CN REPORT

Ketan Mohan Garg (2022248)
Keshav Bindlish (202246)

PART - 1 (UDP PINGER)

INTRODUCTION

In this we have designed a ping application using UDP protocol. As per the protocol specified, the server does not have any connection with the client .In it, the client is generally checking for the server's availability by sending messages to the server. Client sent a total of ten ping messages to the server, and the server responses by messaging back.
The client calculates Round Trip time for each response by the server. Client also generates statistics i.e. maximum round trip time, minimum round trip time, maximum round trip time, packets received, packets lost and the percentage of lost packets.
When a packet comes to the server and if the randomised integer is greater than or equal to 4, the server sends the data back.

ASSUMPTIONS

1. **Fixed Percentage of Packets Lost** - The system is made such that 30% of the total packets sent by the clients must be lost.
2. **TIMEOUT -** Client assumes that if the server does not respond within 1 second then the packet is lost.

Server and Client Working:
Server - The server checks for the arriving pings from the client. As the ping arrives it measures the delay.
Client - The client periodically sends a time-stamped ping message to a server through UDP. It waits for a response from the server and calculates the round-trip time (RTT) and If no response is received from the server within the timeout period that is 1 second, the client assumes that the packet has been lost.

CLIENT SIDE CODE EXPLAINED:

```
import socket, time
server_address = ('127.0.0.1', 12000)
```

Both of the two modules are imported for using time related functions to record the current time, and to make network sockets.

```
TIMEOUT = 1
Pingstosent = 10
```

Timeout is set out to be one second as mentioned above such that if server does not reply to client within 1 second, client assumes that packet is lost. Variable Pingstosent is set to 10.

```
def stats(client_socket,Pingstosent,server_address):
    rtts = []
    lostpackets= 0
    for ping_no in range(1, Pingstosent + 1):
       print(f"Sending ping {ping_no}..")
       sending_time = time.time()
       message = f"Ping {ping_no} {time.time()}"
       try:
        client_socket.sendto(message.encode(), server_address)
        response, _ = client_socket.recvfrom(1024)
        rtt = time.time() - sending_time
        rtts.append(rtt)
        print(f"Reply from server: {response.decode()} RTT = {rtt:.4f}  seconds")
       except socket.timeout:
        print("Request timed out")
        lostpackets+=1

    client_socket.close()
    return rtts,lostpackets
```

Parameter passed to the function ***stats***:
1. Socket for the client
2. No. of pings to be sent
3. Address of the server

rtts is the array used to store the round trip time of each packet or ping sent.
Variable ***lostpackets*** keep the counter of the packets which did not return back to the client after reaching the server.

For Loop: (ping no. - 1 to 10):

1. Stored the sending time of the packet in **Var(sending_time)** using library(time) - ***time.time()***.
2. Message is packed with *ping no.* and with the *current time.*
3. ***Try / except:*** It is established in case of timeout and no response from server is recieved for the packet sent.
4. ***.sendto() function:*** Packet is sent to the server
5. ***.recvfrom() function:*** to receive the response/ packet sent back to client from server
6. Variable ***response*** stores the message sent by the server.
7. Finally the **round trip time(rtt) is calculated** using the *sending_time* and the current time when the response is received from the server.
8. The reply from the server and the round trip time are the **outputs at the terminal**

```
def main():
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    client_socket.settimeout(TIMEOUT)
    rtts,lostpackets = stats(client_socket,Pingstosent,server_address)
    if rtts:
        print(f"\n--Ping statistics--")
        print(f"Packets sent: {Pingstosent}",f"Packets received: {len(rtts)}",f"Packets
lost: {lostpackets}",f"Percentage of Packet loss: {lostpackets * 10}%")
        print(f"RTT min/avg/max = {min(rtts):.4f}/{sum(rtts) /
len(rtts):.4f}/{max(rtts):.4f} seconds")
    else:
        print("No packets were sent.")
```

Main function:
Creates a ***client_socket*** and sets its timeout to be 1.
Returned function stats: ***rtts*** and *lostpackets* are returned by the, where ***rtts*** is the array of all the RTT of the packets, and ***lostpackets*** is the number of packets lost in the entire process.
If the list returned has some values, then the main function prints the ping statistics i.e. the total number of packets received successfully, number of lost packets and the percentage of lost packets and minimum ***min(rtts)***, average ***sum(rtts)/len(rtts)*** and maximum ***max(rtts)*** RTT of the all packets sent. If the ***rtts*** is empty then it just simply prints No packets were sent.

```
main()
```
At the end the main function is called.

PART 2 - UDP HEARTBEAT

Introduction -

In this we have remodified the UDP pinger to design UDP Heartbeat, The Heartbeat can be used to check if an application server is up and running or not, The client sends a sequence number and current timestamp in the UDP packet to the server. Server calculates the time difference and replies back to the client, and if the server response to the client is missed for several number of times then client automatically assumes that server application has stopped.

Assumptions-

1. **No. of Consecutive Miss fixed** - If server does not respond to the client heartbeat packets consecutively for ***three*** times then client assumes that server application had stopped.
2. **Fixed Percentage of Packets Lost** - The system is made such that 30% of the total packets sent by the clients must be lost.

Client Code for UDPHeartbeat-

```
def stats(client_socket,server_address):
    rtts = []
    lostpackets= 0
    ping_no=1
    consec_packets=0
    while (True):
       print(f"Sending ping {ping_no}..")
       sending_time = time.time()
       message = f"{time.time()}"
       try:
        client_socket.sendto(message.encode(), server_address)
        response, address = client_socket.recvfrom(1024)
        rtt = time.time() - sending_time
        rtts.append(rtt)
        consec_packets=0
        print(f"RTT = {rtt:.4f} seconds")
       except socket.timeout:
        print("Request timed out")
        lostpackets+=1
        consec_packets+=1
        if consec_packets>= 3:
           print("Application stopped")
           break
       ping_no+=1
    client_socket.close()
    return ping_no
```

The previous question code was modified for the second part:
Instead of a for loop we used the ***while(True) condition*** to keep on sending the packet to the point till 3 consecutive packets result in time out. A counter was maintained in the ***consec_packets variable*** and as soon as it hit ***the value of 3 the application is stopped*** and no more packets are sent. But the consec_packets is ***set to zero whenever a packet is successfully returned back*** form the server with the message because the sequence of consecutive timeout break due to this.
Message sent to server via the packet stores the time at which the packet was sent to calculate the time it took to reach the server, which is then printed on the terminal of server. ***Ping_no*** keep the count of ***how many packets when sent*** before shutting down of the application.

Server Code for UDPHeartbeat -

```
1   import random, time
2   from socket import *
3   serverSocket = socket(AF_INET, SOCK_DGRAM)
4   serverSocket.bind(('', 12000))
5
6   while True:
7
8       rand = random.randint(0, 10)
9       message, address = serverSocket.recvfrom(1024)
10      message = message.decode()
11      message= float(message)
12      cal_time = time.time() - message
13      print(f"time difference: {cal_time}")
14
15      if rand < 4:
16          continue
17      message = str(message)
18      serverSocket.sendto(message.encode(), address)
```

To designed server for UDP Heartbeat application, we made some changes in the server which are as follows:
The message received by the server is having the sending time for calculating the time difference.
Message is encoded and converted into float as the received message is in string.
Variable ***Cal_time*** stores the time difference for each Heartbeat packet received, and the server also prints the cal_time on the terminal.
After, the message is converted into string and sent back to the client.

NOTE -
After successfully writing our codes, we had run both the server and client on the same machine as well as different machines also.