CY350 - Computer Networks Project

#1 – UDP Ping Utility

In this programming assignment you will build a ping client and server that provides functionality similar to that provided by standard ping programs available in your operating system. The caveat is that this program will use the simpler User Datagram Protocol (UDP) rather than the standard Internet Control Message Protocol (ICMP). The ping utility allows a client machine to send a packet of data to a remote machine and have the remote machine respond back to the client (an action referred to as echoing). Among other uses, the ping utility is used by hosts to determine round-trip times to other machines.

Project Due Dates:

Part 1a – 2359 05FEB24

Part 1b – 2359 12FEB24

Limitations:

You may only use the ***socket*** module, the ***time*** module, the ***random*** module, and the Python 3 programming language for your client and server UDP ping message programs.

Part 1a – Client:

1. Using the starting file provided, write a client program **in Python 3** using the **sockets module** that sends and receives a ping messages to your server program using UDP (not ICMP). Your client will send 10 of these ping messages. The client send message must conform to the following protocol:

   Message Format: ping *sequence_number time*

   Example message (not including time): ping 4 *time*

   The sequence number is an integer value beginning at 1 and ending at 10. Your time format must be:

   Time Format: *day_of_month hrs:mins:secs abbreviated_month year*

   Client message: ping 4 10 10:43:55 Jan 2023

   The client should **either accept the server's IP address and port number** at the **command line argument**, or the program should **prompt the user to enter the server's IP and port**. The IP address and port number WILL NOT be hard coded in Python file.

2. For each ping sent, you must **calculate the round-trip time** of the ping.

a. The **client socket must timeout** if no response is received after 1 second (you must configure this timeout on the socket you create).

b. **If timeout occurs**, the client program will display a message indicating **no response** was received for that associated ping message.

Example timeout message: ping 4 No Response

c. **If NO timeout occurs**, ping response message is received by client. Verify that the server response is correct:

Example response message: pong 4 10 10:43:57 Jan 2023

NOTE: Server program will change received client program message. **" *ping"* will be changed to *"pong" without changing the counter value, date, nor time***.

d. After the *pong* message is received from the server program, **display the round-trip time (RTT) in milliseconds (do not round)**.

3. After all 10 client ping messages either timeout or are responded to by the server program, display the following information:

a. Number of ping messages sent

b. Number of ping message responses received

c. Success rate (%)

d. Maximum RTT

e. Minimum RTT

f. Average RTT

Example output (RTTs: 5.0, 2.0, 2.0, 2.0, 3.0, 3.0, 1.0):

Number of messages sent: 10
Number of messages responded
to: 7 Success rate: 70%
Max RTT: 5.0 ms
Min RTT: 1.0 ms
Average RTT: 2.571429 ms

NOTE: For minimum, maximum, and average RTTs, Do NOT include timeouts. For average RTT.

Part 1a – Testing:

We have provided two functioning servers at the addresses below that you can use to test if your client is functioning correctly. You must be on EECSNet to reach these servers.

10.3.50.104 (Ports 12000, 12001, 12002, 12003)

10.3.50.105 (Ports 12000, 12001, 12002, 12003)

- Read through the instructions above very carefully and make sure that every single element has been addressed.

Part 1a – Hints:

- **See sample output for client and server (server output is optional except for some indication that the server is running.)**
- **If you are testing your client/server on the Windows operating system, you may get 0.000 ms for an RTT. That is fine since on Ubuntu, you "should" not see 0.0.0.0 ms response times.**
- **The Part 1 listener servers have been configured to tell you if your message format or date format is incorrect, but not what is wrong beyond that. Please carefully read the instructions above if you are getting an invalid format response from the server.**
- **The provided code file contains many comments and TODO lines for you in addition to some existing code. It is recommended that you read through all the comments and understand the general concept of what needs to happen before you begin coding. Add your own comments as needed while you think about how to flesh out what we have given you. Don't just start coding.**
- **Part of Project 1 is learning how to break up the big problems into smaller subproblems. We have given you TODOs that should help with this, but you may feel free to adjust these however you wish so long as you achieve the specified outcomes. Test each part to make sure it is working before you move onto the next. An example of this might look something like:**
    - **Get the necessary user input.**
    - **Establish a connection to the server.**
    - **Successfully send and receive a message.**
    - **Successfully format and send a single message in the specified manner.**
    - **Send 10 correctly formatted messages.**
    - **Capture the data from all 10 messages.**
    - **Calculate the specified information and print it out for the user.**

Part 1a – Submission:

You will electronically submit the code for your ping client to the Canvas assignment **Project 1a - Develop Ping Tool - Client Side**. The file should be named **lastname_firstinitial_pr1_client.py** Use in-line documentation (review the CY300 lesson related to adding comments to a Python file or ask for guidance) in your code. Additionally, you must complete an e-Acknowledgement statement in CIS.

Part 1b – Server:

1. Write a server program **in Python 3** using the **sockets module** that receives the client program's ping message, processes it, and then sends it back to the client program. Your server will process one message at a time (since the client will only send one message at a time).

   a. When the server receives a message, there should be **approximately** a 30% chance that the message is not sent back to the client. Basically, the server goes back to listening if the random chance hits about 30% when receiving the message. This will simulate packet loss somewhere in the network.

   NOTE: Think about the Python *random* module. 3 out of 10; 30 out of 100; 300 out of 1000; 3000 out of 10,000.

   b. The server will change the "ping" portion of the received message to "pong" and will NOT change anything else of the original message. The "pong" message with then be sent back to the client for processing of the round-trip time (RTT). Example message received by server from client and the server response are as follows:

      - server **receives** message: **ping 4 10 10:43:57 Jan 2023**
      - server **sends** message: **pong 4 10 10:43:57 Jan 2023**

   NOTE: The ping counter value, date, and time information is NOT changed.

Server HINT: It is not a requirement that the server display anything, however, I suggest that you display a message indicating that the server program is running. It may also help if you display received and sent messages. This will help with confirming you are receiving, processing, and sending the message formatted correctly.

Part 1b – Testing:

Assuming your client is working as specified above, you can test your server by having your client connect to it on the loopback address, 127.0.0.1 (don't hard code this in). If you had issues getting your client to work correctly, we will provide you a working client program that you can use to test after the Part 1 submission has concluded. More information on this will be provided as we get closer to Part 2.

Part 1b – Hints:

- Look at the provided client code and think about how you would modify it to receive a message, modify the message, and send that message back.
- What do you need to do to ensure that there is a 30% chance that a message is dropped by the server? We recommend looking into the **random** module.

Part 1b – Submission:

You will electronically submit the code for your ping client to the Canvas assignment **Project 1b - Develop Ping Tool - Server Side.** The file should be named **lastname_firstinitial_pr1_server.py** Use in-line documentation (review the Cy300 lesson related to adding comments to a Python file or ask for guidance) in your code. Additionally, you must complete an e-Acknowledgement statement in CIS.

- Along with your code, you will submit an analysis, no more than 1 page, describing how you approached the problem, any problems you encountered during implementation, and lessons learned.

Client output (screenshot from MS Visual Code remoted into Ubuntu VM):

```
instructor@cy350-vm:~/Documents/project_1$ python3 client.py
Enter server name: 127.0.0.1
Server Name:  127.0.0.1
Enter server port: 12000
Sever Port:  12000
Pong 1 10 15:20:38 Jan 2024 RTT: 18.6660 ms
Pong 2 10 15:20:38 Jan 2024 RTT: 0.4425 ms
Pong 3 10 15:20:38 Jan 2024 RTT: 0.3083 ms
Pong 4 10 15:20:38 Jan 2024 RTT: 0.2978 ms
Pong 5 10 15:20:38 Jan 2024 RTT: 0.2723 ms
Pong 6 10 15:20:38 Jan 2024 RTT: 0.3047 ms
Pong 7 10 15:20:38 Jan 2024 RTT: 0.2780 ms
No response received for packet 8
Pong 9 10 15:20:39 Jan 2024 RTT: 0.2980 ms
No response received for packet 10
Pings sent: 10
Ping received: 8
Success rate: 80.0%
Minimum round trip time: 0.272274ms
Maximum round trip time: 18.666029ms
Average round trip time: 2.608448ms
```

Server output (screenshot from MS Visual Code remoted into Ubuntu VM)
NOTE: only output that is mandatory is a message indicating the server is ready to receive

```
instructor@cy350-vm:~/Documents/project_1$ python3 server.py
The server is READY!
('127.0.0.1', 48513): Ping 1 10 15:20:38 Jan 2024
REPLIED:  Pong 1 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 2 10 15:20:38 Jan 2024
REPLIED:  Pong 2 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 3 10 15:20:38 Jan 2024
REPLIED:  Pong 3 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 4 10 15:20:38 Jan 2024
REPLIED:  Pong 4 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 5 10 15:20:38 Jan 2024
REPLIED:  Pong 5 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 6 10 15:20:38 Jan 2024
REPLIED:  Pong 6 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 7 10 15:20:38 Jan 2024
REPLIED:  Pong 7 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 8 10 15:20:38 Jan 2024
**DROPPED packet:  Ping 8 10 15:20:38 Jan 2024
('127.0.0.1', 48513): Ping 9 10 15:20:39 Jan 2024
REPLIED:  Pong 9 10 15:20:39 Jan 2024
('127.0.0.1', 48513): Ping 10 10 15:20:39 Jan 2024
**DROPPED packet:  Ping 10 10 15:20:39 Jan 2024
```