CSCI 353: Programming Assignment 2

THIS PROJECT MUST BE DEVELOPED IN PYTHON 3.5+

(Python 2 submissions will receive a 0)

Due on October 9th, 11:59pm

Introduction

The goal of this assignment is to give you hands-on experience developing tools for network traffic management. In this assignment, you will use RAW sockets to generate network traffic. Develop a ping tool to send periodic ping messages to the specified IP address and compute the estimated round trip time (RTT) based on the received response times. The program will need to use RAW sockets. More information about RAW sockets is located at http://sock-raw.org/papers/sock_raw.

Command Line Arguments:

- python3 pinger.py -p <payload> -c <count> -d <destination>

where:

-p <payload>	string to include in the payload. E.g. "Hello World"
-c <count>	the number of packets used to compute RTT. Should default to 10 if no count is given. Optional.
-d <destination>	the destination IP of the ping message

When the program is run without any command line arguments, the program should print usage instructions and exit.

Note: When you have difficulty running the program using the input format above, you may use ">sudo python (py for Windows) pinger.py ...." and please note that in your README.

Example:

```
>[user@laptop:~] sudo pinger –p "hello" –c 4 -d 206.190.36.45
>Pinging 206.190.36.45 with 5 bytes of data "hello"
>Reply from 206.190.36.45: bytes=5 time=69ms TTL=47
>Reply from 206.190.36.45: bytes=5 time=70ms TTL=47
>Reply from 206.190.36.45: bytes=5 time=69ms TTL=47
```

```
>Reply from 206.190.36.45: bytes=5 time=69ms TTL=47
>Ping statistics for 206.190.36.45: Packets: Sent = 4, Received =4, Lost = 0 (0% loss),
```

Approximate round trip times in milli-seconds: Minimum = 69ms, Maximum = 70ms, Average = 69ms

---

Required Structure

Please make sure that you follow the instructions below carefully.
You will create a python function, called **print_ping_stats(message, count, destination),** that will take in 3 arguments: message, count, destination. This function will print out all the terminal outputs, as well as return a string in a comma-separated format: '<min>,<max>,<avg>'. We will be calling print_ping_stats() in the testing script.

For example, if min was 6, max was 10, and average was 8, your returned string would look like this: '6,10,8'. No spaces, just commas. If you were unable to get any numeric output (you were unable to reach the destination or timed out), just return 'N/A,N/A,N/A'.

Message Format

You will need to create the echo request packet sent to the destination IP. The format is as follows, the number in parenthesis is the length in bytes:
Type (8), Code (8), Checksum (16), ID (16), Sequence Number (16), Data (Variable)

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type = 8(IPv4, ICMP) 128(IPv6,ICMP6)								Code = 0								Checksum															
Identifier																Sequence Number															
Payload																															

You will implement "ICMP Header + ICMP Payload" by packing the above items in an ICMP message and then sending it. You may use the following links as an aid to better understand raw sockets and creating an ICMP packet using Python:

1. http://www.binarytides.com/raw-socket-programming-in-python-linux/
2. http://opensourceforu.com/2015/03/a-guide-to-using-raw-sockets/
3. https://docs.microsoft.com/en-us/windows/win32/winsock/tcp-ip-raw-sockets-2

To calculate RTT, you will need to time when you send and when you receive the packet. You should use three decimal digits for the time representation. TTL is found in the header of the echo reply packet that you receive.

Note: You do not need to worry about IP and TCP headers for the echo request packet.

Grading Rubric:

We will be using an automating script to test your pinger against the ping command in the terminal. Example output of that is as follows:

```
$ ping google.com
        PING google.com (172.217.4.174): 56 data bytes
        64 bytes from 172.217.4.174: icmp_seq=0 ttl=56 time=6.966 ms
        64 bytes from 172.217.4.174: icmp_seq=1 ttl=56 time=3.668 ms
        64 bytes from 172.217.4.174: icmp_seq=2 ttl=56 time=3.824 ms
        64 bytes from 172.217.4.174: icmp_seq=3 ttl=56 time=5.643 ms
        --- google.com ping statistics ---
        4 packets transmitted, 4 packets received, 0.0% packet loss
        round-trip min/avg/max/stddev = 3.668/5.025/6.966/1.363 ms
```

Note: Your terminal output for ping may differ based on the operating system.

You will need to output all necessary statistics: the information from each ping (including the ping count, TTL, and time), packets transmitted, packets received, packet loss, min time (smallest value out of all pings), max time (largest value out of all pings), and average time.

Your pinger should output RTT statistics within 3ms or 10% of the terminal ping statistics, whichever is greater, over the course of 100 pings. E.g., if the average ping according to the terminal is 6ms, yours should output that the average ping is between 3ms and 9ms. You will be graded based on the number of passed test cases.

Code and Collaboration Policy

You are encouraged to refer to the socket programming tutorials. You can discuss the assignment and coding strategies with your classmates. However, your solution must be coded and written by yourself. Please refer to the plagiarism policy in the course syllabus.

The submissions will be run through code similarity tests. Any flagged submissions will result in a failing score. Keeping your code private is your responsibility.

Submission Instructions

You can develop and test your code on your local machine. However, we encourage you to commit your changes to your assigned GitHub repo as frequently as possible. For grading purposes, we will only look at your most recent version.

You should also create a README, which includes your name, your USC ID, compiling instructions. Please include in your README your operating system and its endian mode. You can find a chart of the common OS's endian modes at this link. https://geraldonit.com/2017/09/04/big-and-little-endian-operating-systems/
It is important you do so because code that works on a little endian system may calculate the checksum incorrectly on a big endian system and so making it easier for us to realize this error will lead to more accurate grading. You do not have to adjust your code to determine and run on both types of systems, just include your OS and endian mode in the README.

The following files are required for this assignment:

- pinger.py
- README.txt