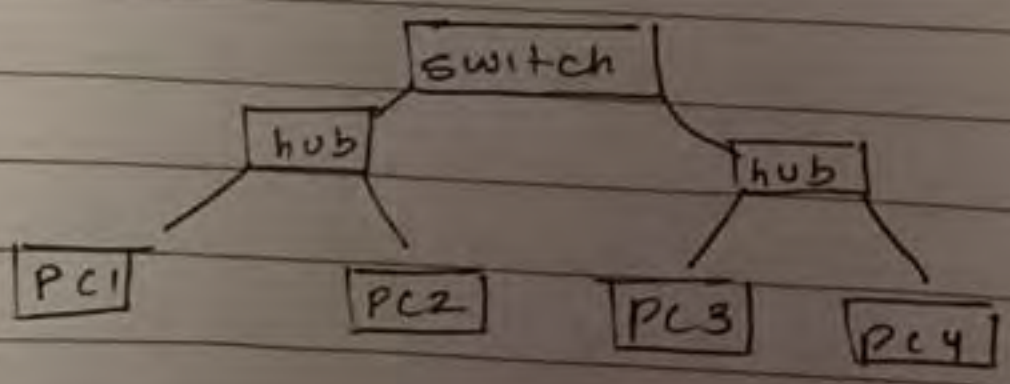


follow the same steps as the hub topology.

OBSERVATION

Though the end devices are connected with each other through a single switch only the pairs of devices can communicate with each other.

3



→ set all the devices by connect them to each other. Then set IP address

→ OUTCOME

PC1 → PC3

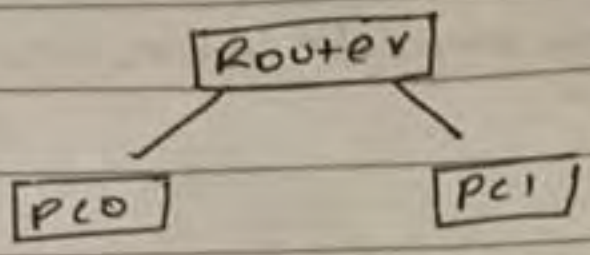
1) PC1 sends message to hub & hub then sends it to PC2 & switch.

2) PC2 rejects the message & switch sends it to the hub.

3) The hub then sends it to PC3 & PC4 simultaneously.

4) PC4 rejects the message & PC3 sends acknowledgement to PC1.

LAB 2:



Step 1 : Place the devices & make connection

Step 2 : Set IP & gateway address of both the devices PC0 by PC1

Step 3 : configure the router IP address same as the respective gateway address of the desktop.

Step 4 : Open desktop control panel & ping the IP address

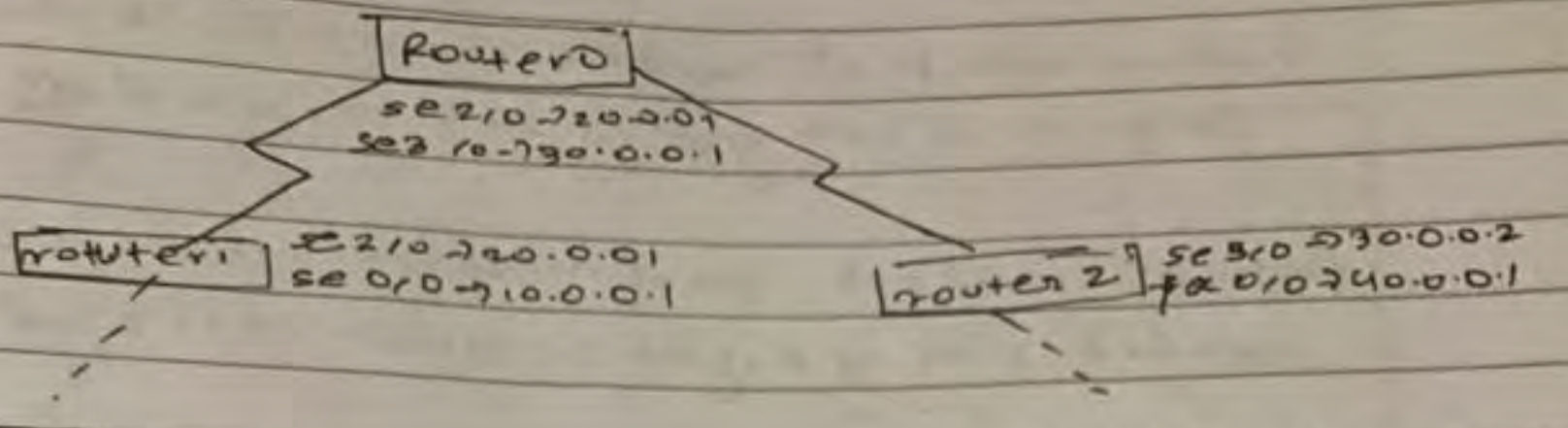
OUTCOME :

PC0 replies from 10.0.0.20 : bytes = 32 time = 10ms
PC0 replies from 10.0.0.20 : bytes = 32 time = 9ms
PC0 replies from 10 : 0.0.20 bytes = 32 time = 7ms
PC0 replies from 10.0.0.20 bytes = 32 time = 5ms

So in total 4 packets sent, 4 received.

Each data packet contains address informations that a router can use to determine if the source & destination are on the same network.

CAB :3:



Observation:

→ each router knows only about its immediate neighbouring signals

→ It doesn't know anything beyound that.

→ R1 knows about 10.0.0.0 by 20.0.0.0

→ R2 knows about 30.0.0.0 by 40.0.0.0

→ R0 knows about 20.0.0.0 by 30.0.0.0

→ To know about further signal, it should go beyound its end points

Command in CLI (for Router 2)

Router>enable

Router# config terminal

Router (config) # ip route 20.0.00 255.0.0.0 30.0.0

Router (config) # ip route 10.0.0.0 255.0.0.10 30.0.0

Router (config) # exit

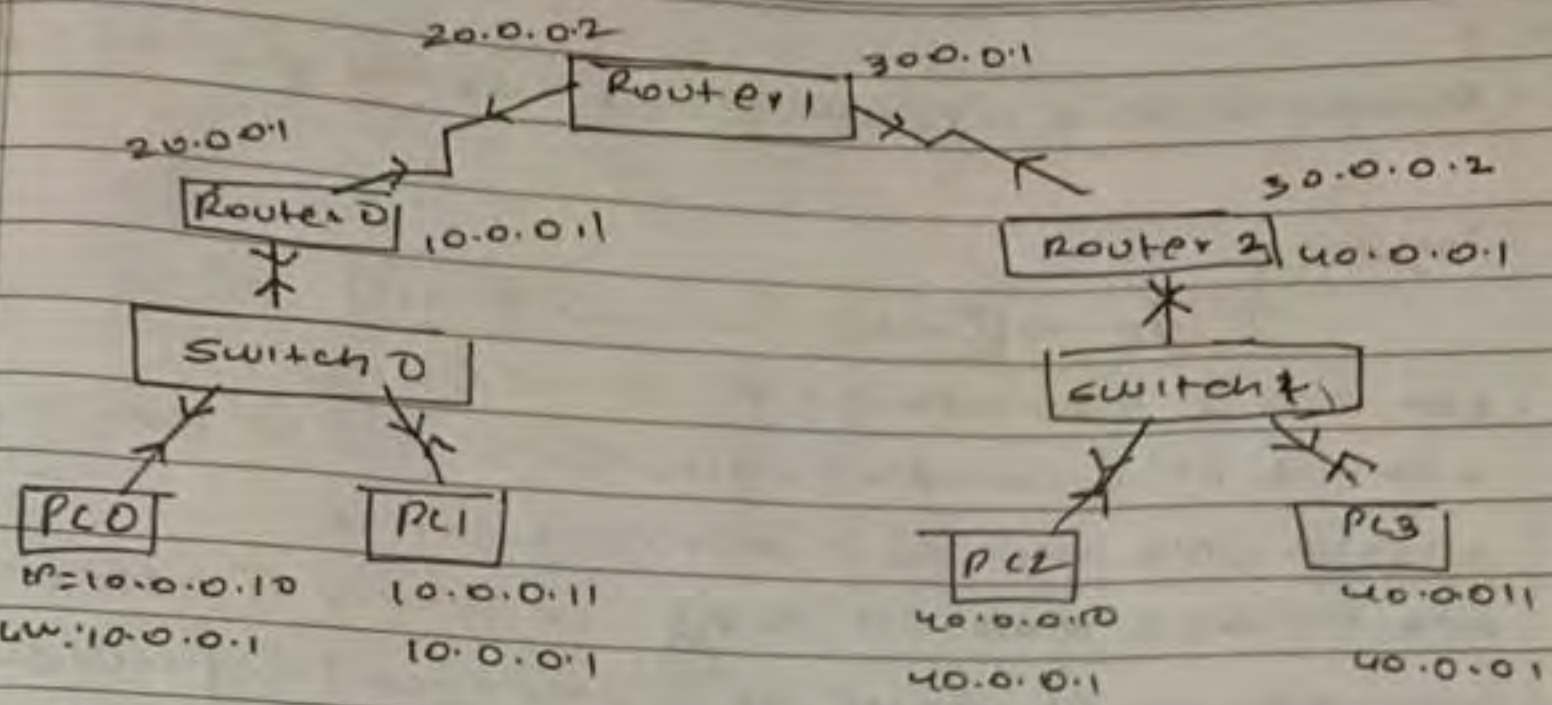
Router # show ip router

Destination host unreachable: means that your computer is not getting an IP address from router.

Request timed out: Error message means the host we are pinging might be: down or it's not known

Reply from 40.0.0.10: bytes=32 time=16ms TTL=125 means that the ping command from 10.0.0.10 to 40.0.0.10 is sucessful.

LAB 4:



Observation

→ each router knows only about its immediate neighboring signals

→ Both R1 & R2 was made to let any signals go through the destination signals

→ first we check the router net connected & connect them statically in the CLI.

CLI for Router 1

ip route 10.0.0.0 255.0.0.0 20.0.0.1

ip route 40.0.0.0 255.0.0.0 30.0.0.2

→ CLI for router 0

ip route 0.0.0.0 0.0.0.0 20.0.0.2

→ cli for router 2

ip route 0.0.0.0 0.0.0.0 30.0.0.1

output before interfacing R0 & R2.

∴ Destination not reachable.

PUT

Replay from 40.0.0.21

Replay from 40.0.2.21

Replay from 40.0.0.21

Replay from 40.0.0.21

LAB 5

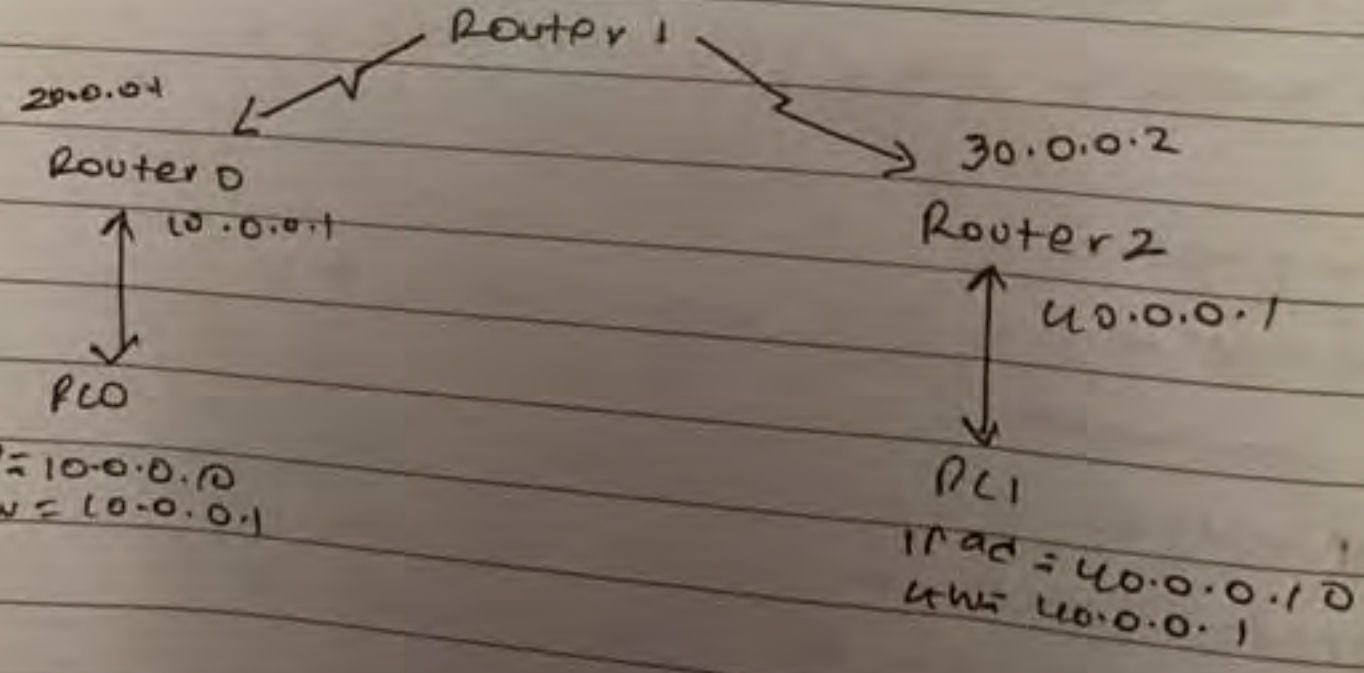
1) Demonstration of WEB server & DNS using packet tracer

topology

PC ⟷ Switch ⟷ server

1. set the ip address for PC server
2. set the DNS server configuration in PC0 config setting
3. enable DNS service in server → services
4. web Browser from PC0 using the server IP address assigned which shows the search for the particular IP addr PC → Desktop → web browser → "enter URL"
5. we can add by edd the server page by server → services → DNS
6. then we can browse from PC0 why newly added Domain
7. By going : server → services → HTTP we can edit & add files.

2) Configuring RIP routing protocol in router topologys:



1. RIP protocal collects information of other router from their neighbours.

2. configure router using RIP
   R0 #interface serial 2/0
   # encapsulation PPP
   # clock rate 64000
   # exit
   # router rip
   # network 10.0.0.0
   # network 10.0.0.0
   # exit

3. R1: #interface serial 2/0
   # encapsulation PPP
   # exit
   # router rip
   # network 20.0.0.0
   # network 30.0.0.0
   # exit
   R0 some for R1 & R2 with network
   → 20.0.0.0, 30.0.0.0 for R1
   → 30.0.0.0, 40.0.0.0 for R2

4. Once config is done, the packets are ready to send:
   ie ping PC1 from PC0
   ping 40.0.0.10

Nol-e : Ro(config router) # version 2 for configuring router by specifying the type routing information protocal.

5 In such a protocal the router will know about other router through in neighbourring router that are connected directly.

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)
```

```
    codeword = data + remainder
    return codeword

def decodeData (code, key):
    remainder = mod2div (code, key)
    return remainder

data = input("enter data :")
print ("dataword :" + str (data))

key = "10001000000100001"
print ("generating polynomial" + key)
codeword = encodeData(data, key)
print ("Checksum :", codeword)
print (" transmitted codeword " + str(codeword))
code = input ("enter transmitted codeword: ")

recieved_data = int (decodeData (code, key))

if recieved_data == 0 :
    print ("no error")
else:
    print ("error")
    print (recieved_data)
```

COMPUTER NETWORKS LAB

VIVEK RAJEEV

Date __/__/__
Page _____
SPLASH

PROGRAM : 7A – DISTANCE VECTOR ALGORITHM

CODE

```
class Topology :

    def __init__(self, array_of_points):
        self.nodes = array_of_points
        self.edges = []

    def add_direct_connection (self, p1, p2, cost):
        self.edges.append ((p1, p2, cost))
        self.edges.append ((p2, p1, cost))

    def distance_vector_routing (self):
        import collections
        for node in self.nodes :
            dist = collections.defaultdict(int)
            next_hop = { node : node }
            for other_node in self.nodes :
                if other_node != node :
                    dist[other_node] = 100000000

        for i in range (len(self.nodes) - 1):
            for edge in self.edges :
                src, dest, cost = edge
                if dist[src] + cost < dist[dest]:
                    dist[dest] = dist[src] + cost
                    if src == node :
                        next_hop[dest] = dest
                    elif src in next_hop :
                        next_hop[dest] = next_hop[src]
        self.print_routing_table(node, dist, next_hop)
        print()
```

```
def print_routing_table(self, node, dist, next_hop):
    print(f'Routing table 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 dijkstra(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)
```

CN - LAB

VIVEK RAJEEV

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 basestate:
    data_to_send = input("enter a string")
    count = 0
    if buffer.checkstate():
        for i in range(0, len(data_to_send)):
    else    if i < client.rate:
                Client.data.append(data_to_send[i])
            else:
                if count < buffer.buffer_size:
                    buffer.buffer.append(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
```

```
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[j])
        j += 1;
```

CN - LAB

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()
```

CN – LAB

IBM18CS142

Date / /

Page

SPLASH

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()
```