Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2018

5G Simulation Framework

Junior Asante and Joel Olsson

Master of Science Thesis in Electrical Engineering

5G Simulation Framework

Junior Asante and Joel Olsson

LiTH-ISY-EX--18/5149--SE

Supervisor: **Ema Becirovic**
ISY, Linköping University
Henrik André-Jönsson
Ericsson AB

Examiner: **Danyo Danev**
ISY, Linköping University

Division of Communication Systems
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

From the first generation, 1G, to the fourth generation, 4G, the development and technological advancements in telecommunications network systems have been remarkable. Faster and better connections have opened up for new markets, ideas and possibilities, to that extent that there now is a demand that surpasses the supply. Despite all these advancements made in the mobile communications field most of the concept of how the technology works and its infrastructure has remained the same. This however, is about to change with the introduction of the fifth generation (5G) mobile communication.

With the introduction of 5G much of the technology introduced will be different from that of previous generations. This change extends to include the entire infrastructure of the mobile communications system. With these major changes, many of the tools available today for telecommunications network evaluation do not really suffice to include the 5G network standard. For this reason, there is a need to develop a new kind of tool that will be able to include the changes brought by this new network standard.

In this thesis a simulation framework adapted for the next generation telecommunication standard 5G is set to be developed. This framework should include many of the characteristics that set 5G aside from previous generations.

Acknowledgments

We want to thank Danyo Danev for his help during our thesis. We also want to thank Ema Becirovic for her diligence and eye for detail during the many iterations of our thesis. We also want to thank our supervisors Henrik André-Jönsson and Pontus Sandberg for their enthusiasm and great feedback throughout our thesis.

Linköping, June 2018
Junior Asante and Joel Olsson

Contents

List of Figures

List of Tables

Notation

Abbreviations

Abbreviation	Meaning
1G	First-Generation
2G	Second-Generation
3G	Third-Generation
4G	Fourth-Generation
5G	Fifth-Generation
AI	Artificial Intelligence
API	Application Programming Interface
BTS	Base Transceiver Stations
BSC	Base Station Controller
CDF	Cumulative Distribution Function
CLI	Common Language Infrastructure
CN	Core Network
D2D	Device-to-Device
DESMO-J	Discrete-Event Simulation Modelling in Java
ENB	eNodeB
ENAML	Enaml is Not A Markup Language
FXML	F-Extensible Markup Language
FDD	Frequency Division Duplex
GUI	Graphical User Interface
ICT	Information and Communications Technology
IDE	Integrated Development Environment
IOT	Internet of Things
ISY	Department of Electrical Engineering
JSL	Java Simulation Library
JVM	Java Virtual Machine

Abbreviations

Abbreviation	Meaning
LTE	Long-Term Evolution
MIMO	Multiple-Input Multiple-Output
MMWAVE	Millimeter wave
MTC	Machine Type Communication
MVC	Model-View-Controller
NB	NodeB
NR	Node-Red
NSS	Network Simulator System
OMNET++	Objective Modular Network Testbed in C++
OOP	Object-Oriented Programming
OSI	Open Systems Interconnection
PDF	Probability Density Function
QML	Qt Modelling Language
RAN	Radio Access Network
RNC	Radio Network Controller
RTS	Real-Time Systems
SSJ	Stochastic Simulation in Java
TCP	Transmission Control Protocol
TDD	Time Division Duplex
UDP	User Datagram Protocol
UE	User Equipment
UE4	Unreal Engine 4
XML	Extensible Markup Language

1

Introduction

With the first-generation (1G) mobile communication network came the possibilities to make the first mobile phone calls. Then came the second-generation (2G) mobile network that enabled text messaging services like SMS and MMS. After that the third-generation (3G) mobile network introduced light weight video streaming and enabled video calling to the public. The fourth-generation (4G) mobile network then brought us high definition video streaming services and is, at the time of writing, the current standard. (Qualcomm, 2014)

The next step is the fifth-generation (5G) mobile network which is expected to push the limits of mobile communication. Alongside the introduction of this new network technology are many new features to address the shortcomings of today's mobile communication solution. These shortcomings have emerged as a result of our changing habits and increased demand on wireless mobile communication. (Nordrum, 2017)

Until now the idea of the cellular infrastructure has been to build few, but powerful, cell towers to provide cellular coverage over large areas. This approach has worked well for the comparably limited connected mobile devices and their intended use. However, providing sufficient coverage and ensuring stable connections with low latencies are some of the many challenges facing the infrastructure today. Emergence of these challenges are due to several reasons:

- A dramatic increase of connected mobile devices.
- A demand for higher bandwidths to support more data transfer.
- A constant demand for higher connections speeds.
- A demand for reduced latency of connections.
- Metropolitan areas getting more densely populated.

(Nordrum, 2017)

1.1 The 5G network

5G is the first telecommunication standard to use frequencies in the 24-86 GHz range (Medbo and Zaidi, no date). With the use of higher unused frequencies of the radio spectrum will introduce problems were signals have difficulties penetrating obstacles. These obstacles can be buildings trees or even the atmosphere. (André-Jönsson, 2018)

To solve this a new approach of having several smaller cellular towers is being implemented, together with advanced technologies like beamforming and massive Multiple-Input Multiple-Output (MIMO). These technologies combined are expected to increase efficiency of the cellular towers and enable higher bandwidths, faster connections and lower latencies to the vast number of devices being connected. (IMT Vision, 2015)

In Figure 1.1, a depiction of the differences between previous technologies and 5G technology can be seen.

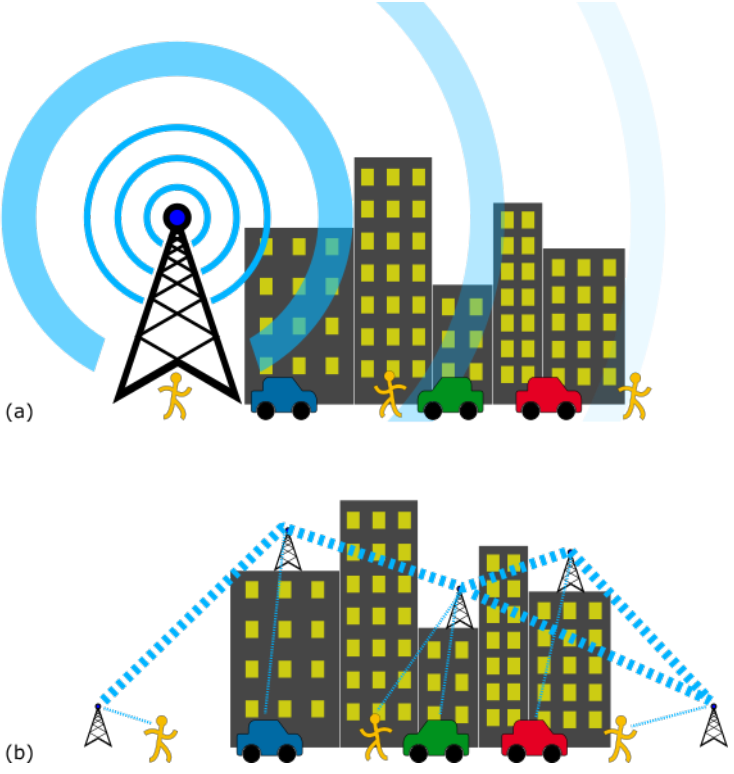


Figure 1.1: *A picture of (a) the current standard of mobile communication infrastructure, with few but powerful cell towers and (b) the future infrastructure with the deployment of the 5G network. Source: Junior Asante*

Along with the expanded features that will be available for mobile communication, machine-type communication (MTC) is one of the new utilities that is developed with 5G. MTC demands, depending on the function, a different set of requirements than conventional mobile communication. Some of the functions and applications that MTC provide are for:

- Metering of for example electric power or gas.
- Control systems and monitoring like real-time systems (RTS) for industries and homes.
- Security systems like surveillance.

Metering devices will be large in numbers and will meter for example, electric power in homes and businesses. This will require high coverage to support even the most remote devices. Meanwhile industry and surveillance might have to meet RTS requirements such as low latency, high reliability and high mobility.

The new technology enables us to use ad-hoc and point-to-point networks. Traditionally there are several drawbacks identified for using these types of networks for MTC, but with the separation of MTC into several different types, these drawbacks might not be as severe, or it might be that the possible pros outweigh the cons (André-Jönsson, 2018). There is however a major drawback of using these types of networks; they use unlicensed frequencies that are more prone to interference. Therefore, the links are more unreliable compared to licensed frequencies. (Shariatmadari et al., 2015)

From previously using the mobile network, mainly for voice calls, (1G, 2G) the trend has now shifted towards data transfer (3G, 4G) and, with that, a mixture of different demands and requirements. Examples of these requirements can come from:

- Consumers that demand high bandwidth with reliable connections for multimedia streaming services.
- Industries that may depend on high bandwidth, reliable connection and low latency for their business.
- Internet of things (IoT) and MTCs, with their ever-increasing numbers, that require everything from low bandwidth, latency insensitive connections to high bandwidth, highly reliable connections.

Planning an infrastructure to meet all these demands and avoid compromises is not an easy task for the world's telecom companies. Because of this having the right tools for the development of a mobile communication infrastructure is essential.

1.2 Motivation

It is not a new phenomenon to use modern computers as aid, not only the planning and development but also the testing of new technologies. A computer simulations ability to rapidly calculate the results of different parameters make them invaluable to many developers, engineers and scientists. They help provide vital data about the environment being simulated, thus forming the base of many important design decisions. There are many reasons for a simulator to be utilized before taking any major design decisions. The most prominent of them being cost savings. Design flaws can be a costly mistake when it comes to establishing a complex system. Anything that can help reduce the risk of incorrect estimations is therefore highly valued.

Of course, simulation software for data networks already exist. However, with the increased demands and the development of 5G, many fail to capture all the important aspects of modern network planning. Because of that, a new kind of simulator is needed.
Our task is to build such a simulator, for trying to understand the impact on the radio access network (RAN) when consequences like dynamic back haul links or multi path protocols are introduced. How does this affect the RAN services and identified RAN traffic types and how does it impact RAN robustness and reliability. It should also aim to show how different networks behave given different parameters like data load, data distribution, costs and more.

1.2.1 What is needed in a new simulator?

One of the major aspects that needs to be considered today is the importance of differentiating between different types of data. For instance, some network users may require low latency as primary priority for their network. By using a connected RTS with time critical data, they might depend on fast response times. Others, on the other hand, might prioritize cost savings over latency. For these consumers, having data that eventually reaches its destination, regardless of latency, will be satisfactory. Furthermore, while some users may require a very large chunk of data every now and then, like downloading a movie or a music album for offline consumption. Others will require a constant stream of data, valuing a reliable connection over bandwidth. Providing a high speed, high bandwidth, low latency and super stable connection to everyone would be an ideal solution to ensure total satisfaction. Due to costs, however, this is not a feasible solution. Therefore, to meet these demands and at the same time, keep the costs down, a preferred solution is to provide just the services required to meet each individual demand. The difficult task is to figure out what kind of hardware is needed where to ensure that the demands are met while at the same time being the most cost-effective solution.

1.3 Purpose

The shift in mobile communication from a few powerful cell towers to many smaller cell towers requires a different approach when planning the network infrastructure. Increasingly more complex networks will be created with a multitude of nodes and connections between these nodes. Without the proper tools it will be hard, if not impossible, to predict the behavior of this kind of network, due to its complexity.

The lower latencies and higher throughput in the radio access network might allow an operator to use techniques like self back-hauling and multi path deliveries to improve robustness and reliability for certain types of traffic.
(André-Jönsson, 2018)

In this thesis a simulator for data flow through a network will be implemented and a focus area will be to analyze different frameworks that can be used to realize this. Besides building the core that will be responsible for the simulation calculations, the surrounding environment needs to be built too. This includes units like the graphical user interface and other supporting functionality. Because of this the simulator and its entire solution will here on be referred to as the Network Simulator System (NSS).

A simulator like the NSS will be especially important when planning the establishment of networks in for instance major capital cities or other densely populated regions. In these regions the requirement for the number of cell towers, or "nodes", as they can be referred to, will be the highest, creating the most complex networks. A simulation of such a network would then be able to provide important data of key factors. These key factors could be anything from load distribution and data latency to cost estimation and much more.

1.4 Problem Statements

The NSS will be aimed to be used by engineers at Ericsson and should therefore fit the purpose and requirements stated by these engineers. One important aspect of this is to make it adaptable to their current model of systems. Building a system of this scale and for this purpose presents a lot of challenges and raises questions that need to be answered. One important preparational task is to identify some of the more extensive and important challenges and questions to focus on. The identified questions are as follows:

- How can the complex model of data traffic load in a sophisticated large-scale network be depicted in such a way that it is easy to comprehend?
- How can such a complex system be designed in a detailed enough level to be useful for its purpose while still preventing excessive use of hardware resources?
- What are some of the tools that can be used to ease the development of such a system and what design choices should be taken?
- What are the most important features that need to be implemented into the NSS for it to be useful for its intended use?

1.5 Limitations

During development of any system it is important not to get side tracked, spending too much time overdeveloping less important parts. This is particularly true if the time spent doing that is at the cost of the development of main core functionality. To help keep this focus during advancement some restrictions and limitations will be applied.

One restriction will be the graphical aspects of the NSS. The main feature of a simulation software will be to provide important and relevant data to the user in an easy to understand manner. For this, less can sometimes be more, especially when it comes to the presentation of data. Therefore, little effort will be put on the aesthetics of the software. The graphics that will be provided will, at best, be on a basic level and is mainly intended to represent data in comprehendible form. This means no elaborated animations or special effects will be added. This restriction comes with the added benefit of reduced constraints on computational resources, which is favorable, as simulations often are quite computationally heavy to begin with.

The next major restriction that will be applied will be the finding and implementing of the algorithms used to route data in individual nodes. For this project, only basic calculations will be implemented. The user will instead be given the option to decide whether to implement their own preferred algorithms or use the built-in. This enables simulation scenarios that can be modified after the user's preferences, allowing for countless simulations possibilities.

Another major restriction imposed will be that focus will mainly be on the lower levels (Media Layer) of the Open Systems Interconnection model (OSI model). It is assumed that building a simulator that can simulate the loads of network nodes under different network settings will not require the level of detail that the higher levels of OSI model would provide. This means that simulations will only regard the flow of data in physical networks rather than on the actual data routing. This compromise will help further reduce the load on hardware resources and improve simulation performance. See Table 1.1 for a depiction of the OSI model.

OSI Model			
Layer		Protocol data unit	Function
Host Layer	7 Application	Data	High-level APIs
	6 Presentation		Translation of data
	5 Session		Communication sessions
	4 Transport	Transmission Control Protocol (TCP) / User Datagram Protocol (UDP)	Transmission of data
Media Layer	3 Network	Packet	Multi-node network managing
	2 Data links	Frame	Data frame transmission between frames
	1 Physical	Bit	Transmission of raw bit streams over physical medium

Table 1.1: *Table of the OSI model showing all the different layers of network communication. Source: (OSI model, no date)*

2

Background

Ericsson is a provider of information and communications technology (ICT) to service providers around the world. With an estimated 40% of the worlds mobile traffic carried through their networks, it is safe to say that they are a major player when it comes to mobile communication networks (Ericsson, no date). With the development of the next generation mobile communications standard, 5G, companies like Ericsson need to update and develop new tools adapted for this new technology. This is the foundation for the two theses that together make up this entire thesis.

This thesis is based on two separate theses combined into one. The first thesis focuses on a framework that simulates network activity and data flow through a network. The framework should be able to handle parameters like latency, throughput limitation, capacity, storage and network nodes in a meshed network to base the simulations on. The purpose of the framework is to give visual representation of these simulations. The second thesis focuses on expanding the NSS with a more sophisticated traffic load generator. This load generator should be user specified and based on some mathematical models that represent load data usage on real networks.

3

Theory

3.1 Introduction

In Velten, 2009 the word simulation is defined as following: “Simulation is the application of a model with the objective to derive strategies that help solve a problem or answer a question pertaining to a system”. This definition goes very well with the intended use and purpose of the resulting software of this project. The theory will be divided into the following sections:

- Cellular Network.
- 5G Network Layout.
- Simulation Framework.
- Statistical Distributions.
- Programming Languages.
- Game Engines.
- Software Platforms.
- System Architecture.

3.2 Cellular Network

As a theoretical introduction to this thesis, some basic concepts in cellular communication will be described (see Figure 3.1 for a basic depiction). The main topics that are going to be described are:

- Core Network.
- User equipment.
- Radio Access Network.

3.2.1 Core Network

A core network (CN) has the purpose to provide the user with the means to send and receive telephone calls and data. As a part of the internet backbone the CN provides paths for data exchange between sub nodes. The CN is made up of a series of routers and switches with the latter being the most common. (Core Network - Definition from Techopedia, no date; Faulkner and Harmer, 1999)

3.2.2 User Equipment

A user equipment (UE) is what the end-user uses to communicate with the cellular network. A UE can either be a computer, mobile phone or any device that has a cellular antenna. (Kasera and Narang, 2004)

3.2.3 Radio Access Network

The Radio Access Network (RAN) has the radio functionality of the network. In 2G the RAN was comprised of two main components; Base Transceiver Stations (BTS) and Base Station Controller (BSC). To ensure good coverage, multiple BTS must often be used to cover an area. Each BTS has a specific area that the antennas cover called cell site. The names macro, micro and pico are used to describe the size of area covered by the cell sites. (Cellular Fundamentals – Radio Access Network (RAN), 2014)

In 3G the RAN structure was changed to NodeB (NB) and Radio Network Controller (RNC). The NB corresponds to the BTS in the 2G RAN and the RNC correspond to the BSC. The instructions that the NB execute are provided by the RNC. (Kasera and Narang, 2004)

In 4G the structure was changed quite significantly. The NB was exchanged to the eNodeB (eNB) and the NC's functions was splitted, where some functions was moved into the 4G CN and some functions was moved to the eNB. The result is that the 4G RAN only consist of interconnected eNBs. Since the eNBs that are interconnected together, mobility is handled directly between eNBs. There

are also features taking advantage of this interconnection, e.g. Elastic RAN that enables several eNBs to send data to the same UE to achieve higher throughput. (Khan, 2009; Ericsson Elastic RAN, 2016)

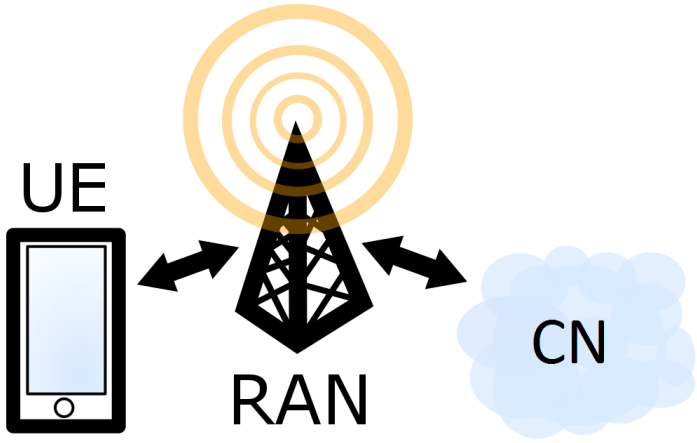


Figure 3.1: *Basic depiction of the interconnection between UE, RAN and the CN. Source: Junior Asante*

3.3 5G Network Layout

In the 5G of telecommunication, old as well as new concepts are used to meet the demands of mobile broadband and MTC. One of the new concepts that will be developed in the 5G is a new type of dynamic system which will adapt to the network demand. For example, an area might need high data load and low latency, therefore this area will be targeted with more resources to meet these demands. Some of the main concepts that will enable this type of functionalities are:

- Device-to-device (D2D) communication.
- A more flexible Frequency Division Duplex (FDD) and Time Division Duplex (TDD).
- Massive Multiple-Input Multiple-Output (MIMO).

(Ericsson, 2016)

D2D communication is available in 4G but will have extended functionality when it comes to 5G. For example, D2D in 5G makes it possible to share data directly between devices to use these devices to extend coverage for other users.

This can lead to reduced load on the NB/eNB but also increase the efficiency of the network. (Tseng et al., 2017)

To avoid interference in areas where the mobile broadband traffic is dense, the available spectrum must be used efficiently. FDD and TDD are techniques that have been used in Long-Term Evolution (LTE) for duplex communication. (Yun et al., 2007)

In 5G, FDD will be used as the major technique to transmit and receive data on the lower frequency bands. TDD will have an important role in dense deployments and in high frequencies (above 10 GHz). TDD will also be implemented differently in 5G compared to previous standards (TDD LTE). Uplink and downlink have restrictions on how they can be configured in TDD LTE. In 5G, TDD will be able to dynamically assign transmission resources and be more flexible than previous implementations. (Ericsson, 2016)

Massive MIMO was proposed in (Marzetta, 2010) and has since been adopted to be one of the core features of 5G. However, massive MIMO must be modified to utilize higher frequencies that will be used in 5G. These frequencies are called millimeter wave (mmWave) and range above 24 GHz (Medbo and Zaidi, no date).

Since 5G will use higher frequencies than previous technologies, the wavelength will be shorter, therefore the antennas will be smaller. Each 5G RAN can, because of this, be equipped with hundreds more antennas than in previous technologies (Swindlehurst et al., 2014; Van Chien, Björnson and Larsson, 2018).

In 5G, the receivers and transmitters will use beamforming to increase the signal strength and with that, either reduce the interference or increase the range between different transmissions (André-Jönsson, 2018). Beamforming also makes it possible to, more effectively, target regions which are subjected to poor coverage due to limitations in previous technologies (Ericsson, 2016).

3.4 Simulation Framework

The two major parts in a simulation framework is the model and the simulator. The model is the depiction of a system or a process that exists while the simulator is the actuator of the framework. The simulator takes a set of input data through the model. The model then produces an output using the input data. (Topçu et al., no date)

3.4.1 Model

A model of a system is a simplified depiction of said system that can provide enough relevant information to answer a given question. According to Velten, 2009 everybody uses mathematical models of some sort, even if unaware of it. An example is for instance a car. Most people have a simplified model of how a

car works. Knowing you must turn the key to turn on the car, shift in the right gear, press the gas pedal to move forward and turn the steering wheel to turn the car. In this simplified model none of the complexity of the engine e.g. pistons, oil, generator, sparkplugs etc. is counted for. The knowledge of these are simply not needed for the intended purpose of the model. This model is sufficient until the car stops or breaks down for an unknown reason. When this happens, the model needs to be updated with more complexity to sufficiently represent the system. Does the car have enough fuel? Or is the battery charged? When creating a model for a system it is recommended to start with smallest possible model with the lowest level of complexity of the system and add complexity as it is needed. (Velten, 2009)

Models can have various meaning and levels of complexity. The goal is not to create a perfect copy of reality, but rather to create a sufficient enough model for the problem that the model is trying to present. Arguably the best model is the least complex one still able to present answers to problems. By taking the essentials and basic concepts of the given system, parts of the system will be described in the model and parts will be left out. This is also called abstract. (Banks, Nelson and Nicol, no date; Topçu et al., no date; Siegfried, 2014)

3.4.2 Simulator

The simulator is the unit that takes a set of inputs through the given model and produces an output. These computations can be done in a continuous time or a discrete-event manner. The prior case is often described with a set of differential equations, and the latter makes computation on the model at discrete timestamps. When using continuous time computations, the state of the system changes continuously with time which is different to discrete-event computations where the state of the system only changes at discrete points in time. (Banks, Nelson and Nicol, 2005)

In (Velten, 2009) the term simulation for a discrete event simulation is described as "the imitation of operation of a real-world process of a system over time".

Simulation Model

When describing a model of a system it can either be described physically or mathematically. The mathematical model of a system is a set of equations that describe the system theoretically. A simulation model is a sub-category to mathematical models. Simulation models can further be divided into a set of subcategories when describing how the input data will be processed:

- Static or dynamic.
- Deterministic or stochastic.

- Discrete or continuous.

(Banks, Nelson and Nicol, no date)

Simulation on static models respond to different inputs while the model of this system maintains a specific state from a certain time instant. Meanwhile, a dynamic model will change over time and will not keep the same state during the simulation. A Monte Carlo simulation is an example of a static simulation, while the number of people in a food store is a dynamic simulation model. Simulation models that are deterministic have no random variable affecting the system. Every input is known and will produce an output. A stochastic simulation model on the other hand, will have one or more inputs that will be randomly determined before being used in the model. (Banks, Nelson and Nicol, no date)

Continuous simulation model uses a continuous function to simulate the given model, see Figure 3.2. The model will change the output continuously.

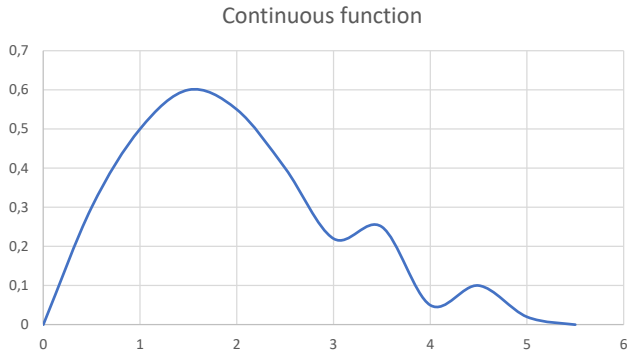


Figure 3.2: *Example of how the mathematical representation of a continuous simulation model. Source: Joel Olsson*

A discrete model of a system has time discrete instants, where a new computation is made for the system i.e. the state of the system will only change when a new time instant arrives. See Figure 3.3 for an example of a discrete system.

3.5 Statistical Distributions

A powerful tool for simulations is the use of mathematical models, more precisely, statistical distributions, or probability distributions. Probability distributions can for instance be used to describe different scenarios with fairly high accuracy.

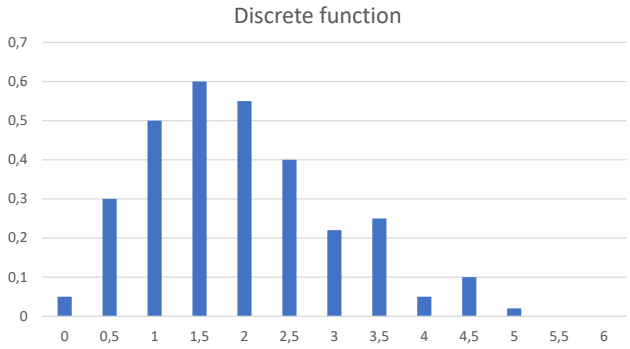


Figure 3.3: *Example of how the mathematical representation of a discrete model of a system could be defined. Source: Joel Olsson*

These mathematical models attempt to describe different types of distributions across a spectrum, like time. In probability distributions the probability of the next outcome is usually more interesting than which single output is most likely. The better these distributions can describe these scenarios the more accurate the simulation will be. However, having accurate mathematical models usually means having to count for a lot of variables or unknowns, which in turn increases the complexity of the calculations. This will in the end lead to longer calculation times and increased demand for resources (Velten, 2009).

3.5.1 Exponential Distribution

The exponential distribution is a probabilistic mathematical model that can be used to describe the time elapsed between two separate events. This can be anything from when the next customer may enter a shop to when the next phone call will occur to a call center. (Exponential distribution, no date)

The probability distribution of the stochastic variable X can be described as:

$$f_x(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if} \quad x < 0 \\ 0, & \text{if} \quad x \leq 0 \end{cases} \tag{3.1}$$

(Forbes, 2011)

3.5.2 Uniform Distribution

Uniform distribution or, continuous uniform distribution is a distribution model where the probability of getting a sample is constant within the sample space (the probability density function (PDF) is constant). (Forbes, 2011)

The continuous uniform distribution is described in Equation 3.2, where $F(x)$ is the cumulative distribution function (CDF).

$$F(x) = \begin{cases} 0, & \text{if} \quad x \leq a \\ \dfrac{x-a}{b-a}, & \text{if} \quad a < x < b \\ 1, & \text{if} \quad x \geq b \end{cases} \tag{3.2}$$

(Forbes, 2011)

3.6 Programming Languages and Libraries

A study of programming languages, different libraries and software platforms have been conducted to find a suitable language for this thesis. The approach used to examine these languages was to look at two factors. Which simulation libraries exist and what tools could be used to develop a Graphical User Interface (GUI). Following this chapter will be a chapter about different libraries available for the different programming languages.

3.6.1 Programming Languages

The main programming languages that have been studied are C/C++, C#, Java and Python. A small description of each of the languages will be given in this section.

C/C++

C++ Is a strongly typed language that compiles directly to a machines native language. Being strongly typed means that it is expected of the programmer to know what he or she is doing. The compilation to native machine language makes it one of the fastest languages available, if optimized. Being built of C language it is compatible with C code, with a few exceptions and can use all C libraries with little to no modification at all. (Albatross, no date)

Some available simulation libraries in C/C++, amongst many are:

- Sim.
- Advanced simulation library.
- Objective Modular Network Testbed in C++ (OMNet++).

(Faraz Fallahi, 2014)

Java

Java is another popular programming language that is platform independent due to it being run from Java Virtual Machine (JVM). JVM interprets the executed java program for the native operating system meaning that any machine with JVM installed can run Java programs. (Java Essentials, Part 1, no date)

Java is a typical high-level object-oriented programming (OOP) language which is designed to be simple to learn and use, if the concept of OOP is understood. It is also considered to be more dynamic than for instance C/C++ as it is designed to adapt to an evolving environment. (Java Overview, no date)

C#

C# is a high-level, general-purpose, OOP language developed by Microsoft and can be seen as a combination of Java and C/C++. The language is very much based on C/C++ programming language but with a strong resemblance with Java. It is designed for Common Language Infra-structure (CLI), which means it has a runtime environment that the executable code is run on. This allows the use of various high-level languages to be run on different computer platforms and architectures. (C# Overview, no date)

Python

Python is yet another high-level interpreted programming language with efficient high-level data structures. Having a simple syntax, dynamic typing and being interpreted makes it versatile language for many areas and platforms. The interpreter can be extended with new functions and data types implemented in other languages like C/C++ which further increases its versatility. (The Python Tutorial - Python 3.6.5 documentation, no date)

3.6.2 Potential Libraries

Writing any extensive software can be a daunting task which is why the use of libraries is common practice. The library support for a programming language is an important factor which is why most languages market the amount of library support that exist for that language. Many libraries however, are so popular that they exist for several different languages which is why libraries are placed in its own chapter. In this chapter some interesting libraries will be brought up and information about some of their key abilities will be presented. The libraries that will be brought up are:

- Sim.
- Advanced Simulation.

- Crazy Eddie's GUI System.
- DESMO-J
- SSJ.
- JavaFX.
- SimPy.
- Enaml.

Sim

Sim is a C++ library for discrete event simulation. It supports event driven and process driven simulation. It will execute and calculate a new output at a certain point in time and will during that time change the state of the system. (Bolier-Anton and Ens, 1994)

Advanced Simulation

Advanced simulation library is an open source library platform for simulating and solving partial differential equations. The computations are made in OpenCL, with a C++ Application Programming Interface (API) available to use for implementation of math expressions. ('Advanced Simulation Library Expanding software ecosystem for the DSP/FPGA/GPU market', 2015)

Crazy Eddie's GUI System

Crazy Eddie's GUI (CEGUI) is a library mainly for creating a graphical interface. It provides windowing and widgets for graphics APIs and engines where functionality of these are missing. The library is object oriented and written in C++ and is targeted at game developers. (Paul D Turner, no date)

DESMO-J

DESMO-J stands for Discrete-Event Simulation and Modelling in Java and is written in Java to model systems in a discrete-event fashion. Amongst other features, DESMO-J is equipped with features such as stochastic distributions, scheduler, simulation clock and report generation. DESMO-J is used to develop simulation systems both in academia and in the industry. (DESMO-J, no date) (van Staden, 2012)

SSJ

SSJ is a simulation library which is organized in sets of packages to aid simulation programming in the Java language. SSJ extends the Java language with tools

to help implement complex models. A simulation project can be divided into several tasks ranging from modeling to implementation. Support is also added for implementation and validation of simulation. (SSJ: SSJ User's Guide., no date)

JavaFX

JavaFX is one of the available graphical tools and libraries that complements Java. A feature of JavaFX is the use of F-Extensible Markup Language (FXML) to build an interface. With this support, it is possible to separate the back-end of the system from the front-end. The graphics can be built by coding FXML or using Scene Builder, which is a drag-and-drop tool for building a GUI. (What Is JavaFX? | JavaFX 2.0 Tutorials and Documentation, 2013)

SimPy

SimPy is one of the simulation libraries for Python based on a discrete-event simulation framework. The library has the functionality to run a simulation by manually stepping each simulation step or by executing the simulation as fast as possible (Team SimPY, 2016). SimPy is used in different domains to build a discrete event simulator to for example, model a supply chain for a forest (Pinho, Coelho and Boaventura-Cunha, 2016).

Enaml

One of the many existing graphical libraries for Python is Enaml. With Enaml it is possible to create a user interface on platforms that can run Python and Qt (Welcome to Enaml - Enaml 0.10.2 documentation, no date).

3.6.3 Platforms and Frameworks

This section will bring up different frameworks and platforms that are of interest. Platform and frameworks are a slightly higher form of abstraction than individual languages and libraries and usually provide generic functionality. (Rich Moy, 2017)

OMNet++

OMNeT++ is a popular network simulation platform in the scientific community and in the industrial settings. It provides a component architecture for models that can be programmed and assembled together. OMNeT++ offers the following components:

- Simulation kernel library.
- NED topology description language.
- OMNeT++ IDE based on the Eclipse platform.
- GUI for simulation execution, links into simulation executable (Tkenv).
- Command-line user interface for simulation execution (Cmdenv).
- Utilities (makefile creation tool, etc.).
- Documentation, sample simulations, etc.

(Heidemann and Isi, 2002)

Qt

Qt is a cross-platform development framework. It comes with its own IDE called Qt Creator and like many other IDEs, it offers intelligent code completion, syntax highlighting, debugging capabilities and more. Qt abstracts GUI programming by offering widget modules, which enables GUIs to be written directly in C++. Another option for GUI creation offered by Qt is with the QtQuick module which enables GUIs to be written in Qt Modeling Language (QML). QML integrates JavaScript and is a declarative object description language. (About Qt - Qt Wiki, no date)

3.6.4 Game Engines

Another approach for developing a simulator-like system is to use a game engine. Game engines are software solutions created to help ease the development of multimedia and cross platform content, primarily video games. With a main feature of providing abstraction from certain daunting tasks, usually platform dependent details of commonalities, these software suits help reduce development overhead. Some of the many commonalities a game engine could help abstract includes physics, graphics, input and computer Artificial Intelligence (AI). With less time spent on adapting such tasks, more time can be spent on the intended software and its unique features. (Game engine, 2018)

There are many different game engines available, like: CryEngine, REDengine Cocos2D-X and so on. Many of them focuses on different areas and are based on different programming languages (List of game engines, 2018). For this research only two engines where focused on as they, at the time, seemed to be the two most popular, according to multiple ranking sites visited like (10 Best Video Game Engines (Rankings & Reviews), 2018).

Unity

Unity is one of the most popular 3rd party game engines, especially for mobile games, on the market. It is estimated that 34% of the top 1000 free games are

made in unity and that unity is somehow included in over 770 million games around the world (Unity - Fast Facts, no date).

Unity comes with an extensible editor with a vast range of included features and functions. Some of these features are:

- Physics engines to take advantage of Boc2D and Nvidia PhysX support.
- Ability to use Timeline, Anima2D, Particles for animation.
- Support for both 2D and 3D development.
- In-editor creation tools to allow prototype, 2D level design and support for 3rd party design tools.
- Access to unity Asset store where off-the-shelf content can be either bought or downloaded for free.
- Advanced profiling tools to determine if your software is CPU or GPU-bound.
- Platform support on over 25 different platforms including Android, iOS, Linux, Windows, XBOX, Gear VR and more.

The programming language in Unity is based on C# and Unity is free to use for beginners, students and hobbyists but range from $35 per month to $125 per month for serious and professional creators (Unity - Products, no date).

Unreal Engine

Unreal Engine is developed by Epic Games, which also happens to be responsible for popular game titles like Gears of War, Unreal Tournament and Fortnite (Epic Games | About, no date). Currently, Unreal Engine 4 (UE4) is the latest release and there are a few similarities between UE4 and Unity. Just like Unity, UE4 comes with a broad variety of features and tools in the domain of graphics, sound, networking, optimization etc. to help ease development. Accompanying these tools is the extensive documentation found on their site as well as the support community available online. UE4 also features an e-commerce platform known as Marketplace which is equivalent to Unity's Asset store. Here developers can get both purchasable and free, game-ready content and code (Epic Games, no date).

The most notable difference between Unity and UE4 is the programming language. UE4 is a C++ coding environment and offers a feature called Blueprint that allows for a node-based scripting method that can be used for developing games and other software without even having to write any code. Being a high-end set of tools UE4 does require a mid-to-high-tier machine to work properly due to its resource demands. The platform support for UE4 is also quite extensive, including Windows, Mac, Linux, XBoxOne, PS4, HTML5, iOS and more.

UE4 is free of charge given the condition that a released software made with UE4 makes less than $3000 per calendar quarter. After that Epic requires a royalty of 5% of the gross revenue. (Epic Games, no date)

3.7 Software Platforms

Using complete systems or software platforms was also an idea that seemed viable. Many simulation platforms available can be tweaked or modified to fit the intended use. How extensive these modifications need to be is of course dependent on the platform and its intended use. Out of the available platforms on the market, two were chosen as an example for the sake of comparison. These were MATLAB's Simulink and IBM's Node-Red.

3.7.1 MATLAB/Simulink

Simulink® is a model and simulation software environment integrated with MATLAB. It uses block diagram for model-based design and multidomain simulations. Some of the supported features are:

- System-level design.
- Simulation.
- Automatic code generation.
- And continuous test and verification of embedded systems.

Thanks to the integration, MATLAB functions and algorithms can be incorporated into models and results can be exported to MATLAB for further analysis. Simulink primarily simulate behavior of system components over time by keeping a clock and determining the order in which blocks are to be simulated. (Create Simple Model - MATLAB Simulink - MathWorks Nordic, no date)

3.7.2 Node-Red

Node-Red (NR) is an open source node-based tool for wiring together hardware devices, API's and online services to create so called "flows". Its browser-based editor makes it easy to create flows by connecting a wide range of nodes. Its built-in library also allows for templates, functions or flows to be re-used. Over 225,000 modules are included in Node's package repository which allows for a great deal of functionality expansion of the nodes. The capabilities of nodes can be expanded even further with the use of JavaScript code that can be coded directly within the editor. (Node-RED, no date)

3.8 System Architecture

When coding any complex project, it is recognized as good practice to organize the software structure in a certain way. An architecture of a system can be seen as

the foundation of which the system is built upon. A good software architecture takes into consideration:

- How the users interact with the software.
- How the application will be managed.
- What the required quality attributes for the application might be.
- How the application will be flexible and maintainable over time.
- What future trends that might affect the application.

The purpose of an architecture in a system is to reduce the business risks associated with complex technical builds. Taking in the above stated considerations a good architecture should bridge both the business requirements of a system with the technical requirements. Through the offering of good flexibility, future drifts in hardware and software technology, as well as user scenarios and requirements can be handled. (MSDN, 2014)

3.8.1 Architectural Patterns

An architectural pattern, sometimes referred to as an architectural style or architectural model, is a set of design principles that shape the framework that makes the system. It promotes the reuse of design by providing solution to problems that are frequently recurring. The key architectural styles can be seen in Table 3.1.

Architectural style	**Description**
Client/Server.	Separates the system in two, where clients make requests to servers.
Component based.	Splits application design to reusable functional or logical components with well-defined communication interfaces.
Domain Driven Design.	Object-oriented style for modeling a business domain and defining objects based on entities within the business domain.
Layered Architecture.	Application concerns are partitioned into stacked layers.
N-Tier/3-Tier.	Functionality is separated into segments with each segment being a tier located on a physically separated computer.
Object Oriented.	Division of responsibilities into reusable individual objects. These objects are also self-sufficient containing data relevant to the object.

Table 3.1: *Key architectural styles and a small description of them.*

More architectural patterns are available and it is not uncommon to combine several styles in a system. In fact, it is unusual not to combine several styles as the complexity and demands of the system grows. One might start adopting the Model-View-Controller (MVC) layer style model but can later be forced to combine it with the 3-tier approach due to security concerns. (Meier et al., 2009)

Client/Server

With the client/server architectural style the system separates the user (client) and the application (server). The client sends requests with specific tasks that the client wants executed, while the server makes the proper response to the client's requests. Some of the main pros of this architectural style are:

- The data security is often higher since all the used data is stored on a server, which in most cases is more secure than the client's.
- Data is centralized, which makes data access and updates available for clients easier to maintain and control.
- The administration of the system is controlled by updating the server system, which separates clients from maintenance and updates of the system's infrastructure.

Although the client/server comes with the benefit of being easy to maintain and control, the architectural style comes with some downsides. Some of the cons with a client/server architecture could be:

- Problems with scalability, since system logic and application data combined on a server system could cause problems with scaling the system.
- Reliability, the clients using the the system is dependent on the server working.

(Meier et al., 2009a)

Component based

A component based architectural style separates the system functions and system logic into multiple subtasks. This makes the functions reusable since the system functionality is defined into separate functions. Furthermore, this makes functions independent from each other since there is low dependency. Some of the pros with a component based architectural structure:

- The development of features can be done independently from each other since the functions have low dependency.
- Parts of the system can be reused since they are implemented separate from the other functionality of the system.

- Third-party functions can be implemented which can reduce the cost of the development.

Some of the drawback with a component based architectural style can be:

- If a new technology is implemented into an existing system, a component based architecture is of no use since all components are outdated.
- Problems with migration to other system and compatibility with other platforms.

(Meier et al., 2009a; Sharma, Kumar and Agarwal, 2015)

Domain Driven Design

In a domain driven design, the architecture of the software will be adapted to the business that it is supposed to be used in. To implement this type of architecture, the developers need good knowledge of the business or have resources that have expertise in the particular domain. With a domain driven design, some of the benefits are:

- The same technical language can be used in the whole development team since the domain specifies the language.
- The system is easy to update and adapt, since the domain model is often flexible.

With a domain driven design architecture some of drawbacks could be:

- The design could become costly since the architecture forces the design to be specific to the domain.
- Should only be used in systems that can benefit on the domain's complexity.

(Meier et al., 2009a)

Layered Architecture

In a layered architecture the system is built with multiple layers where each layer have a specific functionality. The system can communicate between the different layers but it must be done explicitly. Some of the pros of using a layered architecture:

- Since the layers are isolated from each other, changes in individual layers has minor effect on the rest of the system.

- Testing can be done to individual layers, which also makes it possible to have multiple implementations that can be exchanged and tested in the system.
- Reusability of for example the controller in a MVC structure where it could be used to different views.

The layered architecture comes with some drawbacks:

- It can be hard to define the correct layer to implement certain functionality.
- Since the layers are separated from the rest of the system, the implementations could be generic and can lead to low flexibility.

(Meier et al., 2009a; Sharma, Kumar and Agarwal, 2015)

N-Tier/3-Tier

Similar to the layered architecture is n-tier/3-tier, where the layers are instead tiers. The difference is that in the tier architecture each tier run on a separate computer. Some of the pros of tier architecture are that:

- It can easily be maintained since the system is divided into separate tiers that can be changed independently.
- The system can be scaled on each tier separate from the other tiers in the system.

Some of the cons with tier architecture are:

- Since different parts of the system can be run on different machines, the system could be dependent on remote servers.
- Run time of the system could be dependent on the network traffic if the different tiers are at remote locations.

(Meier et al., 2009a; Sharma, Kumar and Agarwal, 2015)

Object Oriented

In the object oriented architecture the main focus is to divide the system in to separate objects. Each object is working independently from the other objects in the system. The objects exchange data with each other by calling functions or to access attributes in the particular object. Some of the pros with an object oriented architecture:

- Objects are easily reused.

- The encapsulation of objects enables testing of individual objects.
- The system can get a high level of consistency by maintaining related functions and attributes connected to the objects.

Some of the cons with a object oriented architecture:

- It could be inefficient where limitations on the hardware is of concern.
- Low maintenance of the code base could lead to unnecessary code existing in the system.

(Pros and Cons of Object Oriented Programming | Green Garage, no date; Meier et al., 2009a)

4

Method

4.1 Introduction

In this chapter the method used to develop the NSS will be described.

4.2 Prestudy

When investigating the different programming languages/platforms to develop the NSS on, the main points that were under consideration were the following:

- Backward compatibility and interoperability.
- Flexibility.
- Portability.
- User interaction.
- Licensing terms and agreements.

Our assessment of the previous points made us list the most important parts that the programming languages/platforms must be capable of handling to ease the implementation of the NSS. Our choice of programming language/platform was based on these points:

- It should be equipped with a simulation library/framework.
- GUI development should be abstracted to reduce the workload required to design a simple GUI.
- Portability.

- If any new skills are needed, the learning curve for that skill should not be too steep (previous knowledge in the programming language/platform is favorable).

Based on these requirements, the choices of preferred languages/platform were narrowed down to four main choices and the selection process of these will be further discussed next. In Table 4.1, pros and cons are displayed for some of the different software that was included in the prestudy.

Language/Software package	Pros	Cons
C/C++.	Experience of language, large community.	Lower level language than other languages. No GUI experience.
Python.	High-level language, previous experience in programming and GUI, big community.	Less experience than other languages, like Java.
Java.	Previous experience in programming and GUI, large community and high-level.	Less extensive simulator framework than for instance C++.
Unity.	High-level, big community, suite includes lots of helpful tools for abstraction.	No previous experience in C#, Licensing terms, costs money for commercial use.
UE4.	Big community, extensive set of tools for many abstractions.	Licensing terms, costs money for commercial use.

Table 4.1: *Pros and cons for the studied software languages/platforms.*

4.3 Choice of Language

After the prestudy four main options were prioritized as the most viable. These were Unity with C#, Unreal Engine with C++, Java and Python. The reason for the selection of the game engines (Unity and UE4) was due to the complete set of tools they provide and the extensive support community that exists around them. Both Unity and UE4 were perceived as excessive for the task of building a simulator since none of the advanced features like 3D graphics and the physics engines were expected to be utilized. Nevertheless, the point of selecting them was to ease the development of the GUI part of the NSS. Since the GUI part was one of the greater concerns of the system, any tools that would help reduce the

burden were preferred. However, after a discussion with our supervisors it was concluded that Unity and UE4 would be excluded due to the licensing terms.

Having narrowed down the choices further to either Python, with the SimPy framework, or Java, with the DESMO-J framework, Java was eventually chosen. Before this decision was made a small test program in both Python and Java was written to try them both out. This was to see how easy it was to get started with either of the frameworks and get a feel for their extensiveness.

Alongside this, another important aspect we looked for was the documentation for the frameworks. Both SimPy and DESMO-J came with simulation examples, good documentation and source codes. They were both easy to set up and we managed to have working simulations within a day or two by following examples given by the documentation. In Figure 4.1 a test simulation with console output in SimPy is shown.

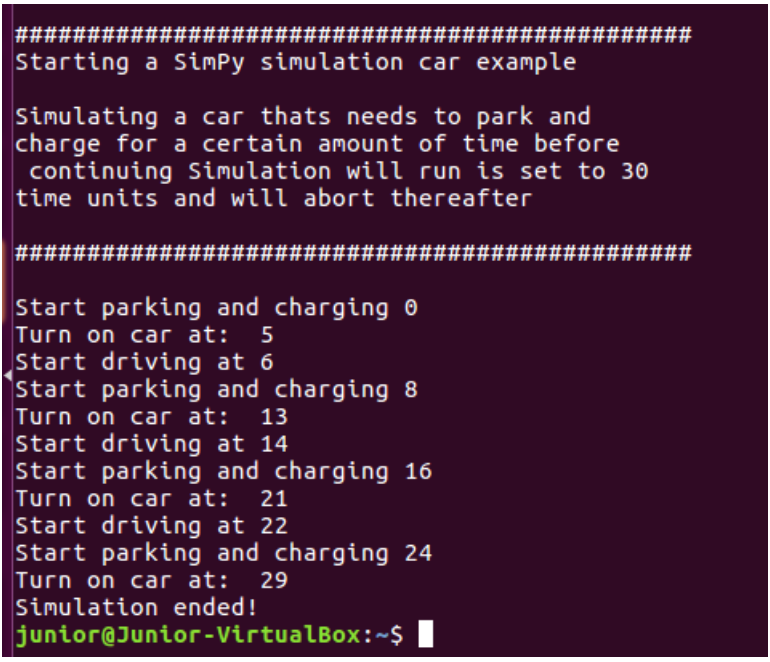


Figure 4.1: *The output result of the simulation run of the test program written with SimPy. Source: Junior Asante*

Because of difficulties distinguishing any major differences that might impact on future development, we eventually decided both SimPy and DESMO-J to be equally suitable for this thesis. The choice then landed on Java on the basis that we felt that we had more experience in Java than we did in Python.

4.4 Resources

While conducting this thesis, different resources were available to our disposal. Since this thesis was conducted at Ericsson most of the resources available to us were provided by the company.

4.4.1 Hardware

The hardware given to us for this thesis was an HP EliteBook 820 G3 laptop, each, with the following specs:

- Intel core i5-6300U processor @ 2,4 GHz – 3GHz.
- 4GB of RAM.
- Windows 7 64bit OS.
- 250 GB of disk space.

As can be seen from the specifications, this is far from the most high-end hardware available on the market. Nonetheless we had this laptop as a limiting factor for the NSS which would force us to be conservative with hardware resources.

4.4.2 Software

Due to security and licensing reasons, not all software packages are suitable to be used for Ericsson work. This was important to take into consideration when choosing a software package, like an Integrated Development Environment (IDE). Many of the widely available "free" software on the internet usually comes with conditions concerning the use of that software for commercial use. Licensing concerns were, for instance one of the main reasons Java was chosen over other development alternatives, like Unity and Unreal Engine.

To work with Java programming, the development environment chosen was Eclipse IDE. Eclipse is a free complete suite popular for Java development as well as development of other languages. Eclipse was downloaded from the Eclipse foundation page together with some extensions added to the IDE.

To ease the development of the graphical interface, an addon to JavaFX was used called Scene Builder. In Scene Builder a GUI can be built with a drag-and-drop interface that creates the core graphical elements used in the NSS. Scene Builder then creates an XML-file that can be used to control what should be displayed in the different containers and menus etc. Figure 4.2 shows the interface of Scene Builder.

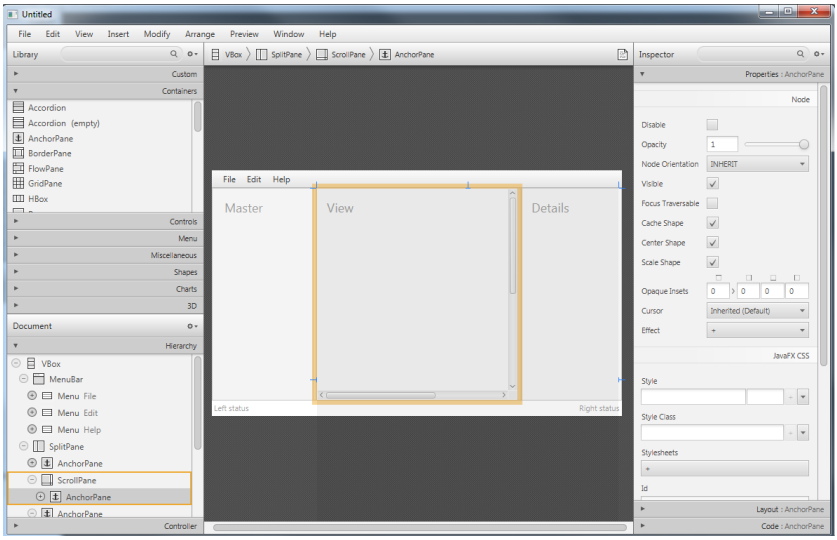


Figure 4.2: *Scene Builder interface where each graphical element can be edited and placed. Source: Joel Olsson*

4.4.3 Experienced Staff

At the site where this thesis was conducted there are around 800 people of varying expertise and background working together to develop products for Ericsson. This is an immense source of knowledge and we were encouraged to gather all the information we needed from these people.

4.5 Choice of Architecture

As mentioned before there are a few architectural styles to choose from and regardless of choice, the resulting architecture is expected to be a combination of several styles. Despite this, having an architectural pattern to start with is a good idea to maintain consistency across the system. To meet the requirements of flexibility and portability, the main idea was to separate the graphical part of the system from the simulation core. These were identified as the largest separate segments of the NSS.

This division of the NSS might resemble the client server style architecture. However, because the segmentation will be within the same application rather than split on different applications it is more of a layered architecture style. Under this style the Model, View, Controller (MVC) architectural pattern seemed like a suitable pattern for the NSS. The MVC model is not an unfamiliar model to us and because of our previous experience of using this architectural style it was chosen as the architectural pattern to base the NSS on. Further explanation

of the MVC model will be given next.

4.5.1 MVC Model

The initial point of model used when coding the NSS was the MVC architectural model. The idea with this architectural model is to have well separated interconnecting parts. This will in turn allow for modifications on one part independent from any other.

Model

The model in MVC consists of the main functionality of the application. Here lies the actual simulation core and all the source code used to run and perform the actual simulation. The DESMO-J framework that is used for the NSS resides mostly in this part of MVC. The model part of MVC can be run without the need of the view part because of the structure of the application. This also means that certain changes and modifications can be done to the core functionality of the application without having to restructure the view part. In Figure 4.3 is a depiction of the MVC model.

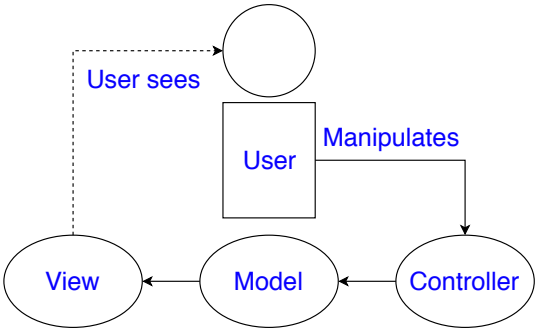


Figure 4.3: *MVC model that illustrates the user interaction with the application. Source Junior Asante*

View

The view part includes everything that relates to the Graphical User Interface (GUI). This includes all the code that describes everything from window size to button positions and all the visual effects. The GUI is written independently from the model and is connected to the model through the exchange of different key variables. This means that the view can be run without the model part and vice versa.

Controller

The controller works in-between the model and the view and communicates with both. The purpose of the controller is to act as a translator between the model and the view. Having this part further increases the independence of the model and view parts.

4.6 System Structure

After deciding on the system architecture, the next step was to decide how the structure of the NSS should be composed. It was clear from the beginning that the NSS would be separated between GUI and the actual simulation core. However, now with the MVC model architecture in mind the idea was to segregate the NSS into three main layers. For the view we had GUI, for the model we had the simulation core and for the controller, everything that should process the user input. This is essentially the link between the GUI and simulation core. Although a bit loose and unspecific, these decisions should help when it comes to decide where to place different Java classes and parts of the program.

Using an IDE, like Eclipse, helped greatly with the structure planning and modifications of the structure, thanks to its included set of tools. However in spite the use of an IDE and an architectural model to follow, the difficulties of separating certain instances while at the same time keeping the consistency of the NSS were realized early in the development. This was recognized when doubts of where to place new components and functions arose due to dependencies to multiple parts of the NSS. Further explanation will follow under 4.9 Major Design Changes.

4.7 Simulator Design

Little was decided about the NSS beforehand, apart from its intended use. Even its functions were more of a concept with a lot of features being desired rather than demanded. This meant countless possible approaches for how the NSS could be constructed and how it could end up as a finished product. Regardless, there still needed to be a plan on how to work on this project and so a first step was to divide the work between us.

4.7.1 Division of work

Being two people working on this thesis we needed to divide the work to be able to work independently without risking our work overlapping. The division was simple, one person starts working on the GUI to get familiar with it and the other

person starts developing the simulator core.

4.7.2 The GUI

The intention was to create the GUI using the same kind of layered style as the rest of the system. This was expected to result in an increased flexibility for the GUI. With this design pattern, the visual aspect of the graphics can be changed while the functionality remains the same.

4.7.3 Simulation Core

The design of the NSS was quite dynamic with several improvements and changes. It was therefore far from clear how the end product would look like or even behave beforehand. Not knowing every detail of the simulator to build poses some challenges in the design phase, as much is unclear in the beginning and every decision taken is followed by some uncertainty over the end result.

The simulation core was derived from the test simulation program constructed during the pre-study phase. Having a simple simulator to start from, the work progressed by adding more complex features to it.

4.8 Adding Features

Shortly after the development of the NSS started a list of requested features and functionality was received from our supervisor. This list can be viewed in Appendix B. The list was used as criteria for what the NSS end product should contain as a finished product. However, at first, not too much attention was given to the list since our focus was aimed at developing the GUI and the core of the simulation, meaning the class packages that allowed for an actual simulation to be conducted. Both the GUI and the core were essential for the NSS to even be considered as a useable simulator and due to their complexity, they required a great deal of development hours. Once these were developed to the point that the user interface could be used to initiate a simulation the work of implementing features from the previously stated feature list could begin.

The requested features were not sorted in any kind of prioritized order, but it was still possible to conclude that some features were more important or useful to have than others. Also noticeable was the fact that some features depended on others to be useful. For instance, the usefulness of being able to plot the results of a load on a link is greatly reduced if the simulated load created by the nodes are fixed and cannot be changed.

4.8.1 Implicit Features

Aside from the requested features, there were some missing functionality that was realized to be, at least as important as the assumed important features. This included functionality like: being able to reset a simulation without having to open a new instance of the NSS and being able to have more than one data generating node in the same network. Because of the limited time resources, the risk of not being able to implement all features was quite evident and so a prioritization of features was needed to decide what to focus on. The resulting list was:

- Enable reset of simulation.
- Thread simulation to avoid GUI freeze during simulation.
- Enable data flows to be added to nodes.
- Enable multiple data generating nodes in the same network.
- Add import and export capabilities.
- Allow for graphical visualization of simulation progress.
- Add name tags to nodes.

Despite the complexity of most of the above listed features they were picked as they were believed to add the most value to the NSS, in terms of user experience and simulation usefulness. The implementation challenges varied considerably between features and a more in-depth explanation will be given in the following chapter.

4.8.2 Complex Features

While some features could be implemented by merely adding another parameter and connect it to a method, others required quite a lot more work to function. The more complex features, to mention a few were:

- The ability to add data flows.
- The ability to use different algorithms to decide how a node should behave.
- The ability to have multiple data generation nodes on the same network.

Because these were all challenging features that required alteration of the simulation core to work, implementing these features meant a great deal of changes to the state of the NSS. When implemented, these changes usually ended up breaking other parts of the system, adding even more work to the workload. The process of adding a complex feature, like any of the ones stated above, usually started with figuring out how to implement the feature with minimal changes to the NSS core. After that the necessary changes were made. Lastly, tests were done to try and find broken parts and if any were found, they were repaired. During extreme cases some of these features required an entire rewrite of large portions of the NSS and these will be further addressed in the next chapter.

4.9 Major Design Changes

Throughout the development of the NSS, countless changes and updates of both the functionality and graphics have been made. The cause for these changes can be anything ranging from optimization of different features to the need to adapt parts for features to work with other, newly developed or updated, functions. Further down the most important changes made to the NSS will be brought up and described more.

4.9.1 System Structure Changes

The structure of the NSS was in the beginning built with the MVC model in mind. This created a layered style structure that was separated into the following main packages: `system_large_simulation_core`, `view`, `application` and `graphics_node_handler`. Each of these layers handle multiple functionalities that can be tied to that specific layer. Within each layer all source files could be found associated with that layer. The problem that was realized with this structure was that individual features of the NSS were harder to isolate and modify. This reduced the flexibility of the system considerably and we realized that a different structure was needed.

Upon some further investigation a page was found that suggested the "package by feature" approach over the "package by layer" that was currently implemented. Essentially this meant that each feature of the NSS should be in its own package for easy separation and modification. This approach sounded much better than the layer approach, specifically for larger projects that needs to maintain flexibility. The structure was therefore changed across the entire system and the previous layers were further split into several packages of features. Our aim was not to abandon the MVC layer style but rather extend it and use it as a basis for further improvement with added flexibility. This meant that anything still associated with GUI would still reside within the view category and anything associated with the simulation core, in the model category and so on. In Figure 4.4 the two different approaches can be seen.

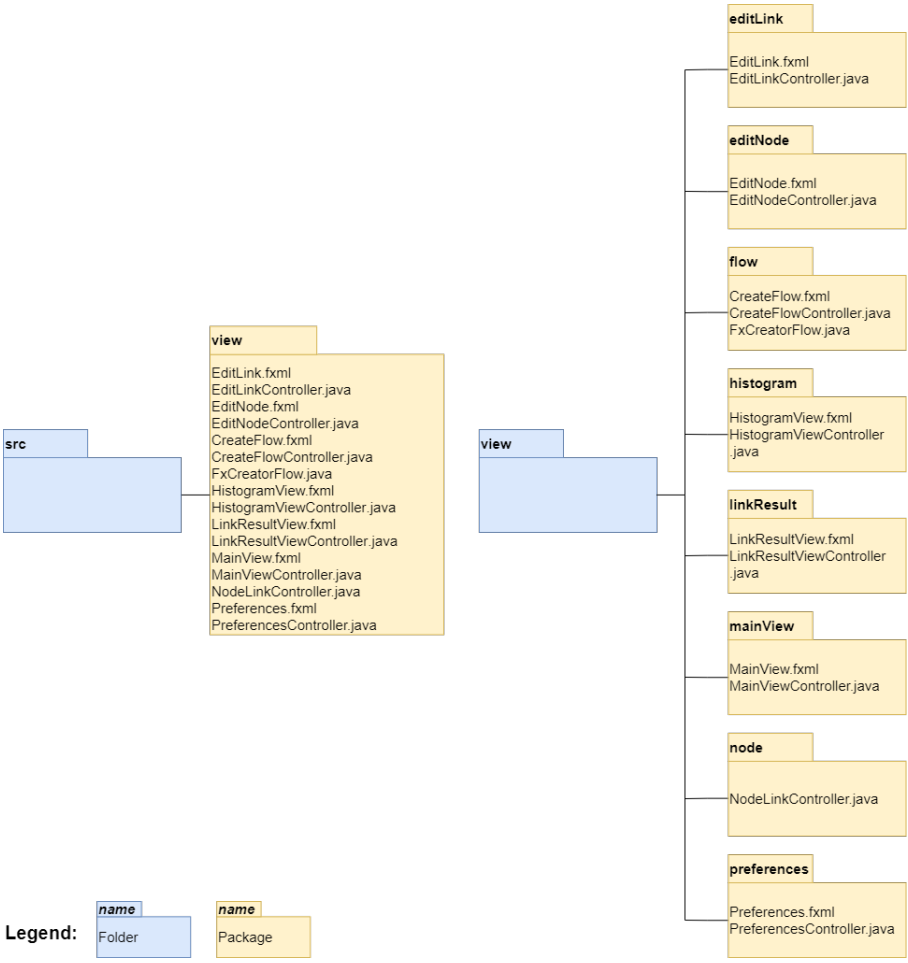


Figure 4.4: *Shows an example of the early, layer style structure (to the left) of the NSS project that was later changed to a "package by feature" style structure (to the right). Source: Junior Asante*

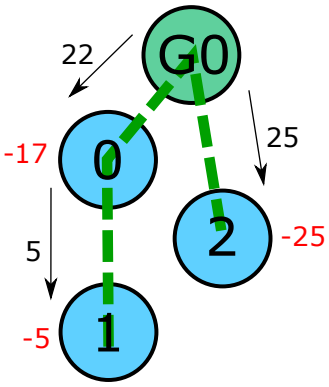
4.9.2 Feature Modifications

Because of the loose descriptions given to most features, the creation and implementation of a new feature started off as realization of a vague idea and was often modified several times during the development of the NSS. Many times, the modifications were minor and mostly needed to adapt the feature to the current developed stage of the NSS. However, the data load feature and the find path feature have been modified so extensively that they can be considered as rewritten.

The data load feature is responsible for simulating the load on the network. This is one of the most important features of the NSS and also one of the most challenging parts to get right due to the many variables to account for.

The first implementation of the data load feature comprised of a global request list and data load objects. A node created a request for data by randomizing a load value using a suitable distribution model. This value was then added to the global load list. A special data generation node, that represented a fixed connection to the CN, produced as many data objects as the sum of requests on the global list. These objects were then, with the help of a path list, sent down the path required to reach each node. Each node in the path received its own amount of requested data objects and the objects for all descending nodes. When data objects were received the node would subtract the amount of data it requested by destroying an equal number of objects, remove its request, add its latency to the data packet objects and send the remaining objects to the next node in the chain. A depiction of how this implementation of the data load was handled can be seen in Figure 4.5.

For this method to work the entire network and how it was connected needed to be known beforehand. A strict path was chosen for each simulation run, meaning paths could not be changed for the entire simulation and only one generator node was supported.



Global List

Node	Req	Load	Gen
N0	17	22	-
N1	5	5	-
N2	25	25	-
G0	-	-	47

Paths

G0	G0
N0	N2
N1	

Figure 4.5: *Shows how data load is handled. A node requests data that is added to the global data list. A generator node (G0) then creates the total amount of data objects that is requested during that time instant and uses the path list to distribute the objects. Each node then extracts its amount of data object and passes down the remaining objects. Source: Junior Asante.*

This approach worked well for smaller networks (< 10 nodes) and helped push the development of the NSS further by being able to simulate simple networks. Once the network size increased, meaning an increase of nodes in the network, the amount of data objects needed to be created increased too. This could lead to considerable overhead on calculations for large networks.

Due to the limited main memory of our hardware it was difficult to estimate the true impact of the NSS during runtime. This was because the main memory of four gigabyte was more or less fully utilized (> 90%) even before a simulation run was initiated. Nevertheless, the likelihood of a scalability problem convinced us that the load feature had to be redone. This was somewhat expected from the beginning but the reason for its development was its simplicity and that we needed something to simulate load early on in the development.

The second approach to the load generating feature was to create a load object that represented the total load of a node. This meant that only one load object was created per node rather than the requested amount. The load amount was then instead a value held within the load object. Another major difference was that the load object now was created by the requesting node and passed on to the previous node in the path until the data generating node was reached. Even for this approach, a path was needed for the node to know where to pass on the data object. Each node that received an object added the load value of the load object to an accumulated load variable to show the total load on that current node.

This approach worked well but had two major flaws. There was no way to distinguish between different types of data loads and the entire load of a node was forced to take one path to reach its destination. Because of this another redo of the load feature was required to support multiple load streams with different priorities.

The third and current iteration of the load feature was created to add multiple stream and priority functionality as this was one of the requested requirements. These issues were handled by adding methods and some rewriting of the second iteration. Each data load was created to be an object, which created a separation between each load. This could then be utilized to observe each data load instead of the accumulated load on a node or link.

Along with this, each data load was integrated to be part of a data flow object. The data flow extends the functionality of the data load. Instead of individual data loads loading nodes and links in the system, each data load is part of a flow that contains data loads. With this approach, the observed load in the system switched from arbitrary values to data flow objects. Furthermore, the data flows added the functionality for the user to extended changes to the data loads created within each data flow.

4.9.3 Find Path Feature

One of the limitations stated for this thesis was that no data routing algorithms would be required to be implemented in the NSS. Instead that was left to the user to implement. However, having no routing algorithm at all meant that it would be impossible to test and verify that the NSS would work as intended. Because of this a simple form of routing was needed. The simplest one we could come up with was having a predetermined path the NSS could follow to route data. This lead to the development of a "find path" feature, that brute force all paths by iterating through the entire network. During this iteration it finds every possible path from every node in the network to a data generating node. For each node and link in the path the corresponding latency was added. This produced a multi-dimensional array list with all nodes and their every possible path to a data generating node. This list could then be sorted after number of hops (nodes in path), nominal latency or any other key value. The list was then used to direct each node of where to send data load objects to eventually reach a data generating node.

This feature proved more useful than previously anticipated. It was only expected to be needed before each run of a simulation to create a map of the network. Each node therefore incorporated this feature within the node itself and was called only once during a simulation. However, with additional features being added, like dynamic path choices during simulation run. It was realized that this feature needed to be extended. It was therefore removed from the node info object and implemented as its own class.

5 Results

5.1 Introduction

The results that was achieved when developing the NSS will be presented in this chapter.

5.2 System Structure

The NSS is built on the following three main packages: `application`, `simulator` and `view`. The `application` package contains the main class that is used to start the NSS. The `simulator` package holds several subpackages that are divided by function. This package also contains the source code to the core simulator as well as the code for the majority of the features. The `view` package contains all the different graphical subsystems. Like the `simulator` package, the content inside the `view` package is also divided into several subpackages after feature.

The resulting structure have lead to further independence and readability in the NSS. Most classes are now packaged with dependent files and classes. Each package includes the functionality for one specific context, e.g. all the classes that control the node behavior and attributes is in one package and similarly for the links etc. Removing a feature is now a matter of deleting a package rather than having to track down all dependencies in the system.

5.3 Design Decisions

During the course of the development there have been some important design decisions. These have been taken due to factors like increased understanding of a particular problem or function, unwanted results from implementation or realization of a problem. Some of the more extensive decisions made will follow.

An example of a design decision was the single link decision. We decided to only allow one link between two nodes. This restriction meant that a link between two nodes only can have one set of attributes.

Another design decision was to reduce the system complexity to ease the transition of development to other developers. This lead to the structural changes of the system mentioned earlier (see 4.9.1 System Structure Changes). Another result of this decision was an increased amount of comments in the source code to help ease the understanding of the various classes and methods.

Yet another example of a design decision was that no cacheable nodes will be implemented. The ability to simulate cached data in a node to change behavior of a network was one of the requested features on the feature list. However, the complexity of such a feature was evident even at an early stage of this thesis. It was eventually decided that the time resources would not allow for a development of such a feature and so the feature was disregarded.

5.4 Application

The `application` package only consists of the NSS main file, which is the file where the program runs from when started.

5.5 Simulator

The `simulator` package is the package that contains all the packages that control the simulation in the NSS.

5.5.1 Core

The `core` package contains all the controllers for the NSS. It constructs the model of the NSS and works as a bridge from the core of the NSS to the graphics.

5.5.2 Stats

The `stats`-package handles all the sampling of the data during a simulation. The class the that executes this is `SimulatorContoller`. At each new time unit in a simulation, the simulation time along with the current loads and flows that load each node and link is stored. This is done until the end of the simulation.

5.5.3 Node

What kind of properties the node has and how the node behaves when a simulation runs is controlled by the node itself. With this structure, changes must be done directly to the node to change how the node responds to the system. This includes, for example, the static attributes, the load that the node has during simulation and how the load should be routed. The package that contains all the functions for a node is called `node`.

CellNode

The `CellNode` is the core object when a node is created. All the simulation functionality and attributes that is associated to a node is stored in the `CellNode` object. Further more, the attributes are stored in the `NodeInfo` class to increase the readability. See appendix A section 1.1 for the attributes and methods of the `CellNode` class.

NodeInfo

All the attributes (see appendix A section 1.2) for a node is contained in the `NodeInfo` class. The attributes that is set for a `NodeInfo` object are the set up that is then used during simulation. There is one attribute that determines if the node is a regular node or a `DataGenNode`. The `DataGenNode` works as data generators for the regular nodes. This means that when a regular node requests data, they must get it from a `DataGenNode`. At least one `DataGenNode` must exist in the setup for the NSS to be able to run.

5.5.4 NodePaths

The package `NodePaths` is the part of the NSS that find the different paths between existing nodes in the system. The paths consist of possible paths between regular nodes and a `DataGenNode` nodes. To find these paths, a method is used to iterate the neighbors of each node in the system and investigating if there is a connection to a `DataGenNode` from the existing neighbors. The method have four input parameters:

- Source node.
- Destination node.
- Visited nodes.
- Visited links.

The method to investigating this works in the following order:

1. Take one **node** from the network and one `DataGenNode` as destination.
2. Iterate the links of the **node**.
   a. Get the *adjacent node* that the link is connected to.
   b. Add **node** to visited nodes. (To ensure that no loops occur).
   c. Check if *adjacent node* is equal to destination.
      i) If true, add *adjacent node* to path.
      ii) If false, add *adjacent node* to visited nodes. Recursive call with *adjacent node* and the visited nodes.

This method is repeated until all regular nodes and `DataGenNode` combinations have been checked. Figure 5.1 shows a flowchart of the path finding algorithm.

5.5.5 Link

Like the node, changes to the link must be done directly to the link itself. A link cannot be created without at least two nodes existing in the network. The package that contains all the functions for a link is called `link`, it contains the following Java classes:

- `DataLink`.
- `DataLinkInfo`.

DataLink

`DataLink` controls the behavior of a link during simulation. All the static attributes that the link contains are stored in a `DataLinkInfo` object in `DataLink`.

DataLinkInfo

The DataLinkInfo class contains all the attributes for a link. Other than the attributes, the class determines whether a link already exists between two nodes.

5.5.6 Flow

The `flow` package is the part of the NSS that takes a new sample from the statistical model that is used for the flow and load the links and nodes in the system. How each generated flow generates loads to links and nodes in the network is determined in each flow. The classes that construct the flows are:

- `DataFlow`.
- `DataFlowInfo`.

DataFlow

During the setup of a node, the user can create `DataFlow` objects. The objects will contain all the attributes that the flow can influence in the network, for example it will contain the `CellNodes` and the `DataLinks` that the `DataFlow` is connected to.

DataFlowInfo

The `DataFlowInfo` object holds all the user defined attributes that is connected to a flow.

5.6 View

All the user input to the NSS is controlled graphically. This includes creation of nodes and links, attribute changes etc. All graphics are part of the `view`-package.

5.6.1 Windows

This is the main frame that displays the different parts of the NSS. Each window has a specific function. For example, one window displays all the nodes and links that have been created, another window displays simulation results and one enable the user to edit a link or node. This separates the different parts of the graphics depending on how the user interacts with the system.

Each window has a controller that puts functionality to the graphical elements in the window. This makes it possible to, for example, display a node when created or save the changes that are made to a node while editing its attributes.

5.6.2 Node

Each node has its own graphical element and has a controller connected to it. To detect when the user interacts with the node, a listener is connected to the graphical element. When an interaction is detected, the controller executes the corresponding action depending on what input it has received. For example, when the user drags a node, the controller changes the x and y coordinates in the corresponding coordinate system that the node is connected to.

5.6.3 Link

The link differs a bit from the node's graphical structure. For example, a link cannot be dragged. The position and magnitude of the link is entirely dependent on the two nodes that the link is connected to. Hence, if one of the nodes changes position the magnitude and the direction of the link changes. In Figure 5.2 is a depiction of some nodes linked together.

5.7 The Simulator

Only a discrete event simulation is available with the NSS and the simulations are handled by the DESMO-J library. When a simulation is started, parameters like simulation time, number of nodes, node properties etc. are set. The simulation environment, called experiment, is then initiated and a schedule list is set up to handle all the events. The events, in this case, consists of `CellNodes`, `DataLinks`, `DataFlows` and a `SimulatorController`. These are placed in

the schedule and for the very first run all the events are scheduled at time instant 0. The simulator then goes through each event on the schedule list and performs the calculations associated with that specific event. When the calculations on an event is finished the event is assigned a new time instant of activation and chronologically placed on the schedule list. The assigned time instant is decided either by random or by a fixed time step. When an event is done it is passivated, which means that the event will be passive until the simulation reaches the time instant of that event. When all events for time instant 0.0 are finished the simulation time is increased to the next time instant decided by the next event on the schedule. This continues until the simulation eventually reaches the stop time set by the user and the simulation is halted.

During the entire simulation the `SimulatorController` event is used to sample data from the simulation. This event is set to be reactivated at a fixed time instant and collects key data value from all other events that is later used to display statistics of load on both nodes and links.

5.7.1 Main Window

When starting the NSS, the user is presented with a minimalistic window divided in a toolbar for simulation controls and five main sections. In figure 5.3 a screen shot of the window is shown. The left topmost section is known as the simulation stats section and it displays statistical information about simulations done. This includes, number of simulation runs, current simulation time and number of simulation resets. The section right below is the nodes and links information section. Here, information concerning node and link properties, like latency and throughput are shown. The middle section is the Main workspace area which is where the network, to be simulated, is built. On the right side of the window two more sections are found. The upper section is currently unused and can be seen as a resource space for future development of the software. The lower section is the console window, where important information like simulation status, debug information and error messages are displayed.

5.7.2 Creating a Network

There are three main building blocks to use when creating a network in the NSS. These are regular nodes, data generation nodes and data links. Regular nodes are the primary objects for creating networks. These objects contain all the configurations that decides how the routing of data flows should behave. A data generation node represents an entry point of internet access to the network and is needed to enable any kind of simulation. The `DataGenNode` will mainly act as a resource point and will not route any data in the network. Lastly there are links that are used to connect nodes with each other. Links possess some properties like capacity and latency that will affect the data flows going through them. All these objects can be seen in figure 5.2.

5.7.3 Simulation runs

When a network has been created and all necessary parameters have been set a simulation can be run by pressing the "Start simulation" button. During a simulation, some visual cues of the status of the simulation will be given. There is a low usage of resources indicator, which indicates if a link or a regular node is underutilized by changing color of that object to yellow. Another visual cue is the node disabled indicator. Which turns a node and its connecting links red when a node is disabled. This can help to indicate fail rate of nodes and may force a reroute of data flows depending on the network structure. En example of a disabled node can be seen in 5.4 where node0 is deactivated during simulation.

5.7.4 Visualization of Results

During the early stages of the development, all results of a simulation was printed in the terminal. This was of course unsuitable as an end product solution. For this reason a histogram feature was implemented that plot the results of a simulation on a graph. For the histogram feature to work a special event called `SimulationController` was used to repeatedly sample data from an ongoing simulation. Depending on the sampling rate, the resolution of the graph could be decided.

During sampling, key data of both nodes and data links are collected and stored. Some of these data are:

- Current simulation time.
- Current load of node and link.
- Current data capacity limit of node and link.
- Latency for both node and links.

Besides gathering pure data of nodes and links some behavioral patterns of data is stored too. For instance, when a node switches paths for data delivery from one possible path to another. Figure 5.5 shows a screen shot of how a plot looks like after a simulation.

5.8 Important Features

As mentioned earlier there were some requests for some added features and functionality from Ericsson for the NSS. For the full list of requested features see Appendix B. The features from our prioritized list were all implemented but not all features from the request list were. Some were realized fully while others were only partially implemented. The result of the most important features will be discussed next.

5.8.1 Import and Export

The Import and export functionality was an important request as it can be quite cumbersome setting up a desired network. For each node that is added to a network, parameters like node data capacity, node latency, cost etc. need to be set. These parameters are set to a default value on creation but will most of the time need to be changed to better fit the testing case. Besides this, data flows need to be added to simulate the intended load produced by a node. After this, all links need to be connected to create the network structure. To avoid having to do all these configurations each time the same test case is to be simulated, one can export the network. This will create three save files, each containing a specific set of data. One file contains all the information regarding the nodes and their properties. The next file will contain the coordinates of these nodes and the third save file will contain information about the connected links associated with those nodes. Thanks to this feature a previously set up network can be reused repeatedly in different instances of the NSS and network created can be shared with others.

After the import/export functionality was added its usefulness was quickly realized. This helped speed up testing of the NSS considerably since now test case networks could be created and stored for later use rather than having to reconfigure all parameters every time. This became even more useful with added functionality to the NSS since that meant more parameters to set. Another important benefit was that test cases could now be shared and multiple tests could be run on the same network configuration.

5.8.2 JavaScript Engine

Another highly prioritized feature was the ability to easily alter the behavior of nodes without having to change the source code. The implementation of a JavaScript engine allows for just that. Combining a scripting language with access to all the parameters that are available including all the public methods in the program allows for highly complex behaviors to be implemented relatively easy. Furthermore, these scripts can be loaded on to multiple nodes and shared to others using the NSS.

To enable the JavaScript feature, a special JavaScript setup file needs to be loaded before behavioral scripts can be utilized. The setup script gathers all the objects available in the NSS and store key parameter values like current simulation time, or node id of the node calling the script as JavaScript variables. This increases the abstraction level for the user as the user now only has to use the predefined JavaScript variables instead of having to figure out how to pull the values from the system. These values can be manipulated as wished and can even be used in local and external methods. The results can then be saved in the same predefined values and once the behavioral script has finished executing, the setup script updates all system variables with the JavaScript variables.

5.8.3 Data Flows

The data flow features allow the user to set up multiple data streams from each node in the network. These streams represent different kinds of data and can be customized independently. The streams can also be set to take a specific path in the network to reach a data generation node. This is the part of the NSS that will load each node and link with data. Each `DataFlow` will create `DataPackets` throughout a simulation that corresponds to data requests from a node. The `DataPackets` will be active for a certain amount of time, to then be consumed and removed from the system which will trigger the creation of a new `DataPacket`. This `DataPacket` will load a set of nodes and links that lead to a `DataGenNode`.

5.9 Data Validation

To validate the simulation data from the NSS, our own simulation logs were compared with the generated trace logs from DESMO-J (see Figure 5.6). The process to validate the data was done in the following steps:

1. Run a simulation with a `DataFlow` connected to each node in the network (except the `DataGenNodes`).
2. Check the prints from the routing algorithm so that:
   (a) The `DataPacket` is loading each node and link in the selected `Path`.
   (b) The result from the DESMO-J trace files are the same as the routing algorithm.
   (c) The `accumulatedLoad` for each node and link in the path is correctly calculated with the new load received from the `DataPacket`.
3. Check the prints from the sampling in `SimulationController` so that they check out with the calculated values in Step 2b.
4. Validate that the plots that are displayed for each `DataFlow` matches the results from Step 2-3.

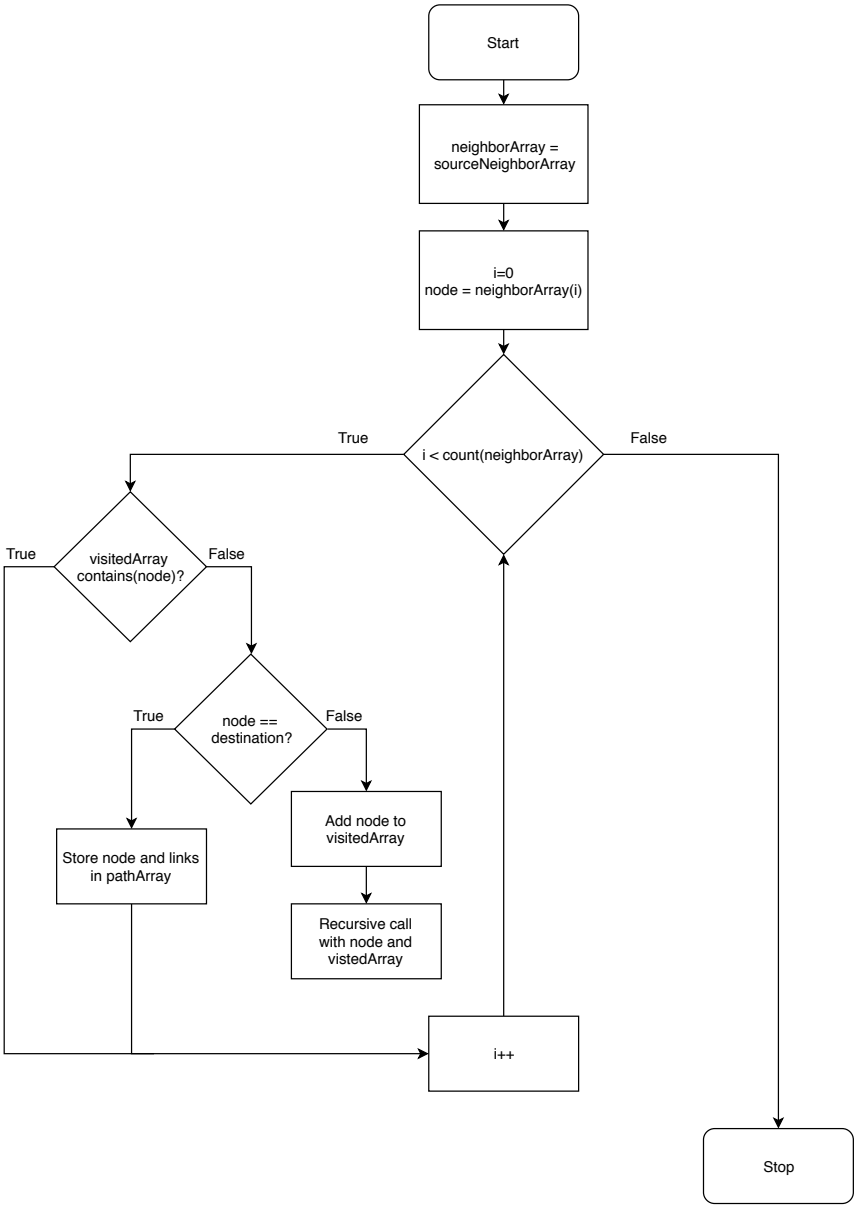


Figure 5.1: *Flowchart of path finding algorithm.*
The "Add node to visited array"-step ensures that no loops occurs. With the recursion call, all the possible paths from a given node to the destination are found. Source: Joel Olsson

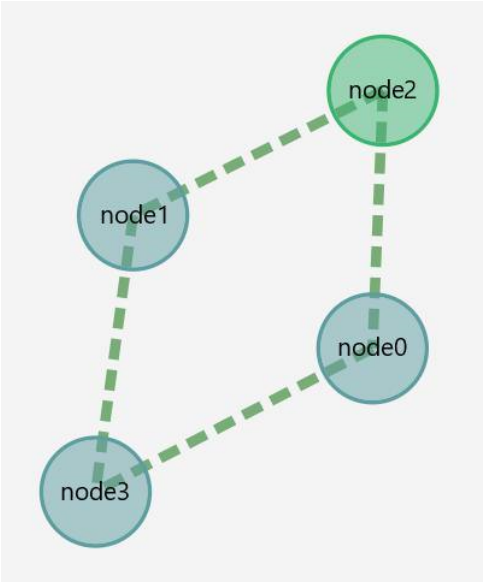


Figure 5.2: *Graphics of nodes and links. The blue circles represent regular nodes (node0, node1, node3), the green circle is a data generating node (node2) and the dotted green lines are links, connecting nodes together. Source: Joel Olsson*

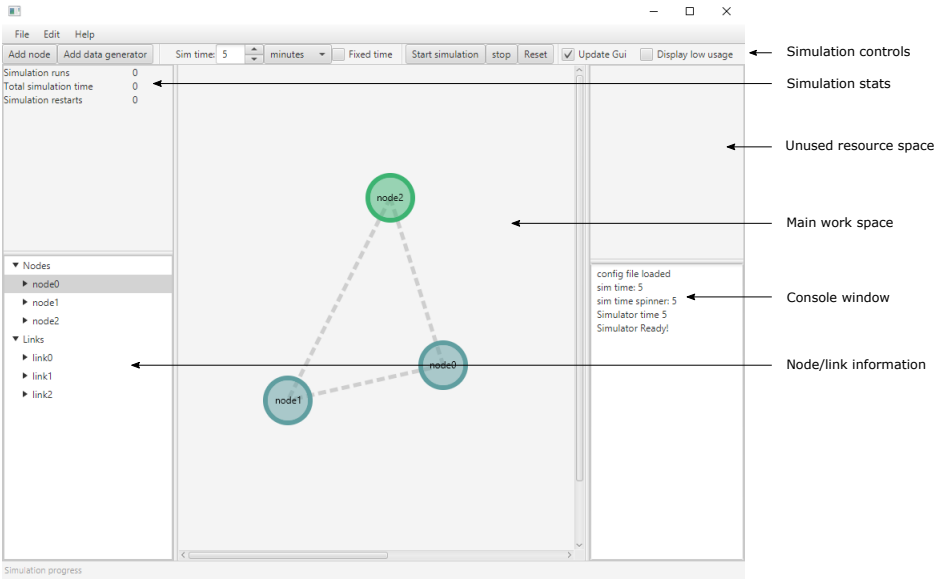


Figure 5.3: *The main window when an instance of the NSS is started. Source: Junior Asante*

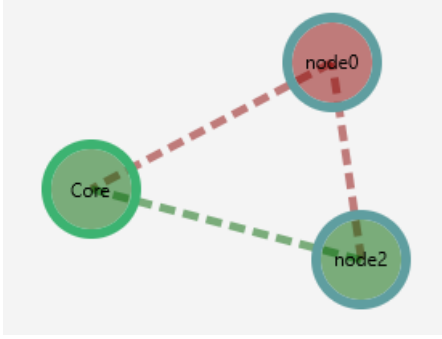


Figure 5.4: *A network example where node0 and its connecting links are disabled blocking any possible routing through that node. Source: Joel Olsson*

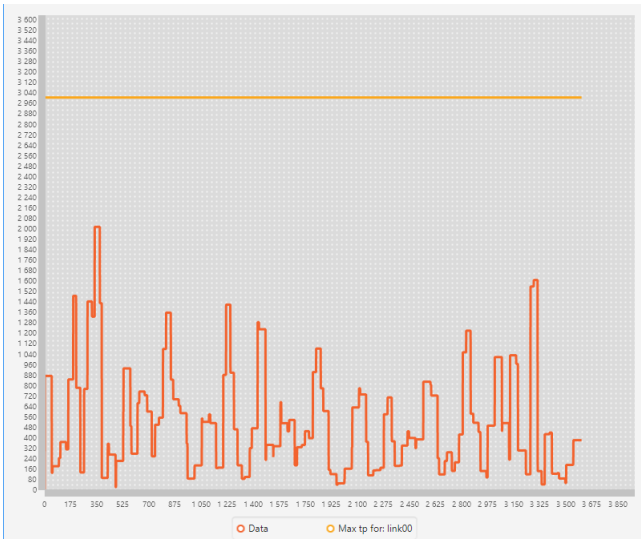


Figure 5.5: *The load results of a link after a simulation runtime of 3600 seconds. The upper bar indicates the load capacity limit of the link. Source: Junior Asante*

51.0000		holds for 1.0000 until 52.0000
52.0000		holds for 1.0000 until 53.0000
53.0000		holds for 1.0000 until 54.0000
54.0000		holds for 1.0000 until 55.0000
55.0000	'flow2#1'	activates 'node#3' immediately (preempted)
		passivates
	'node#3'	passivates
	'flow2#1'	activates 'link0#1' immediately (preempted)
		passivates
	'link0#1'	passivates
	'flow2#1'	holds for 55.0000 until 110.0000
	'SimulatorController#1'	holds for 1.0000 until 56.0000
56.0000		holds for 1.0000 until 57.0000

Figure 5.6: *Figure 13: DESMO-J trace file of a simulation. This file shows all the events that happens to the SimProcesses in the NSS. The leftmost column is the simulation time, the middle column is the name of the SimProcesses and the rightmost column is a description of the event that occurred. Source: Joel Olsson*

6

Discussion

In this chapter follows a discussion of the used method and the results of this thesis.

6.1 Method

The general method used for this thesis worked well. However, there have been some drawbacks in certain areas. In the following sections each part of the thesis will be discussed with the method in focus.

6.1.1 Choice of Platform

The first topic to be discussed is the choice of a platform/programming language to develop the NSS in. It was difficult to get a sense of how well the platform or language was suited for the development of the proposed system, partly due to its complexity. Even with some extensive prestudy the choice of platform was far from given.

One of our main concerns when developing the NSS was the GUI aspect. Having some previous experience with GUI-programming meant that we knew this was going to be a time-consuming part of the NSS. Even with this expectation, the development associated with the GUI took longer than anticipated. This was one of the main reasons game engines were viable as a platform of choice for the development of the NSS.

6.1.2 Resources

The hardware resources used to develop the NSS were barely sufficient to handle all the applications that were running during development. The small main memory was one of the biggest bottlenecks considering that over 90% of it was sometimes utilized even before the IDE was started. However, developing and running the code still worked surprisingly well.

We were offered to use the remote terminal at Ericsson but after some failed attempts to set up our work environment, we decided to continue working on our laptops. In hindsight more effort probably should have been put to getting the terminal to work.

6.1.3 Choice of Architecture

When picking an architecture for this software system, it was realized early on that we needed a well thought separation between the graphical part of the system and the simulator core. However, other parts of the system were more difficult to foresee a given architectural structure for. This was most likely the reason for why the structure of the system had to be changed during the development.

6.1.4 GUI

Since our knowledge concerning GUI development was limited, the GUI became a source of many problems. Even though the Scene Builder tool helped immensely, it could not help with the process of controlling the interface in the same extent as it helped create the interface. For example, nodes and links had the requirement of being dynamically added to the NSS during runtime. This made an implicit requirement on the graphical part of the links and nodes: they should be editable while the NSS is running. To be able to have this functionality, the interaction and creation of nodes and links were written in Java code and not created through Scene Builder. This made the graphical implementation of the NSS more complex to understand since some parts are made in Scene Builder and some parts are written in plain Java.

6.2 Results

When comparing the NSS to the feature list in Appendix B, it can be stated that the project still is under development. Even though most of the features have been implemented to some extent, there were still some features we were not able to implement due to time restraint. A lot of time was spent rewriting parts of the NSS and this is one area that could have been improved by better understanding the complex functions of a network simulator. However, an important

aspect to consider is that development time have been taken to implement and improve other features that were not listed. These are features that either improves the functionality of the simulator or increases the user experience of it, like for instance the ability to pan and zoom the workfield.

6.3 Simulation Results

The simulation results of a simulation can be viewed within the simulator or can be exported to a text file. However, a different approach that could have been better is to build a database to store all the simulation data in. With this implementation, the results could have been more easily utilized by other software, for example Excel, for analyzing the data.

7

Conclusions

Our time at Ericsson was very pleasant and educational. We faced quite a few challenges and managed to solve most of them, gaining a lot of valuable experience along the way. The experience we got from developing the NSS helped us answer the problem statements mentioned in 1.4 Problem Statements and these will be answered next.

7.1 Answering the Problem Statements

The first two problem statements were how the complexity of data traffic load in large scale networks could be modelled in such a way that it was easy to comprehend and how such a complex system could be depicted in a detailed enough level without excessive use of hardware. Since these questions touches the same area they will be answered together. These were questions with no definitive answers. As mentioned in 3.4.1 Model, the best model should be a model that is as abstracted and simple as possible from the environment it tries to model while still providing sufficient data. This idea was applied to the second problem statement as well. One immediate question that arise however is, what is sufficient data? Since the purpose of the NSS is to simulate general data flow in a network there might be any number of information a user might want to extract from such a simulation. To be able to cover every possible data that a user might want to extract would not be feasible. So regarding the model and abstraction level, they were kept simple and placed at a high abstraction level in the beginning and with each new feature added the complexity of the model increased while the abstraction level of details decreased.

The next question asked was: What tools can be used to help ease the development of the NSS and what are the design choices that should be taken? These questions are answered in chapters 4.4 Resources and 4.9 Major Design Changes and therefore a shorter answer will be given here. The most important tool for us was the IDE, as it helped immensely with the structure of the software. It also provided suggestions and tool tips during programming which was of great help. Another major help was the scene builder application which dramatically decreased the time it took to get the window design we wanted. Thanks to the drag and drop possibilities of scene builder the results could be seen instantly rather than having to run the program to see the results.

The final question asked was which features were the most important to implement into the NSS for it to be useful for its intended use. We will disregard from the simulation core as this is not considered a feature but rather a necessity for a simulation to be possible. Another important notice is that evaluation of the most important features will solely be based on the features that attracted the most attention during the presentation of the software at Ericsson. One of the features that seemed to be attracting the most attention was the JavaScript engine. This feature really opened the possibilities since it meant that a user now had access to every variable and method available in the system. Having a feature like that meant that users with very special test cases could modify the behavior of the simulator to fit their need and even add missing features through a JavaScript.

Another feature that seemed to be appreciated was the import and export functionality of networks. Many seemed to dread having to recreate complex networks each time a new instance of the NSS was started. This worry was short lasted, seeing as how entire networks, including relevant parameters, could be exported and later imported.

7.2 Future Work

The NSS is very much an early prototype and there is a lot of improvements that can be done. This was obvious even during the presentation as there were quite a few improvement suggestions, such as being able to set different icons for nodes to easily distinguish between them and being able to export simulation data to a third-party software for data analysis and extensive plot functionality. Besides the new suggestions there were some features from the requirement list that we were not quite able to implement. This includes the possibility to cache data on a node and make it act partially like a data generation node for certain nodes. We had the intention to implement this feature and therefore some preparations have been done but the main implementation of the cache function is still missing. Another point of interest for continued development is to measure the performance of the NSS and look for any major performance issues. This is something that has received very little attention from us and therefore there might be some subopti-

mal solutions implemented in the system.

8

References

10 Best Video Game Engines (Rankings & Reviews) (2018) Game designing. Available at: https://www.gamedesigning.org/career/video-game-engines/ (Accessed: 9 May 2018).

About Qt - Qt Wiki (no date). Available at: https://wiki.qt.io/About_Qt (Accessed: 23 May 2018).

Advanced Simulation Library Expanding software ecosystem for the DSP/FPGA/GPU market (2015). Available at: https://hgpu.org/papers/AvtechScientific-ASL_Presentation.pdf (Accessed: 2 April 2018).

Albatross (no date) A Brief Description - C++ Information. Available at: http://www.cplusplus.com/info/description/ (Accessed: 23 May 2018).

André-Jönsson, H. (2018) 'Discussions on 5G'.

Banks, J., Nelson, B. L. and Nicol, D. M. (2005) Discrete-Event System Simulation Fourth Edition. Available at: https://s3.amazonaws.com/academia.edu.documents/35744346/discrete-event_system_simulation_by_jerry_banks.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1519997949&Signature=uGbYj7kKfzvCEr%2FuASv%2BFkNgatk%3D&response-content-

disposition=inline%3B filenam

(Accessed: 2 March 2018).

Bolier-Anton, D. and Ens, E. (1994) SIM:a C++ library for Discrete Event Simulation. Available at: https://www.researchgate.net/profile/Anton_Eliens/publication/2788746_SIM_a_C_library_for_Discrete_Event_Simulation/links/59292e7eaca27295a806514c/SIM-a-C-library-for-Discrete-Event-Simulation.pdf (Accessed: 2 April 2018).

C# Overview (no date). Available at: https://www.tutorialspoint.com/csharp/csharp_overview.htm (Accessed: 23 May 2018).

Cellular Fundamentals – Radio Access Network (RAN) (2014). Available at: http://www.peterpaulengelen.com/2014/03/cellular-fundamentals-radio-access-network-ran/ (Accessed: 19 April 2018).

Van Chien, T., Björnson, E. and Larsson, E. G. (2018) 'Joint Pilot Design and Uplink Power Allocation in Multi-Cell Massive MIMO Systems', IEEE Transactions on Wireless Communications, 17(3), pp. 2000–2015. doi: 10.1109/TWC.2017.2787702.

Core Network - Definition from Techopedia (no date). Available at: https://www.techopedia.com/definition/6641/core-network (Accessed: 19 April 2018).

Create Simple Model - MATLAB Simulink - MathWorks Nordic (no date). Available at: https://se.mathworks.com/help/simulink/gs/create-a-simple-model.html (Accessed: 3 April 2018).

DESMO-J (no date). Available at: http://desmoj.sourceforge.net/home.html (Accessed: 16 February 2018).

Epic Games (no date) UE4 Support FAQ. Available at: https://www.unrealengine.com/en-US/support/support-faq (Accessed: 3 April 2018).

Epic Games | About (no date). Available at: https://www.epicgames.com/about (Accessed: 2 April 2018).

Ericsson (2016) '5G Radio Access Technologies', White Paper, (April), p. 10. doi: Uen 284 23-3204 Rev C | April 2016.

Ericsson (no date). Available at: https://www.ericsson.com/en/about-us (Accessed: 22 May 2018).

Ericsson Elastic RAN (2016). Available at: https://www.ericsson.com/en/press-releases/2016/11/ericsson-increases-softbank-data-speeds-by-40-percent-in-tokyo-station-with-elastic-ran (Accessed: 20 May 2018).

Exponential distribution (no date). Available at: https://www.statlect.com/probability-distributions/exponential-distribution (Accessed: 2 March 2018).

Faraz Fallahi (2014) Awesome C++. Available at: https://github.com/fffaraz/awesome-cpp (Accessed: 2 April 2018).

Faulkner, D. W. (David W. and Harmer, A. (1999) Proceedings of the European Conference on Networks and Optical Communications 1999 (NOC'99). IOS Press.

Forbes, C. S. (2011) Statistical distributions. 4th edn. Wiley.

Game engine (2018) Wikipedia. Available at: https://en.wikipedia.org/wiki/Game_engine (Accessed: 2 April 2018).

Göbel, J. et al. (2013) The discrete event simulation framework DESMO-J: Review, comparision to other frameworks and latest development, Proceedings - 27th European Conference on Modelling and Simulation, ECMS 2013, 4, pp. 100–109. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84900304437&partnerID=40&md5=468987b7ea363021151c48faefd5755e.

Heidemann, J. and Isi, U. S. C. (2002) 'OMNeT++ Discrete Event Simulator', Audio, (March), pp. 1–9. Available at: https://www.omnetpp.org/intro (Accessed: 3 April 2018).

IMT Vision (2015) 'Framework and overall objectives of the future development of IMT for 2020 and beyond', 0, p. 21. Available at: https://www.itu.int/dms_pubrec/itu-r/rec/m/R-REC-M.2083-0-201509-I!!PDF-E.pdf.

Java Essentials, Part 1 (no date). Available at: http://www.oracle.com/technetwork/java/compile-136656.html#comm (Accessed: 23 May 2018).

Java Overview (no date). Available at: https://www.tutorialspoint.com/java/java_overview.htm (Accessed: 23 May 2018).

Kasera, S. and Narang, N. (2004) 3G networks:architecture, protocols and procedures:based on 3GPP specifications for UMTS WCDMA networks. Tata McGraw Hill India.

Khan (2009) LTE for 4G Mobile Broadband. Cambridge University Press. Available at: http://books.google.com/books?id=6undOWaS_-0C&pgis=1 (Accessed: 20 May 2018).

List of game engines (2018). Wikipedia. Available at: https://en.wikipedia.org/wiki/List_of_game_engines (Accessed: 2 April 2018).

Marzetta, T. L. (2010) 'Noncooperative Cellular Wireless with Unlimited Numbers of Base Station Antennas', IEEE Transactions on Wireless Communications, 9(11), pp. 3590–3600. doi: 10.1109/TWC.2010.092810.091092.

Medbo, J. and Zaidi, A. (no date) Major breakthrough on mmWave propagation and channel modeling | Ericsson Research Blog. Available at: https://www.ericsson.com/research-blog/major-breakthrough-mmwave-propagation-channel-modeling (Accessed: 9 May 2018).

Meier, J. D. et al. (2009a) Architectural Patterns and Styles, Microsoft Press. doi: ee658117.

Meier, J. D. et al. (2009b) Architectural Patterns and Styles, Microsoft Press.

doi: ee658117.

MSDN (2014) 'Chapter 1: What is Software Architecture?', Microsoft Developer Network, pp. 1–4. Available at:
http://msdn.microsoft.com/en-us/library/ee658098.aspx.

Node-RED (no date). Available at: https://nodered.org/
(Accessed: 3 April 2018).

Nordrum, A. (2017) Everything You Need to Know About 5G - IEEE Spectrum. Available at: https://spectrum.ieee.org/video/telecom/wireless/everything-you-need-to-know-about-5g
(Accessed: 29 March 2018).

OSI model (no date). Available at: https://en.wikipedia.org/wiki/OSI_model
(Accessed: 9 May 2018).

Paul D Turner (no date) Crazy Eddie's GUI System: Crazy Eddie's GUI System Mk-2: Developer Documentation. Available at:
http://static.cegui.org.uk/docs/0.8.7/
(Accessed: 23 May 2018).

Pinho, T. M., Coelho, J. P. and Boaventura-Cunha, J. (2016) 'Forest-based supply chain modelling using the SimPy simulation framework', IFAC-PapersOnLine, 49(2), pp. 90–95.
doi: 10.1016/j.ifacol.2016.03.016.

Pros and Cons of Object Oriented Programming | Green Garage (no date). Available at:
https://greengarageblog.org/6-pros-and-cons-of-object-oriented-programming
(Accessed: 23 May 2018).

Qualcomm (2014) 'The Evolution of Mobile Technologies: 1G - 4G LTE', (June), pp. 1–41.
doi: 10.1017/CBO9781107415324.004.

Rich Moy (2017) The Difference Between Programming Frameworks and Languages, Stack overflow. Available at:
https://www.stackoverflowbusiness.com/blog/the-difference-between-programming-frameworks-and-languages
(Accessed: 23 May 2018).

Shariatmadari, H. et al. (2015) 'Machine-type communications: Current status and future perspectives toward 5G systems', IEEE Communications Magazine, 53(9), pp. 10–17.
doi: 10.1109/MCOM.2015.7263367.

Sharma, A., Kumar, M. and Agarwal, S. (2015) 'A Complete Survey on Software Architectural Styles and Patterns', Procedia Computer Science. Elsevier Masson SAS, 70, pp. 16–28.
doi: 10.1016/j.procs.2015.10.019.

Siegfried, R. (2014) Modeling and Simulation of Complex Systems, Proceedings of the 2004 Winter Simulation Conference, 2004.
doi: 10.1007/978-3-658-07529-3.

SSJ: SSJ User's Guide. (2013). Available at:
http://umontreal-simul.github.io/ssj/docs/master/index.html
(Accessed: 3 April 2018).

van Staden, T. (2012) 'Investigation Into the Optimization of Low Speed Communication Protocols for Narrow Band Networks', (December).

Swindlehurst, A. L. et al. (2014) 'Millimeter-wave massive MIMO: the next wireless revolution?', IEEE Communications Magazine, 52(9), pp. 56–62.
doi: 10.1109/MCOM.2014.6894453.

Team SimPY (2016) Overview — SimPy 3.0.10 documentation. Available at:
https://simpy.readthedocs.io/en/latest/index.html
(Accessed: 3 April 2018).

The Python Tutorial — Python 3.6.5 documentation (no date). Available at:
https://docs.python.org/3/tutorial/index.html
(Accessed: 23 May 2018).

Topçu, O. et al. (no date) 'Simulation Foundations, Methods and Applications'. Available at: https://link-springer-com.e.bibl.liu.se/content/pdf/10.1007%2F978-3-319-03050-0.pdf
(Accessed: 2 March 2018).

Tseng, H. W. et al. (2017) 'A resource allocation scheme for device-To-device communication over ultra-dense 5G cellular networks', Proceedings of the 2017

IEEE International Conference on Applied System Innovation: Applied System Innovation for Modern Technology, ICASI 2017, pp. 80–83.
doi: 10.1109/ICASI.2017.7988351.

Unity - Fast Facts (no date). Available at: https://unity3d.com/public-relations (Accessed: 2 April 2018).

Unity - Products (no date). Available at: https://unity3d.com/unity (Accessed: 2 April 2018).

Velten, K. (2009) 'Related Titles Ullmann ' s Modeling and Simulation Computer-Based Environmental Management Mathematical Methods in Science and Engineering Continuum Scale Simulation of Engineering Materials', p. 348.

Welcome to Enaml — Enaml 0.10.2 documentation (no date). Available at: http://enaml.readthedocs.io/en/latest/index.html (Accessed: 3 April 2018).

What Is JavaFX? | JavaFX 2.0 Tutorials and Documentation (2013). Available at: https://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm#A1141718 (Accessed: 3 April 2018).

Yun, S. et al. (2007) 'Hybrid Division Duplex System for Next-Generation Cellular Services', IEEE Transactions on Vehicular Technology, 56(5), pp. 3040–3059.
doi: 10.1109/TVT.2007.900389.

Appendix

A

Class attributes

A.1 Node Classes

The attributes and the methods that a node uses in the simulator.

A.1.1 CellNode

Attributes and methods for the `CellNode` class.

Attributes

In Table A.1 is a complete list of all the attributes that a `CellNode` has.

Name	Type	Explanation
myNodeInfo	private NodeInfo	NodeInfo object that stores the user defined attributes.
accumulatedLoad	private double	Holds the current accumulated load for the node.
currentDataPackets	private ArrayList <DataPacket>	Container that keeps all the current DataPackets that load the node.
currentDataFlows	private ArrayList <DataFlow>	Container that keeps all the current DataFlows that load the node
historyDataPackets	private NavigableMap <Double, ArrayList <DataPacket> >	Container that saves the simulation data for all the DataPackets that have been loading the node. Stores an ArrayList contating currentDataPackets as the value of the map and the simulation time as key.
historyDataFlows	private HashSet <DataFlow>	Container that saves the simulation data for all the DataFlows that have been loading the node. Stores all the DataFlows that has loaded the node in a HashSet.

Table A.1: *CellNode attributes.*

Methods

The `CellNode` methods will be listed and explained in Table A.2. The getters and setter for the class are not listed.

Name	Input	Return type	Explanation
`lifeCycle`		`void`	Method implemented by the DESMO-J library. The method is used to control the behaviour the link during simulation. The link will be passivated to await the next activation and do the corresponding action.
`checkActivation`		`void`	Method that checks what object that activated the node. Gets the activation ID of the object that activated the node. The method will either remove a `DataPacket` should be removed or route data. Depending on the setting on the node, default routing or custom routing will be used.
`defaultRouting`	`DataPacket`	`void`	Method that will use the default routing algorithm for the `DataPacket` that was passed to the method.
`findFlow`	`int`	`void`	Method that will be used to find the `DataFlow` that was loading the node, and then remove the load from the `accumulatedLoad` attribute.

jsRouting	DataPacket	void	Method that will be used to apply custom routing on the passed DataPacket. It will check, from the passed DataPacket attributes, if the destination has been reached and the node in the path should be loaded, otherwise routing will continue.
checkRoutingStatus	DataPacket, boolean	void	Help method for the jsRouting method. Will check the passed data from the DataPacket if the previous routing got ok or not. If the routing went ok, the passed boolean will be true and the path will be loaded. Else the routing will will not be handled.
loadPath	DataPacket	void	Method that will be used by the jsRouting to load the path (both links and nodes) that the data routed through.
checkFlows		void	Method that will activate all DataFlows that is defined in the node.
newRouteToNode	CellNode, DataFlow	boolean	Method that will be used when default routing from a node to a specific dataGenNode is selected. The method will try to use routePath method to route from the passed CellNode to the dataGenNode . Returns true if the routing was successful.

removeDataFlow	Path, DataFlow	boolean	Method that will be used to remove a DataFlow from all the links and nodes in a Path.
routePath	Path, DataFlow	boolean	Default routing method. Will be used to route a DataPacket , that is contained in a DataFlow , data on a Path .
checkPriority	DataFlow	ArrayList <DataFlow>	Will return a ArrayList of DataFlows of the flows that should be rerouted, since they are prioritized lower than the previously routed DataFlows.

Table A.2: *The CellNode methods.*

A.1.2 NodeInfo

All the attributes and methods that are defined in NodeInfo is displayed in this section.

Attributes

All the attributes of NodeInfo is included in Table A.3.

A.2 Link Classes

The classes and their corresponding attributes and methods that creates the links in the NSS.

A.2.1 DataLink

Attributes and methods for the `DataLink` object.

Attributes

The attributes contained in the `DataLink` class. In Table A.4 can all attributes be found.

Methods

The methods implemented in the `DataLink` class. No getters or setters are listed. A list of the methods can be seen in Table A.5.

A.2.2 DataLinkInfo

Attributes of `DataLinkInfo` and methods.

Attributes

The attributes in the `DataLinkInfo` class can be seen in Table A.6.

Methods

The methods for the `DataLinkInfo` class are shown in Table A.7.

A.3 Flow Classes

A.3.1 DataFlow

Attributes, classes and methods for the `DataFlow` object.

Name	Type	Explanation
`nodeId`	`private int`	Id for the node.
`nodeName`	`private String`	The name of the node.
`nodeUptime`	`private double`	The percentage of the time the node is up.
`nodeLatency`	`private double`	The latency of passing data through the node.
`nodeThroughput`	`private int`	The maximum throughput of the node.
`dataGenNode`	`private boolean`	Determines if the node is a `dataGenNode` or a regular node. A `dataGenNode` generates data.
`nodeCachable`	`private boolean`	Determines if the node is capable of caching data.
`nodeActive`	`private boolean`	Determines if the node is active.
`nodeLinkArray`	`private HashSet <Integer>`	Set that keeps all the neighbouring nodes.
`nodePathMap`	`private HashMap <Integer, NodePaths>`	A map that stores `NodePath` objects. Each pair corresponds to all the paths that exist to a `dataGenNode`.
`dataFlowMap`	`private ArrayList <DataFlow>`	List that stores all the `DataFlows` that this node have. This only includes the `DataFlows` that has the corresponding node as start point.

Table A.3: *`NodeInfo` attributes.*

Name	Type	Explanation
myDataLinkInfo	DataLinkInfo	The object that holds the attributes that the user input for link.
accumulatedLoad	double	The accumulated load that the node got during simulation.
historyDataFlows	private HashSet <DataFlow>	Container that stores all the DataFlows that has loaded the link during simulation.
historyDataPackets	private NavigableMap <Double, ArrayList <DataPacket»	All the DataPackets that has loaded the link during simulation.
currentDataPackets	private ArrayList <DataPacket>	All the DataPackets that load the link currently during simulation.
currentDataFlows	private HashSet <DataFlow>	All the DataFlows that load the link at the specific simulation time

Table A.4: *DataLink attributes.*

Name	Input	Return type	Explanation
`lifeCycle`		`void`	Method implemented by the `DESMO-J` library. The method is used to control the behaviour the link during simulation. The link will be passivated to await the next activation and do the corresponding action.
`checkActivation`		`void`	Method that checks what object that activated the link. Gets the activation ID of the object that activated the link.
`findFlow`	`int`	`void`	Method that checks what `DataFlow` that activated the link with the passed `int`. The method will remove the load from the `accumulatedLoad` attribute that the `DataFlow` loaded the link with.

Table A.5: *`DataLink` methods.*

Name	Type	Explanation
`dataLinkName`	`private String`	The name of the link.
`linkId`	`private int`	The id of the link.
`linkUptime`	`private double`	The percentile of the uptime of the link. For example, 0.99 will correspond the link having a 99% of being and up and 1% of being offline.
`linkLatency`	`private double`	Latency of passing the link.
`linkThroughput`	`private int`	The total amount of throughput that the link can handle.
`linkEnabled`	`private boolean`	Attribute that determines if the link is enabled or not.
`linkStability`	`private double`	Stability of the link.
`node0Id`	`private int`	One of the `CellNode` IDs that the link got.
`node1Id`	`private int`	One of the `CellNode` IDs that the link got.
`linkActive`	`private boolean`	Attribute that indicates if the link is active or not.

Table A.6: *`DataLinkInfo` attributes.*

Name	Input parameters	Return type	Explanation
`addLink`	`int, int`	`boolean`	Will check if the input parameters (two node ids) are connected. Returns true if connected, false if no connection is between the nodes.

Table A.7: *`DataLinkInfo` methods.*

Name	Type	Explanation
cellNode	private CellNode	The CellNode that is connected to the DataFlow.
dataPacket	private DataPacket	The object that keeps the DataPacket for the DataFlow.
dataFlowInfo	private DataFlow-Info	Object that keeps all the attributes that is defined by the user for the DataFlow.
historyDataPackets	private ArrayList <DataPacket>	All the historyDataPackets that was created during simulation for the DataFlow
historyDataSampling	private NavigableMap <Double, DataPacket>	All the data that DataPackets that was active and at that timestamp they were active.
myLoadGenerator	private LoadGenerator	The load generator for the DataFlow that will create the load for DataPacket object.
dataFlowDist	private DataFlowDistribution	The data distribution object for the DataFlow.
DataPacketStatus	enum	enum that hold status of routing of DataPackets.

Table A.8: *DataFlow attributes.*

Attributes

The attributes contained in the DataFlow class. In Table A.8 can all attributes be found.

Methods

The methods contained in the DataFlow class. In Table A.9 can the methods be found.

Name	Input parameters	Return type	Explanation
`lifeCycle`		`void`	Method that gets implemented from the DESMO-J library. The flow will activate itself after a time span and use the `checkFlow` to check what action should be done for the `DataFlow`.
`checkFlow`		`void`	Method that will first store consumed `DataPackets`, then remove them `DataPacket` from the nodes and links (by activating them). A new `DataPacket` will then be created, and a check what kind of routing that is used by the connected `CellNode`, and the routing will be started.
`removeDataPacket`		`void`	Method that will be used when a `DataPacket` is consumed. The method will iterate through the nodes and links and activate them to remove the `DataPacket`.

Table A.9: *`DataFlow` methods.*

DataPacket

The `DataPacket` class is a help class for the `DataFlow` object. It will create data for the `DataFlow` and hold attributes. In Table A.10 are the attributes for the `DataPacket` object.

A.3.2 DataFlowInfo

The attributes for the `DataFlowInfo` class. The only methods that exists are getters and setters for the class and they will not be displayed.

Attributes

The attributes for the `DataFlowInfo` object can be found in Table A.11.

Name	Type	Explanation
`flowId`	`private int`	The ID of the `DataFlowInfo` object.
`flowName`	`private String`	The name of the `DataFlowInfo` object.
`flowSourceNodeId`	`private int`	The ID of the `dataGenNode` that the data will route data to.
`flowSourceNodeId`	`private int`	The ID of the `CellNode` that the `DataFlowInfo` is connected to.
`flowDestinationNodeId`	`private int`	The ID of the `CellNode` that the `DataFlowInfo` is connected to.
`flowAverageThroughput`	`private double`	The average load that the `DataFlow` will create.
`flowCachable`	`private boolean`	If the flow should be cachable.
`flowEncodingLatency`	`private double`	The encoding latency of the data.
`flowDecodingLatency`	`private double`	The decoding latency of the data.
`flowCost`	`private int`	Arbitrary value of the cost of the `DataFlow`.
`flowPriority`	`private int`	The priority of the `DataFlow`.

flowPath	private Path	The path that the flow will route data through, used with default routing.
flowLatencyRequirementOk	private double	Latency requirement of the flow.
flowLatencyRequirementDeg	private double	Latency requirement of the flow.
flowLatencyRequirementNOk	private double	Latency requirement of the flow.
flowThroughputRequirementOk	private double	Throughput requirement of the flow.
flowThroughputRequirementDeg	private double	Throughput requirement of the flow.
flowThroughputRequirementNOk	private double	Throughput requirement of the flow.
dataFlowDistributionTypeObject	DistributionTypeObject	Distribution type of the flow.

Table A.11: *The* `DataFlowInfo` *attributes.*

A.4 NodePaths

Class that will find paths between regular `CellNode`s and `dataGenNode`s. Here is tables of the attributes, methods and classes for the `NodePaths`.

A.4.1 Attributes

In Table A.12 are all the attributes for `NodePaths`.

A.4.2 Methods

In Table A.13 are all the methods for `NodePaths`.

A.4.3 Path

The class `Path` hold all the attributes connected to a path between a regular `NodeInfo` object and a `dataGenNode`. In Table A.14 are all the attributes for the `Path` class.

Name	Type	Explanation
nodeIds	private LinkedList <Integer>	Node IDs of the nodes that the DataPacket is loading.
linkIds	private LinkedList <Integer>	Link IDs of the links that the DataPacket is loading.
sourceId	private int	Id of the dataGenNode that the node gets data from.
visitedNodes	private LinkedList <Integer>	The visited nodes, used while custom routing.
visitedLinks	private LinkedList <Integer>	The visited links, used while custom routing.
nodesActive	private LinkedList <Boolean>	Used in combination with visitedNodes and store if they are active, used while custom routing.
linksActive	private LinkedList <Boolean>	Used in combination with visitedLinks and store if they are active, used while custom routing.
flow	private DataFlow	The DataFlow that the DataPacket is connected to.
load	private double	The amount of data that the nodes and links should be loaded with.
totalLatency	private double	The total amount of latency that the DataPacket got to reach the source node.
requestedLoad	private double	The amount of data that the DataPacket created. Attribute is used while routing and is plotted as simulation result.
resultedLoad	private double	The amount of data that the DataPacket got through the nodes and links. Attribute is used while routing and is plotted as simulation result.

Table A.10: *DataPacket attributes.*

Name	Type	Explanation
paths	private LinkedList <Path>	Container for Path objects.
destinationNodeId	private int	ID of the dataGenNode that the Paths goes to.

Table A.12: *NodePaths attributes.*

Name	Input parameters	Return type	Explanation
findPaths	NodeInfo, int, LinkedList <Integer>, LinkedList <Integer>	void	Method that will find all Paths between a NodeInfo and a ID for a node. The two LinkedList<Integer>s are visited nodes and visited links. The Paths will be stored in the nodePathMap in the NodeInfo object that was passed to the method.

Table A.13: *NodePaths methods.*

Name	Type	Explanation
nodeIds	private LinkedList <Integer>	Container that hold all the IDs for the nodes that is included in the path.
linkIds	private LinkedList <Integer>	Container that hold all the IDs for the links that is included in the path.
totalLatency	private double	Value that hold the accumulated latency through all the links and nodes in the Path.

Table A.14: *Path attributes.*

B

Requested Features

Appendix of the feature request list presented to us for the development of the NSS.

B.1 Nodes

- Inputs for receiving packets.
- Outputs for delivering packets.
- A flow should be able to be tracked, e.g. a flow should be able to be set to a specific output or specifically spread over multiple outputs. Algorithms for doing this is not included.
- Data for a certain flow should be able to be stored locally on the node (from the client to the node).
- Flows should be possible to be stored locally (from the node to the source – alternatively that the source is moved to the node).
- Each flow is affected by a latency when passing through a node.
- One should be able to set up a service up-time for a node based on multi-connectivity.

B.2 Connections

- Each connection has a certain maximum throughput.
- Each connection has a certain latency associated.
- A possibility to be able to add costs for certain connections should be enabled (to trigger activation of certain connections).
- One should be able to dynamically connect or disconnect connections.
- One should be able to decide where possible connections are.
- Link reliability: Service uptime for a link (e.g. LTE system availability is 99.9%).

B.3 Flows

- A flow should have a starting point and end point.
- Each flow should have a demand on maximum latency that should be fulfilled. Several levels of this is possible (ok, ok but degraded performance, not ok).
- Each flow should have a demand for a certain throughput (OK, partially OK, NOK).
- Each flow should have a priority.
- A flow should be able to be split up if for instance a source is moved to a node.

B.4 Observability

- One should be able to see how much data is flowing through a link.
- One should be able to see how many flows were handled successfully.
- How many of the flows got OK/partially Ok/NOK performance?
- One should be able to see where the source and destination for each flow is.

B.5 Availability

- One should be able to write an algorithm to decide whether a connection should be disabled or not.
- One should be able to write an algorithm to decide how many packets that should be routed.
- One should be able to write an algorithm to decide whether a service should be moved to a node.
- One should be able to write these algorithms without access to the source code of the simulator.
- A scenario over a period e.g. a day where different flows are brought up and closed during different time instants after a mathematical model should be possible.