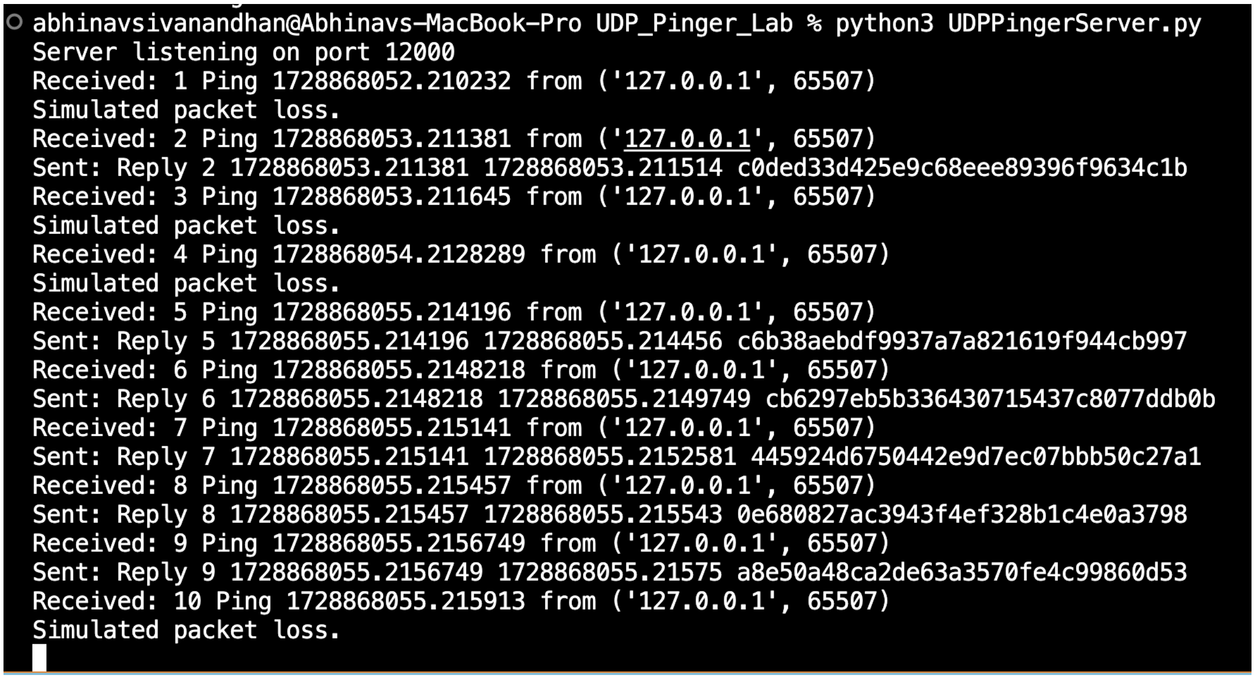
Computer Networking Socket Programming Lab 2

- Abhinav Sivanandhan
- as17904

Content:

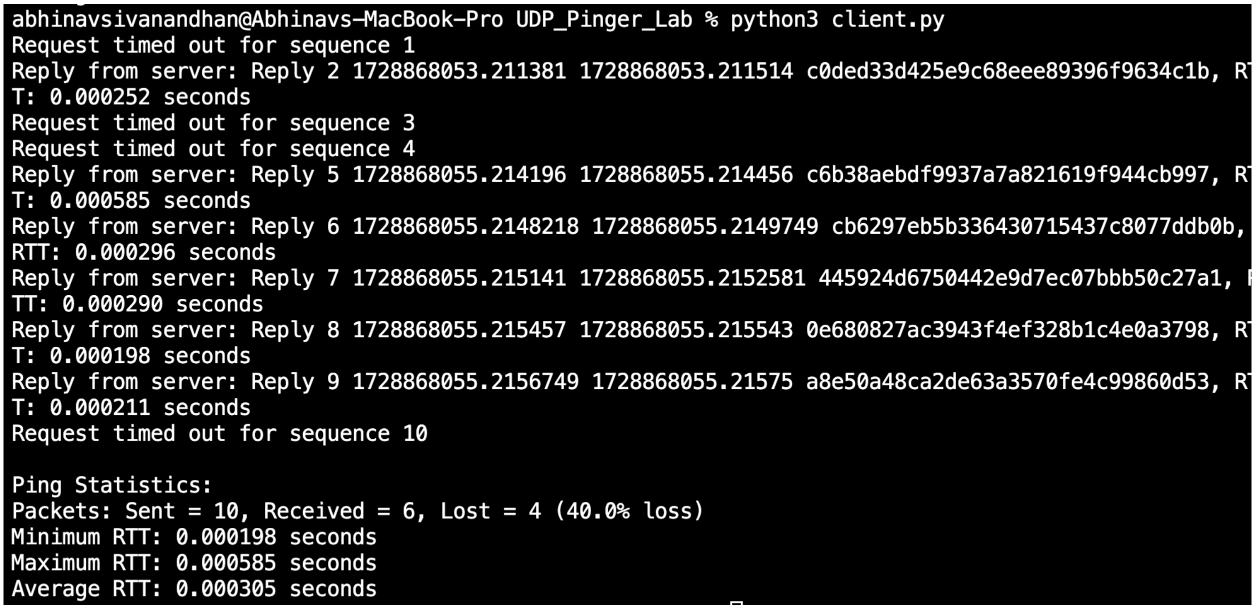
- UDP Pinger Screenshots and Code
- Optional exercise 1 Screenshots and Code
- Optional exercise 2 Screenshots and Code

Output at UDPPingerServer.py:



```
○ abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab % python3 UDPPingerServer.py
  Server listening on port 12000
  Received: 1 Ping 1728868052.210232 from ('127.0.0.1', 65507)
  Simulated packet loss.
  Received: 2 Ping 1728868053.211381 from ('127.0.0.1', 65507)
  Sent: Reply 2 1728868053.211381 1728868053.211514 c0ded33d425e9c68eee89396f9634c1b
  Received: 3 Ping 1728868053.211645 from ('127.0.0.1', 65507)
  Simulated packet loss.
  Received: 4 Ping 1728868054.2128289 from ('127.0.0.1', 65507)
  Simulated packet loss.
  Received: 5 Ping 1728868055.214196 from ('127.0.0.1', 65507)
  Sent: Reply 5 1728868055.214196 1728868055.214456 c6b38aebdf9937a7a821619f944cb997
  Received: 6 Ping 1728868055.2148218 from ('127.0.0.1', 65507)
  Sent: Reply 6 1728868055.2148218 1728868055.2149749 cb6297eb5b336430715437c8077ddb0b
  Received: 7 Ping 1728868055.215141 from ('127.0.0.1', 65507)
  Sent: Reply 7 1728868055.215141 1728868055.2152581 445924d6750442e9d7ec07bbb50c27a1
  Received: 8 Ping 1728868055.215457 from ('127.0.0.1', 65507)
  Sent: Reply 8 1728868055.215457 1728868055.215543 0e680827ac3943f4ef328b1c4e0a3798
  Received: 9 Ping 1728868055.2156749 from ('127.0.0.1', 65507)
  Sent: Reply 9 1728868055.2156749 1728868055.21575 a8e50a48ca2de63a3570fe4c99860d53
  Received: 10 Ping 1728868055.215913 from ('127.0.0.1', 65507)
  Simulated packet loss.
```

Output at client.py:



```
abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab % python3 client.py
Request timed out for sequence 1
Reply from server: Reply 2 1728868053.211381 1728868053.211514 c0ded33d425e9c68eee89396f9634c1b, R
T: 0.000252 seconds
Request timed out for sequence 3
Request timed out for sequence 4
Reply from server: Reply 5 1728868055.214196 1728868055.214456 c6b38aebdf9937a7a821619f944cb997, R
T: 0.000585 seconds
Reply from server: Reply 6 1728868055.2148218 1728868055.2149749 cb6297eb5b336430715437c8077ddb0b,
RTT: 0.000296 seconds
Reply from server: Reply 7 1728868055.215141 1728868055.2152581 445924d6750442e9d7ec07bbb50c27a1, R
TT: 0.000290 seconds
Reply from server: Reply 8 1728868055.215457 1728868055.215543 0e680827ac3943f4ef328b1c4e0a3798, R
T: 0.000198 seconds
Reply from server: Reply 9 1728868055.2156749 1728868055.21575 a8e50a48ca2de63a3570fe4c99860d53, R
T: 0.000211 seconds
Request timed out for sequence 10

Ping Statistics:
Packets: Sent = 10, Received = 6, Lost = 4 (40.0% loss)
Minimum RTT: 0.000198 seconds
Maximum RTT: 0.000585 seconds
Average RTT: 0.000305 seconds
```

UDPPingerServer.py:

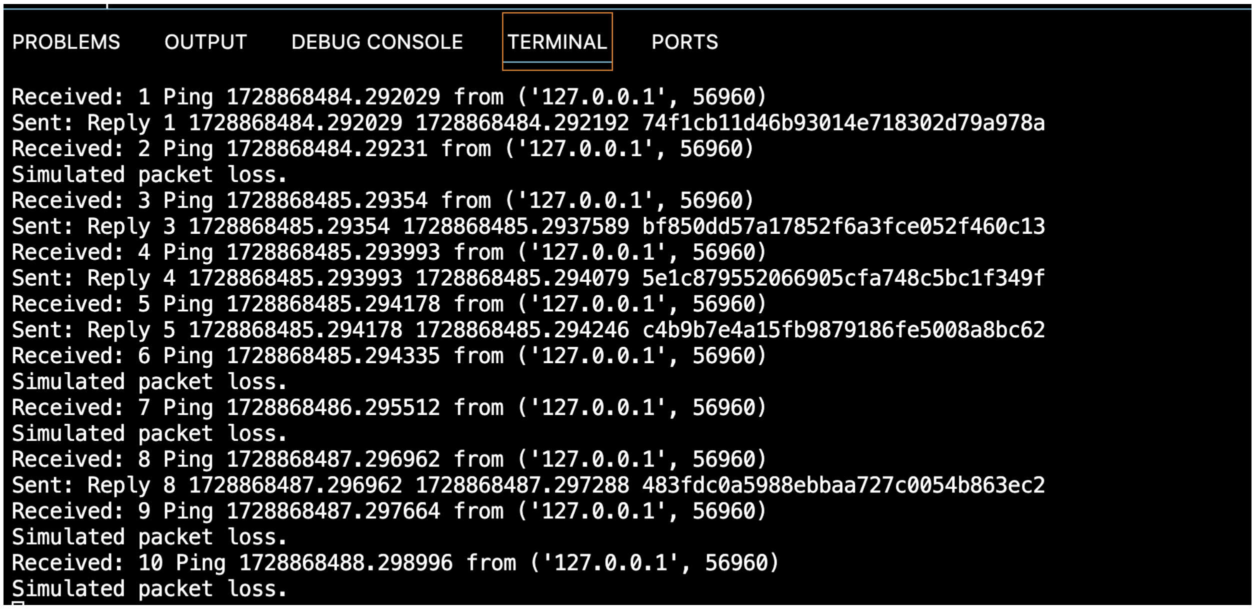
```
import random
from socket import *
import time
import hashlib
import sys

def serve(port):
    # Create a UDP socket (IPv4 + UDP)
    server_socket = socket(AF_INET, SOCK_DGRAM)
    server_socket.bind(('', port))  # Bind the socket to the specified port
    print(f"Server listening on port {port}")

    while True:
        try:
            # Simulate 30% packet loss with a random number
            rand = random.randint(0, 10)

            # Receive the client packet and address
            message, address = server_socket.recvfrom(1024)
            s_time = time.time()  # Server's receive time

            # Print the received message for debugging
            print(f"Received: {message.decode()} from {address}")

            # Simulate packet loss (skip sending response for 30% of packets)
            if rand < 4:
                print("Simulated packet loss.")
                continue

            # Parse the client message into sequence number, type, and timestamp
            m = message.decode().strip().split()
            if len(m) != 3:
                print(f"Invalid message format: {m}. Ignoring.")
                continue  # Ignore malformed messages

            seq = m[0]  # Sequence number (e.g., '9')
            ping_type = m[1]  # Type (should be 'Ping')
            c_time = m[2]  # Client's timestamp
            # Ensure the message type is 'Ping' to proceed
            if ping_type != "Ping":
                print(f"Unexpected message type: {ping_type}. Ignoring.")
                continue

            # Generate an MD5 message digest
            h =
hashlib.md5(f"seq:{seq},c_time:{c_time},s_time:{s_time},key:randomkey".encode()).hexdigest()

            # Prepare the reply message
            resp = f"Reply {seq} {c_time} {s_time} {h}"

            # Send the reply to the client
            server_socket.sendto(resp.encode(), address)
            print(f"Sent: {resp}")

        except KeyboardInterrupt:  # Handle Ctrl+C gracefully
            print("\nServer shutting down.")
            server_socket.close()  # Close the socket and release resources
            sys.exit()
        except Exception as e:  # Handle other exceptions
            print(f"Error: {e}")
            continue

if __name__ == '__main__':
    # Start the server on port 12000
    serve(12000)
```

Client.py:

```
from socket import *
import time

def ping(host, port):
    # List to store responses and RTTs
    resps = []
    client_socket = socket(AF_INET, SOCK_DGRAM)  # Create UDP socket
    client_socket.settimeout(1)  # Set 1-second timeout for responses

    # Send 10 ping messages
    for seq in range(1, 11):
        # Record the send time
        send_time = time.time()
        # Prepare the ping message in the correct format
        message = f"{seq} Ping {send_time}"

        try:
            # Send the message to the server
            client_socket.sendto(message.encode(), (host, port))

            # Receive the reply from the server
            data, server = client_socket.recvfrom(1024)
            recv_time = time.time()  # Record the time reply is received

            # Calculate RTT
            rtt = recv_time - send_time
            server_reply = data.decode().strip()  # Decode server's response

            # Store the reply and RTT in the response list
            resps.append((seq, server_reply, rtt))
            print(f"Reply from server: {server_reply}, RTT: {rtt:.6f} seconds")

        except timeout:
            # If no reply is received within 1 second, consider it timed out
            print(f"Request timed out for sequence {seq}")
            resps.append((seq, 'Request timed out', 0))

    # Close the socket after sending all pings
    client_socket.close()
    return resps

def compute_statistics(resps):
    # Filter successful RTTs for statistics calculation
    rtts = [rtt for seq, reply, rtt in resps if reply != 'Request timed out']
    packet_loss = (1 - len(rtts) / 10) * 100  # Calculate packet loss percentage

    print("\nPing Statistics:")
    print(f"Packets: Sent = 10, Received = {len(rtts)}, Lost = {10 - len(rtts)} ({packet_loss:.1f}%
loss)")

    if rtts:
        print(f"Minimum RTT: {min(rtts):.6f} seconds")
        print(f"Maximum RTT: {max(rtts):.6f} seconds")
        print(f"Average RTT: {sum(rtts) / len(rtts):.6f} seconds")

if __name__ == '__main__':
    # Ping the server on localhost at port 12000
    responses = ping('127.0.0.1', 12000)
    # Display ping statistics
    compute_statistics(responses)
```

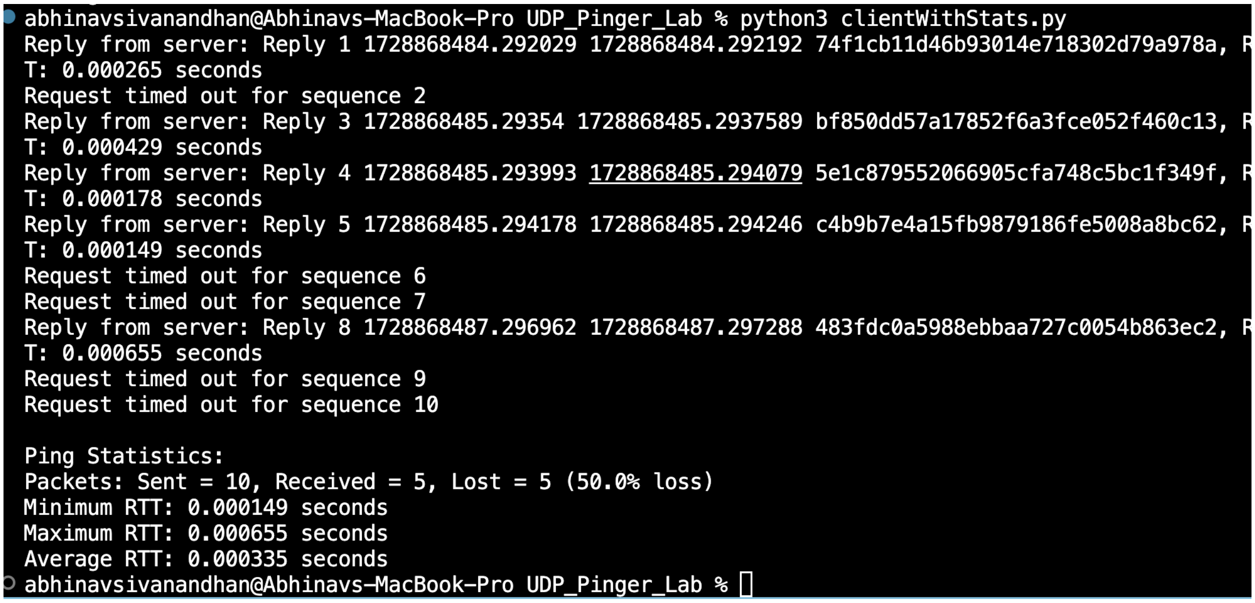
Optional Exercise 1 output:

At server:



```
Received: 1 Ping 1728868484.292029 from ('127.0.0.1', 56960)
Sent: Reply 1 1728868484.292029 1728868484.292192 74f1cb11d46b93014e718302d79a978a
Received: 2 Ping 1728868484.29231 from ('127.0.0.1', 56960)
Simulated packet loss.
Received: 3 Ping 1728868485.29354 from ('127.0.0.1', 56960)
Sent: Reply 3 1728868485.29354 1728868485.2937589 bf850dd57a17852f6a3fce052f460c13
Received: 4 Ping 1728868485.293993 from ('127.0.0.1', 56960)
Sent: Reply 4 1728868485.293993 1728868485.294079 5e1c879552066905cfa748c5bc1f349f
Received: 5 Ping 1728868485.294178 from ('127.0.0.1', 56960)
Sent: Reply 5 1728868485.294178 1728868485.294246 c4b9b7e4a15fb9879186fe5008a8bc62
Received: 6 Ping 1728868485.294335 from ('127.0.0.1', 56960)
Simulated packet loss.
Received: 7 Ping 1728868486.295512 from ('127.0.0.1', 56960)
Simulated packet loss.
Received: 8 Ping 1728868487.296962 from ('127.0.0.1', 56960)
Sent: Reply 8 1728868487.296962 1728868487.297288 483fdc0a5988ebbaa727c0054b863ec2
Received: 9 Ping 1728868487.297664 from ('127.0.0.1', 56960)
Simulated packet loss.
Received: 10 Ping 1728868488.298996 from ('127.0.0.1', 56960)
Simulated packet loss.
```

At client:



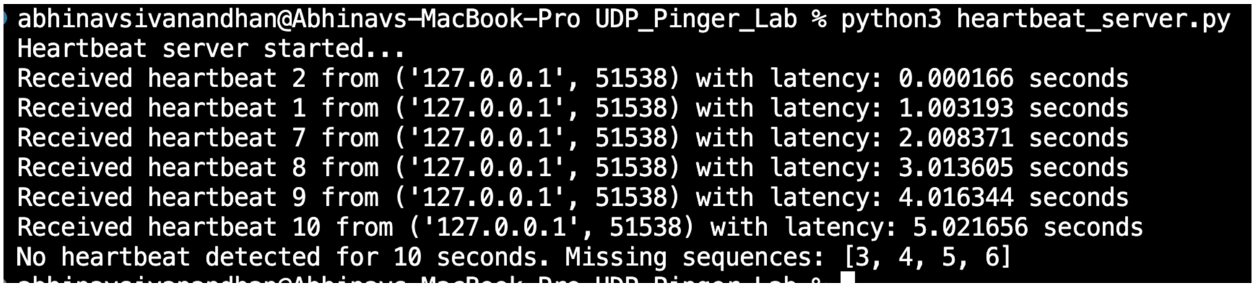
```
abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab % python3 clientWithStats.py
Reply from server: Reply 1 1728868484.292029 1728868484.292192 74f1cb11d46b93014e718302d79a978a, R
T: 0.000265 seconds
Request timed out for sequence 2
Reply from server: Reply 3 1728868485.29354 1728868485.2937589 bf850dd57a17852f6a3fce052f460c13, R
T: 0.000429 seconds
Reply from server: Reply 4 1728868485.293993 1728868485.294079 5e1c879552066905cfa748c5bc1f349f, R
T: 0.000178 seconds
Reply from server: Reply 5 1728868485.294178 1728868485.294246 c4b9b7e4a15fb9879186fe5008a8bc62, R
T: 0.000149 seconds
Request timed out for sequence 6
Request timed out for sequence 7
Reply from server: Reply 8 1728868487.296962 1728868487.297288 483fdc0a5988ebbaa727c0054b863ec2, R
T: 0.000655 seconds
Request timed out for sequence 9
Request timed out for sequence 10

Ping Statistics:
Packets: Sent = 10, Received = 5, Lost = 5 (50.0% loss)
Minimum RTT: 0.000149 seconds
Maximum RTT: 0.000655 seconds
Average RTT: 0.000335 seconds
abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab %
```

clientWithStats.py:

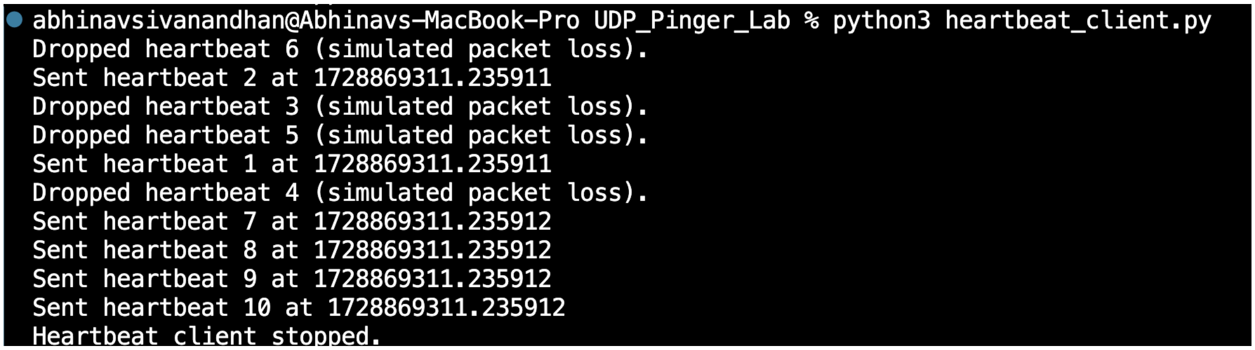
```
from socket import *
import time

def ping(host, port):
    resps = []  # To store responses and RTTs
    client_socket = socket(AF_INET, SOCK_DGRAM)  # Create UDP socket
    client_socket.settimeout(1)  # Set 1-second timeout for responses

    # Send 10 ping messages with proper formatting
    for seq in range(1, 11):
        send_time = time.time()  # Record the time when the ping is sent
        message = f"{seq} Ping {send_time}"  # Prepare the ping message

        try:
            # Send the message to the server at the specified host and port
            client_socket.sendto(message.encode(), (host, port))

            # Receive the reply from the server
            data, server = client_socket.recvfrom(1024)
            recv_time = time.time()  # Record the time reply is received

            # Calculate the round-trip time (RTT)
            rtt = recv_time - send_time

            # Decode the server's reply
            server_reply = data.decode().strip()

            # Store the reply and RTT in the response list
            resps.append((seq, server_reply, rtt))

            # Print the server's reply and RTT
            print(f"Reply from server: {server_reply}, RTT: {rtt:.6f} seconds")

        except timeout:
            # If no reply is received within 1 second, consider it timed out
            print(f"Request timed out for sequence {seq}")
            # Append the timeout information to the responses list
            resps.append((seq, 'Request timed out', 0))

    # Close the socket after sending all pings
    client_socket.close()
    return resps
```

```
def compute_statistics(resps):
    # Filter out successful RTTs for statistics calculation
    rtts = [rtt for seq, reply, rtt in resps if reply != 'Request timed out']

    # Calculate packet loss percentage
    packet_loss = (1 - len(rtts) / 10) * 100

    # Display the ping statistics
    print("\nPing Statistics:")
    print(f"Packets: Sent = 10, Received = {len(rtts)}, Lost = {10 - len(rtts)} ({packet_loss:.1f}%
loss)")

    # If there are successful pings, calculate RTT statistics
    if rtts:
        print(f"Minimum RTT: {min(rtts):.6f} seconds")
        print(f"Maximum RTT: {max(rtts):.6f} seconds")
        print(f"Average RTT: {sum(rtts) / len(rtts):.6f} seconds")

if __name__ == '__main__':
    # Ping the server on localhost at port 12000
    responses = ping('127.0.0.1', 12000)

    # Display the ping statistics after all pings are sent
    compute_statistics(responses)
```

Optional Exercise 2 output:



```
abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab % python3 heartbeat_server.py
Heartbeat server started...
Received heartbeat 2 from ('127.0.0.1', 51538) with latency: 0.000166 seconds
Received heartbeat 1 from ('127.0.0.1', 51538) with latency: 1.003193 seconds
Received heartbeat 7 from ('127.0.0.1', 51538) with latency: 2.008371 seconds
Received heartbeat 8 from ('127.0.0.1', 51538) with latency: 3.013605 seconds
Received heartbeat 9 from ('127.0.0.1', 51538) with latency: 4.016344 seconds
Received heartbeat 10 from ('127.0.0.1', 51538) with latency: 5.021656 seconds
No heartbeat detected for 10 seconds. Missing sequences: [3, 4, 5, 6]
abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab %
```



```
abhinavsivanandhan@Abhinavs-MacBook-Pro UDP_Pinger_Lab % python3 heartbeat_client.py
Dropped heartbeat 6 (simulated packet loss).
Sent heartbeat 2 at 1728869311.235911
Dropped heartbeat 3 (simulated packet loss).
Dropped heartbeat 5 (simulated packet loss).
Sent heartbeat 1 at 1728869311.235911
Dropped heartbeat 4 (simulated packet loss).
Sent heartbeat 7 at 1728869311.235912
Sent heartbeat 8 at 1728869311.235912
Sent heartbeat 9 at 1728869311.235912
Sent heartbeat 10 at 1728869311.235912
Heartbeat client stopped.
```

heartbeat_server.py:

```
from socket import *
import time
import sys

def serve(port):
    server_socket = socket(AF_INET, SOCK_DGRAM)  # Create UDP socket
    server_socket.bind(('', port))  # Bind to the specified port
    server_socket.settimeout(10)  # Set a 10-second timeout
    print("Heartbeat server started...")

    received_packets = set()  # Store all received sequence numbers
    expected_packets = set(range(1, 11))  # Expected sequence numbers from 1 to 10

    while True:
        try:
            # Wait for incoming data from the client
            data, address = server_socket.recvfrom(1024)
            recv_time = time.time()  # Record the time the packet is received

            # Parse the received message
            message = data.decode().strip().split()
            seq, send_time = int(message[1]), float(message[2])

            # Store the received sequence number
            received_packets.add(seq)

            # Print received packet details
            print(f"Received heartbeat {seq} from {address} with latency: {recv_time - send_time:.6f}
seconds")

        except timeout:
            # After 10 seconds of no packets, check for missing packets
            missing_packets = expected_packets - received_packets

            if missing_packets:
                print(f"No heartbeat detected for 10 seconds. Missing sequences:
{sorted(missing_packets)}")
            else:
                print("All packets received successfully.")

            break  # Stop the server loop after timeout

        except KeyboardInterrupt:
            # Handle server interruption
            print("Server interrupted. Shutting down.")
            server_socket.close()
            sys.exit()

if __name__ == '__main__':
    # Start the server on port 12000
    serve(12000)
```

Heartbeat_client.py:

```
from socket import *
import time
import random

def heartbeat(host, port, num_packets):
    client_socket = socket(AF_INET, SOCK_DGRAM)  # Create UDP socket
    packets = [(seq, time.time()) for seq in range(1, num_packets + 1)]  # Create packet list

    # Simulate sending 60% of packets out of order
    out_of_order_count = int(0.6 * num_packets)  # Calculate 60% of total packets
    shuffled_packets = packets[:out_of_order_count]  # First 60% of packets
    remaining_packets = packets[out_of_order_count:]  # Remaining packets

    # Shuffle the first 60% to simulate out-of-order delivery
    random.shuffle(shuffled_packets)

    # Combine shuffled and remaining packets
    packets = shuffled_packets + remaining_packets

    try:
        for seq, send_time in packets:
            # Simulate 40% chance of dropping the packet
            if random.random() < 0.4:
                print(f"Dropped heartbeat {seq} (simulated packet loss).")
                continue  # Skip sending this packet

            # Prepare the message
            message = f"Heartbeat {seq} {send_time}"
            client_socket.sendto(message.encode(), (host, port))
            print(f"Sent heartbeat {seq} at {send_time:.6f}")

            # Wait 1 second before sending the next packet
            time.sleep(1)

    except KeyboardInterrupt:
        print("Client interrupted. Stopping heartbeats.")

    finally:
        # Close the socket after sending all packets
        client_socket.close()
        print("Heartbeat client stopped.")

if __name__ == '__main__':
    host = '127.0.0.1'  # Server IP address
    port = 12000  # Server port
    num_packets = 10  # Total number of packets to send

    # Start the heartbeat client
    heartbeat(host, port, num_packets)
```