CMSC 417 Computer Networks **Spring 2016**

Project 0: Pingmap

Assigned: January 27 *Due: February 4, 11:59:59 PM.*

1 Description

In this assignment, you will write `pingmap`, a tool that combines some features from the popular `ping` and `nmap` tools. `ping` is a popular tool used to determine connectivity and Round Trip Time (RTT) to an IP address. `nmap` is a tool that can be used to "map" a network — to find which addresses in the network are used, which ports are open, what operating systems are used, etc.

For this project, you will:

1. Implement a `pingmap-client` *in C* that sends packets over UDP to a (local) server and receives its echo.
2. Find the average round trip time (RTT) between your client and the server.
3. Handle lost / reordered pings. UDP has few guarantees!
4. Search a given IP address to determine which ports the server is running on.

2 The Server

To facilitate testing your code, we provide you with a `pingmap-server` that will echo the packets that it receives. By default, it listens on port `12345`.

This server is configurable, allowing it to more drastically demonstrate the potential issues inherent in UDP. RTTs would typically be very short in our test environment, so `pingmap-server` artificially delays packets. Similarly, though it is unlikely that packets will be dropped due to natural causes in our simulated test environment, `pingmap-server` can be configured to drop a certain percentage of the packets it receives.

To execute the server, run:

```
python bin/pingmap-server.py
```

Its command line arguments are:

- `-r`: Occasionally reorder packets.
- `-l`: Occasionally drop packets.
- `-n`: Run on a random port.

3 The Client

Your task is to design the pingmap-client. Upon entering your repository, we must be able to run `make` followed by `bin/pingmap-client` in order to execute your code.

3.1 Messaging a Specific Port

Design your `pingmap-client` so that it sends messages to localhost (`127.0.0.1`). Your client should send messages based on the following command line arguments:

- `-p`: Port number on which the server is listening (the server listens on 12345 by default)
- `-n`: Number of messages to be sent to the server
- `-i`: Interval of time between subsequent messages (specified in milliseconds).

For example, the command below specifies that the client should send 10 messages to localhost on port 12345, each separated by 100 milliseconds:

```
bin/pingmap-client -p 12345 -n 10 -i 100
```

Your client should be capable of processing the command line arguments in any order, just like a typical Unix command is capable of processing command line arguments in any order. Also, like typical Unix commands, your client should be capable of handling only a subset of these arguments. (Hint: There's a useful function for doing this.)

If some of the arguments are not specified on the command line, you should default to:

- port number = 12345
- Number of messages = 1
- Interval of time between subsequent messages = 0

Your client's task is to calculate each message's RTT before exiting. If you do not receive a response to your message within one second, mark the packet as dropped. You may include any information in the message's payload that you'd like; it will be echoed back to you. The mechanics of calculating RTT and message ordering are completely up to you. Upon completion, `pingmap-client`'s output (to `stdout`) will be of the following form:

```
Message 0: 10ms
Message 1: 120ms
Message 2: Dropped
Message 3: 11ms
...
Message n: 9ms
```

The RTT of the messages must be output in the order they were sent, regardless of the order of the responses that you received.

When you submit, ensure your code does not print any other text to `stdout`. Feel free to print any debugging messages that you'd like to `stderr`.

3.2 Searching For the Server

To search for the port running the server on `127.0.0.1`, your program is executed by:

```
bin/pingmap-client -r
```

Your client will send a message on each port (> 1023) on `127.0.0.1` and listen for responses to determine which port the `pingmap-server` is running on. `pingmap-server` will never run on a port less than 1024. Your `pingmap-client`'s output will be of the following form:

```
Port: 12345
```

Where `12345` is the port that the `pingmap-server` is running on. Again, please ensure that no other text is printed to `stdout`.

4 Getting Started

We will not be using the CS Submit server for this course. Instead, you will submit your code on GitHub. If you are unfamiliar with Git, there are several excellent tutorials available online, and you may ask questions on Piazza. To set this up:

1. Create a GitHub account (or login to an existing one)
2. Navigate to `http://ter.ps/417p0`
3. Click "Accept this assignment"

This will create a private repository for you with any starting code or tests that we give to you. These are yours to modify, provided your code is still buildable by running `make` and executable by running `pingmap-server`. Additionally, for this project, *your implementation must be in C.*

4.1 Vagrant

In your initial repository, we provide a `Vagrantfile`. Vagrant is a tool that allows you to easily spin up clean virtual machines on your computer within seconds. This `Vagrantfile` describes a Vagrant VM that should be sufficient for your code to run in. To get started with Vagrant:

1. Install Vagrant. `https://www.vagrantup.com`
2. `cd` into the directory containing the `Vagrantfile`.
3. Call `vagrant up` to boot the VM.
4. Call `vagrant ssh` to `ssh` into the VM.
5. Test code. All files in the directory containing the `Vagrantfile` will be mounted to `/vagrant` within the VM.
6. Call `vagrant destroy` to destroy the VM.

You are welcome (and encouraged) to test your code in the VM, but be careful! Only the `/vagrant` directory is synced with your filesystem, and any other files on the VM will be lost when you call `vagrant destroy`. We won't be able to recover your code if you accidentally lose it.

4.2 Submitting

To submit your code, simply push to the `master` branch in your repository on GitHub. Our autograder will detect this push, test your code, and add a GitHub comment to the commit with any public test results. This comes with the following caveats:

- To encourage you to test your own code before submitting, we will not test your code as frequently as the submit server normally would. Despite this, you can expect, at a minimum, your most recent submission to be tested once every 24 hours.
- Pushes to branches other than `master` are perfectly fine, but will not be counted as submissions.
- Be sure you actually *push* your submission, and don't just commit it locally.
- Your grade will be based on the last commit *pushed* before the deadline. If you push after the deadline, it will not look at any new commits included in that push, even if the commit's timestamp indicates it was committed before the deadline.
- Don't force push (`git push -f`).
- There may be secret tests, including using servers that are similar but not identical to `pingmap-server`.

5 Grading

Our tests will allow you to receive partial credit for only fulfilling a subset of these requirements.