EXCERCISE I

Working with networking command line tools

AIM:

Working with networking command line tools in ubuntu terminal

Syntax and commands:

1. Ping

Tests network connectivity and latency to a host.

Example:

ping google.com

2. traceroute (tracert on Windows)

Displays the route packets take to reach a host.

Example:

traceroute google.com

3. ifconfig (Linux/macOS) / ip (Linux)

Shows or configures network interfaces.

Example:

ifconfig

ip addr

4. netstat

Displays network connections, routing tables, and more.

Example:

netstat -tuln

5. nslookup

Queries DNS for IP address or domain details.

Example:

nslookup google.com

6. dig

Performs DNS lookups

Example:

dig google.com

7. curl

Fetches content or data from a URL.

Example:

curl http://google.com

8. wget

Downloads files from the internet.

Example:

wget http://github.com/file.zip

9. scp (Secure Copy Protocol)

Securely transfers files.

Example:

scp file.txt it3122235002111@remote_host:/home/it3122235002111

10. ssh (Secure Shell)

Connects securely to a remote host.

Example:

ssh it3122235002111@remote_host

11. ftp

Transfers files using the FTP protocol.

Example:

ftp ftp.server.com

12. telnet

Connects to a remote system or tests ports.

Example:

telnet remote_host 80

13. route

Views or manipulates the system's routing table.

Example:

route -n

14. ipconfig (Windows)
    Displays network configuration on Windows.
Example:
    ipconfig /all
15. nmap
    Performs network scanning and port discovery.
Example:
    nmap -p 80 google.com
16. arp
    Displays or manipulates the ARP table.
Example:
    arp -a
17. mtr (My Traceroute)
    Combines ping and traceroute functionality.
Example:
    mtr google.com
18. ss
    Displays socket statistics (modern alternative to `netstat`).
Example:
    ss -tuln
19. tcpdump
    Captures and analyzes network packets.
Example:
    sudo tcpdump -i eth0
20. nc (Netcat)
    Versatile tool for networking, scanning, and testing.
Example:
    nc -l -p 1234
21. hostname
    Displays or sets the system's hostname.
Example:
    hostname
22. bmon (Bandwidth Monitor)
    Monitors network bandwidth usage in real time.
Example:

bmon

23. iwconfig (Linux)

Configures wireless network interfaces (Linux).

Example:

iwconfig

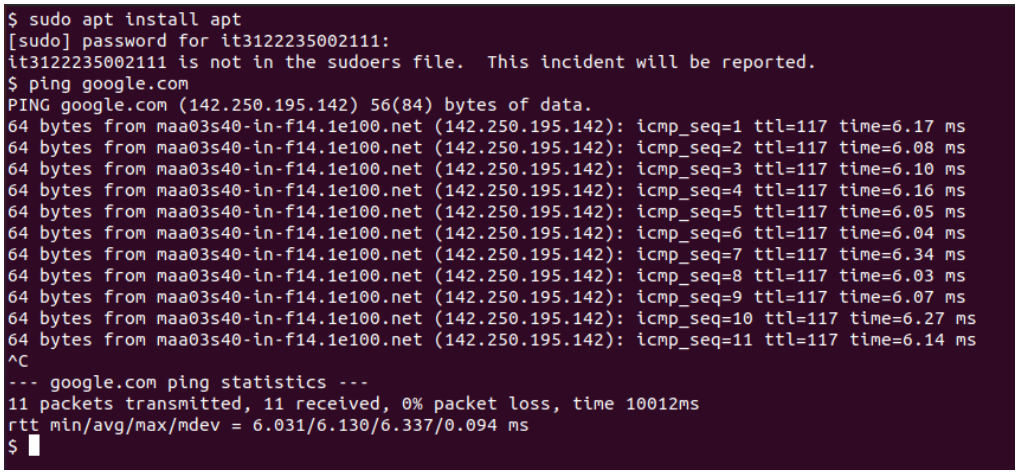
24. netcat

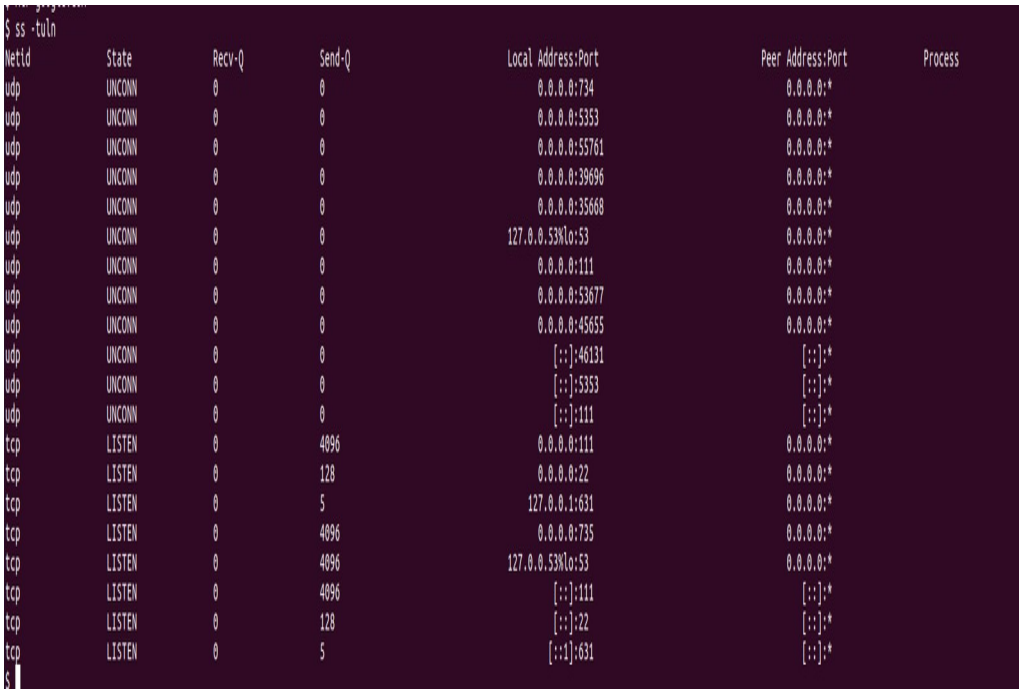
Same as nc (different naming on some systems).

Example:

netcat -l -p 1234

Screenshots:






```
$ nslookup google.com
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.195.206
Name:   google.com
Address: 2404:6800:4007:827::200e

$ dig google.com

; <<>> DiG 9.18.28-0ubuntu0.20.04.1-Ubuntu <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44626
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             236     IN      A       142.250.195.206

;; Query time: 0 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Mon Jan 27 14:02:06 IST 2025
;; MSG SIZE  rcvd: 55
```

```
$ ss -tuln
Netid           State            Recv-Q           Send-Q                     Local Address:Port                     Peer Address:Port           Process
udp             UNCONN           0                0                          0.0.0.0:734                           0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:5353                          0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:55761                         0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:39696                         0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:35668                         0.0.0.0:*
udp             UNCONN           0                0                    127.0.0.53%lo:53                            0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:111                           0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:53677                         0.0.0.0:*
udp             UNCONN           0                0                          0.0.0.0:45655                         0.0.0.0:*
udp             UNCONN           0                0                             [::]:46131                            [::]:*
udp             UNCONN           0                0                             [::]:5353                             [::]:*
udp             UNCONN           0                0                             [::]:111                              [::]:*
tcp             LISTEN           0                4096                       0.0.0.0:111                           0.0.0.0:*
tcp             LISTEN           0                128                        0.0.0.0:22                            0.0.0.0:*
tcp             LISTEN           0                5                        127.0.0.1:631                           0.0.0.0:*
tcp             LISTEN           0                4096                       0.0.0.0:735                           0.0.0.0:*
tcp             LISTEN           0                4096                 127.0.0.53%lo:53                            0.0.0.0:*
tcp             LISTEN           0                4096                          [::]:111                              [::]:*
tcp             LISTEN           0                128                           [::]:22                               [::]:*
tcp             LISTEN           0                5                            [::1]:631                              [::]:*
$
```

```
$ hostname
vnlab
$ bmon
sh: 20: bmon: not found
$ iwconfig
lo        no wireless extensions.

enp3s0    no wireless extensions.
```

```
$ netcat -1 -p 1234
netcat: invalid option -- '1'
usage: nc [-46CDdFhklNnrStUuvZz] [-I length] [-i interval] [-M ttl]
          [-m minttl] [-O length] [-P proxy_username] [-p source_port]
          [-q seconds] [-s source] [-T keyword] [-V rtable] [-W recvlimit] [-w timeout]
          [-X proxy_protocol] [-x proxy_address[:port]]           [destination] [port]
$ nc -1 -p 1234
nc: invalid option -- '1'
usage: nc [-46CDdFhklNnrStUuvZz] [-I length] [-i interval] [-M ttl]
          [-m minttl] [-O length] [-P proxy_username] [-p source_port]
          [-q seconds] [-s source] [-T keyword] [-V rtable] [-W recvlimit] [-w timeout]
          [-X proxy_protocol] [-x proxy_address[:port]]           [destination] [port]
$
```

```
$ traceroute google.com
traceroute to google.com (142.250.195.142), 30 hops max, 60 byte packets
 1  _gateway (10.7.0.1)  13.490 ms  16.882 ms  16.856 ms
 2  10.107.0.2 (10.107.0.2)  0.302 ms  0.274 ms  0.288 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
$
```

```
$ sudo ip addr
[sudo] password for it3122235002111:
it3122235002111 is not in the sudoers file.  This incident will be reported.
$ netstat -tuln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:735             0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp6       0      0 :::111                  :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
udp        0      0 0.0.0.0:734             0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:55761           0.0.0.0:*
udp        0      0 0.0.0.0:39696           0.0.0.0:*
udp        0      0 0.0.0.0:35668           0.0.0.0:*
udp        0      0 127.0.0.53:53           0.0.0.0:*
udp        0      0 0.0.0.0:111             0.0.0.0:*
udp        0      0 0.0.0.0:53677           0.0.0.0:*
udp6       0      0 :::46131                :::*
udp6       0      0 :::5353                 :::*
udp6       0      0 :::111                  :::*
$
```

```
$ arp -a
_gateway (10.7.0.1) at 18:8b:45:8a:eb:e7 [ether] on enp3s0
? (10.7.9.123) at 5c:60:ba:3e:ae:03 [ether] on enp3s0
? (10.7.7.9) at 7c:57:58:c3:77:8c [ether] on enp3s0
? (10.7.5.2) at dc:4a:3e:80:66:59 [ether] on enp3s0
```

Result:

The above mentioned basic linux networking commands commands are run in the ubuntu terminal and exercised successfully.

UIT2411 – Network Programming Lab

Assignment 2:

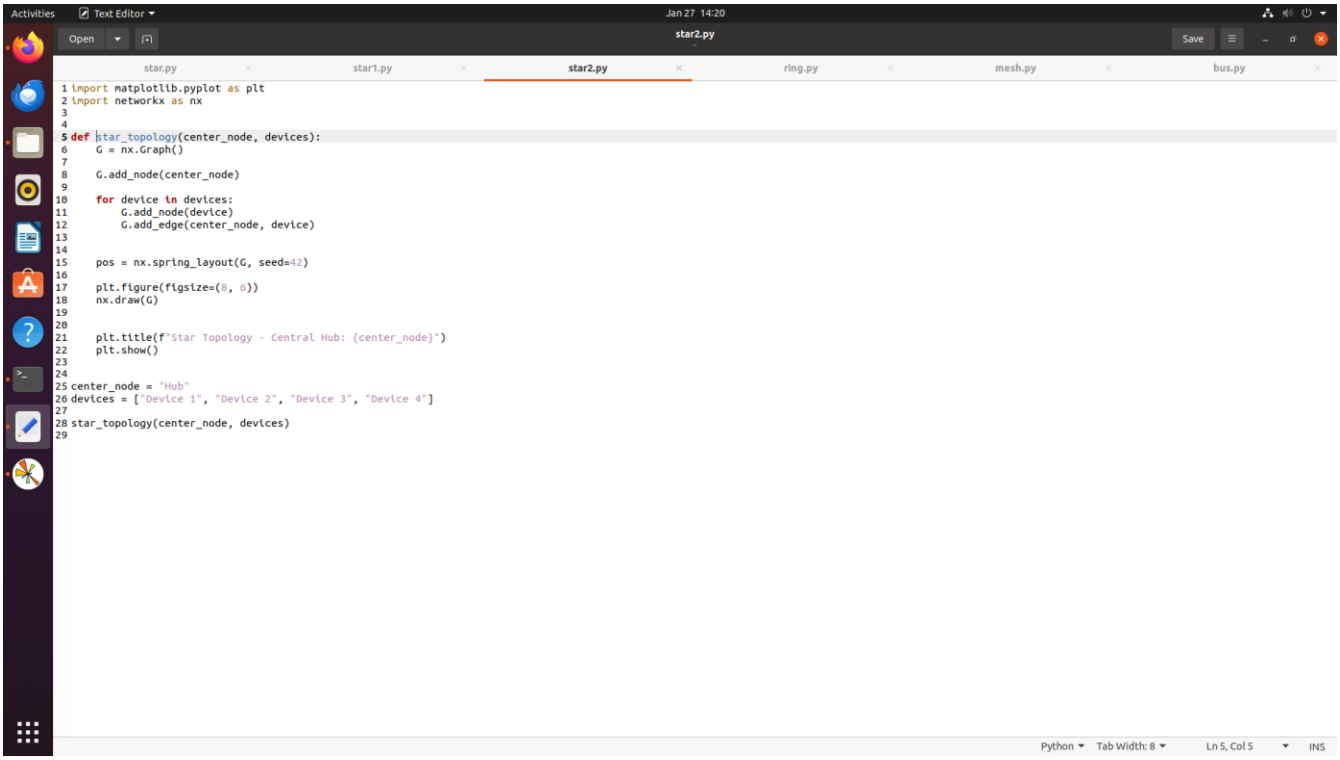
Simulation of Network Topologies using Twisted python

AIM:

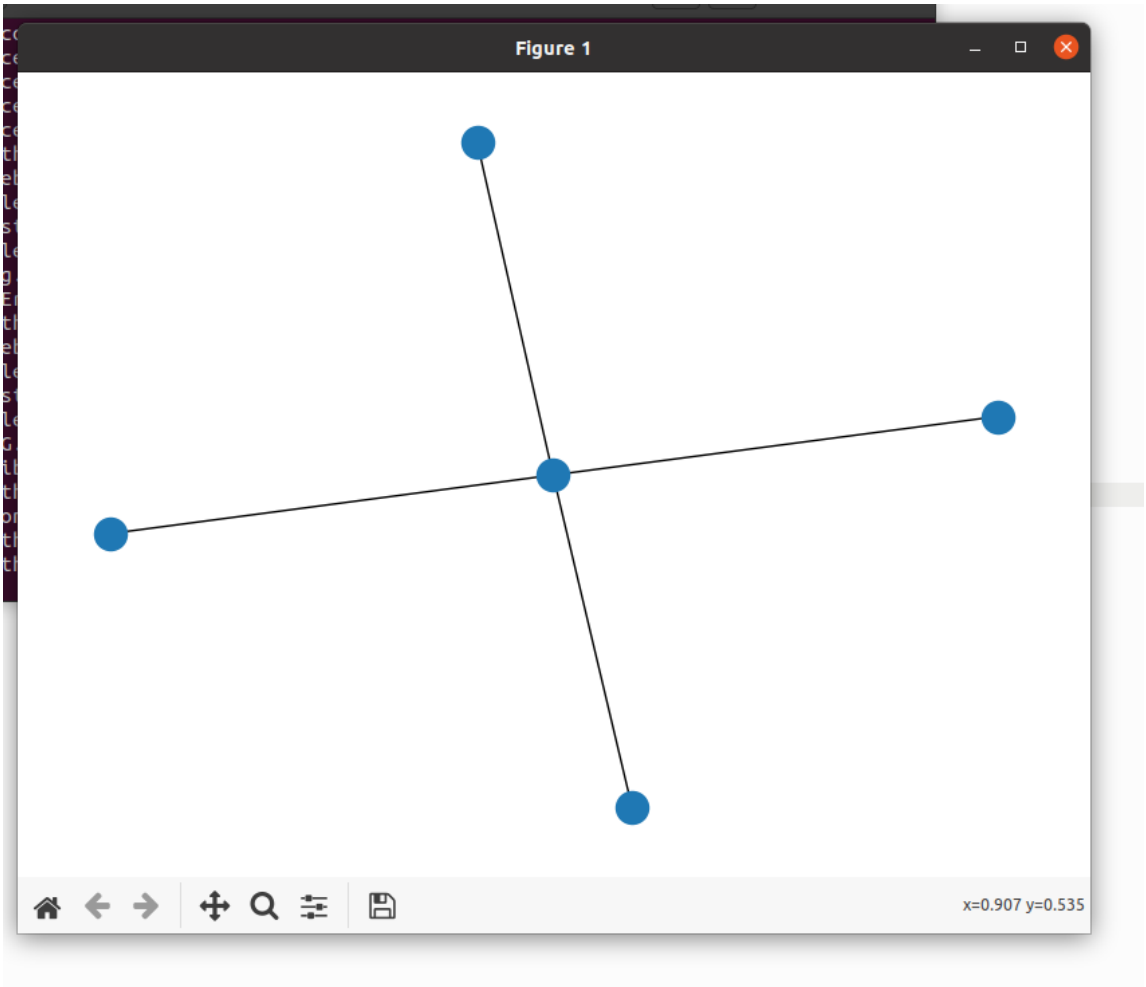
1. Simulate different types of network topologies (e.g., star, ring, mesh, bus) using Python.
2. Visualize the topologies and their properties (such as connectivity, degree, etc.).
3. Understand how network topologies impact the overall structure of a network.

STAR TOPOLOGY:

PYTHON CODE:

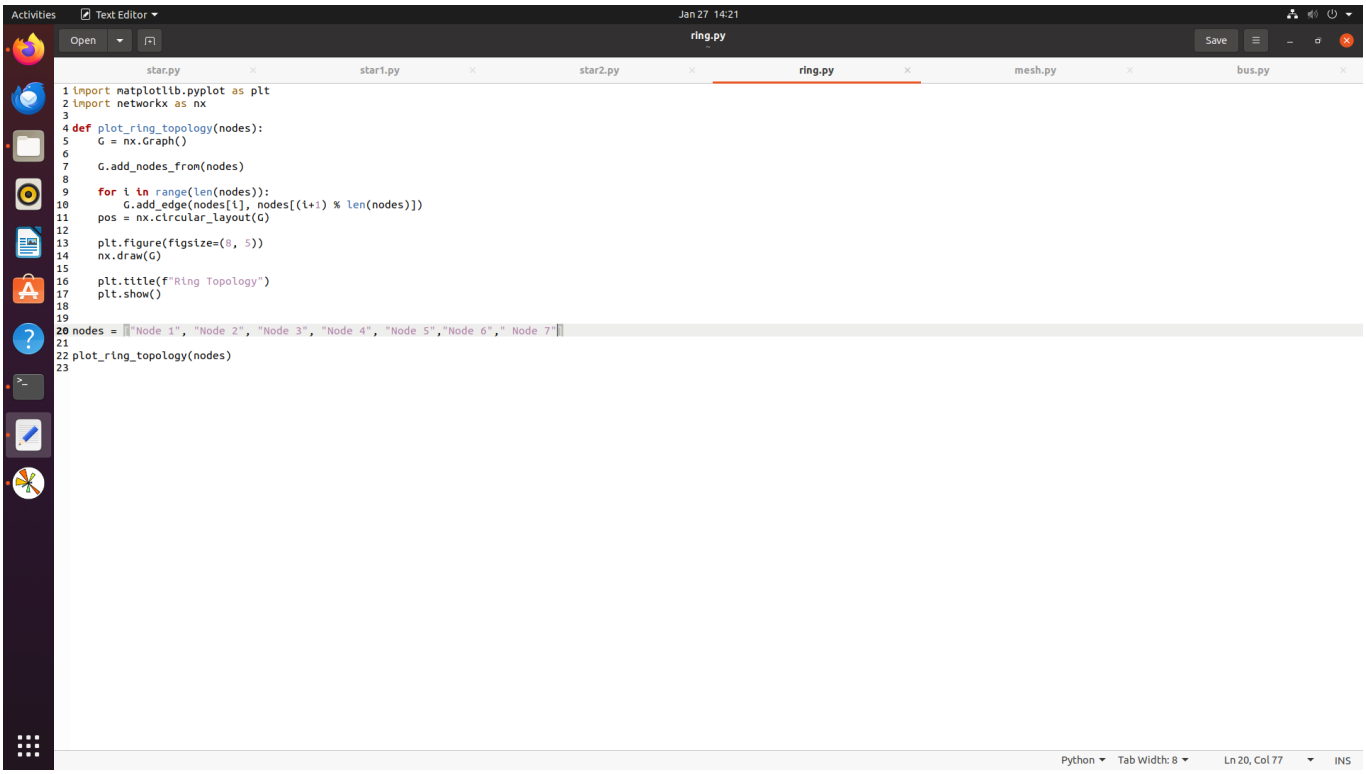


OUTPUT (VISUALIZATION):

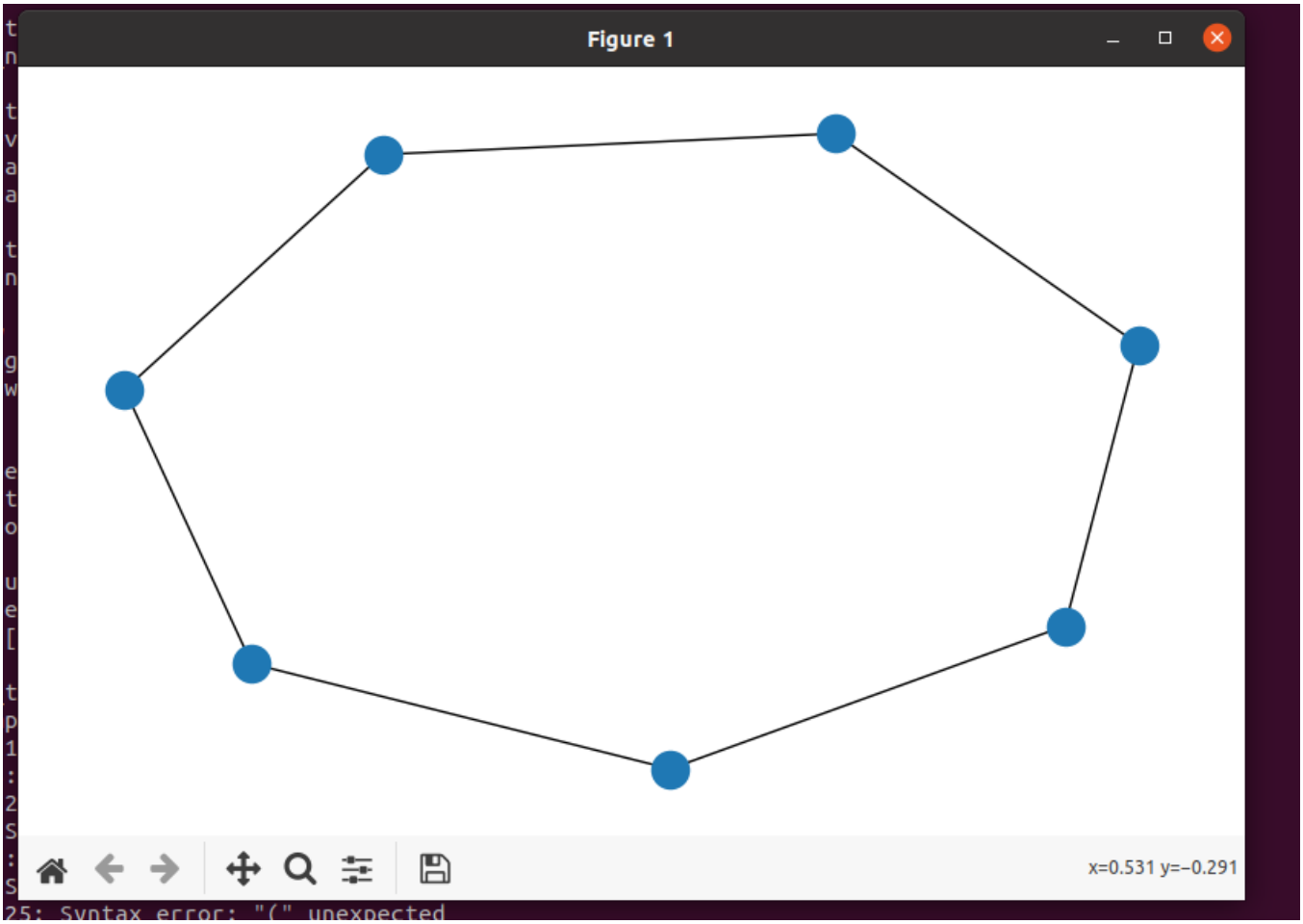


RING TOPOLOGY:

PYTHON CODE:

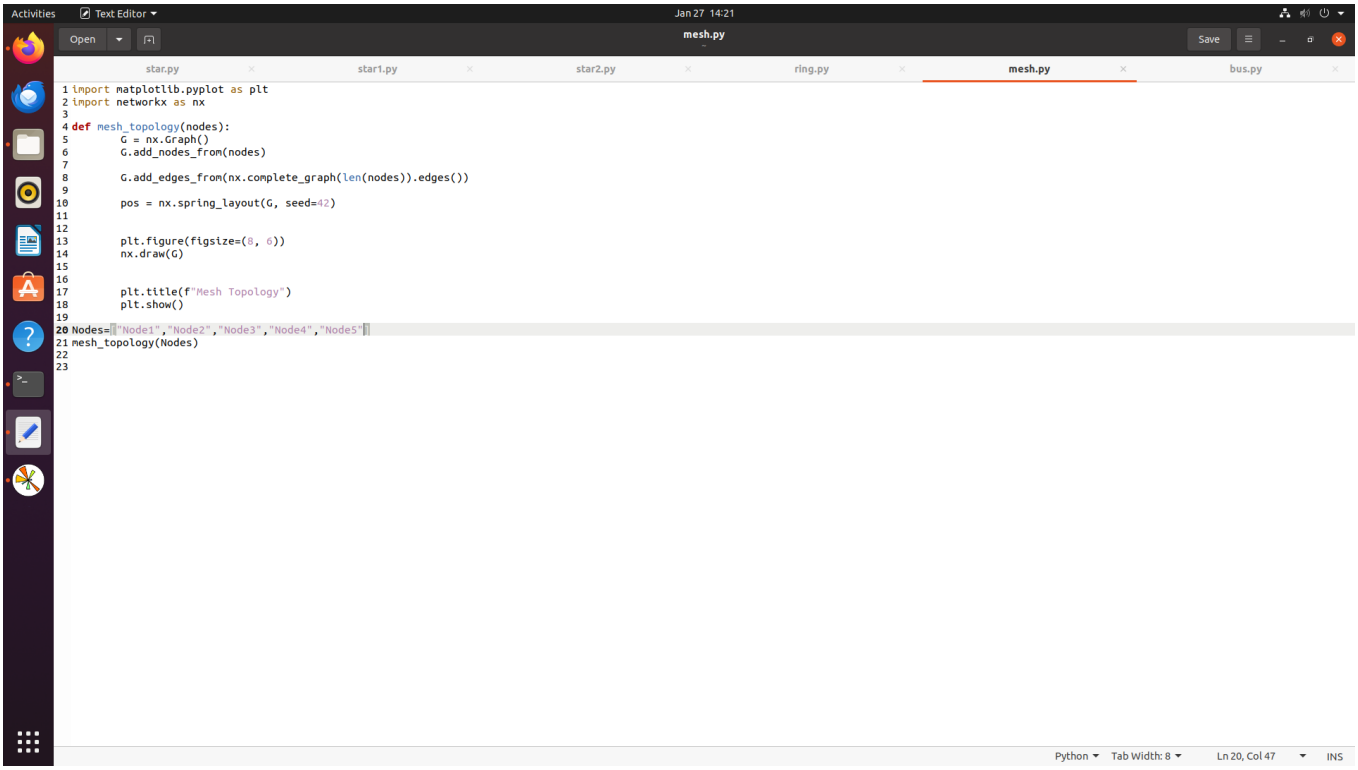


OUTPUT (VISUALIZATION):

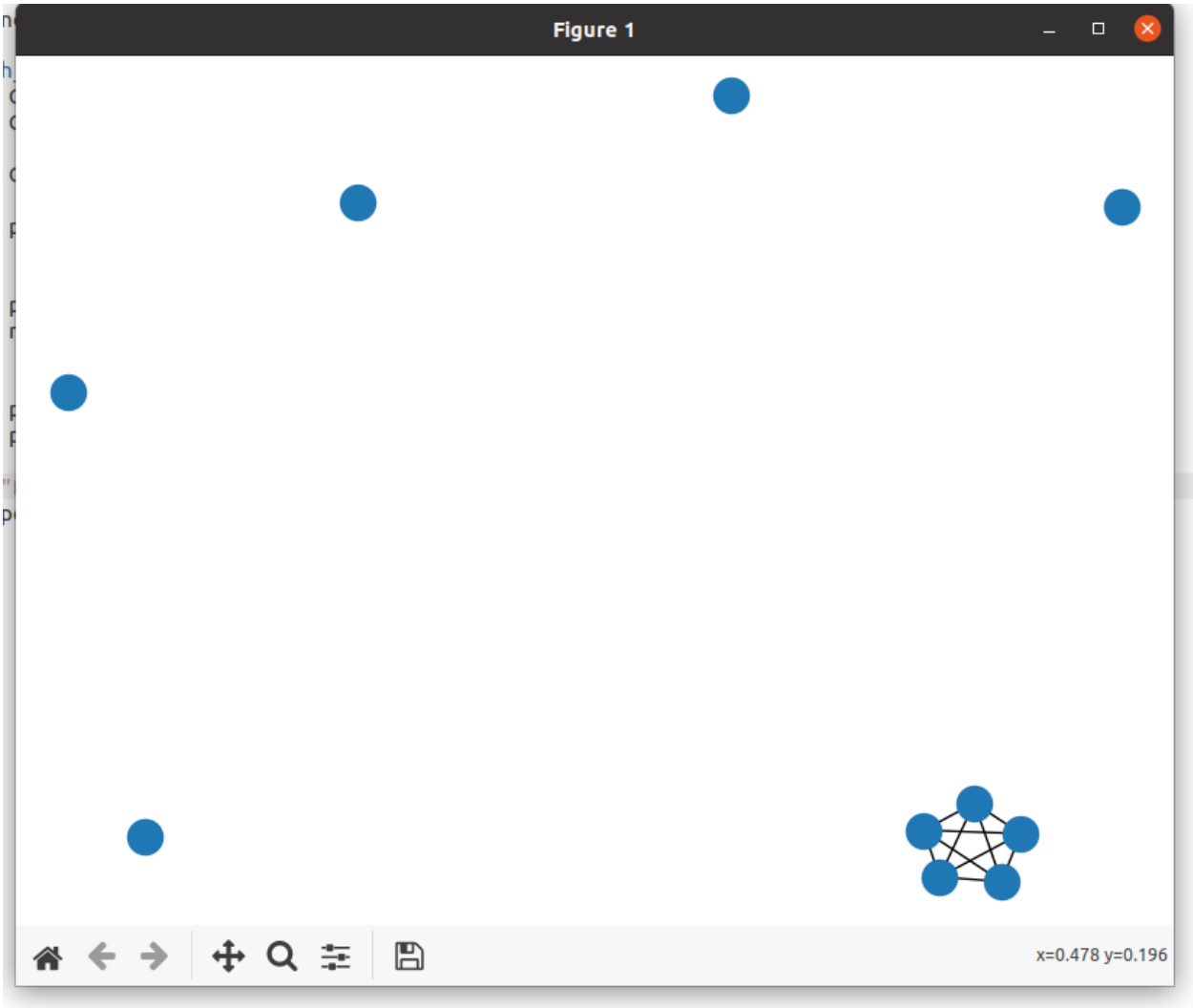


MESH TOPOLOGY:

PYTHON CODE:



OUTPUT (VISUALIZATION):



STAR NETWORK:

SERVER-SIDE CODE:

```
from twisted.internet import protocol, reactor from
twisted.protocols.basic import LineReceiver

class StarServerProtocol(LineReceiver): def connectionMade(self):
print(f"Client connected: {self.transport.getPeer()}")
self.sendLine("Welcome to the Star Server!")

def lineReceived(self, line):
   print(f"Received from client: {line}")
   self.sendLine(f"Echo: {line}")

def connectionLost(self, reason):
   print(f"Client disconnected: {self.transport.getPeer()}")


class StarServerFactory(protocol.Factory): def buildProtocol(self,
addr): return StarServerProtocol()

def start_server(): port = 12345 # Port to listen on print(f"Server
starting on port {port}") reactor.listenTCP(port, StarServerFactory())
reactor.run()

if name == "main": start_server()
```

OUTPUT:

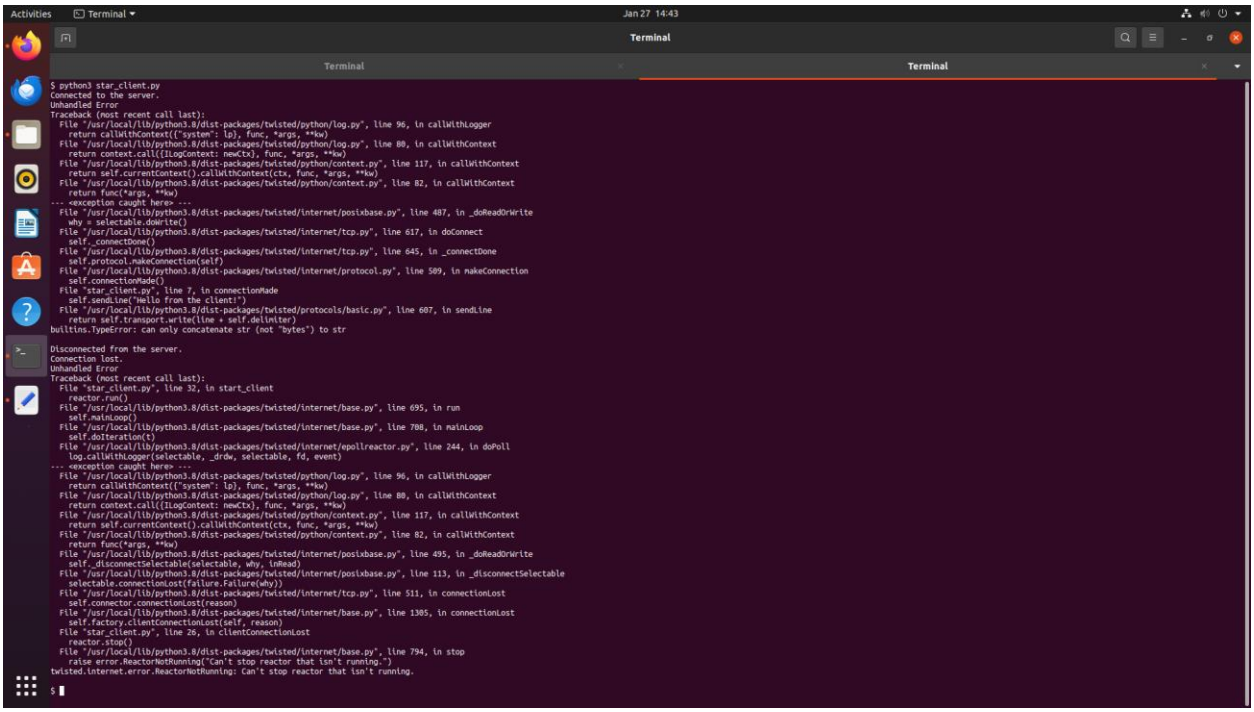
```
$ python3 STAR_SERVER.py
Server starting on port 12345
python3 star_client.py
Client connected: IPv4Address(type='TCP', host='127.0.0.1', port=52218)
Unhandled Error
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/twisted/python/log.py", line 80, in callWithContext
    return context.call({ILogContext: newCtx}, func, *args, **kw)
  File "/usr/local/lib/python3.8/dist-packages/twisted/python/context.py", line 117, in callWithContext
    return self.currentContext().callWithContext(ctx, func, *args, **kw)
  File "/usr/local/lib/python3.8/dist-packages/twisted/python/context.py", line 82, in callWithContext
    return func(*args, **kw)
  File "/usr/local/lib/python3.8/dist-packages/twisted/internet/posixbase.py", line 482, in _doReadOrWrite
    why = selectable.doRead()
--- <exception caught here> ---
  File "/usr/local/lib/python3.8/dist-packages/twisted/internet/tcp.py", line 1422, in doRead
    protocol.makeConnection(transport)
  File "/usr/local/lib/python3.8/dist-packages/twisted/internet/protocol.py", line 509, in makeConnection
    self.connectionMade()
  File "STAR_SERVER.py", line 7, in connectionMade
    self.sendLine("Welcome to the Star Server!")
  File "/usr/local/lib/python3.8/dist-packages/twisted/protocols/basic.py", line 607, in sendLine
    return self.transport.write(line + self.delimiter)
builtins.TypeError: can only concatenate str (not "bytes") to str

Client disconnected: IPv4Address(type='TCP', host='127.0.0.1', port=52218)
```

CLIENT-SIDE CODE:

PYTHON CODE:

```
from twisted.internet import defer, protocol, reactor from twisted.protocols.basic import LineReceiver

class StarClientProtocol(LineReceiver): def connectionMade(self): print("Connected to the server.") self.sendLine("Hello from the client!")

def lineReceived(self, line):
    print(f"Received from server: {line}")

def connectionLost(self, reason):
    print("Disconnected from the server.")
    reactor.stop()


class StarClientFactory(protocol.ClientFactory): def buildProtocol(self, addr): return StarClientProtocol()

def clientConnectionFailed(self, connector, reason):
    print(f"Connection failed: {reason}")
    reactor.stop()

def clientConnectionLost(self, connector, reason):
    print("Connection lost.")
    reactor.stop()


def start_client(): server_address = ('localhost', 12345) factory = StarClientFactory() reactor.connectTCP(*server_address, factory)

if __name__ == "__main__": start_client()
```

OUTPUT:



RING NETWORK:

SERVER-SIDE CODE:

from twisted.internet import protocol, reactor from twisted.protocols.basic import LineReceiver

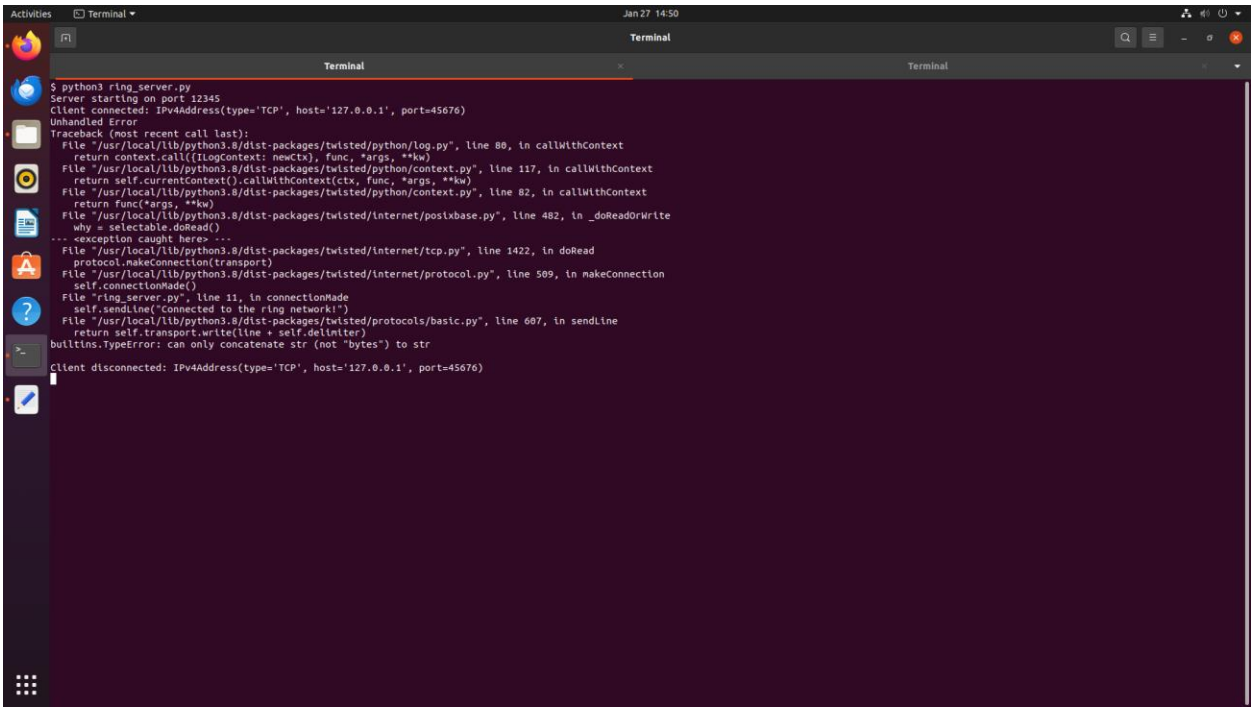
```
class RingServerProtocol(LineReceiver): def init(self):
self.next_node = None self.prev_node = None

def connectionMade(self):
    print(f"Client connected: {self.transport.getPeer()}")
    self.sendLine("Connected to the ring network!")

def lineReceived(self, line):
    print(f"Received message: {line}")
    if line.startswith("Target:"):
        self.sendLine(f"Message received by {self.transport.getPeer()}")
    else:
        if self.next_node:
            print(f"Forwarding message to next node: {self.next_node}")
            self.next_node.sendLine(line)

def connectionLost(self, reason):
    print(f"Client disconnected: {self.transport.getPeer()}")
    if self.prev_node:
        self.prev_node.next_node = self.next_node
    if self.next_node:
        self.next_node.prev_node = self.prev_node


class RingServerFactory(protocol.Factory): def init(self): self.nodes
= []

def buildProtocol(self, addr):
    protocol = RingServerProtocol()
    self.nodes.append(protocol)
```

```
    if len(self.nodes) > 1:
        prev_node = self.nodes[-2]
        protocol.prev_node = prev_node
        prev_node.next_node = protocol
    return protocol
```

def start_server(): port = 12345 print(f"Server starting on port {port}") reactor.listenTCP(port, RingServerFactory()) # Start the server reactor.run()

if **name** == "**main**": start_server()

OUTPUT:



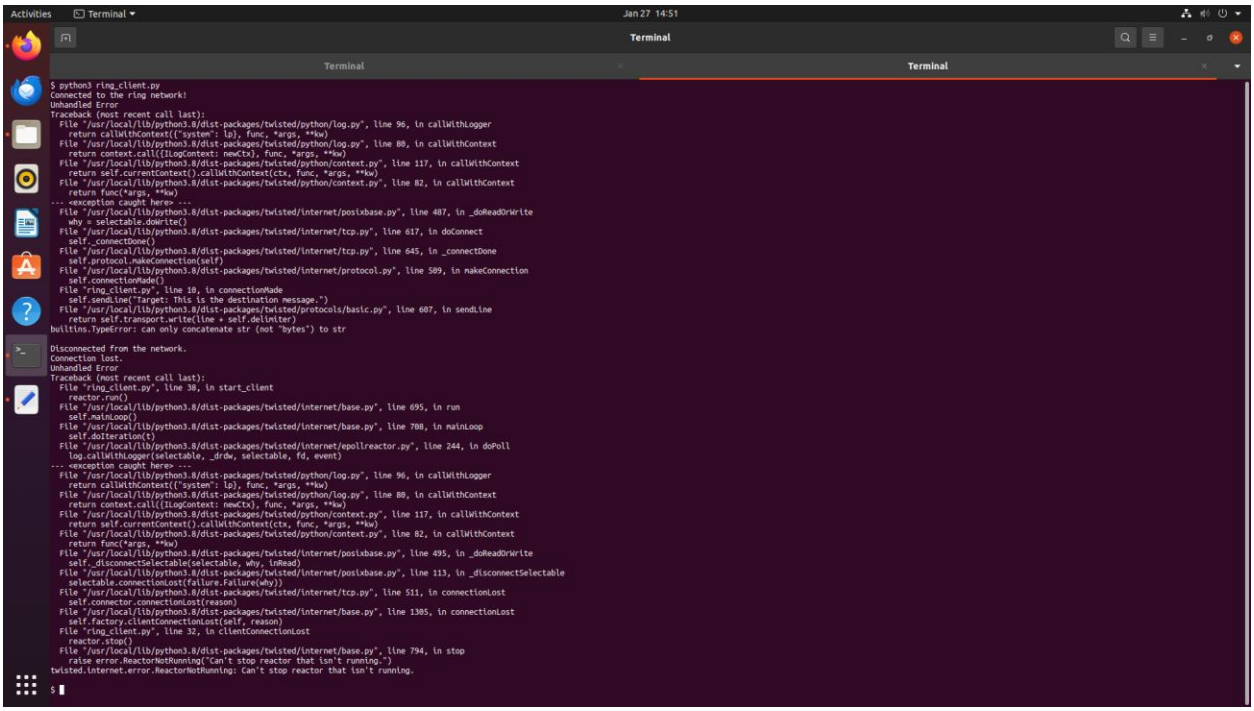
CLIENT-SIDE CODE:

PYTHON CODE:

```
from twisted.internet import protocol, reactor from
twisted.protocols.basic import LineReceiver

class RingClientProtocol(LineReceiver):

def init(self):

self.target_node = None

def connectionMade(self):
   print("Connected to the ring network!")
   self.sendLine("Target: This is the destination message.")

def lineReceived(self, line):
   print(f"Received from ring: {line}")
   if line.startswith("Message received"):
      print("Message has reached its destination!")
      reactor.stop()

def connectionLost(self, reason):
   print("Disconnected from the network.")
   reactor.stop()


class RingClientFactory(protocol.ClientFactory): def
buildProtocol(self, addr): return RingClientProtocol()

def clientConnectionFailed(self, connector, reason):
   print(f"Connection failed: {reason}")
   reactor.stop()

def clientConnectionLost(self, connector, reason):
```

```
    print("Connection lost.")
    reactor.stop()


def start_client(): server_address = ('localhost', 12345)

 factory = RingClientFactory() reactor.connectTCP(*server_address,
factory) reactor.run()

if __name__ == "__main__": start_client()
```

OUTPUT:



MESH NETWORK:

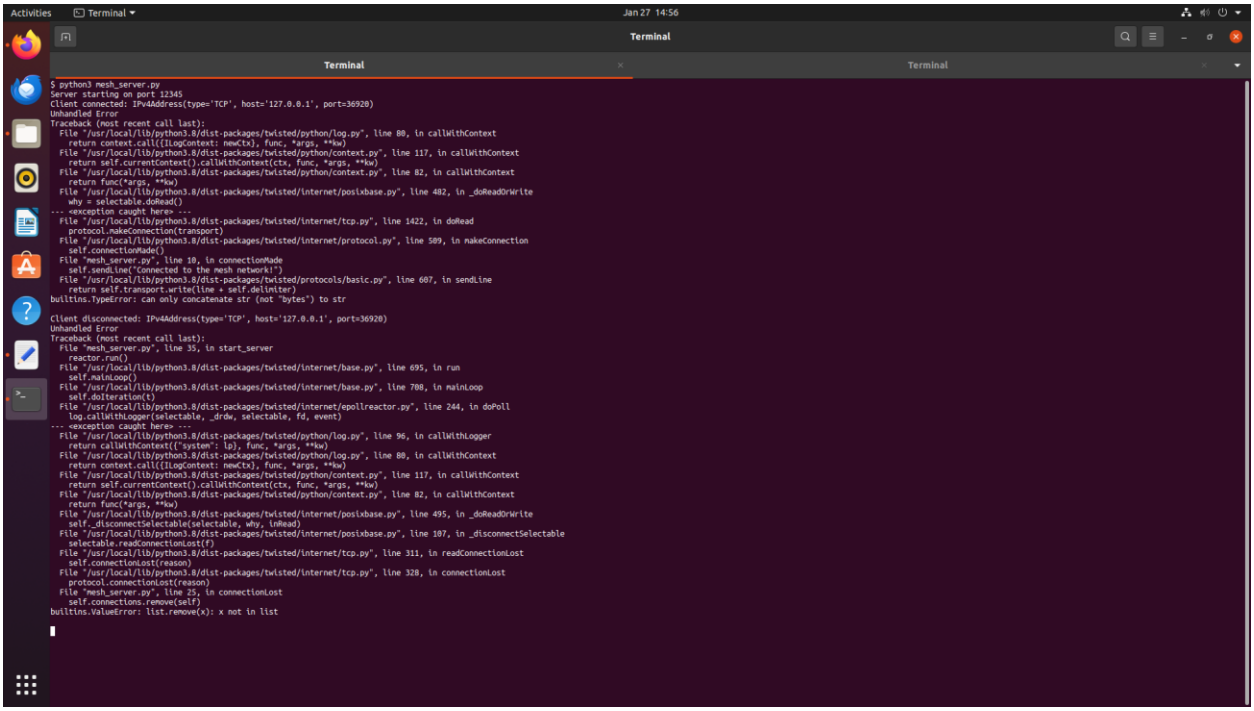
SERVER-SIDE CODE:

from twisted.internet import protocol, reactor from twisted.protocols.basic import LineReceiver

```
class MeshServerProtocol(LineReceiver): def __init__(self):
self.connections = []

def connectionMade(self):
    print(f"Client connected: {self.transport.getPeer()}")
    self.sendLine("Connected to the mesh network!")
    self.connections.append(self)

    for conn in self.connections:
        if conn != self:
            conn.sendLine(f"New client {self.transport.getPeer()} has
joined.")

def lineReceived(self, line):
    print(f"Received message: {line}")
    for conn in self.connections:
        if conn != self:
            conn.sendLine(line)

def connectionLost(self, reason):
    print(f"Client disconnected: {self.transport.getPeer()}")
    self.connections.remove(self)


class MeshServerFactory(protocol.Factory): def buildProtocol(self,
addr): return MeshServerProtocol()

def start_server(): port = 12345 print(f"Server starting on port {port}")
reactor.listenTCP(port, MeshServerFactory())

 reactor.run()
```

if **name** == "**main**": start_server()

OUTPUT:



CLIENT-SIDE CODE:

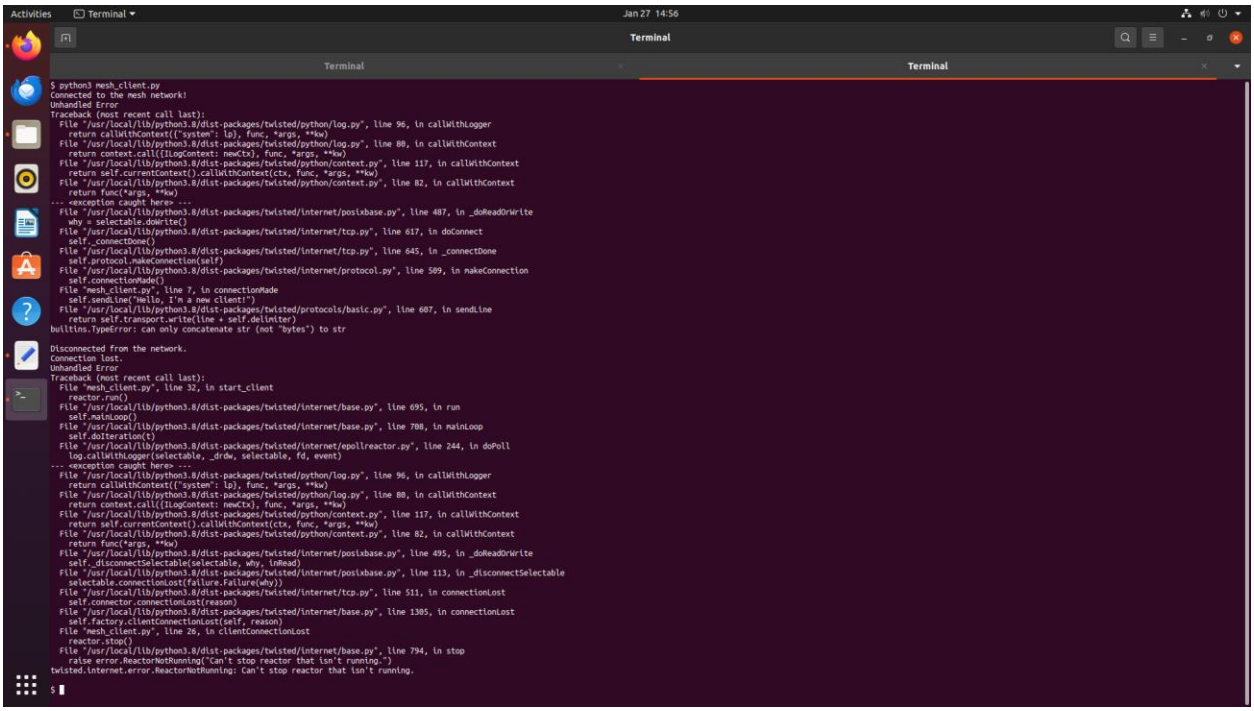
PYTHON CODE:

from twisted.internet import protocol, reactor from twisted.protocols.basic import LineReceiver

class MeshClientProtocol(LineReceiver): def connectionMade(self): print("Connected to the mesh network!") self.sendLine("Hello, I'm a new client!")

```
def lineReceived(self, line):
    print(f"Received from mesh network: {line}")

def connectionLost(self, reason):
    print("Disconnected from the network.")
    reactor.stop()


class MeshClientFactory(protocol.ClientFactory): def
buildProtocol(self, addr): return MeshClientProtocol()

def clientConnectionFailed(self, connector, reason):
    print(f"Connection failed: {reason}")
    reactor.stop()

def clientConnectionLost(self, connector, reason):
    print("Connection lost.")
    reactor.stop()


def start_client(): server_address = ('localhost', 12345)

 factory = MeshClientFactory()
reactor.connectTCP(*server_address, factory)

 reactor.run()

if name == "main": start_client()
```

OUTPUT:



BUS NETWORK:

SERVER-SIDE CODE:

PYTHON CODE:

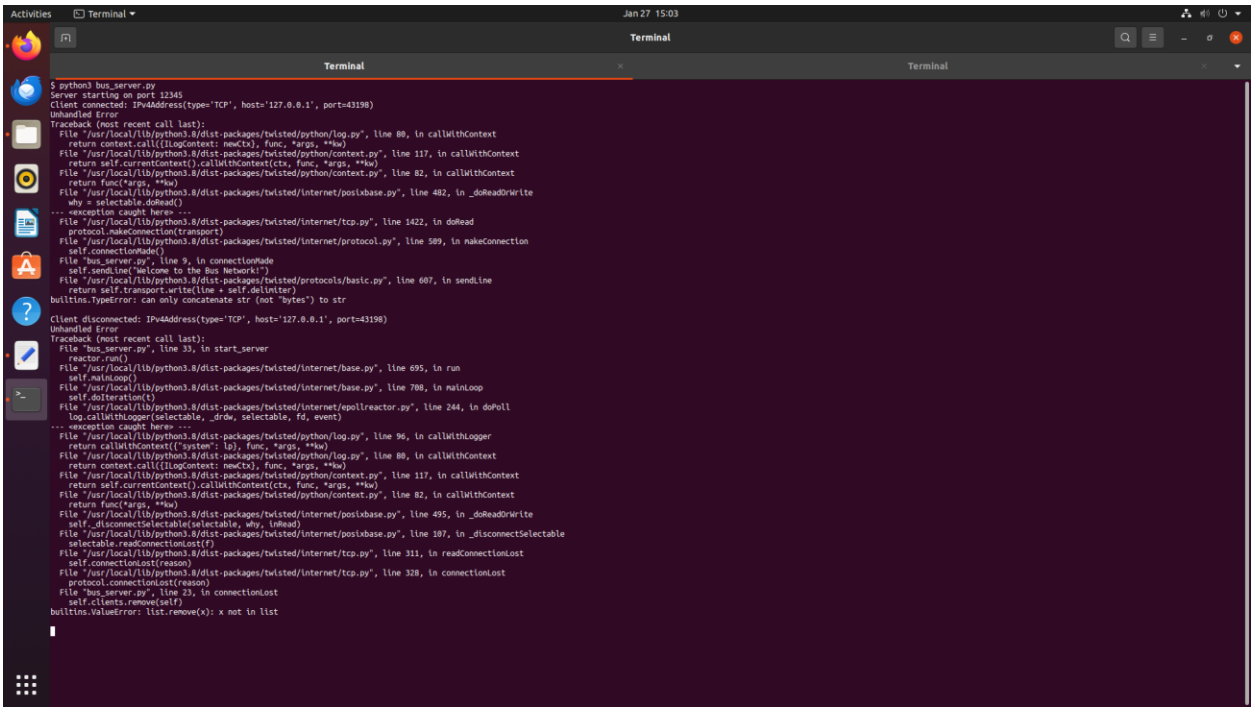
from twisted.internet import protocol, reactor from twisted.protocols.basic import LineReceiver

class BusServerProtocol(LineReceiver): def **init**(self): self.clients = []

def connectionMade(self): print(f"Client connected: {self.transport.getPeer()}") self.sendLine("Welcome to the Bus Network!") self.clients.append(self) for client in self.clients: if

```
client != self: client.sendLine(f"New client {self.transport.getPeer()}
has joined the bus.")

def lineReceived(self, line):
   print(f"Received message: {line}")
   for client in self.clients:
      if client != self:
         client.sendLine(line)

def connectionLost(self, reason):
   print(f"Client disconnected: {self.transport.getPeer()}")
   self.clients.remove(self)


class BusServerFactory(protocol.Factory): def buildProtocol(self,
addr): return BusServerProtocol()

def start_server(): port = 12345 print(f"Server starting on port {port}")
reactor.listenTCP(port, BusServerFactory())

 reactor.run()

if name == "main": start_server()
```

OUTPUT:



CLIENT-SIDE CODE:

from twisted.internet import protocol, reactor from twisted.protocols.basic import LineReceiver

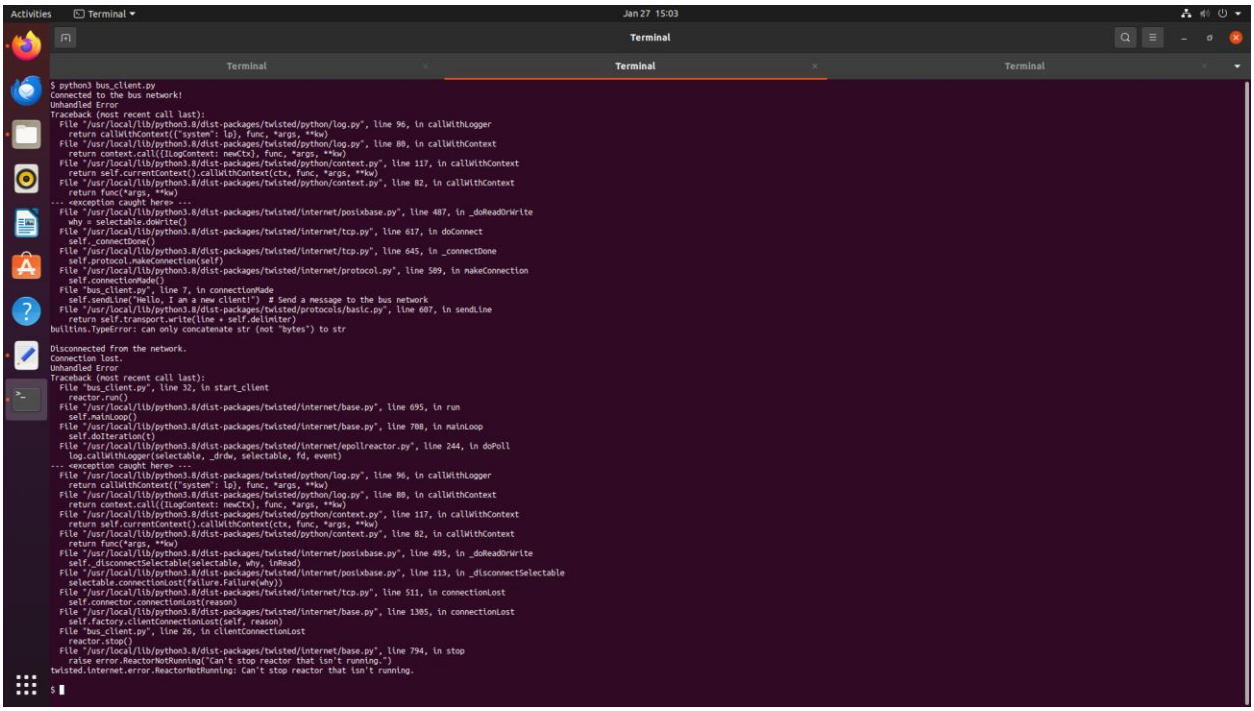
class BusClientProtocol(LineReceiver): def connectionMade(self): print("Connected to the bus network!") self.sendLine("Hello, I am a new client!") # Send a message to the bus network

def lineReceived(self, line):
    print(f"Received from bus network: {line}")

def connectionLost(self, reason):

```
    print("Disconnected from the network.")
    reactor.stop()  # Stop reactor when the client disconnects


class BusClientFactory(protocol.ClientFactory): def
buildProtocol(self, addr): return BusClientProtocol()

def clientConnectionFailed(self, connector, reason):
    print(f"Connection failed: {reason}")
    reactor.stop()

def clientConnectionLost(self, connector, reason):
    print("Connection lost.")
    reactor.stop()


def start_client(): server_address = ('localhost', 12345) # Server
address and port factory = BusClientFactory()
reactor.connectTCP(*server_address, factory) # Connect to the
server reactor.run()

if name == "main": start_client()
```

OUTPUT:


Activities
Terminal
Jan 27 15:03
Terminal
Terminal
Terminal
Terminal
$ python3 bus_client.py
Connected to the bus network!
Unhandled Error
Traceback (most recent call last):
File "/usr/local/lib/python3.8/dist-packages/twisted/python/log.py", line 96, in callWithLogger
return callWithContext({"system": lg}, func, *args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/python/log.py", line 80, in callWithContext
return context.call({ILogContext: newCtx}, func, *args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/python/context.py", line 117, in callWithContext
return self.currentContext().callWithContext(ctx, func, *args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/python/context.py", line 82, in callWithContext
return func(*args, **kw)
--- <exception caught here> ---
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/posixbase.py", line 487, in _doReadOrWrite
why = selectable.doWrite()
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/tcp.py", line 617, in doConnect
self._connectDone()
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/tcp.py", line 645, in _connectDone
self.protocol.makeConnection(self)
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/protocol.py", line 589, in makeConnection
self.connectionMade()
File "bus_client.py", line 7, in connectionMade
self.sendLine("Hello, I am a new client!") # Send a message to the bus network
File "/usr/local/lib/python3.8/dist-packages/twisted/protocols/basic.py", line 607, in sendLine
return self.transport.write(line + self.delimiter)
builtins.TypeError: can only concatenate str (not "bytes") to str
Disconnected from the network.
Connection lost.
Unhandled Error
Traceback (most recent call last):
File "bus_client.py", line 32, in start_client
reactor.run()
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/base.py", line 695, in run
self.mainLoop()
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/base.py", line 708, in mainLoop
self.doIteration(t)
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/epollreactor.py", line 244, in doPoll
log.callWithLogger(selectable, _drdw, selectable, fd, event)
--- <exception caught here> ---
File "/usr/local/lib/python3.8/dist-packages/twisted/python/log.py", line 96, in callWithLogger
return callWithContext({"system": lg}, func, *args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/python/log.py", line 80, in callWithContext
return context.call({ILogContext: newCtx}, func, *args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/python/context.py", line 117, in callWithContext
return self.currentContext().callWithContext(ctx, func, *args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/python/context.py", line 82, in callWithContext
return func(*args, **kw)
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/posixbase.py", line 495, in _doReadOrWrite
self._disconnectSelectable(selectable, why, inRead)
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/posixbase.py", line 113, in _disconnectSelectable
selectable.connectionLost(failure.Failure(why))
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/tcp.py", line 511, in connectionLost
self.connector.connectionLost(reason)
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/base.py", line 1305, in connectionLost
self.factory.clientConnectionLost(self, reason)
File "bus_client.py", line 26, in clientConnectionLost
reactor.stop()
File "/usr/local/lib/python3.8/dist-packages/twisted/internet/base.py", line 794, in stop
raise error.ReactorNotRunning("Can't stop reactor that isn't running.")
twisted.internet.error.ReactorNotRunning: Can't stop reactor that isn't running.
$

Assignment 3: Implementation of PING and TRACEROUTE using Twisted Python

Aim:

To simulate the functionality of the PING and TRACEROUTE commands using the Twisted Python networking framework.

Code:

```
from twisted.internet import reactor, defer
from twisted.names import client
from twisted.internet.protocol import DatagramProtocol
import struct, time


class PingClient(DatagramProtocol):
    def __init__(self, target):
        self.target = target
        self.start_time = None


    def startProtocol(self):
        self.transport.connect(self.target, 0)
        self.start_time = time.time()
        self.transport.write(b'PING')
        print(f'Pinging {self.target}...')
```

```
    def datagramReceived(self, data, addr):
        rtt = (time.time() - self.start_time) * 1000
        print(f'Reply from {addr}: time={rtt:.2f}ms')
        reactor.stop()

class TracerouteClient(DatagramProtocol):
    def __init__(self, target, max_hops=30):
        self.target = target
        self.ttl = 1
        self.max_hops = max_hops

    def startProtocol(self):
        self.trace()

    def trace(self):
        self.transport.socket.setsockopt(0, 2, self.ttl)  # Set TTL
        self.transport.write(b'TRACE', (self.target, 33434))
        print(f'{self.ttl}: Sending packet to {self.target}')
        reactor.callLater(1, self.check_timeout)

    def datagramReceived(self, data, addr):
        print(f'{self.ttl}: Reply from {addr[0]}')
        if addr[0] == self.target or self.ttl >= self.max_hops:
            reactor.stop()
        else:
```

```
            self.ttl += 1
            self.trace()

    def check_timeout(self):
        if self.ttl < self.max_hops:
            print(f'{self.ttl}: *')
            self.ttl += 1
            self.trace()
        else:
            reactor.stop()

def resolve_target(hostname):
    d = client.lookupAddress(hostname)
    d.addCallback(lambda result: result[0][0].payload.dottedQuad())
    return d

def run_ping(hostname):
    d = resolve_target(hostname)
    d.addCallback(lambda ip: reactor.listenUDP(0, PingClient(ip)))

def run_traceroute(hostname):
    d = resolve_target(hostname)
    d.addCallback(lambda ip: reactor.listenUDP(0, TracerouteClient(ip)))

if __name__ == '__main__':
```

```
import sys
if len(sys.argv) < 3 or sys.argv[1] not in ['ping', 'traceroute']:
    print('Usage: python script.py [ping|traceroute] <hostname>')
    sys.exit(1)

if sys.argv[1] == 'ping':
    run_ping(sys.argv[2])
else:
    run_traceroute(sys.argv[2])

reactor.run()
```

```
from twisted.internet import reactor, defer
from twisted.names import client
from twisted.internet.protocol import DatagramProtocol
import struct, time

class PingClient(DatagramProtocol):
    def __init__(self, target):
        self.target = target
        self.start_time = None

    def startProtocol(self):
        self.transport.connect(self.target, 0)
        self.start_time = time.time()
        self.transport.write(b'PING')
        print(f'Pinging {self.target}...')

    def datagramReceived(self, data, addr):
        rtt = (time.time() - self.start_time) * 1000
        print(f'Reply from {addr}: time={rtt:.2f}ms')
        reactor.stop()

class TracerouteClient(DatagramProtocol):
    def __init__(self, target, max_hops=30):
        self.target = target
        self.ttl = 1
        self.max_hops = max_hops

    def startProtocol(self):
        self.trace()

    def trace(self):
        self.transport.socket.setsockopt(0, 2, self.ttl)  # Set TTL
        self.transport.write(b'TRACE', (self.target, 33434))
        print(f'{self.ttl}: Sending packet to {self.target}')
        reactor.callLater(1, self.check_timeout)

    def datagramReceived(self, data, addr):
        print(f'{self.ttl}: Reply from {addr[0]}')
```

```
        if addr[0] == self.target or self.ttl >= self.max_hops:
            reactor.stop()
        else:
            self.ttl += 1
            self.trace()

    def check_timeout(self):
        if self.ttl < self.max_hops:
            print(f'{self.ttl}: *')
            self.ttl += 1
            self.trace()
        else:
            reactor.stop()

def resolve_target(hostname):
    d = client.lookupAddress(hostname)
    d.addCallback(lambda result: result[0][0].payload.dottedQuad())
    return d

def run_ping(hostname):
    d = resolve_target(hostname)
    d.addCallback(lambda ip: reactor.listenUDP(0, PingClient(ip)))

def run_traceroute(hostname):
    d = resolve_target(hostname)
    d.addCallback(lambda ip: reactor.listenUDP(0, TracerouteClient(ip)))

if __name__ == '__main__':
    import sys
    if len(sys.argv) < 3 or sys.argv[1] not in ['ping', 'traceroute']:
        print('Usage: python script.py [ping|traceroute] <hostname>')
        sys.exit(1)

    if sys.argv[1] == 'ping':
        run_ping(sys.argv[2])
    else:
        run_traceroute(sys.argv[2])
```

```
C:\Users\mslat\OneDrive\Documents>python "C:\Users\mslat\OneDrive\Documents\network ex 3.py" ping google.com
Pinging 216.58.200.142...
```

Result:

The code for ping and traceroute has been typed and executed successfully.

Exercise 4:

Study of socket programming and client server model

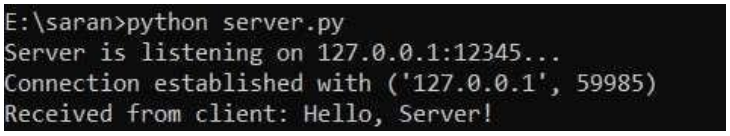
Implement a basic socket programming using client-server communication over a network.

Server :

```
import socket

HOST = '127.0.0.1'
PORT = 12345

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))

server_socket.listen(5)
print(f"Server is listening on {HOST}:{PORT}...")
```

```
while True:
    client_socket, client_address = server_socket.accept()
    print(f"Connection established with {client_address}")

    message = client_socket.recv(1024).decode('utf-8')
    print(f"Received from client: {message}")

    response = "Message received!"
    client_socket.send(response.encode('utf-8'))
    client_socket.close()
```

Output:



Client:

```
import socket

HOST = '127.0.0.1'
PORT = 12345

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))
```

```
message = "Hello, Server!"
client_socket.send(message.encode('utf-8'))

response = client_socket.recv(1024).decode('utf-8')
print(f"Server says: {response}")
client_socket.close()
```

Output:

UIT 2411 – NETWORK PROGRAMMING LAB

EXERCISE 5

Conducting Network Packet Capturing Using Wireshark

WIRESHARK:

- Wireshark is a free and open-source network protocol analyzer used to capture and inspect network traffic in real-time.
- It helps in understanding how data is transmitted over networks and is widely used for network troubleshooting, security analysis, and protocol debugging.

FEATURES:

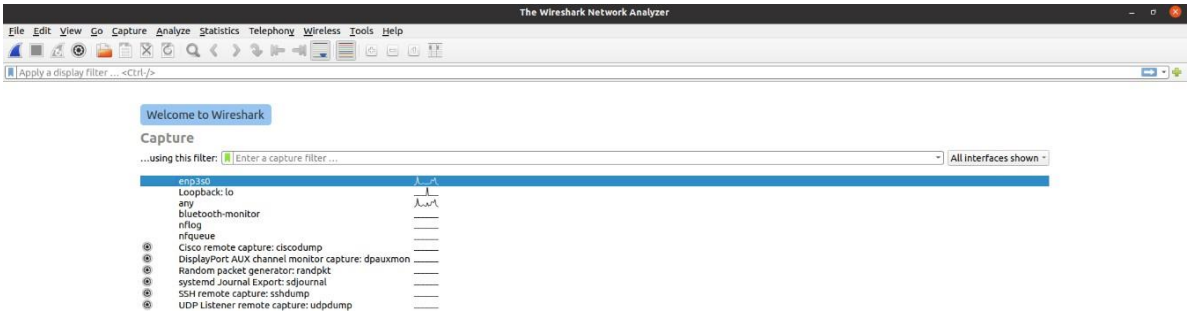
- Captures live network traffic from Ethernet, Wi-Fi, and other interfaces.
- Allows users to inspect and filter specific packets for detailed analysis.
- Supports Multiple Protocols – Works with TCP, UDP, HTTP, DNS, ARP, ICMP, and many other protocols.
- Graphical & Command-Line Interface – Provides both a user-friendly GUI (Wireshark) and a CLI tool (tshark).
- Export & Save Captures – Saves captured network data in .pcap files for future analysis

STEPS:

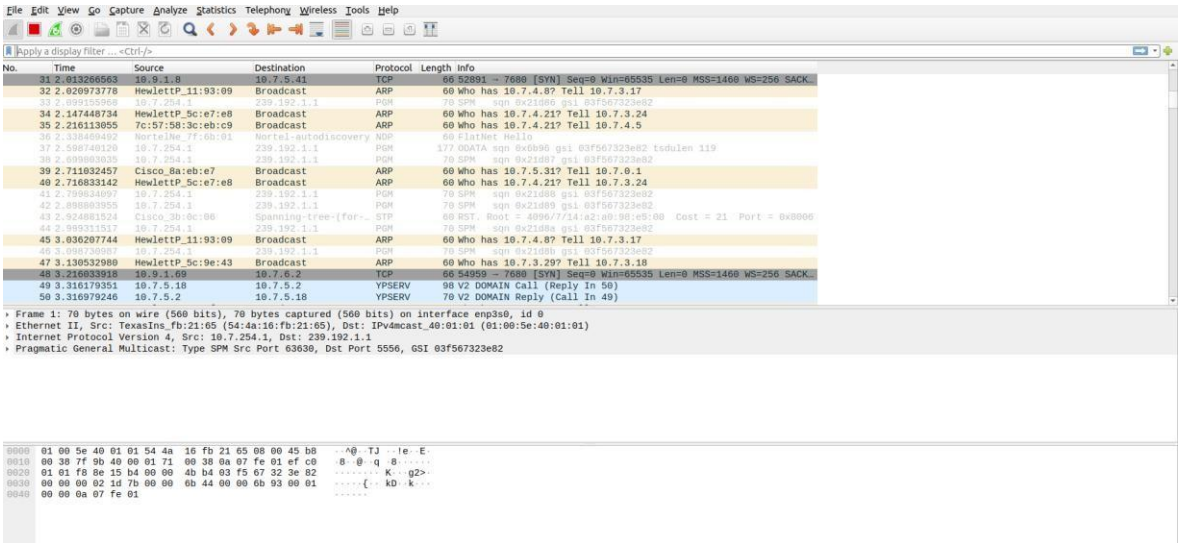
1. Open Wireshark and choose the network interface (Wi-Fi, Ethernet, etc.).
2. Click on the Start button to start monitoring live network traffic.
3. Use filters to target certain traffic (e.g., port 80 for HTTP).
4. Interpret captured packets by choosing and checking information such as source/destination IP and protocol.
5. Click Stop when finished, then save the capture as .pcap file for later analysis.

IMPLEMENTATION:

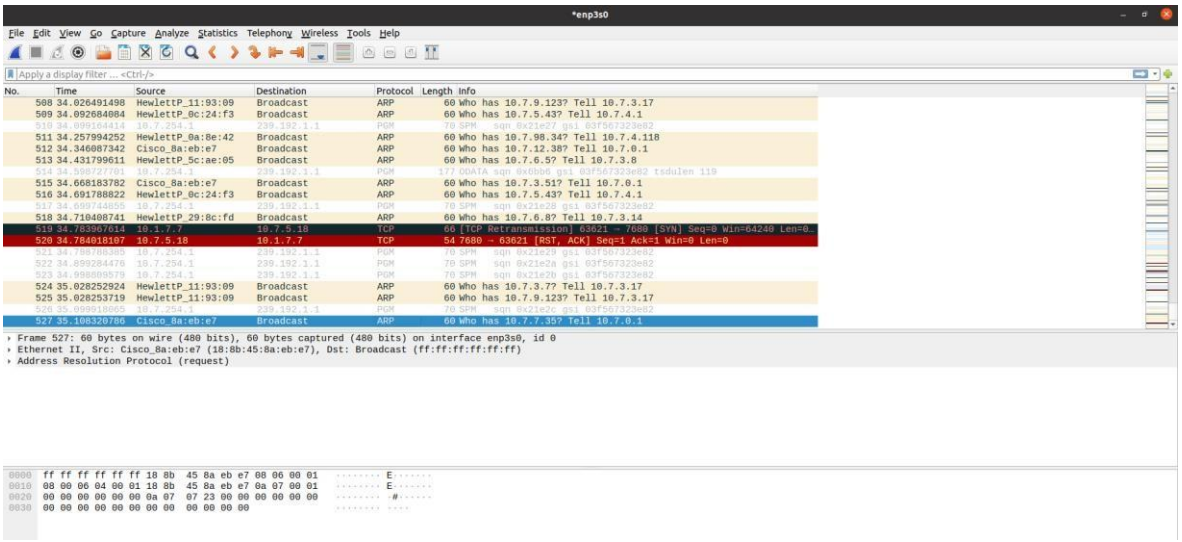
a. Wireshark Window:



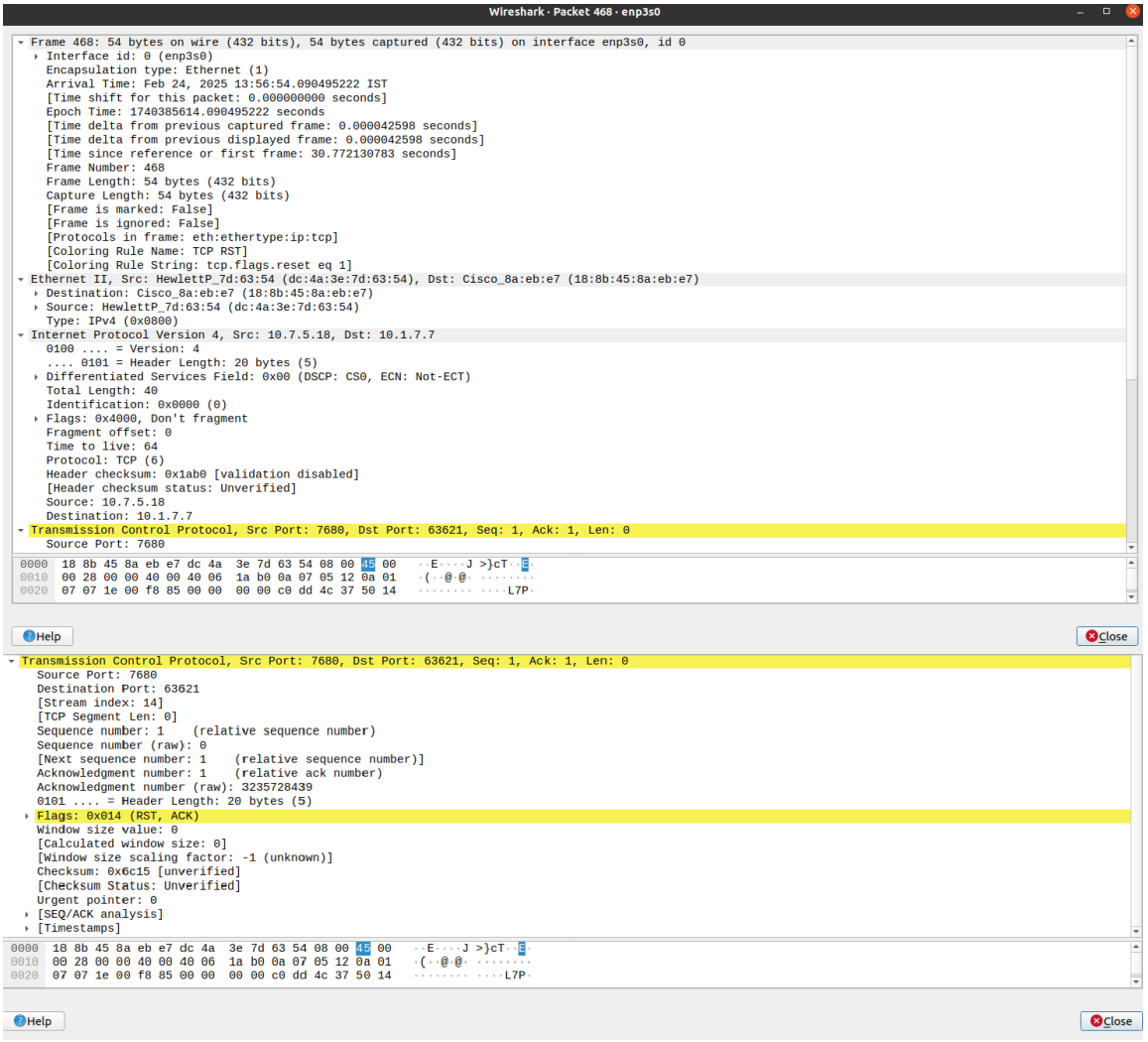
b. Capturing Packets:



c. Captured packets :

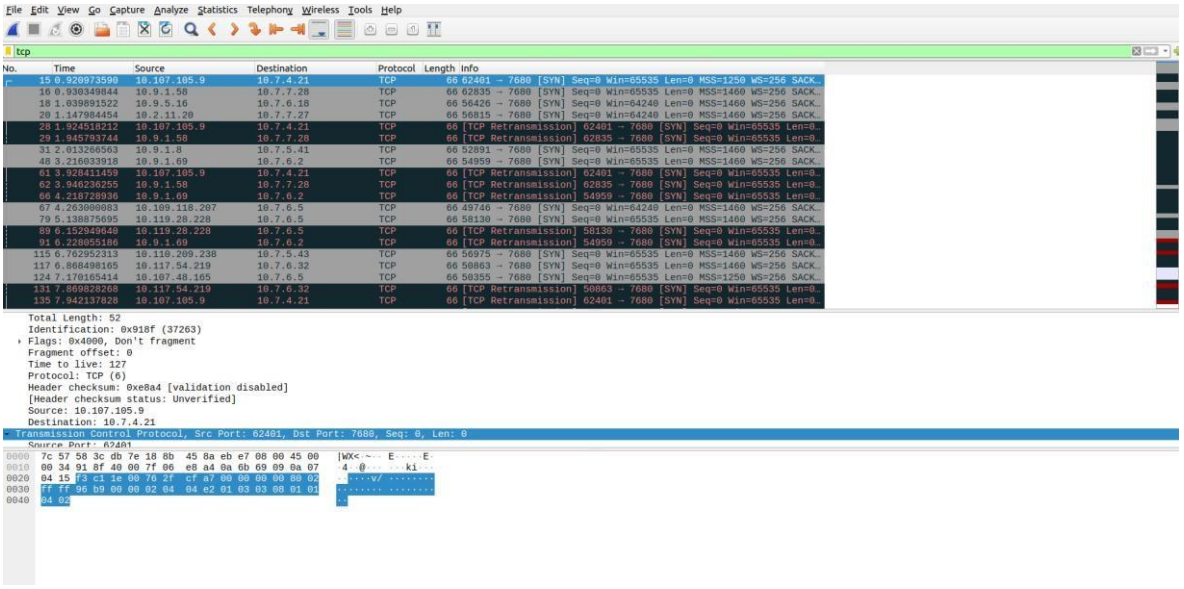


d. Details about the packets : (TCP Protocol)

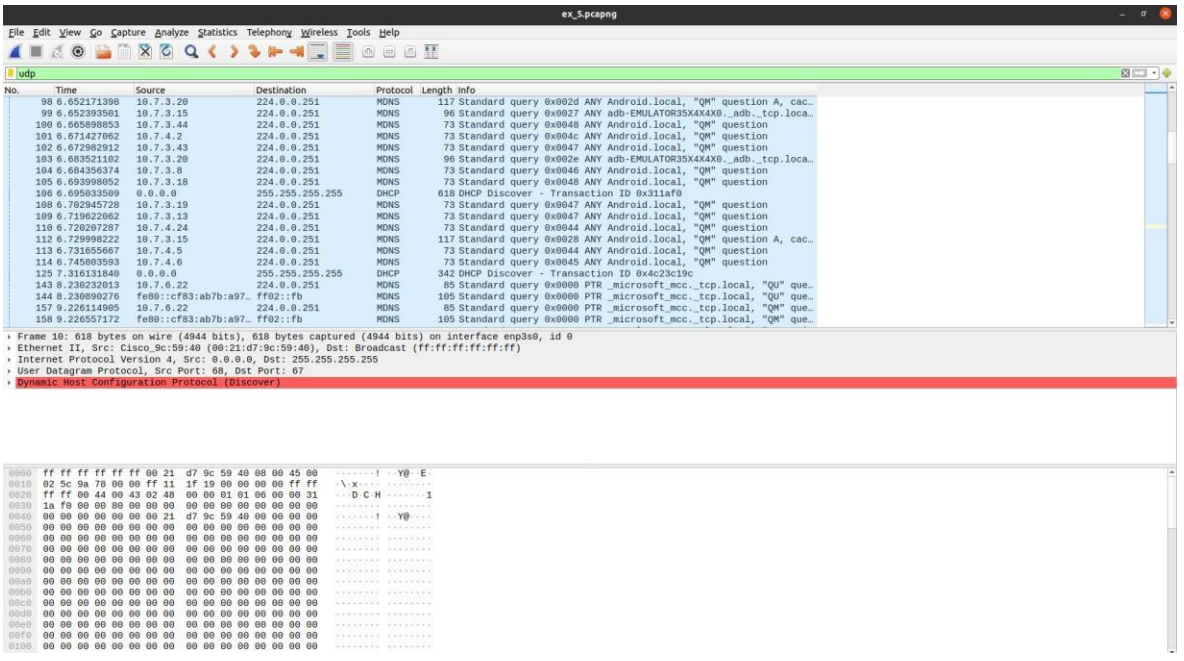


Applying Filters

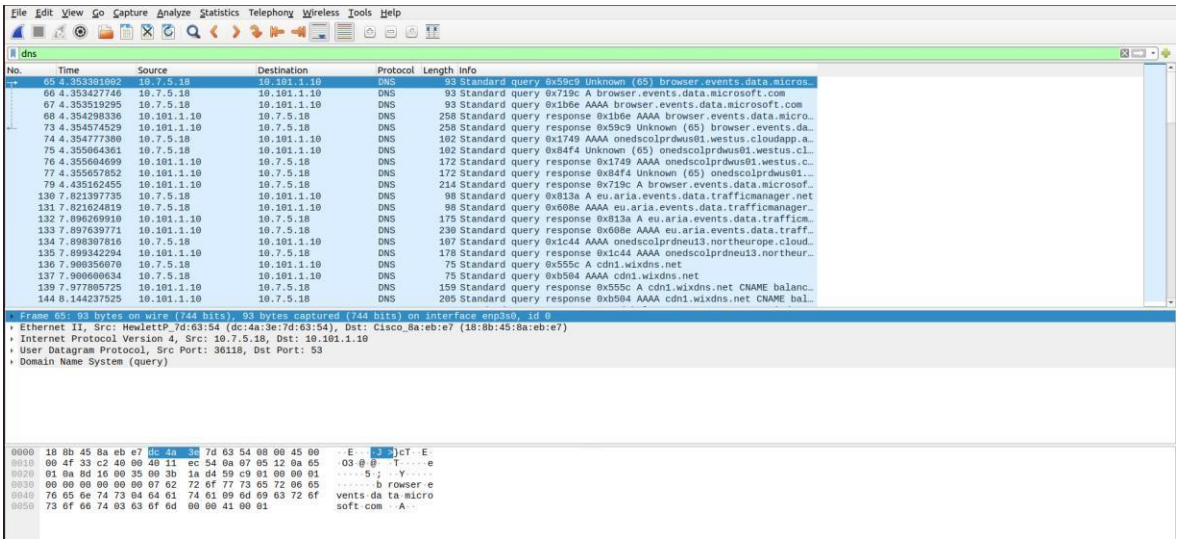
e. TCP Protocol : (Transmission Control Protocol)



f. UDP Protocol : (User Datagram Protocol)



g. DNS Protocol : (Domain Name System)



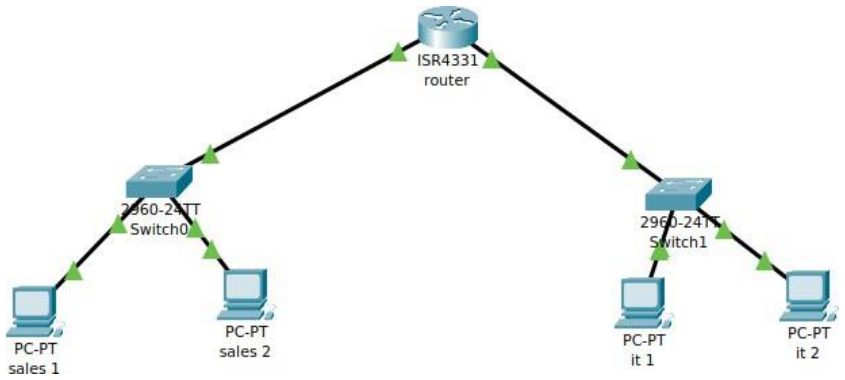
UIT 2411 – NETWORK PROGRAMMING LAB

EXERCISE 6

Network Simulation using Cisco Packet Tracer

1. You are a network administrator tasked with setting up a small office network with **two departments (Sales and IT)**. Each department has **two PCs**, connected via a switch and a router. Your goal is to **assign IP addresses, configure the devices, and verify connectivity**.

Connection:

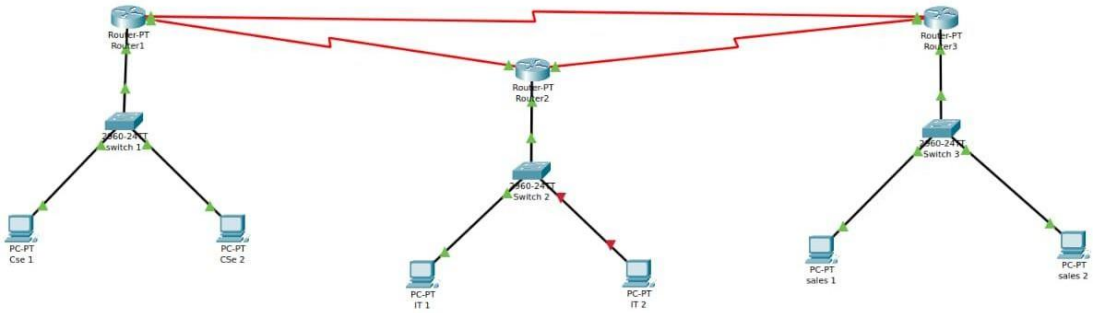


Ping to verify connectivity:

```
Cisco Packet Tracer PC Command Line 1.0
C:\>ping 192.168.20.1

Pinging 192.168.20.1 with 32 bytes of data:

Reply from 192.168.20.1: bytes=32 time<1ms TTL=255
Reply from 192.168.20.1: bytes=32 time<1ms TTL=255
Reply from 192.168.20.1: bytes=32 time<1ms TTL=255
Reply from 192.168.20.1: bytes=32 time<1ms TTL=255

Ping statistics for 192.168.20.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>ping 192.168.10.2

Pinging 192.168.10.2 with 32 bytes of data:

Reply from 192.168.10.2: bytes=32 time<1ms TTL=127
Reply from 192.168.10.2: bytes=32 time<1ms TTL=127
Reply from 192.168.10.2: bytes=32 time<1ms TTL=127
Reply from 192.168.10.2: bytes=32 time<1ms TTL=127

Ping statistics for 192.168.10.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

2. You are a network engineer setting up communication between three branch offices connected via three routers (R1, R2, and R3). Each router manages a different network. Your goal is to configure routing so that all networks can communicate.

Connection:



Ping to verify connectivity:

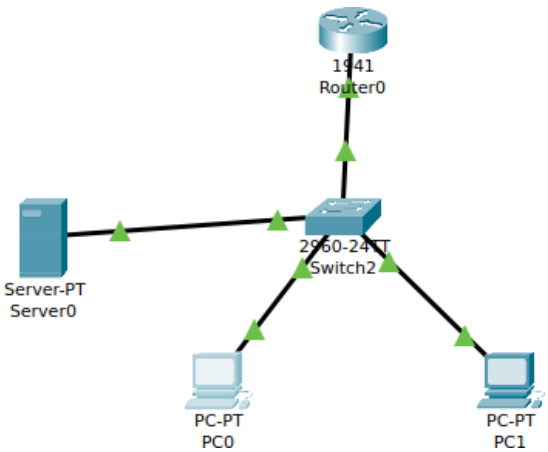
```
C:\>ping 192.168.10.1

Pinging 192.168.10.1 with 32 bytes of data:

Reply from 192.168.10.1: bytes=32 time<1ms TTL=255
Reply from 192.168.10.1: bytes=32 time<1ms TTL=255
Reply from 192.168.10.1: bytes=32 time<1ms TTL=255
Reply from 192.168.10.1: bytes=32 time<1ms TTL=255

Ping statistics for 192.168.10.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

3. A company has a server that provides:

- Web Services (HTTP - TCP)
- DNS Services (UDP)
- FTP Services (TCP)
- VoIP Services (UDP)

Two clients (PC1 & PC2) will communicate with the server using different protocols

Connection:



Ping to verify connectivity:

```
C:\>ping 192.168.20.2

Pinging 192.168.20.2 with 32 bytes of data:

Reply from 192.168.20.2: bytes=32 time<1ms TTL=127
Reply from 192.168.20.2: bytes=32 time<1ms TTL=127
Reply from 192.168.20.2: bytes=32 time=1ms TTL=127
Reply from 192.168.20.2: bytes=32 time<1ms TTL=127

Ping statistics for 192.168.20.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms
```

RESULT:
The above tooling concepts are executed succcessfully

EX NO : 7

Applications using TCP Sockets like – Twisted Python

a. Echo Client/Server

CODING:

echoserver.py:

```
from twisted.internet import reactor, protocol

class EchoServer(protocol.Protocol):
    def dataReceived(self, data):
        """Called when data is received from a client."""
        print(f"Received from client: {data.decode()}")
        self.transport.write(data)  # Echo the message back

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return EchoServer()

# Run the server on port 12345
print("Twisted Echo Server is running on port 12345...")
reactor.listenTCP(12345, EchoFactory())
reactor.run()
```

echoclient.py:

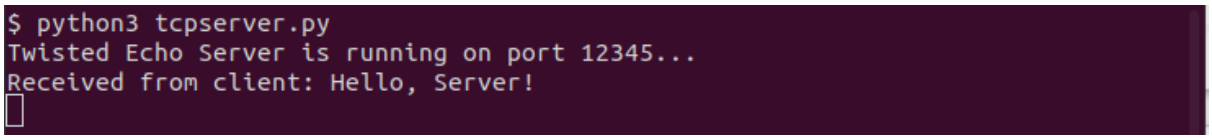
```
from twisted.internet import reactor, protocol
```

```
class EchoClient(protocol.Protocol):
    def connectionMade(self):
        """Called when the connection is established."""
        message = "Hello, Server!"
        print(f"Sending: {message}")
        self.transport.write(message.encode())  # Send message

    def dataReceived(self, data):
        """Called when data is received from the server."""
        print(f"Received from server: {data.decode()}")
        self.transport.loseConnection()  # Close connection after receiving
response

class EchoClientFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return EchoClient()

    def clientConnectionFailed(self, connector, reason):
        print("Connection failed!")
        reactor.stop()

    def clientConnectionLost(self, connector, reason):
        print("Connection closed!")
        reactor.stop()

# Connect to the server at localhost:12345
reactor.connectTCP("localhost", 12345, EchoClientFactory())
```

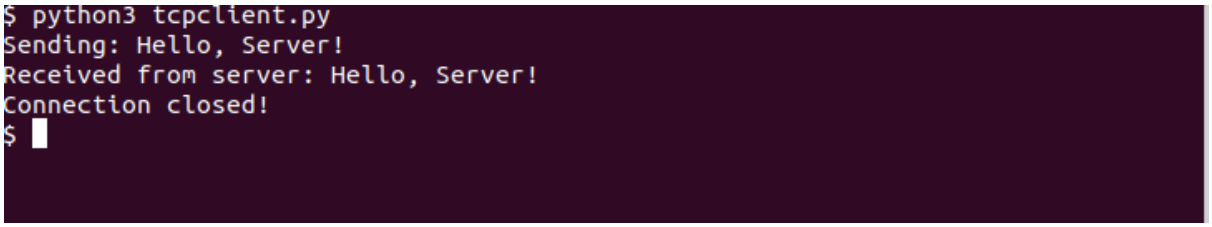
reactor.run()

OUTPUT:

Server end:



Client end:



b. Multiple Client/Server

CODING:

mulserver.py:

```
from twisted.internet import reactor, protocol


class EchoServer(protocol.Protocol):
    def connectionMade(self):
        """Called when a new client connects."""
        self.client_ip = self.transport.getPeer().host
        print(f"New connection from {self.client_ip}")

    def dataReceived(self, data):
        """Called when data is received from a client."""
        print(f"Received from {self.client_ip}: {data.decode()}")
        self.transport.write(data)  # Echo the message back

    def connectionLost(self, reason):
        """Called when a client disconnects."""
```

```
        print(f"Connection closed with {self.client_ip}")

class EchoFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return EchoServer()

# Run the server on port 12345
print("Twisted Multi-Client Echo Server is running on port 12345...")
reactor.listenTCP(12345, EchoFactory())
reactor.run()
```

mulclient.py:

```
from twisted.internet import reactor, protocol

class EchoClient(protocol.Protocol):
    def connectionMade(self):
        """Called when the connection is established."""
        print("Connected to the server.")
        self.sendMessage()

    def sendMessage(self):
        """Send a message to the server."""
        message = input("Enter message: ")  # Take user input
        self.transport.write(message.encode())

    def dataReceived(self, data):
        """Called when data is received from the server."""
```

```
        print(f"Server echoed: {data.decode()}")
        self.sendMessage()  # Ask for new message input

    def connectionLost(self, reason):
        """Called when the connection is lost."""
        print("Disconnected from the server.")

class EchoClientFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return EchoClient()

    def clientConnectionFailed(self, connector, reason):
        print("Connection failed!")
        reactor.stop()

    def clientConnectionLost(self, connector, reason):
        print("Connection closed!")
        reactor.stop()

# Connect to the server at localhost:12345
reactor.connectTCP("localhost", 12345, EchoClientFactory())
reactor.run()
```

OUTPUT:

Server end:

```
$ python3 tcp2server.py
Twisted Multi-Client Echo Server is running on port 12345...
New connection from 127.0.0.1
Received from 127.0.0.1: hii
Received from 127.0.0.1: i am smriti
Received from 127.0.0.1: From it dept
```

Client end:

```
$ python3 tcp2client.py
Connected to the server.
Enter message: hii
Server echoed: hii
Enter message: i am smriti
Server echoed: i am smriti
Enter message: From it dept
Server echoed: From it dept
Enter message:
```

c. Chat

CODING:

chatserver.py:

```
from twisted.internet import reactor, protocol


class ChatServer(protocol.Protocol):
    clients = []  # List to store connected clients

    def connectionMade(self):
        """Called when a new client connects."""
        self.clients.append(self)  # Add client to list
        print("New client connected.")
        self.sendMessage("Welcome to the Chat Server!\n")
        self.broadcastMessage("A new user has joined the chat.\n", sender=self)

    def dataReceived(self, data):
        """Called when data is received from a client."""
```

```
        message = data.decode().strip()
        print(f"Received: {message}")
        self.broadcastMessage(f"{message}\n", sender=self)

    def connectionLost(self, reason):
        """Called when a client disconnects."""
        self.clients.remove(self)  # Remove client from list
        print("A client has disconnected.")
        self.broadcastMessage("A user has left the chat.\n", sender=self)

    def sendMessage(self, message):
        """Send a message to the current client."""
        self.transport.write(message.encode())

    def broadcastMessage(self, message, sender=None):
        """Send a message to all clients except the sender."""
        for client in self.clients:
            if client != sender:
                client.sendMessage(message)

class ChatFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return ChatServer()

# Start the server on port 12345
print("Twisted Chat Server is running on port 12345...")
reactor.listenTCP(12345, ChatFactory())
```

reactor.run()

chatclient.py:

```
from twisted.internet import reactor, protocol
import threading
import sys

class ChatClient(protocol.Protocol):
    def connectionMade(self):
        """Called when connected to the server."""
        print("Connected to the chat server.")
        self.listenForInput()

    def dataReceived(self, data):
        """Called when data is received from the server."""
        message = data.decode().strip()
        print(f"\n{message}\n> ", end="", flush=True)  # Display message and
prompt

    def sendMessage(self, message):
        """Send a message to the server."""
        self.transport.write(message.encode())

    def listenForInput(self):
        """Runs in a separate thread to handle user input."""
        def inputLoop():
            while True:
                message = input("> ")
```

```
            if message.lower() == "exit":
                print("Disconnecting...")
                self.transport.loseConnection()
                reactor.stop()
                break
            self.sendMessage(message)
        threading.Thread(target=inputLoop, daemon=True).start()

class ChatClientFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return ChatClient()

    def clientConnectionFailed(self, connector, reason):
        print("Connection failed!")
        reactor.stop()

    def clientConnectionLost(self, connector, reason):
        print("Connection closed!")
        reactor.stop()

# Connect to the server at localhost:12345
reactor.connectTCP("localhost", 12345, ChatClientFactory())
reactor.run()
```

OUTPUT:

Server end:

```
$ python3 tcpchatserver.py
Twisted Chat Server is running on port 12345...
New client connected.
Received: heyy welcome
Received: this is Smriti this side
Received: bye
```

Client end:

```
$ python3 tcpchatclient.py
Connected to the chat server.
>
Welcome to the Chat Server!
> heyy welcome
> this is Smriti this side
> bye
>
```

d.File Transfer

CODING:

filetranserver.py:

```
from twisted.internet import reactor, protocol


class FileTransferServer(protocol.Protocol):
    def connectionMade(self):
        """Called when a new client connects."""
        self.file = open("received_file.txt", "wb")  # Save the file with this name
        print("Client connected, waiting for file...")


    def dataReceived(self, data):
        """Receive file data from the client."""
        self.file.write(data)
        print(f"Received {len(data)} bytes...")


    def connectionLost(self, reason):
```

```
        """Called when the client disconnects."""
        self.file.close()
        print("File transfer complete. Client disconnected.")

class FileServerFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return FileTransferServer()

# Start the server on port 12345
print("Twisted File Transfer Server is running on port 12345...")
reactor.listenTCP(12345, FileServerFactory())
reactor.run()
```

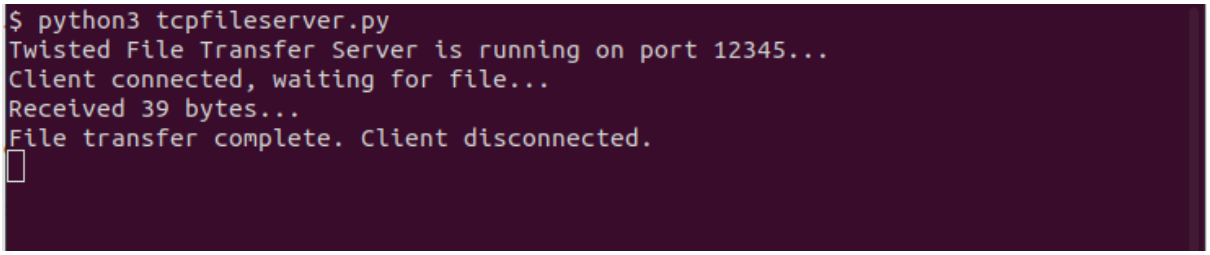
filetransclient.py:

```
from twisted.internet import reactor, protocol

class FileTransferClient(protocol.Protocol):
    def connectionMade(self):
        """Send file when connected to the server."""
        filename = "FILE.txt"  # Change this to the file you want to send
        try:
            with open(filename, "rb") as file:
                self.transport.write(file.read())  # Send the file
                print(f"Sent file: {filename}")
        except FileNotFoundError:
            print("File not found! Please check the filename.")
        self.transport.loseConnection()
```

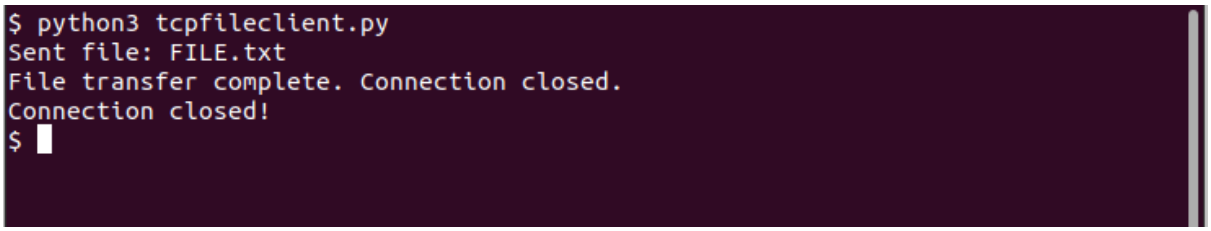
```
    def connectionLost(self, reason):
        """Called when the transfer is complete."""
        print("File transfer complete. Connection closed.")

class FileClientFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return FileTransferClient()

    def clientConnectionFailed(self, connector, reason):
        print("Connection failed!")
        reactor.stop()

    def clientConnectionLost(self, connector, reason):
        print("Connection closed!")
        reactor.stop()

# Connect to the server at localhost:12345
reactor.connectTCP("localhost", 12345, FileClientFactory())
reactor.run()
```

OUTPUT:

Server end:



Client end:


$ python3 tcpfileclient.py
Sent file: FILE.txt
File transfer complete. Connection closed.
Connection closed!
$

Ex No.: 8 Date: / / 2025

Applications using UDP Sockets like - Twisted Python

AIM:

To implement various applications on UDP sockets with twisted python.

a.Echo Client/Server:

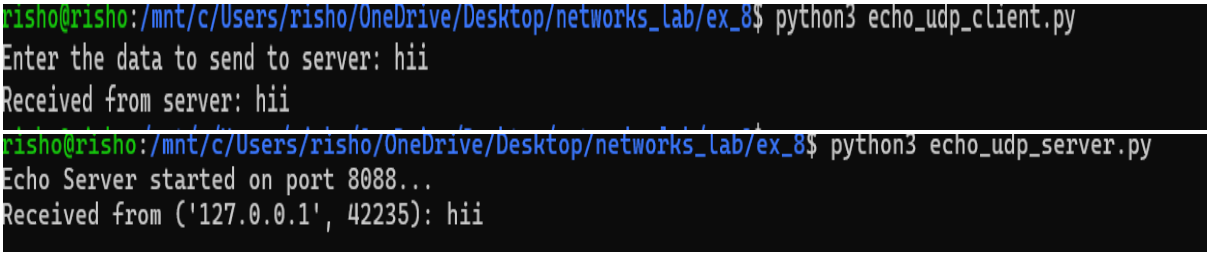
CODE (server.py):

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPEchoServer(DatagramProtocol):
    def datagramReceived(self, data, addr):
        """Called when a message is received from a client."""
        message = data.decode()
        print(f"Received from {addr}: {message}")
        self.transport.write(data, addr)  # Echo back

# Start UDP server on port 8000
reactor.listenUDP(8000, UDPEchoServer())
print("UDP Echo Server running on port 8000...")
reactor.run()
```

CODE (client.py):

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPEchoClient(DatagramProtocol):
    def startProtocol(self):
        """Called when the client starts."""
        self.transport.write(b"Hello, Server!", ("127.0.0.1", 8000))

    def datagramReceived(self, data, addr):
        """Handles the echoed response from the server."""
        print(f"Received from server: {data.decode()}")
        reactor.stop()  # Stop after receiving response

# Start UDP client
reactor.listenUDP(0, UDPEchoClient())  # Use an ephemeral port
reactor.run()
```

OUTPUT:



b. Multiple Client/Server:

CODE (server.py):

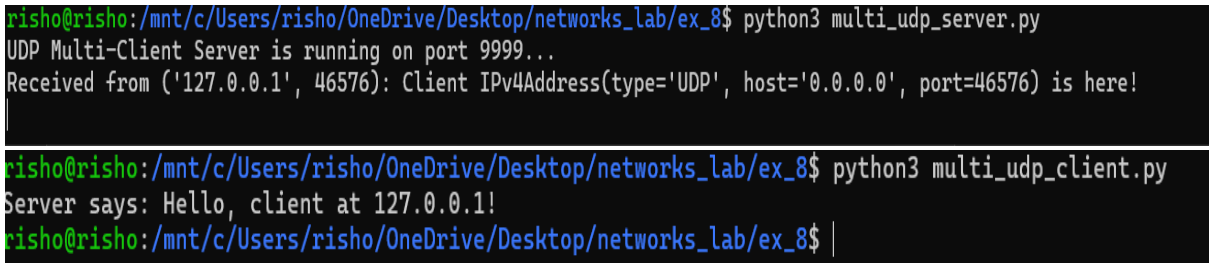
```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPEchoServer(DatagramProtocol):
    def datagramReceived(self, data, addr):
        """Called when a message is received from a client."""
        message = data.decode()
        print(f"Received from {addr}: {message}")

        # Echo the message back to the same client
        self.transport.write(data, addr)

# Start UDP server on port 8000
reactor.listenUDP(8000, UDPEchoServer())
print("UDP Echo Server running on port 8000...")
reactor.run()
```

CODE (client.py):

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPEchoClient(DatagramProtocol):
    def __init__(self, message):
        self.message = message

    def startProtocol(self):
        """Called when the client starts."""
        self.transport.write(self.message.encode(), ("127.0.0.1", 8000))

    def datagramReceived(self, data, addr):
        """Handles the echoed response from the server."""
        print(f"Received from server: {data.decode()}")
```

```
        reactor.stop()  # Stop after receiving response

# Start multiple clients with different messages
def start_clients():
    reactor.listenUDP(0, UDPEchoClient("Hello from Client 1!"))
    reactor.listenUDP(0, UDPEchoClient("Hello from Client 2!"))
    reactor.listenUDP(0, UDPEchoClient("Hello from Client 3!"))

start_clients()
reactor.run()
```

OUTPUT:



c. Chat:

CODE (server.py):

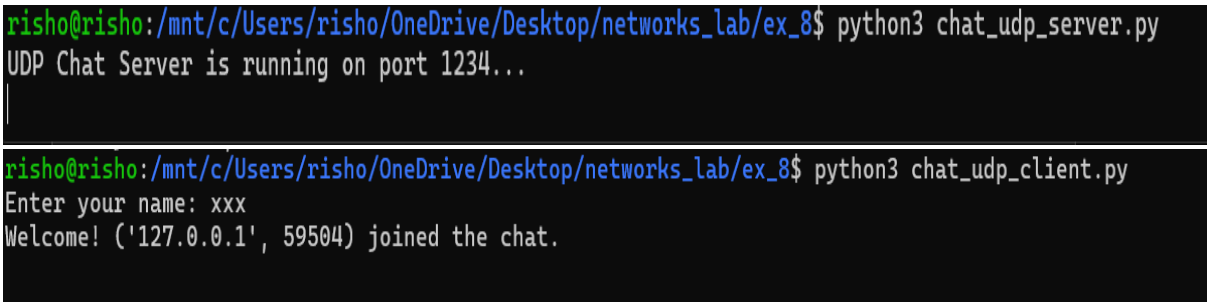
```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPChatServer(DatagramProtocol):
    def __init__(self):
        self.clients = set()  # Track connected clients

    def datagramReceived(self, data, addr):
        """Called when a message is received from a client."""
        message = data.decode().strip()
        print(f"Client {addr}: {message}")

        if addr not in self.clients:
            self.clients.add(addr)

        if message.lower() == "bye":
            print(f"Client {addr} disconnected.")
            self.clients.remove(addr)
            return
```

```
        response = input("Server: ")  # Server replies
        self.transport.write(response.encode(), addr)

# Start UDP server on port 8000
reactor.listenUDP(8000, UDPChatServer())
print("UDP Chat Server running on port 8000...")
reactor.run()
```

CODE (client.py):

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPChatClient(DatagramProtocol):
    def startProtocol(self):
        """Called when the client starts."""
        self.server_addr = ("127.0.0.1", 8000)
        self.sendMessage()

    def datagramReceived(self, data, addr):
        """Handles incoming messages from the server."""
        print(f"Server: {data.decode()}")

        if data.decode().strip().lower() == "bye":
            print("Disconnected from server.")
            reactor.stop()
            return

        self.sendMessage()

    def sendMessage(self):
        """Take user input and send messages."""
        message = input("You: ")
        self.transport.write(message.encode(), self.server_addr)

        if message.lower() == "bye":
            reactor.stop()

# Start UDP client
reactor.listenUDP(0, UDPChatClient())  # Use an ephemeral port
reactor.run()
```

OUTPUT:



d. File Transfer:

CODE (server.py):

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPFileServer(DatagramProtocol):
    def __init__(self):
        self.file = open("received_file.txt", "wb")  # File to save received data

    def datagramReceived(self, data, addr):
        """Called when a chunk of the file is received from the client."""
        if data == b"EOF":  # End of file signal
            print("File transfer complete. Closing file.")
            self.file.close()
            return

        self.file.write(data)  # Write received chunk to file
        print(f"Receiving file chunk from {addr}")

# Start UDP server on port 8000
reactor.listenUDP(8000, UDPFileServer())
print("UDP File Server running on port 8000...")
reactor.run()
```

CODE (client.py):

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol

class UDPFileClient(DatagramProtocol):
    def __init__(self, filename):
```

```
        self.filename = filename
        self.server_addr = ("127.0.0.1", 8000)

    def startProtocol(self):
        """Called when the client starts. Begins file transfer."""
        try:
            with open(self.filename, "rb") as f:
                while chunk := f.read(1024):  # Read in chunks of 1024 bytes
                    self.transport.write(chunk, self.server_addr)
                    print("Sent a chunk...")
        except FileNotFoundError:
            print("File not found!")

        self.transport.write(b"EOF", self.server_addr)  # End of file signal
        print("File transfer complete.")
        reactor.stop()

# Start UDP client and send "sample.txt"
reactor.listenUDP(0, UDPFileClient("sample.txt"))  # Use an ephemeral port
reactor.run()
```

OUTPUT:

```
risho@risho:/mnt/c/Users/risho/OneDrive/Desktop/networks_lab/ex_8$ python3 file_transfer_udp_server.py
UDP File Transfer Server running on port 8088...
```

```
risho@risho:/mnt/c/Users/risho/OneDrive/Desktop/networks_lab/ex_8$ python3 file_transfer_udp_client.py
File Transferred! Waiting for ACK...
```

RESULT:

Thus the UDP applications of echo client server,chat and ftp are implemented in Twisted Python.

Ex No.: 9 Date: / / 2025

ARP/RARP protocols using Twisted Python

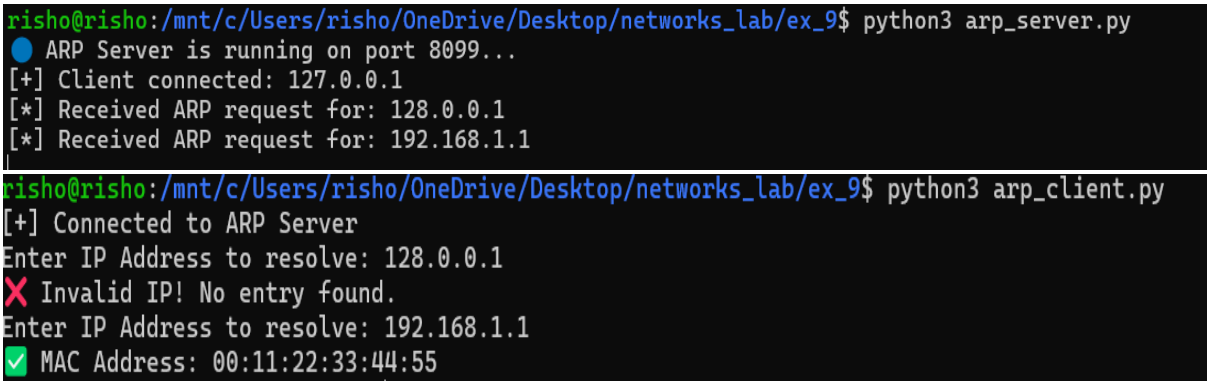
AIM:

To implement the ARP and RARP protocols in python using twisted module

i) ARP protocol - CODE:

```
from twisted.protocols.basic import LineReceiver
from twisted.internet.protocol import Factory
from twisted.internet import reactor

class ARP_Protocol(LineReceiver):
    def __init__(self):
        self.mappings = {
            '192.168.1.67': 'AA:AA:AA:AA:AA:AA',
            '192.168.1.34': 'B2:C3:A1:DF:JK:L3'
        }

    def connectionMade(self):
        self.sendLine(b"Enter the IP address to find MAC address:")

    def lineReceived(self, data):
        self.get_addr(data.decode().strip())

    def get_addr(self, addr):
        mac_addr = self.mappings.get(addr, None)
        if mac_addr is None:
            self.sendLine(b"IP address not found")
        else:
            self.sendLine(f"The MAC address for {addr} is: {mac_addr}".encode())

class ARP_Factory(Factory):
    def buildProtocol(self, addr):
        return ARP_Protocol()

if __name__ == "__main__":
    reactor.listenTCP(8000, ARP_Factory())
    print("ARP Lookup Server running on port 8000...")
    reactor.run()
```

OUTPUT:

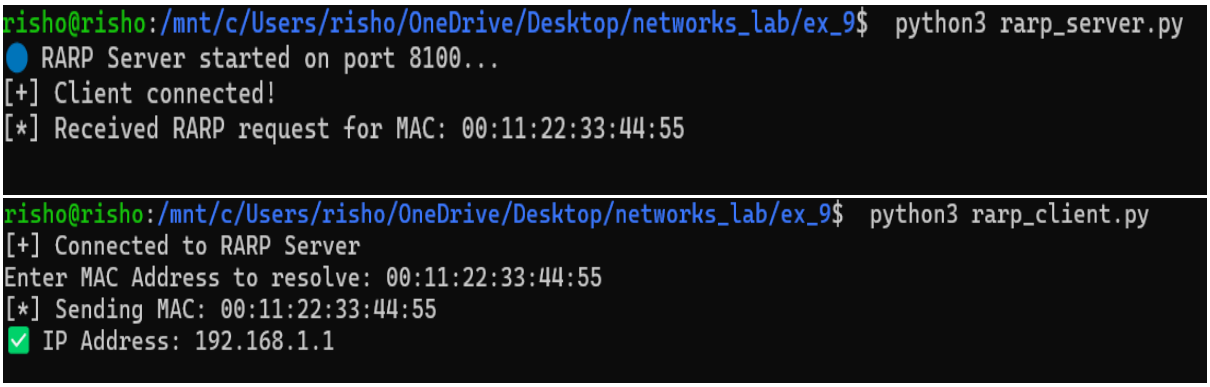


ii) RARP protocol - CODE:

```
from twisted.protocols.basic import LineReceiver
from twisted.internet.protocol import Factory
from twisted.internet import reactor

class RARP_Protocol(LineReceiver):
    def __init__(self):
        self.mappings = {
            'AA:AA:AA:AA:AA:AA': '192.168.1.67',
            'B2:C3:A1:DF:JK:L3': '192.168.1.34'
        }

    def connectionMade(self):
        """Prompt the user to enter a MAC address upon connection."""
        self.sendLine(b"Enter the MAC address to find IP address:")

    def lineReceived(self, data):
        """Process the input MAC and return the corresponding IP address."""
        self.get_addr(data.decode().strip())

    def get_addr(self, addr):
        """Find and send the IP address corresponding to the given MAC."""
        ip_addr = self.mappings.get(addr, None)
        if ip_addr is None:
            self.sendLine(b"MAC address not found")
        else:
            self.sendLine(f"The IP address for {addr} is: {ip_addr}".encode())

class RARP_Factory(Factory):
    def buildProtocol(self, addr):
```

```
        return RARP_Protocol()

if __name__ == "__main__":
    reactor.listenTCP(8000, RARP_Factory())
    print("RARP Lookup Server running on port 8000...")
    reactor.run()
```

OUTPUT:



RESULT:

Thus the ARP and RARP protocols are implemented in python using twisted module and the outputs are recorded.

Ex No.: 10 Date: / / 2025

Implementation of Stop and Wait Protocol and Sliding Window Protocol using Twisted Python

AIM:

To implement the Stop and wait protocol and Sliding window protocol in twisted python.

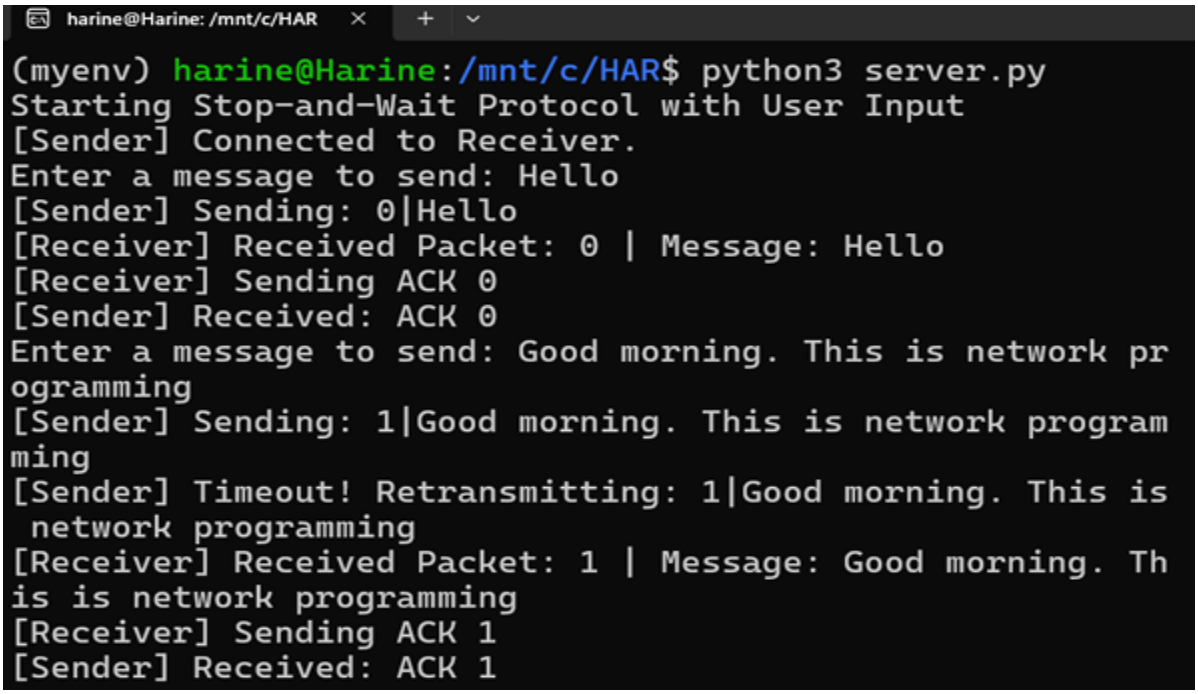
i) Stop and Wait Protocol - CODE:

```
from twisted.internet import reactor, protocol
import time

# Sender (Client)
class SenderProtocol(protocol.Protocol):
    def __init__(self):
        self.seq_num = 0  # Sequence number (0 or 1)
        self.timeout = 3   # Timeout in seconds
        self.ack_received = True

    def connectionMade(self):
        print("[Sender] Connected to Receiver.")
        self.getUserInput()

    def getUserInput(self):
        """Prompt user for a message and send it."""
        self.message = input("Enter a message to send: ")
        self.sendPacket()

    def sendPacket(self):
        """Send the current message with a sequence number."""
        if self.ack_received:
            self.ack_received = False
            packet = f"{self.seq_num}|{self.message}"
            print(f"[Sender] Sending: {packet}")
            self.transport.write(packet.encode())
            reactor.callLater(self.timeout, self.checkTimeout)

    def dataReceived(self, data):
        """Handle received ACKs."""
        ack = data.decode()
        print(f"[Sender] Received: {ack}")
        if ack == f"ACK {self.seq_num}":
            self.ack_received = True
            self.seq_num = 1 - self.seq_num  # Toggle between 0 and 1
            self.getUserInput()  # Ask for next message
```

```
    def checkTimeout(self):
        """Retransmit if ACK is not received."""
        if not self.ack_received:
            print(f"[Sender] Timeout! Retransmitting: {self.seq_num}|{self.message}")
            self.sendPacket()

# Receiver (Server)
class ReceiverProtocol(protocol.Protocol):
    def __init__(self):
        self.expected_seq_num = 0

    def dataReceived(self, data):
        """Handle received packets."""
        message = data.decode()
        seq_num, received_message = message.split("|", 1)  # Extract sequence number and
message
        seq_num = int(seq_num)

        print(f"[Receiver] Received Packet: {seq_num} | Message: {received_message}")

        if seq_num == self.expected_seq_num:
            print(f"[Receiver] Sending ACK {seq_num}")
            self.transport.write(f"ACK {seq_num}".encode())
            self.expected_seq_num = 1 - self.expected_seq_num  # Toggle expected sequence
number
        else:
            print("[Receiver] Duplicate packet detected. Resending last ACK.")
            self.transport.write(f"ACK {1 - self.expected_seq_num}".encode())

# Factories for Sender and Receiver
class SenderFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return SenderProtocol()

class ReceiverFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return ReceiverProtocol()

# Start Sender and Receiver
def startSender():
    reactor.connectTCP("localhost", 8000, SenderFactory())
```

```
def startReceiver():
    reactor.listenTCP(8000, ReceiverFactory())
    reactor.callLater(2, startSender)  # Start sender after receiver is running

if __name__ == "__main__":
    print("Starting Stop-and-Wait Protocol with User Input")
    startReceiver()
    reactor.run()
```

OUTPUT:



ii) Sliding Window Protocol - CODE:

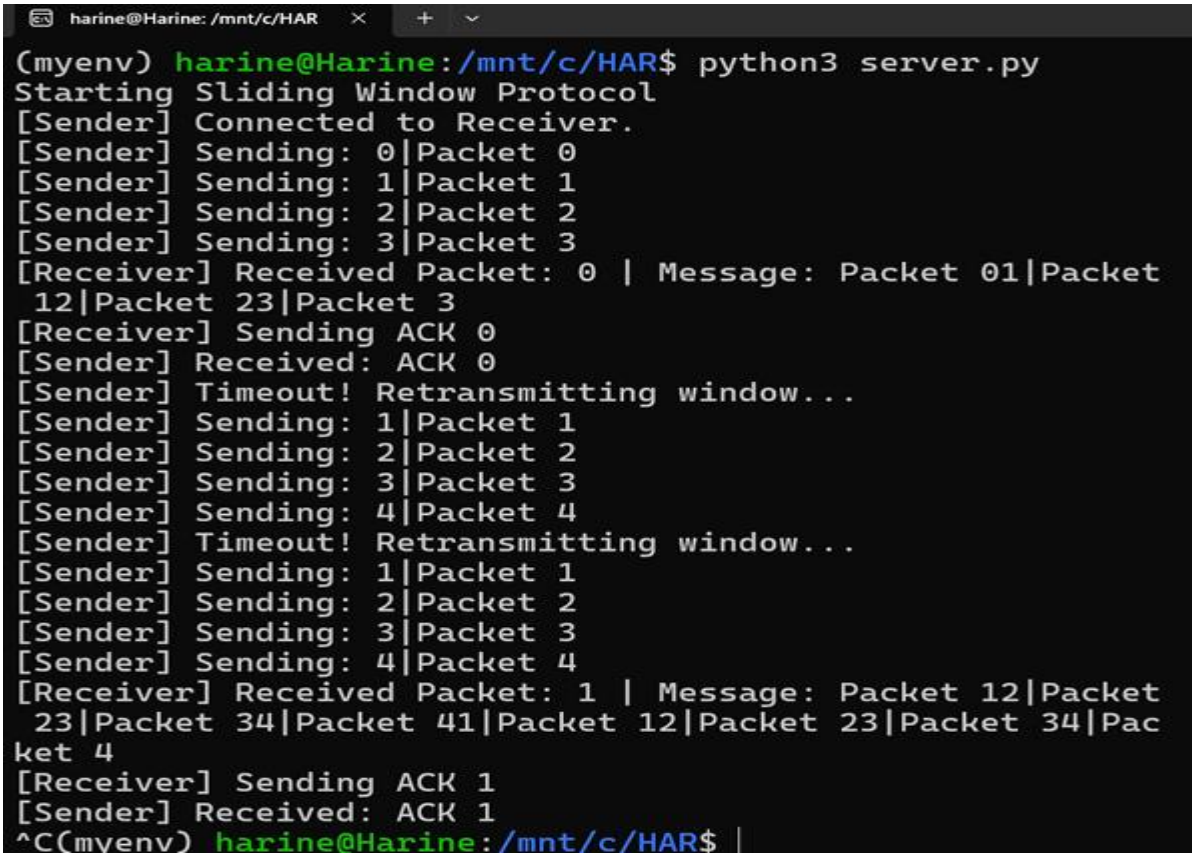
```
from twisted.internet import reactor, protocol
import time

# Constants
WINDOW_SIZE = 4  # Size of the sliding window
TIMEOUT = 3      # Timeout in seconds

# Sender (Client)
class SenderProtocol(protocol.Protocol):
    def __init__(self):
        self.base = 0         # Sequence number of the first packet in the window
```

```
    self.next_seq_num = 0  # Sequence number of the next packet to send
    self.buffer = [f"Packet {i}" for i in range(10)]  # Data to be sent
    self.window = []       # Current window of packets
    self.timer = None      # Timer for timeout
    self.is_waiting_for_ack = False

def connectionMade(self):
    print("[Sender] Connected to Receiver.")
    self.sendWindow()

def sendWindow(self):
    while self.next_seq_num < len(self.buffer) and len(self.window) < WINDOW_SIZE:
        packet = f"{self.next_seq_num}|{self.buffer[self.next_seq_num]}"
        print(f"[Sender] Sending: {packet}")
        self.transport.write(packet.encode())
        self.window.append(packet)
        self.next_seq_num += 1

    if not self.is_waiting_for_ack:
        self.startTimer()

def startTimer(self):
    """Start the timer for the oldest unacknowledged packet."""
    self.is_waiting_for_ack = True
    self.timer = reactor.callLater(TIMEOUT, self.handleTimeout)

def handleTimeout(self):
    """Handle timeout by retransmitting the entire window."""
    print("[Sender] Timeout! Retransmitting window...")
    self.is_waiting_for_ack = False
    self.next_seq_num = self.base
    self.window = []
    self.sendWindow()

def dataReceived(self, data):
    """Handle received ACKs."""
    ack = data.decode()
    ack_num = int(ack.split()[1])
    print(f"[Sender] Received: {ack}")

    if ack_num >= self.base:
        self.base = ack_num + 1
```

```
            self.window = self.window[ack_num - self.base + 1:]
            if self.base == self.next_seq_num:
                self.is_waiting_for_ack = False
                if self.timer and self.timer.active():
                    self.timer.cancel()
            else:
                self.startTimer()
            self.sendWindow()
# Receiver (Server)
class ReceiverProtocol(protocol.Protocol):
    def __init__(self):
        self.expected_seq_num = 0

    def dataReceived(self, data):
        """Handle received packets."""
        message = data.decode()
        seq_num, received_message = message.split("|", 1)  # Extract sequence number and
message

        seq_num = int(seq_num)

        print(f"[Receiver] Received Packet: {seq_num} | Message: {received_message}")

        if seq_num == self.expected_seq_num:
            print(f"[Receiver] Sending ACK {seq_num}")
            self.transport.write(f"ACK {seq_num}".encode())
            self.expected_seq_num += 1
        else:
            print(f"[Receiver] Out-of-order packet. Expected: {self.expected_seq_num}, Received:
{seq_num}")
            self.transport.write(f"ACK {self.expected_seq_num - 1}".encode())  # Send last valid
ACK
# Factories for Sender and Receiver
class SenderFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return SenderProtocol()

class ReceiverFactory(protocol.Factory):
    def buildProtocol(self, addr):
        return ReceiverProtocol()
# Start Sender and Receiver
def startSender():
```

```
    reactor.connectTCP("localhost", 8000, SenderFactory())
def startReceiver():
    reactor.listenTCP(8000, ReceiverFactory())
    reactor.callLater(2, startSender)  # Start sender after receiver is running
if __name__ == "__main__":
    print("Starting Sliding Window Protocol")
    startReceiver()
    reactor.run()
```

OUTPUT:



RESULT:

Thus the Stop and wait protocol and Sliding window protocol is implemented in twisted python.

Ex No.: 11 Date: / / 2025

Implementation of HTTP Web client program to download a web page Using TCP sockets using twisted python

Aim:
To write a program for creating socket for HTTP for web page upload and download

Code:
Server.py

```
import socket

HOST = 'localhost'
PORT = 8080

def handle_upload(connection):
    with open('uploaded.html', 'wb') as file:
        while True:
            data = connection.recv(1024)
            if not data:
                break
            file.write(data)
    connection.sendall(b'Webpage uploaded successfully.')
    connection.close()

def handle_download(connection):
    with open('example.html', 'rb') as file:
        data = file.read()
        connection.sendall(data)
    connection.close()

def start_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((HOST, PORT))
    server_socket.listen(1)
    print(f'Server started. Listening on {HOST}:{PORT}...')

    while True:
        connection, address = server_socket.accept()
        request = connection.recv(1024).decode()

        if request == 'UPLOAD':
```

```
            handle_upload(connection)
        elif request == 'DOWNLOAD':
            handle_download(connection)

        connection.close()

start_server()
```

client.py

```
import socket

HOST = 'localhost'
PORT = 8080

def upload_webpage(filename):
    with open(filename, 'rb') as file:
        data = file.read()

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((HOST, PORT))
        client_socket.sendall(b'UPLOAD')
        client_socket.sendall(data)

        response = client_socket.recv(1024).decode()
        print(response)
        print("Webpage uploaded successfully.")

def download_webpage():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
        client_socket.connect((HOST, PORT))
        client_socket.sendall(b'DOWNLOAD')

        data = client_socket.recv(4096)  # Increased buffer size to receive larger files
        with open('downloaded_example.html', 'wb') as file:
            file.write(data)

        print("Webpage downloaded successfully.")

# Example usage
upload_webpage('example.html')
download_webpage()
```

Output:

```
PS C:\Users\HP\downloads> python ex11_server.py
Server started. Listening on localhost:8080...
```

```
PS C:\Users\HP\downloads>  python ex11_client.py

Webpage uploaded successfully.
Webpage downloaded successfully.
PS C:\Users\HP\downloads>
```

DOWNLOADED WEB PAGE:

Welcome to the Example Webpage!

This is a sample webpage for testing purposes.

Result:

Thus, the HTTP protocol for webpage upload and download has been implemented using socket programming in python

Ex No.: 12 Date: / / 2025

Implementation of Remote Procedure Call (RPC) using Twisted Python

Aim:

To write a program to implement Remote Procedure Call (RPC) for Client-Server Communication using Twisted Python.

Problem Description:

RPC allows a program to call a procedure or function on a remote computer as if it were a local function call. This enables distributed computing and facilitates communication between different processes or systems. It should provide the necessary functionality to enable remote procedure calls, handling the communication between the client and the server.

Code:

Server.py

```
from twisted.spread import pb
from twisted.internet import reactor
class MyService(pb.Root):
 def remote_add(self, x, y):
    print("ADDITION:\n",x+y)
    return x + y
 def remote_subtract(self, x, y):
    print("SUBTRACTION:\n",x-y)
    return x - y
 def remote_multiply(self, x, y):
    print("MULTIPLICATION:\n",x*y)
    return x * y
 def remote_divide(self, x, y):
    if y != 0:
        print("DIVISION:\n",x/y)
        return x / y
    else:
        raise ValueError("Cannot divide by zero.")
service = MyService()
factory = pb.PBServerFactory(service)
reactor.listenTCP(6473, factory)
reactor.run()
```

client.py

```
from twisted.spread import pb
from twisted.internet import reactor
```

```
def add_handle_result(result):
 print("Result of Addition:", result)

def sub_handle_result(result):
 print("Result of subtraction:", result)

def mul_handle_result(result):
 print("Result of multiplication:", result)
def div_handle_result(result):
 print("Result of division:", result)

def connection_error(err):
 print("Connection error:",err)
 reactor.stop()
def connect():
 factory = pb.PBClientFactory()
 reactor.connectTCP("localhost", 6473, factory)
 d = factory.getRootObject()

 d.addCallback(lambda obj: obj.callRemote("add",int(input("Enter number1:")) ,int(input("Enter
number2:"))))

 d.addCallback(add_handle_result)
 d.addCallback(lambda _: factory.getRootObject())
 d.addCallback(lambda obj: obj.callRemote("subtract",int(input("Enter
number1:")),int(input("Enter number2:"))))

 d.addCallback(sub_handle_result)
 d.addCallback(lambda _: factory.getRootObject())
 d.addCallback(lambda obj: obj.callRemote("multiply",int(input("Enter
number1:")),int(input("Enter number2:"))))

 d.addCallback(mul_handle_result)
 d.addCallback(lambda _: factory.getRootObject())
 d.addCallback(lambda obj: obj.callRemote("divide",int(input("Enter
number1:")),int(input("Enter number2:"))))

 d.addCallback(div_handle_result)
reactor.callWhenRunning(connect)
reactor.run()
```

Output:

```
PS C:\Users\HP\downloads> python ex12_server.py
ADDITION:
 14
SUBTRACTION:
 1
MULTIPLICATION:
 25
DIVISION:
 3.0
PS C:\Users\HP\downloads>
```

```
PS C:\Users\HP\downloads> python ex12_client.py
Enter number1:12
Enter number2:2
Result of Addition: 14
Enter number1:3
Enter number2:2
Result of subtraction: 1
Enter number1:5
Enter number2:5
Result of multiplication: 25
Enter number1:6
Enter number2:2
Result of division: 3.0
PS C:\Users\HP\downloads>
```

Result:

Thus, the RPC using Twisted and Perspective Broker in Python has been implemented and tested successfully.

Ex No.: 13 Date: / / 2025

Implementation of Subnetting using twisted python

Aim:

To implement the concept of subnetting and find the hosts using twisted python

Problem Description:

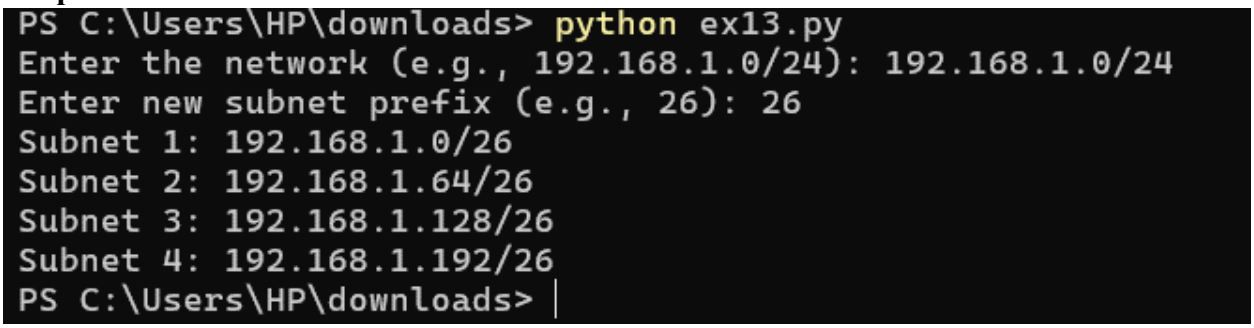
A subnet is a smaller network, also referred to as a sub network. An IP network is logically divided into several smaller network components by subnets. A subnet is used to divide a large network into a number of smaller, linked networks, which helps to minimize traffic. Subnets reduce the need for traffic to use unnecessary routes, which speeds up the network. To help with the lack of IP addresses on the internet, subnets were developed.

Code:

```
from twisted.internet import reactor, defer
import ipaddress

class SubnettingService:
    def __init__(self, network, new_prefix):
        self.network = network
        self.new_prefix = new_prefix

    def calculate_subnets(self):
        try:
            net = ipaddress.ip_network(self.network, strict=False)
            if self.new_prefix <= net.prefixlen:
                raise ValueError("New prefix must be larger than current prefix for subnetting.")
            subnets = list(net.subnets(new_prefix=self.new_prefix))
            return subnets
        except ValueError as e:
            return str(e)

    def get_subnets(self):
        d = defer.Deferred()
        reactor.callLater(1, lambda: d.callback(self.calculate_subnets()))
        return d

class SubnetClient:
    def __init__(self, network, new_prefix):
        self.service = SubnettingService(network, new_prefix)
```

```
    def display_subnets(self, subnets):
        if isinstance(subnets, str):
            print("Error:", subnets)
        else:
            for i, subnet in enumerate(subnets, 1):
                print(f"Subnet {i}: {subnet}")
        reactor.stop()

    def run(self):
        d = self.service.get_subnets()
        d.addCallback(self.display_subnets)
        reactor.run()

if __name__ == "__main__":
    network = input("Enter the network (e.g., 192.168.1.0/24): ")
    new_prefix = int(input("Enter new subnet prefix (e.g., 26): "))
    client = SubnetClient(network, new_prefix)
    client.run()
```

Output:



Result:

Thus, the subnetting has been implemented and tested successfully using Twisted Python.

Exercise 14: Applications using TCP and UDP Sockets like a) DNS b) SNMP c) SMPT using Twisted Python

---

a) DNS Program using UDP Sockets

Aim:

To write a program to implement Domain Name System (DNS) using UDP sockets for resolving the domain name into IP addresses .

Code:

dns_server.py

```
import socket
def dns_server():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind(('localhost', 5353))
    print("DNS Server is running on port 5353...")
    while True:
        data, addr = server_socket.recvfrom(1024)
        domain = data.decode()
        print(f"Received domain request for: {domain}")
        try:
            ip_address = socket.gethostbyname(domain)
            response = f"IP address of {domain} is {ip_address}"
        except socket.gaierror:
            response = f"Domain {domain} not found"
        server_socket.sendto(response.encode(), addr)

if __name__ == "__main__":
    dns_server()
```

dns_client.py

```
import socket
def dns_client():
```

```
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_address = ('localhost', 5353)
    while True:
        domain = input("Enter domain name to resolve (or type 'exit' to quit): ")
        if domain.lower() == 'exit':
            break
        client_socket.sendto(domain.encode(), server_address)
        data, _ = client_socket.recvfrom(1024)
        print("Response from DNS server:", data.decode())
    client_socket.close()


if __name__ == "__main__":
    dns_client()
```

Sample Input and Output:

Server side:

```
pter/../..\debugpy\launcher' '49786' '--' 'c:\Users\SSN\Downloads\dc
Starting server...
('127.0.0.1', 57818) wants to fetch data!
('127.0.0.1', 57818) wants to fetch data!
('127.0.0.1', 57818) wants to fetch data!
('127.0.0.1', 63194) wants to fetch data!
('127.0.0.1', 63194) wants to fetch data!
('127.0.0.1', 63194) wants to fetch data!
```

Client side:

```
pter/../..\debugpy\launcher' '49888' '--' 'c:\Users\SSN\Downloads\dcn_lap\DNS_client.py'
enter domain name for which the ip is needed:www.facebook.com
the ip for the domain name www.facebook.com:NOT Found!
continue? (y/n)y
enter domain name for which the ip is needed:www.google.com
the ip for the domain name www.google.com:192'165'1'1
continue? (y/n)y
enter domain name for which the ip is needed:www.amazon.com
the ip for the domain name www.amazon.com:192.165.1.5
continue? (y/n)n
```

Result:

The DNS program was successfully implemented using UDP sockets in Python and resolved domain names to their IP addresses.

b) SNMP - Simple Network Management Protocol Simulation using UDP

Aim:

To simulate the Simple Network Management Protocol (SNMP) using UDP sockets.

Code:

snmp_server.py

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol
class SNMPProtocol(DatagramProtocol):
    def datagramReceived(self, data, addr):
        print("Received data from {}: {}".format(addr, data.decode()))
        response = "SNMP response: Device is online"
        self.transport.write(response.encode(), addr)
def run_server():
    reactor.listenUDP(161, SNMPProtocol())
    print("SNMP Server is running on port 161...")
    reactor.run()

run_server()
```

snmp_client.py

```
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol
class SNMPClientProtocol(DatagramProtocol):
    def startProtocol(self):
        self.transport.connect('127.0.0.1', 161)
        self.sendRequest()
    def sendRequest(self):
        request = "SNMP request"
        self.transport.write(request.encode())
```

```
    def datagramReceived(self, data, addr):
        print("Received SNMP response from {}: {}".format(addr, data.decode()))
        reactor.stop()
def run_client():
    reactor.listenUDP(0, SNMPClientProtocol())
    reactor.run()

run_client()
```

Sample Input and Output:

Server Output:

```
SNMP Server is running on port 161...
Received data from ('127.0.0.1', 57839): SNMP request
```

Client Output:

```
Received SNMP response from ('127.0.0.1', 161):
SNMP response: Device is online
```

Result:

Thus, the Simple Network Management Protocol (SNMP) using Twisted Python was implemented and tested successfully.

c) SMTP - Sending Email using Python SMTP Library

Aim:

To send an email using the **SMTP protocol** in Python using the smtp lib and email libraries.

Code:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart


def send_email(sender_email, sender_password, recipient_email, subject, message):
    smtp_server = "smtp.gmail.com"
    smtp_port = 587
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = recipient_email
    msg['Subject'] = subject
    msg.attach(MIMEText(message, 'plain'))


    try:
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(sender_email, sender_password)
        server.sendmail(sender_email, recipient_email, msg.as_string())
        server.quit()
        print('Email sent successfully!')


    except smtplib.SMTPException as e:
        print('Error sending email:', str(e))


sender_email = 'jai2110682@ssn.edu.in'
sender_password = '22Jan2022@ssn'  # Be cautious storing passwords in code!
recipient_email = 'harsh2110303@ssn.edu.in'
```

subject = 'Successful Implementation of SMTP Protocol'

message = 'SMTP protocol was successfully implemented Harsh and Jai'

send_email(sender_email, sender_password, recipient_email, subject, message)

Sample Input and Output:

```
Email sent successfully!
```

Result:

The SMTP protocol was successfully implemented using Python and the email was sent from the sender to the recipient.

Exercise 15: Study of Network Stimulator (NS) and Stimulation of Congestion Control Algorithm using NS

---

a) Study of Network Stimulator (NS-2)

Aim:

To study the use of Network Simulator (NS-2).

Code:

```
# Create a network topology
set ns [new Simulator]
# Enable trace file generation
set tracefile [open "congestion_control.tr" w]
$ns trace-all $tracefile
# Create nodes
set node(0) [$ns node]
set node(1) [$ns node]
# Create a duplex link
$ns duplex-link $node(0) $node(1) 10Mb 10ms DropTail
# Setup TCP agent and attach to node(0)
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $node(0) $tcp
# Setup TCP Sink agent and attach to node(1)
set sink [new Agent/TCPSink]
$ns attach-agent $node(1) $sink
# Connect TCP to Sink
$ns connect $tcp $sink
# Setup FTP traffic
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ns at 0.1 "$ftp start"
```

```
# Apply congestion control algorithm
$tcp set cong_algorithm NewReno
# Visuals
$ns duplex-link-op $node(0) $node(1) orient right-down
$ns color 1 Blue
$ns color 2 Red
# End simulation
$ns at 10.0 "$ns finish"
$ns run
# Trace and NAM setup
$ns flush-trace
close $tracefile
# Launch NAM
set namfile "congestion_control.nam"
set nam [open $namfile w]
$ns namtrace-all $nam
$ns nam-end-wireless $nam
# Exit
$ns halt
$ns delete
```

Sample Input and Output:

A Trace file was generated and a NAM window was opened simulating the network congestion control.

Result:

Thus, the Network Simulator (NS-2) is studied in detail.

b) SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS-2

Aim:

To simulate network congestion control algorithm using network simulator (NS-2).

Code:

```
set ns [new Simulator]

set tracefile [open congestion_control.tr w]
$ns trace-all $tracefile
set namfile [open congestion_control.nam w]
$ns namtrace-all $namfile

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.5Mb 20ms DropTail ;# Bottleneck link

$ns queue-limit $n2 $n3 10

$ns duplex-link-op $n2 $n3 queue-type RED
$ns duplex-link-op $n2 $n3 queue-pos 0.5
$ns duplex-link-op $n2 $n3 queue-limit 10
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
```

```
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set type_ FTP
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $n3 $sink1
$ns connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

$ns at 0.5 "$ftp0 start"
$ns at 1.0 "$ftp1 start"
$ns at 4.0 "$ftp0 stop"
$ns at 4.0 "$ftp1 stop"
$ns at 5.0 "finish"

proc finish {} {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam congestion_control.nam &
    exit 0
}
$ns run
```

Sample Input and Output:

Output NAM file was generated and simulation of network control was observed in NAM window.

```
merudhula@Merudhula-PC:/mnt/c/Users/Admin/OneDrive/Desktop/NS2$ ns congestion.tcl
node 0 received ping answer from  4 with round-trip-time 506.0 ms.
node 4 received ping answer from  0 with round-trip-time 506.0 ms.
node 0 received ping answer from  4 with round-trip-time 506.0 ms.
node 4 received ping answer from  0 with round-trip-time 506.0 ms.
nam:
ns: finish: couldn't execute "xgraph": no such file or directory
    while executing
"exec xgraph congestion.xg -geometry 300x300 &"
    (procedure "finish" line 3)
    invoked from within
"finish"
merudhula@Merudhula-PC:/mnt/c/Users/Admin/OneDrive/Desktop/NS2$
```

Trace file:

+ 0.2 0 2 ping 64 ------- 0 0.1 4.2 -1 0 – 0.2 0 2 ping 64 ------- 0 0.1 4.2 -1 0 r 0.210256 0 2 ping 64 ------- 0 0.1 4.2 -1 0 + 0.210256 2 3 ping 64 ------- 0 0.1 4.2 -1 0

- 0.210256 2 3 ping 64 ------- 0 0.1 4.2 -1 0

+ 0.3 4 3 ping 64 ------- 0 4.2 0.1 -1 1 - 0.3 4 3 ping 64 ------- 0 4.2 0.1 -1 1 r 0.341024 4 3 ping 64 ------- 0 4.2 0.1 -1 1 + 0.341024 3 2 ping 64 ------- 0 4.2 0.1 -1 1 - 0.341024 3 2 ping 64 ------- 0 4.2 0.1 -1 1 r 0.411963 2 3 ping 64 ------- 0 0.1 4.2 -1 0 + 0.411963 3 4 ping 64 ------- 0 0.1 4.2 -1 0 - 0.411963 3 4 ping 64 ------- 0 0.1 4.2 -1 0 r 0.452987 3 4 ping 64 ------- 0 0.1 4.2 -1 0 + 0.452987 4 3 ping 64 ------- 0 4.2 0.1 -1 2 - 0.452987 4 3 ping 64 ------- 0 4.2 0.1 -1 2 r 0.494011 4 3 ping 64 ------- 0 4.2 0.1 -1 2 + 0.494011 3 2 ping 64 ------- 0 4.2 0.1 -1 2

- 0.494011 3 2 ping 64 ------- 0 4.2 0.1 -1 2

+ 0.5 0 2 tcp 40 ------- 1 0.0 4.0 0 3 - 0.5 0 2 tcp 40 ------- 1 0.0 4.0 0 3 r 0.51016 0 2 tcp 40 ------- 1 0.0 4.0 0 3 + 0.51016 2 3 tcp 40 ------- 1 0.0 4.0 0 3 - 0.51016 2 3 tcp 40 ------- 1 0.0 4.0 0 3 r 0.542731 3 2 ping 64 ------- 0 4.2 0.1 -1 1 + 0.542731 2 0 ping 64 ------- 0 4.2 0.1 -1 1 - 0.542731 2 0 ping 64 ------- 0 4.2 0.1 -1 1 r 0.552987 2 0 ping 64 ------- 0 4.2 0.1 -1 1 + 0.552987 0 2 ping 64 ------- 0 0.1 4.2 -1 4 - 0.552987 0 2 ping 64 ------- 0 0.1 4.2 -1 4 r 0.563243 0 2 ping 64 ------- 0 0.1 4.2 -1 4 + 0.563243 2 3 ping 64 ------- 0 0.1 4.2 -1 4

- 0.563243 2 3 ping 64 ------- 0 0.1 4.2 -1 4

+ 0.6 1 2 tcp 40 ------- 2 1.0 5.0 0 5 - 0.6 1 2 tcp 40 ------- 2 1.0 5.0 0 5 r 0.61016 1 2 tcp 40 ------- 2 1.0 5.0 0 5 + 0.61016 2 3 tcp 40 ------- 2 1.0 5.0 0 5 - 0.61016 2 3 tcp 40 ------- 2 1.0 5.0 0 5 r 0.695717 3 2 ping 64 ------- 0 4.2 0.1 -1 2 + 0.695717 2 0 ping 64 ------- 0 4.2 0.1 -1 2 …..

Result:

Thus, the Simple Network Management Protocol (SNMP) using Twisted Python was implemented and tested successfully.

Exercise 16: Implementation of the routing algorithms

a) Link State routing b) Flooding c) Distance vector using Twisted Python

---

a) Link State routing

Aim:

To implement the Link State Routing algorithm using Twisted Python, where each router maintains a complete topology of the network and computes the shortest path to all other routers using Dijkstra's algorithm.

Code:

```
from twisted.internet import reactor, protocol
import pickle
import heapq
class Router(protocol.Protocol):
    def __init__(self, factory):
        self.factory = factory
        self.name = factory.name
        self.routing_table = {}  # To store the topology
        self.neighbors = {}  # To store neighbors and costs
    def connectionMade(self):
        print(f"Router {self.name} started.")
        n = int(input(f"Enter how many neighbors Router {self.name} has: "))
        for i in range(n):
            p = input("Enter Neighbor Name: ")
            print(f"Enter {p}, Cost: ")
            cost = int(input())
            self.neighbors[p] = cost
            print(f"Connected to {p}")
        self.factory.topology[self.name] = self.neighbors
        self.transport.write(pickle.dumps(self.factory.topology))
        u = input("Compute shortest distance? (1 for yes, 0 for no): ")
        if u == '1':
            distances = self.compute_shortest_paths()
```

```
            print(f"Shortest distances from {self.name}: {distances}")
    def dataReceived(self, data):
        topology = pickle.loads(data)
        self.factory.topology.update(topology)
        print(f"Router {self.name} received topology update: {self.factory.topology}")
    def compute_shortest_paths(self):
        distances = {node: float('infinity') for node in self.factory.topology}
        distances[self.name] = 0
        pq = [(0, self.name)]
        visited = set()
        while pq:
            current_distance, current_node = heapq.heappop(pq)
            if current_node in visited:
                continue
            visited.add(current_node)
            for neighbor, cost in self.factory.topology[current_node].items():
                distance = current_distance + cost
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    heapq.heappush(pq, (distance, neighbor))
        return distances

class RouterFactory(protocol.ClientFactory):
    def __init__(self, name):
        self.name = name
        self.topology = {}  # Shared topology across all routers
    def buildProtocol(self, addr):
        return Router(self)
    def routerConnectionFailed(self, connector, reason):
        print("Connection failed.")
        reactor.stop()
```

```
    def routerConnectionLost(self, connector, reason):
        print("Connection lost.")
        reactor.stop()
if __name__ == "__main__":
    name = input("Enter name of your node: ")
    factory = RouterFactory(name)
    reactor.connectTCP("localhost", 8000, factory)
    reactor.run()
```

Sample Input and Output:

Input:

Enter name of your node: A

Enter how many neighbors Router A has: 2

Enter Neighbor Name: B

Enter B, Cost: 2

Enter Neighbor Name: C

Enter C, Cost: 4

Compute shortest distance? (1 for yes, 0 for no): 1

Output:

```
Router A started.
Connected to B
Connected to C
Router A received topology update: {'A': {'B': 2, 'C': 4}}
Shortest distances from A: {'A': 0, 'B': 2, 'C': 4}
```

Result:

The Link State Routing algorithm was successfully implemented. Router A built its topology by collecting neighbor information, shared it with other routers, and computed the shortest paths to all other nodes using Dijkstra's algorithm.

b) Flooding

Aim:

To implement the Flooding routing algorithm using Twisted Python, where a message sent by one node is broadcast to all other nodes in the network.

Code:

Server.py

```
from twisted.internet import reactor, protocol

class FloodingProtocol(protocol.Protocol):
    def connectionMade(self):
        print("New client connected:", self.transport.getPeer())
        self.factory.clients.append(self)
        self.transport.write(b"Welcome to the server!\n")
    def dataReceived(self, data):
        print("Received data from client:", data.decode())
        self.floodClients(data)
    def connectionLost(self, reason):
        print("Client disconnected:", self.transport.getPeer())
        self.factory.clients.remove(self)
    def floodClients(self, data):
        for client in self.factory.clients:
            if client != self:  # Don't send back to the sender
                client.transport.write(data)


class FloodingFactory(protocol.Factory):
    def __init__(self):
        self.clients = []
    def buildProtocol(self, addr):
        return FloodingProtocol()


if __name__ == '__main__':
    port = 8000
    print("Starting TCP server on port", port)
```

```
    factory = FloodingFactory()
    reactor.listenTCP(port, factory)
    reactor.run()
```

client.py

```
from twisted.internet import reactor, protocol
class FloodingClient(protocol.Protocol):
    def connectionMade(self):
        print("Connected to server. You can start sending messages.")
        self.sendData()
    def dataReceived(self, data):
        print("Received data from server:", data.decode())
    def connectionLost(self, reason):
        print("Connection lost:", reason)
        reactor.stop()
    def sendData(self):
        message = input("Enter a message to send (or 'quit' to exit): ")
        if message.lower() == 'quit':
            self.transport.loseConnection()
        else:
            self.transport.write(message.encode())
            reactor.callLater(0, self.sendData)

class FloodingClientFactory(protocol.ClientFactory):
    def buildProtocol(self, addr):
        return FloodingClient()
if __name__ == '__main__':
    host = 'localhost'
    port = 8000
    print("Starting TCP client...")
    reactor.connectTCP(host, port, FloodingClientFactory())
    reactor.run()
```

Sample Input and Output:

Server Output:

```
Starting TCP server on port 8000
New client connected: <Peer: 127.0.0.1:12345>
Received data from client: Hello, everyone!
```

Client 1 Input and Output

```
Starting TCP client...
Connected to server. You can start sending messages.
Enter a message to send (or 'quit' to exit): Hello, everyone!
Enter a message to send (or 'quit' to exit): quit
Connection lost: [ConnectionDone]
```

Client 2 Output:

```
Starting TCP client...
Connected to server. You can start sending messages.
Received data from server: Hello, everyone!
Enter a message to send (or 'quit' to exit): quit
Connection lost: [ConnectionDone]
```

Result:

The Flooding algorithm was successfully implemented. The server broadcasts messages from one client to all other connected clients, ensuring that every node in the network receives the message.

c) Distance Vector Routing

Aim:

To implement the Distance Vector Routing algorithm using Twisted Python, where each router maintains a routing table and periodically updates it based on information received from neighbors using the Bellman-Ford algorithm.

Code:

```
from twisted.internet.protocol import DatagramProtocol
from twisted.internet import reactor


class DistanceVectorRoutingProtocol(DatagramProtocol):
    def __init__(self, host, port):
        self.host = host
        self.port = port
        self.routing_table = {host: (host, 0)}  # (next_hop, cost)
        self.neighbors = {}  # Neighbor: cost

    def startProtocol(self):
        self.transport.joinGroup("224.0.0.0")
        print(f"Started routing protocol on {self.host}:{self.port}")
        if self.port == 8000:
            self.neighbors = {"B": 2, "C": 4, "D": 7}
            for neighbor, cost in self.neighbors.items():
                self.routing_table[neighbor] = (neighbor, cost)
            self.broadcastRoutingTable()

    def sendRoutingUpdate(self, destination, cost):
        routing_update = f"{self.host},{destination},{cost}"
        self.transport.write(routing_update.encode(), ("224.0.0.0", self.port))

    def datagramReceived(self, datagram, address):
        routing_update = datagram.decode()
```

```
        source, destination, cost = routing_update.split(",")
        cost = int(cost)
        print(f"Received routing update from {address}: {routing_update}")
        self.updateRoutingTable(source, destination, cost)

    def updateRoutingTable(self, source, destination, cost):
        if source in self.neighbors:
            cost_to_source = self.neighbors[source]
            total_cost = cost_to_source + cost
            if destination not in self.routing_table or total_cost < self.routing_table[destination][1]:
                self.routing_table[destination] = (source, total_cost)
                print(f"Updated routing table: {self.routing_table}")
                self.broadcastRoutingTable()

    def calculateShortestPaths(self):
        print("Routing table after convergence:")
        print("Destination   Next Hop   Cost")
        for dest, (next_hop, cost) in self.routing_table.items():
            print(f"{dest}\t\t{next_hop}\t\t{cost}")

    def broadcastRoutingTable(self):
        print(f"Broadcasting routing table from {self.host}:")
        for dest, (next_hop, cost) in self.routing_table.items():
            self.sendRoutingUpdate(dest, cost)

if __name__ == "__main__":
    host = "A"
    port = 8000
    protocol = DistanceVectorRoutingProtocol(host, port)
    reactor.listenMulticast(port, protocol, listenMultiple=True)
    reactor.callLater(5, protocol.calculateShortestPaths)
    reactor.run()
```

Sample Input and Output:

Output (Router A)

```
Started routing protocol on A:8000
Broadcasting routing table from A:
Routing table after convergence:
Destination   Next Hop   Cost
A             A          0
B             B          2
C             C          4
D             D          7
```

Output (Router B):

```
Started routing protocol on B:8001
Received routing update from ('127.0.0.1', 8000): A,B,2
Updated routing table: {'B': ('B', 0), 'A': ('A', 2)}
Broadcasting routing table from B:
```

Result:

The Distance Vector Routing algorithm was successfully implemented. Router A initialized its routing table, shared updates with neighbors, and updated its table based on received information, converging to the correct shortest paths.

Exercise 17: Implementation of the OSPF routing algorithm

Aim:

To implement the OSPF (Open Shortest Path First) routing algorithm

Code:

server.py

```
from twisted.internet.protocol import DatagramProtocol
from twisted.internet import reactor, task
import json
import heapq
class OSPFRouter(DatagramProtocol):
    def __init__(self, router_id):
        self.router_id = router_id
        self.link_state_db = {self.router_id: {}}  # Link-state database: {router: {neighbor: cost}}
        self.routing_table = {self.router_id: {'cost': 0, 'next_hop': self.router_id}}  # Shortest path table
        self.neighbors = {}  # Discovered neighbors: {neighbor_id: address}
        self.hello_interval = 5
        self.sequence_number = 0  # For LSA freshness
    def startProtocol(self):
        self.transport.joinGroup("224.0.0.1")
        print(f"Router {self.router_id} started on port 9999")
        n = int(input(f"Enter the number of neighbors for Router {self.router_id}: "))
        for _ in range(n):
            neighbor_id = input("Enter Neighbor ID: ")
            cost = int(input(f"Enter cost to {neighbor_id}: "))
            self.link_state_db[self.router_id][neighbor_id] = cost
        self.send_hello_message()
        self.hello_loop = task.LoopingCall(self.send_hello_message)
        self.hello_loop.start(self.hello_interval)
    def send_hello_message(self):
        hello_message = {'type': 'HELLO', 'router_id': self.router_id}
```

```
        self.transport.write(json.dumps(hello_message).encode(), ("224.0.0.1", 9999))
    def datagramReceived(self, datagram, address):
        message = json.loads(datagram.decode())
        if message['type'] == 'HELLO':
            self.handle_hello_message(message, address)
        elif message['type'] == 'LSA':
            self.handle_lsa_message(message)
        elif message['type'] == 'REQUEST_ROUTING_TABLE':
            self.handle_request_message(address)
    def handle_hello_message(self, message, address):
        neighbor_id = message['router_id']
        if neighbor_id not in self.neighbors:
            print(f"Router {self.router_id} discovered neighbor {neighbor_id}")
            self.neighbors[neighbor_id] = address
            if neighbor_id not in self.link_state_db:
                self.link_state_db[neighbor_id] = {}
            self.send_lsa_message()
    def send_lsa_message(self):
        self.sequence_number += 1
        lsa_message = {
            'type': 'LSA',
            'router_id': self.router_id,
            'sequence_number': self.sequence_number,
            'link_state_db': self.link_state_db
        }
        self.transport.write(json.dumps(lsa_message).encode(), ("224.0.0.1", 9999))
    def handle_lsa_message(self, message):
        router_id = message['router_id']
        sequence_number = message['sequence_number']
        new_lsa = message['link_state_db']
        if router_id not in self.link_state_db or self.sequence_number < sequence_number:
            self.link_state_db.update(new_lsa)
```

```
        print(f"Router {self.router_id} updated LSDB: {self.link_state_db}")
        self.compute_shortest_paths()
        self.send_lsa_message()
    def compute_shortest_paths(self):
        distances = {router: float('infinity') for router in self.link_state_db}
        distances[self.router_id] = 0
        previous = {router: None for router in self.link_state_db}
        pq = [(0, self.router_id)]
        visited = set()
        while pq:
            current_distance, current_router = heapq.heappop(pq)
            if current_router in visited:
                continue
            visited.add(current_router)
            for neighbor, cost in self.link_state_db[current_router].items():
                distance = current_distance + cost
                if distance < distances[neighbor]:
                    distances[neighbor] = distance
                    previous[neighbor] = current_router
                    heapq.heappush(pq, (distance, neighbor))
        self.routing_table = {}
        for router in self.link_state_db:
            if router == self.router_id:
                self.routing_table[router] = {'cost': 0, 'next_hop': self.router_id}
                continue
            cost = distances[router]
            if cost == float('infinity'):
                continue
            current = router
            while previous[current] != self.router_id:
                current = previous[current]
                if current is None:
```

```
                break
            if current is not None:
                self.routing_table[router] = {'cost': cost, 'next_hop': current}
        print(f"Router {self.router_id} updated routing table: {self.routing_table}")
    def handle_request_message(self, address):
        response = {
            'type': 'LSA',
            'router_id': self.router_id,
            'routing_table': self.routing_table
        }
        self.transport.write(json.dumps(response).encode(), address)
    def stopProtocol(self):
        self.hello_loop.stop()

if __name__ == '__main__':
    router_id = input("Enter Router ID: ")
    protocol = OSPFRouter(router_id)
    reactor.listenMulticast(9999, protocol, listenMultiple=True)
    reactor.run()
```

client.py

```
from twisted.internet.protocol import DatagramProtocol
from twisted.internet import reactor
import json
class OSPFClient(DatagramProtocol):
    def startProtocol(self):
        self.transport.joinGroup("224.0.0.1")
        print("OSPF Client started. Requesting routing tables...")
        self.send_request()
    def send_request(self):
        request_message = {'type': 'REQUEST_ROUTING_TABLE'}
        self.transport.write(json.dumps(request_message).encode(), ("224.0.0.1", 9999))
```

```
        reactor.callLater(10, self.send_request)
    def datagramReceived(self, datagram, address):
        message = json.loads(datagram.decode())
        if message['type'] == 'LSA':
            self.display_routing_table(message['router_id'], message['routing_table'])
    def display_routing_table(self, router_id, routing_table):
        print(f"\nRouter {router_id} Routing Table:")
        print("Destination\tCost\tNext Hop")
        for dest, info in routing_table.items():
            print(f"{dest}\t\t{info['cost']}\t{info['next_hop']}")

if __name__ == '__main__':
    protocol = OSPFClient()
    reactor.listenMulticast(9999, protocol, listenMultiple=True)
    reactor.run()
```

Sample Input and Output:

Server (Router A):

Enter Router ID: A

Enter the number of neighbors for Router A: 2

Enter Neighbor ID: B

Enter cost to B: 2

Enter Neighbor ID: C

Enter cost to C: 4

Router A started on port 9999

Router A discovered neighbor B

Router A updated LSDB: {'A': {'B': 2, 'C': 4}, 'B': {'A': 2, 'D': 3}}

Router A updated routing table: {'A': {'cost': 0, 'next_hop': 'A'}, 'B': {'cost': 2, 'next_hop': 'B'}, 'C': {'cost': 4, 'next_hop': 'C'}, 'D': {'cost': 5, 'next_hop': 'B'}}

Server (Router B):

Enter Router ID: B

Enter the number of neighbors for Router B: 2

Enter Neighbor ID: A

Enter cost to A: 2

Enter Neighbor ID: D

Enter cost to D: 3

Router B started on port 9999

Router B discovered neighbor A

Router B updated LSDB: {'B': {'A': 2, 'D': 3}, 'A': {'B': 2, 'C': 4}}

Router B updated routing table: {'B': {'cost': 0, 'next_hop': 'B'}, 'A': {'cost': 2, 'next_hop': 'A'}, 'D': {'cost': 3, 'next_hop': 'D'}, 'C': {'cost': 6, 'next_hop': 'A'}}

Client Output:

```
OSPF Client started. Requesting routing tables...

Router A Routing Table:
Destination Cost    Next Hop
A       0   A
B       2   B
C       4   C
D       5   B

Router B Routing Table:
Destination Cost    Next Hop
B       0   B
A       2   A
D       3   D
C       6   A
```

Result:

The OSPF routing algorithm was successfully implemented using Twisted Python. The routers (A and B) exchanged Hello messages to discover neighbors, shared their link-state databases via LSAs, and computed the shortest paths using Dijkstra's algorithm. The client successfully requested and displayed the routing tables from both routers, showing the correct shortest paths to all destinations in the network.