NETWORK SIMULATOR GROUP 3

Software Project

Group members:

Rafael, Selim, Haapaniemi Lari, Juhan

Table of Contents

1. Overview

The project that our team has developed allows the creation and performance evaluation of various networking traffic scenarios and configurations. The software developed makes use of the C++ Qt library to create a user interface in which the network set-up can be created and simulated in.

The network consists of four main elements: routers, endhosts, links and packets. Nodes (routers and endhosts) are connected to each other through links and packets (pieces of data) will travel through the network from a source node to a destination node.

The user is allowed to test any network combination scenario within network grid of 6x8, by using either using the UI developed or by specifying the configuration in csv files in the repositories home directory.

In summary, the network is able to forward packets through the network under certain configurations. Nevertheless, on some occasion's routers will not be able to forward packets due lack of a network aware routing tables. This is due to issues in the integration of networking algorithms with the router class.

2. Software structure

2.1 Backend

2.1.1 The difference to the original structure

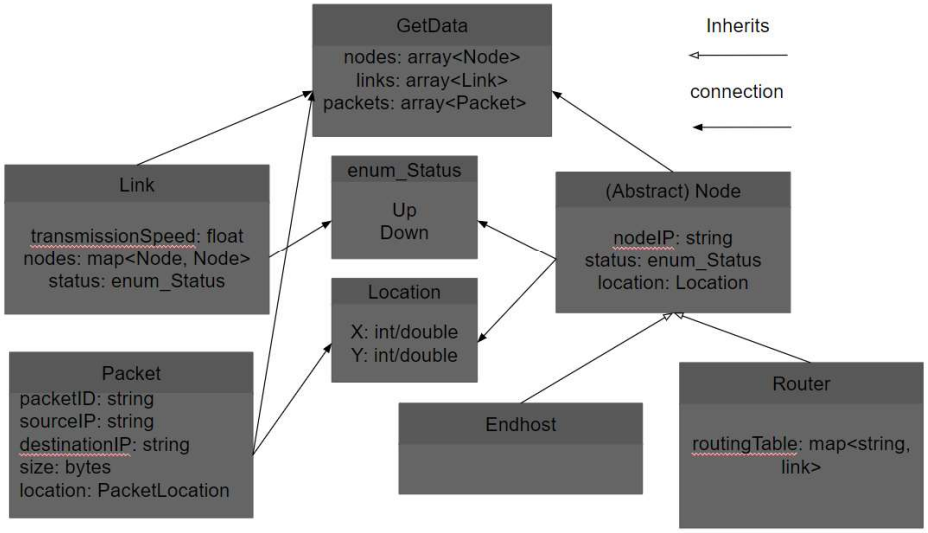

Figure 1. The backend structure slightly altered, with the most important values showing.

The structure of the backend is roughly the same as the original plan, with a couple of adjustments done. For example, the buffer for the router has been removed, since we did not implement queue behaviour, instead multiple packets can travel in the link at the same time. This is something we would have liked to implement, and had the surrounding structure for, but at the end did not quite meet the deadline. The endhost does not store packets in the sense that they are added to a specific node, since this feature was not seen as an important one when trying to show off a network structure. However, the like the buffer, the structure does support adding this. While these features were cut, we added new features that were not originally intended to router, getData and link.

2.1.2 Node, Endhost and Router

Endhost and router are subclasses of the node superclass. The node class consists of values shown in figure 1. The location and ip given have to be unique and are checked in getData class. The node class does not allow copying, since only one iteration of the node should exist at the same time. The endhost class has queue system implemented, but it is not used in the final product. Therefore, it has the same features as the node class, but is separated from it for clarity's sake. The router class on the other hand was meant for implementing networking functionalities by interacting with the algorithms proposed. Initially, the router would select an algorithm to generate paths from source (itself) to all destinations. The second element of this path is the next hop (or next node). This process would be repeated

iteratively in each router until all routing tables are generated, creating an optimized path for a packet for every source and destination pair.

2.1.3 Routing table, algorithms

The router class is intended to be used for forwarding packets from node to node. This class stores a routing table as a map, in which the first element indicates the destination IP address, and the second element indicates the link to be used. Here is an example illustrated below:

Node 1 Routing table	
IP1	Link1*
IP2	Link2*
IP3	Link3*
Other	Link2*

Therefore once, a packet arrives to a router. The packet's destination IP address is searched in the table and then the link it should use is returned. To create this routing table several protocols have been proposed as it is described in following sections. The Algorithms will use the getData class which retrieves the network configuration and provide for each router a routing table with the optimal paths to be followed for packet based on a link's performance.

2.1.4 Link and Packet

The link class is used to represent a link between two nodes. The link can only connect endhost to router, or a router to a router, meaning that endhosts cannot be connected. The link calculates its of length and cost when created based on the two nodes given to it, based on the transmission speed and distance. These are important features especially in the graphical representation, and in the routing table.

The packet class consists of the source node, destination node, its own id, size and location, which is represented as a double because of the graphical interface. It also has a boolean flag, that represents whether the packet has reached its destination.

2.1.5 GetData

GetData class is used to create, update, delete and store the Packets, Nodes, Links, and Routers. Also, GetData class is a connection point between the backend and frontend of the application.

When an application is started it will collect information about the Packets, Nodes, Links, and Routers from the CSV tables if they exist on the root directory and it will store them into the containers. The class has many functions so that frontend and backend classes could create/update/delete these items. And all functions have checks so that only valid items could be created/updated. When the application is closed the GetData class will save the current state of Packets, Nodes, Links, and Routers, and it will store them into the CSV tables.

2.2 Frontend LARI

The user interface for the application was developed using the QT creator and QT library. The structure of the frontend can be seen in the figure below. The frontend consists of one main page and three setup pages as well as two update pages.

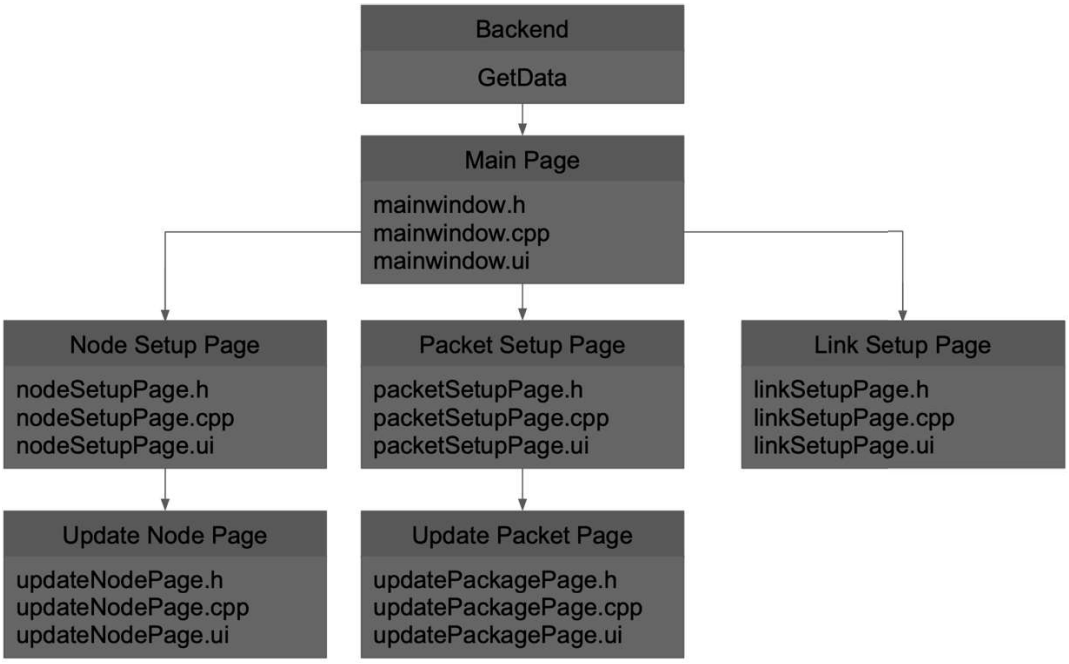

Figure 2. Frontend structure

2.2.1 Main page

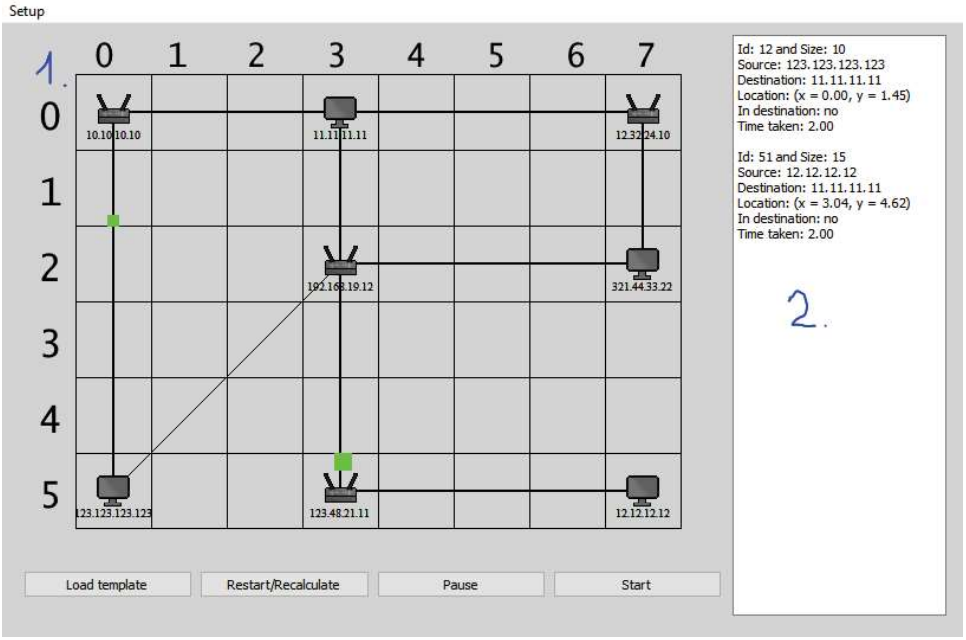

Figure 3. Main page, where 1. is network view and 2. is statistics view.

The main page can be roughly divided into two areas. These are the network view and the statistics view (shown in figure 2). The network view (1) consists of four buttons which of three are strictly for interacting with the networking simulation. How this view is used is explained in the third section, but we will quickly mention the functionalities here also. The Load Template loads the default template for the routing system, the Restart/Recalculate recalculates the simulation and loads it from the beginning, Pause pauses the simulation, and Start makes it run. Above the buttons we see the actual simulation, which draws the network based on values retrieved from the backend. These can be altered in the setup pages. The greed dot represents a package, and its size depends on the size of the package. The current state of the simulation does not consider link speeds, and instead utilizes the first path it found.

The statistics view shows information about the packets in the simulation. These values are retrieved from the package class in real time, except for the time taken, which is calculated in the simulation part, which is hosted in the frontend mainwindow.

The main page is inside the same class, which is called mainwindow. It has multiple different containers to be able to update the textures in real time and draw the simulation correctly. It also keeps track of the current state (represented by 0, 1 or 2). The mainwindow structure ideally would have been separated into multiple different classes, but our unfamiliarity with the QT library decided us to take the safe route and implement the entire simulation in the same place to begin with. Sadly, we did not have time to separate this functionality into different pieces.

2.2.2 Setup pages

Setup pages are used to create, delete, update the Nodes, Packets or Links used in the application.

In the node setup page, user can create, delete, update endhosts and routers as well as see created nodes in the tables as seen in the figure below. When the router is selected from the table it will also show links to which it is connected.

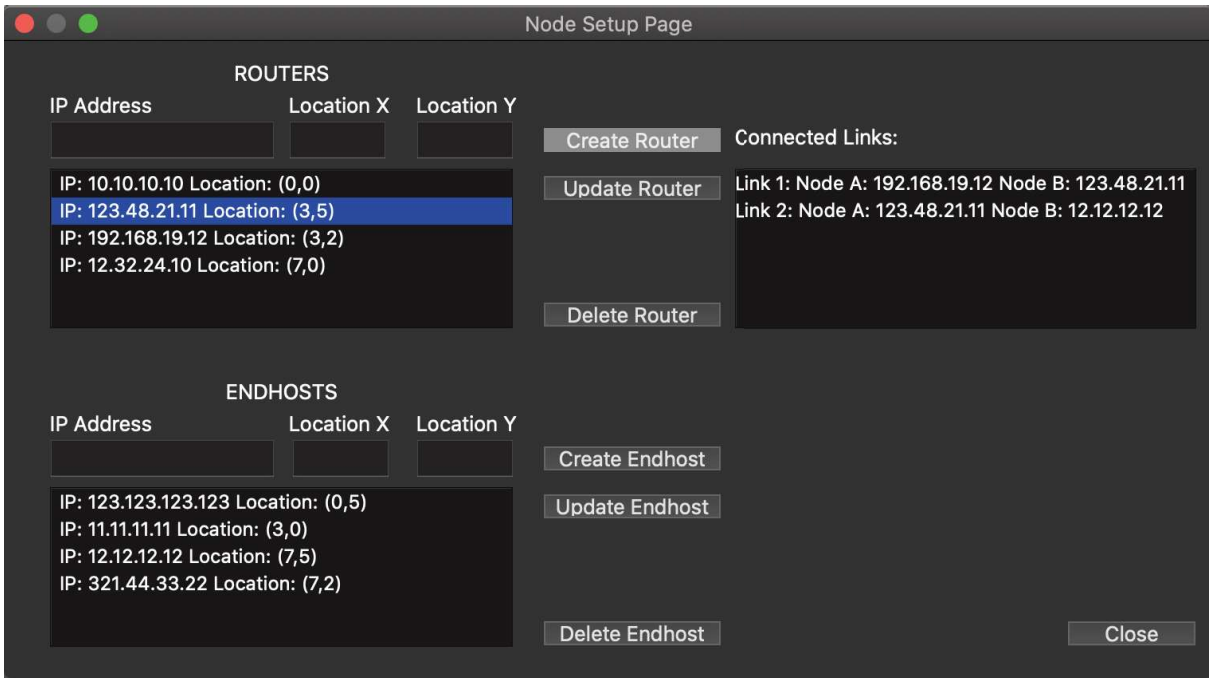

Figure 4. Node Setup Page

When a user tries to create an item that is not valid, for example, if the user does not enter the IP address or location when the pop-up message will appear, and the item will not be created.

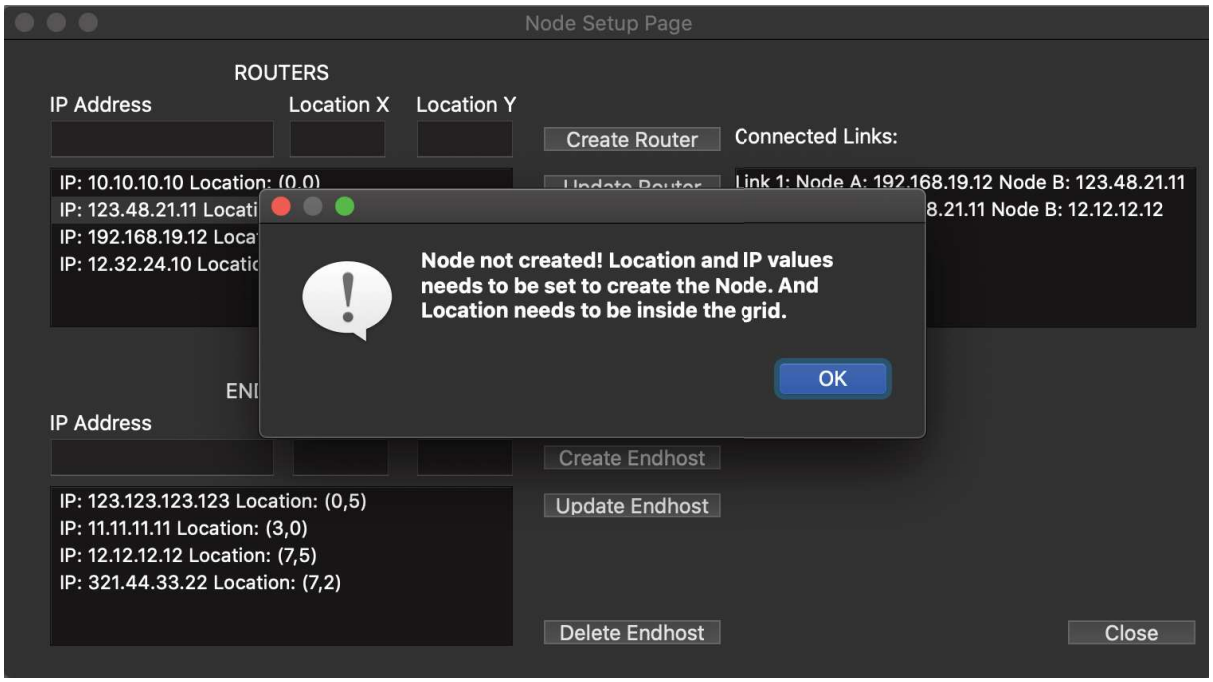

Figure 5. Node Setup page warning message

Similarly, when a user tries to delete or update the item without selecting the item from the table, it will show the message and the action is aborted.

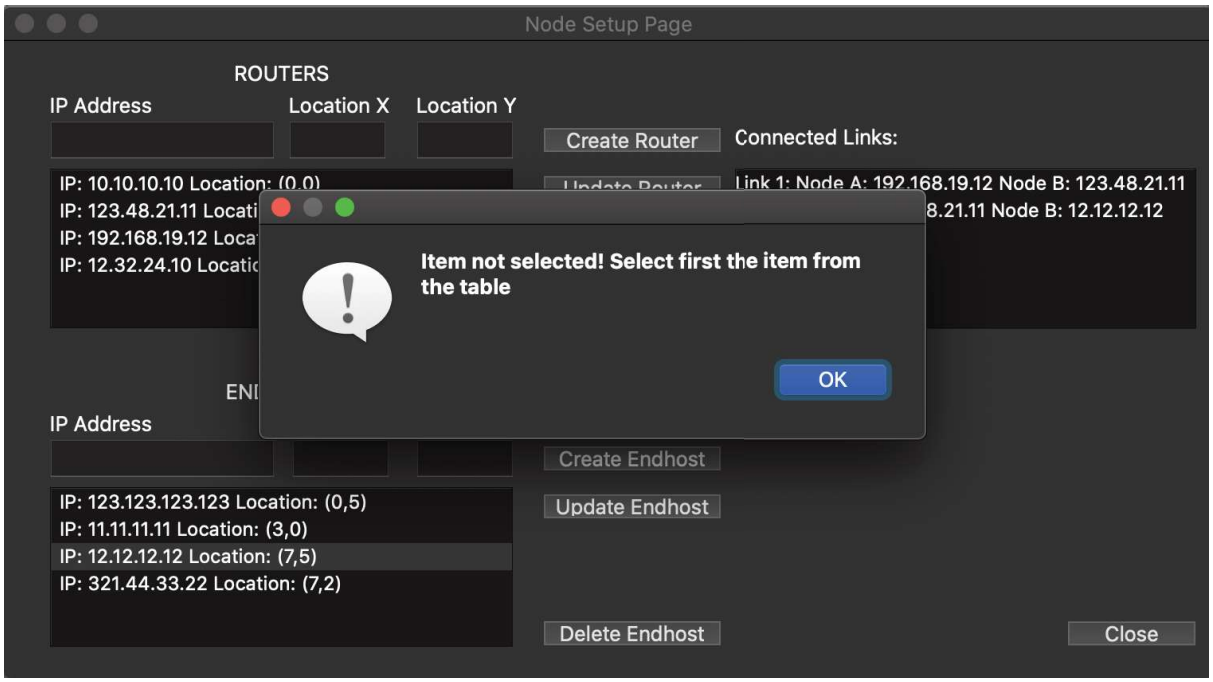

Figure 6. Node Setup page item not selected warning message.

When a user selects the item from the table and presses the update button, the new window will pop up where user can update the selected item. And pop-up messages will appear if the update is not valid, and update of the item will be cancelled.

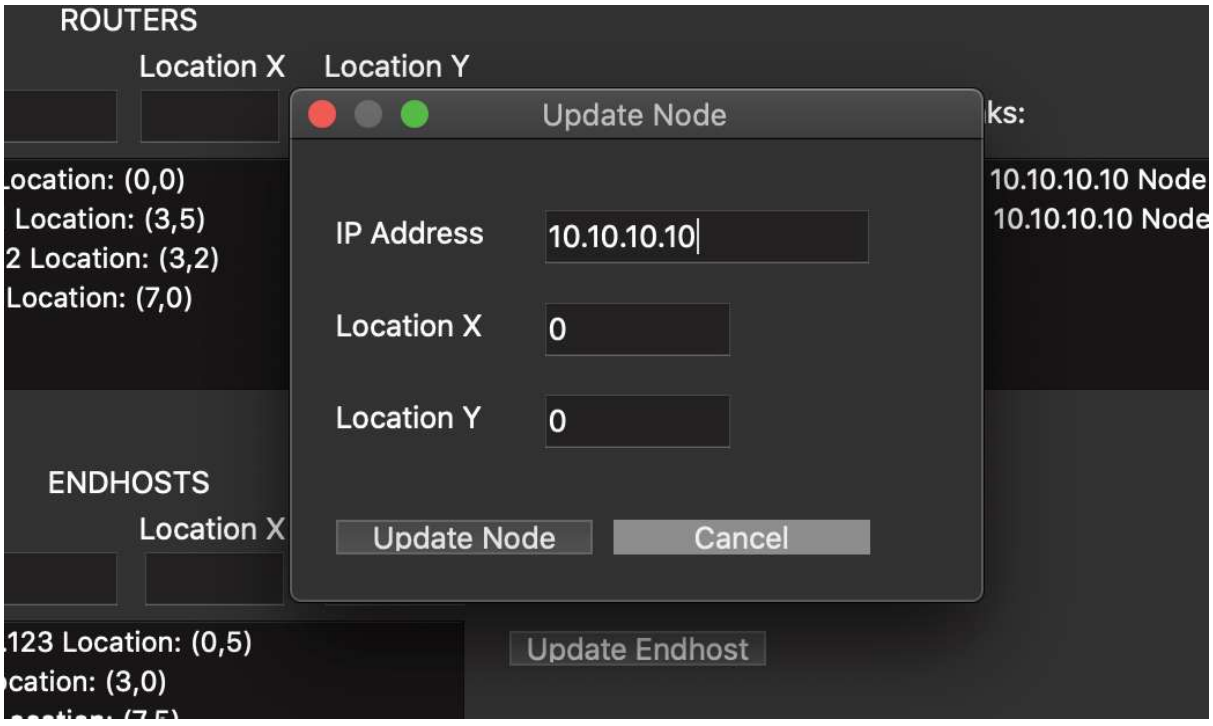

Figure 7. Update node page.

On the packet setup page user can create, delete, and update the packets as well as see all the created packets. And similarly, as in the node setup page, the pop-up message will appear when a user tries to create an item that is not valid, or the item is not selected from the table when the delete or update button is pressed.

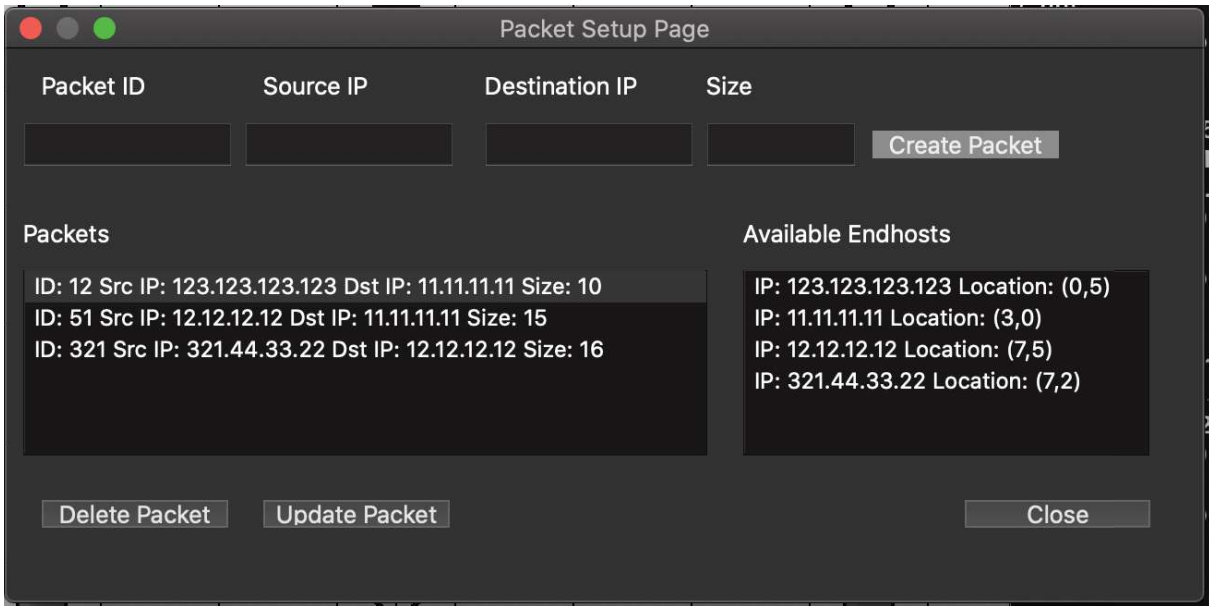

Figure 8. Packet Setup Page.

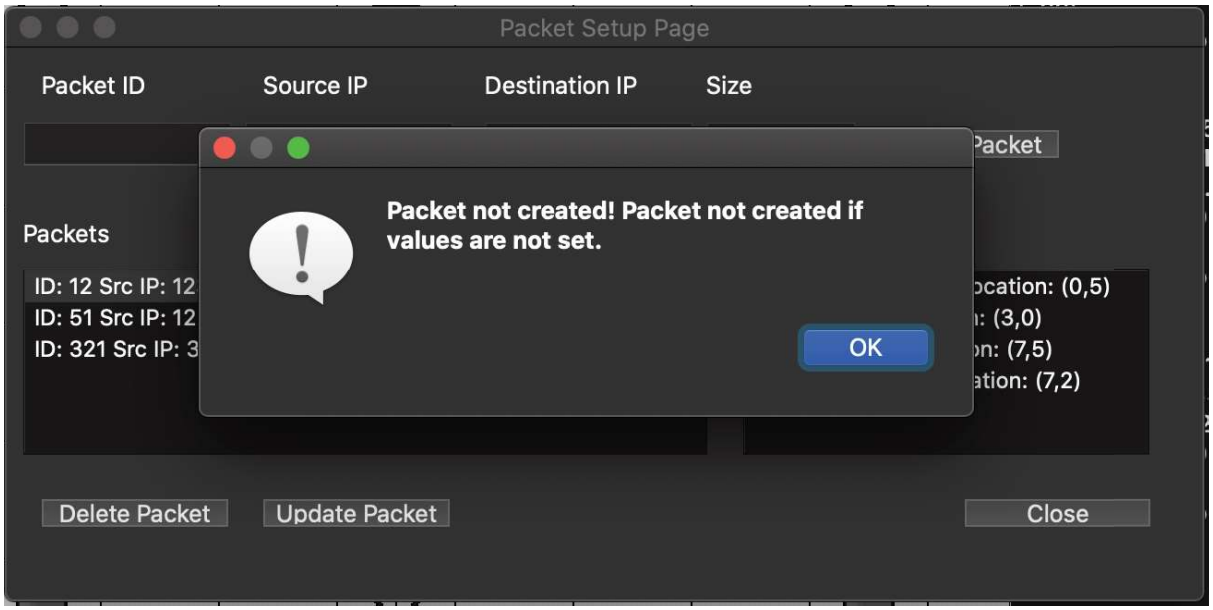

Figure 9. Packet setup page with a warning.

When a user selects the packet from the table and presses the update button when the new window will pop up where the user can update the packet values.

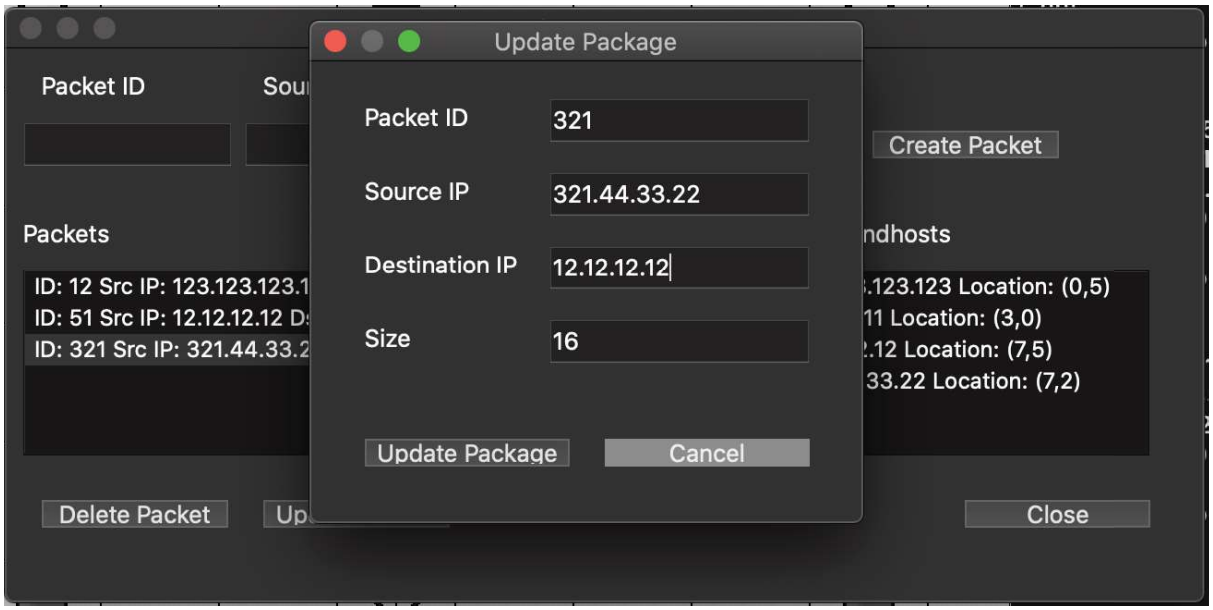

Figure 10. Update package page.

On the link setup page, the user can create and delete the links as well as see all the links created. Similarly, to other setup pages, the link page will not allow to create links that are not valid, and it will inform the user about these situations with pop-up messages.

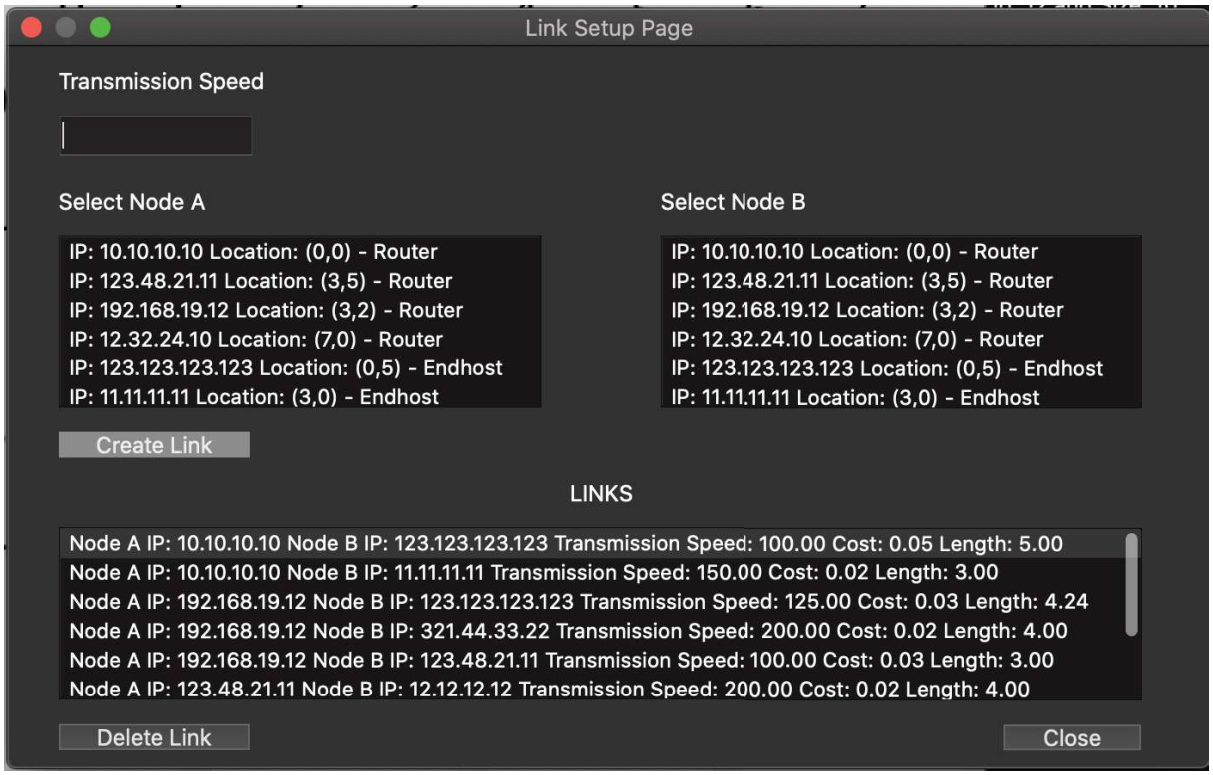

Figure 11. Link Setup Page

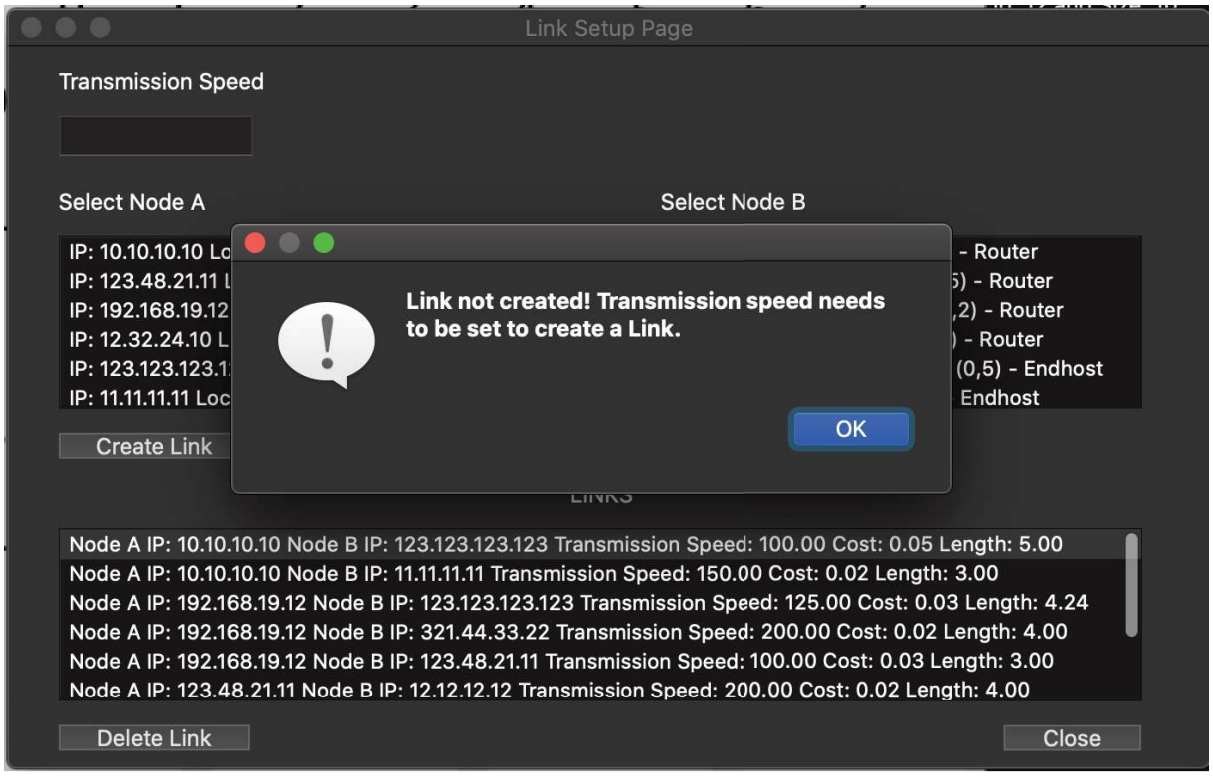

Figure 12. Link setup page with a warning.

3. Instructions for building and using the software

3.1 Building and starting the application

The application is built using cmake and best option would be to build the application in the Aalto's Linux VM's. Our application has been only configured to work with Unix operating systems.

Using Aalto Linux VM:

1. Access VM using Aalto Virtual Desktop Infrastructure
   - Instructions: https://www.aalto.fi/en/services/vdiaaltofi-how-to-use-aalto-virtual-desktop-infrastructure
2. Select Linux VM

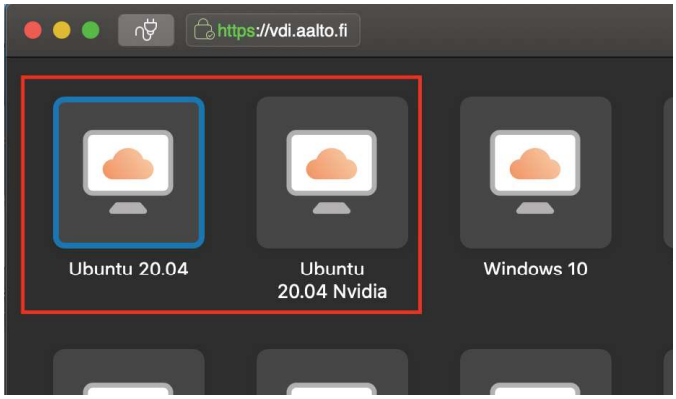

Figure 13. Aalto Linux VM's

3. Clone or download as a zip the project and go to the root directory of the project using the terminal.
4. Build the project using the cmake

```
> mkdir build
> cd build
> cmake ..
> cmake --build .
```

Figure 14. Steps to build the project

5. Start the application from the build directory
   - Use command "./NetworkSimulator " in the build directory

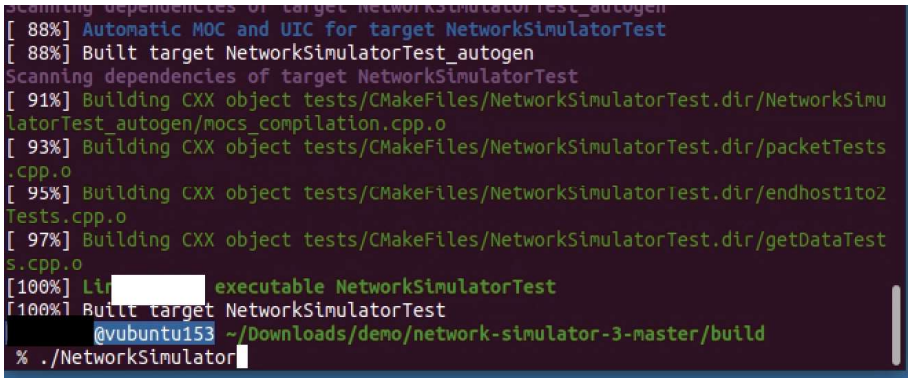

Figure 15. Start the application

3.2 User guide

After the program has been compiled and the exe file run, you are greeted by the main screen introduced in figure 2, with a different scenario shown in figure 15.

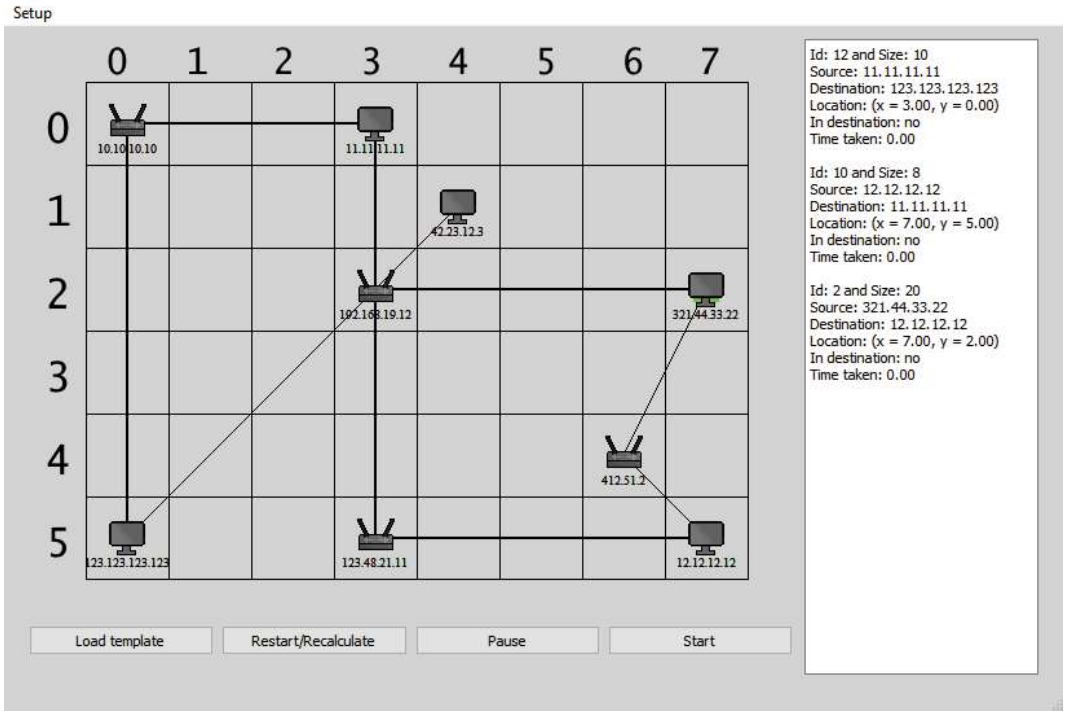

Figure 16. Example of a networking scenario.

The main screen has a dropdown menu on the top left corner, which will open the menus shown in figure 4. In the grid you can see two different nodes, the router, which is represented by a picture of a router, and an endhost, which is represented by a computer screen. The routers represent crossing points for endhosts, and the endhost are used to send and receive packets. The nodes are connected by link, which are given a speed when creating it. This determines the speed of the packet on the link relative to its size. When creating a packet, you choose the start point, and its destination, along with its size. The packets are represented by different sized green dots.

The right side of the screen shows the statistics about the current simulation. It will tell whether the packet has reached its destination, its current location (which is also represented by real time movement of the green dot), and the time it has taken to reach the current point.

The four buttons are used to load the default template (the network is saved on a different csv file when closed, so if you want to go back to the original, this button allows it). The Restart/Recalculate button is used to restart the simulation and recalculate it if changes have been made to for example nodes. The Pause and Start buttons are used to control the simulation state.

Since there is a problem with the way that routes are calculated, there is a possibility that the simulation cannot calculate the packets path to the destination. This is shown by the fact that if the start button is pressed, no packets are visible, and the statistics screen has a message that tells you that the simulation cannot be created. This can happen even if there is a sensible route for a packet, because if a certain endhost has multiple routers connected to it, and those routers are not connected to other routers, the calculation path might not be able to compile. If there is only a singular packet that cannot be calculated that one will just count to infinity, while the other packets will be simulated.

In the menus opened by the drop down you can add new routers and endhosts, and alter existing ones. The packet menu allows you to create new packets for the simulation, and link menu allows you to link two nodes together, possibly opening a faster route. These are explained in more detail in the software structure section.

4. Testing

4.1 Backend testing

Endhost and Link testing: (endhost1to2.cpp)

To make sure that the endhost functionality works, this file “sends” a packet from one endhost to another one. This was in the early stage of development, so we still used to flip statuses and allowed sending a packet straight from one endhost to another. This file also tests packets and link.

GetData class testing: (getDataTests.cpp)

GetData class was tested using google unit tests. For the most part the unit test was created for each method in the class to ensure that methods behave as expected and these tests were run every time when project was built. On total 26 test case were created to test the GetData class.

Packet class testing: (packetTests.cpp)

Packet class was tested using google unit tests as well and tests were run every time when project was built. Two-unit tests were created to test the functionality of the methods on the packet class.

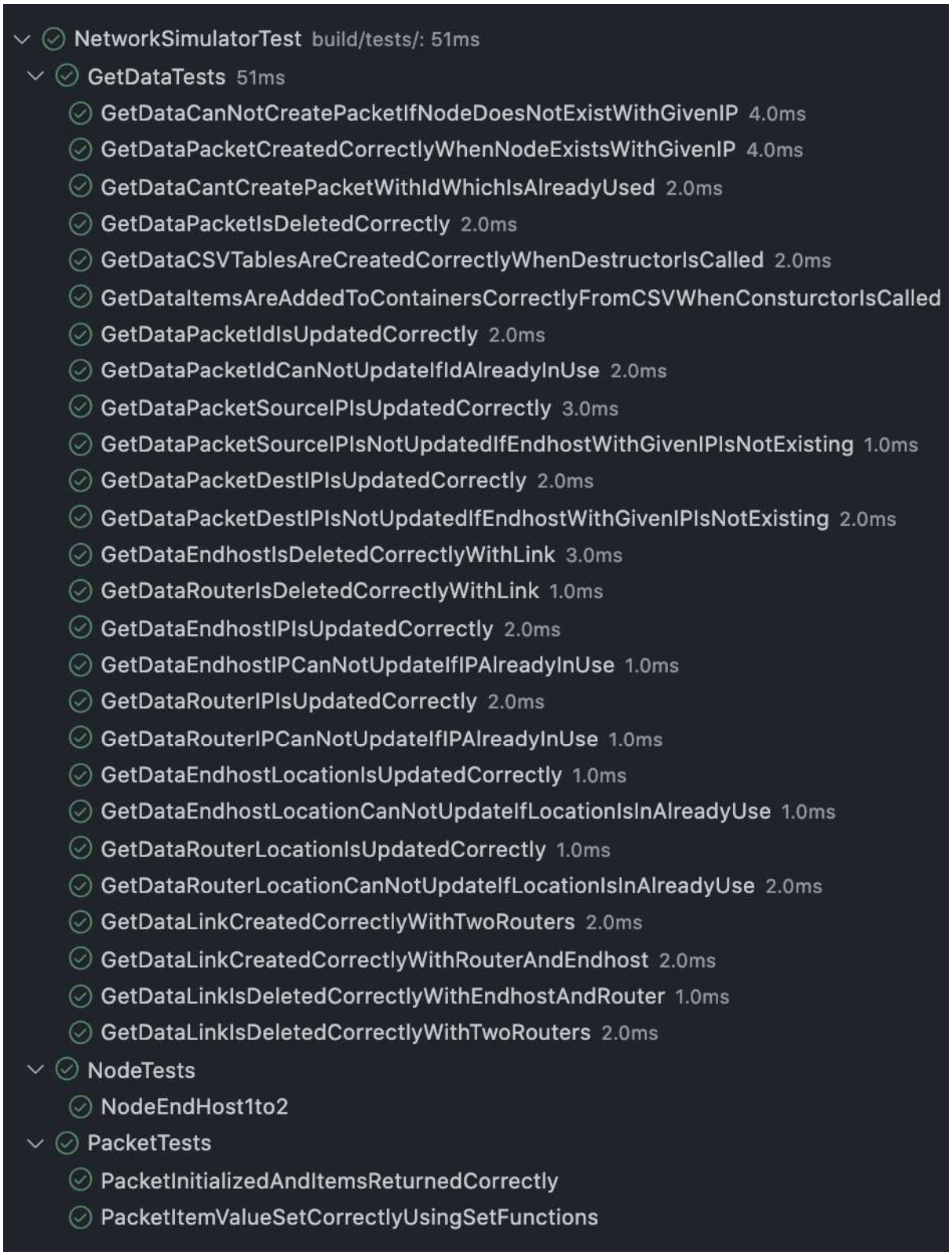

Figure 17. Endhost, GetData, Packet, Link class passing unit tests

Router class and Dijkstra class testing (dijkstratest branch):

The routing table was tested with a script which called the getData method, then generated the paths calling the Dijkstra routing algorithm which inserted each row of the table, described in the previous section, in every router object. Nevertheless, the testing of the routing table did not succeed in handling routes which had same source and destination.

4.2 Frontend testing

Frontend was tested by running a variety of scenarios, where different number of packets were sent from one endhost to another. The error handling is not the most graceful if something goes wrong, because of the complexity of error handling in Qt. There are several problems with the simulation in the frontend, especially when finding complex paths where a single router is not connected to a multiple router. An example of a scenario which does not work is the following (figure 17):

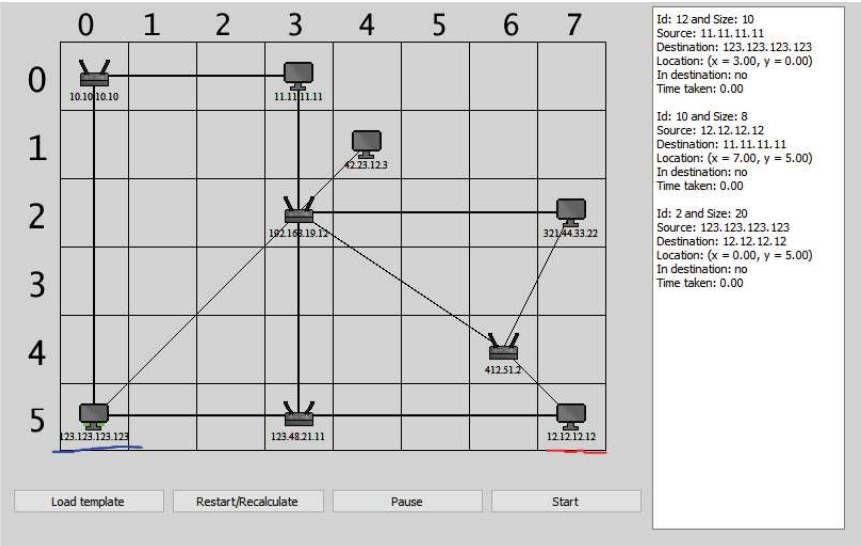

Figure 17: Here we can see the blue marked as the source node, and red as the destination (packet id 2).

The reason why the simulation is figure 17 does not work for packet id 2 is because the path calculated first tries to utilize the path 10.10.10.10 router, and when it realizes that it cannot find a path that way, it reverts to another one. When it tries to execute that path, the old router (10.10.10.10) has not been deleted, and the path created does not compile successfully. However, if we were to create a situation like figure 18.

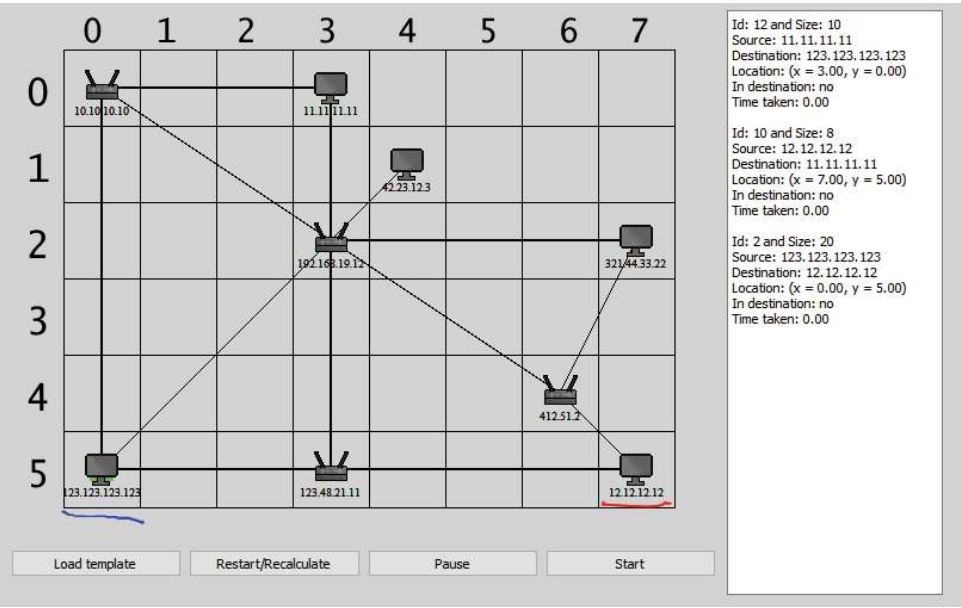

Figure 18: A scenario where the packet routing works

In figure 18, the router path is not punished for adding that router, and while the path might not be optimal, the packet eventually arrives in its destination. If the simulation worked recursively, this could have been avoided, since in that case the router would have returned only a singular path. The reason why it was not implemented that way was the fact that the backend routing did not work successfully, so we had to utilize the frontend mock-up simulation, which was only meant to be place holder before the actual routing system could be used.

5. Work log

Work log: a detailed description of the division of work and everyone's responsibilities. For each week, a description of what was done and roughly how many hours were used for the project by each project member.

Week (Date)	Lari (Task/Time)	Juhan (Task/Time)	Selim (Task/Time)	Rafael (Task/Time)
Week 44 01.11-7.11	Created the project plan with other team members. **5 hours**			
Week 45 8.11-14.11	Set up the basic environment, created node, endhost and link classes. **5 hours**			
Week 46 15.11-21.11	Combined packets to work with the node class, and implemented some unit testing. **6 hours**			
Week 47 22.11-28.11	Started working on the main page, implemented the basic layout, textures and widgets **10 hours**			

Week 48 29.11-05.12	Added packet simulation to the frontend and animations for the process. **14 hours**
Week 49 06.12-12.12	Implemented packet statistics screen, improved the simulation for the frontend, fixed multiple bugs in the frontend implementation. Wrote my part of the final report. **20 hours**