Programming Assignment 2: UDP Ping Client and Server
Using UDP sockets, you will write a client and server program that enables the client to determine the round-trip time (RTT) to the server. To determine the RTT delay, the client records the time on sending a ping request to the server, and then records the time on receiving a ping response from the server. The difference in the two times is the RTT.

The ping message contains 2 4-byte integers and must be encoded in network-byte order as follows[1]:

- *4-byte **message type** with an integer value of 1 or 2*
    - *Message type = 1 for a ping request (message from client to server)*
    - *Message type =2 for a ping response (message from server to client)*
- *4-byte* **sequence number** with a unique integer value starting from 1[2]. In the ping response, the server should echo back the client's sequence number.

Both the client and server program should take the following input parameters:

- IP address of server
- IP port of server

The client program will read in the above input parameters, and send 10 ping requests consecutively to the server running at the specified IP address and port, waiting for a response each time. After each response is received, the client calculates and prints the RTT for the message. If no response is received within a certain amount of time (one second)[3], the client notes that the request timed out and then sends the next request up to the maximum. The program output should print out trace information when data is sent and received, and account for error conditions. Trace output must include:

- At start of output, print a message indicating the IP address and port of the server being pinged
- For each ping response received, print RTT along with sequence number of ping message
- For no ping response, print "Message timed out" along with sequence number of the ping message
- After completion, print the following statistics (similar to output of UNIX ping utility);
    - Number of packets sent, received, lost (% loss rate)
    - Min, Max, Average RTT for all acknowledged ping packets

[1] See struct.pack() and struct.unpack() functions to encode integers into a sequence of bytes in network byte order in https://docs.python.org/3/library/struct.html
[2] A message sequence number is a unique identifier for the packet. Start numbering from 1, and increment the number by one for each message sent. For example, the sequence number of the first ping request sent from the client is 1, the second ping request has a sequence number of 2, and so on, up until the last ping request which should have a sequence number of 10.
[3] The socket library provides the ability to set a timeout value on a socket. An exception is raised if the timeout value expires before the socket operation has completed.

The server will read in the input arguments, bind to the specified IP address and port, and wait in an infinite loop to receive ping requests from the client. On receiving a ping request, the server program will randomly[4] decide whether to respond to ping requests to simulate network packet loss. In the case that the server responds, it sends a ping response containing the appropriate message type, and client sequence number. In the case that the server decides not to respond, no ping response is sent, and the server waits for another ping request. To implement random "loss", the server should generate a random integer between 0 and 10 and if the result is < 4, do not respond to the packet.

In a ping request, the application data has the following format:

```
 0              1             2                3               4     (bytes)
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |              Message Type (1= request)                       |
 ------------------------------------------------------------------
 |                Message sequence number (e.g. 5)              |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

In a ping response, the application data has the following format:

```
 0              1             2                3               4     (bytes)
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |              Message Type (2 = response)                     |
 ------------------------------------------------------------------
 |                Message sequence number    (e.g. 5)           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Examples of the client program trace output are as follows:

```
Pinging  127.0.0.1, 12000:
Ping message number 1 timed out
Ping message number 2 RTT: 0.002154 secs
….
Ping message number 10 RTT: 0.000194 secs
```

Examples of the server program trace output are as follows:

```
The server is ready to receive on port:  12000
Message with sequence number 1 dropped
Responding to ping request with sequence number 2
…
Responding to ping request with sequence number 10
```

[4] A programming language will typically provide a random number generator in a "random" standard library. For example, in Python, this can be found using "import random" and "random.int()" function.

Notes:

1. **Timestamp Resolution:**

When testing your program, especially when running client and server on the same host, you might find little difference in the timestamp values. The values will depend on the test environment, and timer resolution on your machine. If there are issues, you should simulate longer delays, by adding in a random "wait" or "sleep" function on the server before sending a ping response on receipt of a message.

2. **Data structure Alignment and Padding**

Compilers on modern processors will typically try to align data structures to that optimal for the machine. For example, 4-byte values are stored at addresses divisible by 4, and 8-byte values are stored at addresses divisible by 8. When sending multiple values of different sizes, you may find that in data structures, such as “struct” in C/C++, the compiler adds padding. For example, if you are sending values of different sizes using a C struct to store a 4-byte integer, followed by an 8-byte integer, 4-bytes of padding may be inserted between the 2 integers to ensure 64-bit alignment. Thus, a struct of 12 bytes may become a struct of 16 bytes with this padding. (You can determine the size of the structure and the relevant fields in the structure using the sizeof() operator, and also looking at the wireshark output). This type of padding is acceptable for this exercise; just make sure to note this information when submitting your assignment.

References:

- Python
  - https://docs.python.org/3/library/socket.html
  - https://docs.python.org/3/library/struct.html
  - https://docs.python.org/3/library/time.html
  - https://docs.python.org/3/library/random.html
- C
  - https://linux.die.net/man/7/socket
  - https://linux.die.net/man/3/byteorder

Submission Instructions:
Please submit the following individual files to Canvas by due date. **Please, NO zip files.**
- ✓ Submit the client and server source program files (please name the file *client.py* and *server.py* respectively and include name, UCID, section in comments at top of source files)
- ✓ Submit a wireshark .pcap file captured while running the program (please name *ping.pcap*)
- ✓ Submit screenshots in .pdf format showing the trace output of the client and server
- ✓ Submit the README file (C programs only; No need for README for Python programs)

Grading Rubric: Total of 20 points
- Program (20)
    - Ping request packet format as specified (4)
        - Message type = 1 (request)
        - Sequence number increments from 1
    - Ping response packet format as specified (4)
        - Message type = 2 (response)
        - Sequence number echoed back to client
    - Both messages in network byte order (2)
    - Client sends 10 ping requests, and server responds (or not) randomly [at least 3 responses are demonstrated, and at least 2 drops] (6)
    - RTT calculated correctly for each acknowledged packet (1)
    - Min RTT tracked and printed (1)
    - Max RTT tracked and printed) (1)
    - Loss rate calculated and printed (1)

Late submissions not accepted on this assignment.

Academic Integrity
If academic integrity standards are not upheld, no credit is given. This includes copying of program or wireshark lab or .pcap file from any source, or hard-coding of results in your program.