Experiment 12 b): 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

```
# ---------------------------------------------------------

# Title: UDP Ping (Echo) Client-Server Program

# ---------------------------------------------------------

# Name:

# Roll No:

# Date:

# ---------------------------------------------------------


import socket

import time

import sys


# ------------------------

# UDP Ping Server

# ------------------------

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}")

        # Echo the same message back to client
```

```
        server_socket.sendto(msg, client_address)



# ------------------------
# UDP Ping 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)  # 1 second timeout

    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()



# ------------------------
# Run as server or client
# ------------------------
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 = 0.52 ms

Reply from 127.0.0.1:12000 | Ping 2 1728575343.125 | RTT = 0.47 ms

Reply from 127.0.0.1:12000 | Ping 3 1728575344.127 | RTT = 0.49 ms

Reply from 127.0.0.1:12000 | Ping 4 1728575345.129 | RTT = 0.54 ms

Reply from 127.0.0.1:12000 | Ping 5 1728575346.131 | RTT = 0.50 ms

(If the server doesn't reply in time, it shows "Request timed out.")

RESULT:

The **UDP Echo (Ping) Client-Server** program was successfully implemented.
The client sent multiple ping messages to the server, and the server echoed them back.
Round Trip Time (RTT) was measured for each message, demonstrating **unreliable but connectionless communication** at the **Transport Layer (UDP)**.