Programming Assignment 2

CST 311, Introduction to Computer Networks, Spring 2020

READ INSTRUCTIONS CAREFULLY BEFORE YOU START THE ASSIGNMENT.

This programming assignment is due on Wednesday, February 4, 2020.

Assignment must be submitted electronically to iLearn on https://ilearn.csumb.edu by 11:55 p.m. on the due date. Late assignments will not be accepted.

This assignment is to be done with your Team per the Programming Process document with the steps below to develop and write the client part of a client server application. The naming convention of the file should be PA2_your_last_names_in_alphabetic_order.py. **Put your names in the program as well.** Your client program must work with the server program given to you below. Your program must have sufficient comments to clearly explain how your code works.

This assignment is worth 175 points. The grading objectives for the assignment are given below.

UDP Pinger

In this assignment, you will learn the basics of socket programming for UDP in Python. You will learn how to send and receive datagram packets using UDP sockets and also, how to set a proper socket timeout. Throughout the assignment, you will gain familiarity with a Ping application and its usefulness in computing statistics such as packet loss rate.

You will first study 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 complete code for the Ping server below. Your task is to write the Ping client.

Server Code

The following code fully implements a ping server. You need to compile and run this code before running your client program. *You do not need to modify this code.*

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

# Notice the use of SOCK_DGRAM for UDP packets

serverSocket = socket(AF_INET, SOCK_DGRAM)

# Assign IP address and port number to socket

serverSocket.bind(('', 12000))


while True:

        # Generate random number in the range of 0 to 10

        rand = random.randint(0, 10)

        print (“random number is “, rand)

        # Receive the client packet along with the address it is coming from

        message, address = serverSocket.recvfrom(1024)

        # Capitalize the message from the client

        message = message.upper()

        # If rand is less is than 4, we consider the packet lost and do not
respond

        if rand < 4:

                print (“Packet is lost “)

                continue

        # Otherwise, the server responds

        serverSocket.sendto(message, address)
```

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 encapsulated 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 assignment 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. See the Message Format below on the message to be sent to the Server for each ping. Also print the message that is sent 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 to 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. If the packet is lost, print “Request timed out”.

The program must calculate the round-trip time for each packet and print it out individually. Have the client print out the information similar to the output from doing a normal ping command – see sample output below. Your client software will need to determine and print out the minimum, maximum, and average RTTs at the end of all pings from the client along with printing out the number of packets lost and the packet loss rate (in percentage). Then compute and print the estimated RTT, the DevRTT and the Timeout interval based on the RTT results.

Here is a sample output from a ping of Google 4 times:

```
Pinging google.com [216.58.195.78] with 32 bytes of data:
Reply from 216.58.195.78: bytes=32 time=13ms TTL=54
Reply from 216.58.195.78: bytes=32 time=14ms TTL=54
Reply from 216.58.195.78: bytes=32 time=16ms TTL=54
Reply from 216.58.195.78: bytes=32 time=13ms TTL=54
Ping statistics for 216.58.195.78:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
```

Minimum = 13ms, Maximum = 16ms, Average = 14ms

Note – your Team will earn 3% if your Team can write the client code to do the assignment with the calculations without the use of a list (or array) and for other efficient and effective use of storage and program speed.

Grading Objectives

On the Server side

(1) (10 %) the Server must change the received message to upper case (5%). Server must display a message "Packet has been lost", if a packet is "lost" based on the requirement that 30% of the client's packets are simulated to be lost (5%)

Specifically, your client program should

(2) (10%) Send the ping message using UDP (Note: Unlike TCP, you do not need to establish a connection first, since UDP is a connectionless protocol.)

(3) (10 %) Print the response message from server.

(4) (10 %) Calculate and print the round trip time (RTT), in seconds, of each packet, if server responds.

(5) (7 %) If the packet times out on the client side and is considered lost, print "Request timed out".

(6) (10 %) Determine and print minimum RTT (3%) and maximum RTT (3%). Calculate and print average RTT (4%)

(7) (10 %) Calculate and print packet loss percentage.

(8) (10 %) Calculate and print the estimated RTT. Consider alpha = 0.125.

(9) (10 %) Calculate and print DevRTT. Consider beta = 0.25. Calculate and print Timeout interval.

(10) (10 %) Turned in pseudo code from all members (2%), decision on best approach from pseudo code (2%), network steps (1%), code review (2%), test plan (2%) and listing of well documented client code (1%).

(11) (3%) Efficient and effective code – program and calculations done without needing to use a list (or array)

During development, you should run the UDPPingerServer.py on your machine, 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.

Message Format

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

This is 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.

Note:

Print the client message that is sent to the server.
Print the time you receive the response from the server.

The Process

1. Each student reads the assignment.
2. Team meets to select the Team Lead who works with the team to assign roles, discusses the assignment together for a common understanding of the problem and what is needed and then sets up a schedule.
3. Everyone writes their own pseudo code* (will be turned in)
4. On date per schedule - Team leader holds meeting for each member to go over their pseudo code and team selects the best approach (network solution, efficiency, effective, concise, robust, etc)
5. Per schedule, Server code is entered and client side is coded with documentation comments on both and then reviewed by the team.
6. Team leader works with team on the plan for testing. No test data is needed on this assignment.
7. Team works on test and debugging. As part of the test plan, do first on one machine, then when working, have the ping server and ping client running on different machines.
8. TA may check in with Team Lead on assignment progress.
9. Complete assignment before deadline.

What to Hand in

Team leader will work with the team to prepare the following to turn in:

1. Pseudo code from each member
2. Basis of decision on best approach (pseudo code)
3. Network steps needed (example addresses, sockets)
4. Interesting findings from code review discussion
5. Test plan
6. Test results – do calculations on RTT, minimum, maximum, average, loss count, loss %, estimated RTT, DevRTT and Timeout interval for Grading Objectives #4 through #9 manually and compare with the computer calculations. Note – you need to show only the calculations on the numbers from running the ping client and ping server on different machines.
7. Show the final code - ping server and ping client that is well documented (Grade Objective #10)

Optional Extra-credit Exercises

(10 points) Another similar application to the UDP Ping would be the UDP Heartbeat. The Heartbeat can be used to check if an application is up and running and to report one-way packet loss. The client sends a sequence number and current timestamp in the UDP packet to the server, which is listening for the Heartbeat (i.e., the UDP packets) of the client. Upon receiving the packets, the server calculates the time difference and reports any lost packets. If the Heartbeat packets are missing for some specified period of time, we can assume that the client application has stopped.
Implement the UDP Heartbeat (both client and server). You will need to modify the given UDPPingerServer.py, and your UDP ping client.