

INDEX

S.NO	DATE	NAME OF THE PROGRAM	PG.NO	MARKS	SIGN
1.		Stop and Wait Protocol	1		
2.		Sliding window Protocol	5		
3.		Study of socket programming with client server model	9		
4.		Simulating ARP	13		
5.		Simulating RARP	17		
6.		Simulating PING command	21		
7.		Simulating TRACEROUTE	25		
8.		Application using TCP Sockets Creation for Echo Client and Echo Server	28		
9.		Application using TCP Sockets Creation for Chat	32		
10.		Application using TCP Sockets Creation for File Transfer	36		
11.		Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS	40		

IMPLEMENTATION OF STOP AND WAIT PROTOCOL

EXP: 1

DATE:

AIM:

To write a python program to perform stop and wait protocol

ALGORITHM:

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.

PROGRAM:**CLIENT:**

```
import socket
s=socket.socket()
s.listen(5)
c,addr=s.accept()
while True:
    i=input("Enter a data: ")
    c.send(i.encode())
    ack=c.recv(1024).decode()
    if ack:
        print(ack)
        continue
    else:
        c.close()
        break
```

SERVER:

```
import socket
```

```
s=socket.socket()
```

```
s.connect(('localhost',8000))
```

```
while True:
```

```
    print(s.recv(1024).decode())
```

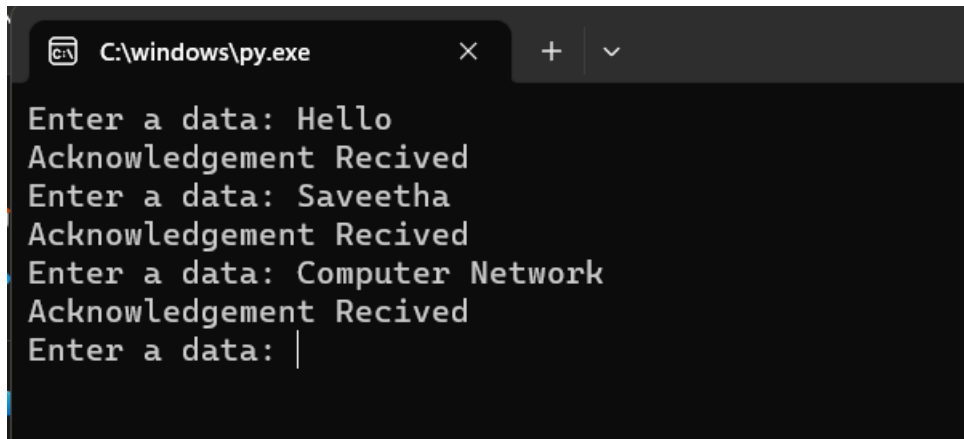
```
    s.send("Acknowledgement Recived".encode())
```

6. Stop the program

```
s.bind(('localhost',8000))
```

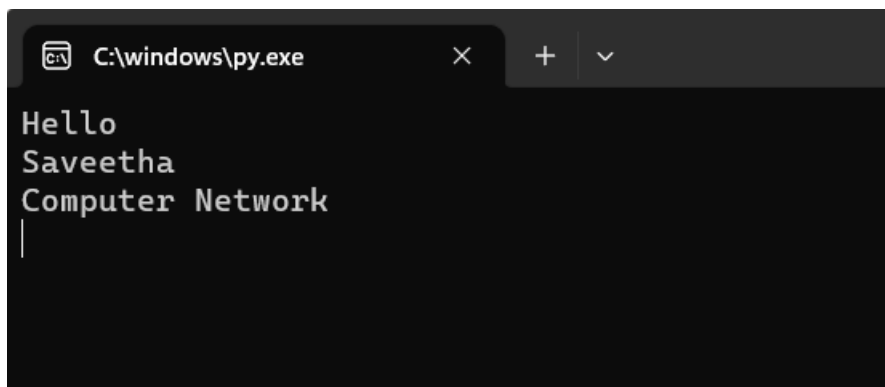
OUTPUT:

CLIENT:



```
C:\windows\py.exe
Enter a data: Hello
Acknowledgement Recived
Enter a data: Saveetha
Acknowledgement Recived
Enter a data: Computer Network
Acknowledgement Recived
Enter a data: |
```

SERVER:



```
C:\windows\py.exe
Hello
Saveetha
Computer Network
|
```

RESULT:

Thus, python program to perform stop and wait protocol was successfully executed.

IMPLEMENTATION OF SLIDING WINDOW PROTOCOL

EXP : 2

DATE:

AIM:

To write a python program to perform sliding window protocol

ALGORITHM:

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

PROGRAM:**CLIENT:**

```
import socket

s=socket.socket()

s.bind(('localhost',8000))

s.listen(5)

c,addr=s.accept()

size=int(input("Enter number of frames to send : "))

l=list(range(size))

s=int(input("Enter Window Size : "))

st=0

i=0

while True:

    while(i<len(l)):

        st+=s

        c.send(str(l[i:st]).encode())

        ack=c.recv(1024).decode()

        if ack:

            print(ack)

            i+=s
```

SERVER:

```
import socket  
s=socket.socket()  
s.connect(('localhost',8000))  
while True:  
    print(s.recv(1024).decode())  
    s.send("acknowledgement recived from the server".encode())
```


OUTPUT:

CLIENT:

```
Command Prompt - py 1B_Client.py

C:\Users\Computer Networks>py 1B_Client.py
Enter number of frames to send : 10
Enter Window Size : 3
acknowledgement recived from the server
acknowledgement recived from the server
acknowledgement recived from the server
acknowledgement recived from the server
```

SERVER:

```
Command Prompt - py 1B_Server.py

C:\Users\Computer Networks>py 1B_Server.py
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]
[9]
```

RESULT:

Thus, python program to perform stop and wait protocol was successfully executed.

STUDY OF SOCKET PROGRAMMING AND CLIENT – SERVER MODE

EXP : 3

DATE:

AIM:

To implement socket programming date and time display from client to server using TCP Sockets

ALGORITHM:

Server:

1. Create a server socket and bind it to port.
2. Listen for new connection and when a connection arrives, accept it.
3. Send server"s date and time to the client.
4. Read client"s IP address sent by the client.
5. Display the client details.
6. Repeat steps 2-5 until the server is terminated.
7. Close all streams.
8. Close the server socket.
9. Stop.

Client:

1. Create a client socket and connect it to the server"s port number.
2. Retrieve its own IP address using built-in function.
3. Send its address to the server.
4. Display the date & time sent by the server.
5. Close the input and output streams.
6. Close the client socket.
7. Stop.

PROGRAM:**CLIENT:**

```
import socket

from datetime import datetime

s=socket.socket()

s.bind(('localhost',8000))

s.listen(5)

c,addr=s.accept()

print("Client Address : ",addr)

now = datetime.now()

c.send(now.strftime("%d/%m/%Y %H:%M:%S").encode())

ack=c.recv(1024).decode()

if ack:

    print(ack)

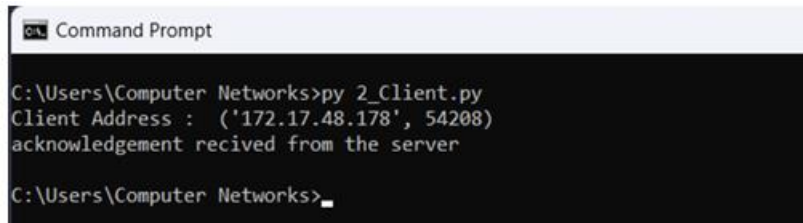
c.close()
```

SERVER:

```
import socket  
s=socket.socket()  
s.connect(('localhost',8000))  
print(s.getsockname())  
print(s.recv(1024).decode())  
s.send("acknowledgement recived from the server".encode())
```

OUTPUT:

CLIENT:

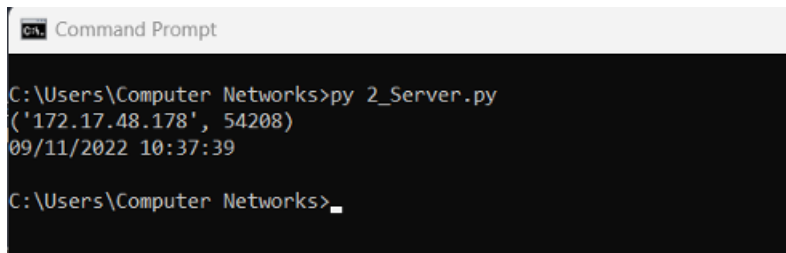


```
Command Prompt

C:\Users\Computer Networks>py 2_Client.py
Client Address : ('172.17.48.178', 54208)
acknowledgement recived from the server

C:\Users\Computer Networks>
```

SERVER:



```
Command Prompt

C:\Users\Computer Networks>py 2_Server.py
('172.17.48.178', 54208)
09/11/2022 10:37:39

C:\Users\Computer Networks>
```

RESULT:

Thus, the program to implement socket programming date and time display from client to server using TCP Sockets was successfully executed.

WRITE A CODE SIMULATING ARP /RARP PROTOCOLS

EXP : 4

DATE:

AIM:

To write a python program for simulating ARP protocols using TCP.

ALGORITHM:

Server:

1. Start the program
2. Accept the socket which is created by the client.
3. Server maintains the table in which IP and corresponding MAC addresses are stored.
4. Read the IP address which is send by the client.
5. Map the IP address with its MAC address and return the MAC address to client.

Client:

1. Start the program
2. Using socket connection is established between client and server.
3. Get the IP address to be converted into MAC address.
4. Send this IP address to server.
5. Server returns the MAC address to client.

PROGRAM:**CLIENT:**

```
import socket

s=socket.socket()

s.bind(('localhost',8000))

s.listen(5)

c,addr=s.accept()

address={"165.165.80.80":"6A:08:AA:C2","165.165.79.1":"8A:BC:E3:FA"};

while True:

    ip=c.recv(1024).decode()

    try:

        c.send(address[ip].encode())

    except KeyError:

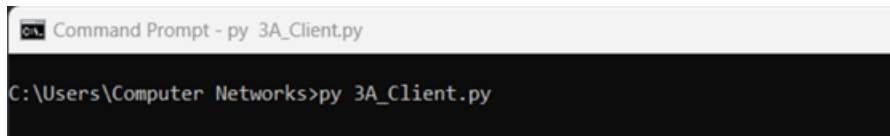
        c.send("Not Found".encode())
```

SERVER:

```
import socket  
s=socket.socket()  
s.connect(('localhost',8000))  
while True:  
ip=input("Enter logical Address : ")  
s.send(ip.encode())  
print("MAC Address",s.recv(1024).decode())
```

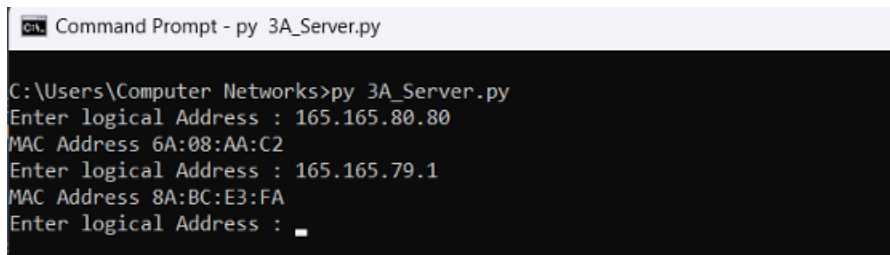

OUTPUT:

CLIENT:



```
Command Prompt - py 3A_Client.py
C:\Users\Computer Networks>py 3A_Client.py
```

SERVER:



```
Command Prompt - py 3A_Server.py
C:\Users\Computer Networks>py 3A_Server.py
Enter logical Address : 165.165.80.80
MAC Address 6A:08:AA:C2
Enter logical Address : 165.165.79.1
MAC Address 8A:BC:E3:FA
Enter logical Address : 
```

RESULT:

Thus, the python program for simulating ARP protocols using TCP was successfully executed.

PROGRAM FOR REVERSE ADDRESS RESOLUTION PROTOCOL (RARP) USING UDP

EXP : 5

DATE:

AIM:

To write a python program for simulating RARP protocols using UDP

ALGORITHM:

Server:

1. Start the program.
2. Server maintains the table in which IP and corresponding MAC addresses are stored.
3. Read the MAC address which is send by the client.
4. Map the IP address with its MAC address and return the IP address to client.

Client:

1. Start the program
2. Using datagram sockets UDP function is established.
3. Get the MAC address to be converted into IP address.
4. Send this MAC address to server.
5. Server returns the IP address to client.

PROGRAM:**CLIENT:**

```
import socket

s=socket.socket()

s.bind(('localhost',9000))

s.listen(5)

c,addr=s.accept()

address={"6A:08:AA:C2":"192.168.1.100","8A:BC:E3:FA":"192.168.1.99"};

while True:

    ip=c.recv(1024).decode()

    try:

        c.send(address[ip].encode())

    except KeyError:

        c.send("Not Found".encode())
```

SERVER:

```
import socket  
s=socket.socket()  
s.connect(('localhost',9000))  
while True:  
    ip=input("Enter MAC Address : ")  
    s.send(ip.encode())  
    print("Logical Address",s.recv(1024).decode())
```

OUTPUT:

CLIENT:

```
Command Prompt - py 3B_Client.py  
C:\Users\Computer Networks>py 3B_Client.py
```

SERVER:

```
Command Prompt - py 3B_Server.py  
C:\Users\Computer Networks>py 3B_Server.py  
Enter MAC Address : 6A:08:AA:C2  
Logical Address 192.168.1.100  
Enter MAC Address : 8A:BC:E3:FA  
Logical Address 192.168.1.99  
Enter MAC Address : 9A:B7:C3:BC  
Logical Address Not Found  
Enter MAC Address : _
```

RESULT:

Thus, python program for simulating RARP protocols using UDP was successfully executed.

WRITE A CODE SIMULATING PING COMMAND

EXP : 6

DATE:

AIM:

To write the python program for simulating ping command.

ALGORITHM:

Step 1: start the program.

Step 2: Include necessary package in java.

Step 3: To create a process object p to implement the ping command.

Step 4: declare one Buffered Reader stream class object.

Step 5: Get the details of the server

5:1: length of the IP address.

5:2: time required to get the details.

5:3: send packets, receive packets and lost packets.

5.4: minimum, maximum and average times.

Step 6: print the results.

Step 7: Stop the program.

PROGRAM:**CLIENT:**

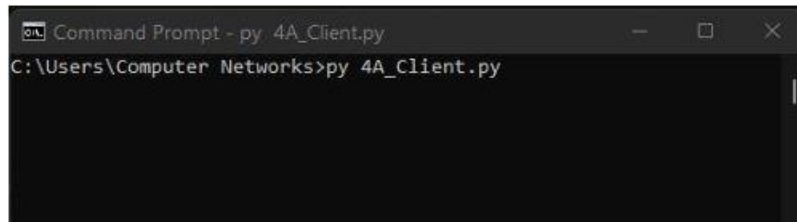
```
import socket
from pythonping import ping
s=socket.socket()
s.bind(('localhost'8000))
s.listen(5)
c,addr=s.accept()
while True:
    hostname=c.recv(1024).decode()
    try:
        c.send(str(ping(hostname, verbose=False)).encode())
    except KeyError:
        c.send("Not Found".encode())
```

SERVER:

```
import socket  
s=socket.socket()  
s.connect(('localhost',8000))  
while True:  
    ip=input("Enter the website you want to ping ")  
    s.send(ip.encode())  
    print(s.recv(1024).decode())
```


OUTPUT:

CLIENT:



```
Command Prompt - py 4A_Client.py
C:\Users\Computer Networks>py 4A_Client.py
```

SERVER:



```
Command Prompt - py 4A_Server.py
C:\Users\Computer Networks>py 4A_Server.py
Enter the website you want to ping www.google.com
Reply from 142.250.77.100, 29 bytes in 27.3ms
Reply from 142.250.77.100, 29 bytes in 31.54ms
Reply from 142.250.77.100, 29 bytes in 26.36ms
Reply from 142.250.77.100, 29 bytes in 79.56ms

Round Trip Times min/avg/max is 26.36/41.19/79.56 ms
Enter the website you want to ping
```

RESULT:

Thus, the python program for simulating ping command was successfully executed

WRITE A CODE SIMULATING TRACEROUTE COMMAND

EXP : 7

DATE:

AIM:

To write the python program for simulating Traceroute command

ALGORITHM:

1. Start the program.
2. Get the frame size from the user.
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server, it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

PROGRAM:

```
from scapy.all import*  
target = ["www.google.com"]  
result, unans = traceroute(target,maxttl=32)  
print(result,unans)
```

OUTPUT:

```
Command Prompt

C:\Users\Computer Networks>tracert www.google.com

Tracing route to www.google.com [2404:6800:4009:800::2004]
over a maximum of 30 hops:

  1    5 ms    3 ms    3 ms  2401:4900:232e:c6f4:b42b:5aff:fe8f:ba7b
  2    *      *      *      Request timed out.
  3   38 ms   29 ms   20 ms  2401:4900:c4:1::109
  4   56 ms   24 ms   29 ms  2401:4900:c0:1::4fd
  5   43 ms   28 ms   38 ms  2404:a800:3a00:1::45
  6   56 ms   25 ms   34 ms  2404:a800::92
  7   52 ms   *       73 ms  2001:4860:1:1::d2e
  8   36 ms   22 ms   37 ms  2404:6800:810b::1
  9   29 ms   26 ms   31 ms  2001:4860:0:1::5662
 10   49 ms   20 ms   24 ms  2001:4860:0:e00::3
 11   76 ms   54 ms   53 ms  2001:4860::9:4001:7733
 12    *      *      *      Request timed out.
 13   87 ms   43 ms   49 ms  2001:4860:0:1::19c7
 14   68 ms   50 ms   50 ms  bom07s15-in-x04.1e100.net [2404:6800:4009:800::2004]

Trace complete.

C:\Users\Computer Networks>
```

RESULT:

Thus, the python program for simulating Traceroute command was successfully executed.

CREATION FOR ECHO CLIENT AND ECHO SERVER USING TCP SOCKETS

EXP: 8

DATE:

AIM:

To write a python program for creating Echo Client and Echo Server using TCP Sockets Links.

ALGORITHM:

1. Import the necessary modules in python
2. Create a socket connection to using the socket module.
3. Send message to the client and receive the message from the client using the Socket module in server .
4. Send and receive the message using the send function in socket.

PROGRAM:**CLIENT:**

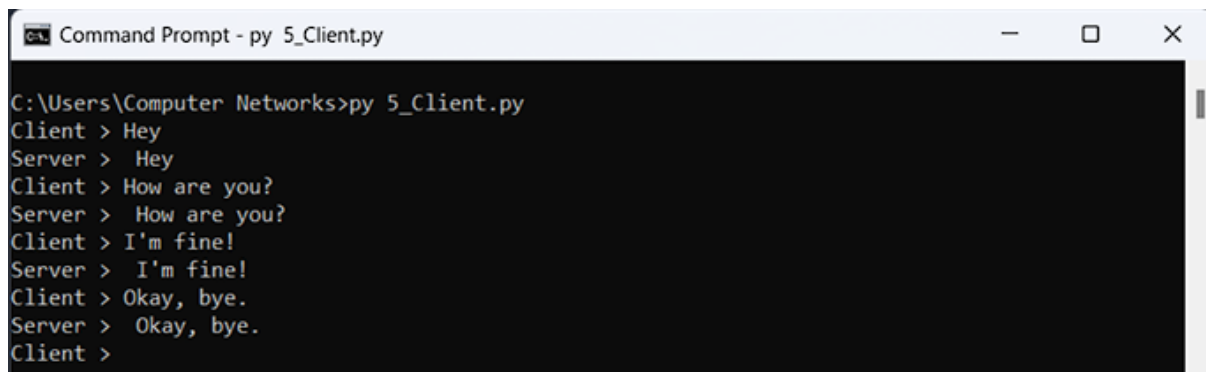
```
import socket  
s=socket.socket()  
s.connect(('localhost',8000))  
while True:  
    msg=input("Client > ")  
    s.send(msg.encode())  
    print("Server > ",s.recv(1024).decode())
```

SERVER:

```
import socket
s=socket.socket()
s.bind(('localhost',8000))
s.listen(5)
c,addr=s.accept()
while True:
    ClientMessage=c.recv(1024).decode()
    c.send(ClientMessage.encode())
```

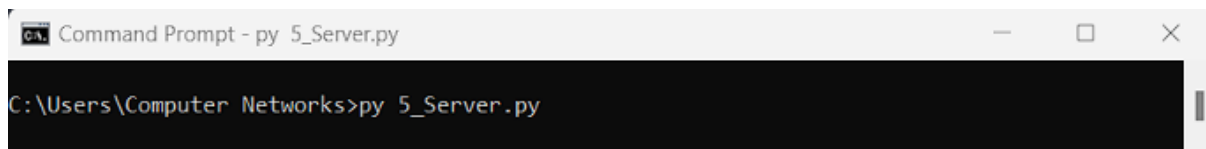
OUTPUT:

CLIENT:



```
Command Prompt - py 5_Client.py
C:\Users\Computer Networks>py 5_Client.py
Client > Hey
Server > Hey
Client > How are you?
Server > How are you?
Client > I'm fine!
Server > I'm fine!
Client > Okay, bye.
Server > Okay, bye.
Client >
```

SERVER:



```
Command Prompt - py 5_Server.py
C:\Users\Computer Networks>py 5_Server.py
```

RESULT:

Thus, the python program for creating Echo Client and Echo Server using TCP Sockets Links was successfully created and executed.

CREATION FOR CHAT USING TCP SOCKETS

EXP: 9

DATE:

AIM:

To write a python program for creating Chat using TCP Sockets Links.

ALGORITHM:

- 1.Import the necessary modules in python
2. Create a socket connection to using the socket module.
3. Send message to the client and receive the message from the client using the Socket module in server .
4. Send and receive the message using the send function in socket.

PROGRAM:**CLIENT:**

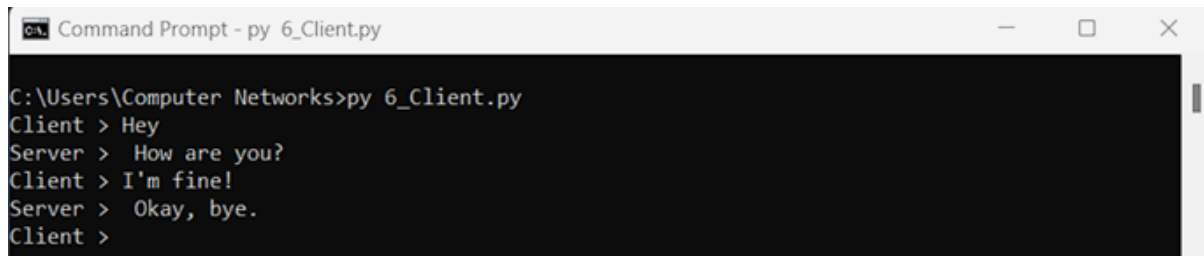
```
import socket
s=socket.socket()
s.connect(('localhost',8000))
while True:
    msg=input("Client > ")
    s.send(msg.encode())
    print("Server > ",s.recv(1024).decode())
```

SERVER:

```
import socket
s=socket.socket()
s.bind(('localhost',8000))
s.listen(5)
c,addr=s.accept()
while True:
    ClientMessage=c.recv(1024).decode()
    print("Client > ",ClientMessage)
    msg=input("Server > ")
    c.send(msg.encode())
```

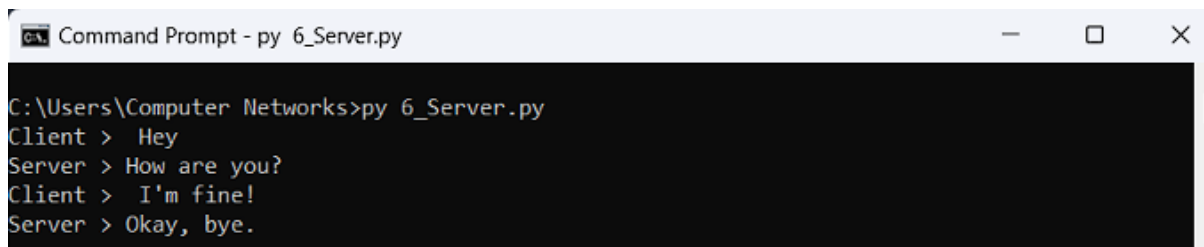
OUTPUT:

CLIENT:



```
Command Prompt - py 6_Client.py
C:\Users\Computer Networks>py 6_Client.py
Client > Hey
Server > How are you?
Client > I'm fine!
Server > Okay, bye.
Client >
```

SERVER:



```
Command Prompt - py 6_Server.py
C:\Users\Computer Networks>py 6_Server.py
Client > Hey
Server > How are you?
Client > I'm fine!
Server > Okay, bye.
```

RESULT:

Thus, the python program for creating Chat using TCP Sockets Links was successfully created and executed.

CREATION FOR FILE TRANSFER USING TCP SOCKETS

EXP: 10

DATE:

AIM:

To write a python program for creating File Transfer using TCP Sockets Links.

ALGORITHM:

1. Import the necessary python modules.
2. Create a socket connection using socket module.
3. Send the message to write into the file to the client file.
4. Open the file and then send it to the client in byte format.
5. In the client side receive the file from server and then write the content into it.

PROGRAM:**CLIENT:**

```
import socket

s = socket.socket()

host = socket.gethostname()

port = 60000

s.connect((host, port))

s.send("Hello server!".encode())

with open('received_file', 'wb') as f:
    while True:
        print('receiving data...')
        data = s.recv(1024)
        print('data=%s', (data))
        if not data:
            break
        f.write(data)

f.close()

print('Successfully get the file')

s.close()

print('connection closed')
```

SERVER:

```
import socket

port = 60000

s = socket.socket()

host = socket.gethostname()

s.bind((host, port))

s.listen(5)

while True:

    conn, addr = s.accept()

    data = conn.recv(1024)

    print('Server received', repr(data))

    filename='mytext.txt'

    f = open(filename,'rb')

    l = f.read(1024)

    while (l):

        conn.send(l)

        print('Sent ',repr(l))

        l = f.read(1024)

    f.close()

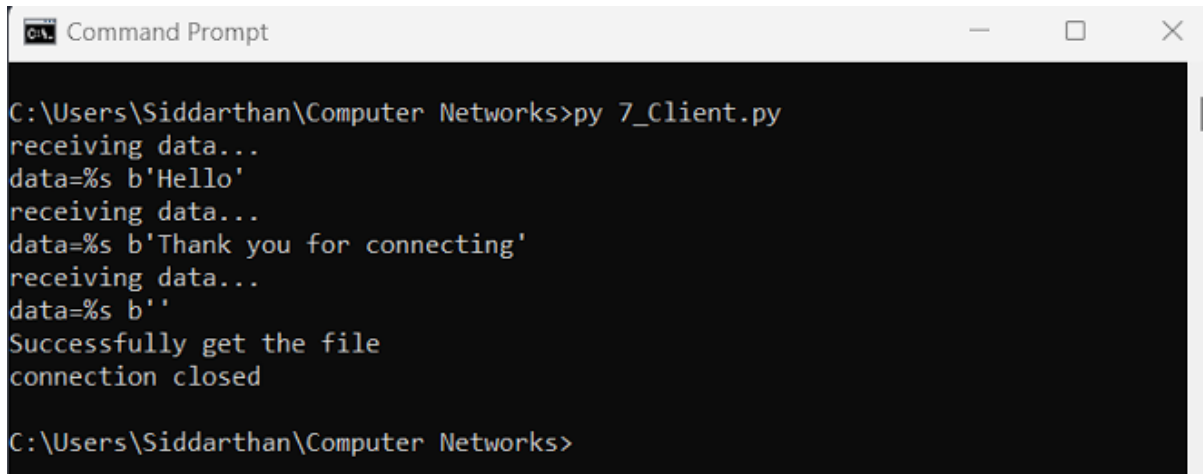
    print('Done sending')

    conn.send('Thank you for connecting'.encode())

    conn.close()
```

OUTPUT:

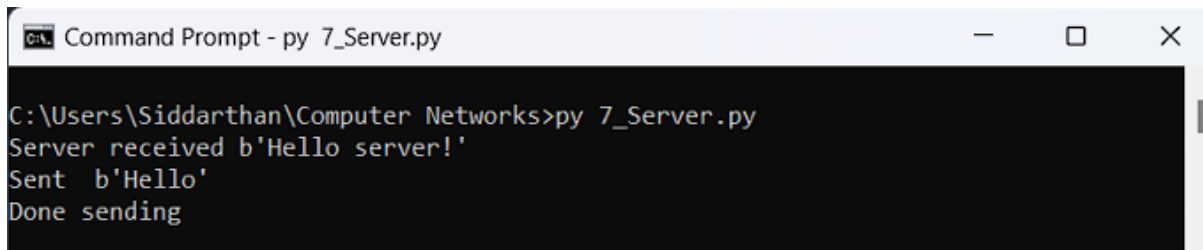
CLIENT:



```
C:\Users\Siddarthan\Computer Networks>py 7_Client.py
receiving data...
data=%s b'Hello'
receiving data...
data=%s b'Thank you for connecting'
receiving data...
data=%s b''
Successfully get the file
connection closed

C:\Users\Siddarthan\Computer Networks>
```

SERVER:



```
C:\Users\Siddarthan\Computer Networks>py 7_Server.py
Server received b'Hello server!'
Sent b'Hello'
Done sending
```

RESULT:

Thus, the python program for creating File Transfer using TCP Sockets Links was successfully created and executed.

STUDY OF NETWORK SIMULATOR (NS) AND SIMULATION OF CONGESTION CONTROL ALGORITHMS USING NS

EXP: 11

DATE:

AIM:

To Study of Network simulator (NS).and Simulation of Congestion Control Algorithms using NS.

NET WORK SIMULATOR (NS2)

Ns overview

1. Ns programming: A Quick start
2. Case study I: A simple Wireless network
3. Case study II: Create a new agent in Ns

Ns overview

1. Ns Status
2. Periodical release (ns-2.26, Feb 2003)
3. Platform support
4. FreeBSD, Linux, Solaris, Windows and Mac

Ns functionalities

- Routing, Transportation, Traffic sources, Queuing disciplines, QoS

- Wireless
- Ad hoc routing, mobile IP, sensor-MAC
- Tracing, visualization and various utilities
- NS(Network Simulators)

Most of the commercial simulators are GUI driven, while some network simulators are CLI driven. The network model / configuration describes the state of the network (nodes, routers, switches, links) and the events (data transmissions, packet error etc.). An important output of simulations are the trace files. Trace files log every packet, every event that occurred in the simulation and are used for analysis. Network simulators can also provide other tools to facilitate visual analysis of trends and potential trouble spots. Most network simulators use discrete event simulation, in which a list of pending "events" is stored, and those events are processed in order, with some events triggering future events— such as the event of the arrival of a packet at one node triggering the event of the arrival of that packet at a downstream node. Simulation of networks is a very complex task. For example, if congestion is high, then estimation of the average occupancy is challenging because of high variance. To estimate the likelihood of a buffer overflow in a network, the time required for an accurate answer can be extremely large. Specialized techniques such as "control variates" and "importance sampling" have been developed to speed simulation.

Examples of network simulators

There are many both free/open-source and proprietary network simulators. Examples of notable

network simulation software are, ordered after how often they are mentioned in research papers:

1. NS (open source)
2. OPNET (proprietary software)
3. NetSim (proprietary software)

Uses of network simulators

Network simulators serve a variety of needs. Compared to the cost and time involved in setting up an entire test bed containing multiple networked computers, routers and data links, network simulators are relatively fast and inexpensive. They allow engineers, researchers to test scenarios that might be particularly difficult or expensive to emulate using real hardware - for instance, simulating a scenario with several nodes or experimenting with a new protocol in the network. Network simulators are particularly useful in allowing researchers to test new networking protocols or changes to existing protocols in a controlled and reproducible environment. A typical network simulator encompasses a wide range of networking technologies and can help the users to build complex networks from basic building blocks such as a variety of nodes and links. With the help of simulators, one can design hierarchical networks using various types of nodes like computers, hubs, bridges, routers, switches, links, mobile units etc. Various types of Wide Area Network (WAN) technologies like TCP, ATM, IP etc. and Local Area Network (LAN) technologies like Ethernet, token rings etc., can all be simulated with a typical simulator and the user can test, analyze various standard results apart from devising some novel protocol or strategy for routing etc. Network simulators are also widely used to simulate battlefield networks in Network-centric warfare. There are a wide variety of network simulators, ranging from the very simple to the very complex. Minimally, a network simulator must enable a user to represent a network topology, specifying the nodes on the network, the links between those nodes and the traffic between the nodes. More complicated systems may allow the user to specify everything about the protocols used to handle traffic in a network. Graphical applications allow users to easily visualize the workings of their simulated environment. Text-based applications may provide a less intuitive interface, but may permit more advanced forms of customization.

Packet loss

Packet loss occurs when one or more packets of data travelling across a computer network fail to reach their destination. Packet loss is distinguished as one of the three main error types encountered in digital communications; the

other two being bit error and spurious packets caused due to noise. Packets can be lost in a network because they may be dropped when a queue in the network node overflows. The amount of packet loss during the steady state is another important property of a congestion control scheme. The larger the value of packet loss, the more difficult it is for transport layer protocols to maintain high bandwidths, the sensitivity to loss of individual packets, as well as to frequency and patterns of loss among longer packet sequences is strongly dependent on the application itself.

Throughput

This is the main performance measure characteristic, and most widely used. In communication networks, such as Ethernet or packet radio, throughput or network throughput is the average rate of successful message delivery over a communication channel. The throughput is usually measured in bits per second (bit/s or bps), and sometimes in data packets per second or data packets per time slot. This measure how soon the receiver is able to get a certain amount of data send by the sender. It is determined as the ratio of the total data received to the end to end delay. Throughput is an important factor which directly impacts the network performance.

Delay

Delay is the time elapsed while a packet travels from one point e.g., source premise or network ingress to destination premise or network degrees. The larger the value of delay, the more difficult it is for transport layer protocols to maintain high band widths. We will calculate end to end delay.

Queue Length

A queuing system in networks can be described as packets arriving for service, waiting for service if it is not immediate, and if having waited for service, leaving the system after being served. Thus queue length is very important

characteristic to determine that how well the active queue management of the congestion control algorithm has been working.