CSE 232 Section B: Computer Networks: Programming Assignment 1: UDP Pinger Lab

Ritika Thakur (2022408) Swarnima Prasad (2022525)

1 About UDP Pinger

A UDP Pinger is a simple application that sends and receives UDP packets to check if a server is reachable and measure the Round Trip Time (RTT). Key Features of UDP:

1. **Connectionless protocol**: UDP does not establish a connection before sending data which makes it fast but unreliable.
2. **Test server responsiveness**: UDP Pinger Client sends a series of messages with timestamps and sequences to the server to test its responsiveness.
3. **Simulate Packet loss**: UDP is unreliable and packet loss may occur depending on the network conenction. UDP Pinger simulates such packet losses but randomly selecting which packets to respond to.
4. **Timeout**: Since UDP is unreliable, client waits for response for a specified time before considering the packet lost.
5. **Round-Trip Time (RTT)**: The client measures the time it takes for the message to go to the server and return (RTT) providing an estimate of network latency.
6. **Packet Loss Detection**: Packet loss can be detected based on missed responses using:
   (a) RTT for each packet
   (b) Packet loss percentage
   (c) Minimum, maximum and average RTT for all pings

2 Part 1: Implement Client code according to given Server code to implement UDP Pinger

2.1 Server Code

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', 12000))
while True:
    rand = random.randint(0, 10)
    message, address = serverSocket.recvfrom(1024)
    message = message.upper()
    if rand < 4:
        continue
    serverSocket.sendto(message, address)
```

The server code provided to us sits in an infinite loop listening for incoming UDP packets. When a packet comes in and if a randomized integer is greater than or equal to 4, the server simply capitalizes the encapsulated data and sends it back to the client.

2.2 Client Code

```
class UDPPingerClient:
    tot_packets = 10
    lost_packets = 0
    RTTs = []
    min_rtt = max_rtt = avg_rtt = None

    def __init__ (self, serverName, serverPort):
        self.serverName = serverName
        self.serverPort = serverPort
        self.clientSocket = socket(family = AF_INET, type = SOCK_DGRAM, proto=0,
            fileno=None)
        self.clientSocket.settimeout(1)

    def calculateRTT(self, end_time, start_time):
        return  end_time - start_time

    def ping(self, tot_packets):
        for i in range(1, tot_packets + 1):
            send_time = time.time()
            message = f"Ping {i} {send_time}"
            try:
                self.clientSocket.sendto(message.encode(), (self.serverName, self.
                    serverPort))
                start_time = time.time()
                response, serverAddress = self.clientSocket.recvfrom(1024)
                end_time = time.time()
                rtt = self.calculateRTT(end_time, start_time)
                self.RTTs.append(rtt)
                print(f"Reply from {self.serverName}: {response.decode()}")
                print(f"RTT: {rtt:.6f} seconds")
            except timeout:
                print("Request timed out")
                self.lost_packets += 1
        if self.RTTs:
            self.min_rtt = min(self.RTTs)
            self.max_rtt = max(self.RTTs)
            self.avg_rtt = sum(self.RTTs) / len(self.RTTs)

    def print_stats(self):
        print(f"\n--- Ping statistics ---")
        print(f"Packets: Sent = {self.tot_packets}, Received = {self.tot_packets -
            self.lost_packets}, Lost = {self.lost_packets} ({(self.lost_packets / self
            .tot_packets) * 100}% packet loss)")
        if self.RTTs:
            print(f"RTTs: Min = {self.min_rtt:.6f}s, Max = {self.max_rtt:.6f}s, Avg =
                {self.avg_rtt:.6f}s")

    def close(self):
        self.clientSocket.shutdown(SHUT_RDWR)
        self.clientSocket.close()
```

The UDPPingerClient.py implements a UDP Pinger Client that sends a series of ping messages to a UDP server and measures the round-trip time (RTT) for each message. It sends 10 ping requests, calculates the RTT for each response, and handles packet loss by counting requests that time out after 1 second. After all pings are sent, it prints the number of packets sent, received, lost, and the packet loss percentage, along with the minimum, maximum, and average RTTs. Finally, it closes the client socket.

2.3 Outputs

2.3.1 Local Host

Below are some outputs when UDPPingerClient.py and UDPPingerServer.py are run on the same machine on different terminals:

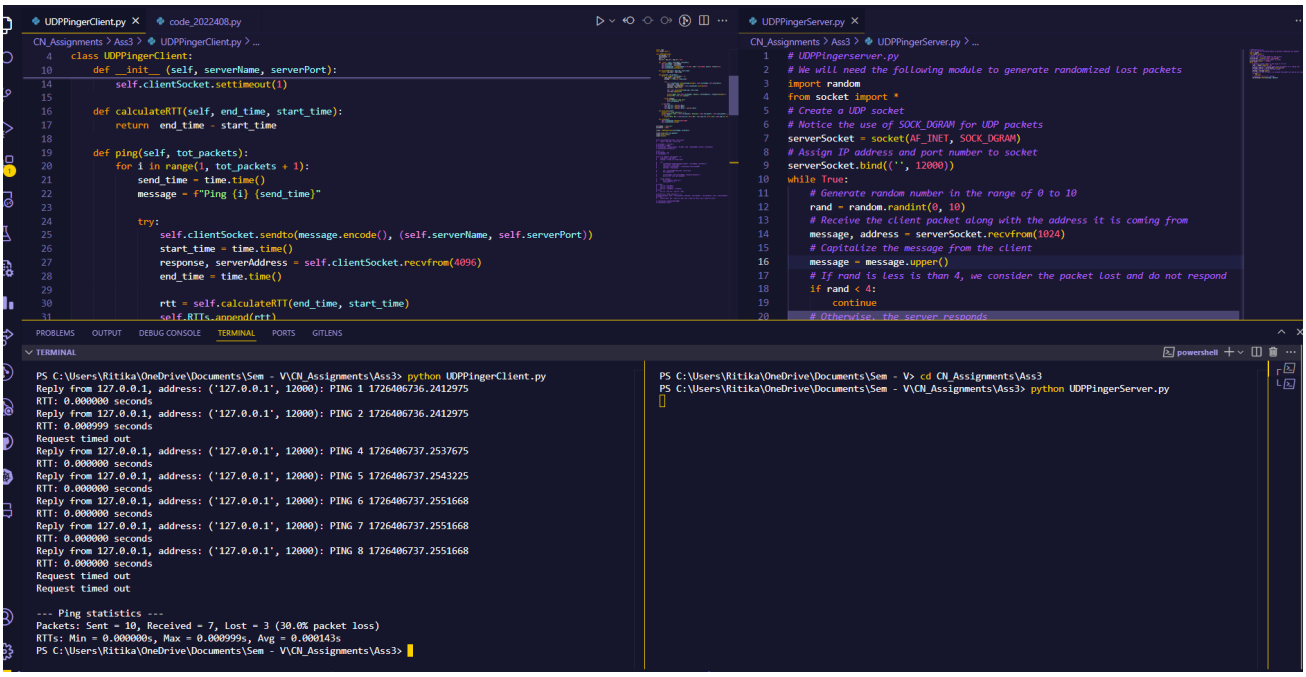

Figure 1: UDPPingerClient simulating 30% loss

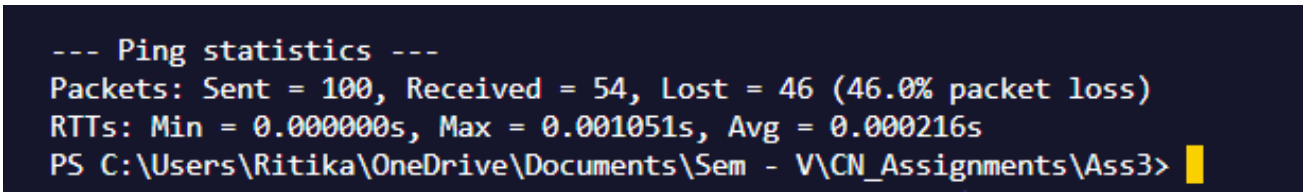

Figure 2: UDPPingerClient for 100 packets

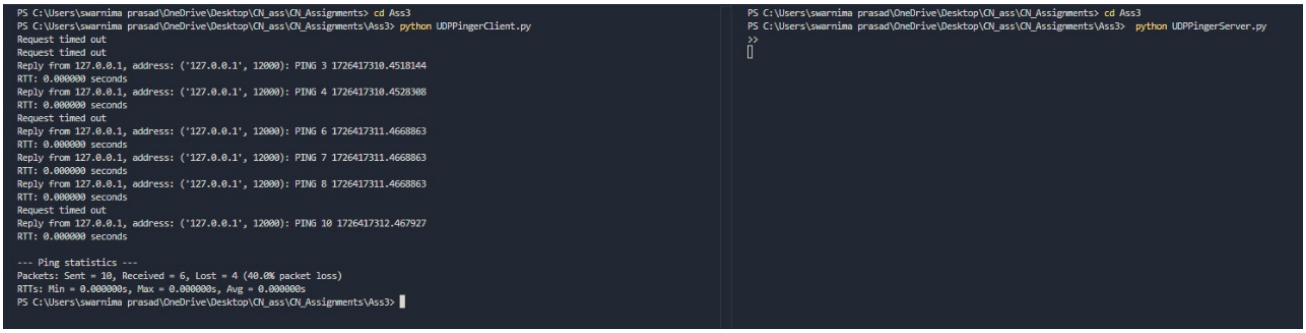

Figure 3: UDPPingerClient for 100 packets

2.3.2 Different Machines

Below are some outputs when UDPPingerClient.py and UDPPingerServer.py are run on different machines:

```
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments\Ass3> python UDPPingerClient.py
Reply from 192.168.32.232: PING 1 1726776829.0851982
RTT: 0.006491 seconds
Reply from 192.168.32.232: PING 2 1726776829.092689
RTT: 0.010354 seconds
Reply from 192.168.32.232: PING 3 1726776829.10406
RTT: 0.006994 seconds
Reply from 192.168.32.232: PING 4 1726776829.1110542
RTT: 0.009669 seconds
Reply from 192.168.32.232: PING 5 1726776829.1207235
RTT: 0.007297 seconds
Request timed out
Reply from 192.168.32.232: PING 7 1726776830.1329608
RTT: 0.006393 seconds
Request timed out
Request timed out
Reply from 192.168.32.232: PING 10 1726776832.155001
RTT: 0.008133 seconds

--- Ping statistics ---
Packets: Sent = 10, Received = 7, Lost = 3 (30.0% packet loss)
RTTs: Min = 0.006393s, Max = 0.010354s, Avg = 0.007905s
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments\Ass3>
```

Figure 4: UDPPingerClient simulating 30% loss across different machines

3 About UDP Heartbeat

A UDP heartbeat is a lightweight message sent at regular intervals between a client and a server to check if the connection is still alive and responsive. In networking, it's often used to monitor the status of the server, ensuring that it's up and running, or to keep a connection active without transferring large amounts of data.

UDP heartbeats are typically used when:

1. Speed is critical: For example, in real-time systems like video games, VoIP, or monitoring tools, UDP is preferred because of its low latency.
2. The application can tolerate some packet loss: Losing occasional heartbeats is acceptable as long as it's infrequent, and missing several in a row may signal a problem.

4 Part 2: Modify Client and Server code to implement UDP Heartbeat

4.1 Server Code

```
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', 12000))
print("The server is ready to receive heartbeats...")
while True:
    rand = random.randint(0, 10)
    message, clientAddress = serverSocket.recvfrom(1024)
    if rand < 4:
        continue
    decoded_message = message.decode()
    print(f"Received message: {decoded_message}")
```

```
    _, sequence_number, sent_time = decoded_message.split()
    receive_time = time.time()
    time_diff = receive_time - float(sent_time)
    response_message = f"{sequence_number} {time_diff:.6f}"
    serverSocket.sendto(response_message.encode(), clientAddress)
```

4.2 Client Code

```
class UDPPingerClient:
    tot_packets = 1000
    missed_heartbeats = 0
    consecutive_misses = 0
    total_sent=0

    def __init__ (self, serverName, serverPort):
        self.serverName = serverName
        self.serverPort = serverPort
        self.clientSocket = socket(family = AF_INET, type = SOCK_DGRAM)
        self.clientSocket.settimeout(1)

    def ping(self, tot_packets):
        for i in range(1, tot_packets + 1):
            send_time = time.time()
            self.total_sent+=1
            message = f"Ping {i} {send_time}"
            try:
                self.clientSocket.sendto(message.encode(), (self.serverName, self.
                    serverPort))
                response, serverAddress = self.clientSocket.recvfrom(1024)
                recv_time = time.time()
                response_message = response.decode()
                sequence_number, time_diff = response_message.split()
                time_diff = float(time_diff)
                print(f"Received heartbeat response {sequence_number}: Time difference
                     = {time_diff:.6f} seconds")
                self.consecutive_misses = 0
            except timeout:
                self.consecutive_misses += 1
                self.missed_heartbeats += 1
                print(f"Heartbeat {i}: Request timed out")
            if self.consecutive_misses == 3:
                print("Server is down! 3 consecutive heartbeat responses were missed."
                    )
                break

    def print_stats(self):
        print(f"\n--- Heartbeat statistics ---")
        print(f"Packets: Sent = {self.total_sent}, Received = {self.total_sent - self.
            missed_heartbeats}, Lost = {self.missed_heartbeats} ({(self.
            missed_heartbeats / self.total_sent) * 100}% packet loss)")

    def close(self):
        self.clientSocket.shutdown(SHUT_RDWR)
        self.clientSocket.close()
```

4.3 Outputs

4.3.1 Local Host

Below are some outputs when UDPHeartbeatClient.py and UDPHeartbeatServer.py are run on the same machine on different terminals:

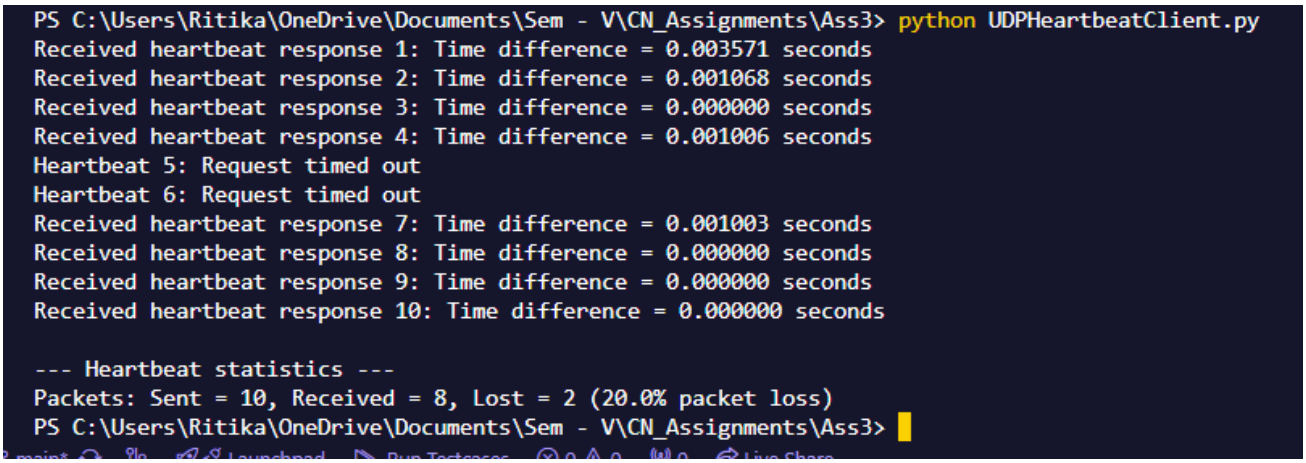

Figure 5: UDP Heartbeat with 10 packets

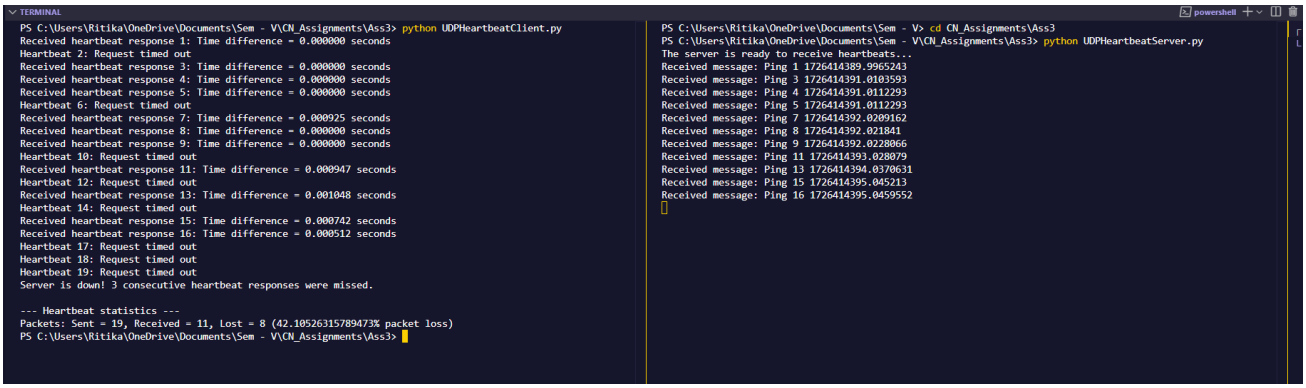

Figure 6: UDP Heartbeat with 100 packets

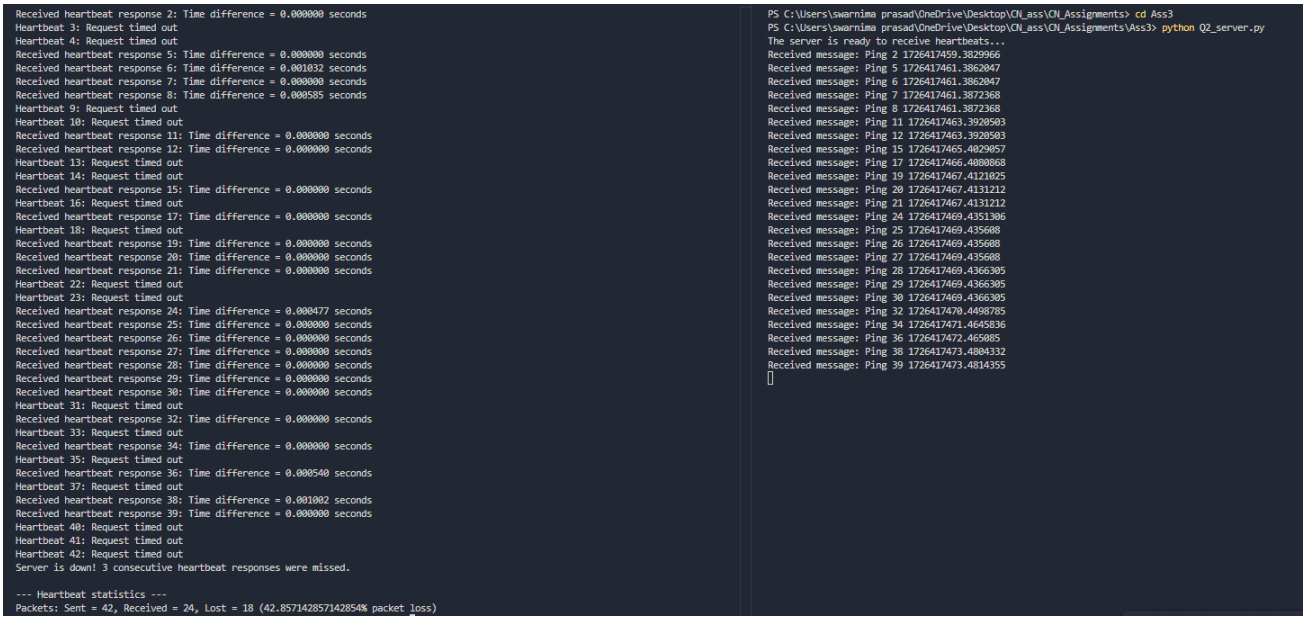

Figure 7: UDP Heartbeat with 1000 packets

4.3.2 Different Machines

Below are some outputs when UDPHeartbeatClient.py and UDPHeartbeatServer.py are run on different machines:

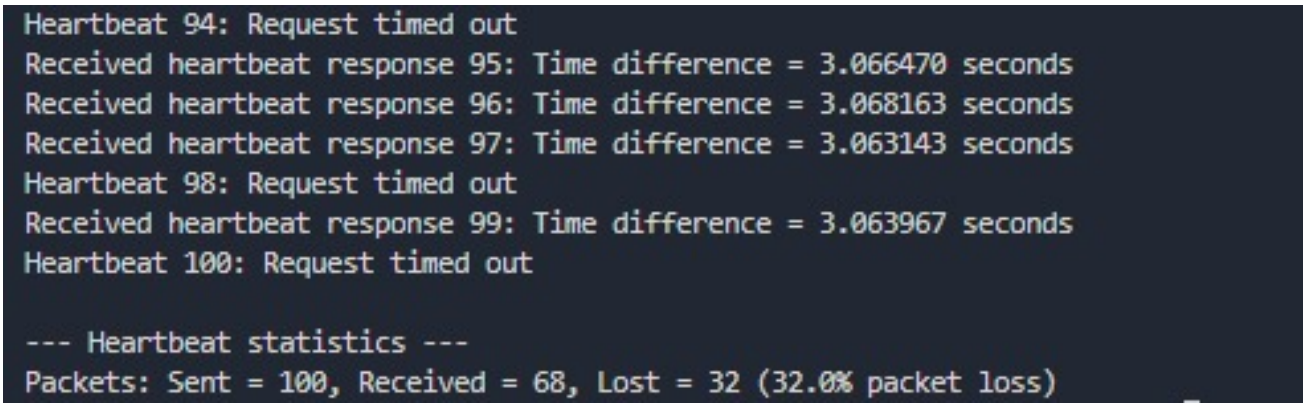

Figure 8: UDP Heartbeat with 100 packets across different machines not facing 3 consecutive losses

```
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments\Ass3> python UDPHeartbeatClient.py
Received heartbeat response 1: Time difference = 3.066142 seconds
Received heartbeat response 2: Time difference = 3.063971 seconds
Heartbeat 3: Request timed out
Received heartbeat response 4: Time difference = 3.063734 seconds
Received heartbeat response 5: Time difference = 3.063859 seconds
Heartbeat 6: Request timed out
Heartbeat 7: Request timed out
Received heartbeat response 8: Time difference = 3.067698 seconds
Received heartbeat response 9: Time difference = 3.064611 seconds
Received heartbeat response 10: Time difference = 3.065667 seconds
Received heartbeat response 11: Time difference = 3.065002 seconds
Heartbeat 12: Request timed out
Received heartbeat response 13: Time difference = 3.068117 seconds
Received heartbeat response 14: Time difference = 3.067029 seconds
Received heartbeat response 15: Time difference = 3.070457 seconds
Received heartbeat response 16: Time difference = 3.065941 seconds
Heartbeat 17: Request timed out
Received heartbeat response 18: Time difference = 3.064579 seconds
Received heartbeat response 19: Time difference = 3.064699 seconds
Heartbeat 20: Request timed out
Heartbeat 21: Request timed out
Received heartbeat response 22: Time difference = 3.069790 seconds
Heartbeat 23: Request timed out
Heartbeat 24: Request timed out
Heartbeat 25: Request timed out
Server is down! 3 consecutive heartbeat responses were missed.

--- Heartbeat statistics ---
Packets: Sent = 25, Received = 15, Lost = 10 (40.0% packet loss)
PS C:\Users\swarnima prasad\OneDrive\Desktop\CN_ass\CN_Assignments\Ass3>
```

Figure 9: UDP Heartbeat for 1000 packets

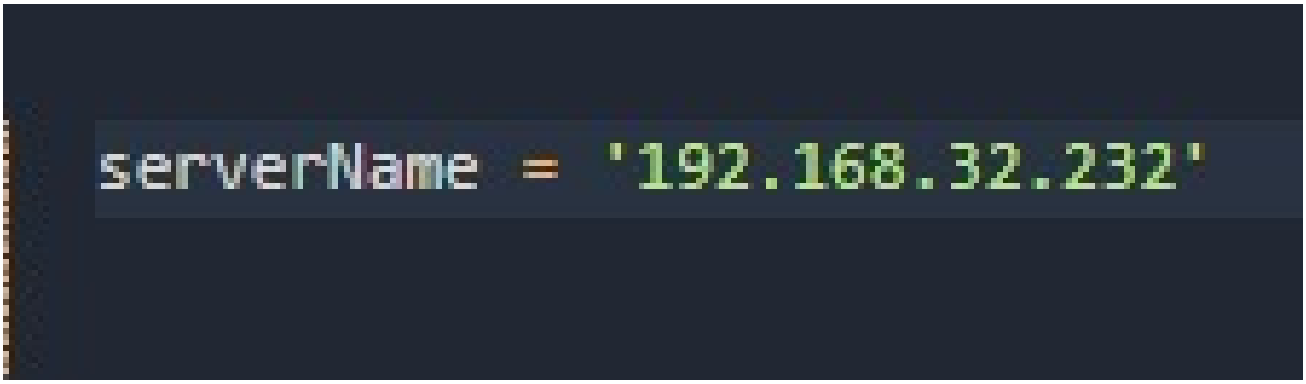

Figure 10: ServerName of the machine on which UDPHeartbeatServer is running

5 Observations and Conclusions

For UDP Pinger:

1. In the first output (Figure 1), the UDPPingerClient simulates a 30% packet loss rate. This is reflected by the missing replies for some of the pings, as indicated by the "Request timed out" message. The loss is consistent with the randomness in packet drops by the server, which uses a random number generator to drop packets.

2. The second output (Figure 2) shows a test with 100 packets, demonstrating the ability of the client to handle a larger number of pings. The packet loss percentage does not shoot up by a huge margin, and the RTT statistics are more representative over a larger sample size.

3. Running UDPPingerServer.py and UDPPingerClient.py on different machines does not impact the packet loss and we can still simulate a 30% packet loss.

For UDP Heartbeat:

1. In testing, we sent 1000 packets to simulate a long-term connection and observe behavior over time.

2. Continuously packets are sent until program logic detects 3 consecutive failures, which could happen after any number of packets, depending on the state of the network or server.

3. While timeout is kept for 1 second the server on average gets down within 50 packets are sent.

4. For different machines we notice that with timeout for 1 second we are able to send 100 packets without 3 consecutive misses.

6 References

- Python Socket Documentation
- Python Socket Howto
- Socket Programming GitHub Repository
- Socket Programming 2 - Tutorial slides