PART-1: UDP Pinger

In this assignment, you will first study a simple Internet ping server written in Python and implement a corresponding client. The functionality provided by these programs is like 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 in uppercase (an action referred to echoing loud!). 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.

UDP 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, ~33% 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 a random number between 0 to 11 (both included)
rand = random.randint(0, 11)
# 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:
   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 (at the application layer) 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.

UDP Client Code (UDPPingerClient.py)

You need to implement the following client program.

The client should send N number of pings to the server, where N is a configurable parameter that your client should accept from the keyboard. 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 and handle exceptions. Exception handling in Python allows you to write more robust and fault-tolerant code by gracefully managing errors and timeouts.

Specifically, your client program should

(1) send the ping messages 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 responses.
(4) otherwise, print "Request timed out"
(5) report the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate and report the packet loss rate (in percentage).

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 containers of the VM assigned to you.

Message Format

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

ping *sequence_number time-stamp*

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

Modified UDP Server Code (`UDPPingerModifiedServer.py`)

The service code given above in this assignment (`UDPPingerServer.py`) was simulating packet loss at the application layer using *randint( )* function. The more realistic way is to emulate packet loss at the network interface card (NIC) level by using tc (traffic control) netem utility in Linux. Check references at the end of this assignment for some tutorials on tc-netem and inject 33% packet loss on the NIC associated with the container where the server code is being executed. Since you are emulating losses at the NIC level, your server program no longer requires any code to simulate packet losses using *randint( )* function and therefore modify it accordingly.

PART-2: TCP Pinger

This part is same as that of PART-1, except that you need to use TCP sockets to write the ping client (`TCPPingerClient.py`), ping server (`TCPPingerServer.py`) and the modified ping server (`TCPPingerModifiedServer.py`) programs. For extra credit, rewrite your TCP server to be a concurrent server (`TCPPingerConcurrentServer.py`)- that is a server that waits on the welcoming socket and then creates a new thread or process to handle the incoming ping messages from different clients.

Programming notes

Here are a few tips/thoughts to help you with the assignment:

- You must choose a server port number greater than 1023 (to be safe, choose a server port number larger than 5000). If you want to explicitly choose you client-side port, also choose a number larger than 5000.
- You need to know your machine's IP address, when one process connects to another. You can also use the nslookup command in Linux. On Windows, see the ipconfig utility. On a Mac, you can run the terminal program and use the ifconfig commend (just type in ifconfig or ifconfig | grep "inet ").
- Many of you will be running the clients and senders on the same machine at the time of initial testing and debugging. This is fine; since you're using sockets for communication these processes can run on the same machine or different machines. Recall the use of the ampersand to start a process in the background. If you need to kill a process after you have started it, you can use the Linux kill command. Use the ps command to find the process id of your client/server.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error.

Programming the assignment in C

If you choose to write your programs in C, you'll need to master the C system calls needed for socket programming. Perosnally, C is my favorite language for network programming since it gets you closest to the operating system's socket-related system calls. These include *socket(), bind(), listen(), accept(), connect(),* and *close()*. For a quick text-only tutorial of socket programming specifically under Linux, see http://www.lowtek.com/sockets/. Here's another nice tutorial: http://beej.us/guide/bgnet/

For creating concurrent server in C, refer a tutorial on pthreads at this link and a tutorial on fork() system call at this link.

On usage of LLMs for completing this assignment

You may complete this assignment with the aid of large language models (LLMs) such as ChatGPT, Bard, etc. The goal is to help you (i) develop a solid understanding of the course materials, and (ii) gain some experience in using LLMs for problem solving. Note that LLMs may not be a reliable as you may see, GPT-like models can generate incorrect and contradicting answers. It is therefore important that you have a good grasp of the lecture materials, so that you can evaluate the correctness of the model output, and also prompt the model toward the correct solution. If you used any LLMs, you must include the program trace (i.e., screenshots of your interaction with the model) in the report as part of the submission; the model output does not need to be the correct answer, but you should be able to verify the code or find the mistake, if any. Submit the code generated by LLMs by adding prefix LLM to the respective client and server programs. Make sure you cite the model properly in the source file, that is, include the model name, version (date), and url if applicable. If the source code generated by the model is correct and you want to rely on it for submission, you then rewrite the code in your own way (i.e., like reading the answer/solution from a guidebook and writing it in your own unique way in the exam) by following the coding style guidelines. Note that you do not lose any marks for the usage of LLMs. Most of the weightage is given for viva or live-test in the lab, so you should really understand the code given by LLMs so as to solve other problems given in the lab at the time of viva.

What to Hand in?

Deliverables in a tar ball on GC

You will hand in the complete source files (3 files each for PART-1 and PART-2 by strictly following the naming conventions for the programs and optionally concurrent TCP server for extra credit) and a short report with screenshots at the client and at the server verifying that your ping programs work as required in case of UDP and TCP sockets by a tar ball (refer man page of tar) with filename as <Prg-Asg1-RollNo>.tar. You should program your client and server to each print an informative statement whenever it takes an action (e.g., sends or receives a message, detects termination of input, dropping a packet, timeout, etc.), so that one can see that your processes are working correctly (or not!). This also allows the TA to also determine from this output if your processes are working correctly. You should hand in screen shots (or file content if your process is writing the output to a file instead of

console/terminal) of these informative messages as well as the required output of the client (avg. RTT and loss rate).

During evaluation, you will be asked to setup several client processes on different containers/machines and show the outputs by varying loss rates. You may also be asked to modify the logic of client and server for different usecases (e.g., telephone directory service by the server)

25% of marks are allotted for documentation (code comments, inline explanations, function descriptions, README file, etc) and the report. Students should aim to make their code and explanations easy to understand for TA. Refer the following page for style guides of C/C++/Python/Java and adhere to these guidelines while coding so as not to lose the marks earmarked for documentation: https://google.github.io/styleguide/ and https://peps.python.org/pep-0008/

Marking Scheme:

- 25% for documentation in source files and the readable report. Note that 10% of the assignment marks may be removed for lack of neatness in the submitted files and the report.
    - If you used any LLMs, you must include the program trace (i.e., screenshots of your interaction with the model) in the report as part of the submission.
- 25% for the submission
- 50% for viva and live-test
- 10% extra for concurrent server
- 10% extra for coding in C/C++ instead of Python/Java

ANTI-PLAGIARISM Statement <Include it in your report>

I certify that this assignment/report is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, packages, datasets, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this assignment/report has not previously been submitted for assessment/project in any other course lab, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that I have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. Additionally, I acknowledge that I may have used AI tools, such as language models (e.g., ChatGPT, Bard), for assistance in generating and refining my assignment, and I have made all reasonable efforts to ensure that such usage complies with the academic integrity policies set for the course. I pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, I understand my responsibility to report honour violations by other students if I become aware of it.

Name <Roll No>:
Date:
Signature: <keep your initials here>

References:

1. A Python socket tutorial is http://docs.python.org/howto/sockets.html

2. https://man7.org/linux/man-pages/man8/tc-netem.8.html
3. https://srtlab.github.io/srt-cookbook/how-to-articles/using-netem-to-emulate-networks.html
4. https://www.cs.unm.edu/~crandall/netsfall13/TCtutorial.pdf
5. https://realpython.com/intro-to-python-threading/
6. https://www.tutorialspoint.com/python/python_multithreading.htm
7. https://docs.python.org/3/library/concurrency.html