## Summary

The goal of this team project is to build a simulator to evaluate the performance of various routing algorithms. This project will implement the following routing algorithms:

1. Random
2. Flooding
3. Shortest path
4. Custom method

A user interface will be implemented for the simulator to display the network topology and let the user edit it. The interface will always provide the metrics of the network and the ability to view and step through a simulation.

## Milestones

The project is divided into 4 iterations, each of which correspond to a milestone with deliverables.

|  |  |  |  |
| --- | --- | --- | --- |
| Milestone | Description | Due Date | Weight |
| 1 | A simple console-based version. Only the random algorithm will be implemented. | Oct 21 | 15% |
| 2 | GUI and unit tests. | Nov 7 | 20% |
| 3 | All routing algorithms are to be implemented and selectable through the GUI. | Nov 21 | 30% |
| 4 | Add the ability to step back in a network (i.e. undo function) and the ability to save/restore a simulation of the network topology. | Dec 5 | 35% |

## Design Choices

The design choices our team made are logged below for each milestone:

**Milestone 2:**

As shown in the UML diagrams below, some changes and additions were made to the overall project design. The algorithms **RoutingAlgorithm**

**Milestone 1:**

For implementing our four routing algorithms, we decided to use the strategy pattern by creating a **RoutingAlgorithm** interface. At this point the **RoutingAlgorithm** interface has all the code for the how each algorithm should run, we will be changing this in Milestone 2. **ALGORITHM** enum class was created to be able to call a specific algorithm.

A **Message** class was created to store a random source and destination of a message being sent through the network.

A class was also created for **Graph** which includes a set of nodes and list of edges also modeled as classes (**Nodes** and **Edge**)

A **Simulation** class was created to simulate the project generating a **Graph** and displaying progress and metrics to the user using a **Controller UI** class

**Assumptions:**

* The user should not be able to step through the simulation (for Milestone 1)
* The metrics are therefore printed at the end of the simulation
* The RANDOM algorithm chooses a random neighbor to forward each message at each hop.
* The frequency is the number of hops each message takes before a new message is injected into the network
* The user cannot edit the graph once it has been created (Milestone 1)
* There are no intermediate metrics displayed between hops- only at the end
* The routing tables or the path taken by each message is not shown to the user
* There is a chance of the simulation running infinitely on a large graph and low frequency (functions as designed)

## UML Diagram

UML diagrams of the network for each iteration are shown below:

**Milestone 2:**

**Milestone 1:**

![m1\_uml](https://cloud.githubusercontent.com/assets/14824913/19614758/411b531e-97c6-11e6-9c2b-2970fc85ec8c.png)

(Please Note: Sequence diagrams were not needed for Milestone 1)

## Developer Contributions

Contributions to project for each developer are listed below for each milestone.

**Milestone 2:**

|  |  |
| --- | --- |
| Developer | Contributions |
| Alex Hoecht | Implemented and re-factored back end models |
| Andrew Ward | Created Unit Tests |
| Mohamed Dahrouj | Implemented Visual Graph |
| Shasthra Ranasinghe | Created GUI |
| Everyone | Created block comments for JavaDoc, Created UML Diagram, and integrated all code |

**Milestone 1:**

|  |  |
| --- | --- |
| Developer | Contributions |
| Alex Hoecht | Implemented the random routing algorithm and layed groundwork for other algorithms |
| Andrew Ward | Implemented the Graph class consisting of Node and Edge objects |
| Mohamed Dahrouj | Implemented a Simulation class that creates a simulation and stores metrics |
| Shasthra Ranasinghe | Implemented a console UI to interact with users and Simulation |
| Everyone | Created block comments for JavaDoc, Created UML Diagram, and integrated all code |

**Note to Marker:**

* At our first meeting we decided on a very basic idea of how the simulation should run, with many assumptions and a lot of the functionality missing. The tasks were split equally and a branch was created for each member where we worked on our section of the code
* Once we all had each of sections ready, we collectively decided at our second meeting that it was better for the first milestone if we manually place all the code we have into the master branch and delete our individual branches. We then branched from master in order to have at least a semi-functional program to work with and each member knows how the design was implemented in code
* We still had trouble merging code. To work around this we decided to manually copy the most up-to-date code into M1\_Alex branch and work on that together either from our own computers or on the same computer
* For the first milestone, the total packets metric was implemented
* The JavaDocs are included for all public members in the **doc** folder

## How to Run

The steps required to run the project are outlined below for each milestone:

**Milestone 2:**

To run the project, please create a Java Application under "Run Configurations" using the main() method provided in the **NetworkRunner** class. On the other hand, please refer to CuLearn for the .jar file including source code. For running the jar file, please open a command prompt, and use the command **java -jar milestone2.jar**

**Steps:**

**Milestone 1:**

To run the project, please create a Java Application under "Run Configurations" using the main() method provided in the Simulation class. On the other hand, please refer to CuLearn for the .jar file including source code. For running the jar file, please open the cmd prompt, and type **\*\*java -jar milestone1.jar\*\***.

**Steps:**

1. Enter all the names of the nodes in the graph with spaces separating them

2. Enter the link between nodes by following the instructions on the console. Provide one link at a time and press Enter

3. Enter the frequency (number of hops each message takes before a new message in injected into the network)

4. The simulation might run infinitely if the frequency is low, otherwise the metrics will be printed out and simulation will end.

Last updated: November 5, 2016