CENTRAL UNIVERSITY OF HARYANA

**Department of Computer Science & Engineering under SOET**



**COMPUTER NETWORKS LAB**

**PRACTICAL FILE**

Submitted by

Rachit Sharma

Roll No- 191927

Submitted to

Dr. Benay Kumar Ray

Assistant Professor

Central University of Haryana (SOET)

**Practical-1 :** Write a client-server program which displays the server machine's date and time on the client machine.

*import* socket;

*import* time; s=socket.socket();

*#By default IPV4 and TCP connection*

s.bind(('192.168.0.103',9999)); *#Binding to local pc*

s.listen(2);*#Asking For Connections*

print("Waiting For Connections");

*while* True :

c1 , addr=s.accept()

*#Accepting The Connections*

print("Connected With ",addr);

current\_Time = time.ctime(time.time()) + "\r\n" name=c1.recv(1024).decode()

print("You are connected to person",name) c1.send(bytes(current\_Time,'utf-8')) *#Sending The Time*

c1.close()

# CODE-:

**Server Side-:**

# Client Side-:

*import* socket; c=socket.socket();

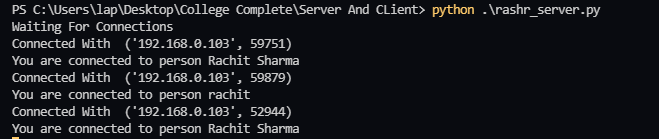
c.connect(('192.168.0.103',9999)) #Connection To Server name=input("Enter Your Name")

c.send(bytes(name,'utf-8'))

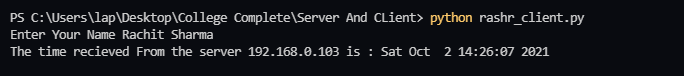
print("The time recieved From the server "+c.getsockname()[0]+" is :",end=" ") print(c.recv(1024).decode()) #Recieving The Time And Printing c.close();

**OUTPUT-:**

# Server Output-:



**Client Output-:**



**Practical-2:** Write a client-server program to create an application for chat using TCP.

# CODE

**Server Code-:** *import* socket; *import* threading;

clients=[] names=[]

server=socket.socket(); server.bind(('localhost',55445)); server.listen()

def broadcast(message): *#Sending Message To All Clients for* client *in* clients:

client.send(message)

def message\_recieve(client): *#Recieving Message From Each Client while* True:

*try*:

message=client.recv(1024) broadcast(message)

*except*:

index=clients.index(client) clients.remove(client) client.close() Name=names[index]

broadcast('{} left the chat'.format(Name).encode('ascii')) names.remove(Name)

*break*

def newClient(): *#Recieving New CLients while* True:

client,addr=server.accept() print("Connected with ",addr)

client.send("Name".encode('ascii')) name=client.recv(2024).decode('ascii') names.append(name) clients.append(client)

print("The Name of Client is ",name)

broadcast('{} joined the chat'.format(name).encode('ascii')) client.send("Connected to server".encode('ascii'))

thread=threading.Thread(target=message\_recieve,args=(client,)) *#BY Threading Recie*

thread.start()

print("Generated the server") newClient()

# Client Code-:

*import* socket;

*import* threading

name=input("Enter Your Name for the Chat ")

client=socket.socket() client.connect(('localhost',55445))

def receiveMessage(): #Recieving Message From Server

*while* True:

*try*:

message= client.recv(1024).decode('ascii')

*if* message=="Name" : client.send(name.encode('ascii'))

*else* :

print(message)

*except*:

print("Error Encountered") client.close()

*return*; *break*;

def write():

*while* True:

message = '{} : {}'.format(name,input('')) client.send(message.encode('ascii'))

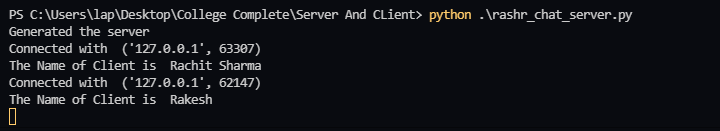
#Recieving And Sending Message Simaltaneously Using Threading

message\_recieve=threading.Thread(target=receiveMessage) message\_recieve.start()

message\_send=threading.Thread(target=write) message\_send.start()

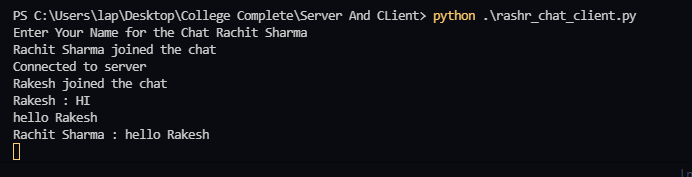
**OUTPUT-:**

# Server Output-:





**Client 1-:**



# Client 2-:

**Practical-3:** Write a client-server program to create an application for chat using TCP.

# CODE

**Server Code-:** *import* socket; *import* threading;

clients=[] names=[]

server=socket.socket(); server.bind(('localhost',55445)); server.listen()

def broadcast(message): *#Sending Message To All Clients for* client *in* clients:

client.send(message)

def message\_recieve(client): *#Recieving Message From Each Client while* True:

*try*:

message=client.recv(1024) broadcast(message)

*except*:

index=clients.index(client) clients.remove(client) client.close() Name=names[index]

broadcast('{} left the chat'.format(Name).encode('ascii')) names.remove(Name)

*break*

def newClient(): *#Recieving New CLients while* True:

client,addr=server.accept() print("Connected with ",addr)

client.send("Name".encode('ascii')) name=client.recv(2024).decode('ascii') names.append(name) clients.append(client)

print("The Name of Client is ",name)

broadcast('{} joined the chat'.format(name).encode('ascii')) client.send("Connected to server".encode('ascii'))

thread=threading.Thread(target=message\_recieve,args=(client,)) *#BY Threading Recie*

thread.start()

print("Generated the server") newClient()

# Client Code-:

*import* socket;

*import* threading

name=input("Enter Your Name for the Chat ")

client=socket.socket() client.connect(('localhost',55445))

def receiveMessage(): #Recieving Message From Server

*while* True:

*try*:

message= client.recv(1024).decode('ascii')

*if* message=="Name" : client.send(name.encode('ascii'))

*else* :

print(message)

*except*:

print("Error Encountered") client.close()

*return*; *break*;

def write():

*while* True:

message = '{} : {}'.format(name,input('')) client.send(message.encode('ascii'))

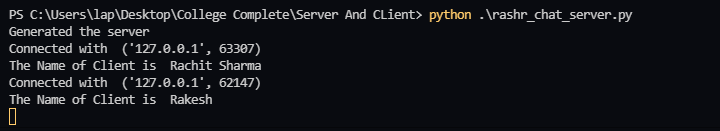
#Recieving And Sending Message Simaltaneously Using Threading

message\_recieve=threading.Thread(target=receiveMessage) message\_recieve.start()

message\_send=threading.Thread(target=write) message\_send.start()

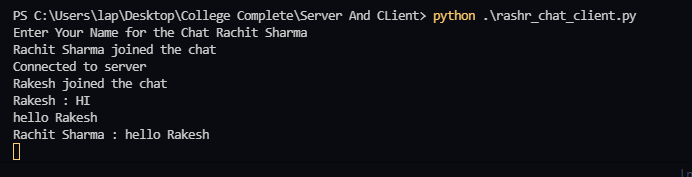
**OUTPUT-:**

# Server Output-:





**Client 1-:**



# Client 2-:

**Practical-4: Chat Server Application using UDP Code-:**

**Server Code-:** *import* socket *import* threading *import* queue

def RecvData(sock,recvPackets):

*while* True:

data,addr = sock.recvfrom(1024) recvPackets.put((data,addr))

def RunServer():

host = socket.gethostbyname(socket.gethostname()) port = 5000

print('Server hosting on IP-> '+str(host))

s = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM) s.bind((host,port))

clients = set() recvPackets = queue.Queue()

print('Server Running...')

threading.Thread(target=RecvData,args=(s,recvPackets)).start()

*while* True:

*while* not recvPackets.empty(): data,addr = recvPackets.get() *if* addr not in clients:

clients.add(addr)

*continue*

clients.add(addr)

data = data.decode('utf-8')

*if* data.endswith('qqq'): clients.remove(addr) *continue*

print(str(addr)+data)

*for* c *in* clients:

*if* c!=addr:

s.sendto(data.encode('utf-8'),c)

s.close()

RunServer()

# Client Code-:

*import* socket *import* threading *import* random

*#Client Code*

def ReceiveData(sock):

*while* True:

*try*:

data,addr = sock.recvfrom(1024) print(data.decode('utf-8'))

*except*:

*pass*

def RunClient(serverIP):

host = socket.gethostbyname(socket.gethostname()) port = random.randint(5000,15000)

print('Client IP->'+str(host)+' Port->'+str(port)) server = (str(serverIP),5000)

s = socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM) s.bind((host,port))

name = input('Please write your name here: ')

*if* name == '':

name = 'Guest1' print('Your name is:'+name)

s.sendto(name.encode('utf-8'),server) threading.Thread(target=ReceiveData,args=(s,)).start() *while* True:

data = input()

*if* data == 'qqq':

*break*

*elif* data=='':

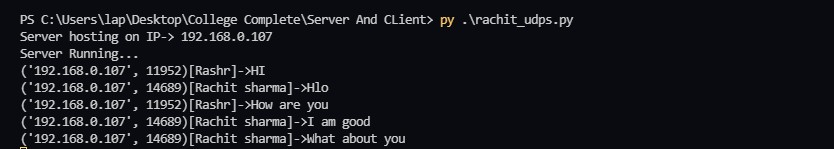
*continue*

data = '['+name+']' + '->'+ data s.sendto(data.encode('utf-8'),server)

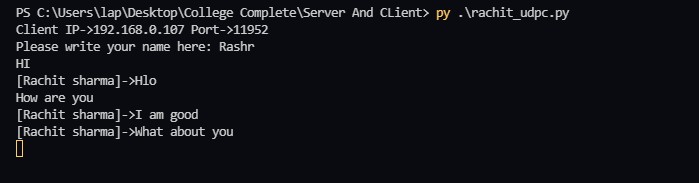
s.sendto(data.encode('utf-8'),server) s.close()

RunClient(socket.gethostbyname(socket.gethostname()))

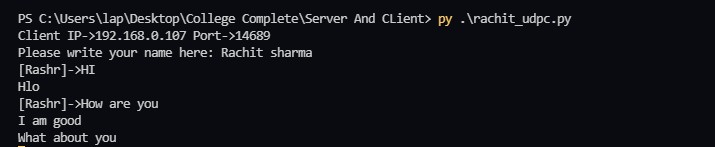
# Output-: Server-:



**Client 1-:**



# Client 2-:



**Practical-5:** Write a program for creating a RMI application. A RMI application must be divided into two parts,Client program and Server program. A Server program creates some remote object, making their references available for the client to invoke methods on it. A Client program makes requests for

remote objects on the server and invokes methods on them.

# Code-: Server Side-:

*import* Pyro4

@Pyro4.expose

class GreetingMaker(object): def get\_fortune(self, name):

*return* "Hello, {0}. Here is your fortune message:\n" \ "Your lucky number is 12345.".format(name)

@Pyro4.expose class sum(object):

def findSum(self,a,b):

*return* a+b;

*# make a Pyro daemon* daemon = Pyro4.Daemon() *# find the name server* ns = Pyro4.locateNS()

*# register the greeting maker as a Pyro object* uri = daemon.register(GreetingMaker) uri2=daemon.register(sum)

*# register the object with a name in the name server* ns.register("example.greeting", uri) ns.register("example.sumfinder", uri2)

print("Ready.")

*# start the event loop of the server to wait for calls*

daemon.requestLoop()

**Client 1 Code-:**

*import* Pyro4

name = input("What is your name? ").strip()

*# use name server object lookup uri shortcut* greeting\_maker = Pyro4.Proxy("PYRONAME:example.greeting") sums=Pyro4.Proxy("PYRONAME:example.sumfinder")

print(sums.findSum(4,5)) print(greeting\_maker.get\_fortune(name))

# Client 2 Code-:

*import* Pyro4;

sumfind=Pyro4.Proxy("PYRONAME:example.sumfinder")

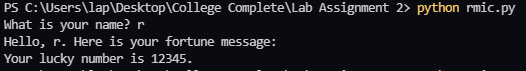
a=int(input("Enter first value for sum ")) b=int(input("Enter second value for sum "))

print("The sum of 2 input is :",sumfind.findSum(a,b))

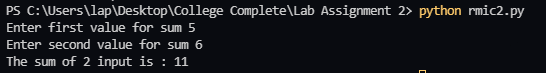
**Output-: Server Side-:**



# Client 1-:



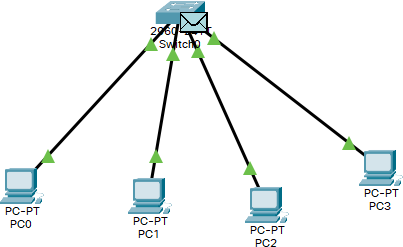
**Client 2-:**



**Practical 6-:** Compare and contrast Network Topologies (Star, Mesh, Ring, Bus) using Cisco Packet Tracer.

# Solution 1-:

**Star Topology-:** Star topology is a network topology in which each network component is physically connected to a central node such as a router, hub or switch. In a star topology, the central hub acts like a server and the connecting nodes act like clients.



**Star Topology**

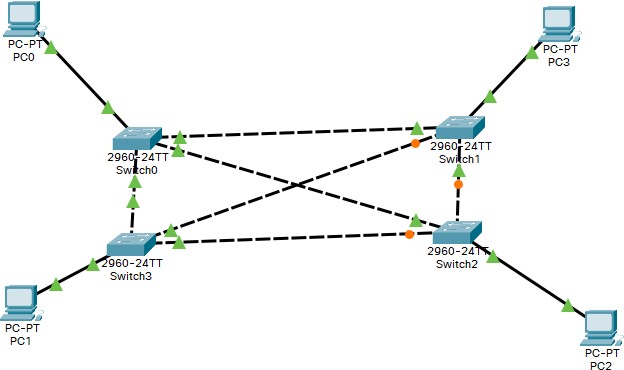
# Advantages-:

* Centralized management of the network, through the use of the central computer, hub, or switch.
* Easy to add another computer to the network.
* If one computer on the network fails, the rest of the network continues to function normally.

# Disadvantages-:

* May have a higher cost to implement, especially when using a switch or router as the central network device.
* The central network device determines the performance and number of nodes the network can handle.
* If the central computer, hub, or switch fails, the entire network goes down and all computers are disconnected from the network.

**Mesh Topology-:** In a mesh topology there is no central connection point. Instead, each node is connected to at least one other node and usually to more than one. Each node is capable of sending messages to and receiving messages from other nodes. The nodes act as relays, passing on a message towards its final destination.



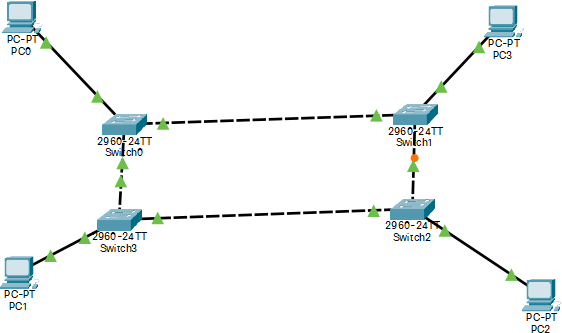
# Mesh Topology Advantages-:

* Messages can be received more quickly if the route to the intended recipient is short
* Due to multiple connections mean each node can transmit to and receive from more than one node at the same time
* New nodes can be added without interruption or interfering with other nodes

# Disadvantages-:

* In full mesh networks can be impractical to set up because of the high number of connections needed
* Large Number of connections require a lot of maintenance
* Cost to apply this topology is also very high

**Ring Topology-:** A ring topology is a network configuration where device connections create a circular data path. Each networked device is connected to two others, like points on a circle. Together, devices in a ring topology are referred to as a ring network.



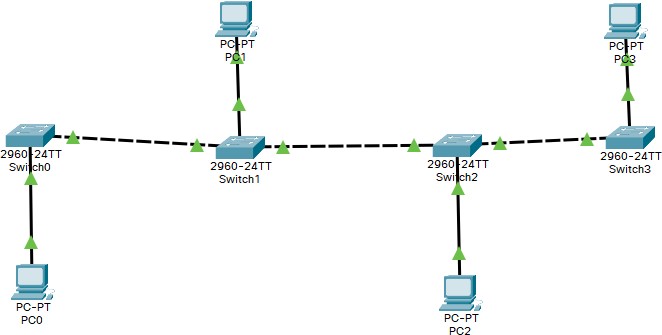
# Ring Topology Advantages-:

* All data flows in one direction, reducing the chance of packet collisions.
* A network server is not needed to control network connectivity between each workstation.
* Data can transfer between workstations at high speeds.
* Additional workstations can be added without impacting performance of the network.

# Disadvantages-:

* All data being transferred over the network must pass through each workstation on the network, which can make it slower than a star topology.
* The entire network will be impacted if one workstation shuts down.
* The hardware needed to connect each workstation to the network is more expensive.

**Bus Topology-:** Bus topology is a specific kind of network topology in which all of the various devices in the network are connected to a single cable or line. In general, the term refers to how various devices are set up in a network.



# BUS Topology Advantages-:

* It is very simple to design.
* Require less cabling compared to other topologies.
* Each to implement for small networks.
* It is easy to expand by simply joining two cables together.
* Very cost-effective.

# Disadvantages-:

* The network stands on a single cable. So, if any damage caused to this cable the whole network falls.
* As the traffic is shared by all the nodes in the network, the performance of the network decreases as the traffic increases.
* It is difficult to find the flaws and faults in the network connected with this method.
* Packet loss is very high.
* This topology is very slow compared to other topologies.

**Practical-7** Configure Enterprise Network and show different subnet, private network using Cisco Packet Tracer. Also use different routing protocols used for communication. Assume here that Enterprise have four branches and is situated at four different places.

# Answer-:

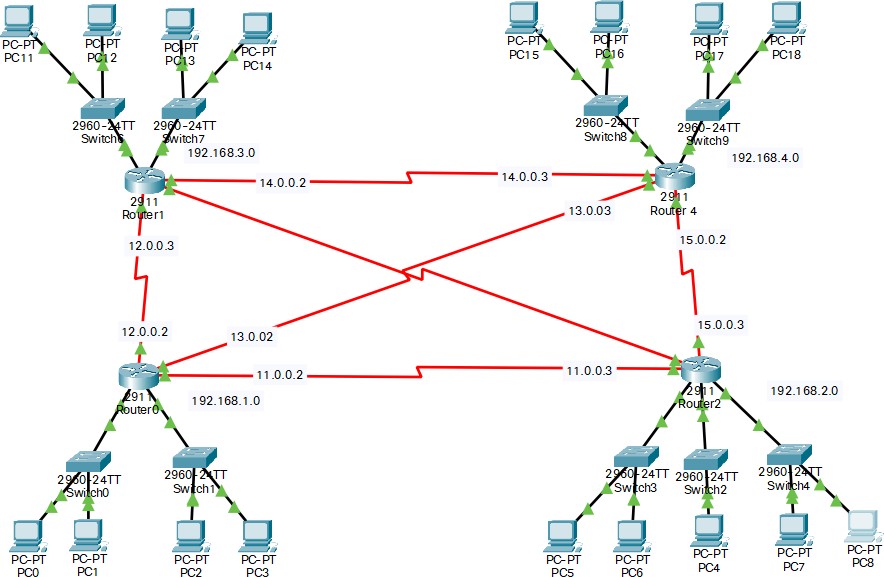
**Enterprise Network** :- An enterprise network is the backbone for facilitating an organization’s communications and connecting computers and devices throughout departments.

An enterprise network environment is usually configured to facilitate access to data and insight into analytics.

Enterprise networking refers to the physical, virtual and logical design of a network, and how the various software, hardware and protocols work together to transmit data.

It is made up of Routers, Switches and Wireless Access Points

# Enterprise network using Static Routing Protocol :-

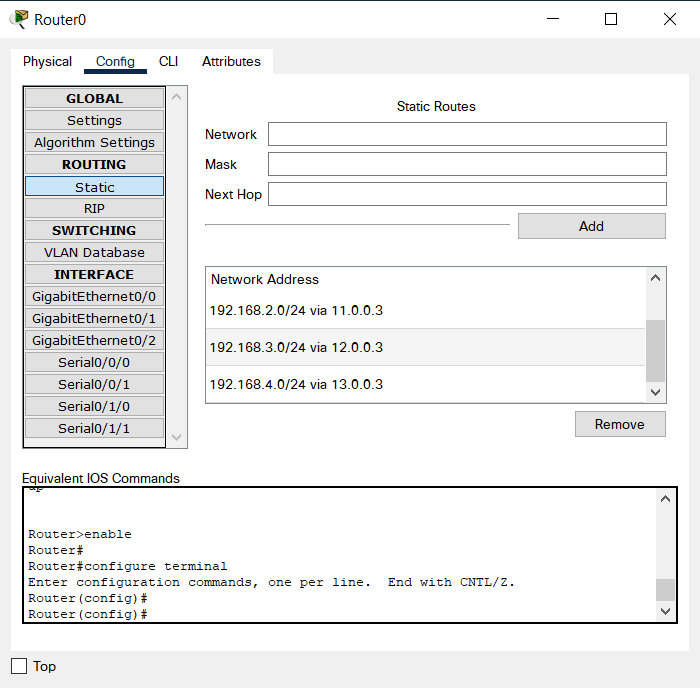


This enterprise network comprises of 4 routers each at different place and all are connected through Mesh Topology.

Communication in this network is govern through Static Routing.

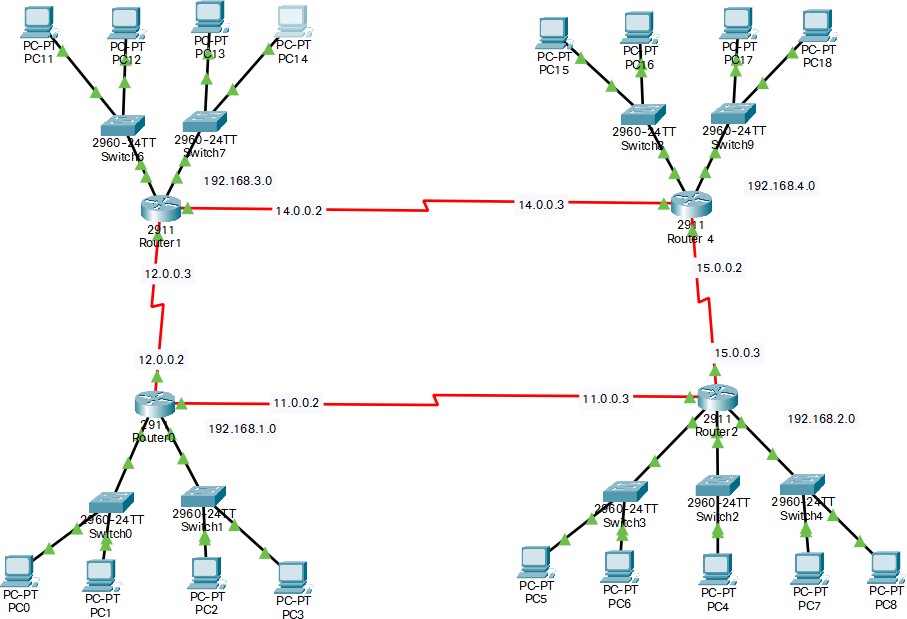
Here, In three of the Routers, the network is divided in 2 subnets having subnet mask 255.255.255.128 And, In one of the Router, the network is divided into 4 subnets having subnet mask 255.255.255.192

# Configuration of Router0 :-



In Static Routing, We have to provide the routes of the networks through the next hop. So, The Routes are fixed for a given Network address.

# Enterprise Network using Dynamic Routing :-

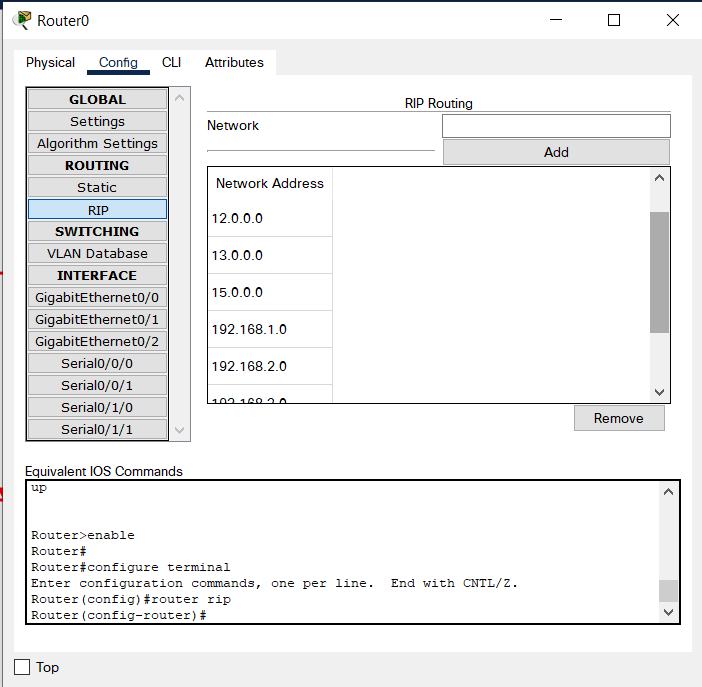


This enterprise network comprises of 4 routers each at different place and all are connected through RingTopology.

Communication in this network is govern through Dynamic Routing. Here, In three of the Routers, the network is divided in 2 subnets.

And, In one of the Router, the network is divided into 4 subnets.

# Configuration of Router0 :-



In Dynamic Routing, We have to provide all the available addresses in the Enterprise Network. The Routing Algorithm decides the path of the message.

It follows the Shortest path to the destination address. Unlike, Static Routing, the paths are not fixed here.

**Practical-8 :** Write a program to calculate checksum based on the given binary strings of 8 bits (data) which is sent with the data as redundant bits. This data + checksum is received at receiver end and checksum is calculated again, if checksum is 0 it means no error in data received, else there exists some error in the received data.

# Answer : Code:

def findChecksum(SentMessage, k): c1 = SentMessage[0:k]

c2 = SentMessage[k:2\*k]

Sum = bin(int(c1, 2)+int(c2, 2))[2:] if(len(Sum) > k):

Sum = bin(int(Sum[0:1], 2)+int(Sum[1:], 2))[2:] Checksum = ''

for i in Sum:

if(i == '1'):

Checksum += '0' else:

Checksum += '1' return Checksum

def checkReceiverChecksum(ReceivedMessage, k, Checksum):

c1 = ReceivedMessage[0:k] c2 = ReceivedMessage[k:2\*k]

ReceiverSum = bin(int(c1, 2) + int(c2, 2))[2:]

if(len(ReceiverSum) > k):

ReceiverSum = bin(int(ReceiverSum[0:1], 2)+int(ReceiverSum[1:], 2))[2:] ReceiverSum = bin(int(ReceiverSum, 2) + int(Checksum, 2))[2:]

ReceiverChecksum = '' for i in ReceiverSum: if(i == '1'):

ReceiverChecksum += '0' else:

ReceiverChecksum += '1' return ReceiverChecksum

# Driver Code SentMessage = "10010101"

k = 4

ReceivedMessage = "10000101" # any other value than this will give error

Checksum = findChecksum(SentMessage, k)

ReceiverChecksum = checkReceiverChecksum(ReceivedMessage, k, Checksum) print("SENDER SIDE CHECKSUM: ", Checksum)

print("RECEIVER SIDE CHECKSUM: ", ReceiverChecksum)

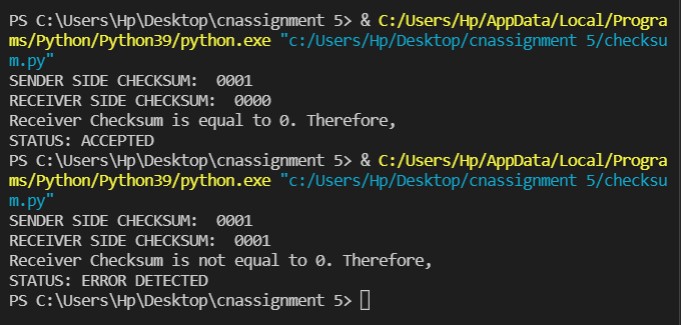
if(int(ReceiverChecksum, 2) == 0):

print("Receiver Checksum is equal to 0. Therefore,") print("STATUS: ACCEPTED")

else:

print("Receiver Checksum is not equal to 0. Therefore,") print("STATUS: ERROR DETECTED")

# Output :-



**Practical-9 :** Write a program to develop a Client that contacts a given DNS server to resolve a given host name.

# Answer : Code:

import requests

def connectToDNS(url):

recieved = requests.get("https://dns.google/resolve?name="+url) recievedJson = recieved.json()

return recievedJson['Answer'][0]['data']

url = input("Enter the url you want to resolve into IP address :- ") print("IP address of", url, "is -->", connectToDNS(url))

**Output-:**

