MPCS 54001, Winter 2019
Project 3
Due: March 10th at 11:59pm

This work must be entirely your own. If you need help, I encourage you to post questions to Piazza and/or see the course staff during their office hours. As a reminder, if you post to Piazza, please don't give away the answer!

Task

In this project, you will implement a UDP-based ping client that communicates with the provided UDP-based ping server. The functionality provided by your client will emulate some of the standard ping utility functionality available in modern operating systems, except that it will use UDP rather than ICMP. (There's currently not a good way to interact with ICMP directly on departmental Linux machines.)

Ping Protocol

This project's ping protocol enables a client machine to send a UDP packet of data to a remote server machine, and have the remote server machine return the data back to the client unchanged (an action referred to as *echoing*, which should sound familiar from Project 1). Among other uses, this project's ping protocol also enables hosts to determine round-trip times and packet loss to other machines.

Unlike real ICMP ping messages, the ping messages in this project are in a human-readable format and will be delivered in UDP payloads. Each message contains a sequence of characters terminated by a carriage return character `\r` and a line feed character `\n`. The message contains the following string as the UDP payload:

```
PING sequence_number timestamp\r\n
```

where `sequence_number` starts at 0 and is incremented for each successive ping message sent by the client, `timestamp` is the time when the client sent the message,

and `\r\n` represent the carriage return and line feed characters that terminate the line. [1]

Ping Server

You can download the complete code for the ping server here.[2]

To compile the ping server:

```
javac PingServer.java
```

To run the ping server:

```
java PingServer --port=<port>
                [--loss_rate=<rate>]
                [--avg_delay=<delay>]
```

The ping server sits in an infinite loop listening for incoming UDP packets. When a packet comes in, the server simply sends the encapsulated UDP data payload back to the client.

Packet Loss

UDP provides applications with an unreliable transport service, because messages may get lost in the network due to router queue overflows or other reasons. In contrast, TCP provides applications with a reliable transport service and takes care of any lost packets by retransmitting them until they are successfully received.

Because packet loss is rare or even non-existent in typical campus networks, the provided ping server in this project injects artificial loss to simulate the effects of network packet loss. The ping server has a command-line parameter `loss_rate` that specifies the target percentage of packets that should be dropped.

The server also has another command-line parameter `avg_delay` that is used to simulate average transmission delay (in milliseconds) from sending a packet across the Internet. You should set `avg_delay` to a positive value when testing your client and

---

[1] `timestamp` should be the number of milliseconds since the UNIX epoch. I recommend using one of the methods listed under the *Project Resources* section at the end of this project description.
[2] Hint: regardless of the language that you use, you should study the server code carefully as it will help you implement your ping client.

server on the same machine, or when machines are close by on the network. You can set `avg_delay` to 0 to find out the true round trip times of your packets.

Ping Client

Your ping client should support the following command-line options as specified below.

```
<executable cmd> --server_ip=<server ip addr>
                 --server_port=<server port>
                 --count=<number of pings to send>
                 --period=<wait interval>
                 --timeout=<timeout>
```

The command used to run your client implementation in your chosen language is represented as <*executable cmd*> above.

The ping client should send `count` total ping requests to the ping server listening at IP address `server_ip` and port `server_port`. Each ping request should be separated by approximately `period` milliseconds.[3] Each ping message contains a payload of data that includes the keyword `PING`, a sequence number, and a timestamp (see Ping Protocol section for message format details).

Because UDP is an unreliable protocol, some of the packets sent from the client to the server may be lost, or some of the packets sent from server to client may be lost. For this reason, the client cannot wait indefinitely for a reply to a ping request. For any given request, the client should wait up to `timeout` milliseconds to receive a reply. If a request times out without a reply from the server, then the client assumes that either its request packet or the server's reply packet has been lost in the network (i.e., packet loss event occurred). You will need to research datagram sockets in your language of choice to generally learn how to send/receive UDP datagrams and specifically to find out how to set the timeout value on a datagram socket.[4]

Statistics

Your ping client should print both individual and aggregate statistics. As each individual reply is received at the client, the client should print the following individual statistics:

---

[3] You will probably want to use a periodic task to implement this behavior. See “Periodic Task” under the *Project Resources* section for references in your chosen language.

[4] The *Project Resources* section has some references on how to use datagram sockets.

1. source IP address of reply UDP packet
2. ping message sequence number
3. individual elapsed time (in milliseconds)

After all replies have either been received or timed out, the client should print the following aggregate statistics:

1. total number of packets transmitted
2. total number of packets received
3. percentage of packet loss
4. cumulative elapsed time (in milliseconds)
5. minimum, average, and maximum round trip times (in milliseconds)

Here is some sample output obtained by modifying real output from the ping utility. Although not strictly required, I recommend that you make your output as close as possible to this sample output.

```
PING 128.135.164.171
PONG 128.135.164.171: seq=1 time=139 ms
PONG 128.135.164.171: seq=2 time=131 ms
PONG 128.135.164.171: seq=3 time=138 ms
PONG 128.135.164.171: seq=4 time=134 ms
PONG 128.135.164.171: seq=5 time=134 ms

--- 128.135.164.171 ping statistics ---
5 transmitted, 5 received, 0% loss, time 3999ms
rtt min/avg/max = 131/135/139 ms
```

Deliverables

Create a directory named `project3` in your individual Phoenixforge Subversion repository. Upload your source code (client source files and a Makefile if you created one) to that directory.

Grading

This project will be worth 45% of the total project grade for the course.

We will compile your client code submissions and grade them against the provided server.

General grading rubric for your ping client is as follows:

- Compiles and starts: **10 points**
- Sends ping requests over UDP to server: **10 points**
- Receives ping replies over UDP from server: **10 points**
- Uses correct ping message format: **10 points**
- Waits correct time interval between sending ping requests: **10 points**
- Times out correctly if reply not received within specified timeout: **10 points**
- Print individual statistics as replies are received: **10 points**
- Print aggregate packet loss statistics: **10 points**
- Print aggregate round trip time statistics: **10 points**
- Style points, grader discretion (efficiency, code correctness, or other considerations e.g., you didn't just hardcode source IP address in statistics): **10 points**

for a total of 100 points.

Project Resources

- Datagram sockets
  - C: tutorial, `poll(2)`
  - Java: tutorial, DatagramSocket
  - Python: documentation, settimeout()
- Timestamps
  - C: `time(2)`
  - Java: `System.currentTimeMillis()`
  - Python: `time.time()`

- Periodic tasks
  - C: time(7), signal(7)
  - Java: java.util.Timer, java.util.TimerTask
  - Python: threading.Timer