**Title**:  
*Implementing Reliable (TCP) and Efficient (UDP) Client-Server Communication in Python.*

**Objective Statement**:  
To design and implement client-server communication using both TCP and UDP protocols in Python. This project aims to provide hands-on experience with connection-oriented and connectionless communication, highlighting their strengths, weaknesses, and practical applications.

**Problem Statement**:  
Develop two client-server applications:

* **TCP**: The server listens for client connections, receives messages, and echoes them back reliably.
* **UDP**: The server receives datagrams from clients and responds by echoing the data without a formal connection.  
  Both implementations must handle real-world challenges like error management, data encoding/decoding, and connection or address tracking.

**Code:**

TCP:

**tcp\_server:**

import socket

def tcp\_server():

host = '127.0.0.1' # localhost

port = 12345 # Port to listen on

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as server\_socket:

server\_socket.bind((host, port))

server\_socket.listen(5)

print(f"TCP Server listening on {host}:{port}...")

conn, addr = server\_socket.accept()

with conn:

print(f"Connected by {addr}")

while True:

data = conn.recv(1024)

if not data:

break

print(f"Received from client: {data.decode()}")

conn.sendall(data) # Echo back the received data

tcp\_server()

**tcp\_client.py**:

import socket

def tcp\_client():

host = '127.0.0.1' # Server's address

port = 12345 # Server's port

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as client\_socket:

client\_socket.connect((host, port))

print("Connected to the TCP server...")

while True:

message = input("Enter message to send (or 'exit' to quit): ")

if message.lower() == 'exit':

break

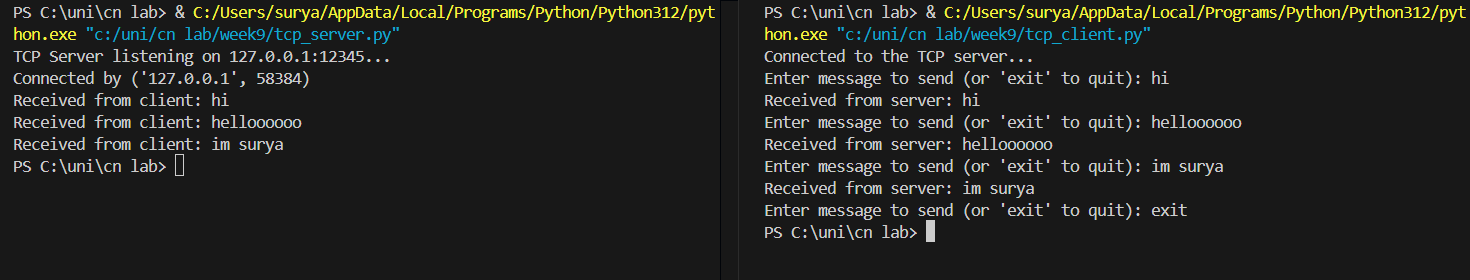
client\_socket.sendall(message.encode())

data = client\_socket.recv(1024)

print(f"Received from server: {data.decode()}")

tcp\_client()

Output:



UDP:

Udp\_server:

import socket

def udp\_server():

host = '127.0.0.1' # localhost

port = 12345 # Port to listen on

with socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) as server\_socket:

server\_socket.bind((host, port))

print(f"UDP Server listening on {host}:{port}...")

while True:

data, addr = server\_socket.recvfrom(1024)

print(f"Received from {addr}: {data.decode()}")

server\_socket.sendto(data, addr) # Echo back the received data

udp\_server()

Udp\_client:

import socket

def udp\_client():

host = '127.0.0.1' # Server's address

port = 12345 # Server's port

with socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM) as client\_socket:

print("Connected to the UDP server...")

while True:

message = input("Enter message to send (or 'exit' to quit): ")

if message.lower() == 'exit':

break

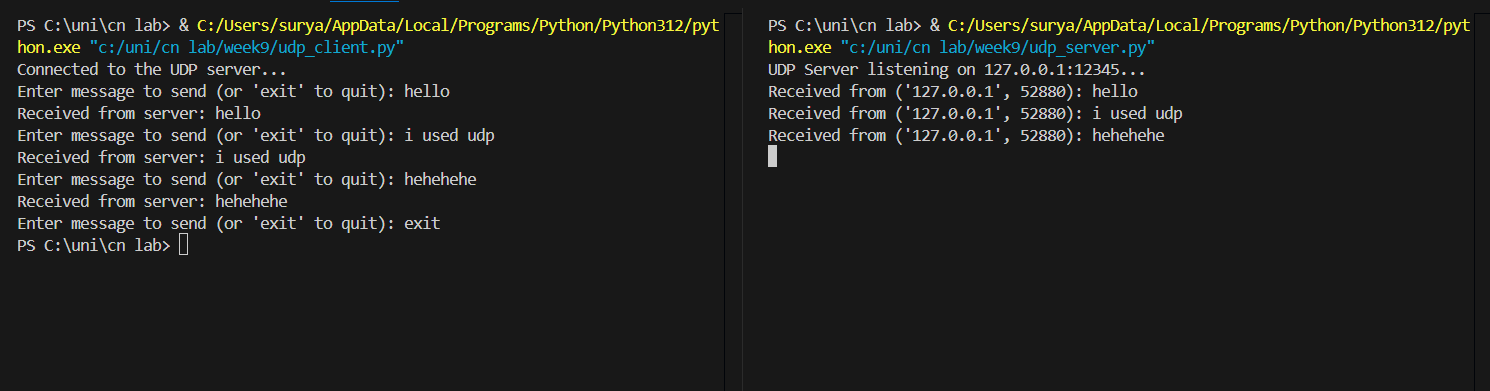
client\_socket.sendto(message.encode(), (host, port))

data, \_ = client\_socket.recvfrom(1024)

print(f"Received from server: {data.decode()}")

udp\_client()

Output:



**Problems Faced During Development and Implementation**:

* **TCP**:
  + Difficulty in handling multiple clients due to single-threaded implementation.
  + Managing connection termination properly to avoid resource leaks.
  + Debugging encoding issues during message transmission.
* **UDP**:
  + Handling packet loss and out-of-order delivery due to its connectionless nature.
  + Manually managing client addresses for responses.
  + Limited feedback for debugging when packets were lost in transit.

**Conclusion**:  
This project provided a comprehensive understanding of TCP and UDP protocols:

* **TCP**: Reinforced the importance of reliable, connection-oriented communication in scenarios like file transfer and web servers.
* **UDP**: Highlighted its lightweight and fast nature for real-time applications like video streaming and gaming.  
  I developed skills in network programming, error handling, and debugging, gaining insights into the trade-offs between reliability and performance in communication protocols.