Scocket Programming Assignment

# UDP PING AND HEARTBEAT IMPLEMENTATION


Abhay Dagar
2022014

Aditya Sharma
2022038

# INTRODUCTION

## Overview

- **Purpose of the Project:** Introduce the basics of network programming.
- **Key Concepts:** Handling unreliable connections and simulating real-world network conditions.

## Lab Focus

- Socket Programming: Learn to send and receive datagram packets using UDP sockets in Python.
- Packet Loss Simulation:  Compute statistics of Packet Loss.

## Project Details

Ping Protocol: Allows a client to send data to a remote server and receive it back unchanged, helping determine round-trip times.

Server Behavior: Listens for incoming UDP packets and randomly decides whether to echo the data back or simulate packet loss artificially/explicitly .

# PART-01

# Implementing Client-Server pinger

# UDP PING SYSTEM - SERVER IMPLEMENTATION

## Listen for Incoming UDP Packets:

The server continuously listens for incoming UDP packets from clients.

## Simulate 30% Packet Loss:

To mimic real-world network conditions, the server randomly drops 30% of the incoming packets.

## Respond with Capitalized Message:

For the packets that are not dropped, the server capitalizes the received message and sends it back to the client.

```python
import random
from socket import *  # Socket library

serverSocket = socket(AF_INET, SOCK_DGRAM)  # Create a UDP socket for the server
serverSocket.bind(('127.0.0.1', 12000))  # Set IP Add and port No of server to listen
print("Started UDP Server IP Address: 127.0.0.1 and Port: 12000")  # Print string on screen
while True:  # Run program forever
    rand = random.randint(1, 10)  # Probability 100%
    message, address = serverSocket.recvfrom(1024)
    message = message.upper()  # Convert client message to uppercase letter
    if rand < 4:  # < 30% without reply message
        continue
    serverSocket.sendto(message, address)  # Send the uppercase message back to the client
```

# UDP PING SYSTEM - CLIENT IMPLEMENTATION

## Send 10 Ping Messages

The client sends a total of 10 ping messages to the server.

## Calculate RTT

For each successful ping (i.e., when a response is received), the client calculates the round-trip time (RTT).

## Handle Timeouts

If no response is received within one second, the client assumes the packet was lost and handles the timeout accordingly.

```python
import socket
import time
while True:
    start_key_press = input("\nPress any key to start...\n")
    print("-------------------------")
    print("Starting Ping")
    print("-------------------------\n")

    mysocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create a UDP socket for the client
    server_address = ('127.0.0.1', 12000)  #Set
    mysocket.settimeout(1)  # Sets a timeout value 1 seconds
    rtt = []  #To store the all rtt
    try:
        for i in range(1,11):
            start = time.time()  # Start time send message to server
            message = 'Ping ' + str(i) + " " + time.ctime(start)
            try:
                sent = mysocket.sendto(message.encode("utf-8"), server_address)
                print("Sent " + message)
                data, server = mysocket.recvfrom(4096)  # Maximum data received 4096 bytes i.e buffer size
                print("Received " + str(data))
                end = time.time()
                elapsed = end - start
                print("Time: " + str(elapsed * 1000) + " Milliseconds\n")
                rtt.append(elapsed)
            except socket.timeout:
                print("#" + str(i) + " Requested Timed out\n")
    finally:
        print("Finish ping, closing socket")
        print("-------------------------")
        try:
            print("Statistics")
            print("Average RTT: " + str(sum(rtt) / len(rtt) * 1000) + " Milliseconds")
            print("Max RTT: " + str(max(rtt) * 1000) + " Milliseconds")
            print("Min RTT: " + str(min(rtt) * 1000) + " Milliseconds")
            print("Packet Loss: " + str((10 - len(rtt)) * 10) + "%")
            print("-------------------------")
            mysocket.close()
        except:
            print("Server Is Down: No packets received")
            mysocket.close()
            continue
```

# UDP PING SYSTEM - KEY FEATURES

## Features:

- Simulates 30% packet loss.
- Calculates and displays RTT for each successful ping.
- Provides summary statistics (min, max, average RTT, packet loss rate).

```
PS C:\Users\adity\Downloads\UDP Pinger> python server.py
Started UDP Server IP Address: 127.0.0.1 and Port: 12000
```

Server

```
Sent Ping 5 Fri Sep 20 13:12:19 2024
Received b'PING 5 FRI SEP 20 13:12:19 2024'
Time: 0.0 Milliseconds

Sent Ping 6 Fri Sep 20 13:12:19 2024
#6 Requested Timed out

Sent Ping 7 Fri Sep 20 13:12:20 2024
Received b'PING 7 FRI SEP 20 13:12:20 2024'
Time: 0.0 Milliseconds

Sent Ping 8 Fri Sep 20 13:12:20 2024
Received b'PING 8 FRI SEP 20 13:12:20 2024'
Time: 0.0 Milliseconds

Sent Ping 9 Fri Sep 20 13:12:20 2024
Received b'PING 9 FRI SEP 20 13:12:20 2024'
Time: 0.0 Milliseconds

Sent Ping 10 Fri Sep 20 13:12:20 2024
Received b'PING 10 FRI SEP 20 13:12:20 2024'
Time: 0.0 Milliseconds

Finish ping, closing socket
------------------------
Statistics
Average RTT: 0.22212664286295572 Milliseconds
Max RTT: 0.9996891021728516 Milliseconds
Min RTT: 0.0 Milliseconds
Packet Loss: 10%
------------------------
```

Client →

# PART-02

# Modify the client and server: UDP Heartbeat

# UDP HEARTBEAT SYSTEM - SERVER MODIFICATIONS

## Calculate Time Difference

he server calculates the time difference between receiving the packet and the client's timestamp.

## Send Information Back

This time difference is sent back to the client.

## Send the Response

If the value of rand< 4 , The server will drop that packet .

```python
import random
from socket import *  #socket library
import time  #time library

serverSocket = socket(AF_INET, SOCK_DGRAM)  # Create a UDP socket for the server
serverSocket.bind(('127.0.0.1', 12000))  # IP and Port no of Server to listen
print("Started UDP Server IP Address: 127.0.0.1 and Port: 12000")  # Print string on screen

while True:  # Run program forever
    rand = random.randint(1, 10)  # random number between 0 and 10
    message, address = serverSocket.recvfrom(1024)
    recv_time = time.time()  # Record the time the message was received
    if rand < 4:  # < 30% without reply message
        continue
    seq_num, timestamp = message.decode().split()
    timestamp = float(timestamp)
    time_diff = recv_time - timestamp
    response_message = f"{seq_num} {time_diff}"
    serverSocket.sendto(response_message.encode(), address)  # Send the time difference back
```

# UDP HEARTBEAT SYSTEM - CLIENT MODIFICATIONS

## Send Heartbeat Messages

The client sends up to 1000 heartbeat messages to the server.

## Detect Server Downtime

The client detects server downtime after 3 consecutive missed responses.

## Provide Detailed Statistics

he client provides detailed statistics, including packet loss rate and round-trip time (RTT).



```python
import socket
import time  # Import time library

consecutive_misses = 0  # Track consecutive missing responses
max_misses = 3  # Maximum allowed consecutive misses

while True:
    start_key_press = input("\nPress any key to start...")

    print("------------------------")
    print("Starting Heartbeat")
    print("------------------------\n")

    mysocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Create a UDP socket for the client
    server_address = ('127.0.0.1', 12000)  # Set IP Address and Port Number of Socket
    mysocket.settimeout(1)  # Sets a timeout value 1 seconds
    rtt = []  # Create an empty list to store round trip time
    try:  # Infinite loop to continuously send messages to the server
        for i in range(1000):  # Adjust the range as needed
            start = time.time()  # Start time send message to server
            message = f"{i} {start}"
            try:
                sent = mysocket.sendto(message.encode("utf-8"), server_address)
                print("Sent " + message)
                data, server = mysocket.recvfrom(4096)  # Maximum data received 4096 bytes i.e buffer
                print("Received " + str(data))
                end = time.time()
                elapsed = end - start
                print("Time: " + str(elapsed * 1000) + " Milliseconds\n")
                rtt.append(elapsed)
                consecutive_misses = 0  # Reset consecutive misses on successful response
            except socket.timeout:
                print("#" + str(i) + " Requested Timed out\n")
                consecutive_misses += 1
                if consecutive_misses >= max_misses:
                    print("Server is assumed to be down after 3 consecutive misses.")
                    break
    finally:
        print("Finish heartbeat, closing socket")
        print("------------------------")
        print("Statistics")
        if rtt:
            print("Average RTT: " + str(sum(rtt) / len(rtt) * 1000) + " Milliseconds")
            print("Max RTT: " + str(max(rtt) * 1000) + " Milliseconds")
            print("Min RTT: " + str(min(rtt) * 1000) + " Milliseconds")
        print("Packet Loss: " + str((1000 - len(rtt)) * 100 / 1000) + "%")
        print("------------------------")
        mysocket.close()
```

# UDP HEARTBEAT SYSTEM - KEY FEATURES

## Features:

- Simulates 30% packet loss.
- Identifies server downtime after 3 consecutive missed responses.
- Provides summary statistics (min, max, average RTT, packet loss rate).
- The server calculates and reports the time difference between receiving the packet and the client's timestamp.

```
PS C:\Users\adity\Downloads\UDP Pinger\part2> python server2.py
Started UDP Server IP Address: 127.0.0.1 and Port: 12000
```

Server

Client →

```
Sent 43 1726820517.3813171
#43 Requested Timed out

Sent 44 1726820518.3919091
Received b'44 0.0'
Time: 0.0 Milliseconds

Sent 45 1726820518.3929527
Received b'45 0.0'
Time: 0.0 Milliseconds

Sent 46 1726820518.3939083
Received b'46 0.0'
Time: 0.0 Milliseconds

Sent 47 1726820518.3939083
Received b'47 0.0'
Time: 0.0 Milliseconds

Sent 48 1726820518.3939083
#48 Requested Timed out

Sent 49 1726820519.4032137
#49 Requested Timed out

Sent 50 1726820520.4138696
#50 Requested Timed out

Server is assumed to be down after 3 consecutive misses.
Finish heartbeat, closing socket
------------------------
Statistics
Average RTT: 0.2472996711730957 Milliseconds
Max RTT: 1.0364055633544922 Milliseconds
Min RTT: 0.0 Milliseconds
Packet Loss: 96.4%
------------------------
```

# WHY WE CHOOSE 1000 ITERATIONS ?

- **Bernoulli Trial Model:**

  Success (Packet Received) Probability: ( p = 0.7 )

  Failure (Packet Lost) Probability: ( q = 1 - p = 0.3 )

- **Recurrence Relation:** $E_n = \frac{2}{p} E_{n-1} + \frac{1}{p}$

- **Probability of one instance of 3 consecutive failure in Different Sample Sizes**

For 10 Requests : 2.7%

For 100 Requests : 79.03%

For 1000 Requests:99.99%

**Expected Value = 174 pings**

# Running On Different System

# Troubleshooting and Solutions

## Network Connection

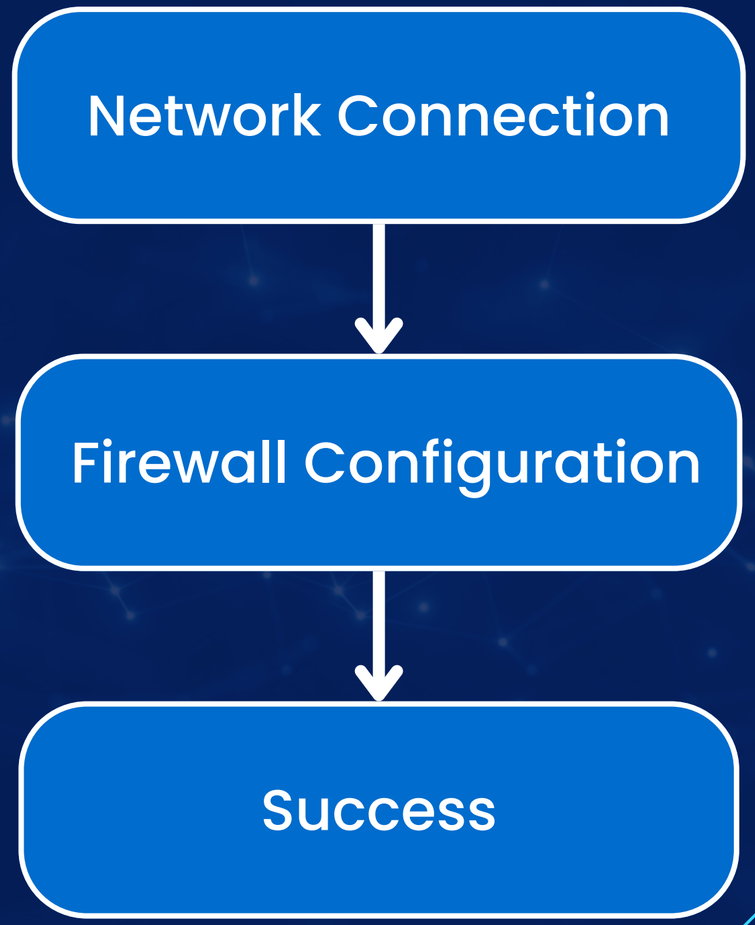
Difficulty in pinging the server.So, We Connected server and client to the same network.

## Firewall Configuration

Server port not open for UDP traffic. So, Added CN_port with UDP and port number 12000 in the firewall and opened it to listen.

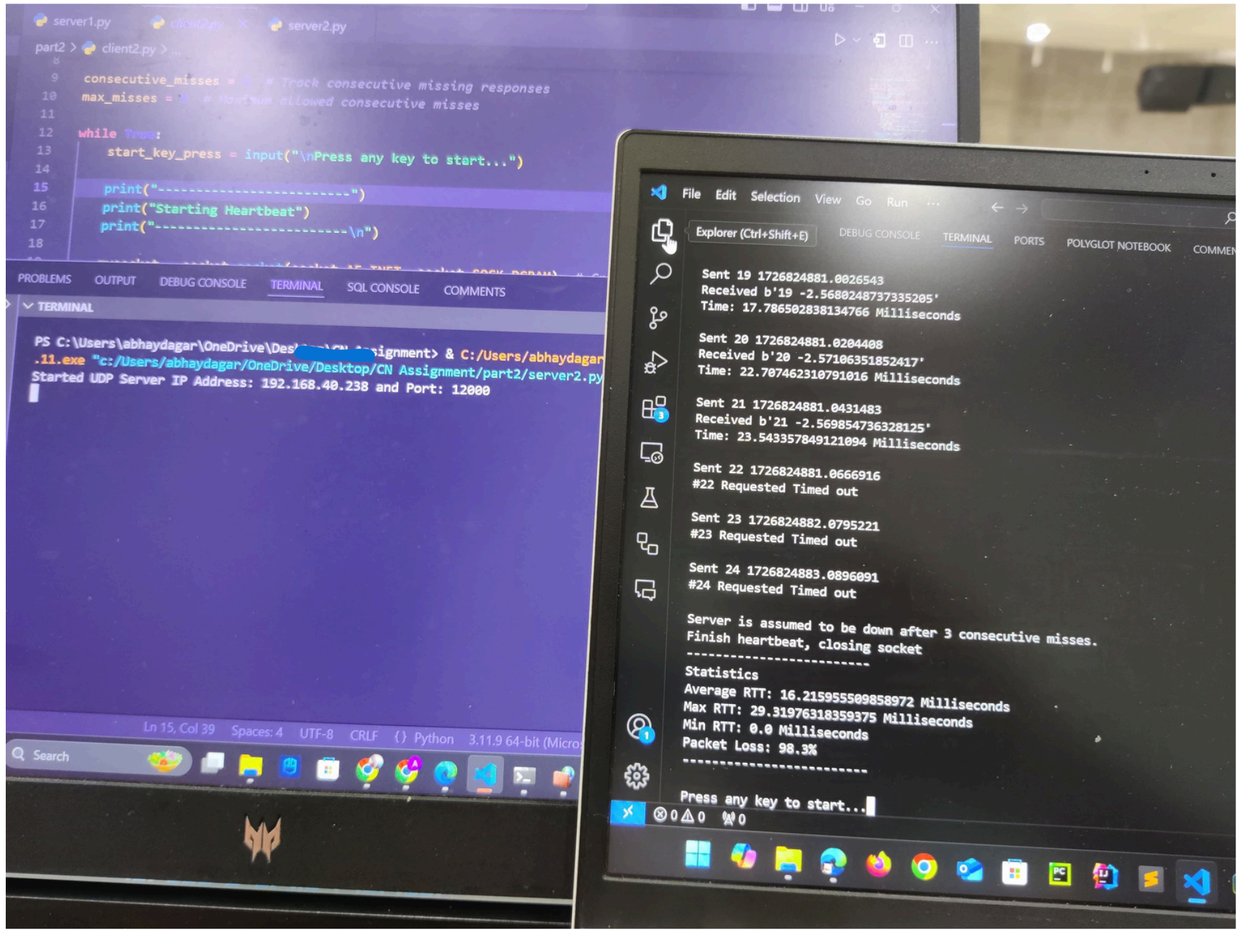
## Success

Server became available to ping successfully

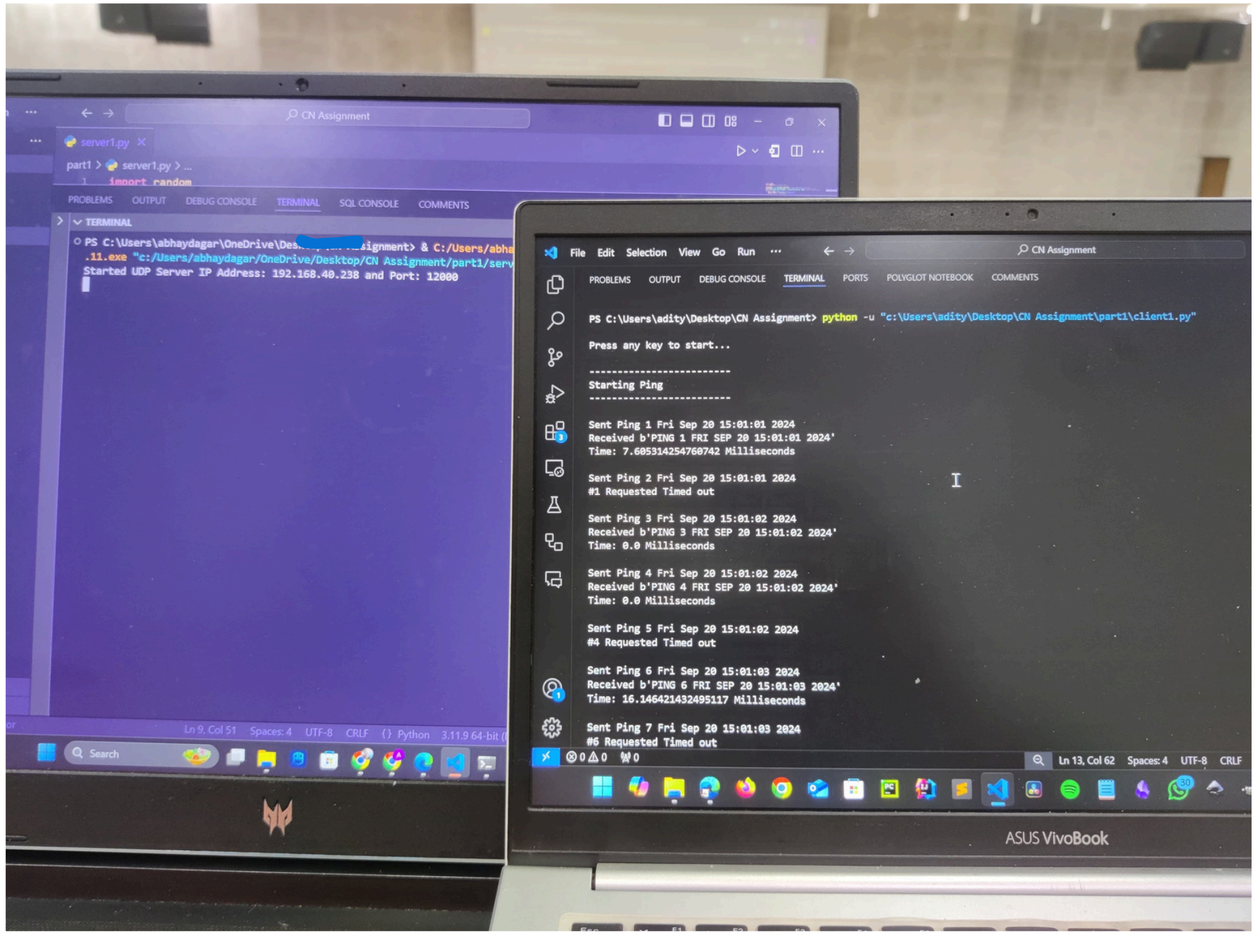


# Running On Different System

## Q1 - Pinger




## Q2 - HeartBeat



# THANK YOU FOR YOUR ATTENTION

Aditya Sharma 2022038
Abhay Dagar 2022014