Lab 2

Socket Programming UDP – Five-Layer Model

Name, Surname	Mnqobi Jeza
Student Number:	230878369
Date:	20 February 2025

Objective: Students will learn how to implement a simple UDP-based client-server communication system in Python. They will explore how data is sent and received over a network using the UDP protocol.

Background: UDP (User Datagram Protocol) is a connectionless transport-layer protocol that allows applications to send messages without establishing a connection. Unlike TCP, UDP does not guarantee reliability, ordering, or error correction.

Reference: Section 2.7.1 in *Computer Networking: A Top-Down Approach, Seventh Edition* by Kurose and Ross.

Requirements:

1. Python 3 installed
2. Basic understanding of socket programming
3. Two machines or two terminals on the same machine

Task 1: Implementing the UDP Server

The server should:

- Create a UDP socket
- Bind it to a specific port
- Continuously listen for messages from clients Send a response back to the client

udp_server.py udp_client.py

udp_server.py > ...

```python
import socket

HOST = "127.0.0.1"  # Localhost
PORT = 12345        # Port number

# Create UDP socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Bind to address
server_socket.bind((HOST, PORT))

print(f"UDP Server listening on {HOST}:{PORT}...")

while True:
    # Receive data from client
    data, client_address = server_socket.recvfrom(1024)
    print(f"Received from {client_address}: {data.decode()}")

    # Send a response back
    response = "Message received!"
    server_socket.sendto(response.encode(), client_address)
```

The client should:

- Create a UDP socket
- Send a message to the server
- Receive a response from the server
- Print the response

udp_server.py udp_client.py

udp_client.py > ...

```python
import socket

SERVER_HOST = "127.0.0.1"
SERVER_PORT = 12345

# Create UDP socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

# Message to send
message = "Hello, UDP Server. This is Client Server!"
client_socket.sendto(message.encode(), (SERVER_HOST, SERVER_PORT))

# Receive response
response, _ = client_socket.recvfrom(1024)
print(f"Server response: {response.decode()}")

# Close socket
client_socket.close()
```

Task 3: Running the Experiment

1. Run the server first:
   - Open a terminal and execute python udp_server.py
   - The server will start listening for messages
2. Run the client:
   - Open another terminal and execute python udp_client.py
   - Observe the messages sent and received
3. Modify the client to send multiple messages and analyze the response.
4. Test sending data from different machines on the same network by changing SERVER_IP.

Questions

1. What happens if the client sends a message when the server is not running?
   - ➢ the client will fail to establish a connection and return an error.
2. How does UDP differ from TCP regarding reliability and order of messages?

- TCP ensures reliable, ordered delivery using acknowledgments and retransmissions, making it ideal for web browsing and file transfers. UDP is faster but unreliable, allowing packet loss and out-of-order delivery, making it better for real-time applications like streaming and gaming.

3. Modify the server to send a timestamp along with the response.

```python
from socket import *

serverName = '192.168.1.100'  # Replace with actual server IP
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_DGRAM)

message = input("Input lowercase sentence: ")
clientSocket.sendto(message.encode(), (serverName, serverPort))

modifiedMessage, _ = clientSocket.recvfrom(2048)
print("From Server:", modifiedMessage.decode())

clientSocket.close()
```

4. Implement error handling in both client and server.

```python
except OSError as e:
    print(f"Server error: {e}")

except KeyboardInterrupt:
    print("\nServer shutting down gracefully...")
```

END