ACN Programming-Assignment-1-My Pingers

PART-1: UDP Pinger

In this **group (of size 2 or 3 students) 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 and check reachability and estimate packet losses. 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, ~20% 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(('', 11000))

while True:
# Generate a random number between 1 to 10 (both inclusive)
rand = random.randint(1, 10)
# 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 greater than 8, we consider the packet lost and do not respond
to the client
if rand > 8:
   continue
```

```
# Otherwise, the server response
serverSocket.sendto(message, address)
```

The server sits in an infinite loop listening for incoming UDP packets from one or more clients. When a packet comes in and if a randomized integer is less than or equal to 8, 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 packet loss over the network. 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 to wait up to one second or so 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 milli-seconds, of each packet if the server responds along with the sequence_number of the packet.

(4) otherwise, print "Request timed out for the packet #"

(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 ping application communicates across the network with the ping server and ping client running on different machines i.e., your machine and your partner's machine. Also check and comment on the behavior of this application when multiple clients simultaneously ping the server and make suitable modifications to the client/server code, if any, to ensure the application works smoothly even when one of the client(s) is killed in the middle of sending N pings to the server.

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 20% packet loss on the NIC associated with the machine 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. Make sure that the modified TCP ping server is a concurrent server- 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 10000). If you want to explicitly choose your 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.
- 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 to bind a new socket to the port ID you previously used (but never closed), you may get an error.

PART-3: ICMP Pinger

Your task in this part of the assignment is to develop your own ping client (`TCPPingerClient.py`). Your application will use ICMP instead of UDP/TCP but, in order to keep it simple, will not exactly follow the official specification in RFC 1739. Note that you will only need to write the client side of the program, as the functionality needed on the server side is built into almost all operating systems.

Code

Below you will find the skeleton code for the ping client using ICMP. You are to complete the skeleton code. The places where you need to fill in code are marked with **#Fill in start** and **#Fill in end**. Each place may require one or more lines of code.

Additional Notes

1. In "receiveOnePing" method, you need to receive the structure ICMP_ECHO_REPLY and fetch the information you need, such as checksum, sequence number, time to live (TTL), etc. Study the "sendOnePing" method before trying to complete the "receiveOnePing" method.
2. You do not need to be concerned about the checksum, as it is already given in the code.
3. This part of the assignment requires the use of **raw sockets**. In some operating systems, you may need administrator/root privileges to be able to run your pinger program.
4. See the end of this programming exercise for more information on ICMP.

Skeleton Python Code for the ICMP Pinger

```
from socket import *
import os
import sys
import struct
import time
import select
import binascii

ICMP_ECHO_REQUEST = 8

def checksum(string):
    csum = 0
    countTo = (len(string) // 2) * 2
    count = 0

    while count < countTo:
        thisVal = string[count+1] * 256 + string[count]
        csum = csum + thisVal
        csum = csum & 0xffffffff
        count = count + 2

    if countTo < len(string):
        csum = csum + string[len(string) - 1]
        csum = csum & 0xffffffff

    csum = (csum >> 16) + (csum & 0xffff)
    csum = csum + (csum >> 16)
    answer = ~csum
    answer = answer & 0xffff
    answer = answer >> 8 | (answer << 8 & 0xff00)

    return answer

def receiveOnePing(mySocket, ID, timeout, destAddr):
    timeLeft = timeout
    while 1:
        startedSelect = time.time()
        whatReady = select.select([mySocket], [], [], timeLeft)
        howLongInSelect = (time.time() - startedSelect)
        if whatReady[0] == []:  # Timeout
            return "Request timed out."

        timeReceived = time.time()
        recPacket, addr = mySocket.recvfrom(1024)

        # Fill in start
        # Fetch the ICMP header from the IP packet
        # Fill in end

        timeLeft = timeLeft - howLongInSelect
        if timeLeft <= 0:
            return "Request timed out."

def sendOnePing(mySocket, destAddr, ID):
    # Header is type (8), code (8), checksum (16), id (16), sequence (16)
    myChecksum = 0

    # Make a dummy header with a 0 checksum
    # struct -- Interpret strings as packed binary data
    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)
    data = struct.pack("d", time.time())

    # Calculate the checksum on the data and the dummy header.
    myChecksum = checksum(header + data)

    # Get the right checksum, and put it in the header
    if sys.platform == 'darwin':
        # Convert 16-bit integers from host to network byte order
        myChecksum = htons(myChecksum) & 0xffff
    else:
        myChecksum = htons(myChecksum)

    header = struct.pack("bbHHh", ICMP_ECHO_REQUEST, 0, myChecksum, ID, 1)
    packet = header + data

    mySocket.sendto(packet, (destAddr, 1))  # AF_INET address must be tuple, not str
     # Both LISTS and TUPLES consist of a number of objects which can be referenced
by their
    # position number within the object.


def doOnePing(destAddr, timeout):
    icmp = getprotobyname("icmp")
    # SOCK_RAW is a powerful socket type. For more details:
    # http://sockraw.org/papers/sock_raw
    mySocket = socket(AF_INET, SOCK_RAW, icmp)

    myID = os.getpid() & 0xFFFF  # Return the current process ID
    sendOnePing(mySocket, destAddr, myID)
    delay = receiveOnePing(mySocket, myID, timeout, destAddr)

    mySocket.close()
    return delay

def ping(host, timeout=1):
    # timeout=1 means: If one second goes by without a reply from the server,
    # the client assumes that either the client's ping or the server's pong is lost
    dest = gethostbyname(host)
    print("Pinging " + dest + " using Python:")
    print("")

    # Send ping requests to a server separated by approximately one second
    while 1:
        delay = doOnePing(dest, timeout)
        print(delay)
        time.sleep(1)  # one second
    return delay
```

Notes:

1. Currently, the icmp client pinger program calculates RTT for each packet and prints it out individually. Modify this to correspond to the way the standard ping program works. You will need to report the minimum, maximum, and average RTTs at the end of all pings from the client. In addition, calculate the packet loss rate (in percentage) like in PART-1 and PART-2.
2. Your program can only detect timeouts in receiving ICMP echo responses. Modify the Pinger program to parse the ICMP response error codes and display the corresponding error results to the user. Examples of ICMP response error codes are 0: Destination Network Unreachable, 1: Destination Host Unreachable. Note that these modifications can be added to PART-1 and PART-2 of the assignment even though you used UDP/TCP sockets for sending pings.

Internet Control Message Protocol (ICMP)

ICMP Header

The ICMP header starts after bit 160 of the IP header (unless IP options are used).

Bits	**160-167**	**168-175**	**176-183**	**184-191**
160	Type	Code	Checksum	
192	ID		Sequence	

- **Type** - ICMP type.
- **Code** - Subtype to the given ICMP type.
- **Checksum** - Error checking data calculated from the ICMP header + data, with value 0 for this field.
- **ID** - An ID value, should be returned in the case of echo reply.
- **Sequence** - A sequence value, should be returned in the case of echo reply.

Echo Request

The echo request is an ICMP message whose data is expected to be received back in an echo reply. The host must respond to all echo requests with an echo reply containing the exact data received in the request message.

- Type must be set to 8.
- Code must be set to 0.
- The Identifier and Sequence Number can be used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every ping process, and sequence number is an increasing number within that process. Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.
- The data received by the echo request must be entirely included in the echo reply.

Echo Reply

The echo reply is an ICMP message generated in response to an echo request, and is mandatory for all hosts and routers.

- Type and code must be set to 0.
- The identifier and sequence number can be used by the client to determine which echo requests are associated with the echo replies.
- The data received in the echo request must be entirely included in the echo reply.

Testing ICMP Pinger error response

For testing ICMP error messages using the icmp pinger, follow these instructions:

1. **Setup**: Identify the destination machine (one of your teammates' machines) and use it as the target for testing. The source machine is where you will execute the icmp pinger.
2. **Configure iptables rules**: On the destination machine, you need to configure iptables to reject ICMP packets from the source machine and specify the error response you wish to test. Use the following command:

```
sudo iptables -A INPUT -s <source_ip_address> -p icmp -j
REJECT --reject-with <error_response>
```

Replace <source_ip_address> with the IP address of the source machine, and <error_response> with one of the following options:

- icmp-host-unreachable
- icmp-port-unreachable

3. **To clear all iptables rules**: Use the following command:

```
sudo iptables -F
```

Note: This procedure is applicable to Linux systems only. Ensure that you have two Linux machines available within your group to perform this test.

For additional details on iptables and its configuration, refer to this iptables manual.

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. Personally, 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 a concurrent server in C, refer to 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, Gemini, 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 as 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 at the time of viva.

What to Hand in?

Deliverables in a tar ball on GC

One of the group members have to upload in the complete source files (3 files each for PART-1 and PART-2 by strictly following the naming conventions for the programs) 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-RollNo1-RollNo2>.tar on the google classroom. 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 set up several client processes and show the outputs by varying loss rates and N. You may also be asked to modify the logic of client and server for different use cases (e.g., telephone directory service by the server or traceroute)

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 & a readable report. Note that 10% of the assignment marks may be removed for lack of neatness in the submitted files & 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

Work Distribution Summary

Each member should fill in their name, the tasks they were responsible for, a brief description of their contribution, and any challenges they faced during the project. **<Add one table per student in the report>**

Task/Section	Group Member <Enter Name of Student>	Contribution <describe the work done, in brief>	Challenges faced (if any).
Research & Info gathering			
Code Development			
Testing & Debugging			
Documentation & Report Writing			
Final Review & Submission			

ANTI-PLAGIARISM Statement <Include it in the report>

We certify that this assignment/report is the result of our collaborative work, based on our collective study and research. All sources, including books, articles, software, datasets, reports, and communications, have been properly acknowledged. This work has not been previously submitted for assessment in any other course unless specific permission was granted by all involved instructors.
We also acknowledge the use of AI tools, such as LLMs (e.g., ChatGPT), for assistance in refining this assignment, if used. We have ensured that their usage complies with the academic integrity policies of this course. We pledge to uphold the principles of honesty, integrity, and responsibility at CSE@IITH. Additionally, we understand our duty to report any violations of academic integrity by others if we become aware of them.

Names <Roll Nos>:
Date:
Signatures: <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