UNIVERSITETET I OSLO
Institutt for informatikk

Revising the User Interface of NEMAN

Master thesis

Suleiman H. Jama

August 1, 2007

Preface

This thesis is the final part of my Master Degree at the University of Oslo Department of Informatics. The thesis has been carried out under the supervision of Professor Thomas Plagemann and Phd. student Matija Pužar (Author of NEMAN) for the Distributed Multimedia Systems (DMMS) research group, the thesis itself is part of the Ad-hoc Infoware project.

In this thesis we revise the graphical user interface (GUI) of the network emulator NEMAN. We discover that this interface has both technical issues and might also introduce copyright violation.

Based on these facts we have designed and developed, with the C++ programming language and the Qt framework, a GPL licensed enhanced user interface, which consists of both a graphical and a command line interface. We have performed several analyses on both the old GUI and the new interface in order to compare the performance.

Acknowledgements

First and foremost I want to thank my supervisors, Professor Thomas Plagemann and Matija Pužar, for their valuable time, guidance and feedback on both the technical and the theoretical aspects of this thesis.

I am also grateful to the PhD and Master students who contributed to this thesis, by participating in the questionnaires.

I would also like to thank my friends Tatjana Andersen, Tonje Klykken, Ahmed Mohammed and Pål Grønsund for reading the report and providing suggestions.

Finally I want to thank my wife Anab Ibrahim and the rest of my family for supporting me throughout this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Mobile Ad-hoc Networks also known as MANETs (the term "Ad-hoc" is Latin and stands for "for this purpose") are wireless and infrastructure less networks that are setup and configured for a particular purpose. During the recent years MANETs became popular subject for research because laptops and even smaller devices such as mobile phones and PDAs are equipped with wireless network interfaces. Devices in a MANET communicate with each other over wireless channels, without any centralized control.

MANETs are highly unstable and dynamic networks in terms of nodes moving randomly, which cause network partitioning (frequent topology change) and at unpredictable time [23]. Network partitioning occurs because the random moving nodes with a high or low degree of mobility and in a dense or scattered area are disconnected. This also affects the communication in a MANET.

There is no infrastructure in a MANET, communication is done from one node to another within the same radio range, or packets are routed via other nodes until they reach the destination. MANETs have their own routing protocols designed specifically for them. With the help of these routing protocols, a route in the network is discovered; this is done by broadcasting to other nodes within the radio transmission range of the sender.

The available network resources and connectivity are also affected. Furthermore, the devices in a MANET are very heterogeneous consisting of Laptops, PDAs and even Mobile Phones with different capabilities to store and forward data packets.

Development and especially evaluation of MANET protocol solutions in the real world are consuming time and resources. However the protocol and applications running over MANETs require sufficient testing both in the real world and in network simulators or emulators. The benefit of using simulators and emulators exceeds in many cases real world testing and saves the developers time and money. Even though real world testing is the ultimate way of testing protocols in a mobile device, when considering the real world hindrance for radio transmission and other obstacles that may cause packet delay, packet collision, hidden node problem etc. Simulation of the lower network layers can model all these known problems under a controlled environment, with repeated parameters the results obtained from a simulation can be very close to a real world testing.

1.1 Motivation and Problem Description

In this thesis, we study a network emulator called NEMAN [20] developed for the Ad-hoc Infoware project, the goal of this project is to develop a set of services for information access and sharing in MANETs. This includes development of middleware services for MANETs with the focus on emergency and rescue operations[9].

NEMAN is developed with the purpose of being cost efficient and at the same time scalable enough to emulate large scale wireless mobile networks on a single physical machine. NEMAN is divided into the core, which is the Topology Manager and a Graphical User Interface (GUI) which is the controlling interface. The GUI has not been designed from scratch along with the Topology Manager, instead the GUI of the MobiEmu project [23] has been modified to be used with NEMAN.

The GUI, which is from another research group, has introduced technical problems related to runtime performance, and could also, introduces copyright related issues. The main issue of runtime performance is related to the fact that the GUI is written in Tcl/Tk, which is a interpreted scripting language widely used in development of simulator interfaces. Interpreted languages tend to be slower in performance in comparison to compiled languages. This GUI is the bottleneck on the size of scenarios used in the emulation. A scenario file describes events that take place in the emulation. Such events are the nodes' movements, link changes, duration of the test and communication between the nodes. When we say that it is the bottleneck on the size of the scenario, it means that the visual performance of the GUI is affected by how many nodes the scenario contains, their mobility speed and frequent link change. Each node in the GUI is a graphical object that is drawn when it moves, when this node is within communication range of another node, a graphical object that represent the link status between those nodes are also drawn.

However, this performance issue only affects the visual experience, and not the network communication. The user performing an emulation will experience that there are delays in loading the scenario file into memory, and while drawing graphical objects that are used to show the communication link status among the nodes. This depends on the length of the scenario in time, how many nodes used and if there are often link changes among the nodes.

The GUI is not licensed to GPL[1]; it is only authorized to be used if the author is credited. A copyright violation might occur if a modification is done to the source code, and then published. However, it is unclear and the author never responded to our questions regarding these issues. A GPL licensed GUI would allow us to modify and distribute it under the conditions of the GPL, without apprehending any copyright violation . Based on the

[1]GNU stands for General Public License and the most widespread such license is the GNU General Public

mentioned problems, this thesis focuses on how the GUI can be improved by either modifying the existing code or developing a new GUI with a compiled programming language.

We decided to avoid using the Tcl/Tk language, and instead decided to develop a new GUI with the C++ programming language, and the Qt toolkit as the chosen framework. This toolkit is a cross platform graphical toolkit for GUI development.

NEMAN has to our knowledge no other user interface other than the modified GUI from the MobiEmu project. We conclude that an independent GPL-based license and fast GUI is needed. Furthermore, feedback from the users and the author of NEMAN show that a command line version of the GUI is also required.

1.2 Claims

In order for MANET researchers to do their work efficiently and with a high degree of reliability we have in this thesis, designed and developed an enhanced user interface for the network emulator NEMAN. This interface consists of two parts, a graphical interface which provides the same functionalities as the earlier GUI, but with additional features and a command line interface which is easily accessible and provides quick emulation setup.

Throughout this thesis, we call our interface which consists of a GUI and a command line for NEMAN Interface 1.0, while the original GUI is called NEMAN Interface 0.5. We have indentified the bottleneck in the visual performance of NEMAN Interface 0.5, and have carried out several benchmark tests on both applications in order to compare them.

In our comparison we have discovered significant difference in performance between the interfaces, and our NEMAN Interface 1.0 is much faster in both loading a scenario file and also in drawing of communication links between the nodes.

We claim to have developed:

- A much faster GUI in NEMAN Interface 1.0 than NEMAN Interface 0.5
- A GUI that includes the same functionalities as the GUI in NEMAN Interface 0.5 including additional features
- A command line interface, which is flexible and provides the ability to do customized emulation experiments

1.3 Method and Approach

The work of this thesis is divided into a theoretical and a practical part.

The theory part consists of literature study with the mixture of books and research papers in topics covering MANETs and network simulators and emulators and user interfaces. We have also evaluated the toolkits and programming languages used in the development of the new interface. In the second part we studied the Qt framework, which is the chosen toolkit, while designing and developing a prototype.

This approach has helped us to gain the theoretical knowledge of the research area and to become familiar with the Qt library early in the process of this thesis. The implementation of NEMAN Interface 1.0 and writing of this paper along with all the results have been carried out in parallel.

1.4 Reader's Guide

The structure of this thesis is divided into 5 chapters.

- **Chapter 2**, here we introduce the theoretical background on MANETs, and related Network simulators and emulators with high standard graphical interfaces. We also compare the studied toolkits with Qt.

- **Chapter 3**, here we outline the design criteria and requirements to develop the new interface, along with analyses on feedback from the questionnaire on user experience of NEMAN Interface 0.5. We also take the NS-2 Nam and OMNet++ user interfaces into consideration, to see the possibility and benefit of porting these interfaces into NEMAN.
- **Chapter 4**, here we present our implemented solution of NEMAN interface version 1.0 based on the design outlined in Chapter 3. We present our implementation in two parts the graphical solution and the command line solution.
- **Chapter 5**, the implemented interface is critically analyzed and evaluated, and compared with version 0.5.
- **Chapter 6**, finally we give an overall summary and discussion about the whole process also noting the things that can be improved. Comments from the end user questionnaire is presented in Section 6.2

Chapter 2

Background

2.1 Mobile Ad-hoc Networks

A Mobile Ad-hoc Network also called MANET is an autonomous system of mobile nodes connected by wireless links, with each node operating both as end-node and as a router to forward packets. The nodes are free to move randomly and organize themselves arbitrarily. This allows a dynamically changing topology, but at unpredictable times. In dynamically changing topology the wireless links can be both bidirectional and unidirectional.

Compared with wired networks when it comes to speed, Manet's physical bitrates operate typically between 1 Mbit/s and up to high rate 108 Mbit/s (802.11n) which is slower than 100-1GB on wired networks. However this bottleneck is not caused by the MANET itself being bandwidth constrained, but rather the outside network, e.g., ADSL lines operates at maximum of 8 Mbit/s throughputs, which is much slower than the wireless network which it is connected to. Most devices in a MANET rely on batteries as an energy source, thus they are energy constrained devices, and energy conservation is needed. A MANET may operate in isolation or may have gateways to a fixed network.

The earliest MANETs were called packet radio networks, and were developed for military purpose by DARPA in the early 1970's. Later SRI International Design and BBN Technologies built and experimented with these earliest systems. MANETs where part of the motivation of the original Internet Protocol suite. Over the years many MANET routing protocols have been suggested, among them we find two protocols which the IETF MANET group is working to standardize. These are accepted as experimental RFCs (Request For Comments, in order to be adapted as internet standard).

- The Ad-hoc On-Demand Distance Vector (AODV) which is a reactive protocol.
- The Optimized Link State Routing (OLSR) which is a proactive protocol.

Protocols which are reactive try to setup a route on demand, the sender request to establish a route to the node it want to communicate with. Proactive protocols aims to let all the nodes in the network know topology change of the network, this results in an overhead of routing traffic, but no delay in communication. MANETs have many application areas such as emergency situations like natural or human-induced disasters, military conflicts, emergency medical situations etc.

Now that we have seen what MANETs are, and how they work, the protocols which operate on MANETs needs to be developed, debugged and tested in environments which can produce the same behavior as MANETs. Whether this behavior is a real MANET, software simulation or emulation, we explain in the next Section.

2.1.1 MANET Testing Approaches

To develop, test and debug new protocols, network emulators and simulators are widely used. Simulators and emulators allow us to have repeated tests that are realistically close to real world test beds. This means that, the executed protocol produces the same results as it would have done in a real world test bed. Network simulators, such as NS-2 [4], provide the necessary tools to conduct customized simulation experiments. However, the provided functionalities by the supported modules in network simulators are merely logical operations rather than real implementations. A modification of the protocol implementation in network simulators is necessary before being deployed to a target network. With emulators we avoid this, since network emulation includes execution of real network protocol implementation code, in a controllable and reproducible laboratory network environment [23].

MANETs can be tested within three different approaches, but first let us explain what test bed is: Test bed is the framework used to conduct scientific tests to test, compare and evaluate protocols and algorithms in the real world.

- **Real world as MANET Test bed** Real World Test bed is a test approach that involves usage of equipment and tools to measure and carry out tests on the computer program under development. The program is usually designed for the equipments and tools used. Normally in a Real World MANET test bed, people carry around the devices, which are running a MANET protocol under test.

  Real World test bed might be the ideal testing approach, since the environment consist of obstacles, buildings, landscapes which can interfere with the MANET, and eventually the implementation must be carried out in a real world test bed before the release of the product. It is however expensive, especially when it comes to testing networks consisting of hundreds of nodes where equal number of participants and equipments are needed. Not to mention that it is difficult to do repeated tests where the same parameters apply, due to random events in the scenario [7]. For real scenario as testbed we conclude that it is neither scalable nor reproducible, though it is theoretically the best test environment for its realism [23].

- **Simulation as MANET Test bed** where software applications are used to model the behavior of a MANET. Simulators can produce a controllable and repeatable test bed environment. As mentioned in Section 2.1.1, the protocol and algorithm modules are computer-modeled instead of real implementation, which is why designers have to implement the protocols twice, once for simulation and once again for deployment [11].

As shown in Figure 2.1, all layers in a simulation are done in software, and are used to describe the simulation of the specific wireless issue and the modeling of the wireless channel. In a simulation it is cheaper to model the behavior of MANETs than with real world simulation. This gives a detailed model of lower layer's behavior, but code needs to be completely rewritten in order to be used on actual physical devices [20].

- **Emulation as MANET Test bed** is capable of testing real implementation of routing algorithms, and protocol stacks. Emulators allow computer software to run on another platform other than the one for which the software was intended for, this is also known as Virtual Machines (VM). Emulation allows the test bed to be cost efficient and present a trade-off between real test beds and simulators, providing a virtual wireless network at the lowest layers, and yet allowing real code to be run in the higher layers. A protocol solution developed for an emulator only needs minor changes when porting to the devices or platform the software is intended for. This reduces the amount of resource in matter of manpower that either develops the system or carries out sequences of field testing.

Before we begin to look closely at these approaches we need to clarify the difference between emulators and simulators and the terms simulation and emulation.

Simulation is used in this thesis to indicate the process of using computer software to model the wireless network, without interacting with network processes. Emulation is the process of simulating a system with the usage of a network emulator that is running on real processes. With both emulation and simulation we are interested in modeling the accuracy of our protocols, but emulation provides the emulated process to interact with real processes, in the same way the final release of the protocol is intended to behave on a device.

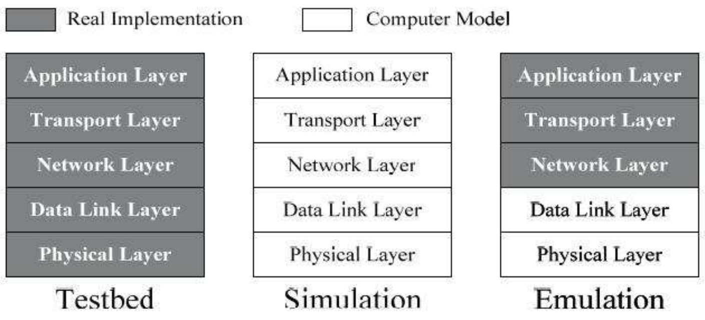


Figure 2.1: MANET protocol test approaches, according to [11].

2.2 Network Emulators and Simulators

As we have understood simulators and emulators provide the necessary ability and test bed environment for applications and protocols designed for MANETs. However, there exist many simulators and emulators with different capability and properties. In order to develop a high standard user interface for NEMAN, we study the interfaces of related simulators and emulators.
The simulators and emulators we have studied, which are NS-2, MobiEmu, GlomoSim, J-sim and OMNet++, have high quality graphical user interfaces, but developers behind these projects have had extended resource and time to develop and maintain them. To expect that we can develop a user interface which can compete with the user interface of e.g OMNet++ [5] during the short time this thesis is given is highly unlikely. An interface which is designed in a good way can continue to be maintained, and with the ability to be extensible, new features can be added over time.

In Gokturk et al. [10] we find that given the high rate of progress in networking research, there is considerable pressure on the resources that network researchers can set aside, for implementing models that are as similar as possible to deployable format for the protocols which is under development. Without an architecture that ensures to isolate the models among themselves, such lack of resources leads to "quick-hacks" style of programming. When these implementations are disseminated among the research community they become part of the simulators, which makes the simulator code hard to manage over time. NS-2 which is explained in Section 2.2.1 is an example to this kind of process, while the OMNet++ kernel is a class library, and you can write your components like any other library, without modifiying OMNet++ source code anywhere. This enforces reusability and in contrary NS-2 is more monolithic to add modules into it, where you have to modify the source code in several places.

To conduct our development of the user interface without falling into the problems mentioned above, we select to study the provided interface of NS-2, OMNet++ and MobiEmu.

2.2.1 OMNet++ and Tkenv

Although OMNet++ is fully object-oriented and component based simulator, we are interested in how the user interface of this simulator works. NS-2 have Tcl script based models, while OMNet++ uses NED based which is editable with the GNED graphical tool. However the current graphical interface of NEMAN also uses Tcl models. The GUI of OMNet++ Tkenv is an interactive execution environment. Tkenv supports examination during execution, and is good debugging tool. Even though it does not support packet flow, or animation like Nam does.

After all OMNet++ simulator has high quality and is gaining wide spread acceptance in the research communities, however we are only interested in the user interface it provides and the possibility of porting it to NEMAN. Since the Tkenv does not support processing of Tcl based models, and animation, we cannot claim that the GUI of OMNet++, Tkenv is a good candidate for NEMAN.

2.2.2 NS-2 and Nam

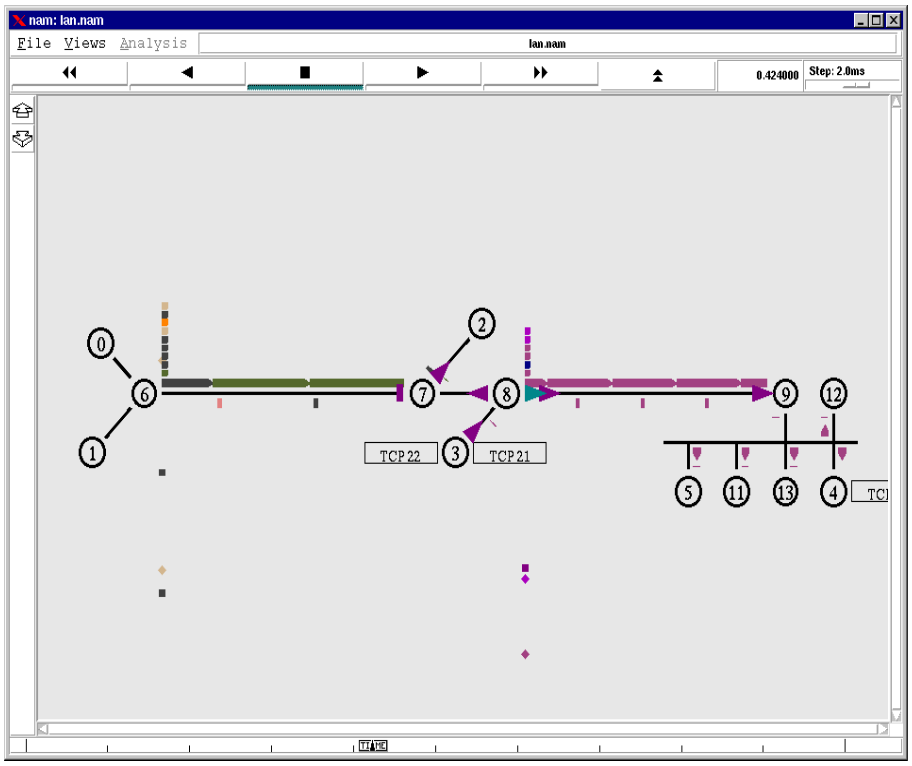


Figure 2.2: The Network Animator Nam

The Network simulator NS-2 is a widely accepted discrete event Network simulator, actively used for wired and wireless Network simulation [16].
NS-2 provides good support for TCP, routing and multicast protocols simulation over local and satellite networks and several Ad-hoc routing protocols.
NS-2 is written in C++ and Object Tcl (OTcl), it is provided with a GUI called Nam (Network Animator), Nam is Tcl/Tk based animation tool for viewing network simulation traces and real world packet traces [3].
As we can see Nam in Figure 2.2, shows animation of TCP based scenario,

Nam is also capable doing packet animation. Further in the Figure we can see the options this GUI offers, such as start simulation, stop, pause and steps which can move forth and back in the animation. Nam is also provided with an editor, that configures or creates scenarios, it can be done with drag and drop fashion, instead of writing the whole scenario.

Nam is a good candidate as a graphical interface for NEMAN, because it is capable of processing Tcl models and it provides a good simulation animator. Even though NS-2 script models (which are whole simulation specification such as topology star, ring etc. Agents which is the protocols and nodes link information) and Nam, trace files are not the same, Nam can only run the trace files which describes node movements and packet routing information. The trace file is itself generated out of the NS-2 script model. This means Nam is not used to control the simulation parameters in NS-2; it is rather used to develop the script models and run the trace files.

Nam's design theory is to be able to read large animation data set and also to be extensible so that it can be used with different network visualization situations. To avoid slowness and at the same time handle large animation data sets, Nam keeps as little as possible of information in memory, and event commands are kept in the trace file and re-read from the file whenever necessary.

We found out that to port NS-2 graphical interface Nam into NEMAN would take substantial amount of changes on both NEMAN, to provide the necessary interface and communication channels, and Nam would also have to be adapted in many places. We also note that, NEMAN requires a GUI to run the emulation with, not only a GUI that shows the animation of a specific scenario. It is also preferable to avoid introducing another GUI which is implemented with interpreted language.

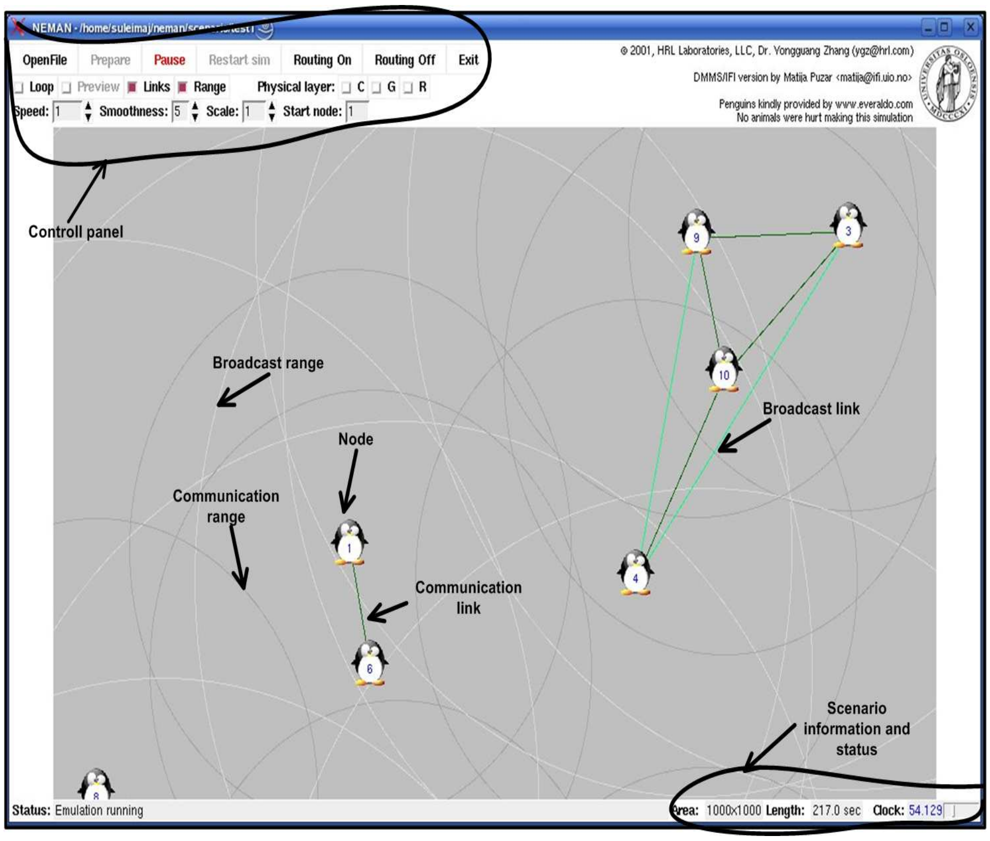

NEMAN - /home/suleimaj/neman/scenario/test1
OpenFile
Prepare
Pause
Restart sim
Routing On
Routing Off
Exit
Loop
Preview
Links
Range
Physical layer: C G R
Speed: 1
Smoothness: 5
Scale: 1
Start node: 1
© 2001, HRL Laboratories, LLC, Dr. Yongguang Zhang (ygz@hrl.com)
DMMS/IFI version by Matija Puzar <matija@ifi.uio.no>
Penguins kindly provided by www.everaldo.com
No animals were hurt making this simulation
Controll panel
Broadcast range
Node
Communication range
Communication link
Broadcast link
Scenario information and status
Status: Emulation running
Area: 1000x1000 Length: 217.0 sec Clock: 54.129

Figure 2.3: GUI of MobiEmu used in NEMAN as version 0.5

2.2.3 MobiEmu

In Wei Li et al. [23] an inexpensive and flexible MANET emulator called MobiEmu is introduced, this emulator is capable of testing Ad-hoc Networks of virtually any scale. The system uses a fixed network of n Linux machines to emulate a MANET of n nodes, thus it is a distributed controlled emulation test bed.

Connectivity changes are indicated by a central server, which also displays the actual network topology. MobiEmu also facilitates the use of User Mode Linux (UML), which allows to run several virtual node instances on a single physical machine [14]. The GUI of this emulator as previously mentioned is adapted into NEMAN, and works as the controlling interface for NEMAN. As already indicated in [23] it is a known problem to the authors of MobiEmu that the disadvantage of using Tcl/Tk based GUI in their emulator is in speed performance. The graph update can sometimes lag[1] behind the emulation time if large number of nodes and links are displayed. This can be compensated by either using a faster computer or reducing the smoothness value, which is an option in the GUI provided to adjust how smooth the drawing should be.

Figure 2.3 shows the GUI of MobiEmu and NEMAN, there has been done modifications to it, in order to work with NEMAN. Also new functionalities are added such as simulation of physical layer models, start and stop routing daemons, ability to scale (zoom in and out) and to set start node. More about the GUI is explained in Section 2.2.4, in the GUI Section.

In Section 2.2.4 we take a close look into the NEMAN architecture in light of NS-2 and MobiEmu's architecture and user interface.

[1]lag is a symptom where result of an action appears later than expected, lag is symptom of latency in computer networks

2.2.4 NEMAN

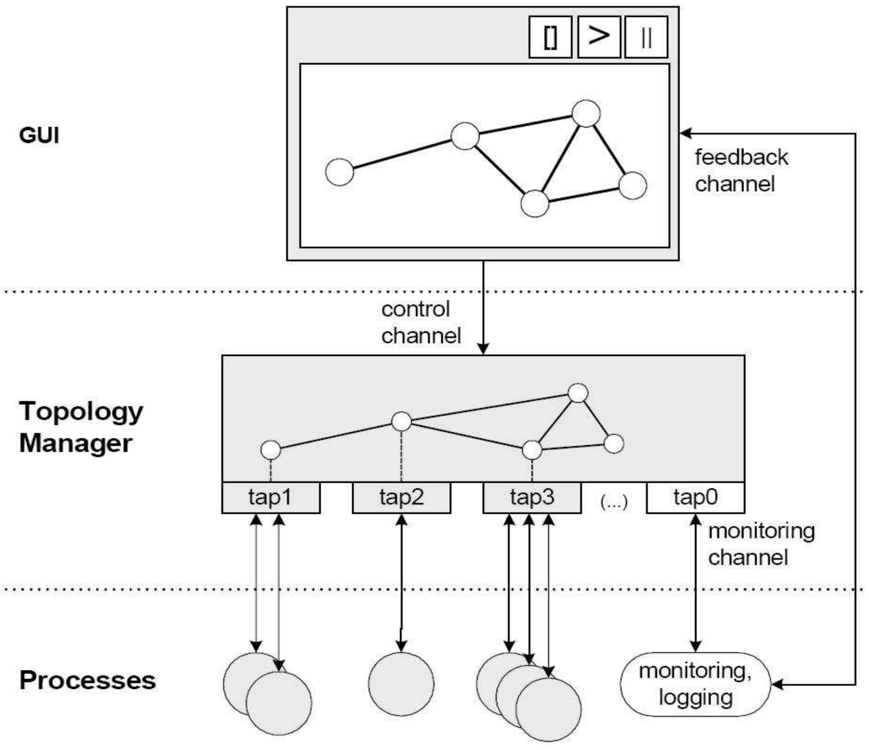


Figure 2.4: The NEMAN Architecture according to [20]

NEMAN is a monolithic based Network Emulator for Mobile Ad-hoc Networks. This means that the emulation setup consist of a single physical machine which holds all virtual nodes, and at least one communication layer is real implementation [13]. NEMAN allows emulation of a whole wireless network on a single Linux machine [19].

With Linux kernel patch sts_2.4.19.patch NEMAN is able to connect virtual network devices (TAP-devices are available in the Linux kernel and provide low level Ethernet tunneling [20]) according to the information given to the topology manager [14], and allows user level processes to hook to them as any other network interface through standard network sockets, with the SO_BINDTODEVICE socket option. The topology manager can be hocked to real processes with the special socket option (SO_BINDTODEVICE) this allows us to a run real user process on a virtual tap-device and will not interfere with packets sent to other processes. Since everything is running on a single physical machine. This allows us to test middleware protocols or application layer processes.

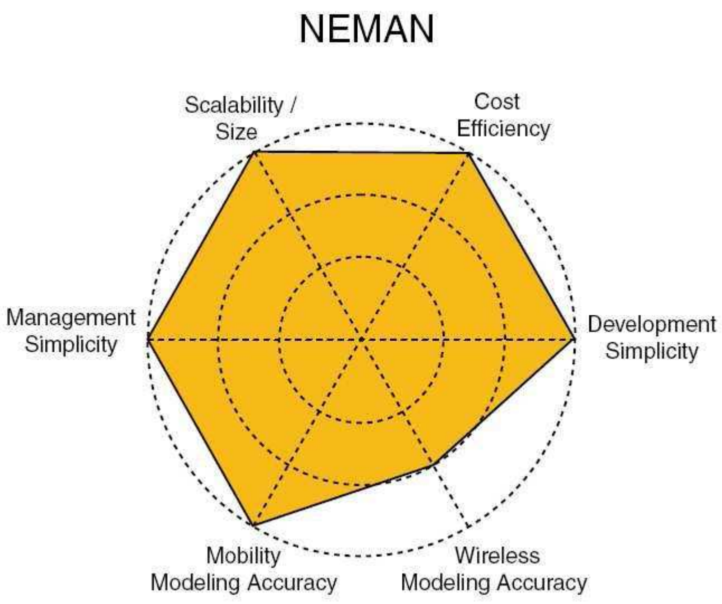


Figure 2.5: NEMAN evaluation from [12].

NEMAN is devided into two parts: The Topology Manager and The GUI.

Topology Manager

The topology manager, also called "TopoMan", is the core of NEMAN and runs on a network machine as a server waiting for instructions. TopoMan listens to control channel on UDP port 3685 for commands sent by the GUI, the commands will be explained in section 2.2.2. In Jonhsen [12] several features are added to NEMAN such as collision detection, random packet loss and gray-zone simulations. Johnsen also evaluated NEMAN shown in Figure 2.5 based on the criteria outlined in [14].

The GUI

In the GUI the user controls the emulation parameters, and its main task is to send topology information to TopoMan, while animation displays the nodes movements and link changes. The animation is just for user's visualization and indicates nodes movement and communication capability. The GUI does not compute the link change between nodes or speed or movement, it reads static information defined in the scenario file and forwards the state of each node according to the timestamps, to TopoMan, which computes changes for the tap-devices.

The GUI is responsible for enabling the needed number of tap-devices before emulation is started, and when it ends, the GUI must disable tap-devices that are not used. This is because several users can be running different scenarios on the same topology manager, therefore sharing all the tap-devices. For that reason we should avoid to enable or disable tap-devices that are used by other people, the GUI controls this by sending UDP command packets to TopoMan which either disables or enables the tap-devices used.

The below statements are valid commands to TopoMan according to the README file in the NEMAN archive [19]

- simphy <0|1> <0|1> <0|1> Turns on or off physical simulation parameters (Collision detection, gray-zones and Random packet loss simulation)
- reset <FIRST> <NUM> resets links between <NUM> nodes starting from node. This is done every time the user wants to start an emulation, to reset the tap-devices from a previous emulation.
- enable <FIRST> <NUM> After the devices are reset, we can turn on tap-devices from FIRST to NUM, in order to be used in an emulation.
- link <node1> <node2> <1|2|n> When the emulation is running, and link information or change has been read from the scenario file, this

command sends link status between node1 and node2 where link status 1 means within communication range, 2 means within broadcast range and everything above are out of range.

- disable <FIRST> <NUM> When the emulation has finished the tap-devices are turned off, from FIRST to NUM.
- route <cmd> <node1> <node2> <gw> sets the route (cmd=1) between <node1> and <node2> through the gateway node <gw>. The parameter <gw> is omitted if the route is being deleted (cmd=0).
- hopbyhop <FIRST> <NUM> <0|1> switches the hop-by-hop option (0=off, 1=on) for <NUM> nodes starting from node <FIRST>. The default is 1 since this is what we usually do want, but can be turned off for some nodes if needed.
- hops <node1> <node2> gives you the route (all the hops) between the two nodes, according to the information it got from the routing daemons.

Route, hop-by-hop and hops are not integrated in the GUI, and is therefore not to be considered.

Below we describe the functionalities NEMAN interface 0.5 provides.

1. **Open scenario files:** which open a scenario file and parse the contents, and puts the nodes on their start position.
2. **Prepare:** shows communication and broadcast link between nodes, this is used to show which nodes that are connected to each other without starting the emulation.
3. **Pause:** pauses the emulation.
4. **Restart emulation:** restarts the emulation.
5. **Routing on:** turns on routing daemon, this starts the used routing protocol which is the Optimized Link State Routing protocol (OLSR).

6. **Routing off:** turns off routing daemon.
7. **Exit:** safely exits the application and kills started process such as udprecv, used to receive UDP packets from TopoMan.
8. **Loop:** Restarts the scenario when it finishes and loops forever.
9. **Preview:** Runs a scenario without sending any topology information to TopoMan, is used to test scenario without emulating it.
10. **Links:** Shows or hides links.
11. **Range:** shows or hides ranges.
12. **Physical layer:** Turns on or off following physical layer simulations Collision detection, gray-zone and random packet loss.

The spin boxes speed, smoothness and scaling is provided to adjust the visualization of the scene while start node allows the user to share tap-devices with other users which is doing emulation on the same topology manager. For example an agreement with another user which is specifying not to use the first 0-99 tap-devices. Thus start node for this user should be 100.

Table 2.1 describes the steps which the GUI sends UDP packets to TopoMan in a short scenario file consisting of 3 nodes.

In order to understand why the visual appearance on NEMAN interface 0.5 is lagging, and how we can do improvements to gain more efficiency. We need to investigate the tools and programming language which the GUI is built upon, and understand the root cause of performance reduction. This comparison study is done in Chapter 5.

The programming language which the GUI of NEMAN is written in is Tcl/Tk, and was originally written for the MobiEmu [23] emulator, and modifications has been done to it to adapt it to NEMAN. The GUI source code consist of one Tcl/Tk source file named "‘iemul"’ and two C source files UDPSend.c and UDPReceive.C.

Time	**Command**	**Nodes**
	reset	0 2
	enable	0 2
0.0	link	0 1 1
21.0	link	0 2 2
36.0	link	1 2 2
47.0	link	0 2 1
47.0	link	0 1 1
64.0	link	0 2 2
65.0	link	1 2 2
76.0	link	1 2 1
86.0	link	0 2 1
	disable	0 2

Table 2.1: Small scenario with 3 nodes.

Scenario Files

The user loads a desired scenario file for the emulation. The scenario file is a list of time stamped location and movement definitions for all nodes. Currently the GUI accepts two types of file formats, the native ns2 format and an extended version generated with the mkdist[2] tool.

The scenario file contains initial coordinates for each node, the position of each node is given with three coordinates X,Y and Z. The scenario example below is the same scenario as explained in Table 2.1, the Z value set to 0.0 indicates we only have a two dimensional scene. The distance between a pair of nodes is given by "setdest" and the value 16777215 is used to indicate that the node is out of reach. The wireless range for transmitters is set to 250 units for this scenario.

```
$god_ set-dist 0 1 1
```

[2]mkdist is a perl script that takes an ns-2 scenario file as input and changes it to use the chosen steps as distance values [12].

In the statement above 1 means the two nodes are within 250 transmission range limit. A distance value larger than 2 (broadcast range which is used to simulate gray-zones) also means that the node are out of reach.

```
$node_(0) set X_ 378.182715892472
$node_(0) set Y_ 93.525072225800
$node_(0) set Z_ 0.000000000000
$node_(1) set X_ 344.861227462892
$node_(1) set Y_ 66.120219816745
$node_(1) set Z_ 0.000000000000
$node_(2) set X_ 103.168148357176
$node_(2) set Y_ 357.655624447909
$node_(2) set Z_ 0.000000000000
```

The lines above state the initial coordinates for each node in the scenario.

```
$ns_ at 0.0 "$node_(0) setdest 397.600517 377.506390 8.747347"
$ns_ at 0.0 "$node_(1) setdest 333.501292 213.787203 2.962066"
$ns_ at 33.0 "$node_(0) setdest 334.649698 88.546541 6.429074"
$ns_ at 42.0 "$node_(2) setdest 171.948357 348.823958 7.106497"
$ns_ at 50.0 "$node_(1) setdest 333.501292 213.787203 0.000000"
```

```
$god_ set-dist 0 1 1
$god_ set-dist 0 2 10
$god_ set-dist 1 2 10
$ns_ at 21.0 "$god_ set-dist 0 2 2"
$ns_ at 36.0 "$god_ set-dist 1 2 2"
$ns_ at 47.0 "$god_ set-dist 0 2 1"
$ns_ at 47.0 "$god_ set-dist 1 2 1"
```

All statements with

```
 $ns_ at [time]
```

show events that occur at given timestamps e.g. after 33.0 seconds node 1 ($node_(0) is actually node 1 and mapped to tap-device 1 when scenario begins) moves to the specified coordinates. The $god_ setdist statement says that the distance between a pair of nodes changes, 1 means that the nodes are within direct communication range, 2 means as mentioned earlier that the nodes are within broadcast range, thus we can simulate gray-zone effects.

2.3 Toolkits and Programming Languages

Besides the toolkit used for NEMAN Interface 0.5, we have studied GTK+, Swing and Qt, to find a suitable toolkit for the new GUI. We then compare Qt with the other toolkits, to show the reason for choosing Qt. The criteria for choosing a toolkit is based on the statements in Table 2.2.

Table 2.2: Toolkit Criteria.

1	Compiled Language
2	Platform Independent
3	Object-Oriented
4	Good Documentation
5	Licensing
6	Personal choice

- **Compiled Language** In order to avoid the slowness of interpreted language; we need to program the new GUI with a fast compiled language, such as C++.
- **Platform Independent** Cross-platform software is popular, and to include users from different platforms to use our GUI, we need to look for a toolkit that supports cross-platform software. Since the topology manager of NEMAN is running on a Linux machine, and the GUI must run on a different computer, there are no problems if the GUI is run from a Windows or Mac based computer.
- **Object-Oriented** this concept allows reuse of code, and maintainability and abstract design. We are interested in such approach that allows

easy understanding and structuring of the code.

- **Good Documentation** the toolkit we are looking for must have good quality documentation, with this we mean, documentation of all libraries the toolkit provides.
- **Licensing** this is an important criteria, and we are looking for GPL licensed toolkit.
- **Personal choice** matters when considering the time limit of this thesis, toolkits that fulfill the other criteria and have equal properties, can be chosen based on personal experience. However we are clearly emphasizing compiled language, platform independent, object oriented, and licensing before personal choice.

2.3.1 Qt

Qt is a fully object-oriented and cross-platform application development toolkit created by the Norwegian company Trolltech. Qt includes C++ class library and tools. Qt toolkit utilizes the high performance which the C++ programming language offers. This is a necessity when one of the main goals of this thesis is to develop a GUI with better performance than NEMAN Interface 0.5. Qt code, which is cross-platform, only needs to be recompiled once in other platforms in order to work.

Qt provides single-source portability across Microsoft Windows, Mac OS X, Linux, all major commercial Unix variants, and embedded Linux. It is fully object-oriented, extensible, and allows true component programming.
In contrary to GTK+, we have had previous experience with Qt which is an advantage that cuts down the time and effort spent on searching for a suitable toolkit.

One of the advantageous sides of Qt, is the online documentation website which provides examples and documentation for each class and its properties in the Qt library. We are already familiar with Qt's API section in Python, which is basically the same API only read in the context of the programming language used. Another good reason for choosing Qt was because of the newly released feature of QGraphicsView framework in Qt 4.2.1 (which is used in this thesis). This framework is an enhanced replacement for the QCanvas[3] module which was previously provided for Qt 3, with a refined set of features.

The framework enables responsive handling of large numbers of canvas items through the use of space-partitioning scheme[4]. The QGraphicsView module's performance is utilized when handling large numbers of static objects with the binary space-partitioning Tree BSP indexing. With BSP Tree moving or adding objects on the graphic scene are done in logarithmic time. Thus for moving objects we gain performance speed by not indexing the BSP this gives us constant time on operations like moving or adding an object to the graphical scene.

2.3.2 Tcl/Tk

Tcl (Tool Command Language) is a scripting (Interpreted) language like Perl but extensible and with cleaner syntax and ease of use. Tk is an extension developed by the creator of Tcl and is used for creating scripts that interact with users through windows. Tcl can be used as a Unix shell and Tk was originally developed to create windows for the X windows enviroment [24].

As we have already described NEMAN Interface 0.5 is written in Tcl/Tk, and due to the runtime slowness of interpreted languages, as matter of course we do not want to implement the new interface in a similar language. However, we must understand this language in order to determine the lack in speed performance. Tcl/Tk is also studied because several popular network simulators

[3]QCanvas is a Qt class that provide 2D area that can contain graphical items

[4]space-partitioning scheme is a mathematical process of dividing a space into none overlapping regions, binary space-partitioning BSP is one of the most common forms of space-partitioning

and emulators interfaces are implemented with Tcl/Tk, those are as earlier mentioned NS-2's Nam, MobiEmu/NEMAN Interface 0.5 and OMNet++'s Tkenv.

Computer languages that are interpreted, executes, or performs instructions by a program called an interpreter. These languages are often faster to develop with programs, compared with their counterparts which are compiled languages. Due to their flexibility to run command by command, also in shell mode (most interpreted languages provide the ability to run in shell mode, where the programmer types command and receives result instantly).

However when it comes to efficiency as mentioned interpreted languages are slower in runtime than compiled languages, because the interpreter must analyze each statement in the program before it performs the desired action, whereas the compiled code just performs the action, in fact compiled languages are in cases 10 times faster than interpreted languages.

Tcl is mostly used for rapidly prototyping, scripted applications and GUI, thus Tcl is not meant for software applications that require heavy computations and fast responsiveness, such as animation based programs. Usability studies in [8] show that users in general do not care about whether a long running task takes several minutes, but they do care when the program does not show an immediate reaction when e.g. a button is clicked. Further, the studies shows that the limit of what a user accepts before the program is considered to be unresponsive can be 0.7 seconds. As we understand Tcl is highly dynamic and flexible language that enables quickly development of prototypes and programs, that do not require fast processing of huge amount of data, such includes graphical objects or animation.

2.3.3 Java and The Swing Toolkit

Java is cross-platform compiled programming language, which at the same time is semi-interpreted by the Java Virtual Machine (JVM), JVM is an emulator where the compiled code is byte code. This byte code is not directly executed by the CPU, but by JVM. JVM is again executed by the CPU, thus a java byte code program does not take place in hardware, as with real compiled languages such as C/C++, but in slower software emulation.

Runtime-efficiency

Prechelt [18] carried out an empirical comparison of 7 programming languages. One of Prechelts interesting remark is that the runtime of a Java program runs at least 1.22 as long as a C/C++ program, and that the average runtime of Java programs is even longer.

In memory management Java and for instance C++ have different approaches, in C++ memory management must be done explicitly by the programmer, in order to avoid memory leak. Java has a different approach since it is provided with a garbage collection, which automatically deallocates memory that is not needed anymore by the program, this is very convenient but the trade offs are greater memory consumption and slower runtime speed [8].

Swing

Java comes with a graphical user interface tool called Swing, Swing partially depends on AWT (Abstract Windowing Toolkit) when it comes to handling events and executing primitive drawings operations, AWT is the original GUI toolkit for Java. Most of the Swing toolkit is implemented in Java itself, due to the runtime problems mentioned about Java, Swing programs are slow when performing computation and to draw and handle user interfaces. The Swing toolkit supports development of sophisticated user interfaces, but code is often several hundreds of lines more to create small programs, compared with other toolkits such as Qt. The reason is because Swing enforces the use of Mode-View-Controller (MVC) architecture, and Qt also supports this architecture but the user is not enforced this approach.

2.3.4 GTK+

GTK+ is a widely popular widget toolkit for the X window system for creating graphical user interfaces. GTK+ was originally developed for the GNU Image Manipulation Program (GIMP) **??**.

GTK+ which is implemented in C is not object-oriented in the way we are familiar with the object-oriented concept in e.g. C++ or Java. Although it provides GLib Object systems which is an implementation of object-oriented framework for C. GTK only provides a C API, but if C++ is preferred one must use GNOME platform bindings (gtkmm) bindings which is an object-oriented add-on to support C++.
GTK+ also offer a GUI layout design tool called Glade Interface Designer, Glade produces XML files. To our knowledge GTK/GTK+ is a powerful (in computation, since the compiled C programming language is used) and widely used toolkit. However, we have chosen not to use GTK+ for reasons which are both technical as stated in our criteria Table 2.2 and of personal choice. The cross-platform approach is not as convenient as with e.g. Qt where you only need to recompile the source code in other platforms in order to work there. Personal choice is regarding the authors experience with GTK+, which is limited.

2.3.5 Toolkit Marks

Now that we have shown the different selected toolkits, we give them marks that show their ability according to our criteria tabell 2.2

Based on our marks of the toolkits in Table 2.3 we can see that both Qt and Swing, meet with our criteria. However due to the runtime inefficiency problems in Java programs, Qt which is C++ based and highly efficient in runtime, we have selected Qt as our chosen toolkit to implement the NEMAN Interface version 1.0.

Table 2.3: Marking toolkits

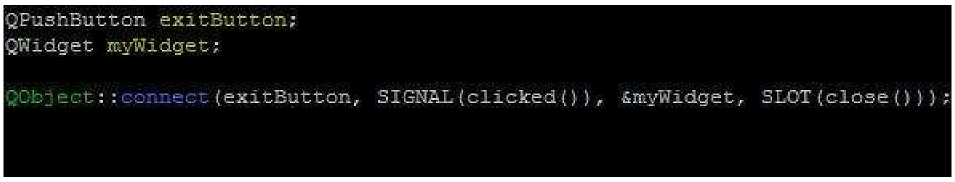
	GTK+	**Tcl/Tk**	**Qt**	**Swing**
Compiled Language	X		X	X
Platform Independent			X	X
Object-Oriented			X	X
Good Documentation	X		X	X
GPL Licensing	X	X	X	X
Personal Experience		X	X	X

2.4 Introducing Qt

Since Qt is the chosen toolkit to develop the new interface with, a short introduction of Qt along with used classes and the Qt designer is given.

2.4.1 Signals and Slots

Signal and slot mechanism is a central feature of Qt which is used for communication between objects, e.g. in GUI programming. If a button widget exitButton wants to interact with another widget myWidget, the exitButton emits a signal to myWidget when a particular event occurs; myWidget provides a slot which is a function that is called in response to a particular signal. Qt widgets have many predefined slots, but it is common to add slots so that you can handle the signals you are interested in. All classes that inherit from QObject or one of its subclasses e.g. QWidget can contain signals and slots. The signal slot mechanism is explained with code in Figure 2.6.



```
QPushButton exitButton;
QWidget myWidget;

QObject::connect(exitButton, SIGNAL(clicked()), &myWidget, SLOT(close()));
```

Figure 2.6: Example code on signal and slot mechanism

In Figure 2.6 when the user clicks the exitButton the object emits signal clicked and myWidget provides the slot close() for that particular signal. As

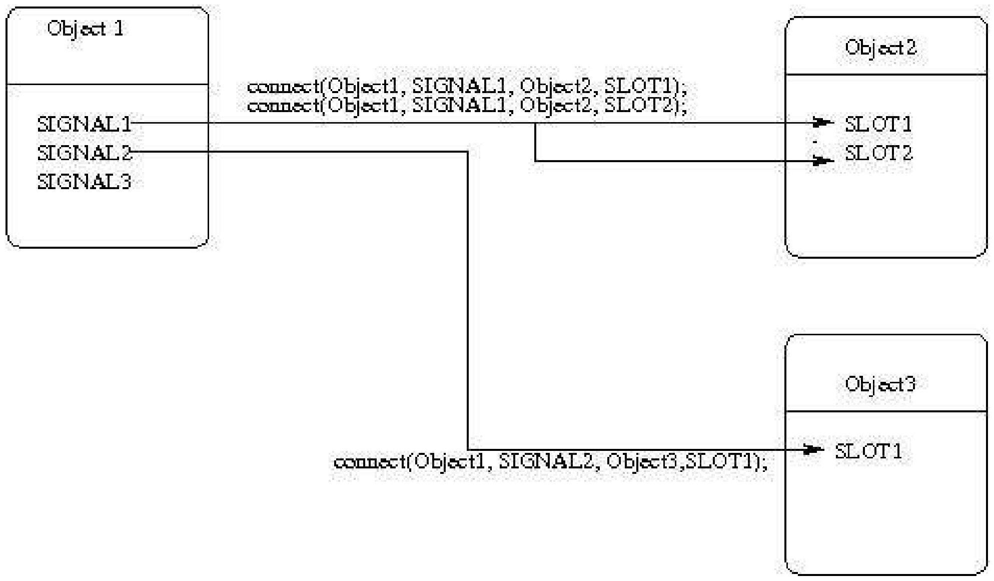

Object 1
SIGNAL1
SIGNAL2
SIGNAL3
connect(Object1, SIGNAL1, Object2, SLOT1);
connect(Object1, SIGNAL1, Object2, SLOT2);
Object2
SLOT1
SLOT2
Object3
SLOT1
connect(Object1, SIGNAL2, Object3,SLOT1);

Figure 2.7: Signal and slot connections

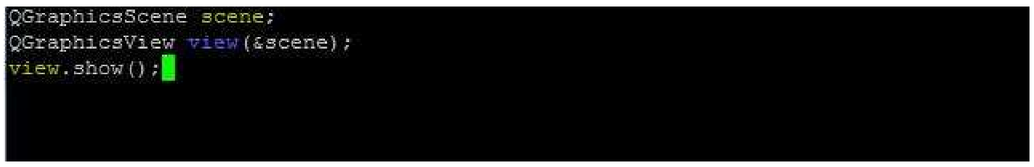
we can see in Figure 2.7 objects can have one signal connected to multiple slots in as many objects as wanted.

2.4.2 Qt Designer

Qt designer is a helpful and essential GUI layout and forms builder tool which is part of the Qt toolkit. Designer makes it easy to visualize user interfaces; the work with the design of the GUI components and the code is separated. Designer creates XML user interface files, and later the Qt user interface compiler (uic) generates C++ source code header files. Additionally user interface compiler for PyQt (Python based Qt) which is among the programming languages that have bindings for Qt, can also process. Each widget added to the form would be handled as an object with its own properties. To access and alter the form that e.g. is a main window or dialog window, one must subclass either QMainWindow or QDialog. Sub classing is used to either extend the functionality of the form by creating your own class, based upon a form created with Qt Designer.

2.4.3 GraphicsView and GraphicsScene

The QGraphicsView framework has been introduced into Qt 4.2 and was released shortly after the work on this thesis began. This framework is the successor of the previous QCanvas modules provided in Qt3, it includes more improvements and refined set of features. QGraphicsView class provides widgets for displaying contents of a QGraphicsScene, which is a class to visualize a scene. This scene can then contain all types of Canvas objects; in Figure 2.8 we visualize our scene by constructing a QGraphicsView object and pass the address of the scene to QGraphicsView constructor.



```
QGraphicsScene scene;
QGraphicsView view(&scene);
view.show();
```

Figure 2.8: Example on creating a graphic scene

Then the scene provides a surface to manage 2D graphical items; the QGraphicsItem class provides foundation for creating our customized item objects. These items include shapes such as line, ellipse, text or custom pixmap (picture).

With these classes we are able to create a visual scene where we can load graphical 2D items which represent the nodes in a scenario. We calculate the coordinates and movements in each node by reading the X and Y coordinates and while the nodes are moving in the scene we update their position by finding the QGraphicsItem::pos.x() and QGraphicsItem::pos.y() positions. The QGraphicsView framework has a huge effect on both the visual look and feel and visual performance, the framework has extensive documentation in [21].

2.5 Summary

In this chapter, we have outlined the research area and the motivation behind this thesis. We have further studied NEMAN Interface 0.5, and found interesting remarks on the performance issue in the MobiEmu project [23]. In our search for a good GUI candidate we also closely inspected popular simulators and emulators such as NS-2 and OMNet++, t o see the possibility of porting such GUI into NEMAN.

We also gained more insight into the topology manager of NEMAN (TopoMan) and the interface it provides for socket communication with any user interface. We have also studied the structure of different scenario files such as the NS-2 formatted and the mkdist formatted, in order to develop an interface which can handle both file formats. In our comparison studie between the toolkits we found Qt as the most suitable toolkit which fulfills the criteria.

Chapter 3

Design

A Graphical User Interface (GUI) is simply the means by which an application communicates with the user, and the user with the application through manipulation of graphical objects. GUI provides humans to interact with the computer, through the use of windows, icons and menus. GUIs stand in sharp contrast to command line interfaces, which only use text and are accessed solely by a keyboard [2]. GUI came into existence because the first interactive user interfaces to computers were not graphical; they were text-and-keyboard oriented [1]. These interfaces also exist today as an alternative to GUI and consist of commands that are memorized and computer responses that are brief.

The advantage of a GUI is to make the usage of software or computer operations more intuitive, and easy to learn even for novice users, because of the graphical representation of the commands which are to be executed.

A GUI gives the possibility to change window size, colors of the text and font size. In additional a GUI present commands, options or data to the user on the appropriate application display and to organize information in a meaningful way to make it user friendly [17]. This opportunity has contributed to allow people in different physical conditions to use computers. Command line based interface requires more effort from average users, but advanced users such as system administrators or Unix users think it is more convenient and powerful.

In this Chapter, we describe the design of NEMAN Interface version 1.0, by utilizing object-oriented analysis and design features. Object-oriented approach gives us the ability to reuse code, and thus makes the maintainability if the program much easier. Furthermore, we perform analyses on a questionnaire about the user experience on NEMAN Interface 0.5 in Section 3.1 and 3.2. In Section 3.2.1 we design NEMAN Interface 1.0 with the use case diagrams and explain in details about the functionalities with state chart diagrams.

3.1 Requirement Analyses

Requirements specification phase in this thesis started early with modeling the problem domain, the two most important specification techniques in object-oriented analysis are use case diagrams and class diagrams [15]. A typical specification document also describes other requirements such as performance, look and feel, usability, maintainability and security, which are important task to take into consideration. Since NEMAN users pose the requirements for the NEMAN Interface 1.0, a questionnaire about requirements is carried out. This questionnaire gives us valuable information about how to proceed in the development process. By interpreting the questionnaire we are able to design a GUI which is capable of meeting the needs of the users; however it is not sufficient enough to be called accurate statistical survey due to the number of participants who are limited to few PhD's, and Master students.

We selected the participants on criteria's based on their experience on NEMAN usage. Some of the participants had few weeks of experience, while others use NEMAN on daily bases, with this in mind, we should be aware that some of the participants might not know about the problems we are investigating at all.

The questions were given to analyze and gather information about the users experience and their view about the functionalities in the NEMAN Interface 0.5, regarding where in the emulation steps they experienced undesired disability in visual performance, and additionally features they want to be added to the NEMAN Interface 1.0. Some of the participants did not answer questions they did not have sufficient knowledge for, we can see that participants answering to the different questions are manifold.

3.2 Empirical Evaluation of NEMAN interface Version 0.5

- **Question 1: How is your experience with the usage of the current GUI?** Possible answers:
  1. -Intuitive
  2. -Good
  3. -Difficult
  4. -Very Difficult
  5. -Dont' know

In Figure 3.1 which shows results from question 1: 3 people reported that the GUI is intuitive to use, while the other 3 reported that it is good to use and one person did not know. This shows that most participants agreed that NEMAN Interface 0.5 is easy to use, due to the minimalistic way the GUI is designed. We are also interested to adopt the same minimalistic design approach for the NEMAN Interface 1.0.

- **Question 2: How fast is the learning threshold of the GUI?** Possible answers:
  1. -Very fast
  2. -Fast

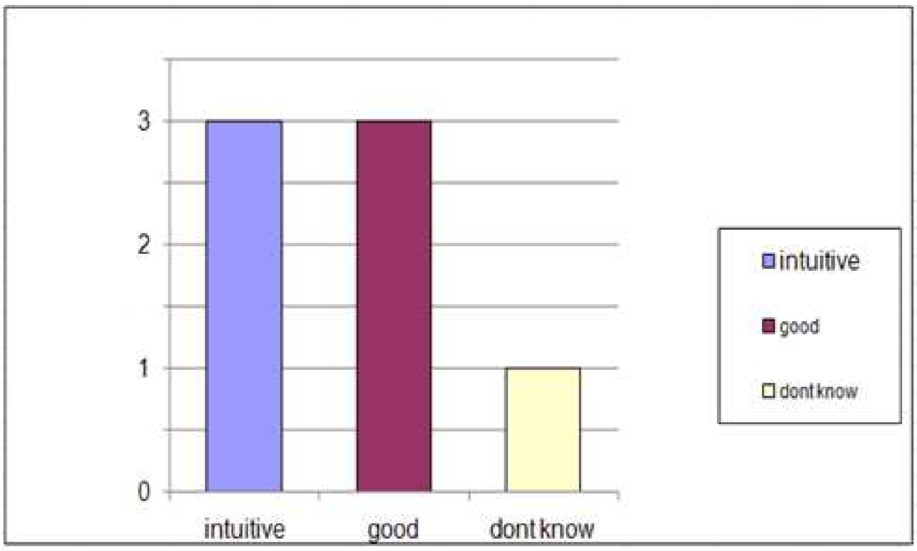


Figure 3.1: Question 1, on user experience

3. -Slow
4. -Very slow
5. -Dont' know

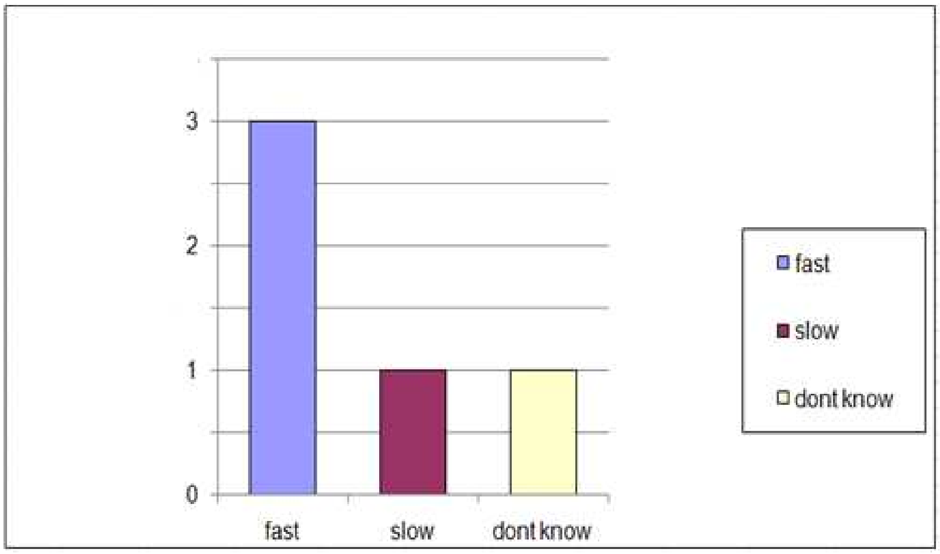


Figure 3.2: Question 2, on learning threshold

Figure 3.2 shows results from question 2: 3 people reported that the learning threshold in NEMAN Interface 0.5 is fast due to the few options in the GUI, hence the user can quickly adapt to it. 1 person reported that the learning threshold is slow, explaining that the GUI is difficult to understand, while one person did not know.

- **Question 3: Are you satisfied with the visual performance of the GUI?** Possible answers:
    1. -Yes
    2. -No
    3. -Dont' know

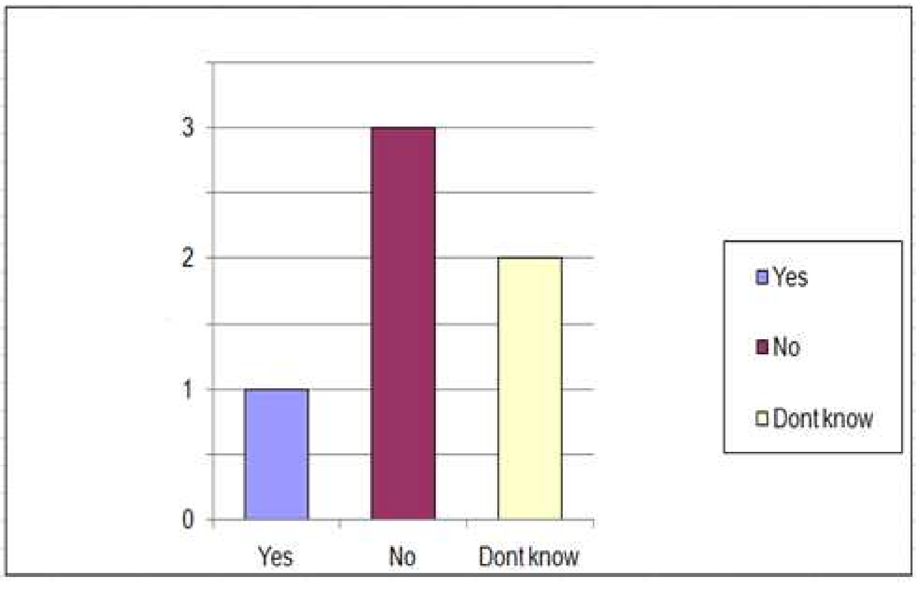


Figure 3.3: Question 3, on visual performance

Figure 3.3 which shows results from question 3: Here 3 people reported that they are not satisfied with the visual performance. The 2 people who answered "don't know" are master students with few weeks of experience with NEMAN. This clearly shows that they have not experienced the performance lag due to their emulation setup which mainly consisted of few nodes. One person reported that he or she is satisfied with the visual performance.

- **Question 4: Are you able to do your work efficient with the current GUI?** Possible answers:
  1. -Yes
  2. -No
  3. -Dont' know

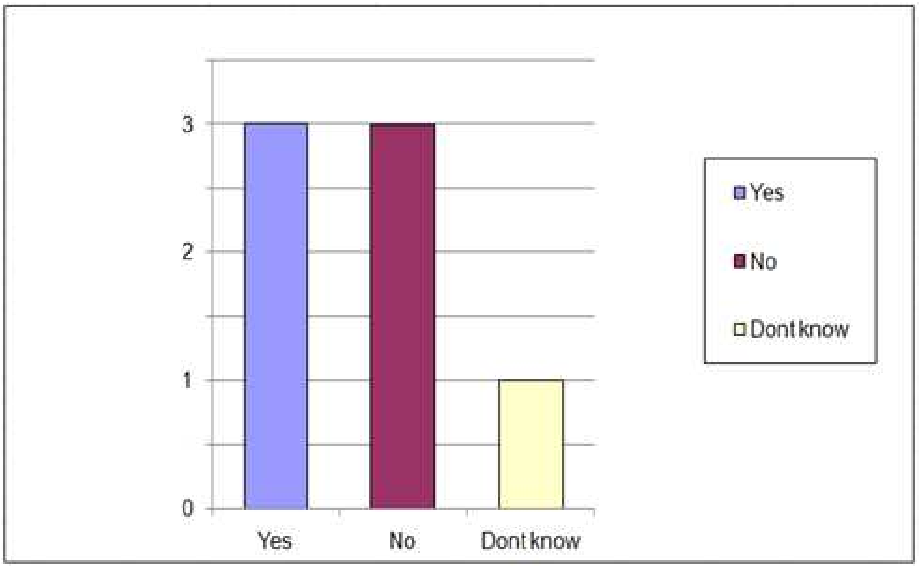


Figure 3.4: Question 4, on work experience

Figure 3.4 shows the final results from question 4: 3 people reported that they are able to do their work efficiently with NEMAN Interface 0.5, on the other hand 3 people disagreed and one person had no opinion. The reason for this result can be caused by the participants only working with few nodes in their scenario files, the udpsend program is as mentioned used to communicate link changes to the TopoMan, when big scenario files are loaded, with e.g. 100 nodes, udpsend's CPU consumption increases. The resources are consumed because the kernel has to load the executable of udpsend, loading the program from disk also takes time and loading all shared libraries and the program itself into memory also takes time. When this is done for 100 nodes that are frequently changing link, performance lag is .

In conclusion to the answers reported back on the questions. NEMAN Interface 0.5 is intuitive and good to use, the learning threshold is quite fast, but there are visual performance lag reported mainly by those who emulate with big scenario files consisting of many nodes. Thus they are not able to do their work efficiently, on the other hand those who emulate with small scenario files, which consists of nodes ranging from 3-50 are capable of doing their work efficiently without visual performance lag.

In addition to the questions represented in the histograms, the participants were also given four none multiple choice questions. Answers to those questions where considered when removing functionalities from the NEMAN Interface 0.5 and adding new ones.

- Which functionalities would you remove from the current GUI?
- Which functionalities would you add to a NEMAN Interface 1.0?
- Which functionalities in the current GUI do you use mostly?
- Which functionalities in the current GUI do you use seldom?

3.3 Object-oriented Design Approach

The use cases show the intended behavior of NEMAN interface version 1.0, and are derived from analysis of user's feedback, and also inspired by the behavior of the NEMAN Interface 0.5. We are trying to keep as much as possible of the functionalities of NEMAN Interface 0.5, to simplify the user's transition to NEMAN Interface 1.0, keeping the look and feel of NEMAN Interface 0.5.

Tables 3.1 and 3.2 explain in more detail the dependencies between the use case diagram shown in Figure 3.5 for the graphical interface, and the command line interface in Figure 3.7.

State charts diagrams in general demonstrate what action objects perform when they receive an event. As we can see the state chart diagrams in Figure

3.6 and Figure 3.8 describing the determination of how objects react to events in parts of the main events in the application.

Table 3.1: Requirement assignment to actors and use cases for the GUI

Requirement	**Actor**	**Use case**
The user runs the GUI of NEMAN	NEMAN user	Display configure GUI
The user choose to load a scenario file	NEMAN user	Load scenario file
The user may choose to change starting node	NEMAN user	Change start node
The user may choose to turn on/off routing daemons	NEMAN user	Turn on or off routing
The user may choose to set physical layer simulation	NEMAN user	set physical layer simulation
The user may choose to preview emulation	NEMAN user	Preview emulation
The user may choose to loop the emulation	NEMAN user	Loop emulation
The user starts the emulation	NEMAN user	Start emulation
The user may choose to exit the GUI	NEMAN user	Exit

Table 3.1 shows the requirement assignment to actors for the GUI, and their corresponding use cases. The actors are NEMAN users. As we can see, a NEMAN user is interacting with the GUI, and has the opportunity to do several configuration steps.

Use cases shows the functional requirements of the system we are developing in a more intuitive method. The requirement assignment from Table 3.1 are displayed visually as use cases in Figure 3.5. As we can see the user is in the middle and the arrows correspond to actions performed.

The state chart diagram in Figure 3.6 describes the functions that are called when a certain action is performed. The state chart shows the steps in the emulation, from when a scenario file is loaded and the events that can happen during that time, for instance when the emulation is stopped, the scenario file is reloaded and the nodes go back to their initial starting point on the scene. When the emulation finishes without user intervention, it enters finished mode. At this point the user can restart the emulation, load a new scenario file or exit.

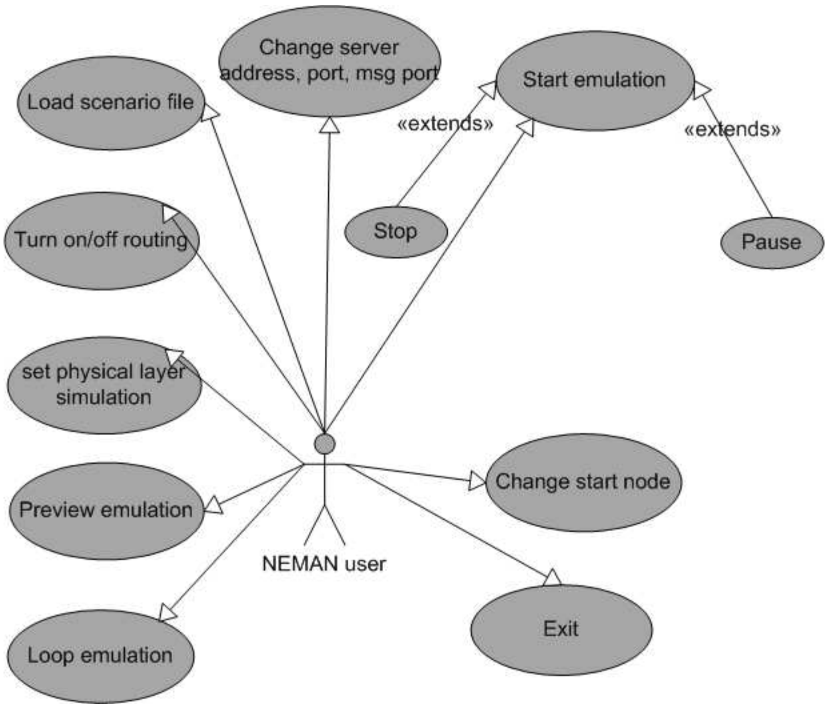


Figure 3.5: Use case for Graphical interface

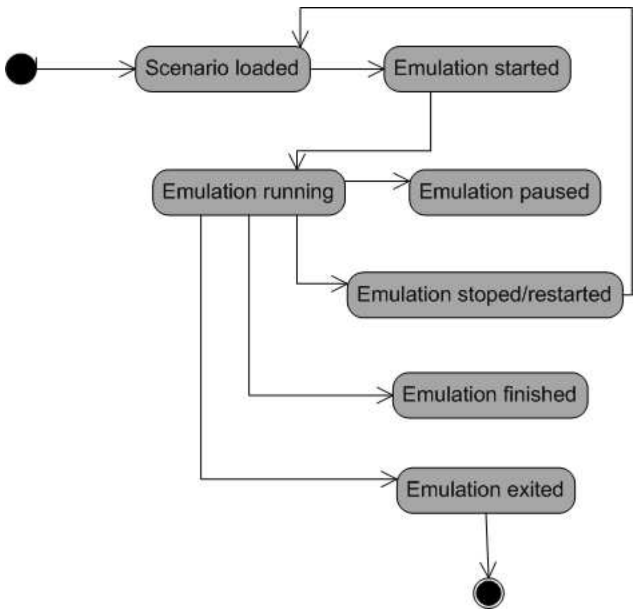


Figure 3.6: State chart diagram for graphical interface

Table 3.2: Assignment of requirements to actors and use cases for command line interface

Requirement	**Actor**	**Use case**
The user sends no command parameters	NEMAN user	Default
Default parameters is set and emulation started	NEMAN user	Start emulation
The user choose to set command line parameters	NEMAN user	Set command line parameter
The user may choose to turn on/off routing daemons	NEMAN user	Turn on or off routing
The user choose to configure server	NEMAN user	change server addr, port and message port
The user may choose to set physical layer simulation	NEMAN user	set physical layer simulation
The user may choose to preview emulation	NEMAN user	Preview emulation
The user may choose to loop the emulation	NEMAN user	Loop emulation
The user may choose to repeat the emulation	NEMAN user	Repeat emulation
The user may choose to cut of time	NEMAN user	Cut of time
The emulation starts	NEMAN user	Start emulation
The user may choose to exit	NEMAN user	Exit with Ctrl-c signal

Table 3.2 shows us the use cases that are for the command line interface when actor starts emulation. Later procedures automatically react to certain event in the emulation, e.g. the user which is called NEMAN user, might start command line based emulation without giving any arguments to the interface. This entails into the start of an emulation with default parameters such as start node is by default set to 1, speed is set to 1, loop is set to 0, repeat is set to 0 and so on. For the topology manager values such as the server address, server port, message port and routing port are read from server.ini file.

Figure 3.7 is the Table 3.2 described a bit more with use case diagram, showing the options the user have, and the steps an emulation in the command line interface might be performed.

As we explained in Table 3.2 this state chart diagram shows, in a more

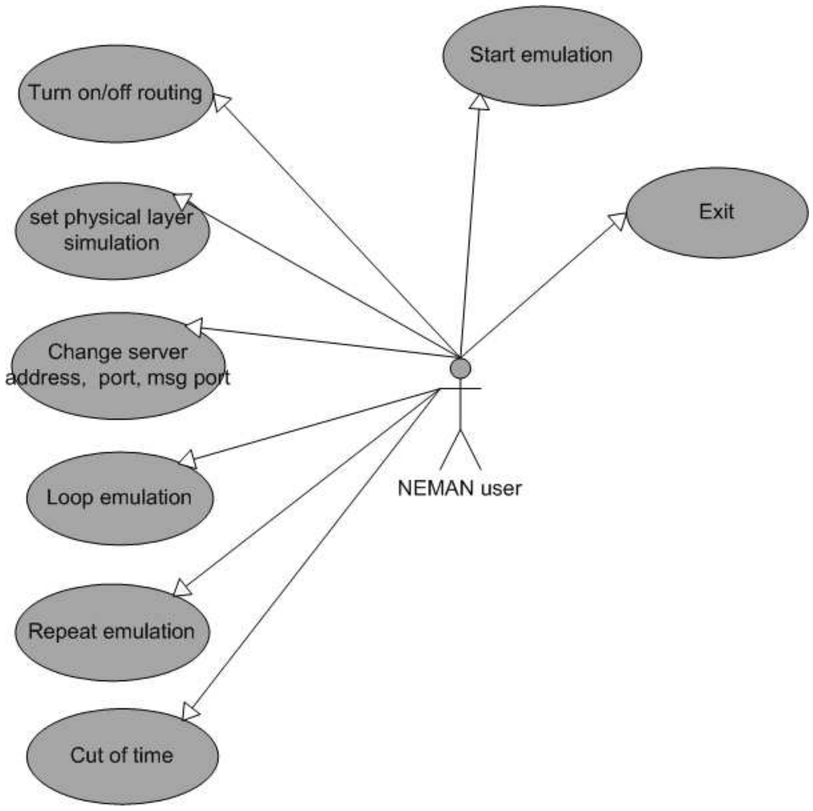


Figure 3.7: Use case for command line interface

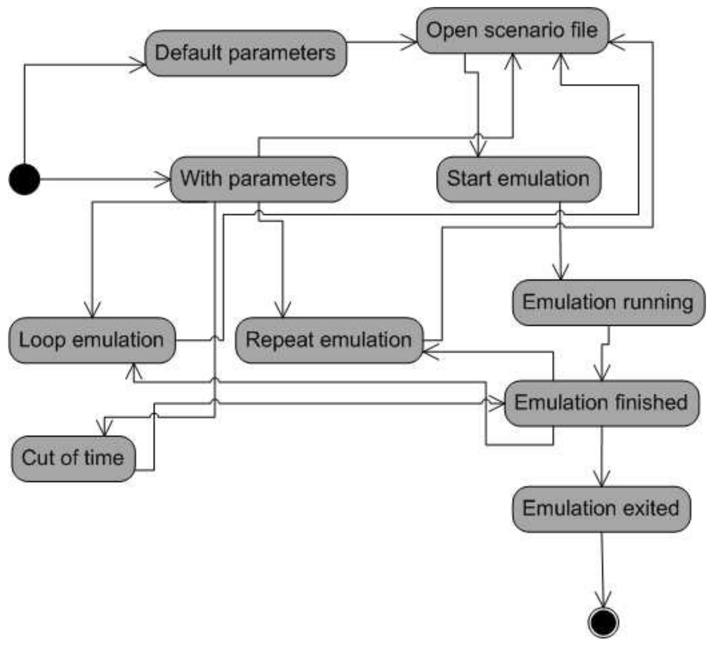


Figure 3.8: State chart diagram for command line interface

detailed way the functions that are called when emulation is started without arguments. As we can see if the interface is run with no arguments after scenario file is given, default values is given and the emulation runs normally with no change in start node, speed or etc. However, if we give arguments to the interface and those arguments are e.g. loop the emulation. Then loop emulation calls open scenario which again calls start emulation, when emulation finishes, finish emulation is called and loop is again called, this continues until the user interferes. The same also happens for the repeat functionality, except that the number of times the emulation should repeat is given, when it has repeat the emulation given number of times, finish emulation calls exit.

3.4 Summary

In this chapter we inquired the requirement analysis and carried out questionnaire to collect information from the users of NEMAN. Those who participated in the questionnaire had different levels of experience with the NEMAN interface version 0.5. That is why we have different answers on questions regarding the visual performance issue. The participants came up with many new ideas for new features; we have thus carefully studied those features which can be implemented in NEMAN Interface 1.0. The features we did not had sufficient time to develop, are described in the Section in Chapter 6.2.

Furthermore, based on the questionnaire an object-oriented design, which involves use case diagrams and state chart diagrams. The state chart diagrams helped us to describe actions that are performed when an object receives an event. We have separated the design of the GUI part of the interface and the command line part, in spite of both interfaces being one and the same software application. The use cases determine the usage of the GUI (version 1.0) which is to be implemented, even though much of the functionality is inspired by NEMAN Interface 0.5. However we have taken this approach so that the users can have an easy transition from NEMAN Interface 0.5 to 1.0.

Chapter 4

Implementation

In this chapter, we describe the implementation of the new Graphical and command line interface for NEMAN. We have chosen to implement the GUI in the C++ programming language, previously described in Section 2.3, because of its flexibility to interact with low level interfaces such as sockets, which the Tcl/Tk programming language of NEMAN Interface 0.5 is not directly capable of. Instead the applications udpsend.c and udpreceive.c are used.

This implementation is done to guarantee object-oriented development, allowing inheritance of the existing classes which simplifies the developers work if new components need to be added to the interface in the future.

We to explain the implementation of the graphical part of NEMAN Interface 1.0 in Section 4.1 and in Section 4.2 we describe the command line implementation. Examples of both the interfaces in action are also described.

4.1 The Graphical Interface

In this section, we explain the process of how the Graphical interface is implemented, with respect to our design outlined in Chapter 3. When considering the design of the GUI we want to implement an intuitive and minimalistic

GUI. We are able to do this by using both self explaining icons and popup balloon messages that describe the buttons. Also the icons improve the look and feel of the GUI, compared with NEMAN Interface 0.5, which does not use icons, but instead plain white buttons with black text. We provide with both the graphical and command line interface a progress bar which is used to show the progress of loading a scenario file and the progress of the total emulation time. NEMAN Interface 0.5 only provides progress bar the total emulation time. Before we begin to explain in detail the implementation of the graphical part, the reader should be aware of that the GUI and the command line in NEMAN Interface 1.0 are one and the same application, but run with different arguments.

4.1.1 Parsing Scenario File's

Like the old GUI a user starts emulation by loading a scenario file to be displayed. The scenario file is parsed with QRegExp Qt's which is regular expression class and the retrieved data is stored into appropriate data structures which are QVectors and QMap which are both similar to standard C++ std::vector and std::map. The scenario file contains information about each node in the scene, their X and Y positions, speed and link status. When emulation is started a QTimer timer object is also started, this timer can run fast or slow according to the speed given. The timer invokes what is to be done for each timestamp in the scenario.

We will now take a look into the steps the scenario file is read and how the data is collected beginning from the top of a scenario file and down.

As earlier mentioned there are two types of scenario file formats: the NS-2 based and the mkdist filtered files. The difference is found at the top of the NS-2 scenario files, and at the bottom on mkdist scenario files, Figure 4.2 shows the NS-2 signature information.

while Figure 4.3 shows signature information on bottom of mkdist based

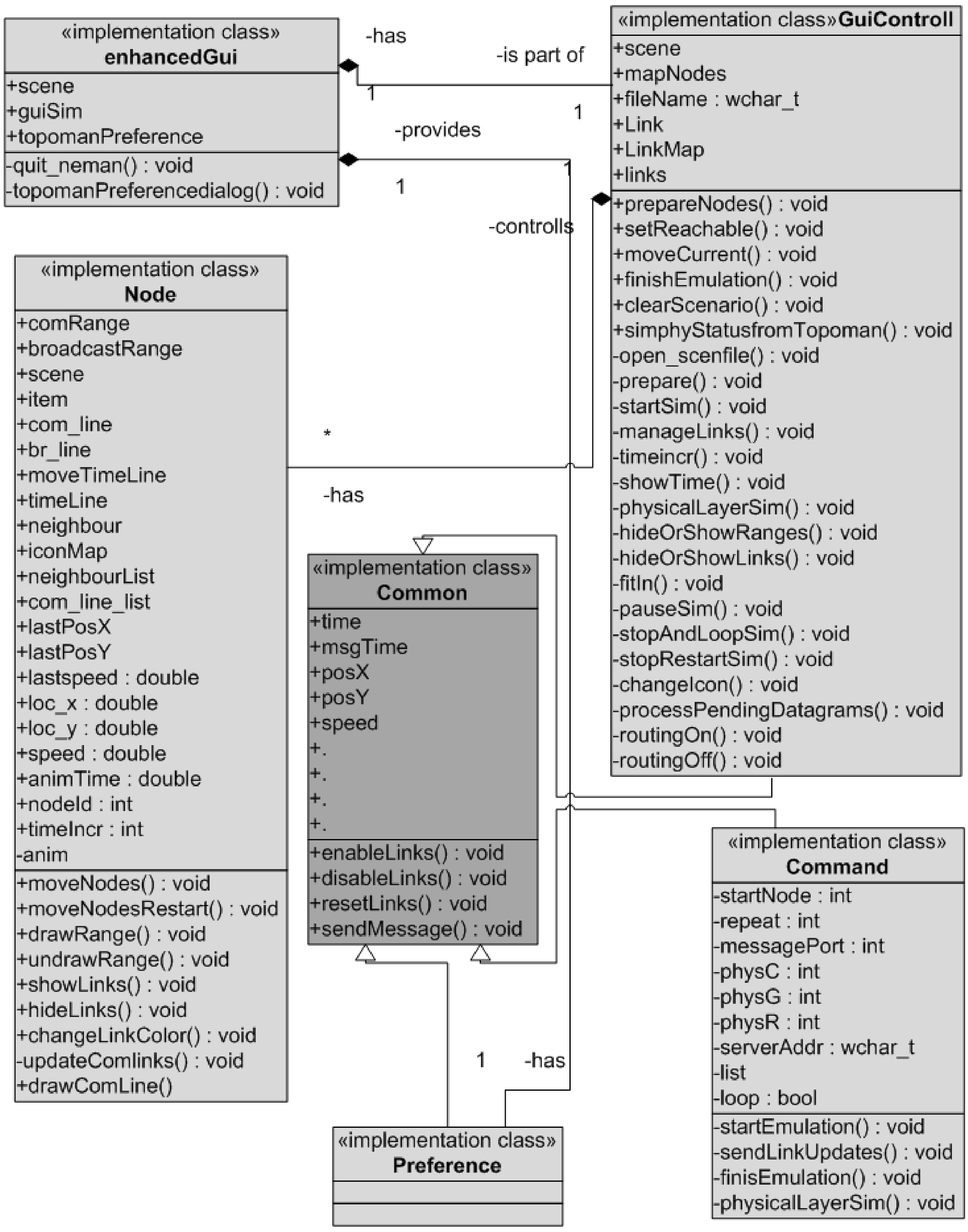
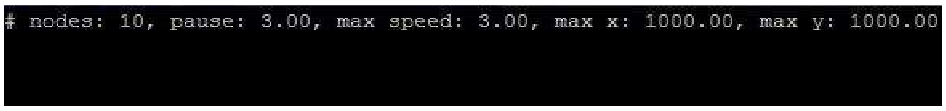


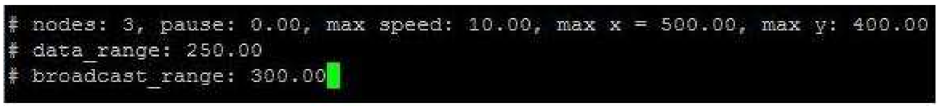
Figure 4.1: class diagram 1



```
# nodes: 10, pause: 3.00, max speed: 3.00, max x: 1000.00, max y: 1000.00
```

Figure 4.2: Information signature on NS-2 based scenario file

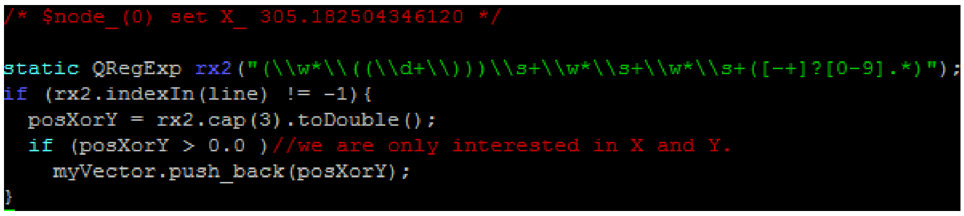
scenario file.



```
# nodes: 3, pause: 0.00, max speed: 10.00, max x = 500.00, max y: 400.00
# data_range: 250.00
# broadcast_range: 300.00
```

Figure 4.3: Information signature on Mkdist based scenario file informaton

Mkdist based scenario file introduce data_range and broadcast_range that are used to indicate the range in meter of the communication and broadcast radio range. In the GUI two circles are drawn, with the node in the center. Since NS-2 scenario files do not include this information, we instead use default circles with 250.00m for communication range and 300.00m for broadcast range. These values are used because communication range of 250m is widely used transmission range for wireless network simulations [22]. This also depends on the program used to generate the scenario file; our program setdest uses 250m as the range of wireless transmitters. Therefore a distance value which is 1 is within 250m range, and a distance value equal to 2 has 300m as distance value.

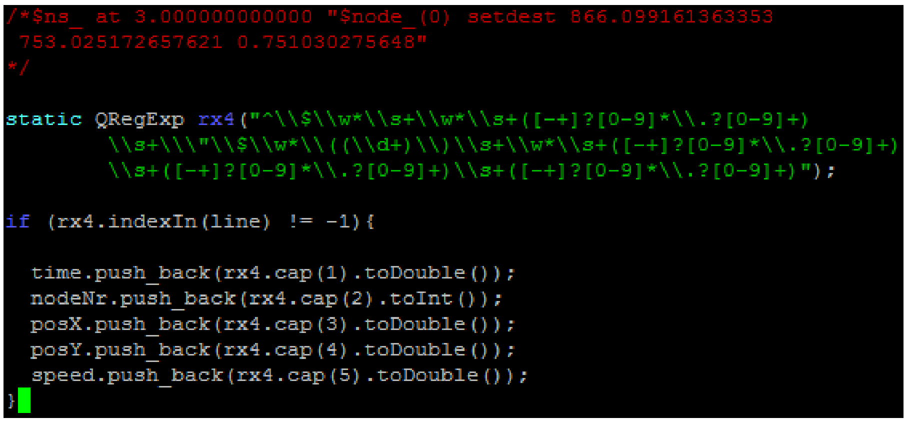


```
/* $node_(0) set X_ 305.182504346120 */

static QRegExp rx2("(\\w*\\((\\d+\\)))\\s+\\w*\\s+\\w*\\s+([-+]?[0-9].*)");
if (rx2.indexIn(line) != -1){
  posXorY = rx2.cap(3).toDouble();
  if (posXorY > 0.0 )//we are only interested in X and Y.
    myVector.push_back(posXorY);
}
```

Figure 4.4: Retrieving the nodes X and Y coordinate values

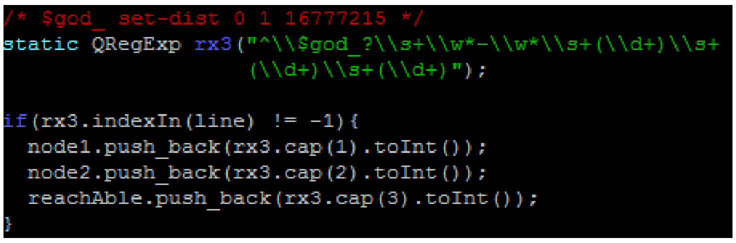
Further in the scenario file we continue to process the information about each node. The code in Figure 4.4 shows how we retrieve the X and Y values, and store them into myVector which is of QVector data structure, myVector is used later when we position the nodes on the graphical scene. The comment in the code is used to show the line in the scenario file which is parsed with regular expression.

For now the X and Y values contain the position where the nodes are located in the 2D space of the scene. In the scenario we also need to retrieve the X and Y values for a moving node. This is shown in Figure 4.5 we first store the time value, in a QVector of real, then the node's unique ID the positions and the speed velocity are also stored in QVectors. The comment in the code can also be read like this: at time 3.0 node 1 is moving toward X = 866.09 and Y = 753.02 with speed 0.751.



```
/*$ns_ at 3.000000000000 "$node_(0) setdest 866.099161363353
 753.025172657621 0.751030275648"
*/

static QRegExp rx4("^\\$\\w*\\s+\\w*\\s+([-+]?[0-9]*\\.?[0-9]+)
        \\s+\\\"\\$\\w*\\((\\d+)\\)\\s+\\w*\\s+([-+]?[0-9]*\\.?[0-9]+)
        \\s+([-+]?[0-9]*\\.?[0-9]+)\\s+([-+]?[0-9]*\\.?[0-9]+)");

if (rx4.indexIn(line) != -1){

  time.push_back(rx4.cap(1).toDouble());
  nodeNr.push_back(rx4.cap(2).toInt());
  posX.push_back(rx4.cap(3).toDouble());
  posY.push_back(rx4.cap(4).toDouble());
  speed.push_back(rx4.cap(5).toDouble());
}
```

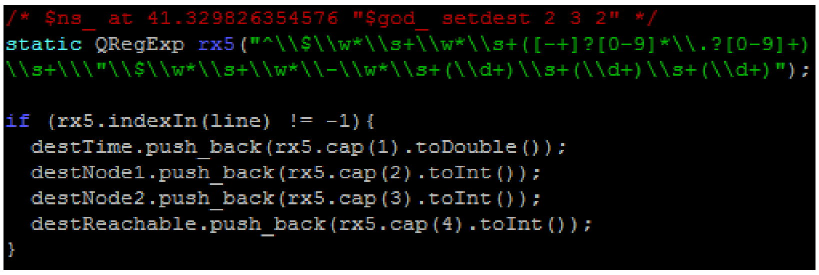
Figure 4.5: Time values, node ID's, coordinates and speed values are retrieved



```
/* $god_ set-dist 0 1 16777215 */
static QRegExp rx3("^\\$god_?\\s+\\w*-\\w*\\s+(\\d+)\\s+
                   (\\d+)\\s+(\\d+)");

if(rx3.indexIn(line) != -1){
  node1.push_back(rx3.cap(1).toInt());
  node2.push_back(rx3.cap(2).toInt());
  reachAble.push_back(rx3.cap(3).toInt());
}
```

Figure 4.6: Link range value between two nodes is retrieved.

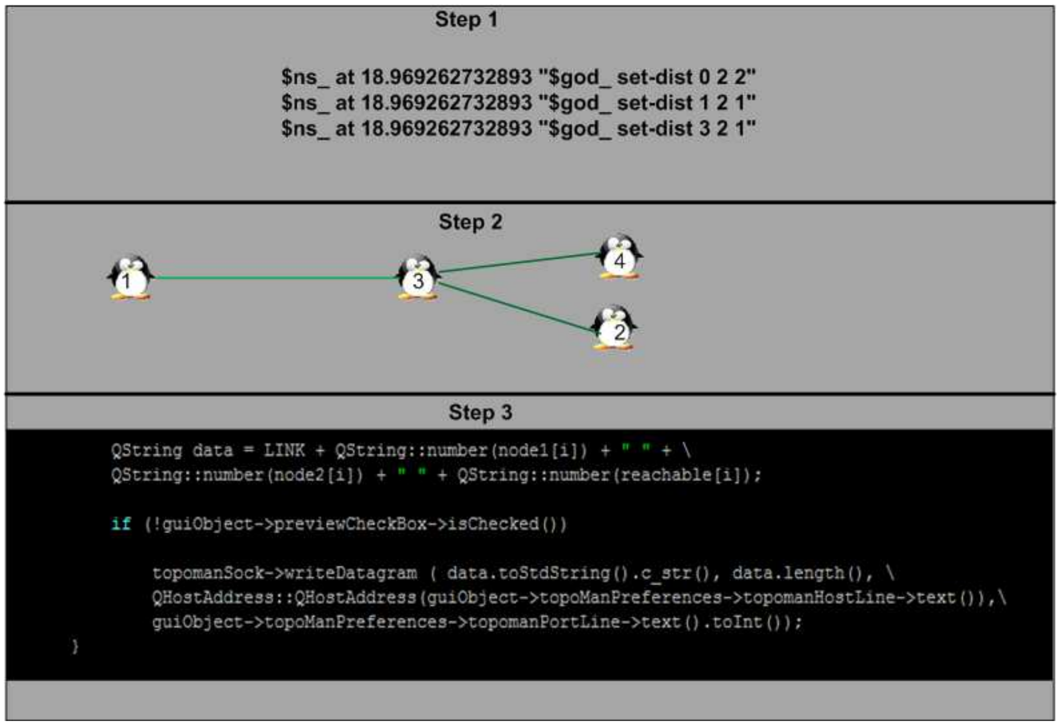
Figure 4.6 shows the link status for the nodes before they begin to move around. Here we collect the first and second node and the link status they share. When prepare emulation is either prepared or started these links are drawn.



```
/* $ns_ at 41.329826354576 "$god_ setdest 2 3 2" */
static QRegExp rx5("^\\$\\w*\\s+\\w*\\s+([-+]?[0-9]*\\.?[0-9]+)
\\s+\\\"\\$\\w*\\s+\\w*\\-\\w*\\s+(\\d+)\\s+(\\d+)\\s+(\\d+)");

if (rx5.indexIn(line) != -1){
  destTime.push_back(rx5.cap(1).toDouble());
  destNode1.push_back(rx5.cap(2).toInt());
  destNode2.push_back(rx5.cap(3).toInt());
  destReachable.push_back(rx5.cap(4).toInt());
}
```

Figure 4.7: Here we retrieve link changes

Figure 4.7 then shows the link status at a certain time in the emulation. In this example after 41.32 seconds node 2 and 3 have link status 2. In destTime we save timestamps for each node, and in destNode1 and destNode2 we save the nodes.



```
$ns_ at 18.969262732893 "$god_ set-dist 0 2 2"
$ns_ at 18.969262732893 "$god_ set-dist 1 2 1"
$ns_ at 18.969262732893 "$god_ set-dist 3 2 1"
```

```
QString data = LINK + QString::number(node1[i]) + " " + \
QString::number(node2[i]) + " " + QString::number(reachable[i]);

if (!guiObject->previewCheckBox->isChecked())

    topomanSock->writeDatagram ( data.toStdString().c_str(), data.length(), \
    QHostAddress::QHostAddress(guiObject->topoManPreferences->topomanHostLine->text()),\
    guiObject->topoManPreferences->topomanPortLine->text().toInt());
}
```

Figure 4.8: Steps from scenario files to UDP packet

Figure 4.8 gives an indication of the process when link status information is read from scenario file. The first step shows part of the scenario file where we have 3 link changes at the same time. In the second step we can see the information transformed into graphical objects of the nodes and the links, which is much easier to understand than reading the scenario file. The dark green line is communication link, while the light green line is broadcast link between the nodes. In the third step, we show code to explain how the information about each link change is sent as a UDP packet to the topology manager. The data (in the code) is e.g. is set to "link 0 2 2".

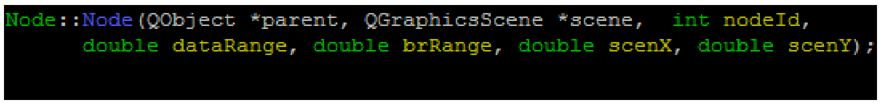
4.1.2 Node structure

An item which is a QGraphicsItem pointer is the visual concept of a node in the scene, we use the terms item as the specific pixmap representation of the node, while node itself is the whole object and all its members. A node object consists of these visual members on the scene:

- **The node pixmap (image of penguin, PDA, or a laptop)**
- **Node ID**
- **Broadcast circle**
- **Communication circle**
- **One or more links to neighbour nodes**

The node class provides a node object which has the QGraphicsItem item pointer for representation of the node in the scene each item has a pixmap which can be chosen to be either the popular penguin image, or a PDA or a laptop image which is provided for convenience. A broadcast and communication range if the scenario requires so. Each Node has a list of neighbor nodes and a list of outgoing lines (either communication or broadcast links) these lists are updated each time a node comes within range of another node, it is further explained how this is done in Section 4.3.3.

When a node pointer is constructed immediately after the scenario has been processed these values are read from the scenario file and given to the node constructor:



```
Node::Node(QObject *parent, QGraphicsScene *scene,  int nodeId,
     double dataRange, double brRange, double scenX, double scenY);
```

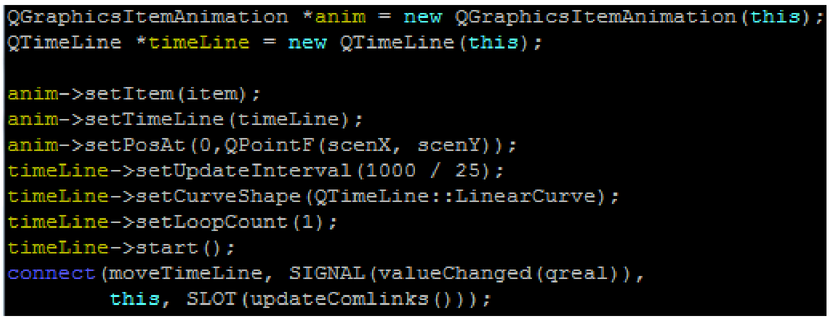
Figure 4.9: The Node class constructor

Figure 4.9 shows the class Node's constructor. The nodeId is the unique id of each node painted in the middle, it is set by number of nodes read from the scenario and the user can change the start node in the spin box field. The dataRange and brRange are used to check if the scenario includes drawing of communication and broadcast ranges. The scenX and scenY are the X and Y coordinates of that node according to the scene. The node class is responsible for positioning each node in the scene, updating each movement and drawing the communication and broadcast lines for each node, the animation techniques used to animate the nodes movements is further explained in Section 4.1.3.

4.1.3 Animation

QGraphicsItemAnimation class provides the animation support for QGraphicsItem, it is used together with a QTimeLine. We connect a slot to the signal valueChanged, which is emitted by the QTimeLine, and we do the needed updates on the moving nodes for each time this signal is emitted. For instance when nodes are moving around we must update the link coordinates between the moving nodes, since the links are only attached to the owner node on one end, but an update of the position to the node on the other end is necessary.

Figure 4.10 shows how we setup a timeline and an animation object to animate the movement of our nodes. The setPosAt function sets the position of an item in a given step value. The setUpdateInterval corresponds to a rate of 25 updates per second, further we set the curve shape of the timeline, in linear curve shape the value grows linearly (e.g., if the duration is 1000 ms, the value at time 500 ms is 0.5). The loopCount function basically keeps



```
QGraphicsItemAnimation *anim = new QGraphicsItemAnimation(this);
QTimeLine *timeLine = new QTimeLine(this);

anim->setItem(item);
anim->setTimeLine(timeLine);
anim->setPosAt(0,QPointF(scenX, scenY));
timeLine->setUpdateInterval(1000 / 25);
timeLine->setCurveShape(QTimeLine::LinearCurve);
timeLine->setLoopCount(1);
timeLine->start();
connect(moveTimeLine, SIGNAL(valueChanged(qreal)),
        this, SLOT(updateComlinks()));
```

Figure 4.10: A short explanation of the animation scheme

track of how many times the animation should be looped, a loop count of 0, will loop the animation in definitely.

4.1.4 The Structure of The GUI

We take a close look into the structure of the GUI, and the features it provides. In Figure 4.11, the GUI and Command line versions are deriving the Common functionalities in emulation from class Common. The Figure shows the basic flow of the application in both mods.

In Figure 4.12, we can see the continuation of the application flow, note that not all of the states in the application are shown in the figure, but only distinctive functions. As with the NEMAN Interface 0.5, we have provided the same functionalities in Figure 4.13 we can see a clean new look with the same options as version 0.5 but with several new features, such as the ability to fit all nodes in the scenario into the visual part of the scene.

In NEMAN Interface 0.5, the scale option is used to zoom in and out in the scene, although not all the nodes in the scenario are visible on the scene. The ability to change preferences on the server the topology manager is running on, these preferences are server address, server port, message port and routing port as shown in Figure 4.14. This option saves the user preferences in a file called <server.ini> in order to open the same preference for the user, when the interface restarted.

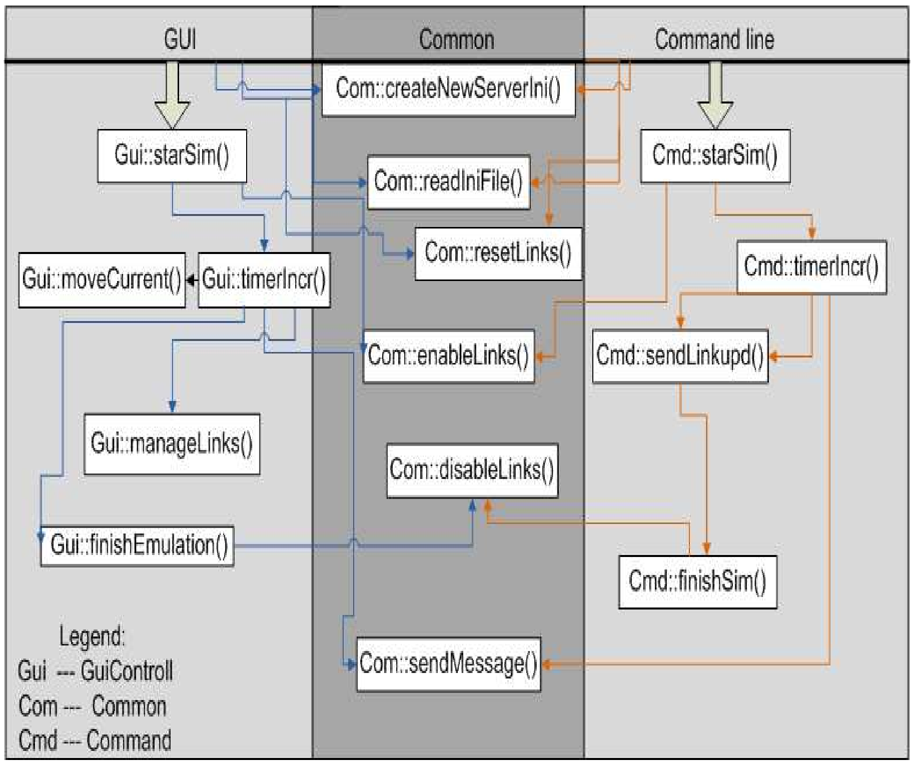


Figure 4.11: Program flow

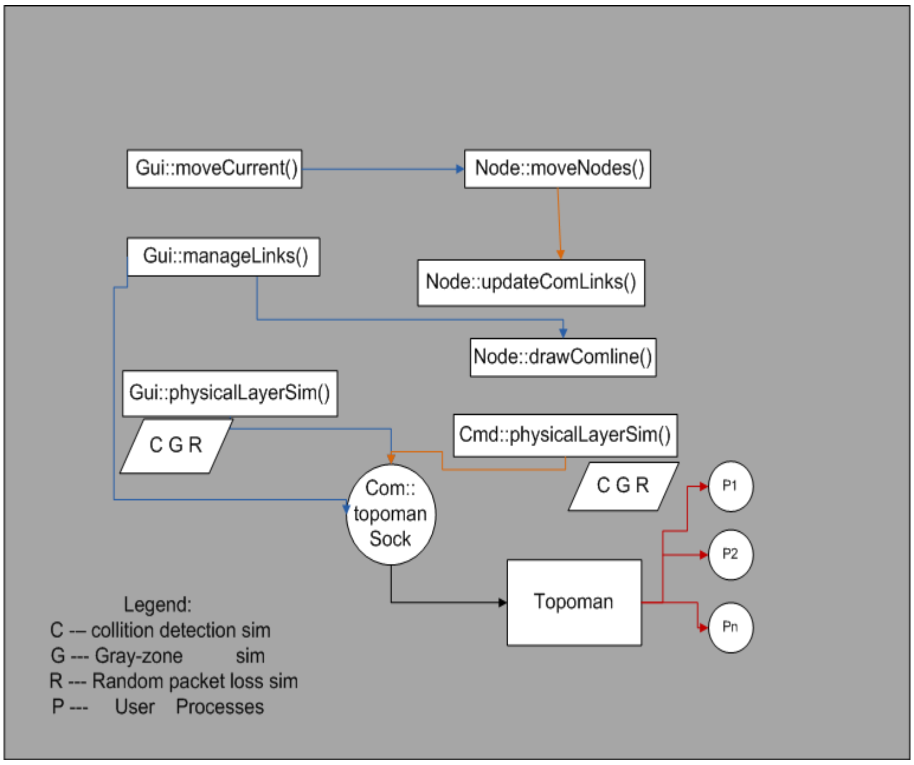


Figure 4.12: Program flow continued

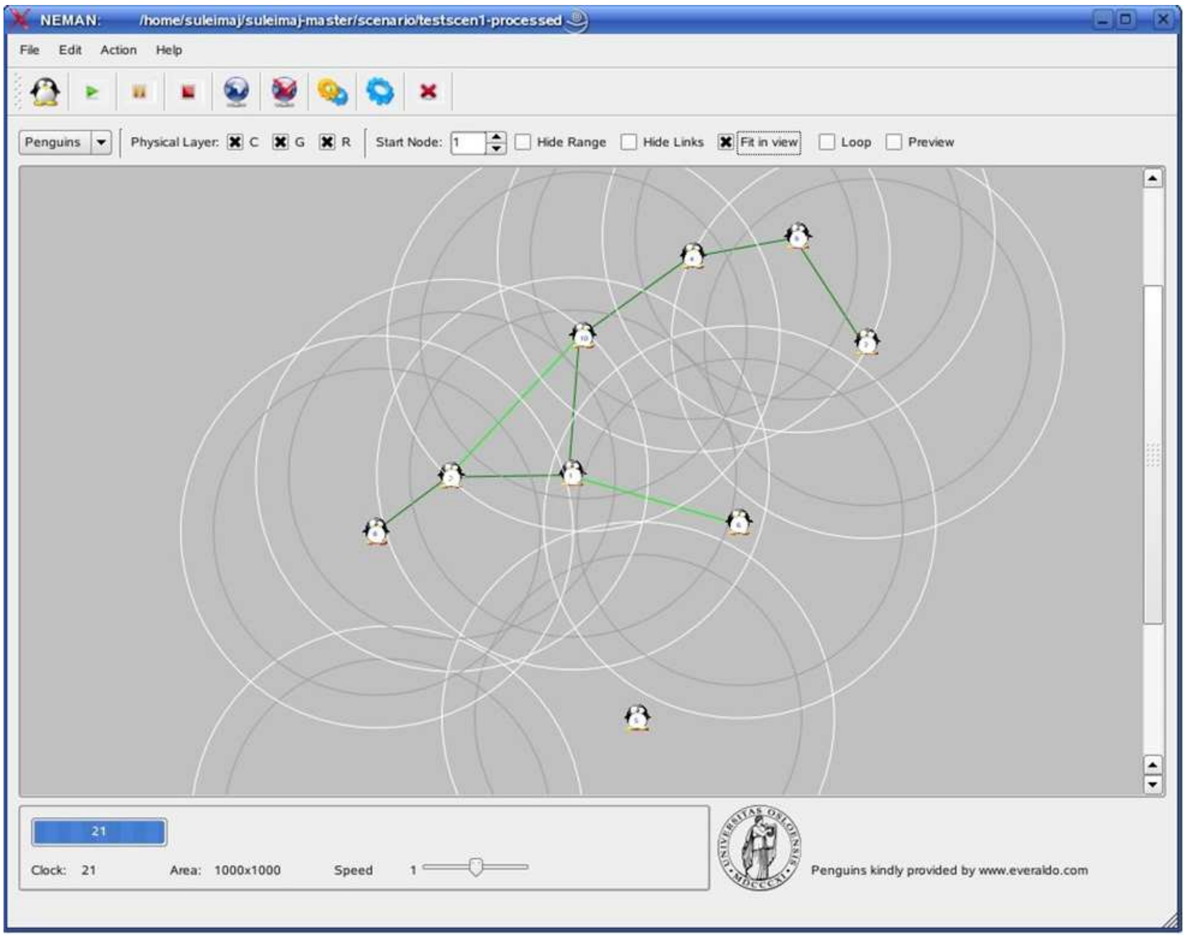
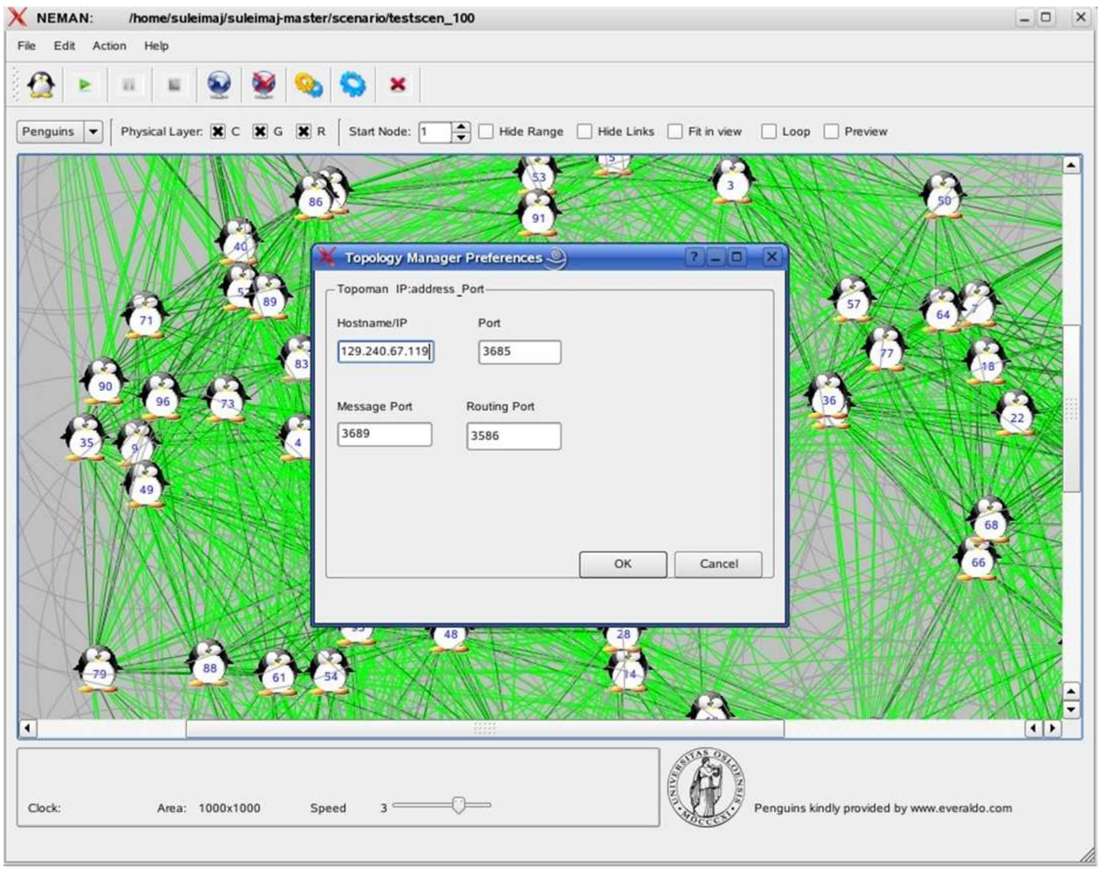


Figure 4.13: NEMAN interface GUI 1.0



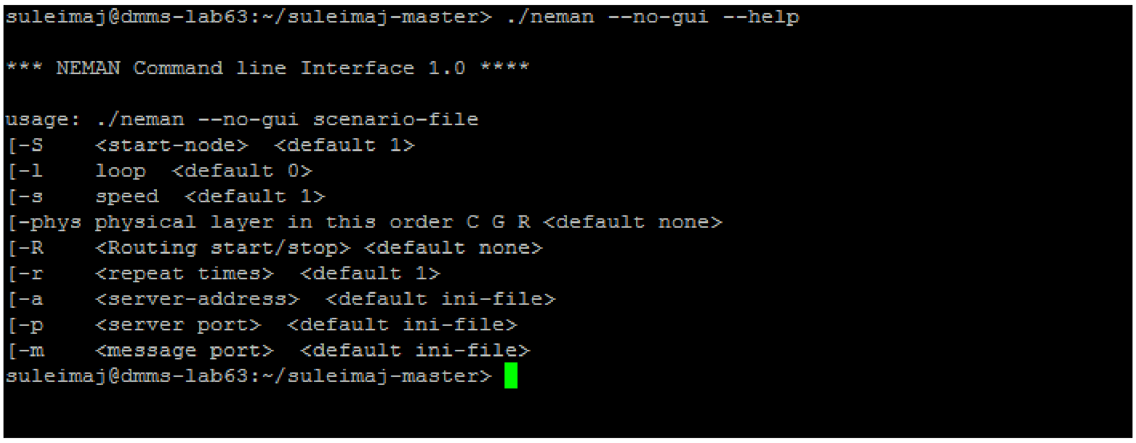
Topoman IP:address_Port

Hostname/IP: 129.240.67.119
Port: 3685
Message Port: 3689
Routing Port: 3586

Figure 4.14: NEMAN interface GUI version 1.0, topology manager preference

4.2 The Command line Interface

So far we have seen the implementation of the graphical part of our interface; here we describe the command line interface. This interface has been introduced as an alternative way to perform emulations. Command line based emulation can be carried out with minimal efforts without running any graphical servers. Much of the architecture behind the command line interface has been already introduced, as we can see in Figure 4.11 and 4.12, part of the code written for the graphical interface which does not involve graphical elements is reused.

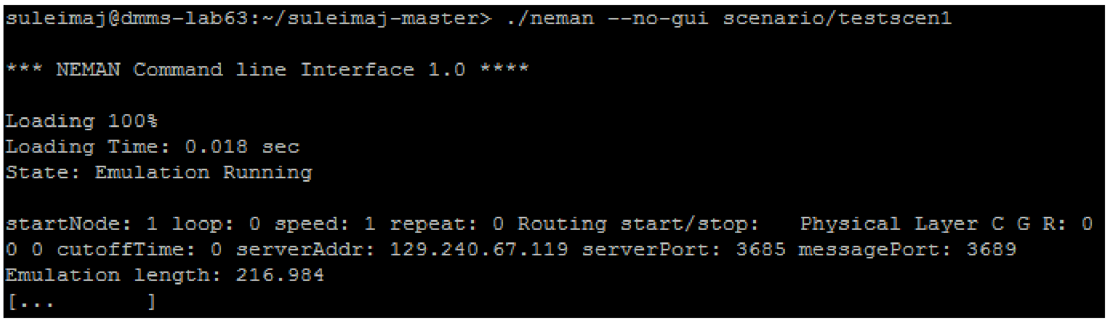


```
suleimaj@dmms-lab63:~/suleimaj-master> ./neman --no-gui --help

*** NEMAN Command line Interface 1.0 ****

usage: ./neman --no-gui scenario-file
[-S    <start-node>  <default 1>
[-l    loop  <default 0>
[-s    speed  <default 1>
[-phys physical layer in this order C G R <default none>
[-R    <Routing start/stop> <default none>
[-r    <repeat times>  <default 1>
[-a    <server-address>  <default ini-file>
[-p    <server port>  <default ini-file>
[-m    <message port>  <default ini-file>
suleimaj@dmms-lab63:~/suleimaj-master>
```

Figure 4.15: The command line options

Figure 4.15 shows the command line arguments the interface provides, this is the help option, displayed whenever the user enters wrong arguments to the interface.

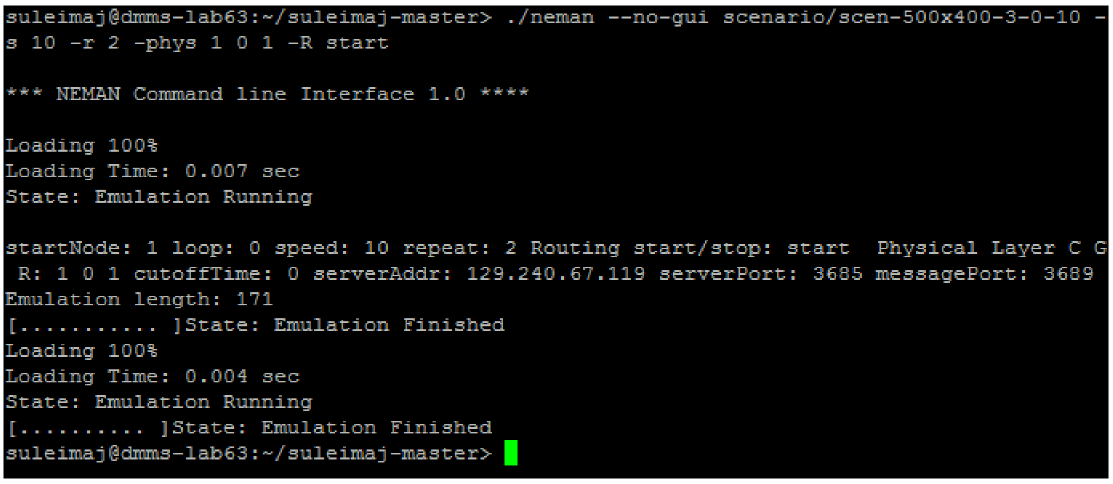
Figure 4.16 shows an emulation running with the command line interface, as we can see testscen1 is the scenario file given, a percentage based progress bar shows the loading of the scenario, and in the line after we can see how much time was spent on loading. After that we can see that the emulation has started to run and the values given to the interface are displayed either user given or default set values for steps like, start node, speed and etc. The length of this emulation in seconds is also given followed by a progress bar



```
suleimaj@dmms-lab63:~/suleimaj-master> ./neman --no-gui scenario/testscen1

*** NEMAN Command line Interface 1.0 ****

Loading 100%
Loading Time: 0.018 sec
State: Emulation Running

startNode: 1 loop: 0 speed: 1 repeat: 0 Routing start/stop:   Physical Layer C G R: 0
0 0 cutoffTime: 0 serverAddr: 129.240.67.119 serverPort: 3685 messagePort: 3689
Emulation length: 216.984
[...        ]
```

Figure 4.16: The command line running default

displaying the progress of the emulation.

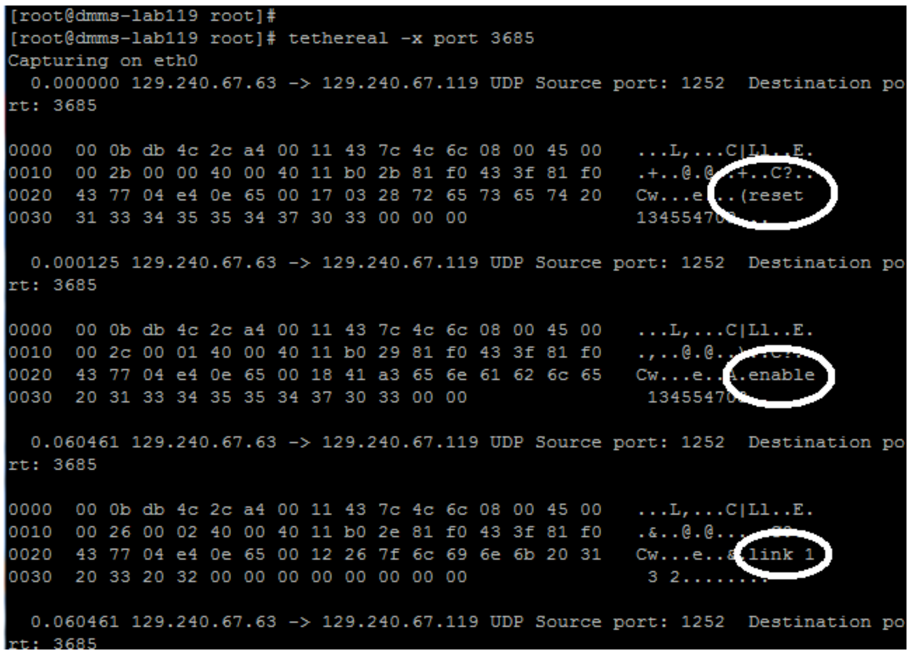


```
suleimaj@dmms-lab63:~/suleimaj-master> ./neman --no-gui scenario/scen-500x400-3-0-10 -
s 10 -r 2 -phys 1 0 1 -R start

*** NEMAN Command line Interface 1.0 ****

Loading 100%
Loading Time: 0.007 sec
State: Emulation Running

startNode: 1 loop: 0 speed: 10 repeat: 2 Routing start/stop: start  Physical Layer C G
 R: 1 0 1 cutoffTime: 0 serverAddr: 129.240.67.119 serverPort: 3685 messagePort: 3689
Emulation length: 171
[.......... ]State: Emulation Finished
Loading 100%
Loading Time: 0.004 sec
State: Emulation Running
[.........  ]State: Emulation Finished
suleimaj@dmms-lab63:~/suleimaj-master>
```

Figure 4.17: The command line repeating the emulation

Figure 4.17 shows an emulation with several arguments given, as we can see this emulation is repeated 2 times with the -r option, speed is also set to -s 10, the chosen physical layers to simulate is collision detection and random packet loss, while gray-zone simulation is disabled. The routing protocol is also started with -R start.

Figure 4.18 shows the topology manager receiving UDP packets from either the command line or the graphical interface when emulation is executed. The payload of the packet is partially readable, commands such as reset, enable and link are visible. With the usage of networking tools such as tethereal



```
[root@dmms-lab119 root]#
[root@dmms-lab119 root]# tethereal -x port 3685
Capturing on eth0
  0.000000 129.240.67.63 -> 129.240.67.119 UDP Source port: 1252  Destination po
rt: 3685

0000  00 0b db 4c 2c a4 00 11 43 7c 4c 6c 08 00 45 00   ...L,...C|Ll..E.
0010  00 2b 00 00 40 00 40 11 b0 2b 81 f0 43 3f 81 f0   .+..@.@..+..C?..
0020  43 77 04 e4 0e 65 00 17 03 28 72 65 73 65 74 20   Cw...e...(reset
0030  31 33 34 35 35 34 37 30 33 00 00 00               134554703...

  0.000125 129.240.67.63 -> 129.240.67.119 UDP Source port: 1252  Destination po
rt: 3685

0000  00 0b db 4c 2c a4 00 11 43 7c 4c 6c 08 00 45 00   ...L,...C|Ll..E.
0010  00 2c 00 01 40 00 40 11 b0 29 81 f0 43 3f 81 f0   .,..@.@..)..C?..
0020  43 77 04 e4 0e 65 00 18 41 a3 65 6e 61 62 6c 65   Cw...e..A.enable
0030  20 31 33 34 35 35 34 37 30 33 00 00                134554703..

  0.060461 129.240.67.63 -> 129.240.67.119 UDP Source port: 1252  Destination po
rt: 3685

0000  00 0b db 4c 2c a4 00 11 43 7c 4c 6c 08 00 45 00   ...L,...C|Ll..E.
0010  00 26 00 02 40 00 40 11 b0 2e 81 f0 43 3f 81 f0   .&..@.@.....C?..
0020  43 77 04 e4 0e 65 00 12 26 7f 6c 69 6e 6b 20 31   Cw...e..&.link 1
0030  20 33 20 32 00 00 00 00 00 00 00 00                3 2........

  0.060461 129.240.67.63 -> 129.240.67.119 UDP Source port: 1252  Destination po
rt: 3685
```

Figure 4.18: Topology Manager, listening on control channel 3685

and tcpdump, we obtained results confirming that our interface is capable of communicating with the topology manager on control channel 3685.

4.3 Summary

In this chapter we have explained the implementation of the graphical and command line interfaces. We have shown with examples how a scenario file, which is either NS-2 based or mkdist filtered are processed by the interface with the QRegExp class. Each step on how we store the information from the scenario files and into the appropriate data structures are also shown.

Furthermore, we have explained how the link changes are read and sent as UDP packets to the topology manager. We explained what a node is and what members it consists of. The Node class is responsible of maintaining each node coordinate and to draw, change or delete links. The Node class also provides the animation schemes.

The command line interface is also separately explained, it has much in common with the GUI. It is designed to offer the same functionalities as the GUI, but it is also much more flexible to carry out customized experiments. Such includes the possibility to be used with scripting programs.

During the implementation we have tested the interface using tethereal and tcpdump to confirm the network communication with the topology manager.

Chapter 5

Evaluation

In this chapter, we compare our NEMAN Interface 1.0 with NEMAN Interface 0.5. We confirm our theory and technical experience which stated that NEMAN Interface 0.5 is the bottleneck in the NEMAN emulation environment.

We have done this by benchmarking both interfaces on several scenario files with difference on sizes (file size in KB/Mb), node numbers, time and size of the 2D scene. In our comparison we are interested in knowing the following for both interfaces:

- The time it takes to load scenario files, and
- the time it takes to draw the links, when prepare or start is clicked.

These two factors determine the level of lag the GUI is exposed to. In the tests we use line count which indicates the number of lines on the scenario file that are processed. We use line count because both NEMAN Interfaces are processing each line with regular expression, thus we are interested in knowing how fast they perform the processing of each line.

5.1 Performance Analyses of NEMAN GUI version 0.5 and 1.0

We perform benchmark tests on the same functionalities in both interfaces. These are as mentioned functions that initiate the loading of scenario files and the drawing of the graphical representation of the communication links among the nodes. We carry out these tests on a Dell Optiplex GX280 with Intel Pentium 4 2.80GHZ CPU and 1GB of physical memory; the computer is running Suse Linux 10.0 operating system.

5.1.1 Timing GUI version 0.5

In order to measure the performance of the specified functions in the Tcl/Tk based GUI, we use the built in clock clicks timing procedure. It returns a high-resolution time value as a system-dependent integer value. The unit value is the highest resolution available on the system such as CPU cycle counter [6]. The code below states how we measure the elapsed time.

```
#we declare a global timer variable.
global timer t0

##iemul code, line: 1367

set t0 [clock clicks -millisec] #start the clock
process_scen_file $fd #function that loads the scenario file
puts "[expr ([clock clicks -millisec]-$t0)/1000.] sec"
```

As we can see we are measuring the function process_scen_file, this is the function used to load and process scenario files. We start the timer and perform the measurements; afterwards we read the time it took.

The code below shows measurements of how fast links are drawn on the event based function "emu_engage", we create our own function that measures emu_engage because it is implemented to be called from called from an event based function as shown in line 965 in the iemul source file.

```
##iemul code, line: 920

proc measurelinks {} {
 set t1 [clock clicks -millisec]
 emu_engage
 puts "[expr ([clock clicks -millisec]-$t1)/1000.] prepare sec"
}
```

measurelinks is called where emu_engage used to be called from

```
##iemul code, line: 965

button $t1Frame.engage -text "Prepare" -command "measurelinks"
```

Now we have setup the ability to time the specified functions, and further we continue with our benchmark.

5.1.2 Timing Interface version 1.0

In NEMAN Interface 1.0 we also measure the same functions, but this time with the use of the Qt library, we measure loading for both the graphical and command line interface, in order to compare their performance with each other as well. We use the QTime class in the Qt library; this class provides clock time functions. QTime reads the current time from the system clock and measure a span of elapsed time. The accuracy of QTime depends on the accuracy of the underlying operating system.

The code shown below explains how we measure the loading time of the GUI and the command line of NEMAN Interface 1.0. We create a QTime object mtime, and start the time, then we call the function which performs the loading of a scenario file, and at the end we show the elapsed time of this function.

```
void GuiControll::open_scenfile()
{
 QTime mtime;
 double t0;
 mtime.start();
 t0 = mtime.elapsed();
 //processing a scenario file
 std::cout<<"time: load:"<<(mtime.elapsed()-t0)/1000.0<<"sec"<<std::endl;
}
```

The code stated below which is more or less the same code as shown above, is only used to measure the time it takes for the links to be drawn when prepare or start button is triggered (only the GUI performs this task).

```
void GuiControll::prepare()
{
 QTime mtime;
 mtime.start();
 double t0 = mtime.elapsed();
 setReachable();
 std::cout<<"time: prepare:"<<(mtime.elapsed()-t0)/1000<<"sec"<<std::endl;
 }
```

5.1.3 Results

This section describes the data we use to conduct our tests with, and the results we obtain. The tests are divided into two groups of each three tables that contain the benchmark data for loading and preparing scenario files, followed by the results showed in graphs.

We begin to test the loading performance for both NEMAN Interfaces; in NEMAN Interface 1.0 we also test the command line loading performance.

Table 5.1: Loading scenarios files, examine 1

Scenario filename	Nodes	Scenario length in sec.	Scene
t1-2000	100	not set	1000 x 1000
t1-4000	100	not set	1000 x 1000
t1-6000	100	1.9	1000 x 1000
t1-8000	100	9.4	1000 x 1000
t1-10000	100	17.5	1000 x 1000

The table contains the name of the scenario files used, prefixed[1] with length in number of lines, the number of nodes, the length of the scenario and size of the graphical scene. Some of the scenario files have time not set; this is because we have shortened some of the scenario files in order to have increasing size.

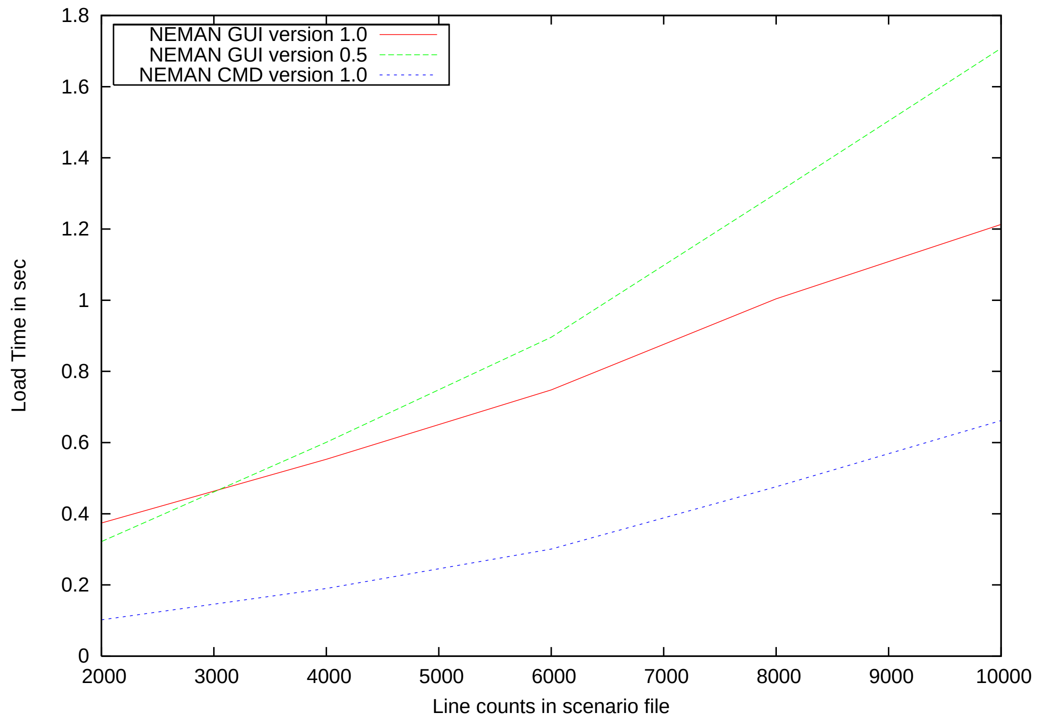


Figure 5.1: Measuring load time on NEMAN Interface 1.0 and NEMAN Interface 0.5, with scenario files on different sizes

[1]The source code and all test samples can be found in http://folk.uio.no/suleimaj/master, or in the CD provided in Appendix C

In the Figure 5.1 which is based on the dataset given in Table 5.1 we see an interesting initial value for NEMAN Interface 0.5, when line count is less than 3000 it is actually faster than our solution and is due to the shortness of the scenario file. As we also can see when it reaches 3000 lines the processing time grows higher than with NEMAN Interface 1.0.

As the Line count grows, the loading time also continues to grow. In the graphical part of NEMAN Interface 1.0 we have a linear rise in loading time; the command line in NEMAN Interface 1.0 also follows the same pattern, except that it is quite faster. This is because the command line skips the processing of graphical objects, which reduces the time.

Table 5.2: Loading scenarios files, examine 2

Scenario filename	**Nodes**	**Scenario length in sec.**	**Scene**
testScen_61	5	50	500 x 400
testScen_219	10	100	500 x 400
testScen_4415	50	200	500 x 400
testScen_31121	100	500	500 x 400
testScen_42576	200	1000	500 x 400

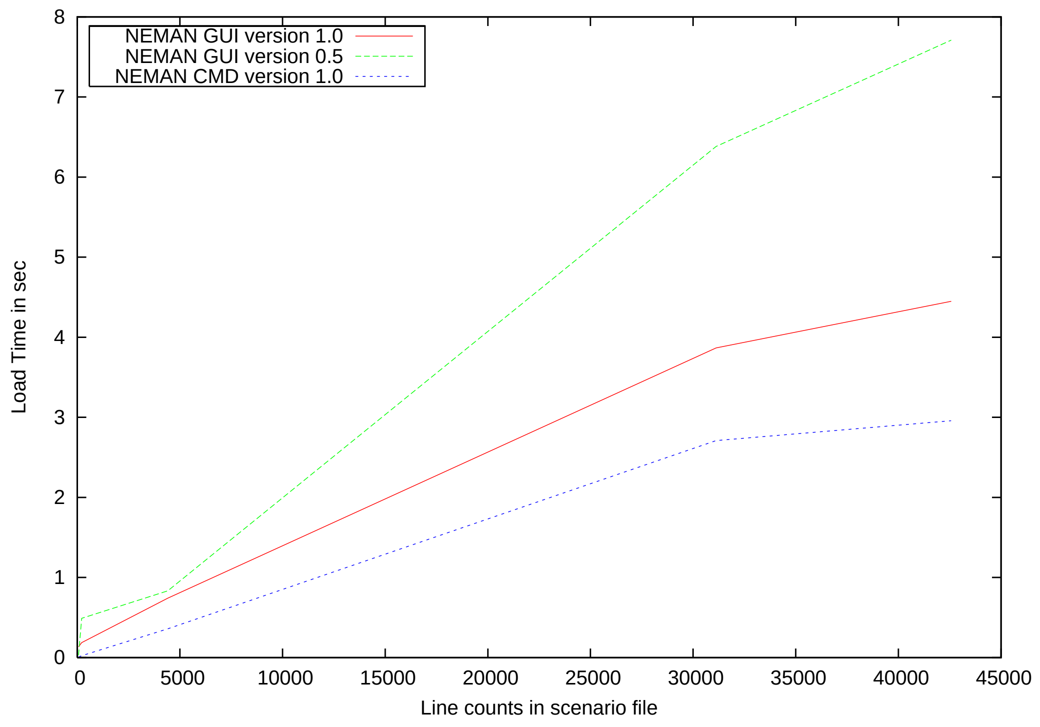


Figure 5.2: Measuring load speed, with the size of scenarios ranging from 61 upto 45000 lines

In Table 5.2 we use scenario files ranging from 61 lines to 42000 in each file, and number of nodes from 5 up to 200 (number of nodes affects the size of the file).

In Figure 5.2 we see in the beginning that NEMAN Interface 0.5 is a bit slower than previous graph, we also notice that after reaching the 5000 lines it enters a steep rise. We have conducted the tests several times on the same scenario file in order to achieve the same results, but we have encountered problems to achieve that with NEMAN Interface 0.5, while NEMAN Interface 1.0 has given the same results in repeated tests. We compensated this problem by restarting NEMAN Interface 0.5 on each test, this gives us fresh data structures, and the results were closely corresponding to each other afterwards. For instance when we run the scenario t3-70000 on first test we achieved loading time of 13.66 seconds, when the same scenario was again loaded the time reached 24.76. However if we restarted the GUI and run the same test we achieved a loading time of 13.54, this shows us that we are forced to restart the GUI for every time we conduct a measurement.

Table 5.3: Loading scenarios files, examine 3

Scenario filename	**Nodes**	**Scenario length in sec.**	**Scene**
t3-17500	100	57.2	1000 x 1000
t2-35000	100	185.2	1000 x 1000
t3-52500	100	321.2	1000 x 1000
t3-70000	100	463.1	1000 x 1000

In our final test result for the loading part, we increment the size of each file with 17500 lines as shown in Figure 5.3, this affects the loading time of NEMAN Interface 0.5, and Figure 5.3 follows same loading time pattern as Figure 5.2.

Table 5.1.3 is showing that we are using small scenario files of 1000 lines up to 5000, however the number of nodes are 100 and the scene is big. This causes, as shown in Figure 5.4, a significant difference in prepare time between NEMAN Interface 1.0 and 0.5 in loading time. NEMAN Interface 1.0 prepare time is almost no noticeable and range between 0.016 seconds and at most

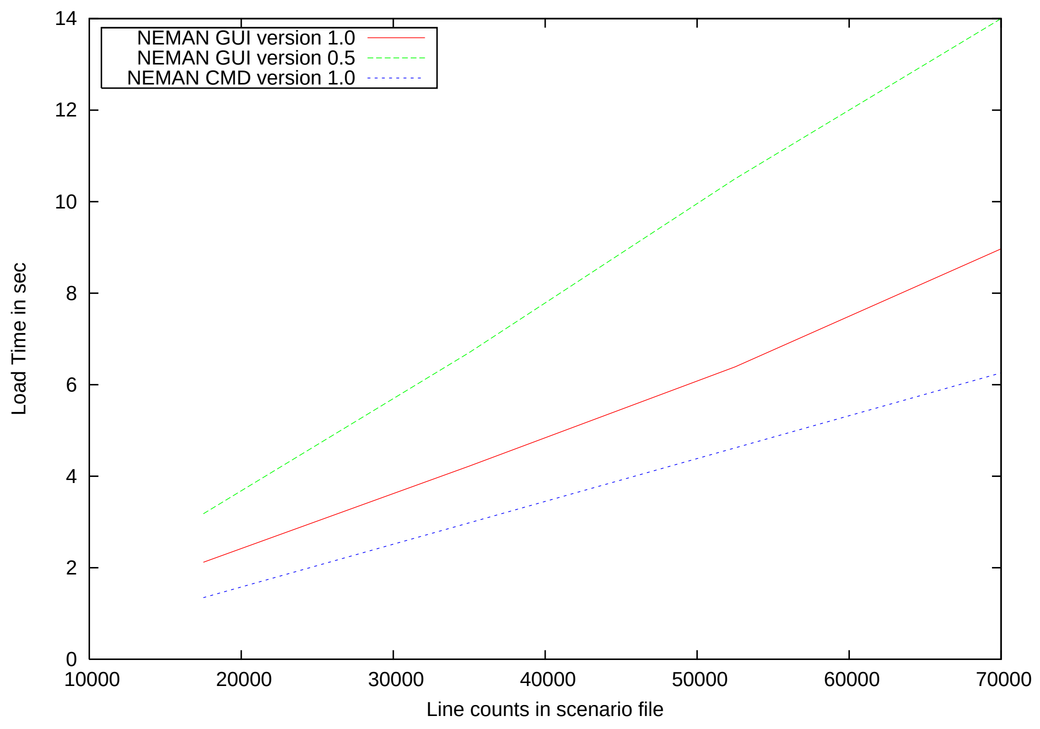


Figure 5.3: Measuring load speed, with the size of scenarios ranging from 17500 upto 70000 lines

Scenario filename	Nodes	Scenario length in sec.	Scene
t3-prepare-1000	100	not set	1000 x 1000
t2-prepare-2000	100	not set	1000 x 1000
t3-prepare-3000	100	not set	1000 x 1000
t3-prepare-4000	100	not set	1000 x 1000
t3-prepare-5000	100	not set	1000 x 1000

Table 5.4: Result on prepare examine 1, udpsend is called

0.099 seconds, while NEMAN Interface 0.5 range between 1.694 seconds and with slowest prepare time of 9.645 seconds. As we mentioned in Chapter 2, the limit delay a user accepts is 0.7 seconds, before considering the program as unresponsive.

We have however found the real bottleneck, and it is caused by as we earlier mentioned udpsend, a C program with the purpose to send commands to the topology manager. Udpsend is called every time a link is established between

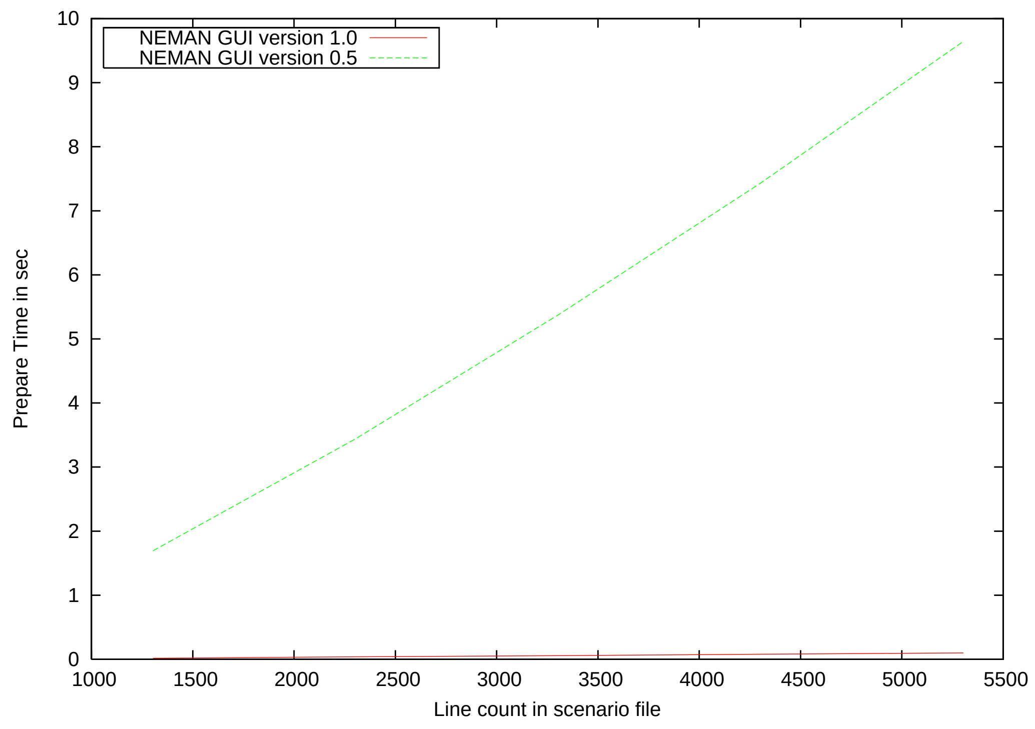


Figure 5.4: Measuring how fast links are drawn when prepare button is clicked on scenarios with different size: udpsend is called on each link that is drawn

two nodes. This overhead can be avoided by implementing socket communication with the topology manager in the Tcl/Tk code, Tcl only supports TCP socket communication, and however there are extension packets that also support UDP socket programming.

Scenario filename	Nodes	Scenario length in sec.	Scene
t3-prepare-1000	100	not set	1000 x 1000
t2-prepare-2000	100	not set	1000 x 1000
t3-prepare-3000	100	not set	1000 x 1000
t3-prepare-4000	100	not set	1000 x 1000
t3-prepare-5000	100	not set	1000 x 1000

Table 5.5: Preparing scenarios files, examine 2

In Table 5.1.3 we are starting with small scenario files of 1000 lines, the scenario files contain 100 nodes each, in a 1000 x 1000 scene. In Figure 5.5 we can see a much faster prepare time for NEMAN Interface 0.5 with

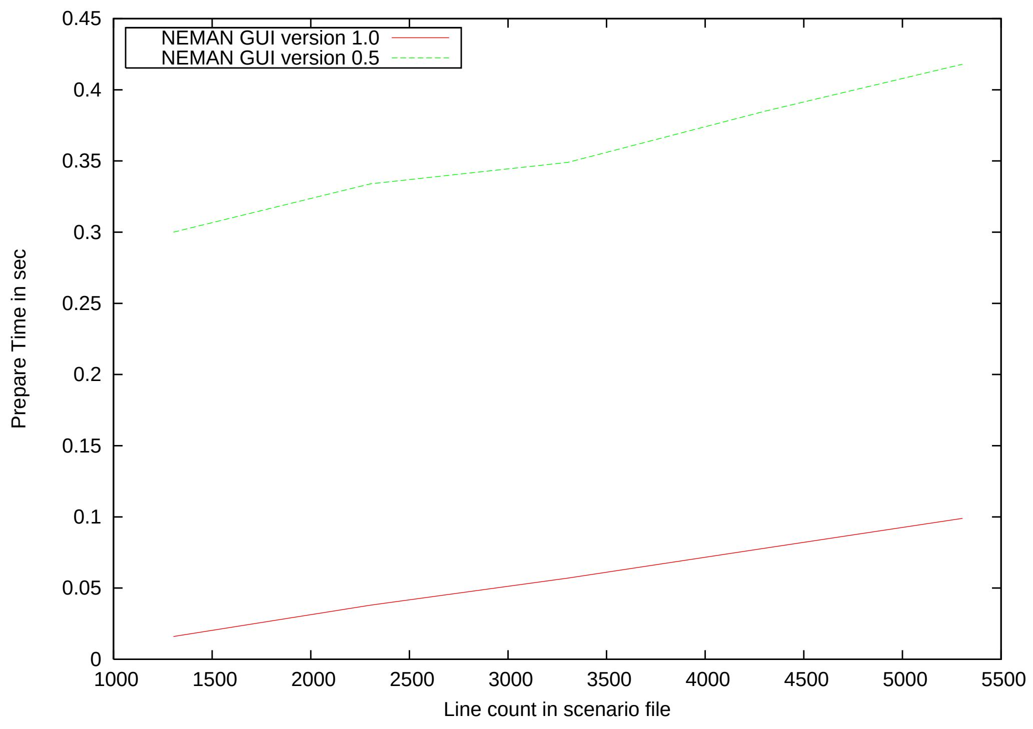


Figure 5.5: Result on prepare examine 2, when udpsend is not called

0.3 seconds when line counts are 1500 and 0.418 as the slowest prepare time. Even though NEMAN Interface 1.0 is still faster, with the same prepare time values as previous test. We have gained 9.277 seconds when we avoid to use udpsend by commenting the following line in the source code of NEMAN Interface 0.5:

```
##iemul code, line: 321
#send_command "link $node1 $node2 $distance"
```

The commented line used to call the udpsend program, and by commenting it we have gained as Figure 5.5 shows 9.277 seconds faster in visual performance. Although there is no communication of link status sent to the topology manager. Although we are interested in finding and mitigating the performance lag on the visual appearance. As already suggested the socket communication in NEMAN Interface 0.5 should be integrated with the application itself, instead of using external programs.

Scenario filename	**Nodes**	**Scenario length in sec.**	**Scene**
prepare_17182_nodes	100	49.9	1000 x 1000
prepare_56922_nodes	200	49.9	1000 x 1000
prepare_84573_nodes	300	22.5	1000 x 1000

Table 5.6: Preparing scenarios files, examine 3

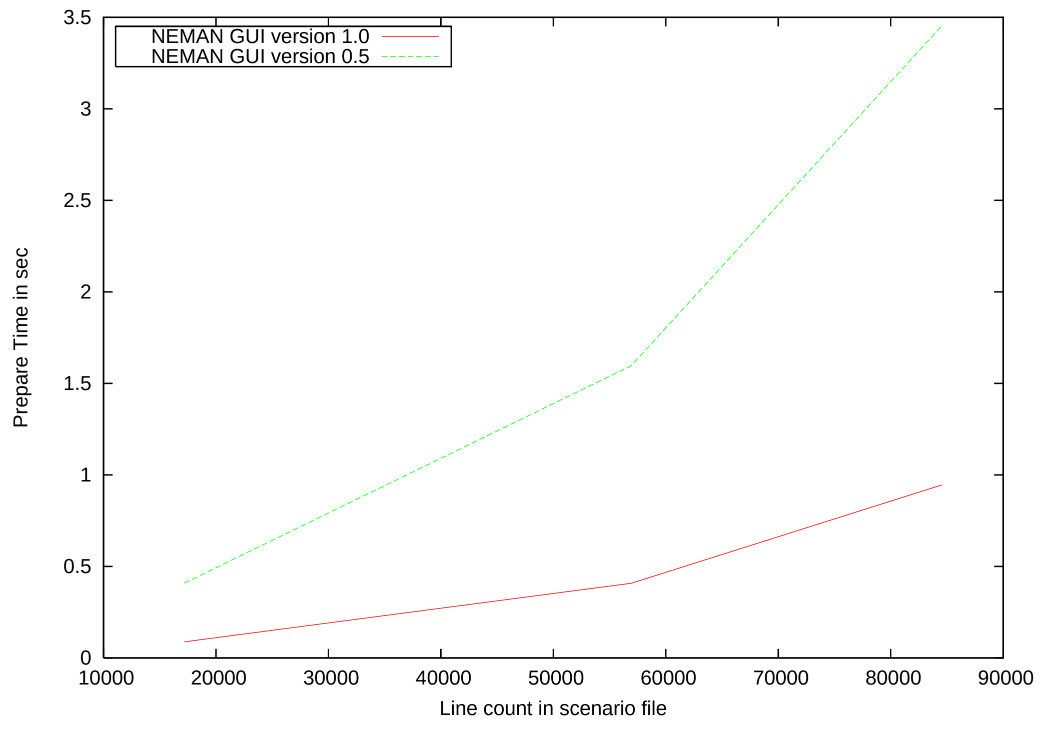


Figure 5.6: Result on examine 3, when udpsend is not called

In Table 5.1.3 we have increased the number nodes from 100 nodes up to 300 nodes, size of the scenario files are ranging from 17000 and up to 84000 lines. During this test we are also testing the prepare time of NEMAN Interface 0.5 without udpsend. The results as we can see looks even more promising for NEMAN Interface 0.5 with 0.409 seconds prepare time in the beginning of the test and at the end 3.455 seconds. Clearly NEMAN Interface 1.0 is 2.509 seconds faster. This is not as significant as with earlier tests where udpsend was used in NEMAN Interface 0.5.

5.2 Results from The End User Questionnaire

We have carried out an end user questionnaire on user experience of NEMAN Interface 1.0. In the same way we did when we evaluated NEMAN Interface 0.5 in order to get feedback from the users of NEMAN.

- **How is your experience with the usage of this application?** possible answers :
  1. -Intuitive
  2. -Good
  3. -Difficult
  4. -Very difficult

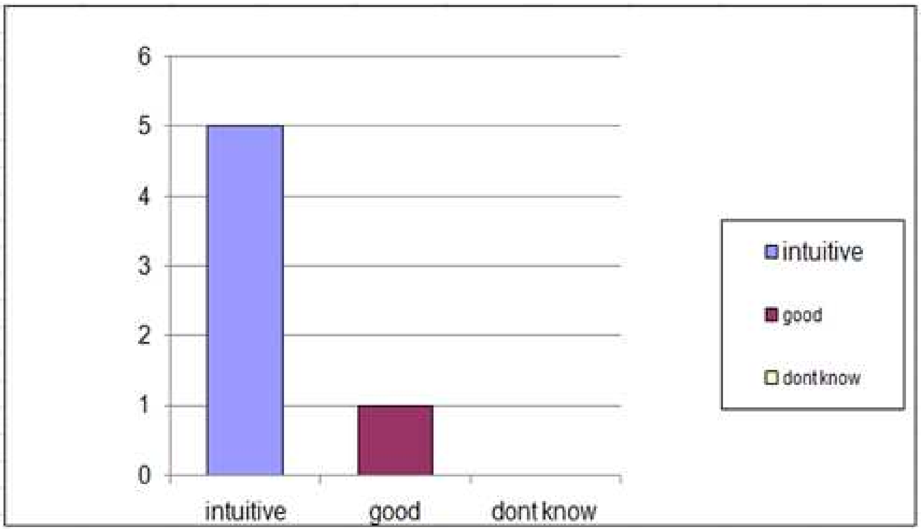


Figure 5.7: Question 1, on user experience

As we can see from Figure 5.7, 4 people answered that the interface is intuitive while a person answered that it is good. This is due to the minimalistic approach. When explaining their choices, the participants answered that even though some of the icons are not intuitive but they have popup help balloons explaining the meaning of each button. Other participants answered that it resembles the other GUI but more intuitive.

- **How fast is the learning threshold?** possible answers :

  1. -Very fast
  2. -Fast
  3. -Slow
  4. -Very slow

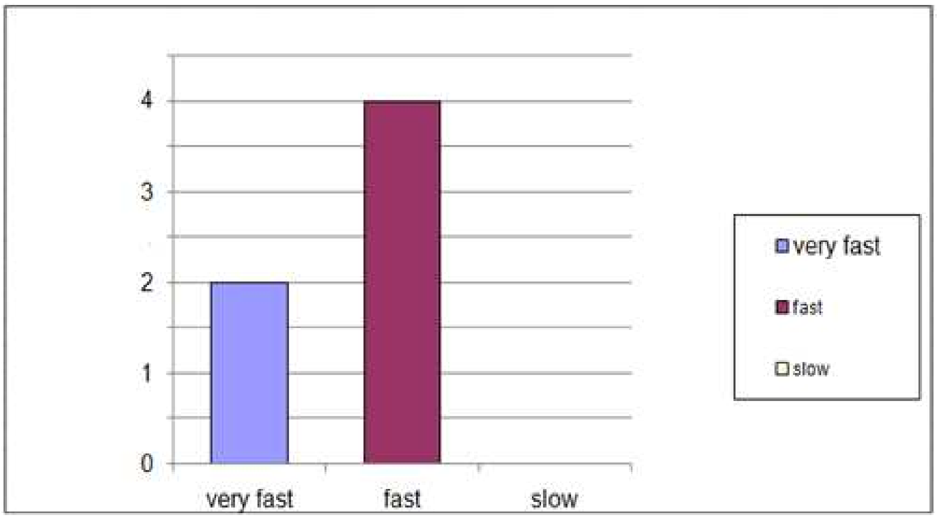


Figure 5.8: Question 2, on learning threshold

In Figure 5.8, 2 people answered the interface is very fast to work with, while 4 people answered it is fast. The participants also explained with, that the GUI is straightforward and that it is easy to find all functions instantly.

- **Are you satisfied with the visual performance of the GUI?** possible answers :

  1. -Yes
  2. -No
  3. -Don't know

In Figure 5.9 5 people answered that they are satisfied with the visual performance, while one person is not satisfied with the visual performance. The

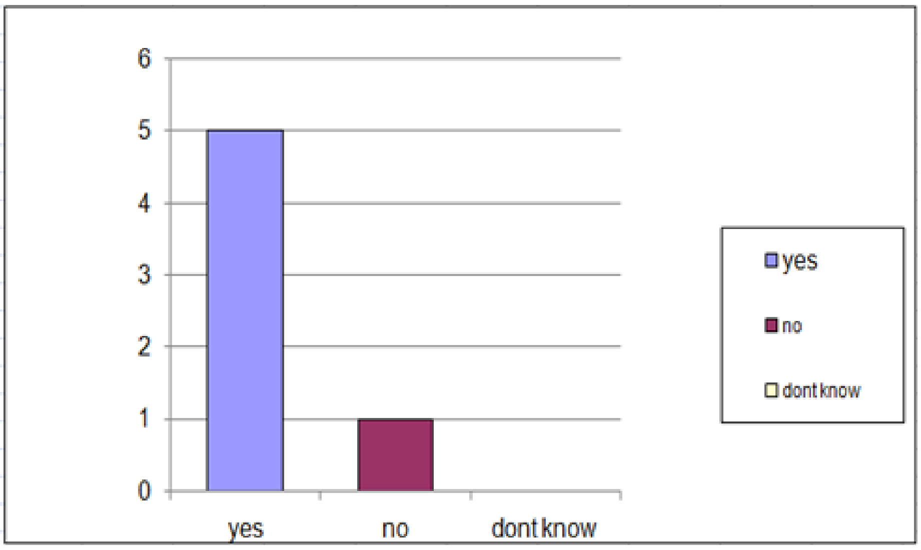


Figure 5.9: Question 3, on visual performance

author of NEMAN has discovered that when 100 nodes are emulated, the timer is not able to follow up with the drawing. We noticed this problem late in the development phase and we are presently working to fix this problem.

- **Are you able to do your work efficient with the application?** possible answers :

  1. -Yes
  2. -No
  3. -Don't know

Regarding the answers on Figure 5.10, 4 people are able to do their work efficient with NEMAN Interface 1.0, while 1 person regard it as not efficient and the last person has no opinion on this question. The author of NEMAN also found few other bugs regarding the fit in view option which needs few adjustments. When considering the command line interface all the participants are pleased with it and no one experienced problems with it.

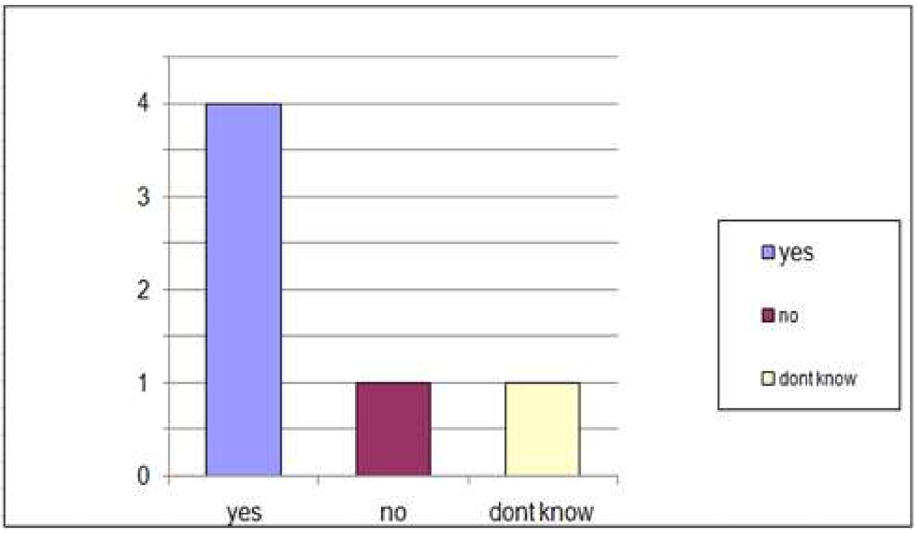


Figure 5.10: Question 4, on work experience

5.3 Summary

In our benchmark comparison of NEMAN interfaces, we have observed that our solution is quite faster in all the tests, we have discovered that NEMAN Interface 0.5 is faster when loading small scenario files, but differences are in milliseconds which are not noticeable for the users. However when big scenario files ranging from 17000 lines and up to 70000 lines are used in emulation, the user will notice the slow loading in NEMAN Interface 0.5. The winner of the loading tests, with fastest loading times is the command line interface.

Load examine	**NEMAN Interface 0.5**	**NEMAN Interface 1.0**
examine 1	0.965	0.778
examine 2	8.595	5.425
examine 3	3.089	1.875

Table 5.7: Average loading time for both interfaces

In Table 5.3 we can see the average loading time for the both NEMAN Interfaces.

We have discovered the bottleneck issue when the nodes are prepared or started in a scenario, the fact that udpsend which is an external program that is called every time a link change has happened in the scenario, have great effect on downgrading the performance of NEMAN Interface 0.5.

In Table 5.3 we can see the average results on the preparing time in seconds from the test examines we have carried out. Considering these facts we have achieved our goal of implementing a fast GUI and command line interface for NEMAN.

Prepare examine	**NEMAN Interface 0.5**	**NEMAN Interface 1.0**
examine 1	5.518	0.0576
examine 2	0.352	0.0576
examine 3	1.82	0.486

Table 5.8: Average prepare time for both interfaces

Chapter 6

Conclusion

In this thesis, we have designed and developed an enhanced user interface, which consists of a GUI and a command line interface for NEMAN. This interface is developed due to the performance lag and the possible license issues that can be caused by the previous GUI of NEMAN.

In this Chapter, we will shortly summarize the work we have carried out in Section 6.1 and in Section 6.2, we present suggestions for improvements.

6.1 Contribution

Mobile Ad-hoc Networks are widely evolving, with the help of emulation tools as test bed for MANETs, researchers are capable of conducting customized emulation experiments to test and debug new protocol solutions. During our research we discovered that the GUI of the Network emulator NEMAN lags in its visual performance.

We have thus designed and developed a new user interface called NEMAN Interface 1.0. This interface, which is developed with Qt and the C++ programming language, outperforms NEMAN Interface 0.5 and also provides several new features.

We have noted that NEMAN Interface 0.5 which is Tcl/Tk based, is using a

C program to send each link change in emulation to the topology manager, and causing a great overhead.

In our evaluation experiments we conducted benchmark tests to compare both interfaces on two aspects: which was the loading time and the preparing time on several scenario files with differences on size, nodes, time length and area size. The results obtained from this test showed that our new interface is significantly faster in loading and preparing time.

During the evaluation stage we also carried out a questionnaire regarding the end users experience on the new interface. Most of the users accepted NEMAN Interface 1.0, as an intuitive tool and also with good performance. The participants of these questionnaires were also given the opportunity to include new ideas and features which are described in Section 6.2.

6.2 Future Work

Due to the short time this thesis was given, there were limited possibilities for making more improvements within the new GUI. New ideas to improve and to add new features were constantly emerging, but, we can say that we have achieved our goals to complete most of the wanted features. Nevertheless there are always possibilities to do more improvements, but with the good documentation and object-oriented approach, we have provided an interface which is flexible to be maintained over time. As mentioned in Chapter 5, we are currently working to improve the ability to show the timer according to how fast the nodes are drawn.

The researcher's feedback regarding comments and suggestions for additional functionalities are the opportunity to:

1. Move a specific node on the scene.
2. Create scenario files from the GUI.
3. Click on a node and send message to another node.

4. List a buffer of the 10 last used scenario files.

These suggestions are functionalities provided for convenience and are worth investigating in the future.

Appendix A

The Header Files

A.1 GuiInit.h

```
#ifndef GUIINIT_H
#define GUIINIT_H

#include "Preference.h"
#include "GuiControll.h"
#include "ui_neman_gui.h"
#include <QResource>
#include <QObject>
#include <QGraphicsView>
#include <QGraphicsScene>

class GuiControll;
/**
 * @Author Suleiman H. Jama
 *
 * @version NEMAN GUI 1.0
 *
 * we set up the graphicsview, scene, and slot connections for
 * buttons and widgets.
 * A GuiControll object is created.
 Documentation of this class is found in http://folk.uio.no/suleimaj/master
 */

class enhancedGui : public QMainWindow, public Ui::neman_gui
{
    Q_OBJECT
    public:
    enhancedGui();
    QGraphicsScene *scene;
    GuiControll *guiSim;
    Preference *topoManPreferences;

    private slots:
        void quit_neman();
        void topomanPreferencedialog();

};
//allows us to access gui objects (pointers) from other classes
extern enhancedGui *guiObject;

#endif //GUI_INIT_H
```

A.2 Common.h

```
#ifndef COMMON_H
#define COMMON_H
#include <QObject>
#include <QImageReader>
#include <QDebug>
#include <QAction>
#include <QMessageBox>
#include <QSettings>
#include <QRegExp>
#include <iostream>
#include <QFileDialog>
#include <QVector>
#include <QGraphicsScene>
#include <QtAlgorithms>
#include <QMap>
#include <QTimer>
#include <QTime>
#include <QTimeLine>
#include <QUdpSocket>

#define LINK "link␣"
#define ENABLE "enable␣"
#define SIMPHY "simphy␣"
/**
```

```
 * @Author Suleiman H. Jama
 *
 * @version NEMAN GUI 1.0
 *
 * This class is the superclass for emulation and provides the base
 * for either GUI based emulation or command line based emulation.
 * Both GuiControll and Command classes inherit from this class.
 Documentation of this class is found in http://folk.uio.no/suleimaj/master
 */
class Common : public QObject
{
    Q_OBJECT
        public:

    Common(QObject *parent);
    Common();

    QStringList readIniFile();// returns server addr, portnumber and messageport
    void enableLinks(int startNode, int numNodes);
    void resetLinks(int startNode, int numNodes);
    void disableLinks(int startNode, int numNodes);
    void sendMessage(int startingNode);
    void createNewServerIni();

    QUdpSocket *topomanSock;
    QVector <double> time, msgTime, posX, posY, speed, destTime, myVector;
    QVector <int> destNode1, destNode2, destReachable, nodeNr, node1, node2,
    reachable, msgId, msgPort;
    QVector <QString> message, tapToIp, tapFromIp;

    QString serverAddr;
    int startNode, repeat, serverPort,messagePort, routingPort;
    int ms, numNodes,iPos, jPos , msgIndex, sliderValue;
    double cutoffTime, maxTime, timerLength, timerVal, displayTime, pause,
     max_speed,
    scenMax_x, scenMax_y, posXorY, comRange, brRange;
    bool startRunning, timeOrDest, stoped,isPause, isScenFile;
    QTimer *timer;
    QTime *elapsedTime;
    QSettings settings;

};
#endif
```

A.3 GuiControll.h

```
#ifndef PARSER_H
#define PARSER_H
#include "Common.h"
#include "GuiInit.h"
#include "Node.h"

#define LINK "link "
#define ENABLE "enable "
#define SIMPHY "simphy "

using namespace std;
class Common;
class enhancedGui;
class Node;
/**
 * @Author Suleiman H. Jama
 *
 * @version NEMAN GUI 1.0
 *
 *
 * All GUI related computation are done in this class.
 * This class provides the parsing of the scenario files
 * Creates node objects which is then put on the scene.
 * Further all events are controlled from this class.
 Documentation of this class is found in http://folk.uio.no/suleimaj/master
 */
class GuiControll : public Common
{
    Q_OBJECT
        public:
    GuiControll(QGraphicsScene *scene);
    ~GuiControll();
    QGraphicsScene *scene;
    void prepareNodes();
    void setReachable();
    void moveCurrent();
    void finishEmulation();
    void clearScenario();
    void simphyStatusfromTopoman();

    double speeds[11];
    QMap<int, Node *> mapNodes;
    QString fileName;
    bool prepareCalled, started;
    bool doNotSend;
    typedef QPair<int , int> Link;
    typedef QMap<Link, QGraphicsLineItem*> LinkMap;
    LinkMap links;

    public slots:
        void open_scenfile();
        void prepare();
        void startEmulation();
        void manageLinks();
        void timeincr();
        void sliderIncr();
        void showTime();
        void physicalLayerSim();
        void hideOrShowRanges();
        void hideOrShowLinks();
        void fitIn();
        void pauseEmulation();
        void stopAndLoopEmulation();
        void stopRestartEmulation();
        void changeIcon();
        void processPendingDatagrams();
        void routingOn();
        void routingOff();
};

#endif //PARSER_H
```

A.4 Command.h

```
#ifndef COMMAND_H
#define COMMAND_H
#include "Common.h"
#include <iomanip>
class Common;
/**
 * @Author Suleiman H. Jama
 *
 * @version NEMAN GUI 1.0
 *
 * This class provides the necessary functions for
 * a simulation without GUI but command line driven.
  Documentation of this class is found in http://folk.uio.no/suleimaj/master
 */
class Command : public Common
{
    Q_OBJECT
        public:

    Command( QStringList list);
    void open_scenfile(QString fileName);

 private:

    void startEmulation();
    void sendLinkUpdates();//same as manageLinks in Scen_Parser
    void finishEmulation();
    void physicalLayerSim();
    void routingOn();
    void routingOff();
    int progressIndex, startNode, repeat, serverPort,messagePort, physC,
    physG, physR, speedArg, progress, dotValue;

    QString serverAddr, routing;
    QStringList list;
    bool loop;

    private slots:
        void timerIncr();

};
#endif
```

A.5 Node.h

```
#ifndef NODE_H
#define NODE_H

#include "GuiInit.h"
#include <QGraphicsTextItem>
#include <QGraphicsItemAnimation>
#include <QTimeLine>
#include <iostream>
```

```
#include <QGraphicsLineItem>
#include <math.h>
class enhancedGui;
/**
 * @Author Suleiman H. Jama
 *
 * @version NEMAN GUI 1.0
 *
 * This class provides all properties for a node to be animated.
 * Documentation of this class is found in http://folk.uio.no/suleimaj/master
 */
class Node : public QObject
{
    Q_OBJECT
        public:
    Node(QObject *parent, QGraphicsScene *scene, int nodeId,
    double dataRange,double brRange, double scenX, double scenY);
    ~Node();
    QGraphicsLineItem *drawComLine(Node *neighbour, int status);

    void moveNodes(qreal deltaX, qreal deltaY, double speed);
    void moveNodesRestart();
    void undrawRange();
    void drawRange();
    void showLinks();
    void hideLinks();
    void changeLinkColor(int status, QGraphicsLineItem *line);
    QGraphicsEllipseItem *comRange;
    QGraphicsEllipseItem *broadcastRange;
    QGraphicsScene *scene;
    QGraphicsPixmapItem *item;
    QGraphicsLineItem *com_line;
    QGraphicsLineItem *br_line;
    QTimeLine *moveTimeLine;
    QTimeLine *timeLine;
    Node *neighbour;
    QMap<QString, QString> iconMap;
    QList<Node *> neighbourList;
    QList<QGraphicsLineItem *> com_line_list;
    qreal lastposX, lastposY;
    double lastspeed;
    double loc_x;
    double loc_y;
    double speed, animTime;
    int nodeId;
    int timeIncr;
 private:

    QGraphicsItemAnimation *anim;

    public slots:
        void updateComlinks();

};

#endif
```

A.6 Preference.h

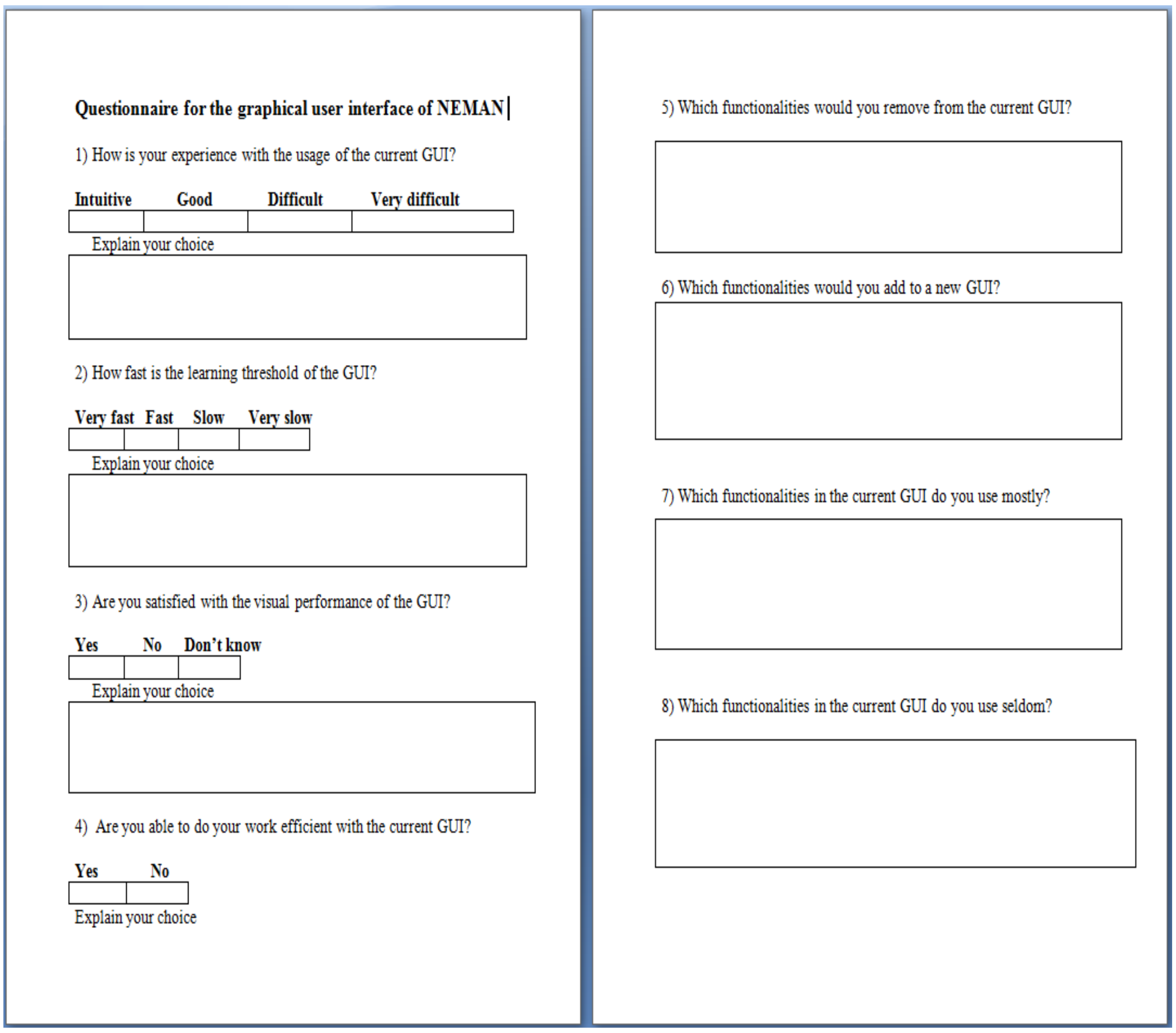
```
#ifndef PREFERENCE__H
#define PREFERENCE__H
#include <QObject>
#include "ui_preferenceWidget.h"
#include "Common.h"

class Common;
/**
 * @Author Suleiman H. Jama
 *
 * @version NEMAN GUI 1.0
 *
 * This class sets up the dialog window for serverAddr/port
   messagePort preferences.
   Documentation of this class is found in http://folk.uio.no/suleimaj/master
 */
class Preference : public QDialog, public Ui::preference
{
    Q_OBJECT
        public:
    Preference();

    public slots:
        void writeToIniFile();
};

#endif
```

Appendix B

Questionnaires

B.1 User Experience Questionnaire

B.2 End User Questionnaire


Questionnaire for the graphical user interface of NEMAN

1) How is your experience with the usage of the current GUI?

Intuitive	Good	Difficult	Very difficult

Explain your choice

2) How fast is the learning threshold of the GUI?

Very fast	Fast	Slow	Very slow

Explain your choice

3) Are you satisfied with the visual performance of the GUI?

Yes	No	Don't know

Explain your choice

4) Are you able to do your work efficient with the current GUI?

Yes	No

Explain your choice

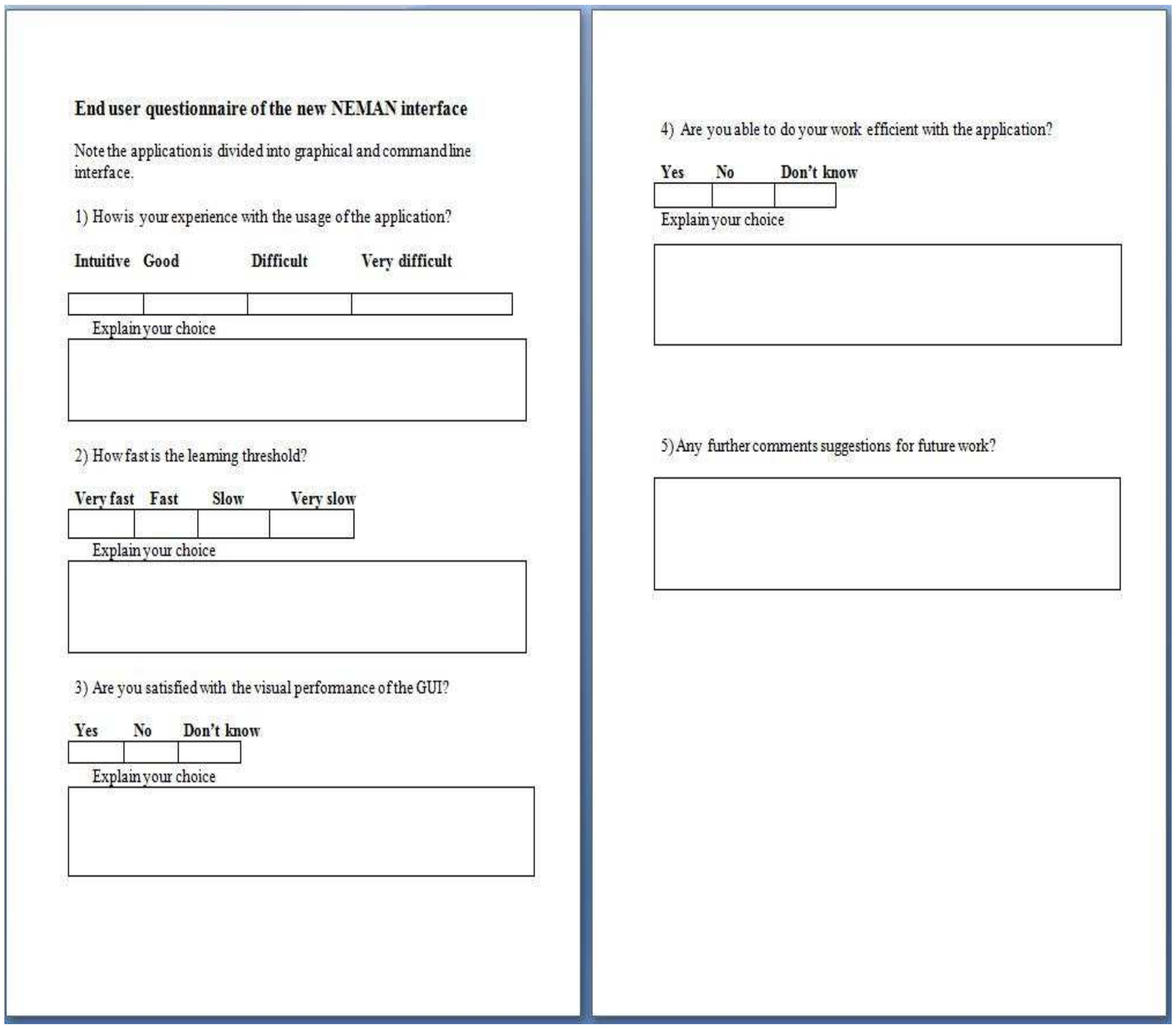
5) Which functionalities would you remove from the current GUI?

6) Which functionalities would you add to a new GUI?

7) Which functionalities in the current GUI do you use mostly?

8) Which functionalities in the current GUI do you use seldom?

Figure B.1: User questionnaire on NEMAN Interface 0.5



End user questionnaire of the new NEMAN interface

Note the application is divided into graphical and commandline interface.

1) How is your experience with the usage of the application?

Intuitive	**Good**	**Difficult**	**Very difficult**

Explain your choice

2) How fast is the learning threshold?

Very fast	**Fast**	**Slow**	**Very slow**

Explain your choice

3) Are you satisfied with the visual performance of the GUI?

Yes	**No**	**Don't know**

Explain your choice

4) Are you able to do your work efficient with the application?

Yes	**No**	**Don't know**

Explain your choice

5) Any further comments suggestions for future work?

Figure B.2: End user questionnaire on NEMAN Interface 1.0

Bibliography

[1] http://searchvb.techtarget.com/sDefinition/0,290660,sid8_gci213989,00.html. last visited: May. 2007.

[2] Gui definition. http://www.bellevuelinux.org/gui.html. last visited: May. 2006.

[3] Nam network animator for: Ns-2. http://www.isi.edu/nsnam/nam. last visited: July. 2006.

[4] The network simulator: Ns-2. http://www.isi.edu/nsnam/ns. last visited: July. 2006.

[5] Omnet++ discrete event simulation system. http://www.omnetpp.org. last visited: July. 2007.

[6] Tcl built-in commands - clock manual page. http://www.tcl.tk/man/tcl8.4/TclCmd/clock.htm#M5. last visited: June. 2007.

[7] Einar Bjerve. Transparent gateways between olsr networks, master thesis. University of Oslo, department of Informatics, 8th May 2006.

[8] Matthias Kalle Dalheimer. Qt vs. java a comparison of qt and java for largescale, industrialstrength gui development. Klarälvdalens Datakonsult AB kalle@klaralvdalens-datakonsult.se.

[9] "Ovidiu Valentin Drugan, Thomas Plagemann, and Ellen Munthe-Kaas". "building resource aware middleware services over manet for rescue and emergency applications". University of Oslo, Department of Informatics, P.O. Box 1080, 0316 OSlo, Norway. "citeseer.ist.psu.edu/732585.html".

[10] "Erek Göktürk, Matija Pužar, and M. Naci Akkøk". "distributing NEMAN network emulator using MICA component architecture". In "Fernando Barros, Claudia Frydman, Norbert Giambiasi, and Bernard Zeigler", editors, *"Proceedings of the AI, Simulation and Planning in High Autonomy Systems (AIS), and Conceptual Modeling and Simulation (CMS) Conference (AIS-CMS 2007) (co-located with the International Modeling and Simulation Multiconference (IMSM 2007))"*, pages "199–205". SCS, "February" "2007".

[11] Weirong Jiang and Chao Zhang. A portable real-time emulator for testing multi-radio manets. In *IPDPS*, 2006.

[12] Jan Erik Johnsen. Improving the physical and mac layer models of neman, 24th May 2006.

[13] Matthias Kropff, Tronje Krop, Matthias Hollick, Parag S. Mogre, and Ralf Steinmetz. A Survey on RealWorld and Emulation Testbeds for Mobile Ad hoc Networks. In *Proceedings of 2nd IEEE International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2006)*, March 2006.

[14] Matthias Kropff, Tronje Krop, Matthias Hollick, Parag S. Mogre, and Ralf Steinmetz. Living document a survey of real-world and emulation testbed for mobile ad-hoc networks. Multimedia Communications lab (KOM) Technische Universitat Darmstadt, February 7, 2006.

[15] Leszek A. Maciaszek. *Requirement Analysis and System Design.* Addison Wesley; 1st edition (April 13, 2001), 2001.

[16] D. Mahrenholz and S. Ivanov. Daniel mahrenholz and svilen ivanov. real-time network emulation with ns-2. In Proceedings of DS-RT'04, Budapest, Hungary, October 2004., 2004. citeseer.ist.psu.edu/mahrenholz04realtime.html.

[17] Aaron Marcus, Nick Smilonich, and Lynne Thompson. The cross-gui handbook for multiple user interface design. Addison-Wesley, 1995.

[18] Lutz Prechelt. An empirical comparison of seven programming languages. volume 33, pages 23–29, Los Alamitos CA USA, 2000. IEEE Computer Society.

[19] Matija Puzar. Neman readme file, in the neman archive. last visited: May 2007.

[20] Matija Puzar and Thomas Plagemann. NEMAN: a network emulator for mobile ad-hoc networks. Telecommunications, 2005. ConTEL 2005. Proceedings of the 8th International Conference on Volume 1, June 15-17, 2005 Page(s):155 - 161, 2005.

[21] Trolltech. The qgraphicsview framework. http://doc.trolltech.com/4.2/qgraphicsview.html. last visited: July. 2007.

[22] Cristian Tuduce and Thomas Grosst. A mobility model based on wlan traces and its validation. In INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communication Societies. pages 664-674, Mar. 2005.

[23] Yongguang Zhang and Wei Li. An integrated environment for testing mobile ad-hoc networks. pages 104–111, 2002. http://doi.acm.org/10.1145/513800.513813.

[24] J. Adrian Zimmer. *Tcl/Tk for Programmers, with Solved Exercises that Work with Unix and Windows.* pub-IEEE, pub-IEEE:adr, "1998".