EXTENSION OF A 5G RAN SIMULATOR BY MODELING USER EQUIPMENT

Johan Sjöstrand

Tutor, Christer Bäckström
Examinator, Peter Jonsson
通过对用户设备进行建模来扩展 5G RAN 仿真器

EXTENSION OF A 5G RAN SIMULATOR BY MODELING USER EQUIPMENT

王维

哈尔滨工业大学                      Linköping University

2019 年 10 月
Dissertation for the Master’s Degree in Engineering

EXTENSION OF A 5G RAN SIMULATOR BY MODELING USER EQUIPMENT

Candidate: Johan Sjöstrand
Supervisor: Nie Lanshun, Professor
Associate Supervisor: Christer Bäckström, Professor
Industrial Supervisor: Henrik André-Jönsson, Main TC
Academic Degree Applied for: Master of Engineering
Speciality: Software Engineering
Affiliation: School of Software
Date of Defence: September 2019
Degree-Conferring-Institution: Harbin Institute of Technology IDA
Abstract

With the upcoming release of the fifth generation of cellular networks, 5G, it is expected to be a lot more connected devices. As a consequence, the importance of the capacity of the networks is increased, especially the radio access network (RAN), which will be changed a lot from previous generations. The best method to ensure a RAN has enough capacity for its targeted area is a simulation, and therefore, there is a demand for such a simulator. The development of a simulator for RAN started last year in another thesis with the possibility to model nodes, links, and set loads. In this thesis, the functionality is extended with the addition of mobility and user equipment (UE) to create load to more accurately simulate the network traffic. Roads are added as an entity to allow mobility, and cells are modeled to determine coverage.

Most requested functionality was implemented, with the main difficulties being integrating the new code into the existing code base. The simulator was then evaluated, both the validity of the model and the performance. To improve the simulator for its purpose, even more, functionalities such as geography elements like buildings or mountains blocking the signal should be considered. Another improvement would be to make links and cells less reliable. The simulator is modeling the perfect day scenario at the moment which does not always reflect upon reality.

**Keywords:** RAN; 5G; Simulation; DESMO-J; UE traffic model
# Glossary

| **Cell** | A limited area with access to a wireless connection. In this paper, a Cell is created by a radio antenna of a *Node* which is its source. |
| **CN** | *Core Network*, one component of the network architecture which is responsible for functionalities such as billing and connecting the network to the internet. |
| **Link** | In this paper, one *Link* connects two *Nodes* or connects one *Node* to the CN. |
| **Node** | A *Node* is a part of the RAN. A Node is a simplification in this paper. A Node represents a baseband unit and one or more antenna units. A node may have one or more Cells and allows UEs to connect to the mobile network. |
| **RAN** | *Radio Access Network*, one component of the network architecture which is responsible for radio connectivity. In this thesis, a RAN consists of Cells, Nodes, and Links. |
| **UE** | *User Equipment*, devices that connect to mobile networks, such as smartphones or sensors. |
| **XML** | *Extensible Markup Language* set rules for encoding documents. *FXML* is an extension used for JavaFX. |
目录

ABSTRACT .......................................................................................... II
GLOSSARY ........................................................................................... III

CHAPTER 1 INTRODUCTION ................................................................. 1
  1.1 BACKGROUND ............................................................................... 1
  1.2 THE PURPOSE OF THE PROJECT .................................................. 3
    1.2.1 Research questions ................................................................. 4
    1.2.2 Delimitations .......................................................................... 4
  1.3 THE STATUS OF RELATED RESEARCH ........................................... 5
    1.3.1 Related works .......................................................................... 5
    1.3.2 Network Simulation ................................................................. 7
    1.3.3 Software development methodologies and frameworks .......... 8
  1.4 METHOD ....................................................................................... 12
    1.4.1 Pre-study ................................................................................ 12
    1.4.2 Defining requirements ............................................................. 12
    1.4.3 Designing the system ............................................................... 13
    1.4.4 Implementing simulator ......................................................... 13
    1.4.5 Evaluating the system ............................................................. 14
  1.5 MAIN CONTENT AND ORGANIZATION OF THE THESIS ............ 16

CHAPTER 2 SYSTEM REQUIREMENT ANALYSIS .................................. 18
  2.1 THE GOAL OF THE SYSTEM ......................................................... 18
  2.2 5G ARCHITECTURE ...................................................................... 18
    2.2.1 Core Network ......................................................................... 19
    2.2.2 User Equipment ...................................................................... 19
    2.2.3 Radio Access Network ............................................................ 20
  2.3 THE FUNCTIONAL REQUIREMENTS .............................................. 21
    2.3.1 Use cases .............................................................................. 21
    2.3.2 Features ................................................................................. 21
<table>
<thead>
<tr>
<th>CHAPTER 3 SYSTEM DESIGN</th>
<th>CHAPTER 4 SYSTEM IMPLEMENTATION AND TESTING</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 SYSTEM ARCHITECTURE</td>
<td>4.1 THE ENVIRONMENT OF SYSTEM IMPLEMENTATION</td>
</tr>
<tr>
<td>3.2 ENTITIES OVERVIEW</td>
<td>4.1.1 Technical conditions</td>
</tr>
<tr>
<td>3.3 CLASS DESIGN</td>
<td>4.1.2 Development and testing environment</td>
</tr>
<tr>
<td>3.3.1 Road</td>
<td>4.2 KEY PROGRAM FLOW CHARTS</td>
</tr>
<tr>
<td>3.3.2 Cell</td>
<td>4.2.1 Main program flow</td>
</tr>
<tr>
<td>3.3.3 User Equipment</td>
<td>4.2.2 Simulation process flows</td>
</tr>
<tr>
<td>3.4 KEY TECHNIQUES</td>
<td>4.3 KEY INTERFACES OF THE SOFTWARE SYSTEM</td>
</tr>
<tr>
<td>3.4.1 The observer pattern</td>
<td>4.4 SYSTEM EVALUATION</td>
</tr>
<tr>
<td>3.4.2 Software reuse</td>
<td>4.4.1 Testing</td>
</tr>
<tr>
<td>3.5 BRIEF SUMMARY</td>
<td>4.4.2 Traffic model evaluation</td>
</tr>
<tr>
<td></td>
<td>4.4.3 System performance test</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CHAPTER 5 DISCUSSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1 REQUIREMENTS</td>
</tr>
<tr>
<td>5.2 DESIGN</td>
</tr>
<tr>
<td>Section</td>
</tr>
<tr>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>5.3 IMPLEMENTATION</td>
</tr>
<tr>
<td>5.4 SYSTEM EVALUATION</td>
</tr>
<tr>
<td>5.5 THE WORK IN A WIDER CONTEXT</td>
</tr>
<tr>
<td>CONCLUSION</td>
</tr>
<tr>
<td>REFERENCES</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENT</td>
</tr>
</tbody>
</table>
Chapter 1 Introduction

1.1 Background

With the roll-out of 5G this year, it is predicted that the number of connected devices will increase exponentially, and the cellular traffic is expected to increase [1]. 5G networks specifications have high demands regarding performance and availability, such as clients being capable of moving up to 500 km/h without losing connection and reducing latency compared to today [2]. There is a high variety of devices to be connected, and they do not necessarily have the same demands on the network. A connected car needs ultra-reliable low-latency communication to receive and send information about its surrounding [2]. A smart-phone might demand very high throughput connection to stream 4k video [2]. A sensor may need to be as energy efficient as possible to have a battery life longer than ten years [2]. 5G needs to fulfill all of these demands [2].

A broader spectrum of radio frequencies needs to be used, compared to the previous generations, to fulfill these demands. 4G operates below 6GHz [3], and these frequencies are already used by a lot of other radio services, like Wi-Fi, Bluetooth, and, microwaves. All of these are operating on 2.4GHz, which affects performance and bandwidth if they interfere with each other [3]. However, it is not as simple as just increasing the frequency, because the higher the frequency a cell operates, the smaller area it can cover [3]. Since 5G aims to utilize the higher frequencies [3], it requires the use of many more and smaller cells than in traditional radio access networks (RAN). Therefore, more nodes and cells are needed to cover the same area, as seen in Figure 1 and Figure 2. In Figure 2, the cells are not visible to make the figure clearer. 5G will still use lower frequencies as the backbone of the RAN and add the higher frequencies to increase system capacity [3].
Since the RANs will become denser of nodes, the logic and coordination between the nodes are increasingly complex [4]. Traditionally in 4G networks, all RAN nodes are coupled with a Core Network (CN) connection. With the increase of nodes, this will be difficult to achieve, and the traffic will be routed within the RAN to the nearest CN connection in 5G. In Figure 1 and Figure 2, the nodes are illustrated as antennas, CN connection as a cloud, and the cells as teardrops. The increase of nodes put higher demands on the capacity of the links between the nodes. Preferably, these links should be fiber cables for maximum throughput and lowest latency [4]. However, this is not always possible since fiber cables are expensive to dig, and some nodes are placed in areas with mountains or rivers where it is simply not possible to reach with a cable [4]. The other solution is transferring data within the RAN with microwave wireless links [4]. These microwave links are more limited in bandwidth than fiber links [4]. Setting up microwave links instead of fiber can be a good idea to save money if the expected traffic allows it. This creates a need to ensure that the links will be able to handle the traffic demands before setting up the network.

**Ericsson**

The work of this thesis was performed at Ericsson. Ericsson is one of the leading providers of Information and Communication Technology to the service providers,
with over 95000 employees worldwide\textsuperscript{1}. Ericsson provides full or partial solutions to telecommunication networks and is currently developing 5G.

1.2 The purpose of the project

RANs are getting more complicated with the roll-out of 5G, and to predict performance there is a need for a simulator. A trial and error approach is not feasible when building a cellular network since the equipment is costly, and the demands of the customers must be met. With a simulator, a RAN can be planned for an area to meet the expected load without any high costs. It can also be used to test different network routing algorithms for load-balancing to increase performance further.

As a consequence of the change in the RAN, some of the tools used for evaluating mobile networks can no longer suffice for 5G standard. Therefore, J Asante and J. Olsson were given the task by Ericsson in a master thesis \cite{5} to develop a tool for simulation of RAN. The thesis resulted in a simulator that can model RAN for evaluating performance and try different routing algorithms. The traffic was determined by a load that was assigned to each node. How the load was set put demands on the user to know about mobile network characteristics.

According to \cite{6}, it is essential to know different User Equipment (UE) events and contexts to understand the characteristics of a real network load in order to simulate a network. Therefore, to make accurate and more realistic simulations of the network load, the UE should also be simulated. Using simulated UEs to generate load will make the simulation model more realistic and accurate when simulating an actual area.

The goal of this thesis is to extend the existing simulator with the support for cells, positioning, and moving UEs to solve load balancing problems among the RAN

\textsuperscript{1} Ericsson ”About us” Accessed: 2019-02-06, URL: \url{https://www.ericsson.com/en/about-us}
nodes with realistic network traffic. Problems that the simulator might answer are, for example, which links do not have to be of fiber. By setting up microwave links instead of fiber links between nodes, the operators can save money when building their cellular networks. However, the tradeoff is lower throughput in these links. With a simulator, the requirements of the network can be estimated to allow the operators to plan before building and identify where microwave links can be used instead of fiber links.

1.2.1 Research questions

During this thesis, these research questions are expected to be answered.

- How could UE behavior be simulated to be sufficient as traffic load input for a RAN-model for simulation?
- How does adding UEs affect a RAN simulators ability to imitate a real RAN?

The thesis will include a literature study and the development of a simulator with a particular focus on implementing the entity UE.

1.2.2 Delimitations

This thesis primarily focuses on the development of a tool to answer the problem statement described above.

To be able to develop this functionality alone during the time frame of a thesis, some limitations and simplifications will be made. If there are multiple overlapping cells of an area, UEs will pick one disregarding any logic in making this choice for simplification. The UE continue using the same cell until it no longer is positioned within the coverage of the cell at which it will then pick a new cell it can reach. This logic means that handover procedures will not be accounted for realistically. The radio access properties such as frequencies, channels, and beaming will not be accounted for in this simulator. The focus here is node-to-node communication, but UEs are added for a more realistic traffic model of the load.
The simulator is developed in the java simulation framework DESMO-J with a graphical user interface developed in JavaFX. This decision will not be changed as J. Asante and J. Olsson already motivated this choice in [5], and it would require too much work for the timeframe of this thesis.

Validation of this simulator will not be guaranteed. The methods used to check are naive and not scientific. They aim to check if the models are close to reality. Many simplifications are done because of lack of knowledge and time regarding validation checking.

1.3 The status of related research

This section will describe related works and present relevant theory for conducting network simulation research.

1.3.1 Related works

N. Mohsen and K. S. Hassan [4] developed a simulation tool in MATLAB for evaluating a promising 5G architecture, cloud radio access network. In their model, they included three core components, the UE, the remote radio heads as RAN nodes, and the cloud controller. Their model assumes that each UE sends their channel-state information resembled in quantized channel-quality indicator for each node and use this for the best resource scheduling. Their results show the cloud-RAN architecture to be promising, but the link capacity between nodes remain the limiting factor and requires perfect design. Their work is interesting for this thesis as guidelines for simulating UE behavior and RAN networks.

A. Karimi et al. [7] investigate solutions for the third generation partnership programs (3GPP) requirements for the use case category ultra-reliable low-latency, where the demands are up to as high as 1-millisecond latency with 99.999% reliability. The proposed solution is a centralized radio access network architecture with fast multi-cell scheduling. They describe two algorithms, one for cell segmentation and one for cell association of a UE. The algorithms are tested in a sophisticated system-level simulation and show significant latency performance of
centralized multi-cell scheduling gains compared to traditional distributed solutions.

C. Tsai and M. Moh [6] investigates load balancing in cloud-RAN by comparing eight load-balancing algorithms in a simulation. Cloud-RAN is an architecture proposed where baseband units are pooled into the same location, connected to remote radio heads. The goal was to reduce latencies for Internet of Things communication. The algorithms chosen to compare was practical and straightforward, as opposed to like some of the load balancing algorithms proposed for cloud-RAN that are intelligent and complex. The authors reasoned that the intelligent and sophisticated algorithms cannot be used in practice yet for 5G networks, and therefore focus on those that may be readily deployed in actual networks. They set up a realistic cloud-RAN environment based on real network data, used for simulation. Their findings are that out of the eight; the queue-length analysis is a simple, light-weighted algorithm that performs almost as good as a more complex waiting-time algorithm. The queue-length algorithm assigns incoming requests to the virtual machine with currently the shortest queue-length.

O. Shaikh and F. Shahzad developed a browser-based network simulator in [8]. They found the need for a simulator that is easily accessible a necessity to quickly prototype a visualization for a wireless network, used for Internet of Things. A simulator for such purpose that is browser-based was unheard of before. The simulator is built with the help of the libraries and frameworks HTML, jQuery, D3.js, JSNetworkX, and Bootstrap. They describe their development progress and decisions for the graphical user interface they implement, which is useful for this thesis, but they do not mention a lot about the simulator core. Although none of the tools used in Visual-netsim is used in this thesis, there exist a few similarities and can be used as inspiration for this thesis.

M. Mezzavilla et al. [9] extends an existing simulator tool to support millimeter wave communication in upcoming 5G networks. Millimeter wave is radio signals operating at a very high frequency, between 6-300 GHz. Millimeter wave technology is expected to be a key enabler for the massive throughput demands of future networks. The higher frequencies allow a broader spectrum to be used, but
it also brings many problems since obstacles can stop the waves and require them to be more directed than radio waves of lower frequencies. The lack of an end-to-end simulator for millimeter wave operations led to this work to extend ns-3 network simulator and the LTE LENA module.

1.3.2 Network Simulation

Systems, Models, and Simulation

In the world of mathematical modeling and simulation, a system is an object of interest, whose properties we want to study [10]. A system could be a car, a nuclear power plant, or a 5G network. To understand such a complex system as a car, we make simplifications. To drive a car, one would not need to understand how the engine works, but that the gas pedal increases the speed, the brake reduces the speed, and the steering wheel changes the direction of the car. This simplification works fine if the car does not break. This simplified description of a car is called a model [10]. Models are used to understand complex systems and to solve problems. A model should not be more complicated than necessary; the goal of using the model should drive the complexity [10]. In the example above, the model should be enough for many people to understand how to drive a car. The best models are the simplest possible that still can be used to understand a system and solve problems for it [10]. Simulation can generally be described as performing experiments on a model [10]. With simulation, we can derive strategies that solve our problems or answer questions regarding a complex system.

Model validation and verification

One of the essential steps in a simulation is validating the model, and it is often overlooked in the literature [11]. How can one know if the model is accurate enough to describe reality for its purpose? In [12], the author describes three types of errors when conducting simulation studies. Type 1 is rejecting the model credibility when it is sufficiently credible. Type 2 is accepting a model which is not sufficiently credible. Type 3 is solving the wrong problem. The methods of validation have often come down to visual comparison [11]. One example is a Turing test, where an expert is presented with two sets of data, one generated and one observed, and the task is to identify the observed set [11]. A positive result is
if the expert fails to do so. These visual methods of validation are classified as informal methods since they do not contain much mathematical formality [12].

**Simulating network traffic load**

Different studies have been published to characterize network traffic and results in many different profiles. Out of these, some recurring characteristics can be identified. In [13], the authors identified two trends of internet characteristics, *high variability*, and *self-similarity*. Sets of values with high variability have an infinite mathematical variance and discontinuous values can always occur [13]. High variability can be described by a heavy-tailed mathematical distribution where generated samples will have high variance [13]. High variance means that most values generated are low, but a few are very high. Self-similarity means that any small sub-range of values will be similar to the whole range of values [13]. Self-similarity causes a burst behavior of the network and is also described by a heavy-tailed distribution [13].

Simulation of agents sending or receiving data in a network can be done by using two heavy-tailed distributions to describe the characteristics of high variability and self-similarity, one for the size of data and one for the time between each data burst [13]. The agent draws a random sample from each distribution and creates data with the size of the first distribution, and then waits the amount of time drawn by the second distribution and repeats [13]. In [13], the authors call this the ON-OFF source generation algorithm.

Since cellular networks are packet switched nowadays, they should have similar behavior and characteristics as in IP-networks. The model mentioned above will be useful when describing UE data characteristics.

**1.3.3 Software development methodologies and frameworks**

The most extensive task of this thesis is the development of the simulator. This section will describe methodologies and frameworks for development relevant to this thesis.
Test Driven Development

The methodology known as Test Driven Development (TDD) is a test-first approach. TDD started being used with the agile movement as a part of eXtreme Programming [14]. TDD is one of the most controversial agile practices regarding programmer productivity and software quality [15]. The basic idea is that before any code is written for a given functionality, a test is created for it [14]. Since the functionality is not yet developed, the test will and should fail. To write the tests before the code is not enough to describe TDD, however [15]. The TDD workflow consists of short cycles of writing tests and just enough code to make the test pass and refactoring later. Different studies have tried to compare TDD to Test-Last approaches, but the results are inconsistent [15]. The authors in [15] argue this might be because TDDs effectiveness is contextual and that the studies only looked at Test-First methodologies and not complete TDD. In [16], they showed that very few programmers follow a strict TDD process, even in projects which are claimed to use TDD. In [15], they conclude that we do not know whether TDD delivers what it promises regarding software quality and programmer productivity, and it is more complicated than just the order of which the tests are written. TDD will, however, result in small development cycles, dividing tasks into small pieces, and also produce unit tests for the functionalities implemented.

Build systems

A build system uses a set of inputs to create a packaged program as output [17]. Often a build system has more purposes, like keeping track of dependencies, running tests, creating documentation [17]. For Java, the most common build systems are Maven, Gradle, and Ant [17]. The purpose of using a build system is to make life easier for the developers or researchers to contribute or use new projects without the need to spend hours setting it up [17]. In an ideal scenario, the person can download the source code and then build the program with one command [17]. Without a build system, the person would have to install every dependency manually, setting up a build path, and running the compiler to be able to run the program.

In [17] they provide empirical evidence that even with a build system, as many as 38% of the Java projects on GitHub still fail to build. The projects tested were
chosen based on evaluation criteria that it should be open source (which would contain a license file), forked at least once (which indicates contributions are encouraged), and contained a recognizable build system file, resulting in 7,264 projects. Then, they categorized the build faults and tried to make associations between build success/fail and project properties. Dependencies and compilation errors caused most of the build failures. Out of the three build tools, Ant, Maven, and Gradle, Maven had the least failures, but also the largest sample of projects. Factors that affected build failure rate are higher file count, project age, and days since the last update. Their study shows that the use of a build system will not automatically fix every problem regarding cooperation in development, but it can still be advantageous if used correctly. Since the simulator very likely will be under continued development after this thesis, it is vital that it is easy to set up.

Ant was not considered when deciding upon build tools since it is the oldest and not used in new projects anymore. The remaining choices were then Maven and Gradle. The advantages of Gradle is its flexibility since one can write own build scripts with Groovy, and excellent performance [18]. The main drawbacks of Gradle are that it is difficult to learn compared to Maven and that updates might not be backward compatible with existing plugins [18]. Maven is simple to use but loses in terms of flexibility and performance [18]. The pom.xml files of a Maven project tend to be harder to read as well [18]. In [18], the author describes his personal experience of moving two projects from Maven to Gradle, and why they decided to go back to using Maven. The author recommends considering the existing knowledge of any of the frameworks and workflow within the team and the complexity of the build when choosing a framework. No tool is the best, but there is a tool that is the best choice for each project.

**DESMO-J Framework**

DESMO-J is a simulation framework in Java. The acronym DESMO-J stands for Discrete-Event Simulation and Modelling in Java, which describes the framework well. It is an object-oriented framework used to create simulation models. The acronym also highlights that it supports discrete-event simulation. Discrete-event simulation models the system as a sequence of events at instants in time. In-between these instants, the state of the system is assumed to remain unchanged,
and it can skip ahead in time. This skip in time means that the simulation can run faster than a continuous system since all time slots are not required to be simulated and it can skip ahead until the next scheduled event. A network simulator can utilize these features. [19]

The implementation is either event-driven or process-driven. Events are scheduled to occur at a specific time in the simulation, and an event can schedule new events [20]. Processes are running continuously during simulation [20]. However, they can be inactivated, either until a specific time or until another process, or event, activates them. When reactivated, they continue their lifecycle method at the time of deactivation. They are not mutually exclusive, meaning the user can schedule events to start processes or processes to schedule events [20]. A limitation of using processes in DESMO-J is that the implementation is relying on java threads [20]. There is an upper limit on how many threads can be created in java depending on the Java virtual machine (JVM) and available memory, even if they are not running simultaneously in parallel [20]. A couple of thousand processes might work, but if millions are required, this will not be allowed, and the user is required to use the event-driven approach [20]. In [20], they propose an alternative using Apache Library JavaFlow instead of java threads if the process approach is preferred, and allowed them to run up to 2.5 million concurrently running processes. JavaFlow allows running multiple mutually exclusive coroutines under the same thread [20]. In telecommunications, a massive number of entities is acting upon the model, and most of them are tricky to implement based on events [20]. How the simulation framework is used should be taken into consideration before the implementation, to avoid running into performance problems.

**JavaFX**

JavaFX is a set of libraries with graphics and media packages integrated into the Java platform. This integration means it can run on all platforms with Java SE Runtime Environment or Java Development Kit. JavaFX uses a XML-based declarative markup language, FXML, to construct application user interfaces. The FXML files can either be coded directly or generated by using the JavaFX Scene Builder application that includes drag-and-drop functionality to create a user interface. [21]
1.4 Method

This method section describes what was done during this thesis project. It is divided into phases of the project, with different goals and results. These phases were the requirements, design, implementation, and evaluation. The last two subsections will describe the frameworks used during the development of the simulator.

1.4.1 Pre-study

The existing simulator is developed with the simulation framework DESMO-J [19], and to get familiarized with the framework, the author followed one tutorial to get accustomed to it. Since cellular networks and simulation were mostly unknown, papers and books from the field were read by the author, which are presented in section 1.3. Topics of studies relevant to the thesis are simulation of RAN networks, network load balancing, and traffic load modeling, to mention a few. The books “From LTE to LTE-Advanced Pro” [2] and “5G NR: The Next Generation Wireless Access Technology“ [1] were a great start to get acquainted with 5G. Another vital step to gain an understanding of the problem was to get accustomed to the existing codebase, which was done by reading the code, creating class diagrams, generating documentation of the code. One good source to understand what was done by the previous developers was their thesis report, which contained much valuable information. This step was vital to familiarize the author with the concepts of simulation, cellular networks, and the existing codebase.

1.4.2 Defining requirements

Before the implementation and design could be started, a definition of what the simulator should be capable of was needed. For this, a requirements document was formed. The entities to be added were identified, and basic requirements on these entities were then added based on the pre-study. However, to ensure the resulting product would be usable by the stakeholders, this was not good enough. The stakeholders of this project were the employees at Ericsson systems department.
There were ongoing discussions with the stakeholders throughout the whole project and especially during the start of it. Definitions of use-cases, functionalities, and goals of the simulator were defined during this phase of the project. This phase was revisited multiple times, as new requirements arose while the development progressed.

1.4.3 Designing the system

Since the new features had to work with the existing codebase, there were many constraints on the design process. The decision made by the former developers of the simulator were hard to change since many dependencies had been introduced. During this phase, sketches of the classes to be implemented were designed, including which fields and functions they should have. Unified Modeling Language (UML) class-diagrams, entity-relationship (ER) diagrams, and activity diagrams were created in this process, which can be seen in Chapter 3 and Chapter 4. These documents gave a good overview of how the communication between the classes would work. Most of the classes already implemented before this thesis were left out of the class-diagram to increase readability. Instead, an ER-diagram was created to get a better overview of all the entities for this simulator. This phase and the implementation phase were iterated back and forth as the author learned more during implementation and realized some solutions were not well thought through. The requirements were also revised a few times during the process.

1.4.4 Implementing simulator

The development of desired features for the simulator was the most substantial part of this thesis. The code base was large and complex from the start, and to avoid making it even more complex, a process was needed to be followed. To avoid depending on the legacy code to run the new functionality, including a test seemed like the right decision. With unit tests, each functionality can be isolated and tested alone, instead of running the whole simulator. To enforce the creation of tests, TDD is one possible approach. TDD would also encourage small cycles and dividing problems into small tasks. For those reasons, it seemed like a suitable methodology for this project.
For the workflow, one entity was handled at a time. First, a test was created for one requirement of the entity, and then the implemented to pass the test was written. The implementation followed the design documents created as much as possible. In some cases, a revision was needed in the requirements, the design, or both, to fit the functionality. There were some difficulties in unit testing with the frameworks used in this thesis. JavaFX and DESMO-J were not suitable in some cases and led to not working according to TDD for everything.

The implementation phase also included deployment, which led to the introduction of a build system. A simulator built upon many frameworks does need a build tool to keep track of the dependencies. The selection is described in section 4.5.

1.4.5 Evaluating the system

Both during and after the implementation, evaluations of the system was performed to ensure it was progressing according to the specifications. This section describes the methods of evaluation.

Feedback
As a part of evaluating the system, consecutive weekly meetings with the stakeholders were held. During these meetings, a short demo was usually held where the product owners could make requests for areas of focus, and how functionalities develop to make sure it went according to plan. If something should be changed, the cycle was revised from requirements to implementation, keeping short feedback loops. This setup was similar to a methodology like SCRUM-sprints but less formal.

Testing
Another essential evaluation step was the testing of the system, which also was a part of the implementation. The tests were created as a byproduct of TDD. Most tests created were connected to one or more of the requirements identified and made sure that each entity worked as intended. The test cases are summarized in 4.4.1.
Model validation
In this thesis, there was an attempt to validate that the new traffic model that includes UEs is more accurate than the previous traffic model. However, due to lack of time, the tests could not be performed according to the descriptions in the literature. The plan was to perform a Turing test. The setup would include an expert, data generated from the new traffic model, data generated from the old traffic model, and data generated from a real UE. Two tests would be had, one comparing the new and the old traffic model, and one comparing the new traffic model with real UE data. The expert would be given two graphs of the size of sent or received data at given times. The task would be to identify which graph was generated by a simulation. If the new traffic model had higher success than the old traffic model, meaning it was identified as a simulation the least, it would mean that the accuracy of the traffic model had improved. In the old traffic model, it was not a UE that generated data; it was a load generator that affected a RAN node directly. Since each node is expected to handle 1000 to 10000 UEs, the data would need to be from 1000 to 10000 real UEs to make it a fair comparison of traffic models. There was not enough time to acquire this data from real UEs.

Instead, a naïve attempt at validating the model was performed. Using the Android application GlassWire\(^2\), the author could record the network activity of a mobile phone. GlassWire created graphs of data sent/received at given times as was needed, but only for one UE which was not sufficient to perform the Turing test described above. With this graph, the new traffic model could at least be compared with one single UE. If the appearance seems to be similar, it can give a hint if it is an improvement. To make any conclusions of model validity, a more precise method is needed, like the one described above.

Performance
The last step of the evaluation of the system is performance testing. The focus on this was to see how the simulator performs when increasing the size of the model. Since there was no point of reference in performance, comparing it to itself will at least show how it scales when the input grows larger. The method of these tests was to set a fixed value for all simulation parameters, create a network with one node with one cell, one core network connection, and one link. The simulation time was set to six minutes. Then a simulation was run two times with each set up while altering the number of UEs. The least and highest number of UEs was 1 and 10 000 respectively. The time to run a simulation was measured with the system clock, and the mean of the two runs was recorded in a graph. The performance effects of nodes, cells, roads, or links were not tested due to lack of time.

1.5 Main content and organization of the thesis

This paper is structured as follows.

Chapter 1 is an introduction to the subject and defines what this thesis wants to find out. It also describes related work. One section describes the method of this thesis, including frameworks used for development.

Chapter 2 describes the requirements for the functionalities that are developed during this thesis. A description of the goal of the system is also included. One section of chapter 2 will build the theoretical framework for what needs to be known to understand the decisions made in this thesis.

Chapter 3 will give the reader an overview of the design of the system functionalities added. Here, class diagrams and an architectural overview will be shown. Furthermore, some design constraints will be described.

Chapter 4 will describe the implementation and validation of the system. This chapter includes activity diagrams for processes in the simulator and describes some test cases. The results of the model validation and performance tests are presented in this chapter.

Chapter 5 will discuss the project as a whole, including what has been achieved and what should have been done differently. The ethical and environmental considerations of this thesis work are discussed, as well.
Chapter 6 concludes the thesis and summarizes the contributions and achievements. This chapter discusses future improvements of the simulator.
Chapter 2 System Requirement Analysis

This chapter will describe the goal of the system that is to be implemented. The 5G architecture will first be introduced to understand the decisions and constraints set by the requirements. The use-cases of the system are then identified in a diagram, which leads to describing the main functionalities. Finally, the functional requirements are presented in a table.

2.1 The goal of the system

The goal of developing the network simulator system is to have a framework that allows the user to plan proposed network layouts by testing the performance. The simulator will allow the user to test different network configurations for a given area and evaluate how they perform. Having this tool is much cheaper than the alternative that is installing the network and evaluating it afterward. It already exists many simulators for a similar purpose, but this simulator focuses on the RAN part of the 5G network. Before this project, it was already possible to model networks with nodes and links in the simulator. The load was set on each node by the user, and the accuracy of this traffic model can be questioned. In an attempt to make the traffic model more realistic, the User Equipment (UE) is to be modeled as well and let UEs create the load. By knowing the population of an area, the number of UEs can be estimated to quickly create a model of the area to generate the load on the RAN.

2.2 5G Architecture

It is essential to understand what is being simulated to define requirements for the system. This section will, therefore, explain the 5G architecture.
The 5G network consists of two main parts, the Core Network (CN) and Radio Access Network (RAN) [1]. Devices connected to the network, such as smartphones, sensors, or even smart fridges, are called User Equipment (UE). The architecture is illustrated in Figure 3.

2.2.1 Core Network

The CN is responsible for the functionalities of a complete network that is unrelated to the radio access. This functionality includes authentication and setting up end-to-end connections. The CN also set up a connection to the internet for the RAN. The separation from RAN allows one CN to serve multiple RAN. [1]

The simulator developed in this thesis puts less emphasis on the CN than the RAN. The only thing it considers is that the RAN needs a connection to the CN, which is the data source in the simulator model.

2.2.2 User Equipment

The term UE is more general than the term phone and is needed since it is predicted that the majority of connected devices to the 5G network will not be phones [3]. Different kinds of UE have different data sending and receiving patterns and different needs, which has led to three categories defined by the 3GPP [1];

- Enhanced mobile broadband
- Massive machine-type communication
- Ultra-reliable and low-latency communication
Enhanced mobile broadband is the evolution of the current cellular network, handling larger volumes of data and supporting higher data-rates. The category Massive machine-type communication target low-energy devices used in, for example, monitoring, like sensors. The essential factors for this category are that devices must be cheap and have long battery life. The last category, ultra-reliable and low-latency, is aimed at services where safety is a critical factor. Traffic safety and production automation come to mind. In the last two categories, data volumes are not as significant as in the first category. [1]

Different kinds of UE do not necessarily have the same needs, and this must be considered when simulating them. Some UE sends data irregularly and moves a lot while others send at a constant rate statically located.

### 2.2.3 Radio Access Network

The RAN is the connection between the UE and CN, responsible for the radio access communication. This network is the focus of the simulator to be developed. The RAN consists of nodes, links between the nodes, and cells. A node is a baseband unit that is connected to at least one antenna unit [4]. A baseband unit is responsible for functionality such as converting data packets to radio signals for the antenna units and processing the signals from the antenna units [6]. The antenna unit might be located remotely or at the same location as the baseband unit [4]. The antenna unit creates one or more cells, which is the area it can cover with a radio signal [4]. For a UE to connect to a node, it must be within the cell’s coverage area. The nodes can also connect with other nodes, either with a fiber cable or a radio microwave link [4].

As mentioned in the introduction, using fiber cable is preferred for the links in terms of performance. However, they are sometimes not feasible to install because of the high costs or obstacles in the terrain. Adding the increase of nodes in a network to the equation makes fiber cables a less attractive alternative. Having fewer fiber links complimented by many microwave links would allow the data to take different routes for the same goal, and therefore increase the overall capacity of the RAN.
2.3 The functional requirements

This section describes the process of creating functional requirements. First, the use cases were identified, which led to listing features needed. Then, the functional requirements and non-functional requirements are listed.

2.3.1 Use cases

Below in Figure 4, a use case diagram shows the functionality of the simulator through the end users’ perspective. The simulator will have a graphical user interface that supports creating a model of the RAN under investigation. The user shall be able to create nodes, links, cells, roads, and UEs, set their properties, custom scripts, and run a simulation. When the simulation is complete, the user shall be able to view the results.

![Use case diagram of the simulator](image)

*Figure 4 Use case diagram of the simulator*

2.3.2 Features

From the use cases, four main features are identified that should be included in the system.

**F1.** The system shall have the entity User Equipment (UE)

**F2.** The system shall have a positioning system for entities.
F3. The system shall have the entity Cell.
F4. The system shall have the entity Road (to allow mobility of UEs).

Existing features
These features already exist in the simulator but will be listed since they might need to be changed in the requirements to implement the new features.

F5. The system shall have node entities.
F6. The system shall have the entities links, where one link connects two nodes.
F7. The system shall produce results of the simulation.
F8. The system shall allow custom scripts to be added.

Below, all the entities of the simulator are illustrated in Figure 5.

2.3.3 List of functional requirements
The functional requirements to implement requested features are described in Table 1. In the following section, the terms “user” describes the end-user of the simulator, not to be confused with the entity User Equipment (UE).
### Table 1 Functional requirements

<table>
<thead>
<tr>
<th><strong>F1</strong></th>
<th><strong>User Equipment (UE)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FR1.1</td>
<td>There shall be possible to run multiple simulation runs with the same UE setup. In the multiple runs, the UEs shall behave the same.</td>
</tr>
<tr>
<td>FR1.2</td>
<td>A UE shall be associated with one road and can only move along that road.</td>
</tr>
<tr>
<td>FR1.3</td>
<td>A UE shall have a position, that corresponds to a location on the positioning system.</td>
</tr>
<tr>
<td>FR1.4</td>
<td>A UE shall have a velocity that will determine how far it moves each time loop, depending on how much time elapsed.</td>
</tr>
<tr>
<td>FR1.5</td>
<td>Once a UE meets the end of a road, it will select a new road. The road selected must be connected to the same endpoint. It selects by random and prefers not using the same road twice.</td>
</tr>
<tr>
<td>FR1.6</td>
<td>When the simulation system is running, every UE will update its position once every given time interval. The user shall be able to set this time interval.</td>
</tr>
<tr>
<td>FR1.7</td>
<td>The UE shall have a data profile which is one or more distributions of probabilities of sending and of how much load given a time.</td>
</tr>
<tr>
<td>FR1.8</td>
<td>The system shall allow the user to create UE profiles, that is saved between simulation runs.</td>
</tr>
<tr>
<td>FR1.9</td>
<td>The UE shall create a data load at nearest DataGen-node, addressed to itself when the data profile of the UE requires data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>F2</strong></th>
<th><strong>Positioning system</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FR2.1</td>
<td>The positioning coordinates shall be measured in a unit, like meters.</td>
</tr>
<tr>
<td>FR2.2</td>
<td>The positioning system shall be able to translate pixel coordinates in the GUI to coordinates in the positioning system. The user shall be able to set this pixel to measuring unit scale.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>F3</strong></th>
<th><strong>Cell</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FR3.1</td>
<td>The user shall be able to add cells to existing nodes in the simulator model. If no node is selected, an error message will show instead.</td>
</tr>
<tr>
<td>FR3.2</td>
<td>A cell must be associated with a node.</td>
</tr>
<tr>
<td>FR3.3</td>
<td>A cell shall have the shape of an arc and must have the properties radius, start angle, and the length of the angle in degrees. The arc must share its center point with the associated node. The arc determines the coverage range.</td>
</tr>
<tr>
<td>FR3.4</td>
<td>The cell must have a function that determines if a UE is within its coverage area.</td>
</tr>
<tr>
<td>FR3.5</td>
<td>The cell shall have the property throughput, which is the maximum capacity it can handle.</td>
</tr>
<tr>
<td>FR3.6</td>
<td>If the maximum throughput is reached, data from new requests cannot be sent to the UE. Thus the system shall drop the data. If the cell fits some, but not all data, it shall send some and deliver a degenerate service.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>F4</strong></th>
<th><strong>Road</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FR4.1</td>
<td>The user shall be able to add roads to the simulator model with the help of a graphical user interface.</td>
</tr>
</tbody>
</table>
FR4.2 A road has a start and endpoint, both of which have coordinates in the positioning system.

FR4.3 The start and endpoints can be shared between many roads, thus connecting the roads.

FR4.4 The user will be able to move a start/endpoint in the UI, which will alter all the roads connected to that particular point.

FR4.5 A road must have a length property. This is used to calculate the UE movement along the road. The length can be set by calculating the distance between start and endpoint.

FR4.6 The road shall have a maximum velocity of which a UE can move along it.

FR5.1 The entity node must have the attributes latency and throughput. If the throughput maximum capacity is full, it will drop the load. If some of the data-load can be delivered, but not all, a degenerative service will be provided.

FR5.2 Nodes can be a Service Gateway (SGW), meaning it is a connection to the core network. SGW-nodes create data-load that is sent to the UEs once requested.

FR5.3 A Node shall have the shape of a circle with a fixed radius. The nodes center point must be available to other entities.

FR6.1 The system shall have the entity link, which is a connection between two nodes.

FR6.2 A link shall have the attributes; maximum throughput, latency, and cost.

FR7.1 The system shall produce visualized results of the simulation — graphs of requested and resulted packages, and total latency for UEs.

FR7.2 The system shall produce graphs during the simulation run, updating the graph when data is produced.

FR7.3 The system shall produce a result file that can be read by excel.

FR8.1 The system shall allow the user to control the simulation by adding custom scripts. Nodes, links, or cells must be controllable and allow the user to turn them off at a specific time in the simulation.

FR8.2 The system shall allow the user to add custom routing algorithm to nodes with scripts.

As seen in the table, cells do not operate at a frequency, which is a stated delimitation of this thesis. However, they do have limitations regarding capacity. Nodes and links have the same limitation. The logic for this simulator is to allow the network to drop parts of the data to a degenerative service. If many UEs compete for the same bandwidth, one UE at a time will get its data for as long as
there is still bandwidth left. There is no priority order; the simulation framework determines the order.

2.4 The non-functional requirements

NF1. The simulator system shall be implemented in Java.
NF2. The Graphical User Interface (GUI) shall be implemented in JavaFX.
NF3. The simulator framework shall be implemented with DESMO-J.
NF4. The GUI shall be intuitive and allow the user to work efficiently.
NF5. The simulator shall use an appropriate build system that makes it easy for new users to download, install, and run the simulator.
NF6. The simulator shall use a configuration file to store the default parameters and settings.
NF7. The simulator shall be light-weight and runnable on a regular laptop.

2.5 Brief summary

Demands on 5G networks will change how the RANs work. A RAN simulator is needed to estimate the capacity and make sure it can handle specific scenarios. The simulator program is required to allow the user to model such a network efficiently. The entities needed in the model are nodes, links, cells, UEs, and roads.
Chapter 3 System Design

This chapter will first describe the system architecture of the simulator. The entities used for modeling the networks will be presented, including the relations and attributes they have. Then, the design of the introduced classes will be illustrated, going into more detail for roads, cells, and UEs. Lastly, the key techniques used to solve design problems will be identified.

3.1 System architecture

Since this thesis work is to extend an existing simulator, there was not much choice in designing the architecture. Instead, the decisions made was how to tweak it to fit the requested functionality.

The simulation system itself is run locally and requires no external machines to work. The architecture follows a model-view-controller (MVC) pattern, as seen in Figure 6, where three components make up the model. The view is a graphical part of what the user sees. The controller is what allows the user to change the simulation model by adding objects to the object handler or run a simulation by activating the system controller. The simulation core will run a simulation which will affect the view, and in the end, produce a result which also can be seen by the user.

Figure 6 System architecture
3.2 Entities overview

An entity-relationship diagram was created, as seen in Figure 7, to visualize the relationships of the entities of the simulation model. This diagram describes the entities used to create a simulation model, where squares represent an entity, rhombus' represents a relation, and ovals represent an attribute. Nodes and links were already implemented before this thesis, but they are included to show their relations to the rest of the entities. In addition to the entities identified in Figure 5, road points are added to represent the start and ends of roads. Since each point allows multiple roads, they are used to connect roads into networks of roads.

3.3 Class design

Many new classes were introduced to extend the functionality of the simulator; some of them presented in Figure 8. All entities have multiple objects associated with them with different purposes. The entities not presented in the figure have a
similar structure. A naming convention has been followed to keep track of the purpose of the class. If the class ends with *Info*, it is the main class of the object that holds all the information. Updates should go through the *Info*-class object before any other object, to make sure the information is up to date and reachable by all objects associated with the entity. The observer-pattern is implemented for all info classes to allow every object associated with an entity to get all updates, but more on this in 3.4. Classes named *Handler* is responsible for the graphics and handle user interaction with the object. They share an object called *mainArea*, which is a pannable canvas where the objects can be placed. Classes named *Process* are responsible for the interactions with the simulation and inherits from the class *SimProcess* from DESMO-J. Process classes implement a lifecycle method that controls what the entity does during simulation time. The inherited class gives support to inactivate the process until a specific time instant or until activated by another process.

**Figure 8** Class specifications and dependencies.
3.3.1 Road

The entity road is not representing a real road but rather a path in which UEs can move. The name road is used as path might be confused with functionality regarding routing which uses the name path. A road is created with two road points as input. The two road points mark the start and end of a road. Many roads can be connected to the same road point to create a network of roads. The road entity contains an info class, and a handler class but no process class. The process class is not needed because a road will not change during a simulation. UEs will use roads, but not change them, and in this simulator, there is no limit on the capacity of a road. Not having a limit is a simplification to make the simulator more lightweight since it is required to run on a regular laptop. The road has the properties distance, start-point, end-point, and maximum speed which are stored in the info class. The handler can be used to edit a road and to delete it. By editing a road, the distance can be changed, which updates a pixel-to-meter scale. The road points still set the pixel coordinates of the start and stop of a road. The road points have a similar design with a handler and an info class. The road point handler allows the user to move the road points, select a road point, and to delete them, while the info class holds the position and a list of connecting roads.

3.3.2 Cell

A cell is created with a node as input, which is required to keep track of the position of the cell and where to route the UE data. It contains an arc object that represents the coverage in the positioning system. The arc has the parameters radius, start angle, length, and use the same center point as the node which the cell belongs to, which can be seen in Figure 9. Also, the cell has a maximum capacity of throughput and latency for transmission of data. It will not account for which radio frequency it operates on, obstacles in the line of sight, or radio waves bouncing off objects. A UE inside of a cell has full reception and can use the throughput regardless of the conditions mentioned. A cell is made up of an info class, a process class, and a handler class for the graphics. The info class holds all the parameter values for the object. The process runs during the simulation and keeps track of how much load the cell is under, and also saves the packets routed through it to produce graphs of the results later. It implements the ResultProducer interface, which includes
functions for getting the needed data to produce graphs. The handler is used to alter the parameters of the cell and delete the cell.

![Diagram of a cell arc](image)

*Figure 9 Properties of the cell arc*

### 3.3.3 User Equipment

The UE entity is the most complex of the entities introduced in this thesis. It consists of an info class, a handler class, and two process classes. One of the process classes is used for movement of the UE, and the other is used for demanding data. A UE is created with a road and data profile as input. A UE is always associated with one road, to restrict where it can move. The info class stores parameters related to the UE such as movement speed, position, data sending behavior, goal point (a road point which the user aims to reach). The handler allows the user to interact with the UEs by editing the parameters or deleting the UE. The handler also creates the graphical object, which is a canvas where an object looking like a cellphone from the '90s is drawn. The first process class determines when and how a UE moves and also creates and activates the second process class, which will determine when the UE wants data. The processes for UEs are described more thoroughly in 4.2.2. *UEDataFlowProcess* also implements the ResultProducer interface, to allow the user to view results from one UE of a simulation.

### 3.4 Key techniques

This section describes the difficulty points of this thesis and the techniques to solve them.

#### 3.4.1 The observer pattern

With the design choice of keeping up to three separate classes for each entity, it is crucial that communication does not fail. With the Info class being the source of
truth for each entity, it seems logical that every update should pass through the info class first. That led to the decision to use the observer pattern. The observer pattern enforces that any update on a given subject will trigger a notification on every observer of the subject [22]. Java has support for this in the java.util package with the interface Observer [23] and class Observable [24], which was also very convenient. The class Observable allows adding Observers to a list. Once an instance of the class is changed, it can run a method that notifies its Observers for a convenient means of communication. This communication was utilized especially for the UE entity, which both can move because of the simulation runtime or because of user interaction. In both cases, the simulation core needs to know the position as well as the GUI and to make sure it stays consistent it is only needed to update the info class which in turn updates both the handler and process class.

3.4.2 Software reuse

When given this project, the code base was already huge. However, much functionality was reused for multiple classes. One example is the classes for viewing results. For the entities link and node, there existed a corresponding class for viewing results. This class created a window with graphs for the simulation result. The functions almost did the same thing, except the input was a node or a link. The results view fetched the simulation data from a list within the node or link. When introducing cells and UEs, which also required a results view, it seemed ridiculous to write the same thing again. Instead, an interface, ResultProducer, was created. The ResultProducer enforced implementing a method to fetch the simulation result list of an entity but allowed each class to have its implementation. With this interface, only one results view class was needed with a ResultProducer as input instead, which could be any of the classes implementing it. A class implementing ResultProducer could be created, SimulationResult (which is not included in the class diagram), which allowed an easy to a method to fetch all the results of the simulation. On creation, SimulationResult fetched all UEs from the object handler and saved the results in a new list containing everything.

Much of the code related to the GUI could be reused as well since many entities share the same functionality written multiple times, but this has not been done
during this thesis because of lack of time. Reusing code will make the system much more maintainable and readable. When shared functionality is written multiple times, one update means multiple updates is needed.

3.5 Brief summary

The architecture of the simulator is simple because it is run locally. A model-view-controller pattern is followed. The entities of the simulation model consist of three types of classes, an info class, a handler class, and a process class, responsible for holding parameters, handling the graphics, and for making decisions and actions during a simulation run. The observer pattern is used for communication among the different classes.
Chapter 4 System Implementation and Testing

This chapter will go through the implementation and testing stage of the thesis. Firstly, the environment in which the development has been done is described. Then, the program flow charts of the most critical processes will be shown and explained. The user interface of the system is then presented. Lastly, the system testing and evaluation process are described.

4.1 The environment of system implementation

This section will describe the environment used for developing the simulator.

4.1.1 Technical conditions

- Eclipse was the integrated development environment (IDE) of choice for development.
- Java 8 was the programming language used for developing the simulator.
- FXML files were used to describe the graphical user interface together with JavaFX. They were either written in the IDE or generated by Scenebuilder.
- The unit tests were written with the testing framework JUnit and the mocking framework Mockito.

4.1.2 Development and testing environment

- Operation system: Windows 10 Enterprise
- Manufacturer: HP
- Model: HP EliteBook 840 G5
- Processor: Intel Core i5-8350U (4 cores, 8 threads)
- RAM: 16GB
4.2 Key program flow charts

This section will describe the program flows of the simulator. First, the main program flow is described, and then the flow for separate entities are described.

4.2.1 Main program flow

When the program is started, it will load an FXML file which specifies how the GUI looks and opens a window accordingly. A controller is loaded to allow the user to interact with the GUI. An initial empty model is then created. Then the flow is reactive, and its behavior is dependant on user actions. The user has the option of creating new entities, set parameters, import/export the model from/to an XML file and start the simulation.

When a simulation is started, all entities are fetched from the object handler in order to create a process for each. Then simulation time is set, and every process gets to run its lifecycle method. When all processes are idle waiting to be awoken or for a specific time instant, the simulation model increments the simulation time. When the simulation time reaches the simulation duration, the simulation terminates, and all processes are stopped. This overhead functionality is handled by the DESMO-J framework, and the developer mostly needs to focus on implementing the entity processes.

The results can be shown after the simulation is complete if the user opens the result view. If the user wants to run a new simulation with the same setup, the simulation should be reset. A simulation reset will delete the whole model and its processes and then create a new model.

4.2.2 Simulation process flows

UEs is the most important entity for every simulation, as it is the UEs that create the load that the network reacts to. It was also the scope of this thesis, to implement UEs, which is the reason the main focus of this section is to describe how the UEs program flow works.
UE process
Initially, the UEs were designed with a single process class, responsible both for data transmissions and movement of the UE, as seen in Figure 10. This solution could sure have worked, by letting the info class of the UE keep track of when to send and move. This solution does, however, not utilize the strengths of the DESMO-J framework. By separating the data transmissions and movement into two separate processes, they can be set passive instead of querying for tasks. Passivating would not be possible with both the move- and send-logic in the same process since a UE can move when not sending and send when not moving. The need for separation was realized when the traffic model for UE data transmission was implemented. The separation seemed like an appropriate change, which also would improve the readability of code consequently.
In Figure 11, the separated implementation of the UE traffic model can be seen. The traffic model is an implementation of the ON-OFF source generation algorithm described in 1.3.2. With the separation from UE movement logic, the process can be inactivated in-between sending a burst. As we can see in the figure, it utilizes two mathematical distributions. These are long-tailed, Paleto distributions, which have very high variance. The distributions were gotten from Ericsson and will not be described in detail. To implement a custom distribution with the help of DESMO-J framework, the uniform distribution, which samples a random number between zero and one, can be extended. This random number was the input to the
Quantile function of the distribution to be implemented and returned when the sampling function was called. This process will start inactive for as long as a double generated by the inactivity distribution, then send a packet burst. Before sending it will set the size of the burst by generating a double from the other distribution. It will also set up a connection to a cell if there is none already. If the UE is out of coverage, it will not send the burst but still, note that its given load could not be sent. Otherwise, the packet burst is routed through the network (from the closest SGW). The process will wait for the burst duration time, which is a variable set by the user. The packet is then saved for statistics, and its load is removed from the network.

![Activity diagram of UE flow process. This process is a modified implementation of the ON-OFF source generation algorithm](image)

The process UEProcess is responsible for movement and has a similar but more simple activity flow, as seen in Figure 12. Similarly, it uses two distributions, one
to determine movement speed and one to determine for how long the UE will keep the speed. The UE will always have a road point as a goal, and speed in x and y is updated accordingly to reach its destination. The UE will update its position in a frequency set by the user (default is once every second of the simulation time). After the update, a check is done to see if the goal was reached, and in that case, the UE needs to set a new road to follow with a new goal. To get variation, an event to change the movement speed is set at the start. Once the event occurs, movement speed will be set to a new value, as well as a new event for changing the speed will be scheduled. The UE can also be static, for example, if the device is a sensor or just other IoT devices, and in that case, this process will be inactive to prevent the UE from moving.

Figure 12 UE Movement Process

4.3 Key Interfaces of the software system

The simulator has a simple graphical user interface (GUI). The focus of the work is not to create something that looks appealing. The goal is to create a powerful tool that allows the user to work efficiently. Many simulators typically have a
command-line interface or use input files instead of a GUI, but having a GUI should hopefully make the simulator more user-friendly with a lower learning curve since the user receives visual feedback. The user is not required to specify the model in an XML file for example. The rest of this section shows screenshots of the simulator to demonstrate how the GUI looks and works.

In Figure 13, the main window of the simulator is shown and is also what the simulator looks like on startup. From this view, the user can start creating a model by adding nodes, roads, links, cells, and UEs. From the file tab, the user can import saved models, export the current, or remove the current setup. The user can change settings and edit the UE data profiles in the edit tab. Other buttons allow the user to set the simulation duration and start it. The area in the middle is the main area, which can be used to zoom in or out and place objects.
In Figure 14, the window for creating UEs is shown. A drop-down menu is used to select a UE data profile. The profiles can be edited as well. The user can create many UEs at once by using the same parameters and profile, to make the modeling more efficient. The profile window is not shown in a figure because it contains sensitive information, but it contains parameters for setting the distributions used in the ON-OFF source generation algorithm.

In Figure 15, the main window is shown once again, but this time populated with a model. The green circles represent radio nodes, and the lighter green represents a service gateway. Gray dotted lines represent links between nodes. The transparent blue arcs represent cells. The thick and solid gray line represents roads, and the gray circles on the ends are road points. The blue square with a black antenna represents UEs.
In Figure 16, a result graph is shown for data routed through one cell during a simulation. The x-axis represents the simulation time in seconds. In this example, a simulation of 6 minutes was run. The green line displays maximum throughput the cell can handle. The yellow line displays the data that was successfully routed through the cell to the UE. The red line shows the data that was requested. Since the red and yellow line represents the same thing when the transmissions are successful, we only see the parts of the red line where the data could not be sent. In this example, we can see that data requested larger than the maximum throughput of a cell was dropped.
Figure 16 Data routed through a cell during simulation

Figure 17 Latency for packets routed through a cell
In Figure 17, we see the resulting graph of the latency of packets transmitted through a cell. In this example, the latency of the entities of the network did never change and therefore showed the same latency when a packet is sent.

### 4.4 System Evaluation

This section will describe the steps taken to evaluate the system created. It will describe the test cases created, which mostly consists of the unit tests that were created during the implementation as a product of TDD. Then the implemented traffic model will be compared to real UE data and the load generator that is replaced by UEs. Lastly, the system performance tests will be presented, where the scalability of the system is evaluated.

#### 4.4.1 Testing

As mentioned in 1.4.4, test-driven development was used during the implementation stage. Down below in Table 2, most of the test cases produced is described.

*Table 2 Test cases*

<table>
<thead>
<tr>
<th>Test name</th>
<th>Functional Requirement</th>
<th>Description</th>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>UE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Constructor test</td>
<td>F1, FR1.2</td>
<td>Create a UE with a start road as input.</td>
<td>UE created with start position equal to the road start position.</td>
</tr>
<tr>
<td>Verify direction test</td>
<td>FR1.4</td>
<td>Update the speed of a UE.</td>
<td>Make sure that the vx and vy are correctly calculated to reach the goal eventually.</td>
</tr>
<tr>
<td>Recalculate speed on update test</td>
<td>FR1.4</td>
<td>Move the goal point of a UE.</td>
<td>The vx and vy are changed since the goal point has moved. This test will ensure the UE reaches its goal after updating speed.</td>
</tr>
<tr>
<td>Observer test</td>
<td></td>
<td>Add an observer to the UE, update the UE.</td>
<td>The observer is notified of change with a status of what was changed.</td>
</tr>
<tr>
<td>Movement test</td>
<td>FR1.4</td>
<td>Call the move function of a UE.</td>
<td>The UE moves in the correct direction and is now closer to its goal.</td>
</tr>
<tr>
<td>Test Type</td>
<td>FRs</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Find goal test</td>
<td>FR1.5</td>
<td>Call the move function of a UE with a speed that ensures reaching its goal.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The UE selects a new goal that is not the same as the old goal.</td>
<td></td>
</tr>
<tr>
<td>Set new goal test</td>
<td>FR1.5</td>
<td>Same as above, but the goal point connected to 2 roads.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The UE selects a new goal associated with another road.</td>
<td></td>
</tr>
<tr>
<td>Increment and set id test</td>
<td>-</td>
<td>Create two UEs.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The UE created last have an id of n+1 if the first UE has id n.</td>
<td></td>
</tr>
<tr>
<td>Pixel to meter scale test</td>
<td>FR2.1, FR2.2</td>
<td>Update the distance of a road, then set movement speed of a UE.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The UE speed is scaled according to pixel to meter scale, translating m/s into pixel/s.</td>
<td></td>
</tr>
<tr>
<td>Cell</td>
<td></td>
<td>Create a cell with a node as input.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR3.1, FR3.3, FR3.5</td>
<td>The cell is created with default values for angle, length, radius, and throughput.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR3.4</td>
<td>Call the function for coverage check with a position.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>True if the position should be within coverage, false otherwise</td>
<td></td>
</tr>
<tr>
<td>Observer test</td>
<td>-</td>
<td>Add an observer to the cell, update the cell.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The observer is notified of change with a status of what was changed.</td>
<td></td>
</tr>
<tr>
<td>Road Point</td>
<td></td>
<td>Create a road point with x and y position as input.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR4.2</td>
<td>A road point is created at the given position.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The observer is notified of the change.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR4.3</td>
<td>Add a road to the road point</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The road is now added to a list of roads for the road point.</td>
<td></td>
</tr>
<tr>
<td>Road</td>
<td></td>
<td>Create a road with two road points as input.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR4.2</td>
<td>A road is created between the two given points.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR4.2</td>
<td>Create a road with four double values as input.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Two road points are created with given values. A road is created between the points.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FR4.5</td>
<td>Set the distance of a road.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The pixel to meter scale is updated according to the new distance [m] compared to the pixel distance of the end and the start point of the road.</td>
<td></td>
</tr>
<tr>
<td>Road point observer test</td>
<td>-</td>
<td>Add an observer to the road. Change the position of the start point of a road.</td>
<td>The distance of the road is updated. The observer is notified of the change of distance</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Increment and set id test</td>
<td>-</td>
<td>Create two roads.</td>
<td>The road created last have an id of n+1 if the first road has id n.</td>
</tr>
</tbody>
</table>

The tests do not cover the simulation logic or GUI at all. The functional requirements connected to the GUI or simulation processes are therefore not covered by tests. Some tests, like the observer tests, are not related to a functional requirement but instead a consequence of the design decisions made in Chapter 3.

### 4.4.2 Traffic model evaluation

As mentioned in the beginning, the evaluation is not scientific. Its purpose is to tell if the model could be an improvement from the previous implementation, and if it is worth the extra hardware resources to simulate UEs for extra precision. It cannot be used to make any conclusions.

By comparing Figure 18 and Figure 19, which illustrates the load produced in a simulation by the traffic model with one UE and recorded data of a real UE, respectively, we can see some similarities in the characteristics. Both graphs have a majority of inactivity periods, but when it sends data, it is usually in bursts. The variation of burst size seems to follow a similar pattern. The different colors in Figure 19 are data for the uplink or downlink. Since the simulator currently does not differentiate uplink or downlink, they are added together in graphs of simulated data. Another note about Figure 19 is that the application used to create the graph rounds the lines. A tool to get the raw data for creating independent graphs would have been preferred.
In Figure 20, Figure 21, and Figure 22 results of simulating 1000, 5000, and 10000 UEs, respectively, are shown. High variation is still a trait, but with this many UEs, there are no periods of inactivity. The horizontal axis represents the time, and vertical the size of data sent/received. The values are cropped of for load since they are not of interest. It is the behavior, and values in relation to each other that is interesting.
Figure 20 6-minute simulation of 1000 UEs

Figure 21 6-minute simulation of 5000 UEs
In Figure 23, the results of a 6-minute simulation with the old traffic model is shown. The values vary a lot in this model as well but seem to be more consistent. The old traffic model is a simulated load that is put on a node, which should represent between 1000 to 10000 UEs. With real data of 1000-10000 UEs, these could be compared with the data presented in the graphs to see which one, of either
the old or new model, seems to be a more realistic representation, but this thesis will settle with guessing.

**Figure 23** 6-minute simulation of the old traffic model

### 4.4.3 System performance test

In Figure 24, the results from the system performance test are shown. It shows the simulation runtime for a 6 minutes simulation of a network with one node and one link. The results show much inconsistency up until 30 UEs but after that, the time increases linearly with the increase of UEs.
The linear behavior of the measurements acts linearly up to 4000 UEs, as seen in Figure 25. Figure 24 and Figure 25 have used the same methodology and parameters, but at different times, which shows that many factors affect runtime, since 250 UEs resulted in different runtimes. Within each of the figures, the conditions have been the same, running two times writing down the mean value of the tries.
During the tests, the system resource utilization was monitored as well, and the more extensive simulations used up a lot of them. However, it did not utilize all of the RAM available. The processor was not utilized 100% either, even during the simulation of 10 000 UEs.

4.5 Build system

As of one the requirements, NF5, the simulator shall have a build system. A build system is needed to make it easier to continue the development of the simulator after this thesis, as continued development is very likely. Before this thesis, the system contained both a pom.xml-file and a build.gradle-file, suggesting that previous developers had started setting up Maven and Gradle. However, none of the build scripts worked, and no instructions were provided in the README-file on how to do it. Luckily, one of the developers was working at the location where the thesis was performed and able to help set up the project in the IDE, Eclipse. The process to set it up was not straight-forward, and after discussions with the supervisor, it was concluded that a solution setting up and building the project was needed to allow further development and usage of the simulator.

The author did have little to no experience in either of the build tools Gradle and Maven. For this project, either tool would have been sufficient for its task, which is keeping track of the dependencies and having a script for running the simulator. The advantages of Gradle outweighed Maven with better performance, which led to the decision to go with Gradle. Having better flexibility did not matter now, but might in the future. Another perk of using Gradle is a much cleaner set up file. Some of the dependencies, like DESMO-J, did not exist in the repositories available, but Gradle allowed creating a custom library from a file path where the DESMO-J JAR could be placed and thus imported.

4.6 Brief summary

This chapter has described the implementation process as well as the evaluation of the system. The frameworks and technical conditions of the development environment are first described. Then the key program flows are presented. Focus is on the flow of the UEs, since implementing UEs has been the most crucial task
in this thesis. The UE uses two main processes, one for movement and one for data sending behavior, which is the traffic model implemented. The interface of this system is then described, which is only one, the GUI. The system evaluation is divided into testing, evaluation of the traffic model, and system performance tests. It shows that the new traffic model is a promising candidate for simulating UE traffic and that the system execution time scales linearly with the number of processes. Lastly, the reason that the build tool Gradle was used for deployment is presented.
Chapter 5 Discussion

This chapter will discuss this thesis as a whole, including the method and result of the requirements, design, and the final implementation phases. Other approaches and possible solutions will be discussed, as well as difficulties with the chosen method.

5.1 Requirements

The process for identifying the requirements was probably the hardest of this thesis. During this phase, the author's knowledge of 5G was still low in combination with no real idea of what could be implemented. It was essential to not set too high ambitions, given the time restraints. Much of this phase depended on discussions with the supervisors at Ericsson to identify what was the required functionality for the simulator. This process could have been more structured if the author had acquired more knowledge of simulation beforehand.

The biggest mistake in this phase was not to consider the performance and scalability of the system and define it as a requirement. Knowing that the system would require to handle at least 1000 UEs would have forced different solutions and might have led to better performance. Using SimProcess for every entity is limited by how many threads the JVM allows. A more structured method of identifying use-cases would have brought this to attention earlier in the thesis since the focus at that stage was if and how the functionalities could be created.

5.2 Design

Much of the design was decided upon already and difficult to change since the simulator was not built from scratch. Two people wrote the existing codebase during the same timespan as this thesis, and this also resulted in a large codebase. Redoing and changing in the design in the existing codebase was therefore out of scope because of lack of resources and time. Smaller changes were, however, necessary, but they did not cause any significant design changes.
The decision to split each entity into different classes was a correct decision, to separate functionality and made the code more readable. This structure makes it easier to know where to look for a given functionality.

### 5.3 Implementation

Although much of the implemented functionality is new, a majority of the development time is spent on integrating the new functionality with the old functionality. The integration was also the biggest challenge. As the authors of [5] mentioned, they did not have any prior experience of developing a GUI which led to a few suboptimal solutions regarding the GUI. Take the delete functionality as an example. Its purpose is to delete a single object of an entity in the simulation model. The old solution would clear out every object on the workspace and recreate everything, except for the deleted object, to remove an object from the workspace. While this solution does its job, it did not allow further development to extend the simulator with new entities and keep the delete functionality as it was. The better solution would be to remove only the object to be deleted, which is possible to do in JavaFX, which also was implemented. Another example is the import/export of a model to an XML-file, which of course would not work after introducing new objects to import/export. Rewriting old functionality to integrate new was surely time-consuming, but the alternative of starting from scratch would probably not have resulted in a complete simulator with the requested functionality during the time frame of this thesis.

Reflecting upon these problems with working on an existing code base show the author being unexperienced working in real software projects since this is probably a common scenario. There is a significant contrast when working in real on-going software project from tasks given by the university. In university courses, the student is often given a smaller isolated problem and required to solve it from scratch. This project was quite the opposite since it is a large problem but with the solution partly given. With the solution mostly given, there is probably no possibility to change it even when there are alternatives.
The second largest task related to the implementation was the UE, and especially its traffic model. The ON-OFF source generation algorithm seems like a reasonable estimation, and as mentioned in 1.3.2, a model should be as simple as possible but still able to act as the real system it represents. Compared to how the load was modeled before, we could see that it looks more realistic with UEs implemented in 4.4.2.

TDD proved to be a valuable method of development, especially in such a project with a large code base already. The test environment allowed a quick method of running the newly written code in isolation to make sure it is working as intended. Otherwise, the whole simulator would need to run to make sure it works. If failures occurred, there is challenging to know if it was because of the newly implemented functionality or some interaction with the old. It is always good to verify that the code runs as expected before integrating it with the rest. With that said, unit tests could not be created for everything, since both the simulation and GUI frameworks ensured complications when used with the testing framework JUnit. If the author had a better experience with testing, this could most likely have been solved within a reasonable time by using advanced mocking features. Instead, some functionality remains untested.

5.4 System evaluation

The test cases produced with TDD can by no means show that the system is absent of bugs. However, they are a useful tool when introducing new functionality to make sure no old functionality breaks. Since the tests only cover the new entities, there is no guarantee the older functionalities are working still.

The traffic model evaluation will not lead to any conclusions regarding the traffic models precision. However, by comparing the graph of one simulated UE and one real UE, similarities can still be seen in inactivity periods and bursts. The real data acquired can be questioned since the author produce it with the help of a mobile application. For a real test, this data should have been gotten elsewhere, preferably recorded from a base station. By adding more UEs in the simulation, the inactivity intervals disappear, which is reasonable, since that many UEs should keep the node
busy at all times. However, the large bursts seem to remain, which intuitively seems reasonable. Maybe an event will trigger many UEs to update at the same time, or one UE requires much data for a short period. The variability looks higher in the new model when comparing the graphs of many UEs with the old traffic model. As stated in 1.3.2, an identified trait of network traffic is high variability. Having a higher variability gives a hint that this model is on the right track in simulating UE data traffic. A more sophisticated test like the Turing test would have to be performed to be more confident and make conclusions. Unless the model is compared to real data from a node, we cannot know for sure how the output should look.

The performance tests show linear scaling of execution time when adding more objects to the simulation model. This result is what can probably be expected since each UE depends on nothing but itself and should lead to a linear increase in resource usage. The problem with these tests was that even though execution time was long, the simulator did not use up all the resources of the environment. The resource utilization means that there is a bottleneck somewhere preventing the simulator from executing even faster. As mentioned in 1.3.3, the DESMO-J framework class SimProcess uses the in-built threads of java. A simple test running multiple java threads to count Fibonacci numbers on the development environment could with no trouble utilize 100% of the CPU, meaning that the problem is not with Java threads, but something within DESMO-J that prevents full resource usage. To find the bottleneck, one would need more knowledge of Java than the author. The intended usage of the simulator was less than 10000 UEs, and within that range, the execution time is still reasonable, making the simulator useful for its intended use. The other entities cells, roads, nodes, and links would have to be tested as well to identify possible bottlenecks and do a more extensive performance test. However, this test shows that performance might be gained by finding and fixing the bottleneck.

5.5 The work in a wider context

In a world where climate change is a significant threat, every energy savings we can do is essential. When building a cellular network, savings can be done by not
overestimating the expected load. Both building and running a network requires much energy. On the other hand, building a network that cannot deal with the requested traffic load is something that companies try to avoid, which would most likely make the customers go to their competitors. Planning to build a RAN is where a RAN-simulator comes in, to find a balance between savings and performance. Building networks with the correct capacity is not only saving energy, but also money which benefits the operator.

From an ethical perspective, we can also say that a correctly working RAN is of high importance where machine-to-machine communication being reliable and fast is crucial. If we are ever to rely on AI as it gets more sophisticated in the future, it is of high importance that the AI acts on the correct stimuli, which can be obtained by the cellular networks. In some cases, the networks are not allowed to have faults since they can lead to enormous consequences. Simulation is a great step to ensure that the capacity is good enough for the intended purpose.
Conclusion

During this thesis, extensions to a RAN-simulator have been developed. The RAN-simulator is used to model and evaluate a RAN of an area and can be used to create the most suitable network. This thesis has been focusing on the process of development for research, not the research itself. Since no research have been conducted, no scientific conclusions can be made from this thesis. With that said, the research questions will be answered.

*How could UE behavior be simulated to be sufficient as traffic load input for a RAN-model for simulation?*

How a UE was implemented during this thesis is described thoroughly in 4.2.2. It consists of two processes running simultaneously, one determining movement and the other requesting or sending data to the network. The traffic model is an implementation of the ON-OFF source generation algorithm described in 1.3.2. The traffic model consists of two distributions which determine the rate of sending or requesting data, and the size of each burst of data. The movement of the UE is not based on any paper and could be improved by looking at crowd simulation, for example.

*How does adding UEs affect a RAN simulators ability to imitate a real RAN?*

The model validation in 4.4.2 does not hold up to the standards of scientific research; it only shows that the accuracy is probably improved with the addition of UEs. Even with a better method, 5G is not out yet, which means that it cannot be compared to real data still. More real data should be collected and given to an expert to improve the confidence of this answer. The results look promising, however, and the simulator can still be useful for its purpose as it is. As mentioned in 1.3.2, a simulation model should not be more complicated than required to solve the given problems.

Overall, this thesis has been an excellent opportunity for improving development skills and adapting to working in the industry. There have been many challenges with many possible solutions where the author had to make a decision, much like working in a real industry.
Future work
There is still much functionality not yet implemented that would make the simulator more accurate. Radio access signals are affected by many parameters which are not represented in the simulator, like geography elements or weather. For example, buildings would block much of the signals, and this is not something which is included. Instead, signals would bounce of buildings introducing very complex calculations. Weather can also reduce the signal significantly. Every simulation is the perfect day scenario currently. The simulations could be more precise with the addition of more parameters. Links should not be 100% reliable either, and adding scripting support for altering link-behavior is another feature to implement. This thesis introduced UEs as an entity, but not the mobility aspect of running a RAN like handovers. The movement model for UEs is simple, and adding support to alter it could be an extension. Other up an coming features of the RAN could be introduced in the simulator, like carrier aggregation, network slicing, and beaming.

To make the simulator scale better with more entity-processes than 2500, [20] suggests a method to utilize JavaFlow instead of Java threads in the DESMO-J class SimProcess as mentioned in 1.3.3. For more extensive networks than the ones simulated in this thesis, this should be taken into consideration unless the hardware is extremely powerful.
References


Acknowledgement

I would like to thank my supervisors at Ericsson, without their input and kind words every week, this would not have been possible. I would also like to thank my examiner and supervisor, Peter Jonsson and Christer Bäckström for being kind and easy to work with. Thanks to Yi Hu for a constructive opposition.