“Network Applications and Design” Homework Assignment #5

“UDP Ping” (4+1 points)

Due date: **May 31st (Fri) 2019, 9AM.**
- Submit softcopy on the server.

In this homework assignment, you’ll implement simple “UDP Ping” server and client programs using Python. The client will send ping packets to the server over UDP, and the server will respond to those ping packets. While doing so, the server may (for experiment purposes) delay the response for certain amount of time or drop the packet with some probability. The client will measure the RTT and loss rates for the ping packets that it has sent. The key objectives are to get experience in socket programming, multi-thread/process programming, and understand latency/timeout/loss rate.

Below are the steps and requirements on what you’ll need to do.

Please download two sample programs, udp_ping_server_skel.py, and udp_ping_client_skel.py. These are very simple server-client programs that will serve as your skeleton/template code.

- What they do is very simple;
  - ✓ Server will wait for client connection
  - ✓ Client will create a UDP ping packet and send to the server.
  - ✓ Server will respond by returning the packet back to the client.
  - ✓ Client will print the returned string and exit.
- The server must be running before the client runs.
- Below is an example of how to run the programs and how it would work on our server.

client	server

`nsl2: $ python udp_ping_client_skel.py`
`Client: send "Ping"`
`Client: recv "Ping"`
 	
`nsl2: $ python udp_ping_client_skel.py`
`Client: send "Ping"`
`Client: recv "Ping"`
	`nsl2:$ python udp_ping_server_skel.py`
	`Server: recv "Ping"`
	`Server: reply "Ping"`
	`Server: recv "Ping"`
	`Server: reply "Ping"`

- Make sure that you use your own designated port number for the server program. For the client socket, you can use your designated port number, or simply use null (0) which will let the operating system assign a random port number to your client socket. Server socket must use a fixed and designated port number.

As you can see, what I’ve given you are very simple programs that does not do much. Your task is to modify and extend these programs to add several new functions and features. The filenames of the new programs that you will create are “UDPPingClient.py” and “UDPPingServer.py” (File names must be exact. Otherwise, they will not be graded.). Below are the descriptions of what you need to do;

UDP Ping Server

- Instead of responding immediately upon receiving a ping packet from the client, the server should intentionally ‘***delay***’ the response. That is, the server should send the response after some time delay. The purpose of this ‘delay’ is to simulate the processing times in a network device.
  - ✓ Delay for each packet should be a random value from uniform distribution between [0, 2] seconds (which means average delay of 1 sec). (Hint: consider rand and time libraries)
  - ✓ (optional-1 for fun) Try different kinds of delays and see what happens.

- Considering the behavior of the client and the delay induced by the server, and also considering the fact that there can be more than one client running simultaneously, the server must be able to handle multiple clients/packets in parallel.
  - ✓ For this purpose, consider using ***multi-thread*** technique. (Hint: consider threading library)
  - ✓ You may also do this using ***multi-process*** technique. Choice is up to you.
  - ✓ Regardless of whether you use multi-thread or multi-process, the server must be able to correctly handle multiple clients concurrently with correct RTT measurement, loss/timeout detection, and packet sequence number ordering (RTT/loss/timeout/sequence number described below).
- Furthermore, instead of responding to all ping packets received from the client, the server should '***drop***' some packets with a given/certain probability. This is to simulate 'packet loss' in the network.
  - ✓ Packet loss probability should be 20%. That is, the server drops a packet randomly from a uniform distribution with average of 20%. (Hint: use rand library)
- Server should exit the program gracefully when user enters 'Ctrl-C'.
  - ✓ What I mean by 'gracefully' is that, when the user enters 'Ctrl-C', the program should not show any error messages. Instead, the program should print out "Bye bye~" and exit.
    - ▪ Same for when the user enters 'Ctrl-C' on the client program.

UDP Ping Client

- Client should transmit **'10'** UDP ping packets.
- Each UDP ping packet should contain a 'sequence number' n indicating that it is the nth packet.
  - ✓ Sequence number should start from 0. (0~9)
  - ✓ Upon sending or receiving a ping packet, sequence number should be printed on the screen.
    - ▪ When sending, print format should be "PING <seqno> sent"
- If a ping reply was received successfully for each ping sent, calculate the **round-trip-time (RTT)** and print it out on the screen (Hint: use datetime library)
  - ✓ This time is the duration between when the client sent the ping to the server and when the client received the reply from the server. Unit MUST be milliseconds.
    - ▪ Sent time should be measured immediately *before* sending the ping, and
    - ▪ Receive time should be measured immediately *after* receiving the reply.
  - ✓ print format should be "PING <seqno> reply received from <IP> : RTT = <xx> ms"
  - ✓ Careful: Do not assume that the server and the client are on the same machine.
- After sending each ping packet, client waits for a response for up to 1 second ('***timeout***').
  - ✓ If there is no response even after 1 second 'timeout', the client regards it as 'packet loss' and moves on to the next ping packet.
    - ▪ In this case, print out "PING <seqno> timeout!"
  - ✓ If response is received within timeout period, the client should process the reply and move on to the next ping packet immediately. Client should not wait for timeout in this case.
  - ✓ (Hint: consider socket's settimeout function)
- When done with all 10 ping packets (including waiting for response), the client should print out the overall results before exiting.
  - ✓ Results must include number of ping sent, received, lost, loss ratio, min/max/avg RTT
  - ✓ Make sure to close all sockets used before exiting.
  - ✓ Careful: Average RTT should not include lost packets.
- In the actual 'ping' program provided by your operating system (which is implemented using ICMP instead of UDP), several options are provided for parameter configuration (see 'ping -h'). Your client program should also provide the following options. (Hint: use getopt, sys libraries)
  - ✓ "-c <IP address>"
    - ▪ set IP address of the UDP ping server. If not specified, default should be 127.0.0.1.
  - ✓ "-p <port number>"
    - ▪ set port number of the UDP ping server. If not specified, default should be your personal port number that matches the default port number on the server side.
  - ✓ "-w <timeout(ms)>"
    - ▪ Set the timeout time. If not specified, default should be 1000ms.

- ✓ Example> `$ python UDPPingClient.py -c nsl2.cau.ac.kr -p 5005 -w 1200`
  - ✓ Exceptions should be handled for incorrect/invalid options.
  - ✓ (optional-2 for fun) Feel free to add more interesting options.
- If the artificial '***delay***' used by the server is longer than the '***timeout***' used by the client, ping response may be received out-of-order.
  - ✓ Make sure you print out the correct sequence number, and
  - ✓ Make sure that you calculate the RTT for each packet correctly!! (for each sequence number)
    - ▪ Care must be taken here. How would you do it? Think! (Example below)

client

```
nsl2: $ python UDPPingClient.py
PING 1 sent
PING 2 sent
PING 2 reply received from 127.0.0.1 : RTT = 30 ms
PING 1 reply received from 127.0.0.1 : RTT = 1800 ms
…
```

Other additional requirements:

- You must implement and run using Python 3. Do not use Python 2.x
- Your program must run on our class Linux server at nsl2.cau.ac.kr. (both server and client on nsl2)
- Running your client on nsl5.cau.ac.kr and your server on nsl2.cau.ac.kr should work.
  - ✓ Both should work without ANY code modification.
  - ✓ However, you MUST submit your homework assignment on nsl2 and nsl2 only.
    - ▪ I will collect your submission from nsl2 only.
    - ▪ I will not look at any code in nsl5, nor any code in any other directory other than the designated submission directory.
- Ideally, your client and server programs should work on any computer on the Internet even if the client and the server programs are **on different machines**. This also means that all your programs should work on any operating system such as Windows, Linux, or MacOS.
  - ✓ For example, you might run your server program on our class Linux server, and run your client program on a MacOS laptop. This should work without any modification to your code.
  - ✓ However, due to security reasons, our university (CAU) blocks all ports when coming into the university from outside, except for a few that have explicit permission (e.g. 7722).
    - ▪ Thus, it may not be possible to connect to your server program on our class server (nsl2.cau.ac.kr) from your client program on your own computer at home.
    - ▪ (The reverse might work, and you may try if you're interested)
- Make sure that you use your own designated port number for the server socket.
- **For the client socket, you should either not set the port number, or must use null (0) port number which will let the OS assign a random port number to your socket.**
- Your code must include your name and student ID at the beginning of the code as a comment.
- Your code should be easily readable and include sufficient comments for easy understanding.
- Your code must be properly indented. Bad indentation/formatting will result in score deduction.
- Your code should not include any Korean characters, not even your names. Write your name in English.

What and how to submit

- You must submit softcopy of "UDPPingClient.py", and "UDPPingServer.py" files.
- Here is the instruction on how to submit the softcopy files:
  - ✓ Login to your server account at **nsl2.cau.ac.kr**.
  - ✓ In your home directory, create a directory "submit_net/submit_<student ID>_hw5"
    (ex> "/student/20149999/submit_net/submit_20149999_hw5")
  - ✓ Put your "UDPPingClient.py", and "UDPPingServer.py" files and only those in that directory. Do not put any code that does not work!

Grading criteria:

- You get 4 points
  - ✓ if all your programs work correctly, AND if you meet all above requirements, AND
  - ✓ if your code handles all the exceptional cases that might occur.
- Otherwise, partial deduction may apply.
- You may get optional extra credit of up to 1 point if you do the optional extra credit task.
- No delayed submissions are accepted.
- Copying other student's work will result in negative points.
- Code that does not compile or code that does not run will result in negative points.

[Optional] Extra credit task: (up to 1 point)

- Do the same thing as above also in **C**.
  - ✓ Your file names should be same as Python counterpart except the file extension (.py → .c). Your C programs should compile with gcc. Provide a Makefile if compiling is not obvious.
- If you do this, not only your C client should work with C server, but your Python programs should also work with your C programs. That is, you should be able to mix and match C and Python programs for server and client, in any combination. Do not submit if it does not work. You will get negative points for that.

Final Note: Please think about this.

1. How did you implement timeout?
2. Try different timeout values using the -w option and see how the result changes.
3. How did you handle cases where ping reply packets are received out-of-order?
4. When calculating average RTT, how did you handle the timeout cases?
5. Can you calculate one-way delay from the client to the server? How can you do this?
6. Can we measure throughput using this program?
