Graphical Network Simulator

Christos Constantinides
H00113142

August 2012

Computer Science
School of Mathematical and Computer Sciences

Dissertation submitted as part of the requirements for the award of the degree of MSc in Software Engineering
DECLARATION

I, Christos Constantinides confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: .................................................. Date: ...........................................................
Abstract

The growth of the internet, and evolution in telecommunications and mobile devices, increased the need for research, education and experiment with networks. In order to achieve these in an easy and costless way, network simulators were developed. Currently there are many simulators available for researchers and students, although most of them seem to have a complicate setup and they are not very effective in visualizing the results in an easy to understand manner. Moreover most of them require a specific operating system to run as a consequence their usage is limited to a specific number of users. In this project an existing multiplatform simulator JNS [1] was taken and integrated with an easy to use graphical user interface to bring usability in all platforms, easy setup of simulation properties and scenario, and nice visualization – animation of the results. An existing visualization system JAVIS [4] (was originally Developed at the University College of London) was used. JAVIS [4] was modified, redesigned and attached to the JNS [1] to provide a graphical animation in package level. The new Graphical Network Simulator provides a GUI for designing a network simulation and scenario and a very detailed animation of packets together with annotation text, and some statistical data as text view and bar graphs.
Acknowledgements

Firstly I would like to thank my supervisor Dr Peter J.B. King for the faith he showed on me, for giving me the opportunity to work on this project and for his overall guidance. I would also like to say a big thanks to my family for their love and their financial support on attending this MSc. Moreover I would like to thank namely: Nicolas Zahariades, Sotiris Achileos, Stavros Katsambis, Neoklis Hadjigeorgiou, Valentinos Papasavva, Giannis Christodoulou and Constantinos Constantinides for their time spent on testing my application occasionally, and for the bugs that they found. Finally I would like to thank my fellow students who participated in the evaluation process of this project.
# Table of Contents

1. Introduction .................................................................................................................. 9
2. Aims and Objectives ....................................................................................................... 9
3. Project Description ......................................................................................................... 10
4. Requirements Analysis ................................................................................................. 10
   4.1. Basic Requirements ............................................................................................... 10
   4.2. Secondary Requirements ....................................................................................... 11
5. Professional, legal, and ethical Issues .......................................................................... 11
6. Literature Review .......................................................................................................... 12
   6.1. Graphical User Interfaces and Animation in Network Simulation ....................... 12
   6.2. JAVIS .................................................................................................................... 13
   6.3. Existing GUI and Visualization Tools for Network Simulations ......................... 15
      6.3.1. (NSG2) Network Simulator Generator 2 ....................................................... 15
      6.3.2. (NAM) Network Animator ........................................................................... 17
      6.3.3. Otter: A general-purpose network visualization tool .................................... 18
      6.3.4. NetAnim ......................................................................................................... 19
      6.3.5. NetExplorer .................................................................................................. 20
      6.3.6. Visual Animation and Editor ......................................................................... 21
      6.3.7. (GNS3) Graphical Network Simulator 3 ....................................................... 22
      6.3.8. Visual Network Simulator ............................................................................. 24
      6.3.9. Visualization and Analysis Tool for Wireless Simulations: iNSpect ............... 25
      6.3.10. GloMoSim ..................................................................................................... 27
      6.3.11. QualNet ......................................................................................................... 28
      6.3.12. G-JSim .......................................................................................................... 29
6.4. Summary .................................................................................................................... 31
6.5. Discussion .................................................................................................................. 32
7. Development Methodology ............................................................................................ 33
8. Development Tools ....................................................................................................... 35
9. JAVIS Analysis .............................................................................................................. 35
10. Graphical Network Simulator ..................................................................................... 39
   10.1. Changes or Improvements from the original JAVIS code .................................. 39
      10.1.1. Refactoring Process ..................................................................................... 39
      10.1.2. Multi format Annotations ............................................................................. 45
10.1.3. Resizable annotation panel ................................................................. 45
10.1.4. More Detailed Animation ........................................................................ 45
10.1.5. Wider animation speed Range ................................................................. 46
10.1.6. Animation events data structure ............................................................ 46
10.1.7. Animation History .......................................................... .......................... 46
10.1.8. Simulator Seed Number ........................................................ ................. 47
10.2. Adaption of JNS ..................................................................................... 47
10.3. New Features ......................................................................................... 48
10.3.1. Visual editing ......................................................................................... 48
10.3.2. Default Values ....................................................................................... 51
10.3.3. Zooming Capabilities ........................................................................... 53
10.3.4. Hand Mode ............................................................................................ 54
10.3.5. Group Selection and Movement ............................................................ 55
10.3.6. Statistics and Bar Graphs ....................................................................... 56
10.3.7. Scenario Form ....................................................................................... 58
10.3.8. Terminal ................................................................................................ 59
10.3.9. Status Panel ......................................................................................... 60
10.3.10. Frame title with opened filename ....................................................... 60
10.3.11. Save /Restore simulation configurations in XML format ...................... 61
10.4. Design Considerations ........................................................................... 61
10.4.1. Graphical User Interface .......................................................... .................. 61
10.4.2. Events ................................................................................................ 65
10.4.3. Network Components .......................................................................... 65
10.4.4. Attributes forms ................................................................................... 67
11. Testing ....................................................................................................... 67
12. Evaluation ................................................................................................ 70
12.1. Evaluation Method/Setup .......................................................................... 70
12.2. Evaluation Results ................................................................................... 71
12.3. After Evaluation Improvements .............................................................. 74
13. Conclusion ................................................................................................ 75
13.1. Future Work ............................................................................................ 76
References ................................................................................................. 79
Bibliography ............................................................................................... 81
APPENDIX A : The Initial plan of the Project .................................................. 82
List of Figures

Figure 1: Screen shot of JAVIS Network Animator ................................................................. 14
Figure 2: Screenshot of NSG2, Script on the Left and Network topology on the right ............. 16
Figure 3: Screenshot of NAM (Network Animator) .................................................................. 17
Figure 4: Network topology generated by Otter ....................................................................... 18
Figure 5: NetAnim, Packet animation. .................................................................................... 19
Figure 6: NetExplorer screenshot. .......................................................................................... 20
Figure 7: An example of existing animation on FTP Basics .................................................... 22
Figure 8: A screenshot of GNS3 ............................................................................................. 23
Figure 9: Screenshot of Visual network Simulator .................................................................. 25
Figure 10: Animation of iNSpect. .......................................................................................... 26
Figure 11: Animation of a wired network. .............................................................................. 28
Figure 12: G-JSim form for target’s node configuration ......................................................... 30
Figure 13: Group selection and movement ............................................................................ 55
Figure 14: Bar graph .............................................................................................................. 57
Figure 15: Main GUI of Graphical Network Simulator ............................................................ 61
Figure 16: File menu .............................................................................................................. 64
Figure 17: Options Menu ....................................................................................................... 64
Figure 18: Class diagram of Events ...................................................................................... 65
Figure 19: Class diagram of network elements. .................................................................... 66
Figure 20: Class diagram of attributes forms ....................................................................... 67
Figure 21: Difficulty Evaluation results .................................................................................. 71
Figure 22: Evaluation Results - Useful for educational purposes ............................................. 72
Figure 23: Evaluation Results - Useful as a developing platform ........................................... 72
List of Tables

Table 1: Features of JAVIS ................................................................. 14
Table 2: Features of NSG2 ............................................................... 16
Table 3: Features of NAM ................................................................. 18
Table 4: Features of NetAnim. ......................................................... 20
Table 5: Features of NetExplorer .................................................... 21
Table 6: Features of Visual Animation and Editor .......................... 22
Table 7: Features of GNS3 ............................................................... 24
Table 8: Features of Visual Network Simulator .............................. 25
Table 9: Features of iNSpect ........................................................... 27
Table 10: Features of GloMoSim. .................................................... 27
Table 11: Features of QualNet .......................................................... 29
Table 12: Features of G-JSim ........................................................... 30
Table 13: Summary table ............................................................... 31
Table 14: Testing tasks ................................................................. 69
Table 15: Investigated Features ...................................................... 76
1. Introduction

A lot of Network simulators exist, but most of them require extra effort and time to read their manuals and documentation to use or to find out their capabilities. Other simulators require the knowledge of a programming language for setting up their input parameters. Students learning networks don’t have a clear image of what actually happened in a network communication. Moreover, most advanced users write their new protocols or applications but then they have no clear way to test and observe how exactly the communication takes place. In summer 2008 an MSc student submitted a Java-Based Simulator for educational purposes (JNS [1]). JAVIS [4] is an animation tool that was used aside with NS-2 [22] simulator to provide some visualization of results on package level. Our first goal was to summarize all the functionality that JNS [1] offers into an easy to use graphical user interface. Secondly after some modification of JAVIS [4] we provided a detailed animation of the simulator’s results.

2. Aims and Objectives

The aim of this project is to modify an existing visualization system JAVIS and integrate it with an existing modified version of JNS [1] in order to enable visualization on simulations. It is needed to extend JAVIS system in order to provide an easy to use Graphical User Interface to give the user the ability to set parameters as input for the simulation and then visualize the output.

Objectives for this project are:

- Modify JAVIS and the JNS to allow visualization of the results of each simulation
- Modify JNS results to adapt them with JAVIS.
- Extend JAVIS in order to enable user input for all the parameters implemented in the JNS simulator.
- Ability to save/read into/from a file a simulation configuration in XML (Extensible Markup Language) format.
3. Project Description

JAVIS is an old work of University College of London in Department of Computer Science. It is an open source object-oriented visualization tool for simulations in java. At the time, the main reason for the implementation of JAVIS was to be a multiplatform clone of NS/NAM [5] (Network Animator) visualizer tool which was written C++. JAVIS inherits most of the features that NAM had such as the simulation annotation TCL (Terminal Command Line) expressions in trace files. NS (Network Simulator) and NS-2 [22] later, used to export trace files that could be imported by the NAM visualizer tool in offline mode (manually). JAVIS currently takes as input most of the trace files generated by the NS. JNS (Java Network Simulator) which is the java implementation of NS-2 was taken by [1] and extended, in order to support more features, make the simulation more realistic and simpler, provide to the university a viable simulation environment for educational purposes that can be ran either independently or as a basis library for future work. The main goal of this project is to provide user friendly setup of each simulation on the new JNS and also visualize the results of it. We went through existing simulators and looked at features they provide in order to summarize their advantages and hopefully be able to provide something better. For the purposes of this project both JAVIS and JNS where used.

4. Requirements Analysis

4.1. Basic Requirements

Multiplatform: The Graphical network Simulator must be able to run in all platforms (Windows, Linux, MacOS). Although java will be used, any other library or sub program that might be needed must be multiplatform as well.

Extensible simulator: It is important to keep the extensibility and scalability of the simulator and MVC (Model View Controller), Observer, Singleton or any other patterns will be used if necessary. The final deliverable should be easy to be extended with new simulation features and network protocols.
Graphical user interface for the input: The user must be able to set all parameters of the simulation via a user friendly graphical interface.

Graphical user interface for the output: The user must be able to observe the simulation results through the graphical user interface.

Ability to import/export simulation configurations: The user must be able to save or load a simulation configuration for an easy setup and run of the same experiment in different time. The configuration file must follow the XML format.

Ability to animate results: The results should be animated in the graphical user interface.

4.2. Secondary Requirements

Immediate feedback from the simulation: Ability to set the simulation into a mode, where each change on the parameters will cause the immediate execution of the simulator and the animation of its results.

Pause/resume the animation: Enable to the user the ability to pause and resume the animation.

Step forward the animation: Provide a step forward button to let the user go to the next event of the animation when needed.

5. Professional, legal, and ethical Issues

The key factors for each project are to avoid plagiarism. Everything written in this report cited to external sources is referenced. For this project two open source projects will be used. Any other extra libraries or tools that might be used will be referenced. Otherwise there are no other Professional, Legal or Ethical issues.
6. Literature Review

6.1. Graphical User Interfaces and Animation in Network Simulation

Network Simulators are widely used in the laboratory for experiments but also for educational purposes in Universities. Most of them are difficult to setup and operate while others results are illegible, and the simulation is difficult to be observed. Especially students need a better representation of results instead of log files, in order to understand deeper the theory of networks. Students and researches working in the labs need to be able to easily setup and in a matter of minutes a simulation, view the results through the GUI and an animation of what actually happened. “A picture is worth a thousand words”. “Visual depictions of graphs and networks are external representations that exploit human processing to reduce the cognitive load of a task.” [2]

The three stages of a Simulation process:

- **Input**: Configuration of the simulator, Specification of the simulation scenario.
- **Simulation**: Simulator processes the given input.
- **Output**: Animation, Console, Log files, Statistics and Graphs.

Connection of GUI tools with Simulator:

**Offline GUI**: The GUI that works as independent system from the simulator. It requires manual input of the simulator generated output files.

**Online GUI**: The GUI that does not require manual manipulation of trace files by the user. Everything can be done with use of GUI components.

**Live**: Is the GUI where the animation is projected while the simulation is performed (at the same time). This can be done with a complete integration of an animator with the simulator (one executable) or with a server-client method as two processes that exchange messages.
**Integrated-Not live:** GUI for input, Simulator and Visualizer are a single executable and no trace files are used for any animation.

### 6.2. JAVIS

Current version of JAVIS can read NS-2 [22] trace files and preview an animation in package level. Although there is information about node location in the trace files, it is ignored by JAVIS and currently there is no manual positioning of nodes. Instead it applies a version of force-optimization layout algorithm several times in order to find a good place for each node (no nodes are overlapping and all nodes are placed in the centre of the screen). At first the nodes are placed randomly in the plain and afterwards the force-optimization layout algorithm is applied, given the current position of each node. For this algorithm two numeric values can be set by the user from the GUI. The first one is the tension force between each two nodes and the second one is the number of how many times the algorithm will be applied. Upon the time that nodes have been placed on the panel, there is an option of a re-layout button that can re-apply the algorithm and an option of a shake button that will randomly change the nodes position a little bit. There is no any other manual or direct way to set the position of a node on the panel. Nodes of network can be represented as shapes (circle, hexagon or square) and they can also have multiple colours. JAVIS by itself does not distinguish hosts, routers or switches. It’s the trace file’s job to contain valid information and specify what shape or colour a node must be. Links are represented as straight lines with nodes at both ends. Link type (simplex or duplex) does not exist and link’s colour can be specified again from the trace file. The link’s bandwidth and delay are stored and used for the animation while calculating the speed the packet will travel graphically on the line, but they are not shown to the user. After the animation begins, the speed of each link can be observed as flowing packets move faster over faster links and vice versa. Packets are represented as boxes (with arrows showing the direction) and their width is different according length of the transferring data. Their colours can be specified again from the trace file according the packet type. The user has also the ability to add new annotations after the
file is loaded and save them back to the trace file. Usually this is used when the simulator’s annotations aren’t that rich so the user can add more and make the animation more understandable for others. It is well designed and implemented, with use of software engineering patterns that make it easily extensible.

![Screen shot of JAVIS Network Animator](image1)

**Figure 1: Screen shot of JAVIS Network Animator**

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI for the Input</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Manual Topologies</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1: Features of JAVIS**
6.3. Existing GUI and Visualization Tools for Network Simulations

In this section existing free or commercial software will be investigated and summarized. Some of them are tools to create simulation topologies and scenarios, while others are developed for network animation or to just present simulation data in a better, more readable and meaningful way.

6.3.1. (NSG2) Network Simulator Generator 2

NSG2 [13] is a free java software maintained by Peng-Jung Wu with the help of Prof Chungnan Lee. NSG2 is a graphical user interface where the user can design a topology using the mouse and set properties for a simulation scenario and then generate the TCL script that can be the input for the NS-2 [22] simulator. It supports wired and wireless networks, duplex and simplex links, agents (TCP and UDP), applications using FTP and CBR and the ability to move around the nodes with the mouse. This is a simple and clear way of producing the input for a simulator. A drawback is that it requires everything to be designed by hand. In cases where we would like to run for example a simulation with some hundreds of nodes the only way to produce it is by hand and that would be really time-consuming. In addition it works as a separated program from the simulator, were you have to save the TCL into a file and then run the NS-2 simulator and load it manually. The use of NSG2 does simplify the process to run a simulation in NS-2 a lot, but there is a lot of space for improvement left (Such as to allow exportation of a configuration file). It is a tool that provides graphical user interaction for the input of a simulation, and not the output.
Figure 2: Screenshot of NSG2, Script on the Left and Network topology on the right.

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GUI for the Output</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Network Animation</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 2: Features of NSG2
6.3.2. **(NAM) Network Animator**

NAM [5] is one of the most widely used visualization systems for the NS and NS-2 simulators. It is written in C++ and is mainly supposed to run on Linux machines, but earlier versions also run in windows with a provided binary executable. It can read trace files that are the output of NS / NS-2 and show an animation of the simulation in package level. For the nodes placement it uses a forced-directed layout algorithm: “First, place nodes randomly on a plane. Calculate the shortest path between any two nodes in the graph. Attach a spring with that length between every pair of nodes. On each iteration, release the springs and let the nodes fall into their lowest energy state” [23]. With this algorithm it can position a few hundreds of nodes, successfully. There is also the option to specify the position of each node by specifying the angle and the length of each link. JAVIS, which we are going to use, is the java implementation of NAM. NAM is only designed for visualizing the results of NS, NS-2 simulation and not for giving input to the simulation. With NAM you can play/stop the animation, step forward, backwards, view messages in the notification area and increase decrease the time step of each iteration of the animation.

![Figure 3: Screenshot of NAM (Network Animator)](image)
### Otter: A general-purpose network visualization tool

Otter [3] is a historical general purpose network visualization tool developed at CAIDA and published in 1999. It was a tool that just visualises the topology of a network. What is interesting about Otter is the quasi-geographical algorithm that it uses to spread the nodes and link them together. It could represent nodes, root nodes, links and paths. It could draw a network of thousands of nodes in a few minutes, and it allowed the user to change the positioning of each individual node or a group of nodes with the mouse. Another feature it had was the ability to change a client node into a root node dynamically (and vice versa) and this would cause an instant refresh of the topology using the quasi-geographical algorithm.

![Network topology generated by Otter. [3]](image)

---

**Table 3: Features of NAM**

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI for the Input</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Manual Topologies</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

---

**6.3.3. Otter: A general-purpose network visualization tool**
6.3.4. NetAnim

NetAnim [7] (Network Animator) is a network animator for the NS-3 [24] (later version of NS-2) simulator. It is based on multiplatform Qt4 toolkit and written in C++. It supports wireless and wired nodes and mobile nodes (moving nodes). It takes as input a trace file in XML format from the NS-3 simulator (offline) and it can animate the simulation in package level. In contrast with NAM, the XML trace file contains the time that the first byte of a package was transmitted and the time that the last byte of the same package was received (for each transmission). This gives to the animator the decision of how many frames to draw for each package transmission. This feature enables the animation of the real world transmission. If you had a hypothetical transparent cable and you could see the packets going through, that’s the speed that they are shown in the animator. Due to fast medium and fast transmissions in wireless networks, some packets are not shown, or they appear and disappear too fast in the animation. For that reason its developers included a slider in the GUI to slow down the animation at will. Package animation has the same appearance with NAM. The user also can resize nodes and set a grid behind the animation. It is a very recent tool with development in progress and promises GUI for designing networks as well.

![NetAnim, Packet animation. [7]](image)
### Features of NetAnim

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Features of NetAnim.

In addition to the standard version of NetAnim there is the choice of NetAnim Advanced Mode [25]. It requires changing a specific configuration file and recompiling the program by including the NS-3 libraries. NetAnim Advanced Mode performs the animation at the same time with the simulation and also the user can pause / resume the simulation. In addition to this it provides some more thorough log files, statistics and some graphs on how many packages passed from each node at a specific time. However the Advanced mode is still in early stages (experimental) and it is not recommended for normal use.

#### 6.3.5. NetExplorer

NetExplorer (Network Explorer) [8] is a resent project hosted on GoogleCode and it is another graphical animator for the NS-3 [24] Simulator. It is compatible with NetAnim, in other words it can read the same trace files. It is written in C++ and supports only Linux distributions with Gnome as desktop environment. Its main functionality is to animate the trace file exported by NS-3. A nice feature that others don’t have is

![NetExplorer screenshot](image)
that you can replay a part of the animation using the slider at the bottom. The GUI looks familiar as it reminds a video player, thus it is straightforward how to be used by anyone. At the moment there is no documentation about it.

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI for the Input</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Manual Topologies</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Ability to Save/load a</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Features of NetExplorer

6.3.6. Visual Animation and Editor

Visual Animator and Editor [6] is a free to use webpage that hosts a network animator. There are many existing examples hosted where the student can watch and learn from them, but also a tool is provided in order to design your own network animations (requires free registration). It can also animate trace files from Dynamips (router emulator) [26] and NS-2. Everything is written in flash and is not open source. This simulation platform focuses more on education. The scenario of each simulation is divided into several titled parts and the user has the ability to run each part individually with “Prev”, “Next” buttons. Concerning animation, each node is represented by an icon of a specific device with text fields for their names and IP inside the animation. The whole animation is very representative, and really helps the students to understand how networks work.
The GUI provided to create your own simulations uses the drag and drop method, you can drag from a toolbox with nodes and links to choose from and drop them into the plain. Then by right clicking on each node or link you can specify its characteristics, and scenario. It also allows the user to save a simulation locally in his machine, although the format is unreadable by any other software or editor (binary file).

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Network Animation</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Features of Visual Animation and Editor

6.3.7. (GNS3) Graphical Network Simulator 3

GNS3 [9] is a Graphical Network simulator that runs on top of virtualization technologies and emulation software. It is open source, software completely written in python and it comes precompiled for all platforms (Windows, Linux, and MACOS). In GNS3 each node can
be an emulated: router, bridge, switch or a virtual machine (host, server). Each router node uses the Dynamips [26] router emulator where the user is free to mount any of the supported Cisco IOS firmware images (not provided). Then with ssh (or telnet) the user can boot and setup the router via the command line. Each host/server node can be either a virtual image of VirtualBox [27] or Quemu [28] virtualization software. GNS3 Provides slots for each node and from the graphical user interface the user can specify links between nodes and in which slots they are connected. There is no limit in what can be simulated in GNS3 as everything that exists in the real operating systems and real routers can be tested, from the network layer up to the application layer. On the other hand, the number of nodes is limited to the machine’s physical resources (Hard Disk, RAM, and CPU). There are some utilities provided to reduce the usage of RAM and CPU, but still the number of nodes can be few dozen on a decent today’s computer hardware. The GUI provided is simple to use but it is only used to design the topology. There is no animation provided (not the aim of GNS3), and events of a simulation can be observed by either the virtual machines with any network monitoring tool, either by the routers emulator console for each node independently.

Figure 8: A screenshot of GNS3 [9]
<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Ability to Save/load a</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td></td>
<td>✔  (console)</td>
</tr>
<tr>
<td>Network Animation</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Table 7: Features of GNS3

6.3.8. Visual Network Simulator

VNS (Visual Network Simulator) [10] is another graphical user interface developed in java and supports the NS-2. With VNS the user can create topologies and the scenario of an NS-2 simulation and export the script file. The user has to manually save the script file and then import it in the NS-2. This interface supports only wired networks. There is no documentation currently available, and the GUI and its usage are simple but not straightforward. Buttons do not have tooltips and for some buttons the functionality is not so obvious (trial and error method). It allows the user to change the position of each node but it requires to first selecting the type of node you are going to move. For example if the button “router” is pressed then you are only allowed to change the position of routers, and right click only on routers. Set of properties on each node or link, can be done by right clicking on the node or link and then by choosing setup, but again it requires that you press the button of that type of component first. The plain zoom level can be change and the default starting zoom position is in place that you have enough room to un-zoom if the designed network needs more space later. Finally, it provides the feature to save a configuration into an xml file for later use.
6.3.9. Visualization and Analysis Tool for Wireless Simulations: iNSpect

iNSpect [11] [12] [14] is a free and open source visualization and animation tool for NS-2 [22]. It is written in C++ based on OpenGL and it is designed to animate wireless networks but it can also be used for wired networks as well. As input it can take the mobility file (input file for NS-2) and the trace file generated from NS-2 as output. The mobility file is used to find the initial position of mobile nodes, and animate when they move or when they are...
stopped; as such information is not kept in the trace file generated by NS-2. It contains a part of the NS-2 code to simulate the movements of the nodes just like NS-2 does. Wireless new routes and transmissions are animated when they occur via lines and arrows starting from the sender and pointing to the recipient node. In each transmission, while the transmission takes place, the sender node becomes blue and the recipient node turns to red until the transmission ends. Upon successful transmission, if the recipient node is an intermediate node it becomes yellow, otherwise if it is the destination node it turns green. In case of unsuccessful transmission, the recipient remains red. With iNSpect there is also the ability to set overlay areas, or obstacles in the shape of circle or rectangle with the help of an XML configuration file. This can make moving nodes reaching “out of signal” areas or can cause transmission failures due to obstacles. In addition the user can choose via the configuration file to show coordinates of each node. With such feature more detailed observation can be made by the user in mobile node movements in a simulation. Finally, iNSpect is used as an evaluation / validation tool for new wireless protocols developed and for future versions of simulators [12].

Figure 10: Animation of iNSpect [12].
### 6.3.10. GloMoSim

GloMoSim[15] (Global Mobile Information System Simulator) is a parallel network simulator for large scale wireless networks. It provides a visualization tool for debugging or verifying the simulation protocols or simulations implemented by the user. In GloMoSim’s GUI each mobility group is represented by different colour and the visualization provided is in packet level. In addition the user can view statistics of each simulation. Moreover the user can pause, resume or step through the animation. When a transmission occurs yellow links are drawn from the sender node to all nodes in its signal range. When the transmission finishes each yellow link is transformed to green if the particular transmission was successful, or to red if it failed [16]. All type of packets supported by the GloMoSim are represented the same by the Visualizer.

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✔</td>
<td>(copyright)</td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Features of iNSpect

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Features of GloMoSim.
6.3.11. QualNet

QualNet [18] is a commercial Network simulator based on GloMoSim [15]. It uses Java for the GUI and the simulator is C-Based optimized and targeted for each platform (LINUX, Windows). QualNet Developer consists of several tools [18]:

- Scenario Designer: To set the parameters required as input of the simulator
- Animator: To animate results of a simulation: It shows packets passing through several network layers, plots dynamic changing graphs and it can run a second simulation aside (with the feature “what-if”) enabling comparison between the results of two simulations.
- 3D Visualizer: Provides a more advanced 2D and 3D representation of topology, events, and statistical data.
- Analyzer: It collects metrics while the simulation is running and then displays them in the form of reports or graphs.
- Packet Tracer: Displays packets over transmissions.
- Command line interface: Alternative access to the simulator.

Figure 11: Animation of a wired network [18].
In Figure 11 there is a screenshot of the Animator tool. It shows a Y network with 4 nodes. The two nodes with id 1 and 2 are sending packets to 3 in order to root them to node 4. Due to the fact that the bandwidth is slow between 3 and 4, a queue grows on node 3 (red lines).

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✓ (Linux, Windows)</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td></td>
<td>✓ (Commercial)</td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Network Animation</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Table 11: Features of QualNet.

6.3.12. G-JSim

G-JSim [20] is a graphical user interface written in java for the J-Sim [21] (Java Simulator) network simulator. J-Sim takes as input a TCL script with the simulation configuration in order to work. G-JSim’s main objective is to make J-Sim more easily accessible by other researchers or students that don’t know the TCL language. In order to achieve that it provides a user friendly interface with forms for specifying network parameters. Afterwards it automatically generates the TCL script and forks a process of J-Sim to begin the simulation. The user does not need to manually handle the TCL script file for the simulation to begin, as everything is done by the GUI. G-JSim can read or write xml configuration files and TCL scripts. In figure 12 we can see a form for configuring the target’s node parameters for a simulation.
Figure 12: G-JSim form for target’s node configuration. [20]

<table>
<thead>
<tr>
<th>Features</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiplatform</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Open Source</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI for the Input</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Manual Topologies</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Automatic Topologies</td>
<td>✓ (random positioning)</td>
<td></td>
</tr>
<tr>
<td>GUI-Simulator connected</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ability to Save/load a Configuration</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>GUI for the Output</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Network Animation</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 12: Features of G-JSim.
## 6.4. Summary

<table>
<thead>
<tr>
<th>Features</th>
<th>Multiplatform</th>
<th>Prog. Lang.</th>
<th>Open Source</th>
<th>Type</th>
<th>GUI to make sim. scenario</th>
<th>Topologies</th>
<th>GUI for results</th>
<th>Save/Load configuration file</th>
<th>Network Animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSG2</td>
<td>Yes</td>
<td>Java</td>
<td>No</td>
<td>Offline</td>
<td>Yes</td>
<td>Manual</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NAM</td>
<td>No</td>
<td>C++</td>
<td>Yes</td>
<td>Offline</td>
<td>No</td>
<td>Auto + Manual</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Otter</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>Auto + Manual</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NetAnim</td>
<td>No</td>
<td>C++ Qt</td>
<td>Yes</td>
<td>Offline</td>
<td>Yes</td>
<td>Manual</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NetExplorer</td>
<td>No</td>
<td>C++</td>
<td>Yes</td>
<td>Offline</td>
<td>No</td>
<td>Manual</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Visual Animator and Editor</td>
<td>Yes</td>
<td>Flash</td>
<td>No</td>
<td>Offline</td>
<td>Yes</td>
<td>Manual</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GNS3</td>
<td>Yes</td>
<td>Python</td>
<td>Yes</td>
<td>Online</td>
<td>Yes + Command line</td>
<td>Manual</td>
<td>Console</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Visual Network Simulator</td>
<td>Yes</td>
<td>Java</td>
<td>No</td>
<td>Offline</td>
<td>Yes</td>
<td>Manual</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>iNSpect</td>
<td>Yes</td>
<td>C++ OpenGL</td>
<td>Yes</td>
<td>Offline</td>
<td>No</td>
<td>Manual</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>GloMoSim</td>
<td>Linux, Tested on Windows NT</td>
<td>C++</td>
<td>No</td>
<td>Offline</td>
<td>No</td>
<td>Manual</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>QualNet</td>
<td>Linux, Windows</td>
<td>C-Based, Java GUI</td>
<td>No</td>
<td>Online</td>
<td>Yes</td>
<td>Manual</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>G-JSim</td>
<td>Yes</td>
<td>Java</td>
<td>Yes</td>
<td>Fork method</td>
<td>Yes</td>
<td>Random or manual</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 13: Summary table

**Parsec is a C-based simulation language, developed by the Parallel Computing Laboratory at UCLA, for sequential and parallel execution of discrete-event simulation models. It can also be used as a parallel programming language.**[17]
6.5. Discussion

According to the simulation tools summarized in Table 13, currently there are no multiplatform, open source tools that combine graphical user interface for setting the simulation and visualization of the results. Some tools claim that are multiplatform but actually they depend on multiple packages instead of just a virtual machine and require specific version of components in order to compile and run. Some tools are designed for Linux but they depend on specific version of other packages to build and run. Most distributions of Linux stop supporting old versions after few years; as a result tools that its development stopped cannot be used in the future. As a result these tools will be dropped from machines in the future. In addition if a second or third dependency software is needed in order to use the simulator in multiple platforms maybe that tool stop its support for multiple platforms or switch from free to a commercial tool as a consequence the decreased usage of that specific simulator.

Some of the applications we investigated are tools that help in the configuration of a simulation while others are meant to provide an animation or statistics of the results. We can clearly define from our results that there is a need for an open-source, multiplatform Graphical Network Simulator that provides easy configuration and way to design the network as well as a detailed animation at packet level afterwards and results, while everything should be attached on a real network simulator. Some nice features that could be inherited are drag and drop capabilities to design the network, specifying a scenario using forms or terminal, the ability to save and load a configuration, ability to pause/resume/replay the animation, control the animation speed and provide some results in a form of statistical data.
7. Development Methodology

To achieve the requirements of this project an incremental methodology followed. The reason why is because in this project some decisions were needed to be taken after a closer look to the current code and after some work was done. An example is the secondary feature of immediate execution of the simulation and feedback visualization for each change in the parameters. (Without the need to press any start button). Due to long animation time (from few seconds to couple of minutes) this feature was considered as inappropriate. It wasn’t expensive in resources though.

Another important thing that made the process anti-waterfall was the initial decision to provide two deliverable versions of Graphical Network Simulator. The first version would be a complete integration of JAVIS [4] and JNS [1]. Although software engineering patterns and high quality programming would be used the JNS probably would lose its scalability. In order to avoid that, a second version of deliverable considered to be the solution, that the graphical user interface would be connected to the JNS using trace files. Finally after some further reading and understanding of JNS and JAVIS we decided to make JNS generate animation events that would be handled by the animator in such a way that the scalability of JNS would be kept while integrated to the new Graphical Network Simulator.

Development Increments/Iterations:

1. Collection of characteristics of Host, Router, Switch ,Duplex Link, Simplex Link from the JNS simulator and creation of main classes that would hold the network in GUI.
2. Creation of panel with buttons that allow setup of the network.
3. Creation of the designer controller and creation of actions to those buttons.
4. Implementation of the GUI model class, for adding hosts, routers, switches and linking them.
5. Implementation of feature that allows clicking and dragging nodes.
6. Implementation of forms to fill the attributes of nodes and links.
7. Implementation of feature that allows read/write a configuration from/to xml.
8. Creation of classes that build simulation and run the simulator.
9. Testing process with the results log files of the JNS using the GUI as input and comparison with the log files results when the simulation is written manually in java.
10. Attempt to generate naïve trace file and use with the current JAVIS system. FAILED: There is not enough information in the log files to generate a complete trace file.
11. Importation of JNS into the current project in a separate package.
13. Changes applied and new code to provide “hop” and “text” events to JNS.
15. Changes applied to provide zooming capabilities in the drawing panel.
17. Development of classes that automatically generate hardware and IP addresses.
18. Provide Default Values.
19. Creation of node group selection and move.
20. More changes and new code applied to provide “en-queue”, “de-queue”, “drop” animation events to JNS.
21. Changes applied to provide a more detailed animation and multi-formatted annotations.
22. Creation of a helper class to collect statistical data from the JNS.
23. Implementation of a Bar Graph class.
24. Changes applied to integrate statistical data with Bar Graphs and in text view.
8. Development Tools

The tools used for this project were the following:

- Eclipse IDE for Java Developers
- JavaSE-1.6
- Enterprise Architect (UML Diagrams)
- Gimp 2.0

9. JAVIS Analysis

JAVIS consists of the following seven packages with their classes:

- **Package animation**
  - **Interface Animation.java**: The interface that any other class can communicate with the class that will implement this interface.
  - **Link.java**: Link network component. It implements the Visual element interface.
  - **Node.java**: Generic node component. It does not distinguish hosts from routers or switches. It implements Visual element interface.
  - **NodeMark.java**: Unclear what this class should do because is been left unimplemented and is not used.
  - **Packet.java**: The packet component of the network. It implements Visual Element Interface
  - **Queue.java**: The queue of each node. It is drawn as a stack of rectangles above a node. It implements Visual Element.
  - **Scheduler.java**: Core class for the animation. It implements the animation interface and it contains the main animation loop. It also keeps all components of the network topology in data structures.
- **VisualElement.java**: The main interface for anything is visible in the animation.

  - animation.layout
    - **ForceLayoutManager.java**: Auto-layout with force-optimization algorithm
    - **GraphLayoutManager.java**: Abstract class of auto-layout.

  - **fileio**
    - **FileIO.java**: Interface for reading trace files.
    - **Input.java**: It is used to parse values from the trace file. (Integer, double, etc.)
    - **TraceFileReader.java**: Implements FileIO interface, it reads a trace file and generates the appropriate events.

  - **Fileio.event**
    - **Event.java**: Abstract class that contains all attributes that any event needs, like time to be executed. All events in this package implement this class.
    - **GroupEvent.java**: Event when a node joins or leaves a multicast group. This class is not implemented and not used.
    - **LinkEvent.java**: It creates a link that connects two nodes.
    - **NodeEvent.java**: It creates a node.
    - **NodeMarkEvent.java**: Not clear what this class should do as it is not implemented and not used.
    - **PacketEvent.java**: It creates a packet event. Packet events can be of different types, like adding a packet to a queue, removing a packet from a queue, dropping a packet, or sending it from node A to B.
    - **QueueEvent.java**: It defines a queue to the specified node. Then the node can accept packet events of type en-queue/de-queue.
    - **StopEvent.java**: Is the final event that causes the animation to stop.
    - **TextEvent.java**: Is the event that appends text to the annotation area.
    - **TextEventTcl.java**: It generates a TCL text compatible with NS [22] and NAM [5]. It is used by “Annotate” button that allows the user to add additional annotations.
• Iface
  o AppletSaveTcl.java: Loads a dialog to save the TCL text back to file.
  o SwingAbout.java: Is the about form with information about the application.
  o SwingAnnotate.java: A form to write in a new annotation.
  o SwingDebug.java: A debug window.
  o SwingInterface.java: The main Interface of the application.
  o SwingPanel.java: The main panel where the animation takes place.
  o SwingPreferences.java: Preferences form.
  o SwingSplash.java: Splash window when the application starts.
  o UserInterface.java: Interface that is implemented by the SwingInterface class. The scheduler class has an instance of a UserInterface and can access its methods.
  o VisualiserPanel.java: The interface of the main panel. Main panel is the panel when the animation occurs.
  o VisualiserPanelObserver.java: The observer interface of the main panel.

• math
  o Matrix.java: Provides some manipulation with matrices.
  o Vector.java: It is used by scheduler and packet classes to calculate the direction of the flowing packet.

• util
  o Colour.java: Contains some standard colours and also generates more colours using the RGB value from the trace files.
  o Debug.java: It just has a static public string value to collect all debug information.
  o DebugListener.java: Interface with one method signature (println(String)).
  o DebugOut.java: Prints the debug message.
  o ImageLoader.java: Loads images from jar directory or from a path.
  o Preferences.java: This class saves and loads automatically the javis.ini file, which is a text format file that keeps all the preferences values of the application.
  o Shape.java: Contains available shape types for nodes and also parses shapes from string.
When the application loads everything is being initialized by the SwingInterface class as is the main or core class of the application. Then SwingInterface initializes Scheduler class. All visual elements/ network components are handled as events. When the user opens a trace file, from the file open trace file menu, the application reads twice that file. In the first pass it identifies and loads only events that are static in an animation execution. These are nodes, links and queues of each node, if any.

**First Pass:**

1. For each line identify what kind of event is.
2. If is static then make a new event object of that event type.
3. Give the rest of the line to that new event to parse it and set its parameters.
4. Add that new event into the event vector and continue reading the file (step 1).
5. If we reached the end of the file then for each event in the event vector create the visual element object and add it to scheduler appropriate vector (nodes, links or queues).
6. Delete the event object from events vector.
7. When events vector size becomes 0 then is time for pass 2.

In the second pass all dynamic events are loaded and the animation starts. The animation ends when it reaches the StopEvent which normally is the last line event in a trace file.

**Second Pass:**

1. For each line identify what kind of event is.
2. If it is a dynamic event the make a new object of that event type.
3. Pass the rest of the line to that new event to parse it and set its parameters.
4. Add the newly created event in the events vector and continue to step 1.
5. If we reached the end of the file we are ready for animation.

The animation can be started by the press of the play button. Then a thread is created that runs the main loop of the animation. Very briefly, on any loop iteration we have a time
increment that equals the value of the time slider. For each iteration any events that were scheduled before or the exact current time they are handled. After handling the events the thread is delayed by the “fps” value that exists in preferences and then it loops until no events left or in case of StopEvent.

What looks quite inefficient is the need to read the file twice, and even more, each time the user presses the rewind button it will cause a full opening file process from the beginning.

10. Graphical Network Simulator

10.1. Changes or Improvements from the original JAVIS code

10.1.1. Refactoring Process

Firstly all code was updated to the JDK 1.6 and all warnings were resolved, by replacing deprecated methods with new ones. All data structures have been replaced by newer data types and with the use of generics.

Some core classes like the SwingInterface.java and Scheduler.java were completely re-implemented. SwingInterface class had the Graphical User interface components, their actions and some other methods, and because it was really big and difficult to read it was decided to split it according to the MVC pattern. This would also help future developers to easy understand the code but also very easy to adapt new features or even disconnect completely the Graphical User Interface with the controller class and do something else. For the new vertical panel with buttons to design the network topology it was decided to have its own view, controller and model because its purpose is completely different from the main interface. All other forms or panels created have their JComponents together with their actions in the main class. The reason why is because the data that these forms hold is less important, classes are short in size and readable anyway (no overuse of patterns).

Scheduler was the main class of JAVIS that handled the animation, it was modified many times by many programmers during the years, and that was easily noticeable by different
styles of coding, duplication of code or even code that could be written simpler in fewer lines and do the same job as before. So it was reshaped carefully across those many refactoring stages during the implementation process.

From the original JAVIS code the following classes were considered as not useful and were deleted:

- **animation.Link.java**: Is been replaced by GLink.java which is more consistent with JNS features.
- **animation.Node.java**: Is been replaced by GNode.java which is more consistent with JNS features.
- **fileio.FileIO.java**: We don’t need to read from a trace file.
- **fileio.Input.java**: We don’t have any trace lines to parse.
- **fileio.TraceFileReader.java**: Trace files do not exist in the current project.
- **fileio.event.GroupEvent.java**: There is nothing implemented in this class and it was not in use. Removed to make system simpler.
- **fileio.event.LinkEvent.java**: Links are not events any more. They are designed using GUI.
- **fileio.event.NodeEvent.java**: Nodes are not events any more. They are designed using GUI.
- **fileio.event.NodeMarkEvent.java**: It is not implemented and it is unclear what it should do.
- **fileio.event.QueueEvent.java**: Queues are not events any more. They are automatically defined and each node has its own queue.
- **fileio.event.TectEventTcl.java**: Trace files are not used any more.
- **iface.AppletSaveTcl.java**: Anything has to do with trace files have been removed.
- **iface.SwingAnnotate.java**: There is no need for the user to add extra annotations, because annotations will come directly from the simulator.
- **util.Colour.java**: We don’t need to have a class which is given by Java libraries.
- **util.Shape.java**: Shapes on nodes are not used as they have been replaced with icons.
It is noticeable that events that were unimplemented have been removed. Someone could ask: Why eliminating the ideas of future animation events on this application? Well the answer is that the following thoughts have been taken into account:

1. We should keep the number of files as low as possible to end up with a robust application.
2. All needed animation events are defined by the simulator itself.
3. Unimplemented features should not exist in the source code. It closes the door to future developers to use their imagination and creativity.

The following classes have been created for the new Application:

- **Javis.controller.InterfaceController.java**: Is the controller class of the SwingInterface.java
- **Javis.controller.DesignController.java**: Is the controller class for designing the network. The view class is the SwingSetupPanel.java.
- **bridge.EventCreator.java**: It is used by the simulator when is needed to create an animation event. It creates the event for the animation.
- **Javis.guiModel.InterfaceAddressDuplicationException.java**: Exception that is thrown when another interface with the same hardware address exist.
- **Javis.guiModel.Network.java**: The Network class contains all nodes, links and interfaces of the network. Its main job is to keep all the information of the simulator that is related with the network topology and configuration.
- **Javis.guiModel.PopupInterfaceSelector.java**: Is the popup window that loads up for the user to select the interface to connect the link, in a linking process.
- **Javis.guiModel.SimScenario.java**: It holds the simulation scenario.
- **Javis.guiModel.SimSetupModel.java**: Is the model class used by the interface that designs the network.
- **bridge.SimulatorCreator.java**: It creates a simulation for the JNS given the current network topology and specified simulation scenario.
- **Javis.iface.EventInputTerminal.java**: Is a (UNIX style) terminal emulator created to provide an alternative way of setting a simulation scenario.

- **Javis.iface.StatusPanel.java**: Is the bottom panel of the application which contains the current status label, zoom slider and the mouse position \((x, y)\) on the drawing panel.

- **Javis.iface.SwingScrollable.java**: Its purpose is to make the drawing panel scrollable in both axes.

- **Javis.iface.SwingSetupPanel.java**: Is the vertical panel on the left side of the application that contains buttons for the network configuration setup (add host, add switch, add link, etc.).

- **Javis.Iface.forms.AttributesForm.java**: Is the basic form that all forms for attributes extend. It calculates the place to load the form according to computer screen size, window size and window position on the screen.

- **Javis.Iface.forms.DefaultValuesForm.java**: Is a form for setting default values for each network component.

- **Javis.Iface.forms.HelpText.java**: Is a form that provides some more information when the user needs hints.

- **Javis.Iface.forms.HostAttributesForm.java**: Is the form for setting host attributes. It extends the AttributesForm class.

- **Javis.Iface.forms.InterfaceForm.java**: Is the form for adding a new or editing an existing interface.

- **Javis.Iface.forms.LinkAttributes.java**: Is the form for setting the attributes of a link. It extends AttributesForm class.

- **Javis.Iface.forms.ScenarioForm.java**: Is the form for setting the simulation scenario.

- **Javis.Iface.forms.StatsForm.java**: Is the form that shows some statistical data in the form of graphs or text.

- **Javis.Iface.forms.StatsTextView.java**: Is the form that shows the statistical data as text.

- **Javis.Iface.forms.SwitchAttributesForm.java**: Is the form for setting the attributes of a switch. It extends AttributesForm class.
- **Javis.networkElements.GNode.java**: Is the abstract class of the node network component.
- **Javis.networkElements.GHost.java**: Is the Host network component. It extends GNode class.
- **Javis.networkElements.GInterface.java**: Is the Interface network component.
- **Javis.networkElements.GLink.java**: Is the link network component.
- **Javis.networkElements.GProtocol.java**: Is the supported by node protocol network component.
- **Javis.networkElements.GSwitch.java**: Is the switch network component. It extends GNode class.
- **Javis.networkElements.NodeSelection.java**: Is the class that handles the group selection of nodes with the mouse.
- **Javis.util.Generator.java**: Generates interface hardware addresses, and IP addresses based on the specified mask from Default Values and the current node ID.
- **Javis.util.XmlUtils.java**: Retrieves / Stores the simulation configuration from/to a file in XML format.

The following classes now follow the singleton pattern:

- SimSetupModel.java
- StatusPanel.java
- Network.java
- Scheduler.java
- SwingPanel.java
- SwingScrollable.java
- SwingInterface.java
Model View Controller pattern:

The first implementation is on the pre-existing SwingInterface.java which needed to be split.

- **SwingInterface.java (View):** Consists of the menu bar, the horizontal panel with simulation and animation controls and the annotation text area.

- **InterfaceController.java (Controller):** It contains many subclasses as action to the buttons or sliders of its View. Core actions are opening or saving a XML configuration, running the JNS and controlling the animation.

- **Rest JAVIS (the model):** Due to the fact that SwingInterface is considered as the main graphical user interface of the application, controller interacts with many methods from all the javis package.

The second implementation of MVC concerns the Visual editing feature for setting network topology and simulation setup.

- **SwingSetupPanel.java (View):** It is the vertical panel with buttons to design the network topology.

- **DesignController.java (Controller):** It contains many subclasses as action listeners to the buttons of the design panel (View). It also contains the mouse listener subclass of the SwingPanel. There is a tight relation between the choice of the network component to add, and the position on the panel that it will be placed. In other words a complete task of adding a host for example, requires interaction with two JComponents (the button and the panel). That’s the reason why we decided to include the mouse listener of the SwingPanel into the DesignController class.

- **simSetupModel.java (Model):** It acts as the model of the SwingSetupPanel and DesignController. It keeps the current input state and handles the actions. The click of a button in the SwingSetupPanel will cause the appropriate input state to be set in the model class. Then when the user clicks on the SwingPanel for the component to be placed the model class will get the point that the mouse was clicked and will handle the input event according to the current state.
10.1.2. Multi format Annotations

The annotations text area that JAVIS had was fairly enough for impersonal annotations but now with the Graphical Network Simulator we wanted to emphasise the sender of the annotation to make annotations easily readable while they occur fast. Furthermore we wanted to be able to emphasise annotations that have to do with failure and dropped or discarded packets. Thus the JTextArea was replaced by JTextPane, and different styles of text or different fonts were defined for these purposes.

10.1.3. Resizable annotation panel

The annotation panel had only visibility of 4 lines; quite small for a few hundreds of annotations that appear in each animation. Also the panel that shows the animation is rather important. After some time was spent looking what else is provided by java libraries it was easy to decide that the solution would be to use a JSplitPane that allows the user to change the size of panels dynamically with the mouse. This solution was simple but really useful because it gives to the user the decision of focusing more or less and switching between the animation and the annotation area according to his needs.

10.1.4. More Detailed Animation

Animation in JAVIS was based on coloured small rectangles flowing from node to node. After a basic level of connection between JNS and JAVIS it was not clear to us if the connection was correct as we didn’t know which packets were represented and when. After some modifications in the PacketEvent and Packet class we were able to pass more information through these classes and now it is displayed above each packet. Now we can distinguish ARP from Ethernet packets and their code types.
10.1.5. Wider animation speed Range

Due to the newer technologies, network medium speeds have been increased, simulations take less time than before and animation events tent to occur really closed to each other in time. Thus the minimum animation time step has been extended to one nanosecond. Finally due to the fact that the range now was wider the slider was a bit touchy, so we decided to increase its size on the panel a little bit more.

10.1.6. Animation events data structure

In JAVIS the events were kept in a vector. Inspired and excided from the NetAnim Advanced Mode[25] which supports animation at the same time with the simulator (Live) we decided to replace that vector with a PriorityBlockingQueue instead of a typical ArrayList. Although live mode could be enabled with some changes to the main loop we didn’t because the time a simulation needs to be executed is considerably much less the time to preform one or two events. So live mode was considered as did not worth the time needed to be spent, but the PriorityBlockingQueue is kept for future projects.

10.1.7. Animation History

In JAVIS the rewind button used to initialize the program reopen and reload the trace file in order to re-perform the animation. This method was considered inefficient in consuming unnecessary hardware resources. Although trace files do not exist in the Graphical Network Simulator it would be again inefficient if we let the simulator to be executed each time. To avoid that we decided to keep a history ArrayList that keeps the animation events of the last execution. Then when the rewind button is pressed the history is copied back to the main data structure for the events and the animation is replayed without any unnecessary or extra cost.
10.1.8. Simulator Seed Number

The animation seed number was one of the first items added to the Settings form as it is a general rule in (stochastic) simulators to be able to control that number. It is really important to be able to reproduce the exact same results between different runs but also be able to make new ones at will.

10.2. Adaption of JNS

In order to adapt the JNS to the JAVIS project we moved the current source of JAVIS into package named “javis”, and then we imported the JNS code into a package named “jns” into the same project. For the communication between the simulator and the graphical user interface another package was created named “bridge”. It consists of two classes which are really important:

**SimulatorCreator**

This class is can get as input the Network class and setup the simulation as it was demonstrated by the author of JNS [1]. In other words it acts as a parser of what the user has designed by the mouse to the JNS simulator. This class is called when the run JNS button is pressed or by the terminal.

**EventCreator**

This class contains methods to create animation events. It is used by the simulator while it runs and thus it creates events for the animator. There are different kinds of methods that may lead to the same animation event but what differs is the location that these methods can be called from. According to the simulator’s network layer or class that an animation event needs to be scheduled there is a different process that needs to be performed for the identification of the packet type, the sender, the recipient or any other information is needed to perform a correct and detailed animation. Any new event is directly pushed to the scheduler events.
The package name as “bridge” was taken because these two classes act like a bridge between the simulator and the animator. Another implementation would be the creation of an Interface instead of the class at least for the EventCreator. On the other hand we wanted to keep the code related to both JNS and JAVIS in a separate package thus no interface was needed.

10.3. New Features

10.3.1. Visual editing
A new panel was created with buttons to allow the user to design a network topology and specify the simulation. The user can add nodes (Hosts, Routers, and Switches) into the drawing panel and link them together with links. Upon being placed, nodes can be moved by dragging and the parameters of each network component can be edited by clicking on them.

The identification of the clicked node is done by getSelectedNode(Point p) in Network class as follows:

```java
public GNode getSelectedNode (Point p){
    for (GNode node: nodes){
        if (p.x <= node.getXpos()+32 && p.x>=node.getXpos() && p.y<=node.getYpos()+32 && p.y>=node.getYpos()){  
            return node;
        }
    }
    return null;
}
```

If a node value and not null is returned from this method, the attributes form of this returned node will load up. All attributes forms inherit from the AttributesForm.class which takes as constructor arguments the position of the mouse pointer when it was clicked and the dimensions of the form. The main functionality of this class is to calculate and find the position where the form should be loaded. It also takes into account the computer screen dimensions the application’s window dimensions and position of that window on the screen in order to load the attributes form in a visible area on the screen. The positioning was inspired from the left click on a standard desktop computer which generally brings a popup window with options. That popup window is never loaded outside the screen or half visible; it appears above, below, to the left or to the right of the mouse pointer, wherever there is enough space for its size.
The identification of a clicked link is a little bit more complex operation and it was decided that this operation requires two methods:

**Network: getSelectedLink(Point p):**

```java
public GLink getSelectedLink(Point p){
    for (GLink link: links){
        double dist=getDistancefromLine(link,p);
        if (dist<=DIST){
            System.out.println("Found Link");
            return link;
        }
    }
    return null;
}
```

For each link in the network we find the distance from the mouse pointer. If the distance between the mouse pointer and the link is less or equal to DIST (which is defined as the value three) then return that link.

This makes the link a clickable object of length the length of the link and width of six pixels which can be clicked easily by anyone.

**Network: getDistancefromLine(GLink link, Point p):**

```java
public double getDistancefromLine(GLink link, Point p){
    Point a=new Point(link.getStartingX(),link.getStartingY());
    Point b=new Point(link.getEndX(),link.getEndY());
    double dSideA= Math.sqrt((p.x-a.x)*(p.x-a.x)+(p.y-a.y)*(p.y-a.y));
    double dSideB= Math.sqrt((p.x-b.x)*(p.x-b.x)+(p.y-b.y)*(p.y-b.y));
    double dAB = Math.sqrt((b.x-a.x)*(b.x-a.x)+(b.y-a.y)*(b.y-a.y));
    double r_numerator = (p.x-a.x)*(b.x-a.x) + (p.y-a.y)*(b.y-a.y);
    double r_denominator=dAB*dAB;
    double r=r_numerator/r_denominator;
    //point L is of line AB such that PL is the minimum distance.
    double lx = a.x + r*(b.x-a.x);
    double ly = a.y + r*(b.y-a.y);
    if (r<0){
        return dSideA;
    }
    if (r>1){
        return dSideB;
    }
    return dPL;
}
```
In order to find the distance of the mouse pointer from the link we need to calculate the distance of a point \( P \) from a line in a two dimensional area. Due to the fact that the link is not infinite, it starts from point where the source node \( A \) is and ends at the destination node \( B \), we need to calculate distance of point \( P \) from a line segment \( AB \). This decision will lead us in 3 cases, and the following equation is being used to identify them: 
\[
 r = \frac{AP \cdot AB}{|AB|^2}
\]

- **Case 1:** If \( r \) value is less than 0 then the minimum distance is the distance from point \( P \) to \( A \) using the equation:
  \[
  d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}
  \]

- **Case 2:** If \( r \) value is more than 1 then the minimum distance is the distance from point \( P \) to \( B \)

- **Case 3:** If \( r \) value is somewhere between 0 and 1 then the minimum distance is the distance from point \( P \) to the point \( L \) of line \( AB \), such as \( PL \) is a line segment that is perpendicular with the line \( AB \). The \( L \) point can be calculated as:
  \[
  L_x = Ax + r \times (Bx - Ax) \quad \text{and} \quad L_y = Ay + r \times (By - Ay)
  \]

**Host Attributes Form**

- **ID:** Not editable field. It only shows the ID of the node.
- **Name:** Any name can be set.
- **Work as router:** Checkbox to transform a host to router and the opposite.
- **Supported protocols:** Shows a list with currently supported protocols and a combo box with currently unsupported protocols. The user can select a protocol from the combo box and press the add button to add it into the supported list. It may also select a protocol from the supported list and press the delete button.
- **Attached Interfaces:** Shows a list

**Interface Form:**

- **Hardware Address:** The interface MAC address.
- **IP Address:** The Interface IP address
Set Default Route: Check box that is checked to enable default route field.
Default Route: The interface default route.
Subnet Mask: The topology network mask.
MTU: The maximum transmission unit of the interface.

Switch Attributes Form:
- ID: Not editable field. It only shows the ID of the node.
- Name: The name of the Switch.

Link Attributes Form:
- Link type: Radio buttons Duplex/Simplex
- Direction: Checkbox inverse. It is only enable when the simplex is selected.
- Bandwidth in bps
- Delay in seconds
- Source Interface: The source interface of the link.
- Destination Interface: The destination interface of the link. (Source and Destination don’t make any difference on duplex links)

10.3.2. Default Values

After some network topologies were produced, it was noticeable that most of the parameters were the same for each node or link and the user had many forms to complete over and over again with the same values in order to design a network simulation. To avoid that we had the idea of default values feature. A form of default values was created that allows the user to set default or initial parameters for each network component added into the network design. The Preferences class was extended to include all the default values, and these values can now be loaded and stored into the existing *.ini file along with pre-
existing values from JAVIS[1] project. The form for specifying Default Values is divided in three sections:

**Hosts/Routers Defaults:**

- **Name for Hosts:** The given name will be concatenated with the node ID for uniqueness.
- **Name for Routers:** Again the given name will have the node ID for uniqueness.
- **MAC Address Mask:** The user is able to specify a mask with hexadecimal values and xx where an incremental mac address generator algorithm will be applied. In other words this field can be filled with hardcoded and incremental part of the hardware address. It can also be set to xx:xx:xx:xx:xx:xx for maximum number of nodes for example.
- **IP Address Mask:** The user is able to specify an IP mask with numbers from 0 to 255 and x where the IP address generator algorithm will be applied.
- **Default Subnet Mask:** The default subnet mask that will be given to all nodes.
- **Supported Protocols:** The supported protocols that each node will have attached by default.

**Switches Defaults:**

- **Name for Switches:** As with Hosts and Routers the given default name will be placed together with the node ID for each new switch.

**Links Defaults:**

- **Default Link type:** The default link type for each new link. It can be simplex or duplex.
- **Bandwidth:** The default bandwidth in bits per seconds for each link.
- **Delay:** The default delay in seconds for each link.

Although Default Values feature reduce considerably the amount of time someone requires to set up a simulation, it might not be liked by some others because it is not perfect. Most of the values set by default will not need any manual change for some small network simulations, but IP Addresses, subnet mask and default route addresses might need some editing before the network topology is completely set. Also any additional interfaces might
need to be manually set up. Thus it was decided to give to the user the option to enable or
disable this feature.

10.3.3. Zooming Capabilities

Due to the fact that computer screen resolutions vary from computer to computer it was expected that opening network configurations on a computer with a small screen that has been saved previously on a larger screen would make some nodes being placed in the borderline or even worst outer of the drawing panel. Although the maintained auto-layout algorithms (shake and re-layout) could partially save the situation, a better solution was investigated and implemented. Firstly the drawing panel was added into a scrollable pane, then with the use of a JSlider the panel size can be increased or reduced according to the user needs. It is implemented in such way that the designed network stays in the centre of the screen while the size of the panel is increased from all four sides. In addition, when the size of the drawing panel decreases, if network nodes hit the edges of the panel they are automatically moved back into the panel keeping in that way all nodes into it. The zooming solution is divided into three steps:

- Step 1: Increase the size of the drawing panel.
- Step 2: Move the horizontal and vertical sliders accordingly to maintain the current visible area while zooming.
- Step 3: Reposition the nodes into the panel. This third step was implemented in two ways initially:
  - Real zooming capabilities: each node keeps its relative position in the panel’s height and width. Nodes new X position = (nodes X position * panel’s new width) / panel’s old width. Nodes new Y position = (nodes Y position * panel’s new height) / panel’s old height.
  - Just move nodes in such a way that in user’s eyes they stay at the same place while the panel’s size is increased in all 4 sides. For this implementation, extra control statements were needed in order to keep nodes into the panel if the panel is smaller than the size of the network design itself.
After some testing it was decided that the second implementation was the most useful for the current project and the first one’s code was commented. The reason why is because the second implementation allows the user to increase the paper and add more nodes in any side of his current topology without the need to move it manually.

10.3.4. Hand Mode

After the use of scrollable pane for the drawing panel the user could scroll horizontally and vertically. Generally, vertical scrolling is easy because most input devices (mouse, touchpad) have that feature. Difficulty comes when the interface requires horizontal scrolling as the majority of users have to click and drag manually the horizontal bar. Hand mode lets the user to “grab” the drawing panel by a click and move it in both horizontal and vertical axis by dragging.

While in hand mode, if the mouse button is pressed, the mouse position and current values of scrollbars are stored as a reference point. Then when the user drags the mouse the following code is executed:

```java
int x=arg0.getX();
int y=arg0.getY();
int xdiff=this.handInitPos.x-x;
int ydiff=this.handInitPos.y-y;
int newHorValue=this.scrollHorValue+xdiff;
int newVerValue=this.scrollVerValue+ydiff;
scrollpane.getHorizontalScrollBar().setValue(newHorValue);
scrollpane.getVerticalScrollBar().setValue(newVerValue);
scrollpane.revalidate();
```

Firstly the difference between the current mouse position and the reference mouse position is calculated by a subtraction. Then the new horizontal/vertical value is the initial value plus the difference.
10.3.5. Group Selection and Movement

Another feature that helps a lot to design the network topology is the group selection and movement (inspired by Otter [3]). The user is able to make a selection of nodes with the mouse and then move their position as a group by dragging any selected node. It feels natural, because of the experience that a standard desktop user already has, as it is similar to dragging shortcut icons on a desktop. The selection can be started (when the mouse mode is selected) by a press of a mouse button in any empty area on the drawing panel (where there is no any node or link below). Then by holding the mouse pressed and move the mouse it will draw a rectangle. All nodes in the rectangle’s area will be selected. Then by dragging any of the selected nodes, all nodes will move together as a group. The NodeSelection class was created as the helper class for this feature as it combines some useful methods including the draw(Graphics) which draws the rectangle while the mouse moves on the screen. Two Points are needed for drawing the rectangle. As shown in Figure 13, A is the point that the mouse button was initially pressed and B the place it was released. With points A and B of course we can find points C and D and lines from A to C, from C to B, from B to D and from D to A are drawn.

Figure 13: Group selection and movement

Now let’s assume a node with position N(Xn, Yn). If \( X_2 < X_n < X_1 \) and \( Y_2 < Y_n < Y_1 \) then the node is selected. A Boolean attribute of the node class is set and when the node is drawn if that
value is true a rectangle as shown in Figure 13 is drawn around it. At the same time each selected node is added into an array list and then when the user drags any of the selected nodes, all selected nodes move accordingly.

10.3.6. Statistics and Bar Graphs

According to JNS [1] some statistical data feature was implemented but after some studying of the code we decided that it would be a better idea to make a new class for statistical data. The new class named Statistics.java is placed into the jns.util package. It contains an instance of the same class and some static final values for the following types of data:

```java
public static final int FRAME_RECEIVED=0;
public static final int FRAME_SENT=1;
public static final int PACKET_DROP=2;
public static Statistics stat=new Statistics();
```

An internal class is defined in the Statistics class. Its purpose is to collect frame received, frames sent and packets dropped for each node independently. All instances of the internal class are kept in a hash map for fast access with the key the owner node ID of that entry. Each node then can add a new entry in the statistics using the method newEntry:

```java
public void newEntry(int source, int type){
    StatEntry se=entriesMap.get(source);
    if (se==null){
        se=new StatEntry(source);
        entriesMap.put(source, se);
    }
    switch (type){
    case FRAME_RECEIVED:
        se.increaseFramesRec();
        break;
    case FRAME_SENT:
        se.increaseFramesSent();
        break;
    case PACKET_DROP:
        se.increasePacketsDropped();
        break;
    default:
        break;
    }
}
```

If it’s the first time that a node sends statistics then a new entry is created for it, otherwise it increases the counters of the existing entry according to the type argument. At the end of a
simulation, Statistics class contains a hash map with statistics from nodes that took part on the specific simulation. Then the statistical data can be printed out directly from the statistics class, results can be requested as a String, or as an array of StatEntry objects.

After the creation of statistical data we decided to provide an alternative view to the text based view that would give the user the ability to compare values between nodes. For the form of statistical data that we collected the most appropriate way was considered the bar graph. BarGraph class was then implemented in such a way that it takes only 2 arrays as input and the title of the graph. The first one of the two arrays is the array of node ids and the second one is their values. BarGraph class extends JComponent and it draws the graph by overriding its paint(Graphics) method. Firstly it assigns colours for each id, based on a RGB random colour generator method that uses the JNS seed number. Then it finds the maximum value from the array of values and that will be the maximum value on the graph. After that it calculates how long the width of each bar should be so they can all fit into the graph’s total width. Finally the length or height of each bar in pixels is calculated by the following formula: \( ngth = \frac{\text{StatisticalValue} \times \text{GraphHeight}}{\text{MaximumStatisticalValue}} \). The rest is taken care from the overridden paint method which draws the title, the background, the lines of values, the bars and their explanation as shown in Figure 14.

![Figure 14: Bar graph](image)

All graphs are created and showed to the user by StatsForm class. It is the form that loads up when the user presses on the “Stats” button. All graphs are placed in a scroll pane and the user can scroll and view as many graphs as our statistics. Currently for the purposes of
this project, three graphs are placed. Frames received per node, frames sent per node, and frames dropped per node.

10.3.7. Scenario Form

This is the main form for setting a scenario to the designed network topology. It supports all three simulation scenarios that are currently supported by the simulator with the following fields as required by each scenario:

- **Send Data** (as a pair of Data Source/Data Sink): “These two applications use TCP to exchange arbitrary amounts of data in order to illustrate the segmentation and reassembly of TCP segments. The amount of data that is sent by the source can be specified, and so TCP can be observed under different conditions related to buffer and window sizes, as well as network congestion. Once the specified data has been successfully transmitted the connection is terminated by the client.”[1]
  - Sender: The node that will send the data.
  - DestIP: The destination IP address of the transmission.
  - Bytes: The amount of data that will be sent.
  - Recipient: The node that will receive the data. (is the node that has the destination IP address)

- **Quote** (as a pair of quote Server/Quote Client): “These two applications use UDP to exchange messages such that the client is able to request a random quote from the server and the server will return one. The randomisation of the quote selection is controlled by the simulator’s random number generation and so the same quote will always be returned for a given numeric seed.”[1]
  - Client: The client that quotes.
  - DestIP: The IP address of the server.
  - Server: The server that will return the quote.

- **Ping:** Transmits an echo request / reply to the specified destination IP address. It does not consider request timeouts or allows multiple requests in a single execution.”[1]
- FromNode: The node that the ping application will run on.
- IP to Ping: The IP address of the node to ping.

### 10.3.8. Terminal

Terminal was created as an alternative way for executing a simulation scenario. It offers a direct execution of a specified scenario as a command, and it was mainly inspired from the UNIX terminal and the Ping command line tool. The main difference thought from the GUI scenario specification and run, is that with terminal the simulator keeps it state after each execution. For example after each simulation execution, routes that have been found will remain for the next execution and no ARP requests/replies will be needed for those routes. In addition statistics are also kept during executions. For that reason the additional command “reset” was implemented to reset the simulator and clear the statistics.

**Available terminal Commands:**

- **help**: Shows the terminal’s available commands.
- **ping <srcID> <IP>**: Executes a ping simulation. The first argument is the id of the source node and the second argument is the IP address to ping.
- **quote <srcID> <IP> <destID>**: Runs the quote client/server pair scenario. The first argument is the ID of the source node the last two are the IP and the ID of the destination node.
- **quoteclient <srcID> <IP>**: Executes the quote client application which is the first from the quote pair. The first argument is the source node ID and the second argument is the destination IP address.
- **senddata <srcID> <IP> <DATA> <destID>**: Runs the data sent/sink pair scenario. The first argument is the source node id the second is the IP address that the data will be sent, the third argument is a numeric value in bytes representing the amount of data to be sent and the fourth and final argument is the destination node id.
- **datasource <srcID> <IP> <DATA>:** It runs only the first scenario of the “send data” pair. The first argument is the source node ID, the second is the destination IP address and the third is a numeric value in bytes and represents the data to be sent.

- **reset:** resets the simulator and clears the statistical data.

Two more simulation applications currently exist by the simulator (quote server, data sink) but they are not meant to be run independently, they are used as pairs in send data and quote scenarios. After supplying them in the first place they caused the simulator to crash and so they were removed. The same applies to “quote client” and “data source” commands, typically they are not used independently but because some animation occurs while executing them and the simulator does not crash we decided to provide them for educational purposes, having in mind that also faulty scenarios give lessons to students.

### 10.3.9. Status Panel

Status Panel was created mainly where extra feedback was required by the application to the user. It gives feedback to the user for his several actions. Currently it does not give feedback in all actions, but it gives to those that the user may feel unsure if he succeeded or not. A good example is the linking process. The user may be unsure if the link was connected to the correct interface. Another nice example is when the user has a loaded file and he wants to save changes back. He can just press CTRL+S and the Status Panel will inform him that the configuration was saved with the time that happened as well.

### 10.3.10. Frame title with opened filename

From the early stages of development we realised that there was no information to the user of which file is loaded. The Controller class of the SwingInterface overcomes to this problem by setting the current opened or saved file name with full path, aside with the application’s title on the application’s frame.
10.3.11. Save /Restore simulation configurations in XML format

One of the main requirements of this project was to provide a feature to store and retrieve simulation configurations from/to a file in XML format. It was achieved successfully and the XML output file contains all the information of a network topology (nodes, links) including all three scenarios from the scenario form. With all these values stored and retrieved to/from the XML the user can have any network configuration ready to run in seconds.

10.4. Design Considerations

10.4.1. Graphical User Interface

Figure 15: Main GUI of Graphical Network Simulator
Add Host: By clicking that button and then clicking again on the drawing panel the user can add a new host.

Add Router: By clicking that button and then clicking again on the drawing panel the user can add a new router. Any Host can become router by checking the router check box in Host’s parameters and it is actually a Host with a Boolean flag set to true. Due to the fact that the process of how to add a router was not clear, it was decided to put an extra button for adding a router, even if it refers to the same type of object.

Add Switch: By clicking that button and then clicking again on the drawing panel the user can add a new Switch.

Add Link: The mouse icon becomes a roll of wire. The user then must click on a node and a window with available interfaces will popup. Then the user must click on an interface for the link to be connected to and the link source point will be set. Then the user must do the same to the destination point to successfully link two interfaces of two nodes.

Cut Tool: When it is clicked the mouse icon becomes a scissors. The user can click with that tool on nodes or links to remove them. If the clicked component is a node and it is linked with one or more other nodes then those links will be deleted too, as it makes no sense having links to nowhere.

Mouse Mode: Mouse mode has been added, to allow the user to cancel any importation of a new node, or to go from any mode back to normal.

Hand Mode: When clicked the mouse pointer becomes a hand that can grab the panel and scroll it vertically or horizontally according to the mouse move.

Shake: When pressed it moves nodes a little bit (it shakes the topology).

Re-layout: When pressed it places the nodes in positions according to a defined auto-layout algorithm that existed in JAVIS. Shake and Re-layout buttons are not very useful in the current project but they may be useful in the future (look at Future Work) so we decided to maintain these functions.
**SetScenario**: Loads the scenario form in order to select a scenario and fill it fields. A warning window pop's up instead if no nodes exist in the drawing panel.

**Run JNS**: It executes the simulator with the current designed network and specified scenario form as input. Then after the simulation finishes it automatically begins the animation. If no network is designed or no simulation scenario is specified a warning window pops up instead with the appropriate message.

**Rewind Button**: Replays the animation without running the simulator again.

**Stop button**: Stops the animation and clears the drawing panel and annotation area.

**Pause button**: Causes the animation to pause.

**Play button**: Resumes the animation.

**Seek button**: Seeks the animation to the next event.

**Current time**: Shows the current time of the animation.

**Time slider**: Changes the animation time increment. The less the time step is, the slower the animation.

**Statistics**: Loads a form with graphs and statistical data.

**Drawing panel**: Is the panel where the network is drawn and where the animation is shown.

**Status bar**: It shows the program status.

**Annotations Tab**: Is the panel where the animation annotation events are printed.

**Terminal Tab**: Is the Terminal to execute network simulators with.

**Zoom slider**: Increases/ decreases the drawing panel size.

**Mouse Position**: It shows the mouse position on the drawing panel.
**File menu**

![File menu](image)

**New**: Initializes the program

**Open**: Shows an open file dialogue for the user to choose an XML configuration file to open.

**Save**: Saves the edited opened file. If a filename does not exist, it shows a save file dialogue for the user to choose filename and path to save the configuration.

**Save As**: Shows the save file dialogue for the user to choose filename and path to save the current configuration.

**Exit**: Exits the Application.

**Options menu**

![Options menu](image)

**Default Values**: Loads the form with default values for any new host, router, switch or link.

**Preferences**: Loads a tabbed form with preferences for Window, Animation or Simulation.
10.4.2. Events

All events continue to inherit from the Event class. According to the class diagram in Figure 18, final static integer have been added in the Event class that define the event types to maintain the identification of each event in the scheduler class.

![Class Diagram of Events](image)

**Figure 18: Class diagram of Events**

10.4.3. Network Components

According to JAVIS, nodes, links, packets and queues inherit from VisualElement abstract class. Although GLink and GNode classes are completely new they still extend the same class. Moreover GNode is now an abstract class and it is inherited by the new classes GHost and GSwitch. In Graphical Network Simulator links connect two interfaces together. In fact
we still store the source node and destination node in each link for fast identification of other interfaces available by that node.
10.4.4. Attributes forms

Another example of polymorphism in the implementation of the Graphical Network Simulator is shown in Figure 20. Each network component form, loads its own properties, on the other hand they have common functions, like the positioning of the form which is related to the user’s screen size, position of the window, and position of the clicked component.

![Class diagram of attributes forms](image)

Figure 20: Class diagram of attributes forms

11. Testing

Due to the fact that the newly created code is mostly depended on the Graphical User Interface JUnit tests considered as not to be a good option. Therefore we made a table of tasks, and some volunteers as testers were used to test occasionally the program, after a
number of increments to see if it stills performs without crashing and gives the correct results.

<table>
<thead>
<tr>
<th>#</th>
<th>Task</th>
<th>Expected result</th>
<th>What actually happens</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Press on the Host button and then click into the drawing area.</td>
<td>The mouse must hold a host and then it must be placed on the clicked point on the drawing area.</td>
<td>As expected.</td>
</tr>
<tr>
<td>2.</td>
<td>Press on the Router button and then click into the drawing area.</td>
<td>The mouse must hold a Router and then it must be placed on the clicked point on the drawing area.</td>
<td>As expected.</td>
</tr>
<tr>
<td>3.</td>
<td>Press on the Switch button and then click into the drawing area.</td>
<td>The mouse must hold a Switch and then it should be placed on the clicked point on the drawing area.</td>
<td>As expected.</td>
</tr>
<tr>
<td>4.</td>
<td>Press on the Link button and then click on any of the existing nodes once</td>
<td>It should popup a menu with its interfaces to choose from</td>
<td>As expected.</td>
</tr>
<tr>
<td>5.</td>
<td>Choose an interface from the popup menu.</td>
<td>A line should be displayed from selected node to the mouse pointer (unroll effect)</td>
<td>As expected.</td>
</tr>
<tr>
<td>6.</td>
<td>Press on the destination node and repeat step 5.</td>
<td>A link should be set between two nodes.</td>
<td>As expected.</td>
</tr>
<tr>
<td>7.</td>
<td>Take the scissors and cut the unlinked node</td>
<td>The node should be deleted.</td>
<td>As expected.</td>
</tr>
<tr>
<td>8.</td>
<td>Press on the hand mode button. Grab the drawing area and move it around.</td>
<td>It should be able to scroll both axes.</td>
<td>As expected.</td>
</tr>
<tr>
<td>9.</td>
<td>Press on the mouse mode button.</td>
<td>The mouse icon should be returned.</td>
<td>As expected.</td>
</tr>
<tr>
<td>10.</td>
<td>Make File-&gt;New</td>
<td>Everything should be cleaned up.</td>
<td>As expected.</td>
</tr>
<tr>
<td>11.</td>
<td>Make two hosts, link them together and then execute “ping 0 192.168.1.2” from terminal (without quotes).</td>
<td>It should execute a ping animation.</td>
<td>As expected.</td>
</tr>
<tr>
<td>12.</td>
<td>Press pause while animation plays.</td>
<td>It should cause the animation to pause.</td>
<td>As expected.</td>
</tr>
<tr>
<td>13.</td>
<td>Press play while animation is paused.</td>
<td>It should resume the animation.</td>
<td>As expected.</td>
</tr>
<tr>
<td>14.</td>
<td>Increase/reduce the speed of the animation with the slider.</td>
<td>It should control the animation speed.</td>
<td>As expected.</td>
</tr>
<tr>
<td>15.</td>
<td>Wait the animation to finish and</td>
<td>Animation should start over.</td>
<td>As expected.</td>
</tr>
<tr>
<td>Task</td>
<td>Description</td>
<td>Expected Outcome</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------------------</td>
<td></td>
</tr>
<tr>
<td>16.</td>
<td>Press the rewind button.</td>
<td>It should jump to the next event.</td>
<td></td>
</tr>
<tr>
<td>17.</td>
<td>Press the seek button.</td>
<td>It should jump to the next event.</td>
<td></td>
</tr>
<tr>
<td>18.</td>
<td>Press the Stats button.</td>
<td>It should load a form with graphs.</td>
<td></td>
</tr>
<tr>
<td>19.</td>
<td>Press the Scenario button.</td>
<td>It should load a form of scenarios.</td>
<td></td>
</tr>
<tr>
<td>20.</td>
<td>Select a scenario and complete its fields and press save. Then do task 18.</td>
<td>It should load the form with fields completed.</td>
<td></td>
</tr>
<tr>
<td>21.</td>
<td>Press on RUN JNS button.</td>
<td>It should show an animation.</td>
<td></td>
</tr>
<tr>
<td>22.</td>
<td>Press the scenario button and test all 3 scenarios.</td>
<td>All scenarios should show animation.</td>
<td></td>
</tr>
<tr>
<td>23.</td>
<td>Make a File-&gt;Save As and save the file somewhere.</td>
<td>It should popup a save file dialogue and then the file should be created. The title of the application should set the opened filename and path.</td>
<td></td>
</tr>
<tr>
<td>24.</td>
<td>Make a File-&gt;New and then File-&gt;Open and open the file you saved before.</td>
<td>It should load the simulation configuration as it was before. The title should set to the opened filename and path.</td>
<td></td>
</tr>
<tr>
<td>25.</td>
<td>Add another node into the network and make File-&gt;Save.</td>
<td>A message File saved at ... (time) should appear on the status bar.</td>
<td></td>
</tr>
<tr>
<td>26.</td>
<td>Change the parameters of a Host, Switch or link by clicking on them and check if they are saved across form openings.</td>
<td>All fields should be saved for each component</td>
<td></td>
</tr>
<tr>
<td>27.</td>
<td>Cut the link between nodes and execute from terminal: “ping 0 192.168.1.2”</td>
<td>A Simulator error message should appear that no link is found on the source node.</td>
<td></td>
</tr>
<tr>
<td>28.</td>
<td>Replace the link and test all other scenarios from terminal</td>
<td>All simulation scenarios from terminal should work.</td>
<td></td>
</tr>
<tr>
<td>29.</td>
<td>Create a new Host and then delete its interface. After that try to set a link.</td>
<td>A warning message should appear that at least one interface is required per node to setup a link.</td>
<td></td>
</tr>
<tr>
<td>30.</td>
<td>Create two nodes and link them together. Then delete one’s interface.</td>
<td>The link should also be deleted.</td>
<td></td>
</tr>
</tbody>
</table>

Table 14: Testing tasks
12. Evaluation

12.1. Evaluation Method/Setup

For the evaluation of this project, two simulation examples with steps were published, as well as the application in the form of an executable binary (Jar) file, and a link to an online survey. Using the software, each volunteer had to follow the steps from the examples and perform these two simulations. Then they had to answer the questions of the online survey.

It is important to mention here that the steps of the examples in the document were carefully written because they play a big role when evaluating Graphical User Interfaces. We didn’t want to explain exactly what to do, and let the user find out. For instance, if we wanted to tell the users to add a new Host into the panel, then we just said that. It is totally wrong to explain to the user that he needs to click on the button with the host icon and then click again into the panel, as such steps would help the user to overcome problematic GUIs and the evaluation would end with wrong results. The Document with the examples and tasks is located in Appendix D. A copy of the online survey is located in Appendix E.

The reason for publishing two tasks (examples) instead of a single task was to give to the users a basic task first and then make them continue with a more complex one. If only one task existed then some users could find a difficulty somewhere and fail, making them unable to evaluate the software, and that would lead us again in wrong results. Another reason for having two examples was the need for evaluating both, form driven and terminal driven scenario execution.

The first of the given examples asks the users to create a network topology with two Hosts and then to setup a simulation scenario that sends data from one to Host to the other, using the form driven scenario setting.

The second example asks to create a network topology with two Hosts connected to a switch and then the Switch to a Router. Then that Router should be connected into another Host. This example required extra modification of nodes IP addresses as well as some subnet masks and the creation of a second interface on the Router’s node. Then using the
terminal, a ping scenario was asked to be performed from one Host of the one subnet to the Host of the other subnet.

Finally the questions for the survey were ordered in such a way that each question would not mislead the user’s mind or interfere with his thoughts on answering the next one. Also a question was added for the user to categorise himself to basic, intermediate or advanced user based on his knowledge in networks.

### 12.2. Evaluation Results

Ten people were interested and evaluated our software, where nine of them have intermediate knowledge in networks and one has a basic knowledge. All users tried to perform both examples and eight of them succeeded in both. Two users succeeded only on the first example.

![Figure 21: Difficulty Evaluation results](image)

Although two users failed to run the second example, surprisingly no one found the application difficult or very difficult to use (strange). Most of the users answered that the application is between easy to normal to use.
All the users agreed that the Graphical Network Simulator is helpful for educational purposes. Two of them have used similar software in the past and they found it helpful. Three users expressed their wish to have such software while they were learning networks.

The majority of the users agreed that the Graphical Network Simulator can be used as a developing platform for new protocols and network applications to be tested. Only two users don’t believe this, and it might be the two that they failed on the second simulation.
The reason why we believe this is because the first example is very simple in a matter that it is difficult to clearly define the application’s objectives.

Finally, in the survey there was another field left for the users to freely write any positive or negative comments about the Graphical Network Simulator. The eight following users wrote comments about the application and they are discussed:

**User 1:** “There are some difficulties with the GUI. Nevertheless it is a nice simulator for students even if it does not show the drop packages.”

**User 2:** “Positive: Nice and simple user interface. Negative: none”

**User 3:** “Very intuitive interface. I like that I could paste some data! Very good animations that make you understand what is happening.”

**User 4:** “Not straightforward to connect the objects.”

**User 5:** “The animations helped me to perform easier the examples.”

**User 6:** “It is a very helpful application to understand how basic network commands work.”

**User 7:** “Very intuitive user interface to create a network topology. The simulator makes it easy to learn how network protocols and devices cooperate in order to exchange data. It might need a few more labels for the user to comprehend the displayed information.”

**User 8:** “The graphics were very good and very helpful to understand how the network works.”

**Discussion:**

The majority of the users commented that the application provides a nice Graphical User Interface to create a network topology and provides a really helpful animation for learning networks or understanding what actually happened. User 1 said that he would like to see dropping packets, well the animation for dropping packets is already implemented and exists in Graphical Network simulator, but the given network simulations of the two examples don’t produce any. User 4 said that it wasn’t very straightforward how to link nodes between each other. The problem was investigated, but it was initially ignored. Few
of the testers at the early stages made errors as they tried to link two nodes by dragging directly from one node to another. User 7 would like to see more information in the animation. The information showed when the animation occurs needs to be minimal and clear, but we could make the packet clickable when the animation is paused. On click, a form could be loaded with a full datagram of the packet.

### 12.3. After Evaluation Improvements

Some of our testers and a tester from the evaluation had some difficulty while setting links between nodes. For that reason we made a second evaluation again, this time it took place in a room with our presence, and any user with minimal knowledge on computers could participate as the goal was only to observe and directly capture any errors while creating the topology. After this second test we came up with two conclusions. The first was about the problem with the link setup. Many users tend to directly drag the link from one node to the other. The problem was that the interface selector didn’t show up, because a click was required for that. As a result the user could not select interface. There was a gap between the error that the user was doing, and the correct way of setting the link. That was a weakness to the GUI to guide the users to the correct way of setting a link. Therefore we made a few changes into the code and now the popup window to select interface appears in any mouse action (click or drag). Additionally when the popup window with attached interfaces loads we changed already connected interfaces to be showed in a different colour from the disconnected ones, and users can now distinguish connected from disconnected interfaces. The second conclusion was that we found another error that the user could recover however after a few seconds. Many users tend to drag the three buttons of nodes (add Host, add Router, add Switch) instead of just to click on them. Due to the limited time left for the current project this second improvement was left as future work.
13. Conclusion

All main objectives and requirements of this project were successfully achieved. The final application is a modification of the old JAVIS visualizer with the adaption of the JNS simulator. It works as it should and it provides an easy to use Graphical User Interface for designing network topology, as well as for specifying the simulation scenario. It produces a nice detailed animation and some statistical data in the form of graphs or text. It does not only meet the requirements of this project but also many more features were introduced that make the user experience using this application even better. Finally the application was tested, evaluated and improved again. A table follows that shows the features that have been investigated on this project:

<table>
<thead>
<tr>
<th>#</th>
<th>Features</th>
<th>Impl.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Multiplatform.</td>
<td>✓</td>
<td>JRE 1.6: Tested on Windows 7, Ubuntu Linux 12.04, and Mac OSX Lion.</td>
</tr>
<tr>
<td>2.</td>
<td>Extensible Simulator.</td>
<td>✓</td>
<td>The only changes to the simulator were the calls to events where needed.</td>
</tr>
<tr>
<td>3.</td>
<td>GUI for Setting all available parameters of a simulation.</td>
<td>✓</td>
<td>Hosts, Routers, Switches, Interfaces, links, scenarios and all their parameters are editable, plus the Simulator’s seed number.</td>
</tr>
<tr>
<td>4.</td>
<td>Animation of the results.</td>
<td>✓</td>
<td>--</td>
</tr>
<tr>
<td>5.</td>
<td>Ability to import/export simulation configurations in XML file format.</td>
<td>✓</td>
<td>Complete save and restoration of all simulator parameters.</td>
</tr>
<tr>
<td>6.</td>
<td>Immediate feedback from the simulator.</td>
<td>✗</td>
<td>Considered unnecessary.</td>
</tr>
<tr>
<td>7.</td>
<td>Replay, Stop, Play, Pause, Seek Next the animation.</td>
<td>✓</td>
<td>--</td>
</tr>
<tr>
<td>8.</td>
<td>Node movement with the mouse.</td>
<td>✓</td>
<td>By click and drag on any node.</td>
</tr>
<tr>
<td>9.</td>
<td>Group node selection and move</td>
<td>✓</td>
<td>Selecting nodes with a rectangle</td>
</tr>
</tbody>
</table>
as a group.

Panel Zooming. ✓ With a slider.

Hand Mode. ✓ Can grab the panel and scroll horizontally or vertically.

Terminal as an alternative way for executing simulations. ✓ --

Default Values for each new network component. ✓ --

Statistical data in the form of text. ✓ --

Statistical data in the form of Graphs. ✓ Creation of bar graphs.

Live Animation while simulator runs. ✗ Considered unnecessary.

Status panel with some application feedback for the user. ✓ --

Show opened xml file in the window’s title. ✓ --

Annotations area changed to be resizable and to support rich text format. ✓ --

| Table 15: Investigated Features |

### 13.1. Future Work

There are many more ideas of new features and improvements that could be applied for this application. The available time was too short for this application’s scope and we tried to stay focused to the most important ones to provide a robust application. The following features or fixes could improve the current project:
• **Alternative design of topology:** By modifying the existing auto-layout algorithms or by making new it would be nice if we could provide some alternative way that could create a network topology automatically. For instance with the completion of a form with how many nodes we would like to have and how these should be connected based on standard topologies (bus, ring, star, extended star, hierarchical, mesh or any custom) and then with a press of a button the topology could be designed.

• **Support for auto generation of subnets:** This feature is needed in order to provide the alternative topology design discussed above. Some clever algorithms should be defined in order to set automatically the subnets and IP addresses based on the topology.

• **Support for wireless nodes:** The trend these years in mobile devices and telecommunications increased the need for scientists and students to experiment or to study wireless networks. Thus both Simulator and GUI should be extended to support mobile nodes.

• **More Statistical Data:** More statistical data should be collected from the Simulator in order to have more meaningful results. Some ideas are data consumption in links, total bytes transferred and total execution time.

• **Improve Simulator’s logging:** Currently there are some lines in the log files of the simulator with buffered data instead of ASCII characters and few other lines that should contain an IP address instead some zero values. With the current project simulator’s log information is directed to text events as annotations in the animation. With the improvement of these log lines the annotations would be even more accurate.

• **Improve current design panel:** While volunteers were testing the application it was noticed that many of them made a direct drag of a node from the button to the panel. In other words they tried to add a node by dragging the button into the panel. Generally speaking, buttons are meant to be pressed and not dragged. But for the buttons add Host, add Router and add Switch things seem to be different. These three buttons have a different meaning for the general user and probably a mouse listener should be added to them to enable adding nodes by a direct drag action.
• **Open multiple files**: During testing we felt many times the need to open two or three simulation configurations at the same time. With the use of a tabbed interface and some minor changes to the code this could be easily achieved.

• **More Information on Packets**: Packets move during the animation and there is limited space on the screen of what information can be shown on them. On the other hand they carry the most important information of a simulation. Therefore another good idea would be to make them clickable like other components of the network. With some modifications in the EventCreator class and a few other classes it would be possible to pass a full packet datagram through the animation. The user would be able to click on packets when the animation is paused and a form could be loaded with the complete datagram.
References

[1] Niall Thomson (2008), A Java-based Network Simulator, Department of Computer Science, Herriot-Watt University, Edinburgh

[2] Tamara Munzner (2000), Interactive Visualization of Large Graphs and Networks, Stanford University, Stanford


[15] Xiang Zeng, Rajive Bargodia, Mario Gerla, GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks, Department of Computer Science, University of California, Los Angeles


http://systems.cs.colorado.edu/~caleb/qualnet_documentation/QualNet-4.5-
UsersGuide.pdf, (Last accessed 28/03/2012)

[19] Daniel Albeseder, Comparison of Network-Simulators, Vienna University of
technology, http://ti.tuwien.ac.at/ecs/people/albeseder/simcomp/simcomp, (Last accessed 28/03/2012)

TOOL FOR WIRELESS SENSOR NETWORKS SIMULATIONS UNDER J-SIM, Superior School of
Technology, Politecnic Institute of Castelo Branco, Department of Informatics, University of
Beira Interior, Institute of Telecommunications - Networks and Multimedia Group, Portugal


21/03/2012)

21/03/2012)


[25] NetAnim Advanced Mode,
http://www.nsnam.org/wiki/index.php/NetAnim_Advanced_mode, (last accessed
28/03/2012)


[27] VirtualBox, https://www.virtualbox.org/, (last accessed 28/03/2012)

Bibliography

Gilberto Flores Lucio, Marcos Paredes-Farrera, Emmanuel Jammeh, Martin Fleury, Martin J. Reed, OPNET Modeler and NS-2: Comparing the Accuracy Of Network Simulators for Packet-Level Analysis using a Network Testbed, Electronic Systems Engineering Department, University of Essex, Essex


IMUNES - An Integrated Multiprotocol Network Emulator/Simulator, http://imunes.tel.fer.hr/imunes/, (last accessed 21/03/2012)


NCTUns 6.0 Network Simulator and Emulator, http://nsl.csie.nctu.edu.tw/nctuns.html (last accessed 21/03/2012)


Phil Levis, Simulating TinyOS Networks, http://www.cs.berkeley.edu/~pal/research/tossim.html, (last accessed 21/03/2012)

Simple Agent Pro, http://www.smplsft.com/SimpleAgentPro.html, (last accessed 21/03/2012)

APPENDIX A: The Initial plan of the Project

The following plan was created during the research period of this project. During the implementation it was not followed strictly because of extra features and changes in the visualization approach.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Study and understand Java GUI</td>
<td>2 days</td>
<td>Mon 5/7/12</td>
<td>Tue 5/8/12</td>
</tr>
<tr>
<td>2. Study and understand JMS Code</td>
<td>2 days</td>
<td>Wed 5/9/12</td>
<td>Thu 5/10/12</td>
</tr>
<tr>
<td>3. Dissertation</td>
<td>77 days</td>
<td>Fri 5/11/12</td>
<td>Thu 8/23/12</td>
</tr>
<tr>
<td>4. Report</td>
<td>71 days</td>
<td>Mon 5/14/12</td>
<td>Thu 8/16/12</td>
</tr>
<tr>
<td>5. Design new GUI MVC</td>
<td>4 days</td>
<td>Fri 5/12/12</td>
<td>Wed 5/16/12</td>
</tr>
<tr>
<td>6. Implement GUI</td>
<td>4 days</td>
<td>Thu 5/17/12</td>
<td>Tue 5/22/12</td>
</tr>
<tr>
<td>7. Implement Controller (Input for JMS)</td>
<td>4 days</td>
<td>Wed 5/23/12</td>
<td>Sat 5/26/12</td>
</tr>
<tr>
<td>8. Import/Export to XML</td>
<td>4 days</td>
<td>Tue 5/29/12</td>
<td>Fri 6/1/12</td>
</tr>
<tr>
<td>9. Visualization approach</td>
<td>20 days</td>
<td>Sat 6/2/12</td>
<td>Thu 6/28/12</td>
</tr>
<tr>
<td>10. Link Java and JMS with</td>
<td>20 days</td>
<td>Sat 6/2/12</td>
<td>Thu 6/28/12</td>
</tr>
<tr>
<td>11. Visualization approach 2</td>
<td>20 days</td>
<td>Fri 6/19/12</td>
<td>Thu 7/26/12</td>
</tr>
<tr>
<td>12. Integrate Java and JMS</td>
<td>20 days</td>
<td>Fri 6/29/12</td>
<td>Thu 7/26/12</td>
</tr>
<tr>
<td>13. Evaluation</td>
<td>9 days</td>
<td>Thu 7/26/12</td>
<td>Thu 8/7/12</td>
</tr>
<tr>
<td>14. Refine</td>
<td>7 days</td>
<td>Wed 8/8/12</td>
<td>Thu 8/10/12</td>
</tr>
<tr>
<td>15. Dissertation Submission</td>
<td>8 days</td>
<td>Sat 8/18/12</td>
<td>Sat 8/18/12</td>
</tr>
<tr>
<td>16. Poster Session</td>
<td>8 days</td>
<td>Thu 8/23/12</td>
<td>Thu 8/23/12</td>
</tr>
</tbody>
</table>
APPENDIX B: Developer’s Guide

**How to implement new node**

**Step 1:** Create the new class in javis.networkElements package. Make that class to extend GNode.

**Step 2:** Create a new button into the SwingSetupPanel class. Then add action to that button in the DesignController class.

**Step 3:** Add code for creation of the new node into the SimSetupModel class.

**Step 4:** Create a new form for its attributes if needed. The new form should be placed in to the javis.iface.forms package and it must extend the AttributesForm class.

**Step 5:** Add new code to the XmlUtils class to enable save/retrieve to/from the XML file.

**Step 6:** Add new code to SimulationCreator class.

**How to implement new animation event**

**Step 1:** Create a new class into the javis.event package. The new class must extend the Event class.

**Step 2:** Create a new class into javis.animation package. The new class must extent the visualElement class.

**Step 3:** Add the new event into the Scheduler class.

**Step 4:** Make changes to the EventCreator class.

**Step 5:** Add the new calls to the JNS code where needed.
How to add a new simulation scenario

Step 1: Add new code to EventInputTerminal class.

Step 2: Add new code to SimulationCreator class.

How to add more statistics

Step 1: Modify JNS to collect more statistics.

Step 2: Modify Stats class to read and show the new statistics when the form loads.
APPENDIX C: User’s Guide

How to add a New Node:

**Step 1:** Press any of the buttons for adding Node (Host, Router, Switch)

**Step 2:** Click on the drawing panel to place the new Node.
How to edit the parameters of a Node:

**Step 1:** Simply click on an existing node. A window with its parameters will load up. Any changes made in this form are saved automatically.
How to edit the parameters of an Interface:

**Step 1:** Click on the Node that you want to edit existing interface.

**Step 2:** Select the interface you want to edit

**Step 3:** Click on the Edit Button. Another form will load with the parameters of the interface.

**Step 4:** Make any changes and press the save button.
How to add a new Interface:

**Step 1:** Click on the node you want to add the new interface.

**Step 2:** Press the New button.

**Step 3:** Complete the form for the parameters of the new Interface and press the Save button.
How to link two Nodes:

**Step 1:** Click on the link button.

**Step 2:** Click on the source Node to start a link from.

**Step 3:** Select the Interface that the link will be connected to, from the list of attached Interfaces.
Step 4: Click on the destination Node. A list of attached Interfaces will load up.

Step 5: Click on the Interface that the link will be connected to.
How to edit the parameters of a Link:

**Step 1:** Simply click on the link. A form with its parameters will load up. Any changes are saved automatically.
How to set a simulation scenario (Form):

**Step 1:** Click on the Scenario button.

**Step 2:** Choose a scenario from the available simulator scenarios and complete its fields.

**Step 3:** Press the save button.
How to execute the specified scenario:

**Step 1:** Click on the Run JNS button. The simulation will be executed and the animation will begin automatically.
Alternative direct execution of a simulator scenario (Terminal):

**Step 1:** Click on the Terminal Tab.

**Step 2:** Write “ping 0 192.168.1.2” for a Ping scenario that will be executed from node with ID 0 and to the IP address 192.168.1.2. By pressing the Enter key on the keyboard it will run the simulator and perform the animation. For available scenarios and commands write help. Write “reset” to reset the simulator and clear statistical information.
How to delete a node or link:

**Step 1:** Press on the Cut Tool button on the panel.

**Step 2:** Click on the node or link to delete.
How to move a group of nodes at once:

Step 1: While on Mouse Mode click and hold the mouse button on the start point.

Step 2: While holding the mouse button move the mouse to the end point.

Step 3: Release the mouse button.

Step 4: Drag any selected node.
How to view statistical data:

**Step 1:** Click on the Stats Button. A window with graphs will be showed. Additionally scroll down and press “View results in a text version” for a text version of results.
APPENDIX D: The two tasks for the Evaluation

**Task 1:**

Please follow the steps in order to design a simple network with two hosts that will exchange data.

1. Create a new Host and place it into the panel.
2. Create another Host and place it into the panel.
3. Add a link from the host’s interface from step 1 to the host’s interface in step 2.
4. Set up a Simulation scenario of Send Data from host of step 1 to the host of step 2 with data to be sent equal to 5000 bytes.
5. Run the JNS.
6. View the animation and then the Statistics.

**Task 2:**

Please follow the steps in order to design a network and execute a ping scenario:

1. Create a Host and place it into the main panel.
2. Create another Host and place it next to the previous one.
3. Create a switch and place it below those 2 hosts.
4. Create a router and place it to the right of the switch.
5. Create a host and place it to the right of the router.
6. Go to the properties of the placed router in step 4 and add a new interface with:
   - Hardware Address: 12:23:34:45:56:67
   - IP Address: 192.168.2.1
   - Subnet Mask: 255.255.255.0
   - MTU: 1500
7. Go to properties of Host placed in step 5 edit the current interface and change the following fields:
   - IP Address: 192.168.2.2
   - Set Default Route: Check this box.
   - Default Route: 192.168.2.1
   - Save.

8. Now go to properties of the host placed in step 1 edit the current interface and change the following fields:
   - Set Default Route: Check this box.
   - Default Route: 192.168.1.4
   - Save.

9. Add a link between Host’s interface in step 1 and the Switch

10. Add a link between Host’s interface in step 2 and the Switch


12. Add a link between the router’s interface with hardware address 12:23:34:45:56:67 to the host’s interface of step 5.

13. Now go to terminal and with the use of terminal’s help find out how to execute a ping command from host we created in step 1 to the host we created in step 5.

14. View the animation and then the Statistics.
APPENDIX E: A copy of the online survey.

1. In which category do you assign yourself according to your knowledge in networks.
   - Beginner
   - Intermediate
   - Advanced

2. Which of the given examples did you try to perform?
   - Example 1
   - Example 2
   - Both

3. In which of the examples did you succeed?
   - Example 1
   - Example 2
   - Both

4. How easy was it to perform the examples?
   - Very Easy
   - Easy
   - Easy to Normal
   - Normal
   - Normal to Difficult
   - Difficult
   - Very Difficult

5. Please write any comments negative or positive about the Graphical Network Simulator you just tried.

6. Do you think this program is useful for educational purposes?
   - No
   - Yes
   - I have used similar application before and it was quite helpful.
   - I wish I had such an application while I was learning networks.

7. Do you think this program can be used as a platform to develop new protocols and to test new network applications?
   - No
   - Yes
APPENDIX F: A simple XML Simulation Configuration

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<SimConfig>
  <Host>
    <id>0</id>
    <pos>42x81</pos>
    <name>Sender</name>
    <protocol>
      <protoID>1</protoID>
    </protocol>
    <protocol>
      <protoID>2</protoID>
    </protocol>
    <protocol>
      <protoID>3</protoID>
    </protocol>
    <protocol>
      <protoID>4</protoID>
    </protocol>
    <protocol>
      <protoID>5</protoID>
    </protocol>
    <interface>
      <ip>192.168.1.2</ip>
      <droute>true</droute>
      <DedaultRouteIp>192.168.1.1</DedaultRouteIp>
      <SubnetMask>255.255.255.0</SubnetMask>
      <MTU>1500</MTU>
    </interface>
    <isRouter>false</isRouter>
  </Host>
  <Host>
    <id>1</id>
    <pos>282x146</pos>
    <name>Foo Host 1</name>
    <protocol>
      <protoID>1</protoID>
    </protocol>
    <protocol>
      <protoID>2</protoID>
    </protocol>
    <protocol>
      <protoID>3</protoID>
    </protocol>
    <protocol>
      <protoID>4</protoID>
    </protocol>
    <protocol>
      <protoID>5</protoID>
    </protocol>
    <interface>
      <ip>192.168.1.3</ip>
      <droute>true</droute>
      <DedaultRouteIp>192.168.1.1</DedaultRouteIp>
      <SubnetMask>255.255.255.0</SubnetMask>
      <MTU>1500</MTU>
    </interface>
    <isRouter>false</isRouter>
  </Host>
  <Host>
    <id>2</id>
    <pos>534x55</pos>
    <name>Receiver</name>
    <protocol>
      <protoID>1</protoID>
    </protocol>
    <protocol>
      <protoID>2</protoID>
    </protocol>
  </Host>
</SimConfig>
```
<ip>192.168.2.1</ip>
<ddefaultRoute>false</defaultRoute>
<DdefaultRouteIp>0.0.0.0</DdefaultRouteIp>
<SdefaultRouteIp>255.255.255.0</SdefaultRouteIp>
<MTU>1500</MTU>
</interface>
</Host>
<isRouter>true</isRouter>
</Switch>
</Switch>
</Switch>
</Switch>
</Switch>
</Switch>
</Switch>
</Switch>
</Switch>
</SimulationScenario>
<SimType>6</SimType>
<Sender>0</Sender>
<Recipient>2</Recipient>
<DestIP>192.168.2.3</DestIP>
<Bytes>5000</Bytes>
<QuoteFrom>0</QuoteFrom>
<QuoteTo>2</QuoteTo>
<QuoteIP>192.168.2.3</QuoteIP>
<PingFrom>0</PingFrom>
<PingIP>192.168.2.3</PingIP>
</SimulationScenario>
</SimConfig>