CS 5700 Programming Project

In this project, you will write socket programming for UDP in Python. You will send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout. This is a group project. You were asked to form new groups (on the course google doc) that you can use for the programming project and seminars.

You will first complete a simple Internet ping server written in Python, and implement a corresponding client. The functionality provided by these programs is similar to the functionality provided by standard ping programs available in modern operating systems. However, these programs use a simpler protocol, UDP, rather than the standard Internet Control Message Protocol (ICMP) to communicate with each other. The ping protocol allows a client machine to send a packet of data to a remote machine, and have the remote machine return the data back to the client unchanged (an action referred to as echoing). Among other uses, the ping protocol allows hosts to determine round-trip times to other machines.

You are given the incomplete code for the Ping server below. Your task is to complete this code and write the Ping client code. In order to do that you can use the help of socket programming for UDP in our textbook. You can also google the use and syntax of some socket or python functions and apply to your program.

Server Code

The following code implements a ping server. You need to complete, compile and run this code **before** running your client program.

In this server code, 30% of the client's packets are simulated to be lost. You should study this code carefully, as it will help you write your ping client.

```
# UDPPingerServer.py
# We will need the following module to generate randomized lost packets import random
from socket import *

# Create a UDP socket
serverSocket = #complete this line
# Assign IP address and port number to socket
serverSocket.bind(('', 1500X))  # binds socket to 'localhost', port 15000+X, where X is your group number

while True:
    # Generate random number in the range of 0 to 10 rand =
    random.randint(0, 10)
    # Receive the client packet along with the address it is coming from message, address = # complete this line
    # Capitalize the message from the client message =
    #complete this line
```

```
# If rand is less than 4, we consider the packet lost and do not respond
if rand < 4:
    continue
# Otherwise, the server responds and sends the message to the client #complete this line
```

The server 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 received data and sends it back to the client.

Packet Loss

UDP provides applications with an unreliable transport service. Messages may get lost in the network due to router queue overflows, faulty hardware or some other reasons. Because packet loss is rare or even non-existent in typical campus networks, the server in this program injects artificial loss to simulate the effects of network packet loss. The server creates a variable randomized integer which determines whether a particular incoming packet is lost or not.

Client Code

You need to implement the following client program.

The client should send 10 pings to the server. Because UDP is an unreliable protocol, a packet sent from the client to the server may be lost in the network, or vice versa. For this reason, the client cannot wait indefinitely for a reply to a ping message. You should get the client wait **up to one second** for a reply; if no reply is received within one second, your client program should assume that the packet was lost during transmission across the network. You will need to **look up the Python documentation to find out how to set the timeout value on a datagram socket.**

During development, you should run the UDPPingerServer.py on your machine **first**, and test your client by sending packets to *localhost* (or, 127.0.0.1).
After you have fully debugged your code, you should see how your application communicates across the network with the ping server and ping client running on different machines.

Specifically, your client program should

(1) send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)
(2) print the response message from server, if any
(3) calculate and print the round trip time (RTT), in seconds, of each packet, if server responds
(4) otherwise, print "Request timed out"
(5) calculate and print the minimum, maximum, and average RTTs at the end of all pings from the client.
(6) calculate and print the packet loss rate (in percentage).
(7) adjust your client and server codes and run them on two different hosts (at different networks). For example, the client code can be running on Richard's laptop while he is at home, and the server code can be running at Lama's laptop while she is at home. Then repeat points 1 and 6 for the new settings.

Message Format

The ping messages in this project are formatted in a simple way. The client message is one line, consisting of ASCII characters in the following format:

Ping *sequence_number time*

where *sequence_number* starts at 1 and progresses to 10 for each successive ping message sent by the client, and *time* is the time when the client sends the message.

What to Hand in

You will hand in the following:

1) The complete client code and server code as python files (*.py) (no screenshots please for your code)

2) Screenshots at the client verifying that your ping program works as required. The screenshots should clearly show the values printed by your program.

3) Screenshots at the client verifying that your ping program works as required after running the client code and server code on different hosts on different networks.

4) A brief report, explaining your code and the choices you made, and the answers to points 3, 5, and 6 that were printed to your screen.

How to run your code

Open 2 prompt windows from the Python platform you usually use, in order to run your code and receive your messages on it.

You can google how to do that for the particular Python platform you use.

For example, if you are using Anaconda3 to open your python program, you can go to start menu on your windows laptop and type “Anaconda prompt”.

After changing your directory to the folder containing your code, you can run your programs, by typing: **python your_filename.py**

Note that you need to use two prompt windows. One is used to run the server code and another to run your client code.