Practical 12.

a) Implement echo client server using TCP/UDP sockets

AIM:
To implement echo client server using TCP/UDP sockets.

Algorithm:

~~import socket~~
~~import time~~
~~def ping-server (host ='127.0.0.1', port =12345):~~
~~with socket.socket (socket.AF_INET,~~
~~socket.sock-GRAM) as s~~

```
import socket
import threading
def handle_client (client_socket, client_address):
        Point (f"[+] New connection from {client_address}")
        while True:
            try:
                msg = client_socket.recv (1024).decode()
                if not msg :
                        break
                Print (f"[Client {client_address }] {msg}")
                client_socket.sendall (f"server received:
                                        {msg}".encode())
            except ConnectionResetError:
                    break
        Point (f"[-] Connection closed {client_address}")
        client_socket.close()
def start_server (host = "127.0.0.1", Port=5000):
        server_socket = socket.socket (Socket.AF_INET,
                                    socket.STOCK-STREAM)
        server_socket.bind ((host, port))
server_socket.listen(5)
Print(f"[SERVER] Listening on{host}:{port}...")

while True:
    client_socket, client_address = server_socket.
                                          accept()
    client_thread = threading.Thread(
              target=handle_client, args=(client_
                                  socket, client_addr)
    )
    client_thread.start()
```

client code

```
    def start_client(server_host ="127.0.01", server_port
                                              =5000):
        Client_socket = socket.socket (socket.AF_INET,
                          socket.SOCK_STREAM)
        client_socket.connect((server_host, server_port))
        Print(f"[CLIENT] Connected to server {server_host}:
                               {server_port}")

        try:
            while True:
                msg = input ("Enter message (or 'quit'
                              to exit): ")
                if msg.lower() == "quit":
                    break
                client_socket.sendall(msg.encode())
                response = client_socket.recv(1024).
                                          decode()
                Print (f"[SERVER RESPONSE]
                                 {response}")
        finally:
            client_socket.close()
            Print ("[CLIENT] Disconnected")
if __name__ == "__main__":
    import sys
```

```
if len(sys.argv) > 1 and sys.argv[1]=="server":
        start_server()
    else:
        start_client()
```

Sample Input and output

Step 1: Run the server

```
$ python chat_program.py server
```

server output:

```
[SERER] Listening on 127.0.0.1:5000...
[+] New connection from ('127.0.0.1', 60628)
[Client ('127.0.0:1', 60628)] Hello server!
[Client ('127.0.0.1', 60628)] How are you?
[-] Connection closed ('127.0.0.1', 60628)
```

Step 2: Run the client

```
$ Python chat_program.py
```

Client Interaction:

```
[CLIENT] Connected to server 127.0.0.1:5000
Enter message (or 'quit' to exit) : Hello server!
[SERVER RESPONSE] Server received : Hello
                                    server!
Enter message (or 'quit' to exit): How are you?
[SERVER RESPONSE] Server received : How are
                                    you?
Enter message (or 'quit' to exit): quit
[CLIENT] Disconnected
```

13/x/25 10 10

RESULT:

The Echo client-server and chat program were successfully implemented using TCP sockets.

The client could send messages to the server and the server echoed the same messages back, confirming reliable end to end com at TC

End to End Communication at Transport Layer (UDP)

Aim:

To implement a UDP-based Echo (ping) client-server program using socket programming that measures Round Trip Time (RTT) for each packet and demonstrates end to end communication at the Transport layer.

Program Code:

```
import socket
import time
import sys
def udp_ping_server (host = "127.0.0.1", port=12000):
        server_socket = socket.socket (socket.AF_INET,
                                       socket.SOCK_DGRAM)
        server_socket.bind ((host, port))
        Print (f"[SERVER] Listening on {host}:{port}")
        while True:
              msg, client_address = server_socket.
                                          recvfrom(1024)
              Print (f"[SERVER] Received '{msg.decode()}'
                          from {client_address}")
              server_socket.sendto (msg, client_address)
```

Client

```
def udp_ping_client (server_host = "127.0.0.1", server_port
                                        = 12000, count=5):
      client_socket = socket.socket (socket.AF_INET,
                                     socket.SOCK_DGRAM)
      client_socket.settimeout(1)
      for  i  in range (1, count +1):
            msg = f"Ping {i} {time.time()}"
            start = time.time()
            client_socket.sendto (msg.encode(),
                          (server_host, server_port))
```

```
try:
    data, _ = client_socket.recvfrom(1024)
    end = time.time()
    rtt = (end - start) * 1000
    print (f"Reply from {server_host}:{server_port} |
              {data.decode()} | RTT={rtt:.2f}ms")

  except socket.timeout:
      print (f"Request {i} timed out")
client_socket.close()
if _name_ == "_main_":
    if len (sys.argv) > 1 and sys.argv[1] == "server":
        udp_ping_server()
    else:
        udp_ping_client()
```

Sample Input and Output

Step 1: Run the Server

```
$ python udp_ping.py server
```

Server Output:

```
[SERVER] Listening on 127.0.0.1:12000
[SERVER] Received 'ping 1  1728575342.123' from
                                    ('127.0.0.1', 60642)
[SERVER] Received 'Ping 2  1728575343.125' from
                                    ('127.0.0.1', 60642)
[SERVER] Received 'Ping 3  1728575344.127' from
                                    ('127.0.0.1', 60642)
```

Step 2: Run the Client

```
$ Python udp_ping.py
```

Client Output:

```
Reply from 127.0.0.1:12000 | Ping 1 1728575342.
                                          123 |
                                    RTT=0.52ms
```

13/8/25

$\frac{10}{10}$

[illegible]

Result:

UDP echo (Ping) client-server program was successfully implemented