CMSC 417 Computer Networks **Spring 2016**

Project 1: Pingmap-Traceroute

Assigned: February 8 *Due: February 22, 4:59:59 PM.*

1 Description

In this assignment, you will extend the `pingmap` tool from *project 0* to add features from the popular `traceroute` tool as well as providing a way to map out a simulated network. `traceroue` is a popular tool used to determine the path between two hosts by using the *time to live (TTL)* field in the IP header to force each hop along the path to generate an error message and send it back thus providing a list of every hop along the path.

In this project, we will extend the base we have from *project 0* to explore some of the concepts around IP forwarding that we've discussed in class. In particular, your client will probe a simulated, IP-like network by sending specially-crafted messages that will be forwarded inside the network and eventually a reply will be returned to your client. In place of the `pingmap-server` we will provide a `pingmap-router` that will only respond to messages that are directed to it and will forward messages not destined to it according to the fields in the message. Each message should formatted as follows and sent as a UDP segment.

bits 0 through 31

	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27	28 29 30 31
0	type	version	TTL	hdr len	flags
1	source	destination	message		
...	message (cont'd)				

Where

- **type** should be set to `0x8`
- **version** should be set to `0x1`
- **TTL** will be decremented at each hop and upon hitting zero an echo will be sent back with the TTL expired flag set.
- **hdr len** is the length of the message header in 16-bit increments. Thus, a typical message header will have length three indicating it is 48 bits long.
- **flags** four bits to indicate special conditions with the message—reserved bits should be set to zero

28	29	30	31
reserved	reserved	reserved	TTL expired

- **source** should be the 1-byte ID of the node sending the message, e.g., your node's ID for messages you send and the server/router's ID for replies.
- **destination** should be the 1-byte ID of the node the message is for, e.g., the server/router's ID for messages you send and your node's ID for replies.

For this project, you will:

1. Extend your `pingmap-client` *in C* to send messages with the new message header and receive their echo from the specified server.
2. Find the average round trip time (RTT) between your client and the specified server.
3. Handle lost / reordered messages and replies as in *project 0*.
4. Produce the list of `pingmap-router`s between your client and the specified server.
5. Provide the average latency to each `pingmap-router` along the path.
6. Provide a map of the simulated network listing all the links visible to you.

2 The Router

To facilitate testing your code, we will provide you with a `pingmap-router` that will forward messages it receives that aren't destined for it (assuming the message has a valid destination), respond to messages that are destined for it, and reply with error messages when a message TTL hits zero.

NOTE: you will not be required to implement any of the ***`pingmap-router`*** *in this project—it will all be provided for you.*

Like the `pingmap-server` in *project 0*, the `pingmap-router` is configurable. Allowing it to simulate a variety of different networks. Instead of providing parameters for loss and reordering, you instead provide it with parameters for each of its simulated links and its simulated forwarding table. This configuration will all be stored in a single file and the `pingmap-router` will take only two parameters: the name of that file and its node ID so that it can discover what parts of the configuration file pertain to it.

The configuration file will consist of lines of three kinds: ones specifying nodes, ones specifying links, and ones specifying forwarding table entries.

The node lines will look like:

```
node <nodeID> <port|client>
```

Where each line specifies a node and the UDP port on which that node will be listening. One node will instead be labeled as the client and will listen on a dynamic port.

The link lines will look like:

```
link <nodeID> <nodeID> <latency-in-ms> <loss-rate>
```

Where each line specifies a symmetric link between two nodes with a specified latency in milliseconds and loss rate specified as a floating point number between 0 and 1. A loss rate of 0 will result in all packets being delivered and a loss rate of 1 will result in all packets being lost.

The forwarding table entry lines will look like:

```
forward <nodeID-curr> <nodeID-dest> <nodeID-next>
```

Where each line says that when the node with ID `nodeID-curr` receives a message destined for the node with ID `nodeID-dest` it will forward it to the node with ID `nodeID-next` assuming that the TTL field is still positive and that there is a link between `nodeID-curr` and `nodeID-next`.

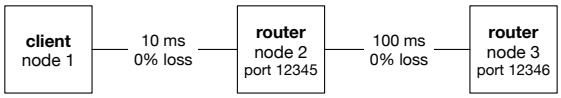

Figure 1: A sample simulated network.

As an example, the configuration file for a 3-node simulated network shown in Figure 1, where the `pingmap-router` nodes have IDs 2 and 3 and where the `pingmap-client` has node ID 1 could be written as follows:

```
node 1 client
node 2 12345
node 3 12346
link 1 2 10 0.0
link 2 3 100 0.0
forward 2 3 3
forward 2 1 1
forward 3 1 2
forward 3 2 2
```

To execute the router, run:

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

Its command line arguments are:

- `-f`: the configuration filename, defaults to `config`
- `-d`: the node ID, defaults to 2

To be clear, each time you run the `pingmap-server` it will behave as though it is one node in the simulated network. Thus, to simulate the network described in Figure 1, you would need to run the `pingmap-server` twice: one for node 2 and one for node 3.

3 The Client

Your task is to extend the `pingmap-client` from *project 0* to add `traceroute`-like features. Note that we have explicitly designed this part so that it will require minimal modifications to the solution to *project 0*. Upon entering your repository, we must be able to run `make` followed by `./bin/pingmap-client` in order to execute your code. Your client should accept the following command line arguments:

- `-d`: Destination Node ID your client should send messages to
- `-l`: Local Node ID your client should send messages from
- `-i`: Interval of time between subsequent messages (specified in milliseconds)
- `-n`: Number of messages to be sent to the router

- `-p`: Port number of the `pingmap-router` that will act as your *gateway* into the simulated network.
- `-t`: Produce a list of hops in order of distance from your client
  - you can ignore anything passed in via `-i` or `-n` when this option is enabled
  - you will be passed at most one of `-t` or `-m`
- `-m`: Produce a map of the links in the simulated network that your client can see
  - you can ignore anything passed in via `-i`, `-n` or `-d` when this option is enabled
  - you will be passed at most one of `-t` or `-m`

For example, the command below specifies that the client should send 10 messages to the node with ID 2, each separated by 100 milliseconds all using the node listening on port 12345 as the first hop (or *gateway*):

```
bin/pingmap-client -p 12345 -n 10 -i 100 -d 2
```

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 any subset of these arguments. (Hint: There's a useful function for doing this.)

As in *project 0*, if some of the arguments are not specified on the command line, you should default to:

- destination node ID = 2
- local node ID = 1
- port number = 12345
- Number of messages = 1
- Interval of time between subsequent messages = 0

3.1 Messaging a Router

When neither `-t` or `-m` are specified, your client should behave exactly the same as the `pingmap-client` from *project 0* except that it should use the new message header so that it can send messages to the simulated network where it might be forwarded by more than one router before getting a response.

Note that while in *project 0* you were sending to a server identified by its port (specified by with the `-p` option on the command line), in this project you will be sending to a router identified by its node ID (specified with the `-d` option). In this project, the `-p` option specifies the port on which you can find your *gateway* router to get access to the simulated network, not the server (or router) you are sending to.

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 message 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 Traceroute `-t`

When your client is passed the `-t` option, it should trace the route to the destination router figuring out each hop along the route (via it's node ID) and calculate the (round-trip) latency to each one. Upon completion, `pingmap-client`'s output (to `stdout`) will be of the following form:

```
Hop 1: 12 10ms
Hop 2: 17 40ms
Hop 3: 3 53ms
...
Hop n: <destNodeID> 80ms
```

This list should always end with the destination and thus there will always be at least one hop. You can assume you will get a reply from a node after a reasonable number, e.g., 5, of attempts.

Note that the latency should be the round trip time to get a reply back from that router and so you should expect the latency reported for each hop to go up, e.g., the latency of hop 1 will be less than that of hop 2 which will be less than that of hop 3 and so on.

3.3 Map `-m`

When your client is passed the `-m` option, it should produce a list of all the links the client can observe in the simulated network and their latency. You can assume that there is at most one link between any two simulated nodes, that each link has the same latency in each direction, and that you will get replies after a reasonable number, e.g., 5, of attempts. Upon completion, `pingmap-client`'s output (to `stdout`) will be of the following form:

```
Link 1 2 40ms
Link 2 7 32ms
Link 17 19 21ms
Link 4 7 19ms
...
Link <x> <y> 17ms
```

For each link, please print the smaller node ID first. As an example, `Link 3 7 11ms` would be a valid line, but `Link 7 3 11ms` is not. You can print the links in any order you like—we will sort the output before comparing it to the correct output.

Unlike in the the previous two instances, the latency should be the one-way latency of the link rather than the the round-trip latency. That is, your goal is to recover the latency of the link as specified in the config file within a reasonable amount of error.

Note that there is an upper bound on the number of links in the network given the above assumptions, but we leave computing that and how to use that in your solution to you.

4 Getting Started, VMs, Vagrant, and submitting

This will all be the same as *project 0*.

5 Grading

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