# Automatic Verification Of 3GPP Throughput Counters In PDCP/RLC/MAC Layer Capacity Testing

University of Oulu
Degree Programme in Information Processing Science
Master's Thesis
Risto-Matti Piirainen
4.10.2017

# Abstract

Counters provide information about the functionality of the base station. That information is highly valuable for mobile operators, who re-configure their networks partly based on that information. Mobile operators also monitor counters to see, what base station is capable of. Each base station has hundreds of different counters, measuring numerous different things continuously. Counter information is provided by the base station software, which takes care of keeping all the counters up-to-date. Nowadays base stations are very efficient, and they are capable to handle thousands of different requests in the blink of the eye. Increasing complexity of the product poses an enormous challenge for counters, since calculating the values for each counter gets more complex.

This research was conducted in Finnish large-scale telecommunications company. Although counters are extremely important for customers, they are not verified effectively in case company's LTE PDCP/RLC/MAC layer's capacity tests. The goal of this research is to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. Design science research was applied to achieve this goal.

In this research, literature review is conducted to gain understanding for LTE, 3GPP throughput counters, and about the capacity testing environment of the case company. Then a new counter verification system for LTE PDCP/RLC/MAC layer's capacity tests is designed and implemented. After the system is implemented, expected counter values need to be calculated for each test case and counter, which are part of this research. Evaluation of the system is made against the system requirements and to accuracy of limit value calculations.

As a conclusion, it can be said that the implementation of the system was a success, but in some of the test cases, counters provided unexpected results. The implemented system was able to catch the faults, but the root causes for problems are not clear. In total, 185 test case – counter combinations were verified, and in almost 13 % of them counter verification failed the test case, because counter provided unexpected value. In future, it would be beneficial to make a root cause analysis for the issues this research pointed out.

*Keywords*
3GPP, automatic verification, capacity tests, counter, LTE, throughput counter

*Supervisor*
Professor Burak Turhan

# Acknowledgements

I would like to use this opportunity to show my gratitude to people, who have made all this possible. I want to thank my supervisor Jorma Taramaa, for hiring me as part of this remarkable company, and from this par excellence opportunity to write my master's thesis here, and from all the support during research. I would also like to thank several people from the case company, who were willing to use their time and expertise, to help this thesis to get done. To mention one of the many, thank you Tarmo Auvinen, Jouko Lindfors and Jukka Setälä from all the help and support that I have received during this time. It turned out to be priceless.

I would like to show my gratitude towards my supervisor from University of Oulu, Professor Burak Turhan. Thank you for bringing your academic wisdom and experience as part of this research. It made this journey a lot easier.

Finally, I would like to thank my family and friends. Gratitude and appreciation towards you goes way beyond words.

Risto-Matti Piirainen

Oulu, October 4th, 2017

# Abbreviations

3G = 3rd Generation of mobile telecommunications technology

4G = 4th Generation of mobile telecommunications technology

5G = 5th Generation of mobile telecommunications technology

3GPP = 3rd Generation Partnership Project

API = Application Programming Interface

A-TTD = Acceptance Test-Driven Development

CA = Carrier Aggregation

CI = Continuous Integration

CPU = Central Processing Unit

CRC = Cyclic Redundancy Check

DL = Downlink

DoD = Definition of Done

DSR = Design Science Research

eNodeB = E-UTRAN NodeB

EPC = Evolved Packet Core

E-UTRAN = Evolved Universal Terrestrial Radio Access Network

FD = Frequency-domain

FDD = Frequency Division Duplexing

HARQ = Hybrid Adaptive Repeat and Request

HSPDA = High Speed Downlink Packet Access

HTML = Hypertext Mark-up Language

HW = Hardware

IEEE = Institute of Electrical and Electronics Engineers

ID = Identification Document

IDE = Integrated Development Environment

IS = Information Systems

IT = Information Technology

IP = Internet Protocol

ISO = International Organization of Standardization

KPI = Key Performance Indicator

LTE = Long Term Evolution

LTE-A = Long Term Evolution Advanced

MAC = Medium Access Control

MIMO = Multiple Input Multiple Output

Mhz = Megahertz

Mbps = Megabytes per second

PDCP = Packet Data Convergence Control

PHY = Physical layer

PDU = Protocol Data Unit

QAM = Quadrature Amplitude Modulation

QCI = QoS Class Identifier

QoS = Quality of Service

reST = restructuredText

RLC = Radio Link Control

RRC = Radio Resource Control

SDU = Service Data Unit

SCH = Shared Channel

SCT = System Component Testing

SUT = System Under Test

SW = Software

TB = Transport Block

TD = Time-domain

TDD = Test-Driven Development

TDD = Time Division Duplexing

TSV = Tab-Separated Values

TTI = Transmission Time Interval

UE = User Equipment

UL = Uplink

Uu = Air-interface between eNodeB and UE

VoLTE = Voice over LTE

VCR = Version Control Repository

# Contents


Abstract ........ 2
Acknowledgements ........ 3
Abbreviations ........ 4
Contents ........ 7
1. Introduction ........ 8
2. Research Problem and Method ........ 10
  2.1 Research Questions ........ 10
  2.2 Design Science Research ........ 11
    2.2.1 Design Science Research Guidelines ........ 11
  2.3 Design science research framework ........ 14
3. Prior Research ........ 17
  3.1 Application: LTE standard ........ 17
    3.1.1 Emergence of LTE ........ 17
    3.1.2 LTE overview ........ 17
    3.1.3 LTE Radio Protocols ........ 18
    3.1.4 LTE data throughput counters ........ 20
  3.2 Fundamentals of case company’s capacity testing environment ........ 22
    3.2.1 Test Automation in Software Testing ........ 22
    3.2.2 Continuous Integration ........ 23
    3.2.3 Capacity testing ........ 24
    3.2.4 Python ........ 26
    3.2.5 Robot framework ........ 26
4. Implementation ........ 29
  4.1 Selected counters for automatic verification ........ 29
  4.2 Selected test cases ........ 31
  4.3 System Component Testing ........ 32
  4.4 Reading counter values in SCT ........ 33
  4.5 Design of the new system ........ 35
  4.6 Implementation of the counter verifier ........ 38
  4.7 Limit value calculations ........ 45
5. Evaluation ........ 48
  5.1 Counter value calculation accuracy ........ 48
    5.1.1 Single UE, Multi UE & Traffic mix test cases ........ 48
    5.1.2 Packet dropping test cases ........ 52
    5.1.3 Summary of the results ........ 54
  5.2 Comparison against the user stories ........ 57
6. Discussion and implications ........ 61
  6.1 Evaluation against the design science research guidelines ........ 62
7. Conclusions ........ 64
  7.1 Limitations ........ 64
  7.2 Future work ........ 65
References ........ 66

# 1. Introduction

Mobile phone industry has revolutionized in past 15 years, and telecom industry has been under a massive change during that time. Industry, which used to mainly focus for providing voice communication services to end users, has become the servant for end users, who use mobile devices as their main device to access the internet. Incremental use of social media, web browsing and video streaming on mobile devices have catalysed the continuous development of mobile networks (Dahlman, Parkvall & Skold, 2013.) In 2017, global cellular data traffic will transfer over 100 exabytes (100 billion gigabytes) of data, and in 2021 cellular data traffic has been projected to surpass 350 exabyte limit (Analysys Mason, 2017). Although from all the mobile connections only 26 percent were 4G ($4^{th}$ Generation of mobile telecommunications technology) connections, 4G traffic accounted 69 percent of all mobile traffic in 2016 (Cisco, 2017). In enormously large, developing markets, like China and India, 4G adoption is still ongoing process and happening rapidly (Analysys Mason, 2017). LTE (Long Term Evolution) and LTE-A (LTE-Advanced), which are usually labelled as 4G mobile technologies, are facing huge challenges, while trying to answer to this widely spreading, more intensive data usage. (Dahlman et al, 2011).

3GPP ($3^{rd}$ Generation Partnership Project) unites telecommunications standard development organizations. 3GPP was originally formed in 1998 to provide technical specifications and reports for 3G ($3^{rd}$ Generation of mobile telecommunications technology). 3GPP provided also 4G (LTE & LTE-Advanced) specification and it will be followed by 5G ($5^{th}$ Generation of mobile telecommunications technology) in the near future. 3GPP has defined quality measurement tools, which are called *counters*. Counters provide information for mobile operators about the base station. Information about crucial things, like functionality related issues and network performance, provide valuable information for mobile operators, in order to optimize their network efficiency (3GPP, 2017.)

The objective of this master's thesis is to research how LTE PDCP/RLC/MAC layer's throughput counters can be automatically verified in simulated capacity tests. In this context *PDCP (Packet Data Convergence Protocol)/RLC (Radio Link Control) /MAC (Medium Access Control) layer* means the layer that is above the physical layer, and consists from these aforementioned protocols (Holma & Toskala, 2011). *Capacity tests* are run to validate, that system can work in predefined load and still meet the other requirements in functionality. In this research, capacity tests refer to case company's LTE PDCP/RLC/MAC layer's simulated capacity tests. Manual testing for large-scale systems, like LTE mobile base station, is expensive, hard-to-execute and inefficient, so system capacity is continuously tested in simulated test environment, with the help of test automation. Automated tests are run all day and night to gather information about the maturity of the product. *Throughput counters* are data throughput measurement indicators. Counter calculators have already been implemented to the product in question.

*Counter verification* is a process, where counter value is checked, and it is checked that checked counter value meets the expected value. Expected value is based on test case parameters. Counter verification has been part of the few test cases in PDCP/RLC/MAC layer's capacity tests, so it is not a new thing. PDCP/RLC/MAC layer's capacity tests

consist from 500+ test cases, so counter verification is not covered efficiently in capacity tests. Problem with the earlier counter verification in capacity tests is that there are no general ways to do it. Earlier solutions have been tailored for designated test cases. Thus, the goal of this research is to create a mechanism, which makes it possible to easily enable automatic counter verification *in any* automated capacity test case.

The motivation for this study comes from the importance of counters. Base stations should work in an acceptable level, even when the traffic levels are exceptionally high. Counters, which are one of the Key Performance Indicators (KPI) of the product in question, provide an essential data for operators, in order to make effective network planning (Gordejuela-Sanchez, Zhang, 2009). In embedded system like baseband station, system performance is achieved with efficient solutions in system design, rather than expanding the memory resources, since they are limited. System performance can be vital for the success of the product. Therefore, everything that happens within the base station, needs to be done as efficiently as possible.

In this aspect, counters are exceptional. Counters are not having any direct positive influence on the system performance or any other functionality of the base station. Vice versa, from the manufacturer's point-of-view counters are basically just "mandatory calculations", that are eating up memory and using resources unnecessarily. But for customers, counters are extremely valuable. In telecom business, customers are basically mobile operators, so the deals that are made, tend to be massive. Case company has customers, who are continuously following these numbers, and if they are not satisfied with them, questions will be asked. Counter values show operators, how their product is currently answering the network usage needs that their customers have. Based on that information, operators can re-configure or re-organize their networks. Other thing is that counters provide a method for customers to see, what they are actually paying for. It is valuable for the case company to know that their products have functional quality measurements.

This research begins by defining the research problem and with specification of research questions. This research applies the design science research methodology, which is also presented in next chapter. After that, literature review is conducted, to understand how the objective of this research is achieved. With the knowledge that literature review provides, and based on requirements that case company sets, automatic verification of LTE 3GPP PDCP/RLC/MAC layer's throughput counters is implemented as a part of existing simulated capacity tests. After implementation is done, it is important to collect results from the test cases, where the new counter verification system is taken into use. That helps the final part of this research, which is to evaluate the research results and to discuss about its limitations, and about future research.

# 2. Research Problem and Method

In this chapter, research problem and method will be introduced, which will start by defining the research questions. It will be followed by defining research method that is applied in this research, which is design science research. Defining the design science research consists from definition of design science research guidelines and their application to this study. Finally, design science research framework will be exercised to this research. The purpose of this chapter is to define the research artifact.

## 2.1 Research Questions

The objective of this research is to provide a solution, which enables the automatic verification 3GPP counters in LTE PDCP/RLC/MAC layer's simulated capacity tests. To achieve that, a counter verification system for capacity test needs to be implemented, and the limit value calculations for counters need to be accurate. To create research question to answer this problem, objective should be divided to smaller pieces. As mentioned in introduction, PDCP/RLC/MAC layer in this context means the layer that is above LTE's physical layer, and it consists from PDCP, RLC and MAC protocols. Capacity tests validate that system under test can work in predefined load and still meet the functional requirements. Throughput counters are measurements for data throughput. The implementation is done for large-scale telecommunications company, so it must meet case company's requirements and support the findings from existing literature. The existing simulated test environment sets also some principles and "ground rules" for the implementation. Evaluation of this research is made against the requirements that case company provides, and based on the accuracy of the counter limit value calculations. Considering the implementation objective and evaluation methods, following research questions should be answered to solve the research problem.

Research Question #1: *How a dynamic, easily configurable counter verification system can be implemented to capacity tests?*

Research Question #2: *What factors affect to the limit value calculation accuracy in simulated test environment?*

By answering to the first question, steps toward implementation of the counter verification system are recognized. This question is broad, and it consists of findings from the earlier literature, case company requirements and from the structure and modules of the existing simulated test environment. The second research question is equally as important than the first one. It is not enough alone to create a method, which enables automatic counter verification in capacity tests. Based on the test case details, it needs to be calculated, what kind of results are expected in each test case with different counters. This question consists of findings of the test case details and calculations. The goal of this research is to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. By answering both aforementioned research questions, that goal is achieved.

## 2.2 Design Science Research

Design science is inherently a problem-solving process. The fundamental principle of design-science research is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact (Hevner, et al. 2004.) That is why design science research is a suitable choice for this research; case company has a concrete problem to solve, and the implemented artifact solves that problem. Also, the design problem and its solution are acquired during this research.

### 2.2.1 Design Science Research Guidelines

Like stated earlier, the goal of this research is to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. Design-science research guidelines (Hevner et al, 2004) are followed in this study. These guidelines and how they are applied in this research, are explained in this chapter. Hevner et al. 2004 mentions that guidelines are not mandatory to follow in design-science research, but they most certainly help. Use of these guidelines is a decision that everyone must make depending on their research (Hevner et al, 2004). Guidelines for design science in information systems research are explained in following table 1 and in more detail below the table 1.

**Table 1.** Design-science research guidelines that are followed in this study (Hevner et al, 2004).

| **Design-Science Research Guidelines** | |
|---|---|
| **Guideline** | **Description** |
| Guideline 1: Design as an Artifact | Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an installation |
| Guideline 2: Problem Relevance | The objective of design-science research is to develop technology-based solutions to important and relevant business problems |
| Guideline 3: Design Evaluation | The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods. |
| Guideline 4: Research Contributions | Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies. |
| Guideline 5: Research Rigor | Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact. |
| Guideline 6: Design as a Search Process | The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment. |
| Guideline 7: Communication of Research | Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences. |

**Guideline 1: Design as an Artifact**

Guideline 1 describes creation of IT artifact. Artifact is created to address an important organizational problem. Artifact should be described effectively. IT artifact does not

only consist from instantiations of the IT artifact, but also from the constructs, models and methods applied in the development and from the use of information systems. Design science research in IT often addresses to design problems of the information system. Produced instantiations can be intellectual or software tools, which aim to improve the process of information system development. Artifacts are not independent of people or the organizational and social contexts, but interdependent and coequal with them in meeting business needs (Hevner et al, 2004.) Constructing a system instantiation that automates a process demonstrates that the process can, in fact, be automated. It provides “proof by construction” (Nunamaker, 1991).

In this paper, artifact is the system that makes automatic 3GPP counter verification possible in LTE PDCP/RLC/MAC layer’s simulated capacity testing, and it should able to be enabled in any of the 500 test cases in capacity tests. Existing literature, and examination of case company’s simulated test environment sharpens the steps how this objective can be achieved. Case company gives the requirements, which specifies the artifact. These requirements include the common functionalities that counter verification system should be able to perform, counters that are chosen to be automatically verified, test cases where automatic counter verification will be implemented, suitability to existing capacity test environment and visual presentation of those counter values. With the implementation of this artifact, it is proven that this process can be automated. Calculation of the expected counter values is also a part of the artifact.

**Guideline 2: Problem Relevance**

Information system research’s objective is to acquire the understanding and the knowledge that enable the implementation and development of technology-based solutions, to solve important and unsolved business problems. Design science aims to approach this goal, and to change the occurring phenomena by creating and constructing innovative artifacts (Hevner et al, 2004.)

In this research, the business problem is that even though the counters are vital from the perspective of customers, they are not still broadly verified in capacity tests. One reason for that is the shortage of system, which would make counter verification easy. Customers are measuring the product quality by the information received from counter values, e.g. by following throughput counters, they can measure product’s data throughput capability. This research will help to find defects earlier, and it can save costs for case company, since the bugs that found earlier are significantly cheaper, faster and easier to fix.

**Guideline 3: Design Evaluation**

Hevner et al (2004) states that the efficacy, equality and utility of a design artifact, needs to be demonstrated rigorously with the help of executed evaluation methods. Evaluation is a decisive component of the research process. Evaluation of the artifact is based on business environment’s requirements. Artifacts can be evaluated in terms of usability, completeness, accuracy, functionality, fit with the organization, reliability, performance and other relevant quality attributes. An effective and complete design artifact meets the requirements and constraints of the problem it tried to solve (Hevner et al, 2004.)

Evaluation of this artifact is strongly based to the case company’s requirements of the counter verification system and it will be evaluated based on how efficiently the given requirements were met. Also, it will be evaluated how accurate the calculations of

expected counter values are. Ideally expected counter values should be close to the actual counter values. Naturally, this kind of measurement will not depend only from the quality of the calculations, but also from quality of the product in question, and it is also possible that there are some issues in simulated test environment, that make counter verification complicated. If there are problems with the product in question, or with simulated test environment, they will be vital findings from the perspective of the case company.

**Guideline 4: Research Contributions**

For any research, the ultimate assessment is: "What are the new and interesting contributions?" In design science research, three different types of contributions are possible to achieve, based on the generality, significance and novelty of the designed artifact. These three types (Hevner et al, 2004) are:

1. *The design artifact.* Usually in design science research, the contribution is the artifact itself. Artifact should be able to solve problems that are not yet solved or apply existing knowledge in new and innovative ways (Hevner et al, 2004.) In this research, artifact should be able to do both things. Artifact should be able to solve the unsolved problem and benefit from existing knowledge in the new and exciting ways.

2. *Foundations.* In the design science, the creative development of novel, correctly evaluated constructs, methods, instantiations or models, that improve and extend the existing knowledge base, are significant contributions (Hevner et al, 2004.) If there are similar, or even existing methods for counter verification in existing literature, they will be extended and improved as part of this research.

3. *Methodologies.* The creative use and development of evaluation methods and new evaluation metrics provide together the contributions of design-science research. Evaluation metrics and measures in particular are design science research's the most crucial components (Hevner et al, 2004.) The evaluation methods, that are used in this research are presented as part of *Guideline 3: Design Evaluation.*

**Guideline 5: Research Rigor**

Rigor paves the way for the conduction of the research. Rigorous methods need to be applied in the both evaluation and construction of the design science research's artifact. In design science research, mathematical formalism is often used in the description of the specified and constructed artifact. Usually success of the researcher is determined by the use of appropriate techniques to construct or develop an artifact, and by the use of appropriate measures in the evaluation of artifact. Exercising the artifact in appropriate environment should not be forgotten either (Hevner et al, 2004.)

**Guideline 6: Design as a Search process**

Design science is inherently iterative. The search for optimal or the best design is usually difficult to fit to realistic information systems problems. Design is a search process, where it is essential to discover an effective solution for a problem. Representation and abstraction of appropriate ends, laws and means are critical components of design science research. Ends are the goals and constraints of solution. Laws are environment's uncontrollable forces. Means are the resources and actions, which are available for construction of solution (Hevner et al, 2004.)

Design phase starts with the literature review of the key parts of LTE PDCP/RLC/MAC layer's 3GPP counters, and of the tools that are needed for building the artifact. Tools include the case company's simulated test environment, and its key components. After that simulated test environment, and especially capacity tests should be taken to further examination, and based on those findings effective solution to problem is designed. To sharpen the solution, all of this is done iteratively, repeating each step until problem is solved.

In this research "ends" point to the goal of this research, which is to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. "Ends" also consider the possible limitations that current simulated test environment and product in question set for this research. In this research "means" refer to the earlier literature, interaction with colleagues, existing code base, case company requirements, simulated test environment, and technologies that are used. In this research, it is very hard to name any "laws", because it seems like there are no uncontrollable forces here.

**Guideline 7: Communication of Research**

Design-science research needs to be presented for technology- and management-oriented audiences. Technology-oriented audiences need more detailed information about artifacts, and presentation in organizational context. This enables practitioners to take the artifact's benefits in use. This also enables the future research and evaluation. Management-oriented audiences need information, to determine if organizational resources need to be used for the construction of the artifact. (Hevner et al, 2004.) This research aims to communicate with technology-oriented audience by presenting the findings in a detail. This research is also aimed for management oriented audiences.

## 2.3 Design science research framework

Design science research framework, which is applied in this research, is presented in this chapter. In Figure 1, Hevner et al's (2004) design-science research framework is applied to this research. Framework was designed to ease the understanding, executing and evaluating the information system research.

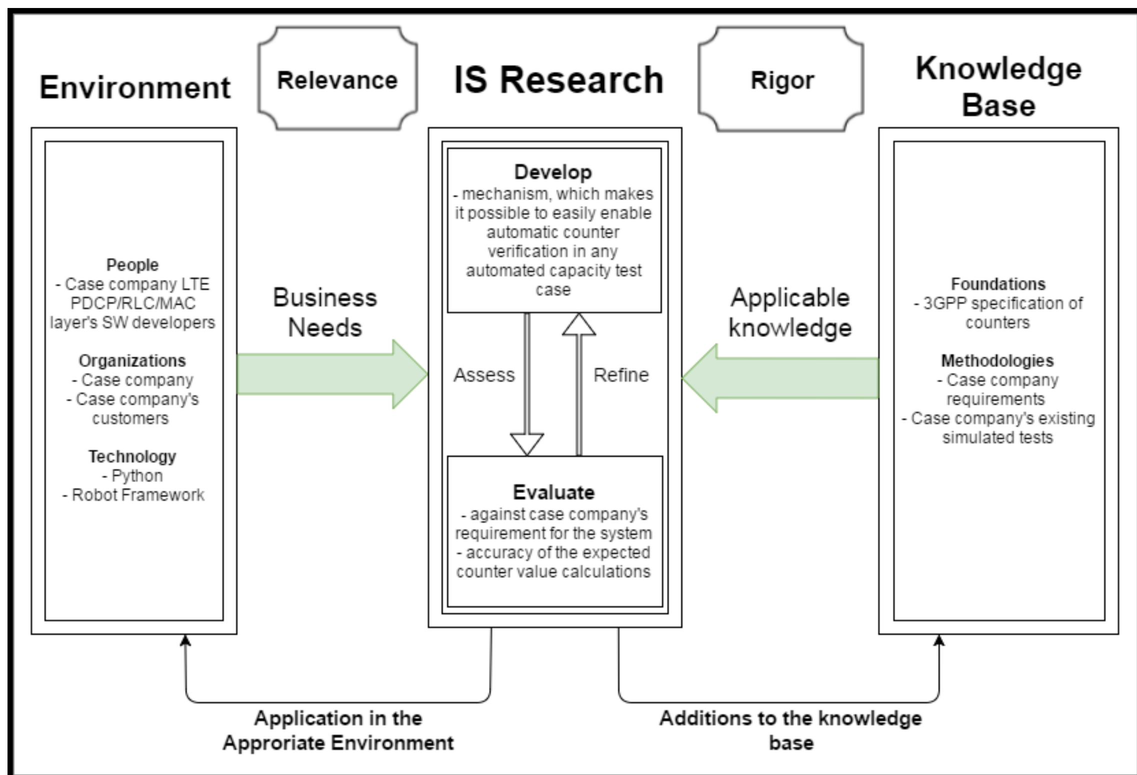



**Figure 1.** Design-science research framework applied to this research (Hevner et al, 2004).

**Environment:** Research is conducted in large-scale telecommunications company. Product in question is developed internationally in multiple sites. Product is an embedded system, mainly developed with C++ and has about 10 million lines of code in it. The product has been in the market for several years by the time this research is conducted, so the improvements which this research can possibly provide, are targeted towards latter deployments. The product consists from multiple system level components that are developed in multiple sites. PDCP/RCL/MAC layer forms one of the system components. There are more than 10 teams working in the PDCP/RLC/MAC layer system component. Teams have on average 8-10 people and they are cross-functional, which in this case means that teams have SW engineers, SW test engineers and SW specification engineers. One of the teams is concentrated in capacity testing, and this research is conducted in collaboration with the capacity testing team, and by collaborating with a couple of other teams.

SW (Software) developers, testers and managers in LTE PDCP/RLC/MAC layer are the stakeholders, that are the most interested about the functionality of the product in question’s counters. Obviously other parts of case company (marketing, executives, etc.) are interested about the product quality too, but LTE PDCP/RLC/MAC layer’s SW developers are the biggest stakeholders in this case.

Case company’s customers are constantly checking counter values to measure, how the base station they bought is functioning. If this implementation can reduce unwanted results on counter values, customers will get more precise data about the eNodeB. Term eNodeB refers to the base station in LTE network (Holma & Toskala, 2009).

Capacity testing is done to know the current capacity that system under test can handle (Humble & Farley, 2010). Capacity tests are run in automation server. For mobile network base stations, it is nearly impossible to run the capacity tests manually. That is why simulated capacity tests are so crucial tests for system. Some of the capacity tests

are run as part of the continuous integration (CI) process. Continuous integration means that developer commits to the shared mainline several times a day, and that includes compiling the code, and running needed tests successfully. In case company's product, those tests include some capacity tests, too. Capacity tests are written with Python and Robot Framework. Python is powerful, fast, easy-to-learn and widely used programming language (Python, 2017). Robot Framework is keyword-based test automation framework (Robot Framework, 2017.)

**Information Systems (IS) Research:** Goal of this reserach is to develop a system, which makes it possible to easily enable automatic counter verification in any automated capacity test case. This artifact is integrated as part of capacity tests which are run with the help of test automation and automation server. Method should be easily replicable to other types of tests than capacity tests. Also, implementation needs to be easily configurable, so it is easy to enable or disable counter verification from any test case in capacity tests. Counters to be automatically verified as part of this research, will be selected from 3GPP specification. Currently, without the few exceptions, counters are not verified in capacity tests.

Evaluation of this artifact is based on case company's requirements, by comparing how the requirements are met. One important part of the counter verification system is to have accurate calculations for expected counter values. Thus, another thing to evaluate is which factors affect to the accuracy of these calculations. In other words, when this artifact (automatic counter verification in capacity tests) is implemented, evaluation of the artifact is partly based on the results of the capacity tests, where implemented counter verification system is part of the test case. Another part is the system's ability to meet the given requirements.

**Knowledge Base:** 3GPP has defined counters, to create universal measures for base station quality. 3GPP provides a lot of useful information about the counters, which help in the design of the system. Case company's simulated test environment can also bring some ideas for the implementation; thus it should be investigated carefully. Simulated test environment is written mainly with Python and Robot Framework, and it consists from about million lines of code. Case company has defined requirements for this soon to be implemented counter verification system. They bring a skeleton for the design of the system. Counters need to be easily configurable for all the test cases in capacity tests. A group of test cases will be selected as part of this research. Counter verification system will be tested and verified as part of those test cases, so test cases should be as diverse as possible, to make sure that system works with every test case. Product in question is deployed in various of different hardware (HW) variants. Test cases that are selected to this research, should cover all the most important HW variants, so it can be verified that the new system is not dependent on HW. Information that this research provides, will help telecom community to gain knowledge, how to build such a system.

# 3. Prior Research

This chapter presents the findings from the earlier literature. It includes earlier research about the product in question (LTE), and about the most essential components that together form the case company's capacity testing environment.

## 3.1 Application: LTE standard

In this chapter, LTE and LTE data throughput counters are explained in basic level. This is started by shortly reviewing the history of LTE. After that, brief overview of LTE as a wireless communication standard is given, followed by LTE radio protocols (PDCP, RLC and MAC), which are introduced. Finally, LTE data throughput counters are further investigated.

### 3.1.1 Emergence of LTE

The development of Long Term Evolution (LTE) started in 2004 by 3rd Generation Partnership Project (3GPP). Even though HSDPA (High Speed Downlink Packet Access) was not deployed back then, the need for next radio system was evident and it was recognized that the work should be started. System standardization was started early enough, since it can take more than 5 years before system targets are set for commercial deployments by using interoperable standards. Wireline capacity evolution, rivalry of other wireless technologies, and the need for additional wireless capacity and for lower cost wireless data delivery were forces that pushed forward the LTE development. (Holma & Toskala, 2009.)

### 3.1.2 LTE overview

LTE's high capacity is based on frequency dimension in the packet scheduling (Holma & Toskala, 2009). In Frequency Division Duplexing (FDD) there are two separate carrier frequencies. One is reserved for uplink transmission and other for downlink transmission. Time Division Duplexing (TDD) uses a single carrier frequency which is used for both uplink and downlink transmissions. In FDD uplink and downlink transmissions can occur simultaneously, and in TDD that is not possible (Dahlman, Sköld & Parkvall, 2011.) User specific allocation in uplink is made continuously, in order to enable single-carrier transmission. Downlink is able to use resource blocks freely around the different parts of spectrum. LTE brings flexibility to spectrum.

Transmission bandwidth can be from 1.4 MHz to 20 MHz, depending on spectrum availability. The 20 MHz is able to provide 150 Mbps downlink user data rate with 2 x 2 MIMO and 300 Mbps with 4 x 4 MIMO (Multiple Input Multiple Output). Uplink peak data rate is limited to 75 Mbps. (Holma & Toskala, 2009.) With 256 QAM (Quadrature Amplitude Modulation) it is possible achieve almost 400 Mbps transmission capacity (Nakazawa, Okamoto, Omiya, Kasai & Yoshida, 2010.)

UE (User Equipment) has two different states; RRC_CONNECTED (Radio Resource Control) and RRC_IDLE. In RRC_CONNECTED state, UE is transferring or receiving data with network. In RRC_IDLE state, UE is monitoring paging channel. That is done to detect incoming calls, get system information and make neighbouring cell

measurements and cell (re)selection. Cell is the geographical area that cellular radio antennas are able to cover. LTE enables higher RRC_CONNECTED quantities per one cell. That makes possible to have a large amounts of small bitrate connections and in same time, to have momentary connections with high UE bitrates. Figure 2 below illustrates how UEs are handled during scheduling. This process consists from three steps. Scheduling algorithm begins with pre-scheduling, which evaluates, if UE can be selected to the sub-frame. In second phase, scheduler selects UEs to the Time-domain (TD) scheduling. Lastly, scheduler performs Frequency-domain (FD) scheduling. This is done based on channel quality feedback (Holma & Toskala, 2011.)

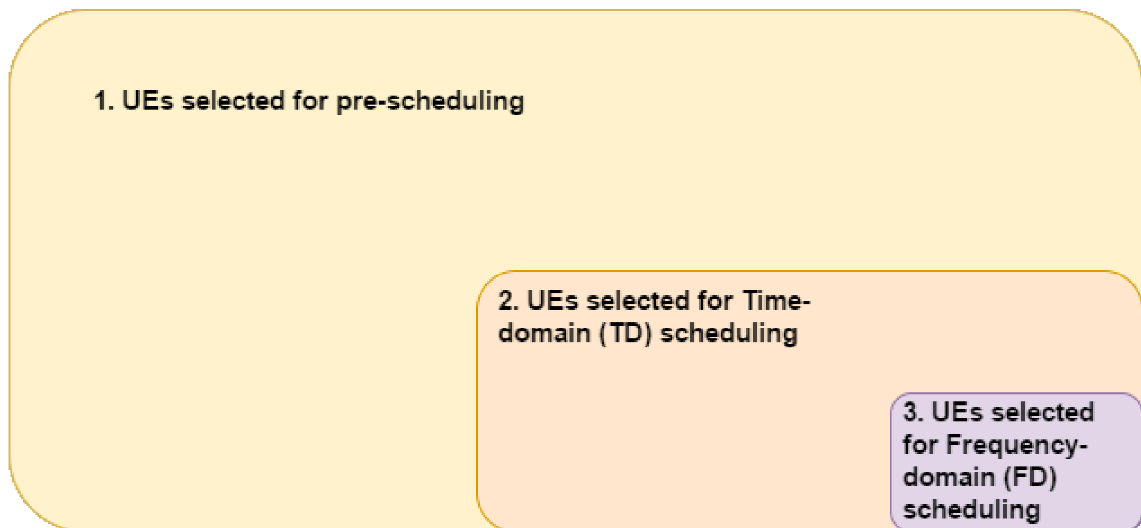



**Figure 2.** Time domain and frequency domain scheduling (Holma & Toskala, 2011).

Carrier aggregation (CA) enables faster data rates. Key idea of CA is to aggregate multiple carriers simultaneously which extends the maximum bandwidth in the uplink or/and downlink directions (Holma & Toskala, 2009). Downlink transmission can aggregate with five carries, and uplink with two carriers (3GPP, 2016.)

LTE uses physical layer's retransmission combining, which is also known as HARQ (Hybrid Adaptive Repeat and Request). In HARQ operation in physical layer, it is receiver's job to store the packets with failed CRC (Cyclic Redundancy Check) checks and to combine the received packet after retransmission is received (Holma & Toskala, 2009.)

### 3.1.3 LTE Radio Protocols

LTE radio interface protocols have one key role; setting up, reconfiguring, and releasing the Radio Bearer. Radio Bearer makes it possible to transfer the EPC (Evolved Packet Core) bearer. The LTE radio interface protocols in Layer 2 consists from Packet Data Convergence Protocol (PDCP), Radio Link Control (RLC) and Medium Access Control (MAC) (Holma & Toskala, 2009.) Above mentioned protocols are described in the following sections.

**Packet Data Convergence Protocol (PDCP)**

PDCP's most important role is to perform IP header compression. Header compression reduces the number of transmitted bits over the radio interface (Dahlman, Sköld & Parkvall, 2011). PDCP also takes care of the ciphering and deciphering of the user plane data, and also most of the control plane data. PDCP also covers integrity protection and

verification. That is made to make sure that control information comes from the correct source. In the uplink direction, all the packets that are not indicated as completed, are retransmitted in PDCP (Holma & Toskala, 2009.)

**Radio Link Control (RLC)**

RLC transfers the PDUs (Protocol Data Unit) that are coming from higher layers. RLC also covers segmentation/concatenation, duplicate detection, retransmission handling and in-sequence delivery to higher layers. Also, protocol error handling e.g. signalling errors, are handled in RLC (Holma & Toskala, 2009.)

**Medium Access Control (MAC)**

Main responsibilities of the MAC are uplink and downlink scheduling, multiplexing of logical channels and HARQ retransmissions. It also covers measurement reporting of traffic volume. That is done to provide information for the RRC (Radio Resource Control) layer about traffic volume experience. Transport format selection and priority management between the logical channels are also MAC's functionalities (Dahlman et al., 2011.)

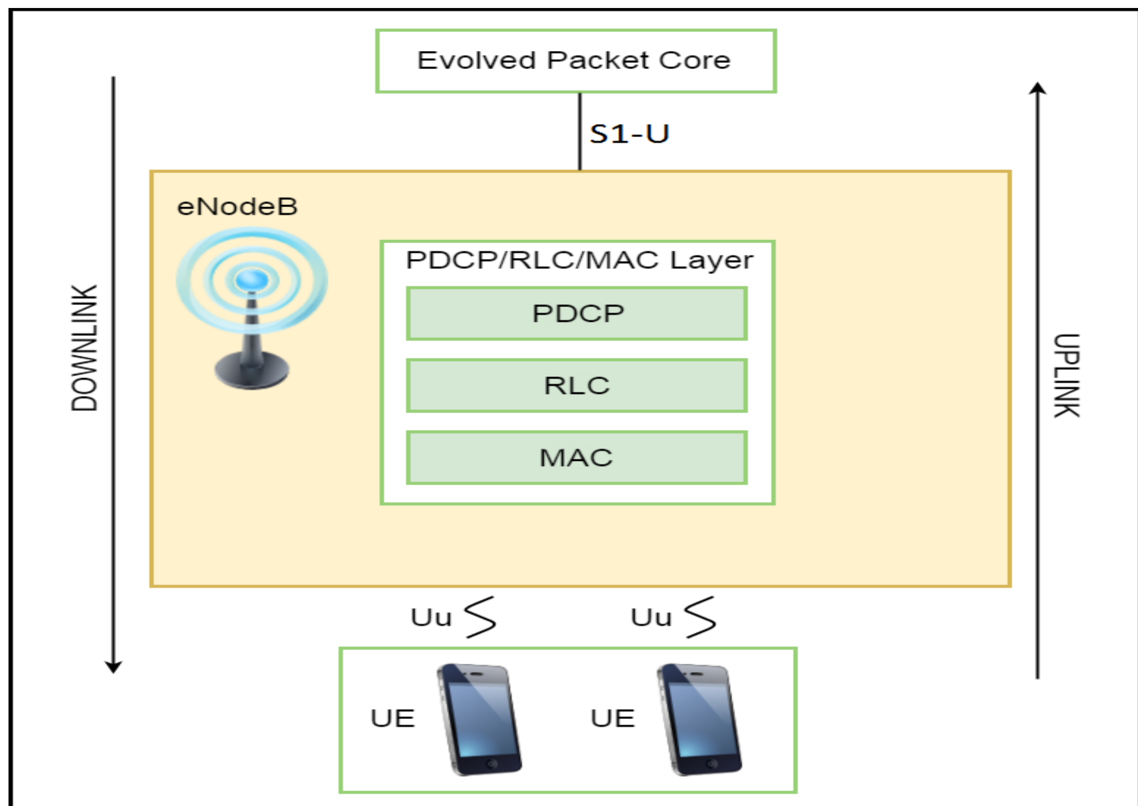



**Figure 3.** Downlink and uplink data flow from PDCP/RLC/MAC Layer's perspective (Holma & Toskala, 2011).

Figure 3 illustrates how downlink and uplink data goes through different PDCP/RLC/MAC layer protocols. In downlink side, data is going from EPC to eNodeB, which acts as a bridge between EPC and UE. Uu is air interface between eNodeB and UE, and S1-U is interface between EPC and eNodeB. EPC is the core network of LTE and it communicates with packet data networks in the outside world (e.g. internet). In figure 3, eNodeB is highly simplified, and all but PDCP/RLC/MAC layer are left out from it. UE is the device that is used for communication by end user. For uplink side,

everything works reversely from UE to eNodeB and from eNodeB to EPC (Holma & Toskala, 2011.)

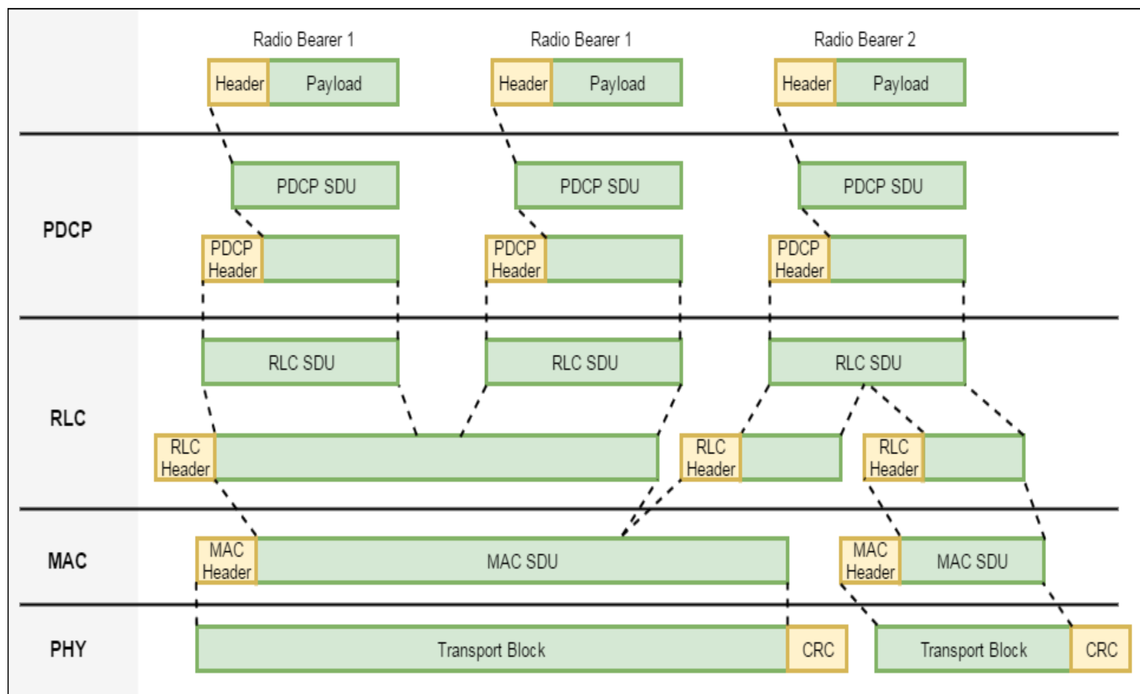



**Figure 4.** LTE data flow (Dahlman et al., 2011).

Figure 4 summarizes how downlink data goes through all the protocol layers. There are three IP packets in example, two in Radio Bearer 1 and one in another Radio Bearer 2. Packets that are received by each protocol layer are called Service Data Unit (SDU) and packets that going out from protocol layer are called Protocol Data Unit (PDU). PDCP layer adds PDCP header to packets that come to PDCP layer from Radio Bearers. RLC layer performs concatenation and/or segmentation for PDCP PDUs, to make RLC PDUs, and adds header to it. MAC layer multiplexes the RLC PDUs and attaches MAC header to outline a transport block. Physical layer (PHY) then attaches CRC to the transport block, via multiple transmit antennas, if possible. Data flow in uplink transmission is similar than in downlink transmission (Dahlman et al., 2011.)

### 3.1.4 LTE data throughput counters

IEEE Standard Glossary of Software Engineering Terminology (1990) defines throughput as “The amount of work that can be performed by a computer system or component in a given period of time.” In terms of data transmission, throughput means the number of successful messages that has been transferred over a communication channel in a given time, and it is usually measured with bits per second (Bianchi, 1998). 3GPP is an organization that unites telecommunication standard organizations. 3GPP has defined quality measurement tools, called counters, which provide information about the functionality of the product (3GPP, 2017.)

In operator investment, network planning and deployment are the most significant phases of modern mobile communications (Hoikkanen, 2007.) Therefore, accuracy in network performance estimation is pivotal (Mishra, 2007). Accurate network performance estimations enables the operator to advance network design and to optimise radio parameter configuration. Accurate estimations are difficult to obtain, since performance estimations are factors that are not always controllable (Fernández-

Segovia, Luna-Ramírez, Toril & Ubeda, 2011). Additionally, Buenestrado, Ruiz-Avils, Toril, Luna-Ramrez & Mendo (2014) state that for cellular networks, it is crucial to assess newly added network features in order to maintain high operability. To assess network performance, operators collect user data regularly (daily or weekly) from counters that are available on the radio network (Pierucci, 2015). Performance estimation can be done with the information that throughput and signal quality statistics provide. That data is based on network performance counters and call traces which are in LTE system's cells.

Counters collect the radio network information, like paging events, physical transmission powers, data throughput and handoff events (Cao, Li, Bu & Sanders, 2012). Investigable counters must be chosen wisely, so that they reflect to real-life use scenarios (Buenestrado et al, 2014). In recent years, mobile operators have started to follow user-centric instead of network-centric service performance indicators (Brooks & Hestnes, 2010). Mobile operators are currently following average user throughput in all radio access technologies, especially after user-centric indicators have become more common (3GPP, 2017).

Mobile network operators measure user experience against KPIs (Key Performance Indicators). Good user experience has become one of the most important factors to attract and retain new customers (Nguyen & Northcote, 2016.) At the moment, 3GPP has defined five different KPIs: Accessibility, Retainability, Integrity, Availability, and Mobility. Throughput counters are categorized under Integrity. Throughput measurements are performed by E-UTRAN (Evolved Universal Terrestrial Radio Access Network) or UE. There are four different types of counters, and every counter is measuring either minimum, maximum, average or sum (3GPP, 2017.)

For example, E-UTRAN IP Throughput is a KPI that illustrates the impact that E-UTRAN has on the end-user's service quality. IP throughput is measured in PDCP/RLC/MAC layer. It is calculated from IP level's payload data volume per elapsed time unit in Uu (Air-interface between eNodeB and UE) interface and it is independent from packet size and traffic patterns. IP Throughput is aimed for data bursts that are large enough to be split across multiple TTIs (Transmission Time Interval). When measuring IP Throughput, there has to be transmittable data in the buffer. IP Throughput in DL (downlink) is calculated with the following formula (3GPP, 2017.)

$$IP\ Throughput\ DL = \frac{\sum_{Samples} \mathrm{ThpVolDl}}{\sum_{Samples} \mathrm{ThpTimeDl}} \tag{1}$$

Formula 1 illustrates what kind of calculations 3GPP throughput counters usually are handling. Formula 1 shows, how IP Throughput in DL, which is used here as an example, is calculated. ThpVolDl means the (payload data) volume on IP level. ThpTimeDl time is the elapsed transmission time of the ThpVolDl. For both ThpVolDl and ThpTimeDl, the last piece of data transmitted in TTI (the buffer is emptied), is excluded (3GPP, 2017.)

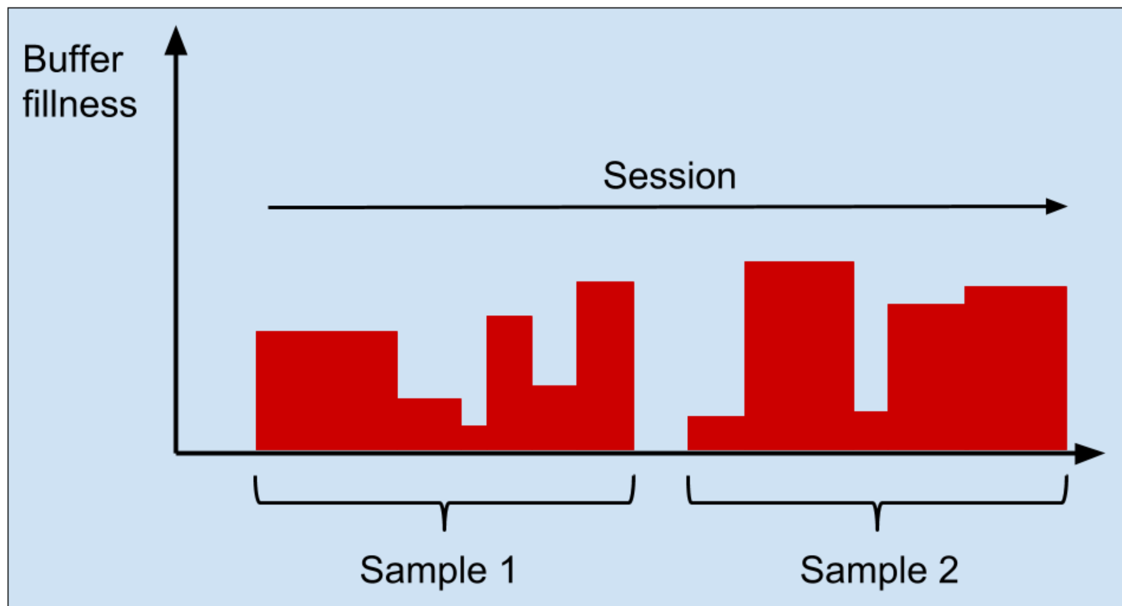



**Figure 5.** Sampling throughput data (3GPP, 2017).

In LTE traffic goes in bursts, so it is important that idle gaps are not considered in throughput calculations. Idle gaps are omitted by dividing each burst of data as a one sample. Idle gaps and sampling are described in figure 5. Samples 1 & 2 have transmitted the data that has come from buffer. When buffer is empty (between the samples), that time is not seen part of IP throughput measurement. To get throughput measurement results that are independent from the bursty traffic pattern, each burst of the data is considered as one data sample (3GPP, 2017.)

## 3.2 Fundamentals of case company's capacity testing environment

In this chapter, all essential parts of case company's automatic capacity testing environment are investigated based on the knowledge from the existing literacy. In the beginning of this chapter, software testing is described briefly, followed with description of test automation. After that, continuous integration as a software implementation practice is explained. In chapter after that, capacity testing is described, and it is followed by Robot Framework and Python which are presented in the end of this chapter.

All these areas are covered, because they are strongly related in this research. In case company, capacity tests are run as automated tests, as part of simulated test environment. Continuous Integration is practiced in the development of the product in question. Case company's CI process consists from various different kinds of tests, that make sure that maturity of the product remains high after each commit. Capacity testing is covered, because this research concentrates on the development of the system, that is piloted in case company's capacity tests. Simulated test environment is built with Robot Framework and Python.

### 3.2.1 Test Automation in Software Testing

Software testing has a significant role in software projects. It is estimated that almost half of the resources of software project are used to software testing. When software is tested, test engineers' aim is to add value to the product. Testing can raise value by improving product quality or reliability. Usually it is done by finding and removing the

errors (Myers, Sandler & Badgett, 2011.) Myers et al (2011) describes software testing as “the process of executing a program with the intent of finding errors.”

White-box testing is practise, where software’s internal structure is tested. Ideally all the parts of software, inputs and outputs are tested. That is possible with small systems that are very simple, but complicated to do with complex systems (Ostrand, 2002.) In white-box testing, test cases are created based on the structure of the test item. Black-box testing (also known as specification-based testing) is based to specification, rather than software’s source code. In black-box testing, external inputs and outputs of the system are the basis of test (ISO Standard, 2013.)

Test automation has a lot of advantages over manual software testing. It can save a lot of time, effort and money. Tests that take hours to run manually, can be executed in a couple of minutes. Besides those benefits, test automation can improve the software quality radically, since test automation can multiply the amount of run test cases. Automated tests increase the efficiency, since tests can run continuously, and that enables software test engineer to test something else. Tests that run automatically, are identical with sequence, timing and inputs time after time. That is impossible to guarantee with manual testing. Especially tests that are run frequently, should be automated. It is more expensive to automate a test case than run it once manually, but when test is run repeatedly, automated test is more economical (Fewster & Graham, 1999.)

In today’s software industry, agile is a very common software development approach. Agile development is based on doing the development in short iterations by frequently adapting and inspecting the product (ISO standard, 2011). Software is also continuously delivered and collaboration with customer and within the team(s) are in crucial role (Agile Manifesto, 2017). Test automation is key player in successful agile-oriented teams. Continuous Integration, which is covered in next chapter, has strong relations to test automation. Regression tests can be run daily, or even more often, so the need for strong automation is evident. Agile development cannot succeed without test automation. Some agile techniques like TDD (test-driven development) ensure that there will be unit tests that can be run automatically. Although unit tests are excellent way to find defects early and quickly, system-level testing is necessary (Graham & Fewster, 2012.)

### 3.2.2 Continuous Integration

In software development, Continuous Integration (CI) is the most popular development process, where software developers’ work is integrated to baseline frequently. After developer makes a commit, it is automatically followed by an automated build and tests. This is done to verify the possible integration errors as soon as possible. In today’s multi-site software development, CI is facing new challenges. CI process can be eased with automating the testing and building processes (Seth & Kare, 2015.)

Duvall (2015) illustrates in figure 6, how CI system works. At first, developer makes a commit to the Version Control Repository (VCR). CI server notices this, because it is continuously polling the repository for new changes. After CI server detects changes in VCR, it retrieves the copy of latest source code from repository. Then build script is executed, which integrates the commit as a part of the software. Build can include compilation, inspections, testing, and deployment. Based on the build results, CI server generates feedback, which is monitored to developers. Feedback should be given as quickly as possible.

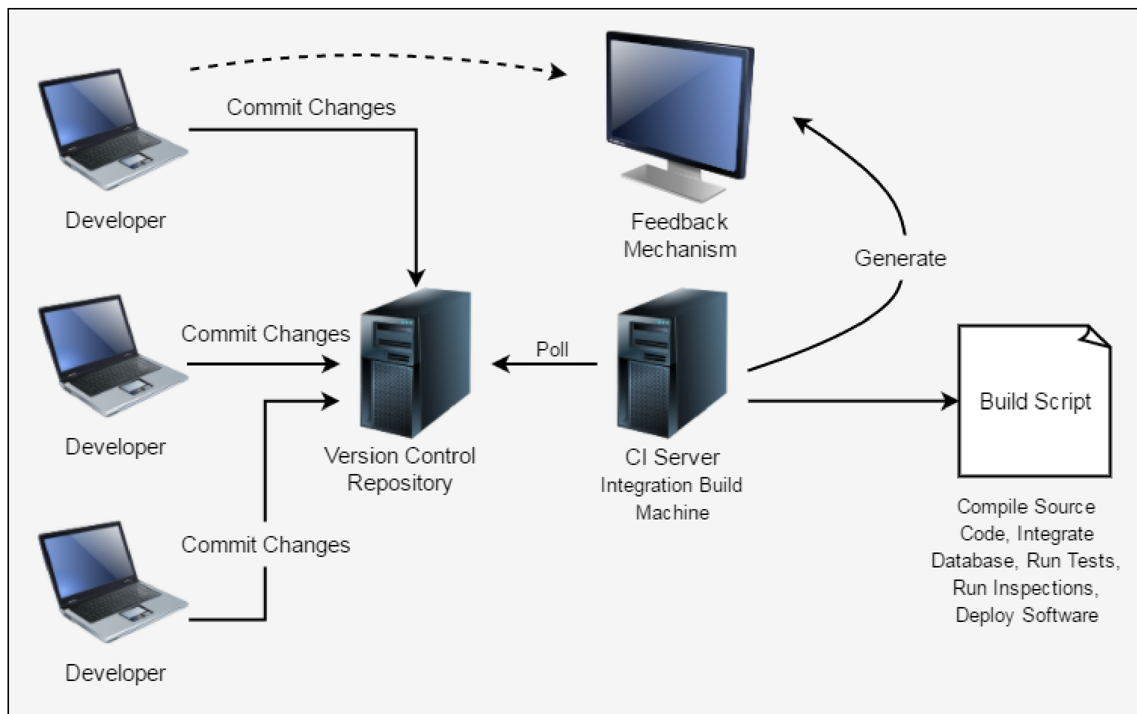



**Figure 6.** Components of CI system (Duvall, 2015).

Main advantages of the implementation of CI are that it can find errors and defects in early phase and reduce software development risks (Lai & Leu, 2015.) CI has various benefits. Besides it will reduce risks and helps to find errors in early phase, it will also reduce repetitive work and processes, generate automatically deployable software, increase the product confidence and enable better visibility for the project (Duvall, 2007.) Many teams that have used CI in their development have noticed that it has reduced the integration problems significantly and allowed team to develop cohesive software much faster (Martin Fowler, 2017.)

When running automated tests as part of CI process, 100 per cent of the automated tests should be passed in order to mark build as passed. This is based on technical criteria, not to assumption that all the project members make perfect work. It is easy to assume that code that does not compile, is not working. The same goes with automated tests. If code raises errors in tests, it is not working code. If code that does not pass all the tests is accepted, it will have negative impact on software quality (Duvall, 2007.)

### 3.2.3 Capacity testing

In this chapter, capacity testing is examined. This chapter explains terms capacity testing, performance testing, throughput testing, capacity, performance, throughput, and stub. Also, the most common capacity test types are explained here. Basic principles of capacity tests are explained.

Before going deeper into capacity testing, terminology should be opened up a little bit. **Performance** is measured with the time that it takes to process a single transaction. **Throughput** is measured with the number of transactions that the system can process in a predefined timespan. Bottleneck of the system is usually limiting the throughput. **Capacity** is the maximum throughput that system can have with a predefined workload, and still stay within the acceptable response times in every individual request (Nygard, 2007.)

ISO Standard (2013) defines capacity testing as “type of performance efficiency testing conducted to evaluate the level at which increasing load (of users, transactions, data storage, etc.) compromises a test item’s ability to sustain required performance.” Performance testing is “type of testing conducted to evaluate the degree to which a test item accomplishes its designated functions within given constraints of time and other resources” (ISO Standard, 2013.)

Capacity tests involves a measurement of a various different characteristics of an application (Humble & Farley, 2010.) Following test types presented below are very common ones.

**Scalability testing.** Humble & Farley (2010) define scalability testing with a question: “How do the response time of an individual request and the number of possible simultaneous users change as we add more servers, services, or threads?”

**Longevity testing (also known as stability testing).** In longevity testing, system is kept running for a long time to see if the behaviour of system changes in a long run. Usually memory leaks and stability problems are caught in this type of testing (Humble & Farley, 2010.)

**Throughput testing.** Number of transactions, page hits or messages that system can handle per second (Humble & Farley, 2010). In LTE, the maximum peak throughput in uplink and downlink transmissions is limited by the size of SCH (Shared Channel) transport block, which is accommodated in every TTI by UE device (Holma & Toskala, 2011).

**Load testing.** When load of the application is increased to the production like measures or beyond them, what happens to the system’s capacity? This is very common type of capacity testing (Humble & Farley, 2010.)

Capacity tests in high-performance systems can be really complicated to write. It is very hard to write a code for the system that is fast enough to pass the tests. That is why it is necessary to set a rate that enables the test to assert pass. When writing a capacity tests, it is very important to implement a no operational stub of the application, technology, or interface (Humble & Farley, 2010.) Stub is a “dummy component used to simulate the behaviour of a real component” (Beizer, 1990).

Capacity tests should be run automatically, every time there are changes implemented to the system. Capacity tests can be fragile, and they can be broken easily when there isnew changes in the system (Humble & Farley, 2010.)

There are few principles that capacity tests should follow (Humble & Farley, 2010):

- Tests should be mocking a real-word scenarios. In that way, important bugs are caught in much more efficient way.
- Threshold of success should be predefined in accurate way. That eases the defining when cases are passed and when not.
- Capacity tests should not take a lot of time. Test duration should be short.
- Tests should be built in a generalized way, so they don’t need a lot of rework when system under testing is changing.
- Tests should be mutable for large-scale scenarios, so real-world situations can be simulated

### 3.2.4 Python

Python is easy to learn, yet powerful object-oriented programming language. From its functionality, Python can be compared to Perl, Ruby, Scheme or Java. Python's syntax is elegant, which makes it easy to read (Python, 2017.) Python is a dynamically typed language. It means that type does not need to be statically specified (e.g. int i). That does not mean that dynamically typed languages are weakly typed. For example, Python will recognize if incorrect type is tried set to a variable, and raise an error. Python is suitable for both small and large systems, and thereby it is still used in various applications (Tratt & Wuyts, 2007.)

Development of Python started in late 80s in Netherlands, by researcher Guido van Rossum. Python's high readability comes from its English keywords and use of code blocks for group identification. Code blocks forces to correct indentation, since it is necessary to run Python program without errors. Python is interpreted, so there is no need for compilation. It is also a high-level programming language, which means it has an automatic memory management. (McGrath, 2014.) Python interpreter can be extended with C or C++ made functions and data types (Van Rossum, Drake & Kuchling, 1999).

### 3.2.5 Robot framework

Robot Framework describes itself as "generic test automation framework for acceptance testing and acceptance test-driven development (A-TTD)." Robot framework is generic framework, and it is technology and application independent (Robot Framework, 2017). In A-TTD, requirements are turned to examples and automatable tests. This creates executable specifications. They are created in requirement workshops with team, product owner and other stakeholders. Pekka Klärck created Robot Framework in 2005 at Nokia Networks. One of Robot Framework's original goals was to support A-TDD. Robot Framework was open-sourced in 2008 (Larman & Vodde, 2010.)

Robot Framework's test case design is made with keyword abstraction. There are two levels of keywords: low-level library keywords and high-level user keywords. Low-level keywords are defined in libraries and high-level user keywords in resource files and test suites. Standard libraries come with Robot Framework, and external libraries needs to be installed separately (Pajunen, Takala & Katara, 2011.)

Figure 7 describes high-level architecture of Robot Framework. Test data is presented in simple, tabular format. Robot framework processes the test data, runs test cases and creates test logs and reports. Test libraries handles the interaction with SUT (system under test). Core framework does not know anything about the SUT. Test libraries can interact directly with the lower-level of SUT, e.g. drivers or use application interface (Robot Framework, 2017.)

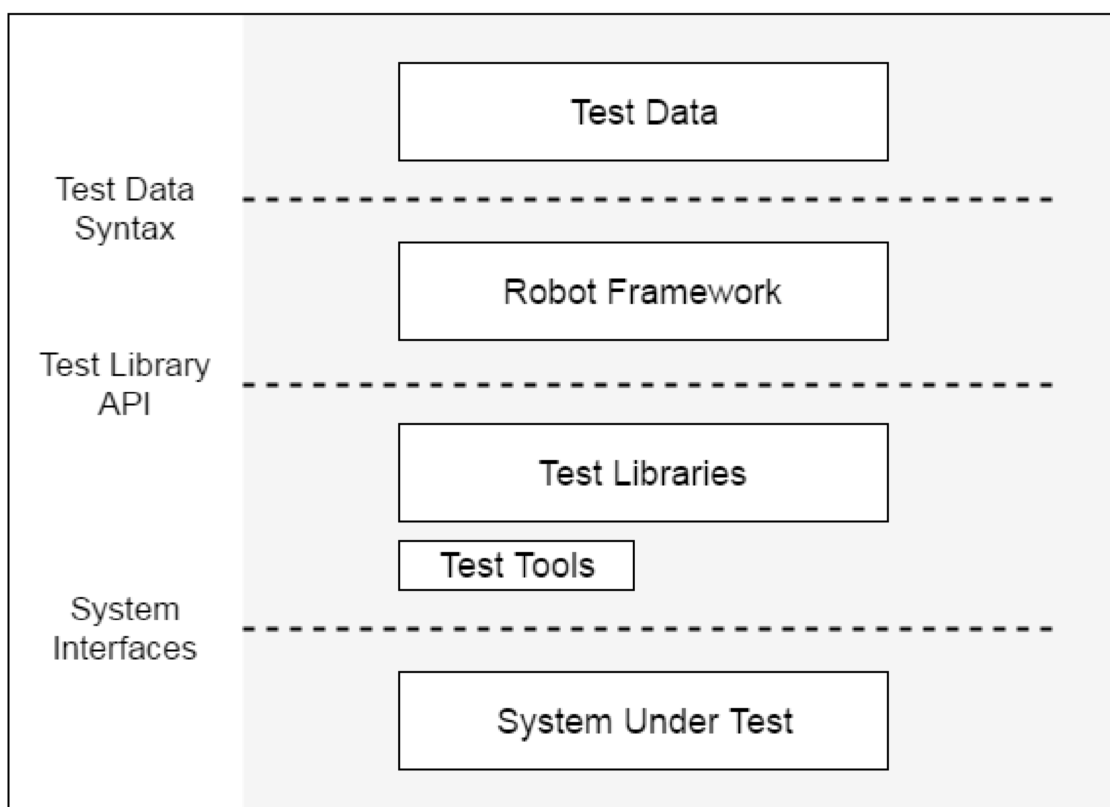



**Figure 7.** Robot Framework’s high-level architecture (Robot Framework, 2017).

Robot Framework is licensed with Apache 2.0 licence, which means that it can be used for free. Robot Framework is written in Python and it can be extended with own libraries that are written in Python or Java. It is also possible to implement libraries with C.

Robot Framework supports various formats for test data. Supported formats are hypertext markup language (HTML), tab-separated values (TSV), restructuredText (reST) or plain text. Plain text format is the most used format among users. Popularity can be explained with its compatibility with version control systems and easiness to use in any given text editor. Some IDEs (Integrated Development Environment) have plugins for Robot Framework (Robot Framework, 2017.)

The Python file below is ExampleLibrary.py. It adds arg1 and arg2 together and returns the sum of those number. In Robot Framework, this file is called custom library, which can be imported to test cases.

```
def calculate_sum_of_args(arg1, arg2):
    return int(arg1) + int(arg2)
```

Example below illustrates the importation of ExampleLibrary.py to Robot Framework test case and Robot Framework’s easy-to-read syntax, the simplicity of using it and expandability with self-made libraries.

```
*** Settings ***
Library  ExampleLibrary.py

*** Test Cases ***
Example sum two numbers
  ${result}=  Calculate Sum Of Args  1  2
  Should be equal as integers  ${result}  3
```

This Robot Framework example above shows how custom library is imported and used. ExampleLibrary.py is introduced under settings. Test will go to pass if custom keyword returns 3, when numbers 1 and 2 are given as arguments.

# 4. Implementation

This chapter is about the implementation details. It includes the presentation of counters and test cases that were selected as part of this research, introduction of the simulated test environment called SCT (System Component Testing), explanation how counter values are read in SCT, design and implementation details of the new counter verification system and finally, the process of limit value calculations.

## 4.1 Selected counters for automatic verification

Counters to be verified were selected in the meeting with a couple of specialists from the case company. Meeting attendees consisted from the capacity testing and counter experts, and they included SW engineers and SW specification engineers. Choice was neither easy nor obvious, since in PDCP/RLC/MAC layer, there are hundreds of different counters to choose from. The idea of this work is to create a mechanism for the case company, which enables dynamical counter verification in capacity tests, and ideally also in other types of tests too.

It was decided that only a limited number of counters is selected to be automatically verified as a part of this research, since the goal is not to verify all the counters at once, but to create a mechanism which enables easily configurable automatic counter verification in the future. In this case, it means that mechanism should be built in such a generic way that counter verification can be easily taken as a part of any type of test. Also, it was acknowledged that each of the selected counters increase the complexity of limit value calculations, so it was decided to select only moderate number of counters. That also supports the main idea of this research, which is to verify few counters as part of this research, and concentrate on easy configurability, limit value calculations and to verification that the system works. Broader counter verification in capacity tests can be started in future. There were also some limitations in simulated test environment that blocked certain counters to be automatically verified. This was noticed and 5 counters were selected.

**Table 2.** Counters that were selected to be automatically verified in the scope of this research.

| Selected counter | Reference to 3GPP specification |
|---|---|
| Average DL cell PDCP SDU (Service Data Unit) bit-rate | 3GPP TS 34.425, Version 13.5.0, Chapter 4.4.1.1 |
| Average UL cell PDCP SDU bit-rate | 3GPP TS 34.425, Version 13.5.0, Chapter 4.4.1.2 |
| Maximum DL cell PDCP SDU bit-rate | 3GPP TS 34.425, Version 13.5.0, Chapter 4.4.1.3 |
| Maximum UL cell PDCP SDU bit-rate | 3GPP TS 34.425, Version 13.5.0, Chapter 4.4.1.4 |
| DL PDCP SDU drop rate | 3GPP TS 34.425, Version 13.5.0, Chapter 4.4.3.2 |

After implementation started, and although we noticed that not all the counters are possible to verify in SCT, it was noted that 3GPP counter “Total Number of DL TBs (Transport Blocks)”, which was originally chosen as a part of this research, cannot be

verified in SCT, because it was not counter which is calculated in PDCP/RLC/MAC layer. It had to be dropped out from this research's scope. It was decided that a new counter will be selected to replace the counter that was dropped. Counter "DL PDCP SDU drop rate" was selected as a replacement. Final list of counters that are verified as a part of this research, is presented in the table 2. Table includes the references to 3GPP specification. Selected counters are described more descriptively above.

**Average DL cell PDCP SDU bit-rate** indicates the average traffic load that is going through PDCP SDUs on downlink (towards UE). It represents the incoming user plane traffic to the eNodeB. PDCP SDUs that are just by passed to another eNodeB are not included in this counter. Counter is updated when PDCP SDU is transmitted towards UE. This counter reports the average DL PDCP traffic throughput, and it is calculated with the following formula:

$$\frac{\sum Number\ of\ bits\ entering\ the\ eNodeB}{Measurement\ period} \tag{2}$$

As the formula 2 above shows, all the bits that enter eNodeB are summarised together and divided with the measurement period. Results are presented as an integer values and in kb/s.

**Average UL cell PDCP SDU bit-rate** indicates the cell bit-rate of PDCP SDUs on the uplink (from UE). Measurement presents the successfully received incoming user traffic. It also indicates the traffic load coming from UE. Counter value is increased when PDCP SDU is received from UE. To another eNodeB forwarded PDCP SDUs are not considered. Result is achieved with the following formula:

$$\frac{\sum Number\ of\ bits\ leaving\ the\ eNodeB}{Measurement\ period} \tag{3}$$

In formula 3, sum of number of bits that have left eNodeB is divided with the measurement period. Results are presented as an integer values and in kb/s.

**Maximum DL cell PDCP SDU bit-rate** presents the maximum PDCP SDUs' cell bit-rate on the downlink. Counter represents the rate of user plane traffic that is coming to eNodeB and it indicates the traffic load towards UE by reporting the maximum DL PDCP traffic throughput. Counter is updated when PDCP SDU is transmitted towards UE. Result is achieved with the following formula:

$$\max\left(\sum_{Samples} DL\ cell\ PDCP\ SDU\ bit\ rate\right) \tag{4}$$

Formula 4 presents how maximum DL cell PDCP SDU bit-rate is calculated. First, all the sampled intervals of DL cells are gathered together, and then maximum value is picked from them. Results are presented in integer value and kb/s which presents the maximum value.

**Maximum UL cell PDCP SDU bit-rate** presents the maximum PDCP SDUs' cell bit-rate on the uplink. Counter represents the rate of user plane traffic that is coming from UE and indicates the traffic load that is flowing past eNodeB, by reporting the maximum UL PDCP throughput. Counter is updated every time when a PDCP SDU is received from UE. Result is achieved with the following formula:

$$\max\left(\sum_{Samples} UL\ cell\ PDCP\ SDU\ bit\ rate\right) \quad (5)$$

Formula 5 shows how maximum UL cell PDCP SDU bit-rate is calculated. All the sampled intervals of UL cells are gathered, and then maximum value is picked from them. Results are presented in integer value and kb/s, which presents the maximum value.

**DL PDCP SDU drop rate** calculates a fraction of PDCP SDUs (IP packets) that are dropped on the downlink data transfer. Only user-plane traffic is considered as a part of this counter. All the packets which context is removed from eNodeB, and when none parts of the packets are transmitted on air interface, are considered as dropped packets.

$$\frac{\sum DL\ dropped\ PDCP\ SDUs}{\sum DL\ handled\ PDCP\ SDUs}\ x\ 100 \quad (6)$$

Formula 6 shows how DL PDCP SDU is calculated. Dropped PDCP SDUs are divided by handled PDCP SDUs, which includes dropped and successfully transferred PDCP SDUs. To get the percentage, the fraction is multiplied with 100.

## 4.2 Selected test cases

The goal of this research was to create a mechanism, which makes it possible to easily enable automated counter verification in any automated capacity test. Capacity tests consist from 500+ test cases, so to achieve this goal, it is necessary to verify counters from as diverse selection of test cases as possible. That is why different types of tests were selected as part of this research.

Test cases, where counter verification will be implemented, were chosen in the meeting with specification engineer of the case company. Case company's three most viable HW variants were chosen to be part of this implementation, to make counter verification easily configurable in the future. In this research, those HW variants are described as HW variant A, B and C. For all those three HW variants, three different types of tests were selected, so in total 9 (3 HW variants * 3 test types) test cases were chosen to be part of this implementation. Two test cases, where packet dropping was part of the test case, were added to verify counter DL PDCP SDU drop rate. To all those 11 test cases, 5 automatically verified counters (presented in previous chapter) are implemented, so in the end there is 55 different automatically verified counter values as a part of this research.

Test types included single UE, multi UE and traffic mix test cases. Single UE test cases measure the theoretical maximum data throughputs that can be achieved, and they have only one UE as a part of the test case. Multi UE test cases have multiple, limited amount of UEs as part of them. Their aim is to measure maximum data throughputs that can be achieved, when there are multiple UEs involved. Traffic-mix cases are similar than multi-UE cases, but the difference is that there is almost maximum amount of UEs involved, and other than data subscribers involved. Where single and multi UE test cases concentrated only to measure data throughputs, in traffic-mix case, there can be users that transmit data, make VoLTE (Voice over LTE) calls and do other activities. Since DL PDCP SDU drop rate counter was selected as a part of this research, two test cases where packet dropping appears needed to be selected. Those test cases were HW variant A packet dropping test case, and HW variant C packet dropping test case. Table

3 below illustrates the selected test cases, and the counters that were verified in each test case.

**Table 3.** List of test cases and the counters.

| Test cases | Counters verified in test case |
|---|---|
| HW variant A, single UE,<br>HW variant A, multi UE,<br>HW variant A, traffic-mix,<br>HW variant B, single UE,<br>HW variant B, multi UE,<br>HW variant B, traffic-mix,<br>HW variant C, single UE,<br>HW variant C, multi UE,<br>HW variant C, traffic-mix,<br>HW variant A: multi UE with packet discarding,<br>HW variant B: multi UE with packet discarding | Average DL cell PDCP SDU bit-rate, Average UL cell PDCP SDU bit-rate, Maximum DL cell PDCP SDU bit-rate, Maximum UL cell PDCP SDU bit-rate, DL PDCP SDU drop rate |

## 4.3 System Component Testing

Code base of the product in question can be divided into two parts; to application code and to (system component) test code. Product in question consists from different system components, where PDCP/RLC/MAC layer is one them. Each of the system components has millions of lines of code, and the product in question has almost total of 10 million lines of software code. System Component Testing's (SCT) purpose is to simulate the real component's behaviour, so it is working as a no-operational stub for the real application. In the case company, SCT is the first level of tests, where the whole PDCP/RLC/MAC layer is tested as a one integrated application. Test cases communicate directly with PDCP/RLC/MAC layer's public interfaces. Those interfaces are also provided to other system components.

Goal of SCT is to verify that system component is working under given performance, capacity, stability, and functionality requirements. SCT also makes sure that component is mature enough to be moved to higher level tests. SCT environment for this system component (PDCP/RLC/MAC layer) is written mainly with Robot Framework and Python. A lot of Robot Framework keywords are extended with Python, and communication between the PDCP/RLC/MAC layer application in eNodeB and Test PC is handled with Python. Robot Framework is basically used to make test cases and suites, and to define test case behaviour, when Python is mainly used to handle more complicated parts of SCT. Python is preferred in more complicated tasks, since it is faster than Robot Framework (Robot Framework, 2017).

Figure 8 represents the SCT setup in PDCP/RLC/MAC layer. SCT tests are run in real hardware, which in this case means the eNodeB. SCT test setup is as follows: there is one test PC attached to eNodeB via router. The test PC takes care of running the test suite, which includes the test cases that are run. Tests are started by downloading the newest software to the test PC, which means the build that includes developer's changes. Robot Framework handles that phase. Then test PC runs tests on SUT (eNodeB) with Robot Framework, which is also responsible of gathering the test results, logs, and crash dumps (if necessary) from the target. Communication between eNodeB

and Test PC is handled by open-source network protocol testing library called Rammbock. Test PC is connected remotely to run the tests in it.

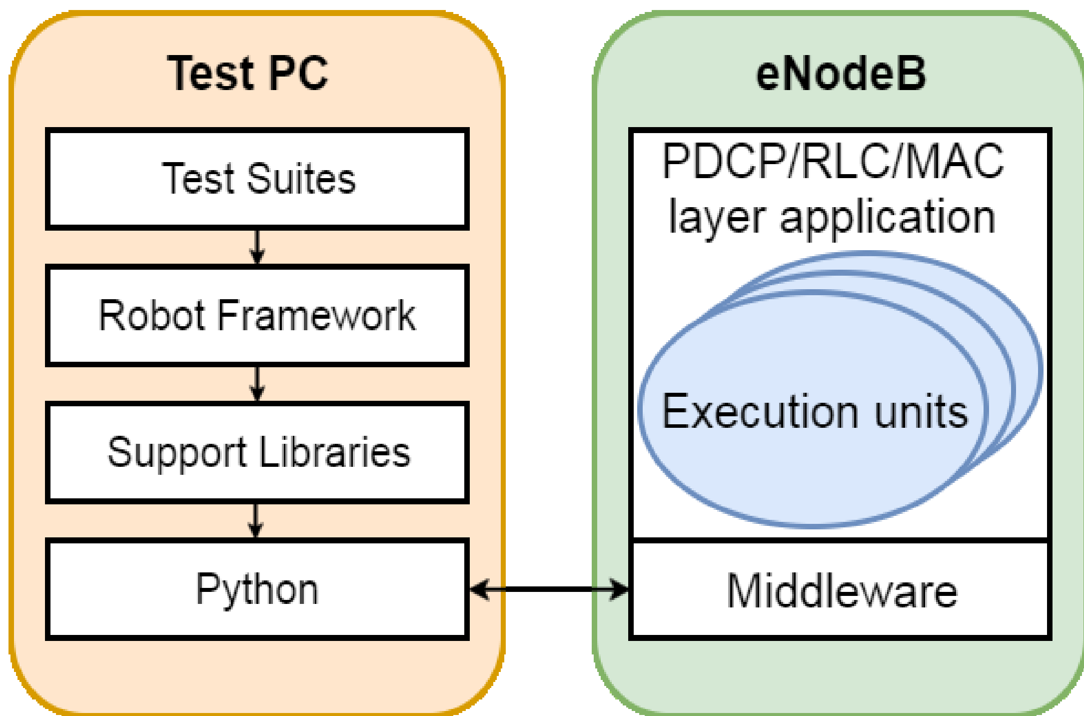



**Figure 8.** PDCP/RLC/MAC layer SCT setup.

## 4.4 Reading counter values in SCT

The code base can be divided to the application code, and to the test code (SCT). Counter value calculations are handled in application code, where all the counter values are updated continuously, every time their values have been changed. To read counter values in SCT, counter values need to be gathered from the application code.

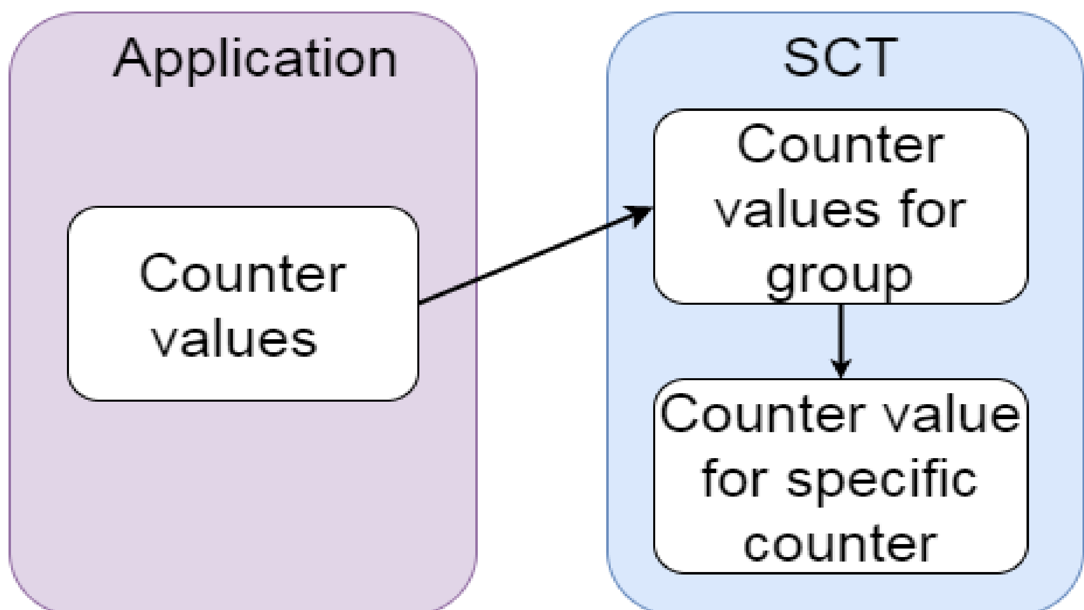



**Figure 9.** Counter value reading in SCT.

Every counter belongs to its own counter group, which consists from similar type of counters. Like Figure 9 presents, to verify counter in SCT, counter values need to be

gathered for the whole counter group, and after that counter value is possible to be read from the group's counter values. Counter value collecting in SCT is handled with the already implemented API that collects the counter values for each group separately. API can handle a one group at a time, so if two separate counters that belong to different groups are read, correct counter group needs to be gathered first, and then it is possible to check value of the counter. If counter values are gathered from the same group, then it is not necessary to retrieve the counter values again. Because PDCP/RLC/MAC layer is tested in SCT, only the counters that are measured in this layer, are possible to verify in SCT.

Counters have been verified in SCT earlier, but there has not been a general way to do it, and custom made Robot Framework keywords have been used. Same goes with the capacity tests, where tests have covered only a couple of counters. Earlier implementations were not able to use any other counters, that were defined in Robot Framework keywords. Robot framework example below illustrates how counter values have been checked earlier in capacity tests.

```
Check values for example counters
    ${expected_value_1} =   Set Variable   10000
    @{counter_value_list} =   Get values for example counters
    ${example_value_1}  =        Get from list
                                         ...  ${counter_value_list}
                                         ...  0
    Should be true   ${example_value_1} > = ${expected_value_1}

    ${expected_value_2} =   Set Variable   20000
    @{counter_value_list} =   Get values for example counters
    ${example_value_2}  =        Get from list
                                         ...  ${counter_value_list}
                                         ...  1
    Should be true   ${example_value_2} > = ${expected_value_2}

Get values for example counters
    Gather_counter_group_to_database   ${example_counter_group}
    ${value_1}=   Get_counter_value_from_database
    ${example_counter_1}
    ${value_2}=   Get_counter_value_from_database
    ${example_counter_2}
    @{counter_values}=    Create list
        Append To List
        ...  ${counter_values}
        ...  ${value_1}
        Append To List
        ...  ${counter_values}
        ...  ${value_1}
    [return]   @{counter_values}
```

Keyword *Check values for example counters* illustrates a keyword that is used for checking that counter values are bigger than predefined values. *Expected_value_1* and e*xpected_value_2* demonstrates the fixed value, which counter should exceed. Counter values are saved to list *counter_value_list* which calls keyword *Get values for example counters.* Keyword *Gather counter group to database* gathers the counter values from the counter database. After gathering, values can be read from database, and there is no need to gather counter values again, if the counters belongs to same group. Then values are added to list *counter_values* which is returned from the keyword. After that, values that *Get values for example counters* retrieves, are compared to expected values. Robot keyword *Should be true* fails the test case, if *example_value1* or *example_value2* are smaller than the expected ones.

This kind of counter verification most certainly works, but the problem is that these keywords needs to be implemented for every counter separately and limit values are not changeable for every test case. In other words, this is not a dynamic solution. Other problem is that counter group needs to be known, in order to be able to verify counter value. Also, there is several hundred of different counters in product in question, so checking the group every time causes a lot of unnecessary work. SCT capacity tests consists from about 500 different test cases, so solution that would be easy to take in use with all the test cases with different setups would be idealistic. Test cases vary from each other, since they try to simulate different use cases. For example, amount of simulated UEs, amount of transferred data or number of cells and pools can vary depending on the test case.

The term pool in this case means the functional part of eNodeB, which is capable for different kind of operations fulfilling certain requirements. It can be for example two circuit boards of eNodeB, which together can do certain things. Sometimes it is possible that person who is running the test, is interested only about the functionality of a certain pool or cell, and if that would be taken in use with current method, a whole new keyword should be implemented. Other problem with aforementioned Robot Framework example is that it only checks that value is bigger than limit value, so there is no maximum value that is checked for result.

## 4.5 Design of the new system

The goal of this research is to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. That is possible to achieve with the current Robot Framework based solution that was described in Chapter 4.5., but verifying counter values like that is requiring a lot of extra work, since all the counter should be verified separately by using different keywords.

Demand for system which would be easily configurable for each ~500 capacity test cases was obvious. That highlighted the fact that a new, different solution was needed. Robot Framework is a widely used and very efficient framework for acceptance testing, but it tends to be a little bit slower than a pure Python based solutions. Especially counter group data can come in huge chunks, which would be slow to parse with Robot Framework, it was decided that Python will be used for the implementation. Like stated earlier in chapter *3.2.4 Python*, Python is easy-syntax, fast and efficient high-level programming language that can easily take care of this kind of problems. It was decided to start with a small set of test cases (described in chapter 4.2.) to verify that the system works with different HW variants and then it can be expanded to other test cases, and finally, to other tests than capacity tests as well.

In Robot Framework based solution the data for each counter group needed to be gathered separately from the counter database and after that it was possible to verify counter values. That seemed like a thing that should be done together as a one process. That would make counter verification much easier, since it would not be necessary to know which group the counter belongs. Thus, it was quickly decided that the new system should be designed in a way that only the counter name needs to be given and the new system handles the counter group data collection and gathers counter value from that.

In design phase, it occurred that it is possible that there is use cases, where counter value needs to be checked for a certain pool or cells, instead of the “normal” value

reading for all the pools and cells that are created as a part of the test case. Thus, it was decided that there should be a feature, which allows the counter measurement for pre-defined amount of pools or/and cells. Also, it was decided that there should be a possibility to verify counter with all the pools (without predefined pools and cells), but still print all the cell values for counters. That makes it possible to check counter values normally, and still be able to know how values are developing per cell.

In chapter *4.6. Reading counter values,* there is an example where test case is failed, if counter value does not meet the requirements. In some cases, it is possible that failing the test case is not the desired outcome of the counter verification, but it would be beneficial to just read and print the counter values. Therefore, it was decided to give options for counter verification. Three different options were decided to be implemented: "fail test case, if value is not in range", "console print value" and "console print value, if value is not in range". Latter two ones provide some extra options, which can be handy especially in limit value calibration and with debugging.

It was decided that system verification should be range-based, which means that result value needs to be in range of defined minimum and maximum value. Those values can be presented as integer values, where min and max are both defined in test case description. Additionally, it was decided that the possibility for percentual range is necessary. That means expected value can be declared with a percent, where the result value needs to fit inside the range that expected value and percentual difference form together.

All the counters are not necessarily retrievable straight from the application code. Some counters need some extra calculation, and counter DL PDCP SDU drop rate, which is part of this research, is a counter like that. Thus, the system needs to be extendable for the counters, that need some special calculations. Ideally this system should be usable in other tests than capacity tests. This needs to be carefully considered in system design.

User stories are the comprehensive way to describe requirements, and they have gained their popularity with the rise of Agile methodologies. User stories are presented in a following form: "As a (role) I want (something), so that (benefit)". Developers and customers can easily read them, since all the technical jargon is reduced from them. User stories are perfect sized, since they divide bigger entities to smaller pieces. User stories are also made for iterative planning. DoD (Definition of Done) is a checklist, that describes what needs to be done in order to complete the user story. User stories also support empirical design, which means that design of the new system can be done by studying prospective users (in this research LTE PDCP/RLC/MAC layer SW developers), in typical situations. (Cohn, 2004.)

Since user stories are perfect fit for iterative planning, and they support empirical design, they were decided to be used for defining requirements. Following user stories illustrate the requirements of the new counter verification system. At first user story ID is represented, then user story itself is written, and finally the definition of done (DoD) of each user story is determined. In all user stories "role" is developer, which means LTE PDCP/RLC/MAC layer SW developers, since they are end users of the system. Naturally, DoD for all these user stories includes the "essentials", like code complete, written unit tests, comments in code and integration tests, so they are not explicitly presented in each user story.

**ID:** US-1
**User Story:** As a developer, I want to enable counter verification in any test case in capacity tests, so that I can verify counter functionality in capacity tests.
**DoD**: Counter verification can be enabled in any capacity test case.

**ID:** US-2
**User Story:** As a developer, I want that counter verifier system is written in Python, so that I can get quicker feedback from capacity tests (since Python is faster than Robot Framework).
**DoD:** System is implemented with Python.

**ID:** US-3
**User Story:** As a developer, I want that counter verifier system gathers counter group automatically, so that I do not need to know the counter group when I activate counter verification.
**DoD:** Counter data is available without knowing the counter group.

**ID:** US-4
**User Story:** As a developer, I want that value comparison is made with range values, so that I can set more precise limits to expected values.
**DoD:** Value comparison is done with range (e.g. min limit <= result <= max limit)

**ID:** US-5
**User Story:** As a developer, I want that expected value can be set as a value with percentual margin, so that I can set changeable percentual limits to counter verification.
**DoD:** Min and max limits can be set as percent from expected value (e.g. min limit = expected value – expected value * range %)

**ID:** US-6
**User Story:** As a developer, I want that expected value can be set as minimum and maximum value, so that I can set range for counter verification.
**DoD:** Min and max limits can be set as two integer values.

**ID:** US-7
**User Story:** As a developer, I want that test case is failed, when value is not between expected limits, so that I know when counter verification fails.
**DoD:** When result is not between limits, test case is failed.

**ID:** US-8
**User Story:** As a developer, I want to have options when expected counter value is not met, so that I have better debugging abilities in such a situation.
**DoD:** If it is declared in test case description, it is possible to not meet the limit values, and not fail the test case.

**ID:** US-9
**User Story:** As a developer, I want that counter printing is possible to do for each cell value, so that I have better debugging capability.
**DoD:** Counter value is possible to be printed for each cell.

**ID:** US-10
**User Story:** As a developer, I want that system is extendable for counters that need extra calculations, so that I can verify all kinds of counters.

**DoD:** Custom counters can be verified in system.

**ID:** US-11
**User Story:** As a developer, I want that value can be verified from specific pool(s) or cell(s), so that I have better options on verification.
**DoD:** Counter values can be verified from specific pool(s) or cell(s).

**ID:** US-12
**User Story:** As a developer, I want that counter verification system is replicable to another than capacity tests, so that I can perform counter verification in other types of tests.
**DoD:** Counter verification system can be adapted to other than just capacity tests.

## 4.6 Implementation of the counter verifier

Like said earlier, the system was decided to be implemented with Python. Test-Driven Development (TDD) was used for the implementation. Test-driven development is a technique where unit tests are written before the actual implementation. TDD consists from three main steps, which are followed repeatedly. Those steps are: 1. *write a test for the next bit of functionality you want to add.* 2. *write the functional code until the test passes.* 3. *refactor both new and old code to make it well structured.* These steps are cycled one unit test at a time, to build a functional system (Martin Flower, 2017.) This seemed a suitable approach, since running a one test case in SCT environment can take several minutes, which would have slowed the feedback time dramatically. With TDD, development was much quicker, and it dramatically decreased the need to test the system in real eNodeB.

Creation of the artifact started with the implementation of the feature, which automatically checks that which group the counter belongs. That most certainly sounded like a thing that one class should do. Class was named self-descriptively as *CounterGroupFinder*. Every counter has two different IDs. In this case, one ID (id1) was used to poll the counter value from counter database, and another ID (id2) is used more in system specifications, but id2 consisted from the counter group ID and counter number, so the counter group was parsed from id2. The latter IDs were not available in SCT code base, so a text file which consisted all the counter names, and the missing IDs was generated. Like Figure 10 below presents, CounterGroupFinder has one public method, *get_counter_group_for_counter()*, which returns the counter group with the support of the helper methods of the class.

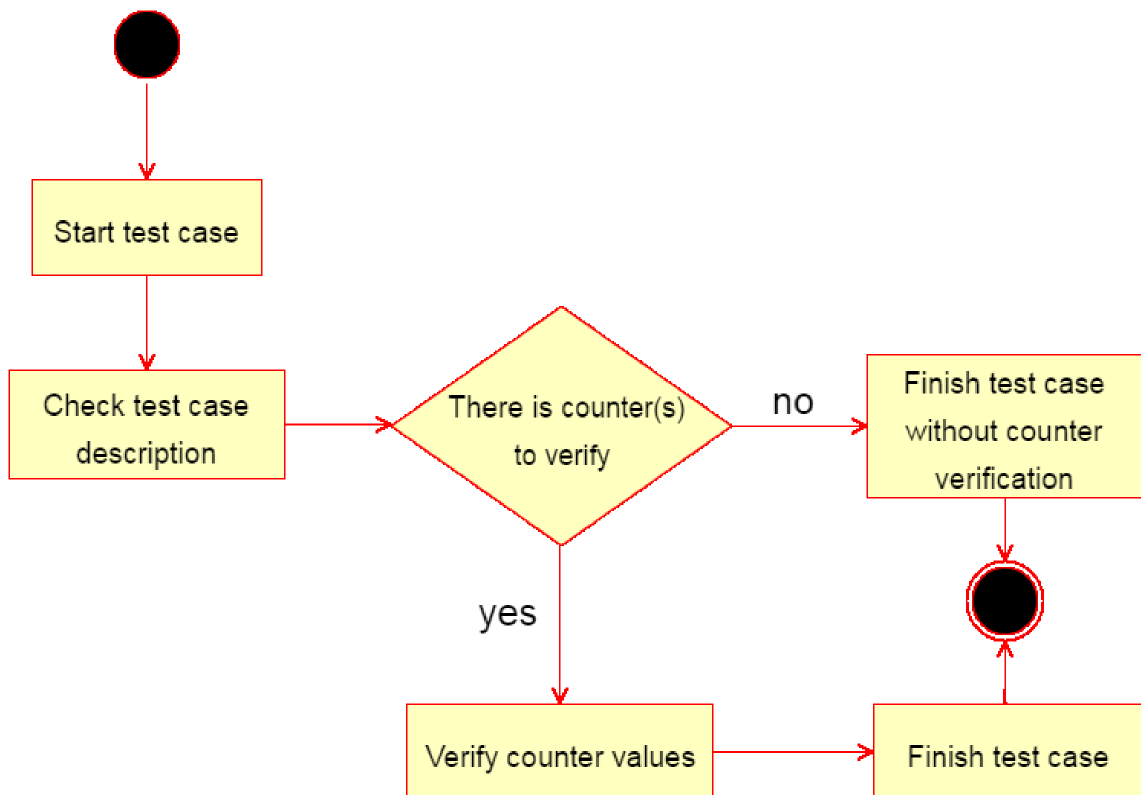



**Figure 10.** Counter verification triggering in test case.

Next part was to design, how counters that need to be verified, were introduced to the test case. Figure 10 shows how the logic works within each test case. At the beginning of the test, test case description is checked. If there are any counters declared, counter verification is triggered during the test run. It was decided that each test case would have own parameter called *counters_to_verify*, which is Python dictionary that consists from all the necessary information of the counter(s). Python dictionary can be seen as an unordered set of “key: value” pairs. Every key in dictionary must be unique (Python, 2017). Test case description example below, that is written in Python, shows how counters to verify can be declared to each test case.

```
counters_to_verify = {'EXAMPLE_COUNTER_NAME_1':
                       {'action': failTestCaseIfNotInRange,
                        'printCellValues': True,
                        'cellsToWatch': [1],
                        'percentualRange':
                           {'limit': 100000,
                            'range_%': 5 }},
                     'EXAMPLE_COUNTER_NAME_2':
                       {'action': 'consolePrint',
                        'limitValues':
                           {'min': 1000,
                            'max': 5000 }}}

ExampleTestCase = TestCaseDescription(
    counters_to_verify = counters_to_verify,
    test_duration = 10s,  #example parameter
    number_of_ues = 100  #other example parameter
    ...
    ...)
```

*Counters_to_verify* is a Python dictionary, where every high-level key is the counter name. Python dictionaries can have dictionaries inside dictionaries. In this case, name of

the counter is the key for the counters_to_verify dictionary, which has the information about the counter. *Action* shows that what is going to be done after the counter value is verified. Action has three different options: *failTestCaseIfNotInRange, consolePrintIfNotInRange* and *consolePrint.* FailTestCaseIfNotInRange fails the test case, if counter value does not stay within the range, consolePrintIfNotInRange prints warning to console, if value is not in range, and consolePrint prints the achieved value regardless it is in range or not. *PrintCellValues* determines that whether counter's cell values are printed to console.

By default, cell values are not printed to console (printCellValues is set to False by default), and it is not necessary to explicitly declare *printCellValues* at all. Same goes with other optional parts, like cells and pools to watch. Like stated earlier, there is two different ranges to use for verifying of the value. *PercentualRange* checks if counter value is within the percentual range and *limitValues* have one minimum and maximum value, and counter value needs to fit between them. *CellsToWatch* tells that which cell's values are checked, and similarly it is possible to check values of different pools, with *poolsToWatch,* which is not used in the example above. By default, all pools and cells are checked.

After creating the aforementioned features, class *CounterVerifier* was created. CounterVerifier has all the main functionalities of the system. Its main functions are to get counter information from the test case description (Figure 11), to get counter values for each counter, check that counter values stay between the declared limits, and to print values, if necessary. That sounds like quite a lot of work to be done by one class. That is why *CounterValueReader* and *CounterPrinter* were created to reduce the workload of CounterVerifier. After that change, CounterVerifier's responsibilities consisted only from checking what needs to be done for each counter, checking the values that are coming from CounterValueReader and passing those details CounterPrinter, if needed. CounterVerifier's public method *verify_counter_values_are_in_limits()* is accessed from Robot Framework, if there are any counters to verify defined in test case description, like presented in figure 10.

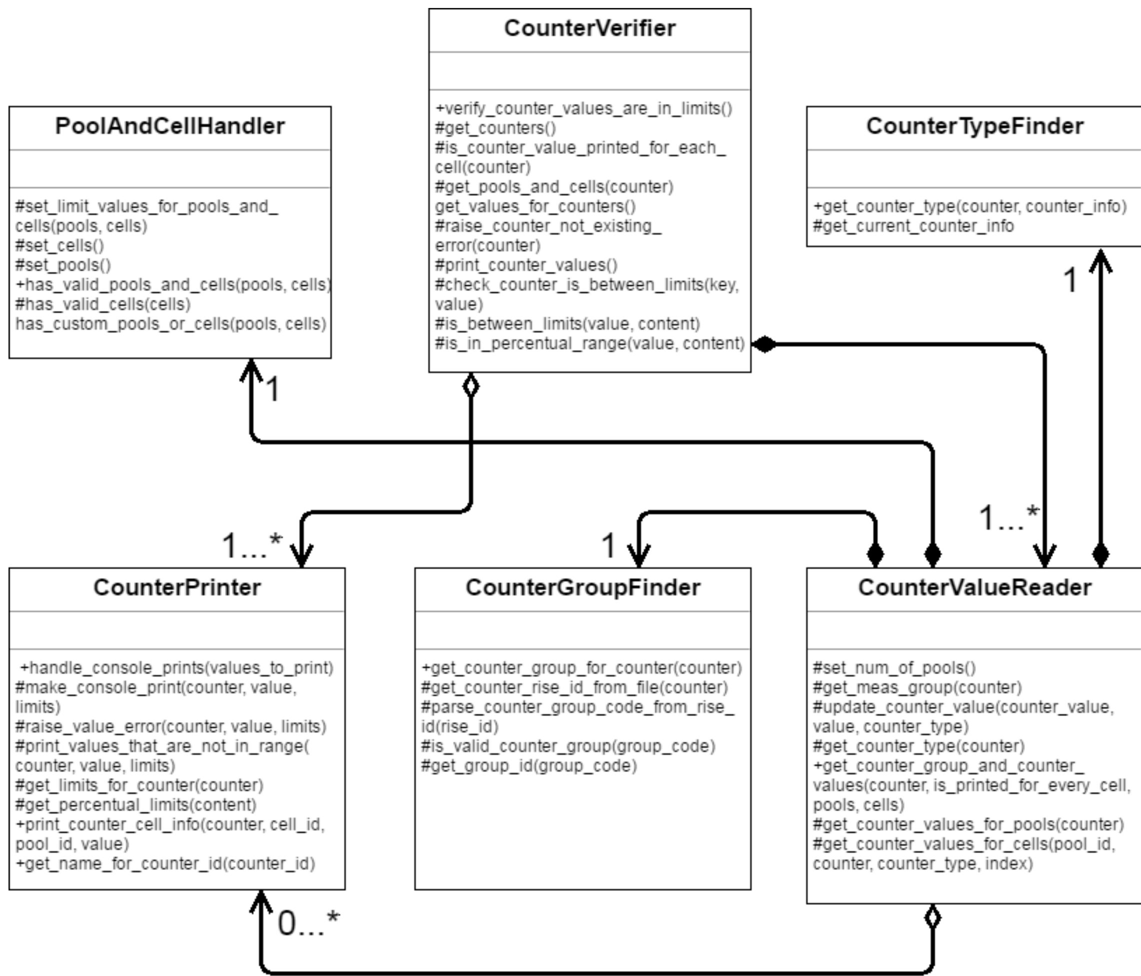



**Figure 11.** Class diagram of the implemented counter verification system.

CounterValueReader takes care of getting the values for each counter, whether for predefined pools and/or cells, or for all pools and cells. CounterValueReader has four helper classes: *PoolAndCellHandler, CounterTypeFinder*, CounterPrinter and CounterGroupFinder. PoolAndCellHandler checks, if there is correct amount of pools and cells in counters_to_verify dictionary, and sets cell and pool values. CounterTypeFinder simply finds out, if counter is measuring minimum, maximum, average or sum value. That information is valuable, when calculations from specific cells or pools are done. Class diagram in Figure 11 above demonstrates the classes that were created in this implementation, and the relations between the classes. Only CounterVerifier can have one-to-many relations to other classes. CounterVerifier loops through all the counters, and can create multiple instances of CounterValueReader and CounterPrinter, if there are more than one counter to be verified. CounterValueReader can have only one instance at a time to its helper classes, and only CounterPrinter makes an exception. If counter value prints for each cell are enabled and there is more than one cell in test case, CounterValueReader can create multiple instances to CounterPrinter.

In the class diagram above (figure 11), class methods starting with "+" are considered as "public methods", and methods starting with "#" are protected, internal methods. In class diagram, there is one or two public methods in each class and "protected" helper methods. Although there is no rule for method size in object-oriented programming, methods should be small, and do only one thing. (Martin, 2009). That approach was applied in this implementation. Each class have one or two public methods, and a lot of helper methods. In strongly typed object-oriented languages, like C++ and Java,

encapsulation has big role in programming. Every method of the class is either public, private or protected, and only public methods are visible to another classes, providing the API to use. Python does not have any declarations that tell about the visibility of the class members, and Python does not force programmer to build access control for classes (Goldwasser & Letscher, 2008.) In Python programmer bears the responsibility, and Python relays on “we are all adults philosophy”. Rather than using private methods, it challenges the programmer to ask herself: *“who are you protecting the attribute from?”* Capsulation in Python is handled with naming conventions. Class elements that start with single underscore are considered as a ‘internal’, and it tries to tell others that “do not use this if you don’t fully understand how this works” They still are visible to others, and methods starting with underscore are accessible from other classes, so it is just a convention (Code like a Pythonista, 2017.)

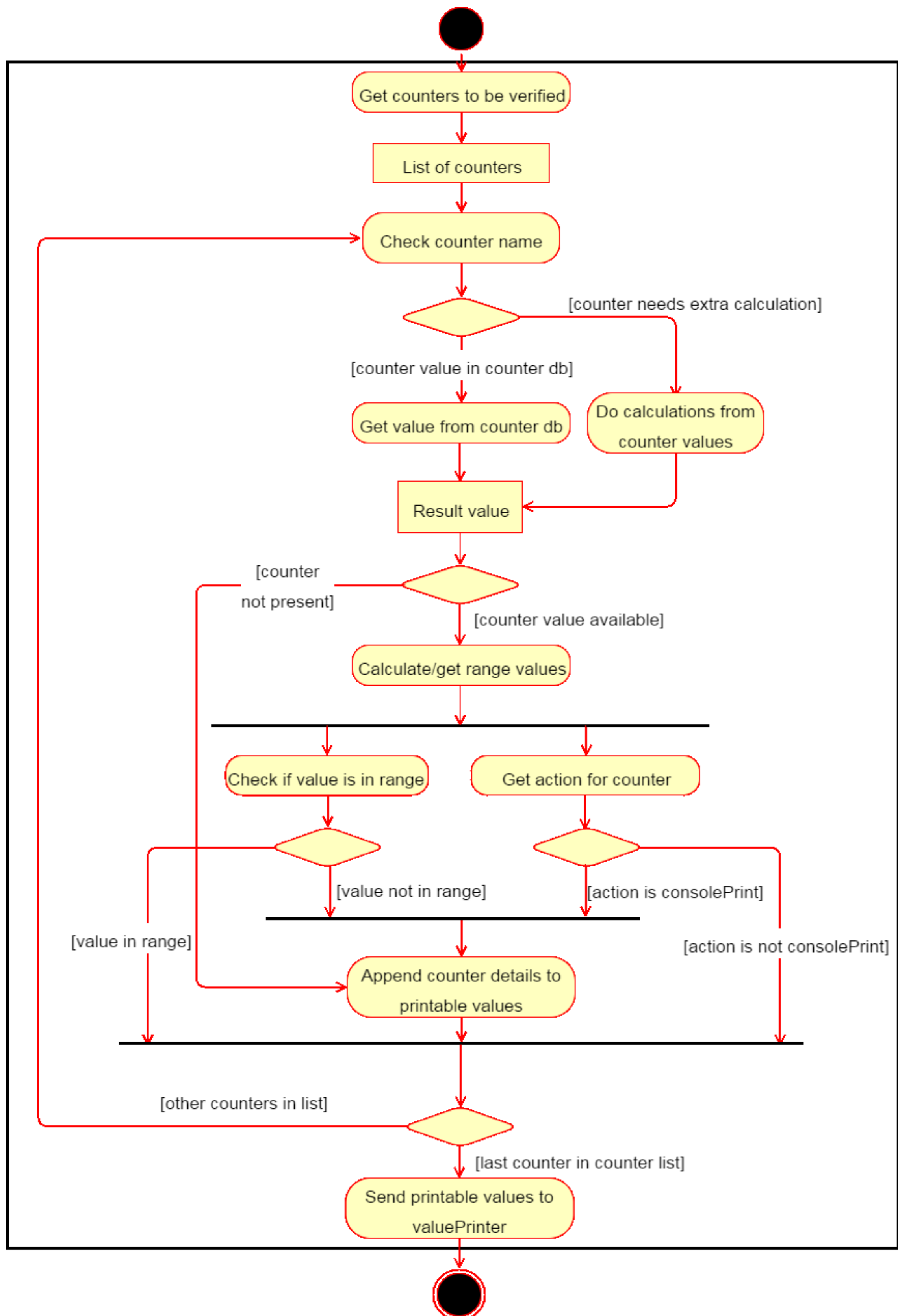



**Figure 12.** Activity diagram of the CounterVerifier's *verify_counter_values_are_in_limits()* method.

Like said earlier, verify_counter_values_are_in_limits() is executed from Robot Framework code, if there are any counters to verify in test case description. That is because capacity tests are Robot Framework based, which means that the basic structure

of capacity tests is made with Robot Framework. Activity diagram in figure 12 above describes how the method verify_counter_values_are_in_limits() works. At the beginning, all the counter values are retrieved from the helper method *get_values_for_counters().* With the help of CounterValueReader class, the method retrieves names and values of all the counters that were introduced in test case description. Then one counter is examined at once. At first it is checked that if there is some integer value for counter. Other ways, if counter value is not available for some reason and “Not present” is printed to console, but it does not fail the test case, if “failTestCaseIfNotInRange” is not selected.

If received value is an integer value, it is checked that if value is between the limits. Limits can be percentual, or pair of limit values. In the same time the desired action for counter is retrieved. In CounterVerifier’s *verify_counter_values_are_in_limits()* method, a new Python dictionary is created. Details of counters that are not between the limits and counters which action is “consolePrint” are added to it. Details include counter name, value, and limit values. After all counters have been checked, Python dictionary is passed to CounterPrinter’s handle_console_prints() method, which takes care of printing the values and fails the test case if necessary.

Counter DL PDCP SDU drop rate is calculated, like said earlier, from the fraction of discarded and handled DL PDCP SDUs. That counter is not implemented to the application by default, but it needs to be calculated as the result of two separate counters, one measuring discarded DL PDCP SDUs and another transferred DL PDCP SDUs. That means the generic solution (CounterVerifier), which was presented earlier, is not usable in this case in the form it was presented. That is because solution is made for asking counter values from counter database, and making the limit calculations based straight on those results coming from the database. That fact was noticed while designing the CounterVerifier, and it was planned that CounterVerifier was developed for already implemented counters at first, and it can be extended to work with “custom”, semi-automatically calculated counters later, as external plugins to the systems.

Like said earlier, DL PDCP SDU drop rate is calculated from the fraction of discarded, and handled DL PDCP SDUs. In application, there are counters measuring discarded and passed DL PDCP SDUs. Handled DL PDCP SDUs is calculated by adding those aforementioned counters together. This means that in order to calculate DL PDCP SDU drop rate, four different calculations are needed. First, handled DL PDCP SDUs, need to be calculated, but that requires calculating passed and discarded DL PDCP SDU, too. After that, it is possible to calculate the fraction. In order to keep CounterVerifier as easily maintainable blocks, and because of the complexity of the DL PDCP SDU calculation, it was decided that a new class should be created, which takes care only about DL PDCP SDU calculations. That seemed a decent way to ensure the readability of the code. If there are any extensions coming in future, they should also be implemented as their own class, to ensure that the code would not get too messy.

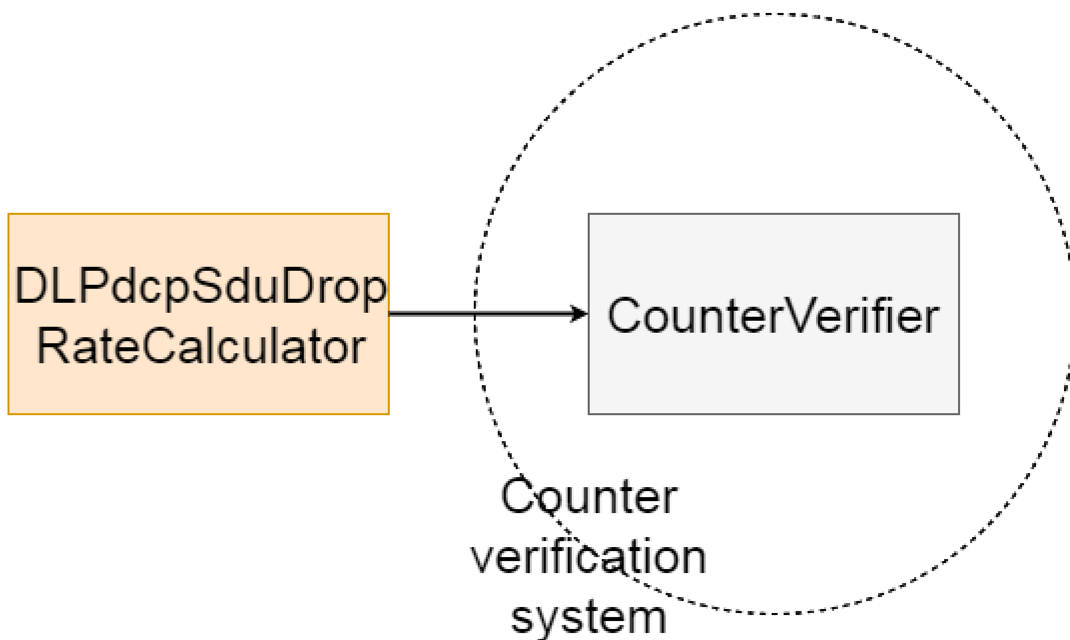



**Figure 13.** Example how counter verification system can be extended with counter, that does not exist in application, but needs to be calculated separately.

Figure 13 illustrates how the DL PDCP SDU drop rate is implemented in counter verification system. DlPdcpSduDropRateCalculator class calculates the value for DL PDCP SDU drop rate counter, if CounterVerifier asks it to do so. CounterVerifier provides an interface for DlPdcpSduDropRateCalculator. CounterVerifier requests to count the value, if DL PDCP SDU drop rate is one of the counters that needs to be verified. Same kind of extensions can be applied to other counters that are not coming directly from the application, but need some custom calculations.

## 4.7 Limit value calculations

When the new counter verification system was ready, next stage was to start the limit value calculations. Limit values for each counter, and for each test case in every hardware variant needed to be calculated manually based on test case parameters. Five counters were selected as part of this research. Three test cases were selected for each of the different hardware variants. For DL PDCP SDU drop rate, test cases that included packet dropping needed to be selected, to verify functionality of the counter. Typically packet dropping is not part of the test case. Two test cases were selected in order to verify that counter's functionality, which raised the number of test cases to 11, where five counters are verified in each test case. That means there was total of 55 different counter - test case combinations as part of this research. For all those test case – counter combinations, excepted values needed to be calculated separately.

For single UE test cases, it was decided that 2 percent range between the result and expected value was accepted. For multi-UE and traffic mix test cases, 5 percent range was acceptable. This meant if the expected value was for example 100, 105 would be the maximum and 95 the minimum value that was accepted, otherwise test case would fail. Limit value calculations turned out to be more complex than originally estimated. There was a person in case company, who had done similar calculations earlier, and he had developed even the Excel-based tool to calculate values easily from test case parameters. In the beginning, this tool seemed to work nicely for the counters, which measure maximum DL and maximum UL cell PDCP SDU bit-rate. After counters that measure average values were started to be verified, it was noticed that in short 5 second

test cases, values are continuously too low compared to expected value. Majority of the current capacity tests are 5 seconds long, so this was very problematic.

This finding started a broader investigation on SCT capacity tests, which revealed that SCT environment is the bottle neck. In ideal conditions, like in the test cases that are part of this research, average value should become very close, or even equal with the maximum value. Ideal conditions mean that there are no issues like connection quality problems which are very typical in real-life environment, and have negative impact to achieved data rates. Typically, test cases last 5 seconds, and when average bit-rate counters were tested in longer single UE and multi UE test cases, there was no problems with average counters. It was noticed that if average bit-rate counters are going to be verified, test case needs to be longer than five seconds.

It was noticed that in SCT capacity tests, at the beginning of the test run there is a little gap in the bit-rate before it increases to the achieved level. This is caused by data generator, which makes all the necessary preparations, before starting to generate data, which causes a gap in the beginning of the data-rates. This gap in bit-rate decreases the average value so significantly in short test cases, that even the 5 percent range does not help to pass the test case. This was important finding from counter verification and from limit value calculation perspective. Average counters malfunctioned so badly in short test cases, that it was decided they will be left out of them in the scope of this research. Average counters did not provide comparable results in the test cases that were part of this research. In the other words, this means all the single UE, multi UE and traffic mix cases, which were selected as part of this research (18 test cases), were drop out.

When both Average DL cell PDCP SDU bit-rate and Average UL cell PDCP SDU bit-rate were dropped from all but packet discarding test cases, the number of test case – counter combinations decreased dramatically. Packet discarding test cases lasted both 30 seconds, which means that average counters can be verified in some of this research’s test cases, but sample size from average counters decreased dramatically. Table 5 below presents the final status of what counters are measured in which test case.

**Table 5.** Final list of test cases and the counters.

| Test case | Counters verified in test case |
| --- | --- |
| HW variant A, single UE,<br>HW variant A, multi UE,<br>HW variant A, traffic-mix,<br>HW variant B, single UE,<br>HW variant B, multi UE,<br>HW variant B, traffic-mix,<br>HW variant C, single UE,<br>HW variant C, multi UE,<br>HW variant C, traffic-mix | Maximum DL cell PDCP SDU bit-rate,<br>Maximum UL cell PDCP SDU bit-rate,<br>DL PDCP SDU drop rate |
| HW variant A: multi UE with packet discarding,<br>HW variant B: multi UE with packet discarding | Average DL cell PDCP SDU bit-rate,<br>Average UL cell PDCP SDU bit-rate,<br>Maximum DL cell PDCP SDU bit-rate,<br>Maximum UL cell PDCP SDU bit-rate,<br>DL PDCP SDU drop rate |

Like table 5 shows, only in test cases where packet discarding was part of the case, all the counters were verified in the test case. This was caused by the aforementioned

average counters' malfunctioning in the 5 second test cases, which led to dropping off the average counters from short cases. Only the packet discarding cases were long enough for average measurements, and they are the only test cases, where all the counters that were part of this study, are enabled. Total number of verified counters dropped from 55 counters to 37, because of this reduction.

Limit value calculations did not get any easier after this average issue was solved. There was a lot of uncertainty how values should be calculated, caused by the test runs where expected values were not reached. When counter did not return expected value, that caused some doubts if expected value is miscalculated in the first place, or if the whole calculation process should be changed. It turned out that because the limit value calculations were complex, gaining the reliability to expected values requires some time and routine for calculations. For each counter, calculation process was different. Calculation process involved a lot of test case data, and the knowledge of system specification engineers was mandatory in order to find out the correct values of each counter-test case combination. Finding the way to calculate correct limit values turned out be time consuming, and it was more challenging than just implementing the system for counter verification.

In system's development phase, unit tests helped to keep feedback times low, but the same thing was not possible in limit value calculations, since values come from the application. One test run to verify calculated limit values in real eNodeB, can take over from 10 to 30 minutes, depending on the queue, which slowed down the limit value testing as well. After all, despite all the difficulties, the calculation methods for each test case – counter combinations were found. After the correct methods were found for one of the counters, the rest of similar counters' calculations were executed quickly, by applying the same formulas to other test cases. Next step was to gather data from test runs, to evaluate the accuracy of the calculations. To verify the limit value calculation accuracy, each test case that were part of this research, was run 5 times to gather sample data. The results from those runs are described in the next chapter.

# 5. Evaluation

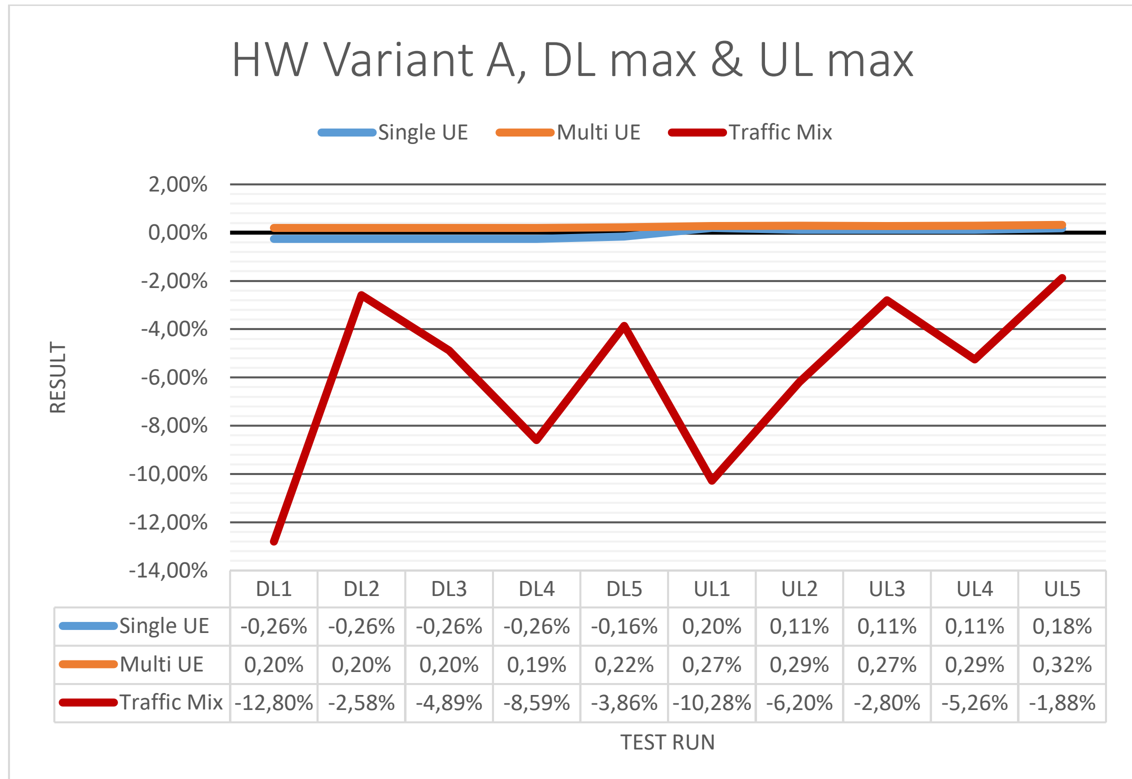
In this chapter, the results that this research provided, are evaluated. Evaluation is separated to two different sections. The first section analyses and illustrates the gathered data from test runs. In second section, evaluation against user stories is made.

## 5.1 Counter value calculation accuracy

In this chapter, results of the test runs are described and analysed. At first, single UE, multi UE and traffic mix cases are analysed for each HW variant. That is followed by the analyse of packet dropping test cases. Finally, all the counter value calculation results are summarized, and evaluated.

### 5.1.1 Single UE, Multi UE & Traffic mix test cases

In total, 37 different counter – test case combinations were as part of this study. All those test cases were run five times each, to gather information about the functionality of the counters. Test cases were run in a row, where for example HW variant A's all single-UE, multi-UE and traffic mix cases were run one after the other, to ensure counter values function in situations where multiple cases are run in a row. In other hand, they were run as a single test cases as well, to verify running a single test case works as it should.

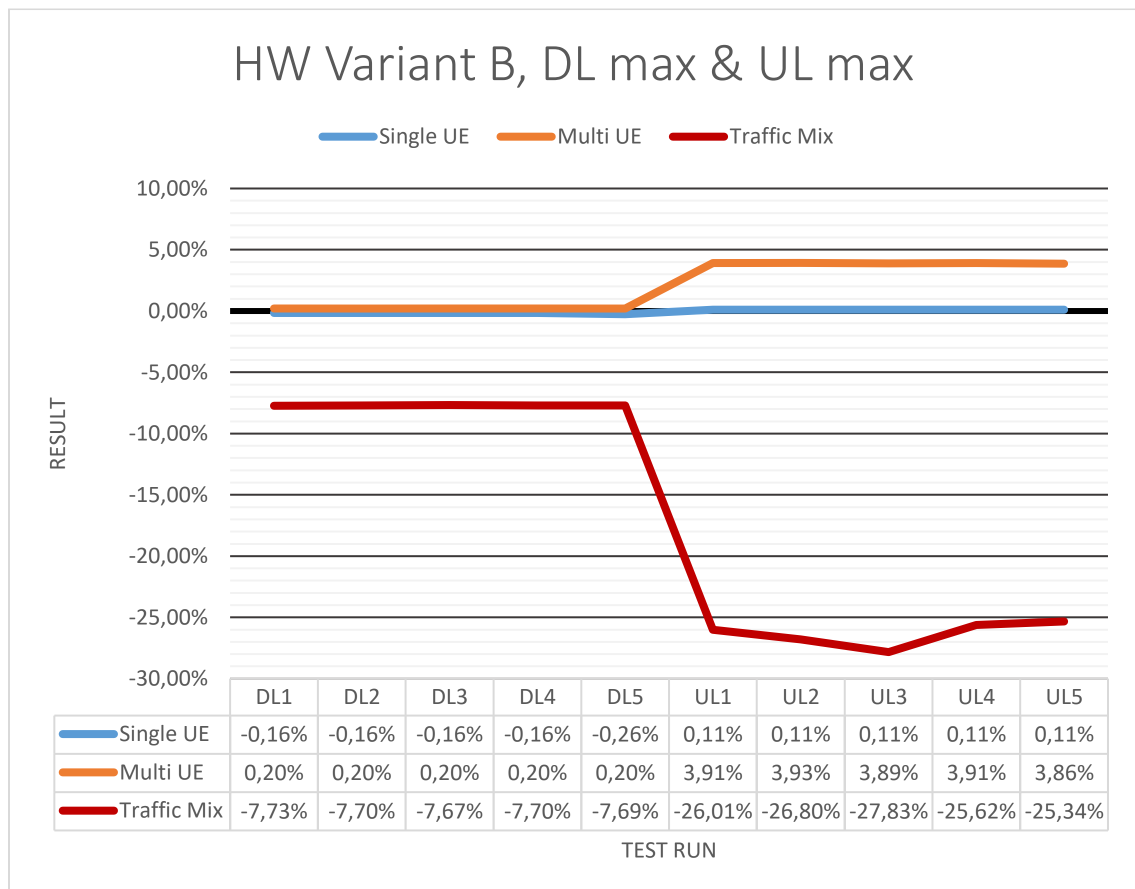



| | DL1 | DL2 | DL3 | DL4 | DL5 | UL1 | UL2 | UL3 | UL4 | UL5 |
|---|---|---|---|---|---|---|---|---|---|---|
| Single UE | -0,26% | -0,26% | -0,26% | -0,26% | -0,16% | 0,20% | 0,11% | 0,11% | 0,11% | 0,18% |
| Multi UE | 0,20% | 0,20% | 0,20% | 0,19% | 0,22% | 0,27% | 0,29% | 0,27% | 0,29% | 0,32% |
| Traffic Mix | -12,80% | -2,58% | -4,89% | -8,59% | -3,86% | -10,28% | -6,20% | -2,80% | -5,26% | -1,88% |

**Figure 14.** HW variant A's test case results from counters maximum DL cell PDCP SDU bit-rate & maximum UL cell PDCP SDU bit-rate. Packet dropping test case is excluded from these results.

Figure 14 below shows the HW Variant A's from Single UE, multi UE and traffic mix test cases. Percent describes the difference between the expected and actual counter value. HW variant A's packet dropping test case is not illustrated in this chart, since there is not packet dropping in this test case, so packet dropping counter was zero in all the test runs. DL and UL with running number describes the counter maximum DL cell PDCP SDU bit-rate and maximum UL cell PDCP SDU bit-rate and running number tells which test run the percent is illustrating. Like it can be seen from the Figure 14, single UE and multi UE cases were estimated very accurately. The biggest gap was -0.26 % in single UE cases and the most accurate result was 0.11 %. Single UE results had negative difference on downlink and positive deviation in uplink side. In multi UE, the biggest difference was 0.32 %, and the smallest difference was 0.19 %. All the results were slightly bigger than the expected values. In traffic mix test cases, the accuracy was very volatile. From total of 10 test run (DL & UL together), 4 stayed within the 5 percent range. The biggest difference was 12.80 % and the smallest -1.88 %. Uplink and downlink failed equally, since there was 3 unacceptable over 5 percent differences in both DL and UL counters. All the result values were distinctly smaller than expected values.

Figure 14 illustrates, how HW variant A's single UE and multi UE test cases were calculated very accurately, and therefore in those test cases counter verification can be activated. In traffic mix test case, results varied a lot, and 6 test runs did not stay within the 5 percent limit that traffic mix case has. The reasons for volatility of the results are unknown, and they should be further investigated, and necessary actions should be made before traffic mix cases can be taken as part of automated test runs.




| | DL1 | DL2 | DL3 | DL4 | DL5 | UL1 | UL2 | UL3 | UL4 | UL5 |
|---|---|---|---|---|---|---|---|---|---|---|
| Single UE | -0,16% | -0,16% | -0,16% | -0,16% | -0,26% | 0,11% | 0,11% | 0,11% | 0,11% | 0,11% |
| Multi UE | 0,20% | 0,20% | 0,20% | 0,20% | 0,20% | 3,91% | 3,93% | 3,89% | 3,91% | 3,86% |
| Traffic Mix | -7,73% | -7,70% | -7,67% | -7,70% | -7,69% | -26,01% | -26,80% | -27,83% | -25,62% | -25,34% |

**Figure 15.** HW variant B's test case results from counters maximum DL cell PDCP SDU bit-rate & maximum UL cell PDCP SDU bit-rate. Packet dropping test case is excluded from these results.

Figure 15 shows HW variant B's results from Single UE, Multi UE, and traffic mix cases. Percent shows the difference between the result and expected value. Packet dropping test case is not illustrated in this chart. Like figure 15 shows, DL PDCP SDU drop rate is not presented in this chart, since there is not PDCP SDU dropping in test case, and all the results from that counter were zero, as expected. DL represents the maximum DL cell PDCP SDU bit-rate and UL stands for maximum UL cell PDCP SDU bit-rate. Running number (e.g. UL1) tells the number of the test run, and is the result for DL or UL counter. HW variant B's single UE calculations were estimated very accurately. Allowed difference between the result and expected value was 2% in single-UE test cases. The highest difference percent in HW variant B's single-UE tests was 0.26 %, and the smallest 0.11%. Like with HW variant A, all the downlink counters were smaller than expected, and uplink counters greater than expected. Estimation accuracy can be explained with the simplicity of the single UE test case. There is only one UE as part of the test case, and there is no other than data subscribers in the test case. Multi UE tests allow 5 percent range in results. The biggest difference in multi UE cases was 3,91 %, which is still allowed result for multi UE test case. On downlink side differences were small, all test runs had only 0.2 % difference. In traffic mix test cases result varied remarkably, and in UL3 difference was -27,83 %. Even the smallest difference was -25.34 % on UL. Both uplink and downlink results were negative on all the test runs.

Figure 15 illustrates, how accurate the estimations were for HW variant B's single UE, multi UE and to traffic mix test cases. Similarly, like with HW variant A, single UE and multi UE cases were all within the acceptable limits. Thus, in those test cases counter verification can be activated as a part of those test cases. In multi UE test case, UL cell PDCP SDU bit-rate counter should be followed for some time, since it had almost 4 % difference at worst. In HW variant B's traffic mix test case, counter verification cannot be enabled, since all the runs had failing results. Reasons for failures should be taken into further investigation.

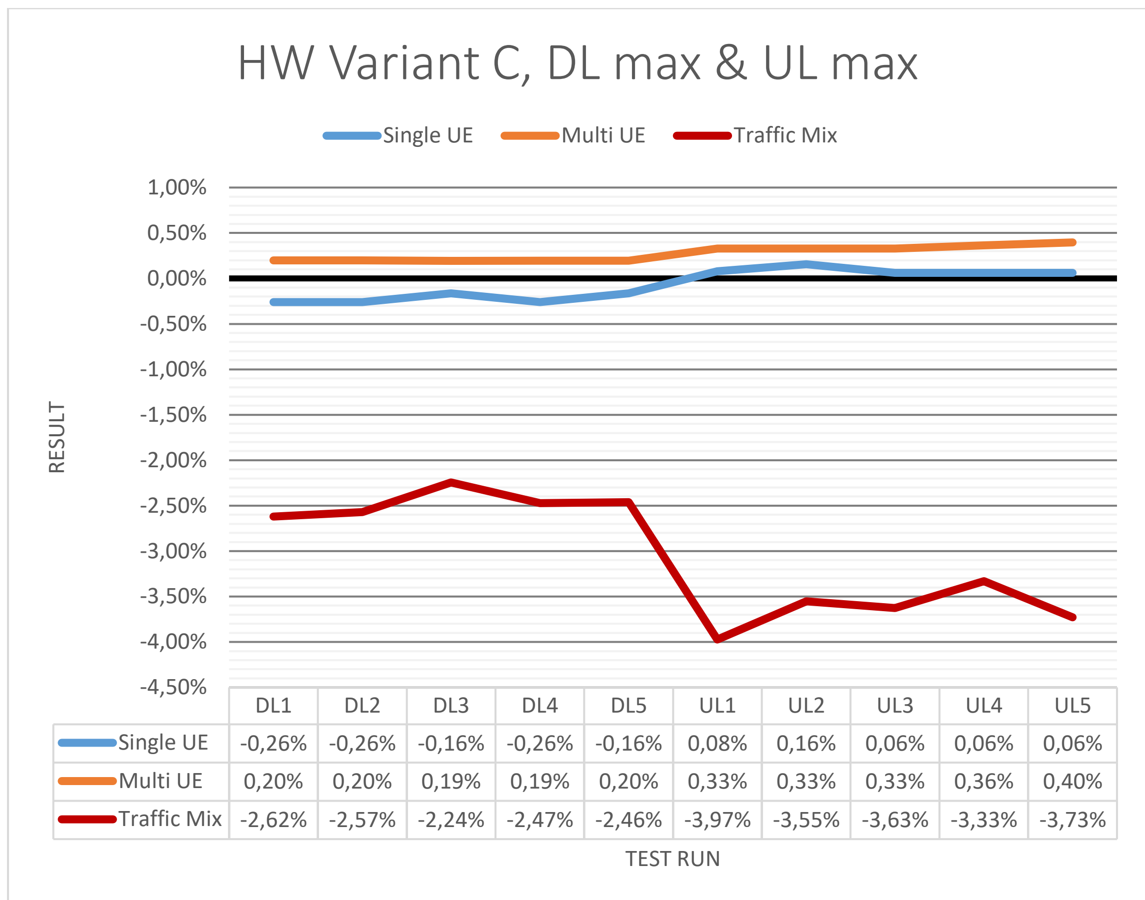
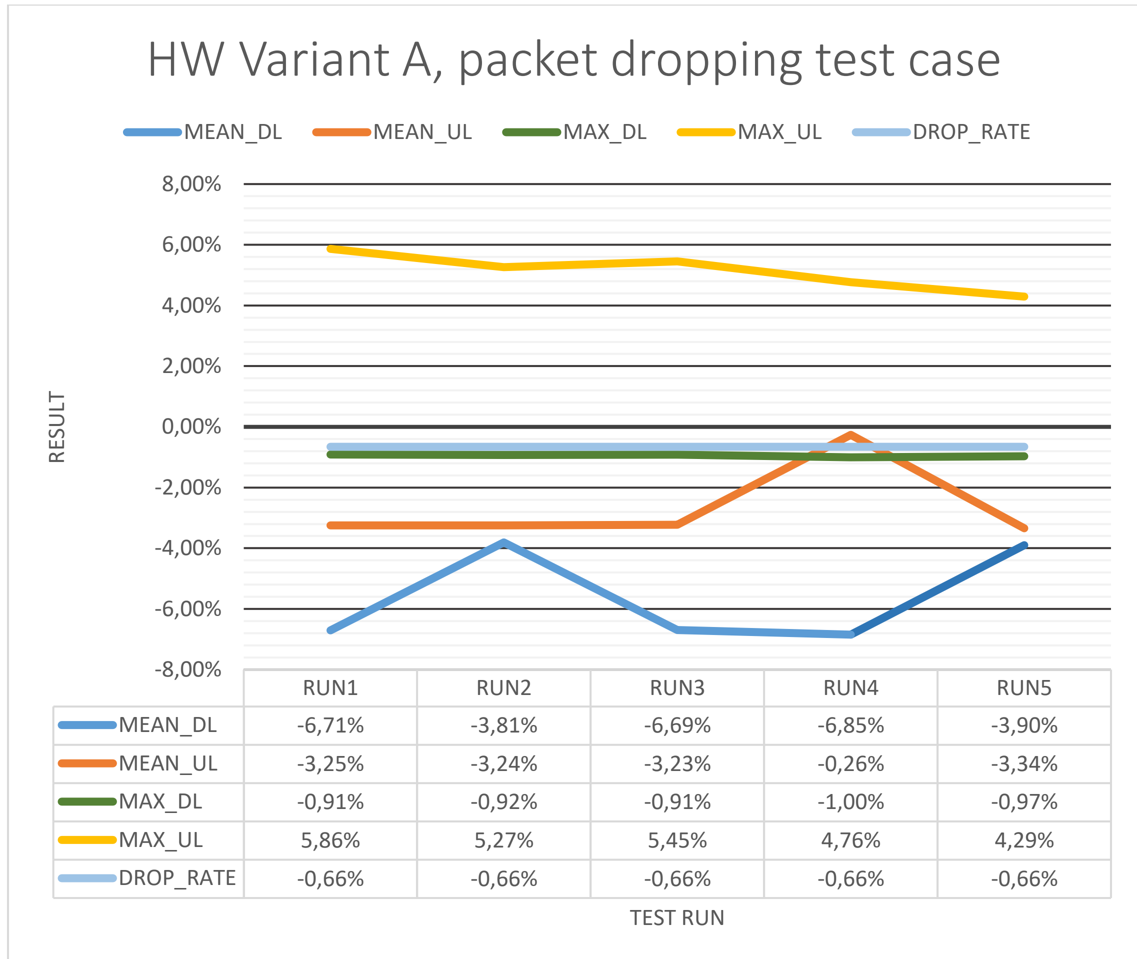



**Figure 16.** HW variant C's test case results from counters maximum DL cell PDCP SDU bit-rate & maximum UL cell PDCP SDU bit-rate. Packet dropping test case is excluded from these results.

Figure 16 illustrates HW variant C single UE, multi UE and traffic mix test cases. Percent shows the difference between the expected and gained value. Figure 16 considers only maximum DL and UL PDCP SDU bit-rate counters and drop-rate counter is not illustrated in the figure 16, since there is not packet dropping in the case, and counter showed correctly value zero on all the runs. Like it can be seen from figure 16, with all different test types (single UE, multi UE & traffic mix), result stayed within the acceptable 2 (single UE) and 5 (multi UE & traffic mix) percent range. Single UE and multi UE were again well estimated. The biggest difference was -0.26 % with single UE, and the smallest difference was only 0.08 %. On downlink, all results were negative and on uplink all were positive. The biggest difference with multi UE test cases was 0.4 % and the smallest was 0.2 % In all the multi UE test runs results were positive. On HW variant C, all the counter stayed within the 5 percent limit in all the test runs. This did not happen with HW variants A and B. The biggest difference in traffic mix cases was -3.97 %, and the smallest -2.24 %. The results in all traffic mix test case test runs were negative. Traffic mix results were very stable compared to other HW variants' traffic mix test cases.

Figure 16 illustrates how HW variant C's single UE, multi UE and traffic mix test cases limit values were estimated. HW variant C was exceptional compared to HW variants A and B, since in HW variant C all the test runs in all test cases were within the range. For HW variant C, counter verification can be taken into use in all the test cases. Traffic mix case had a small variation, so it should be followed for some time in nightly test runs.

### 5.1.2 Packet dropping test cases

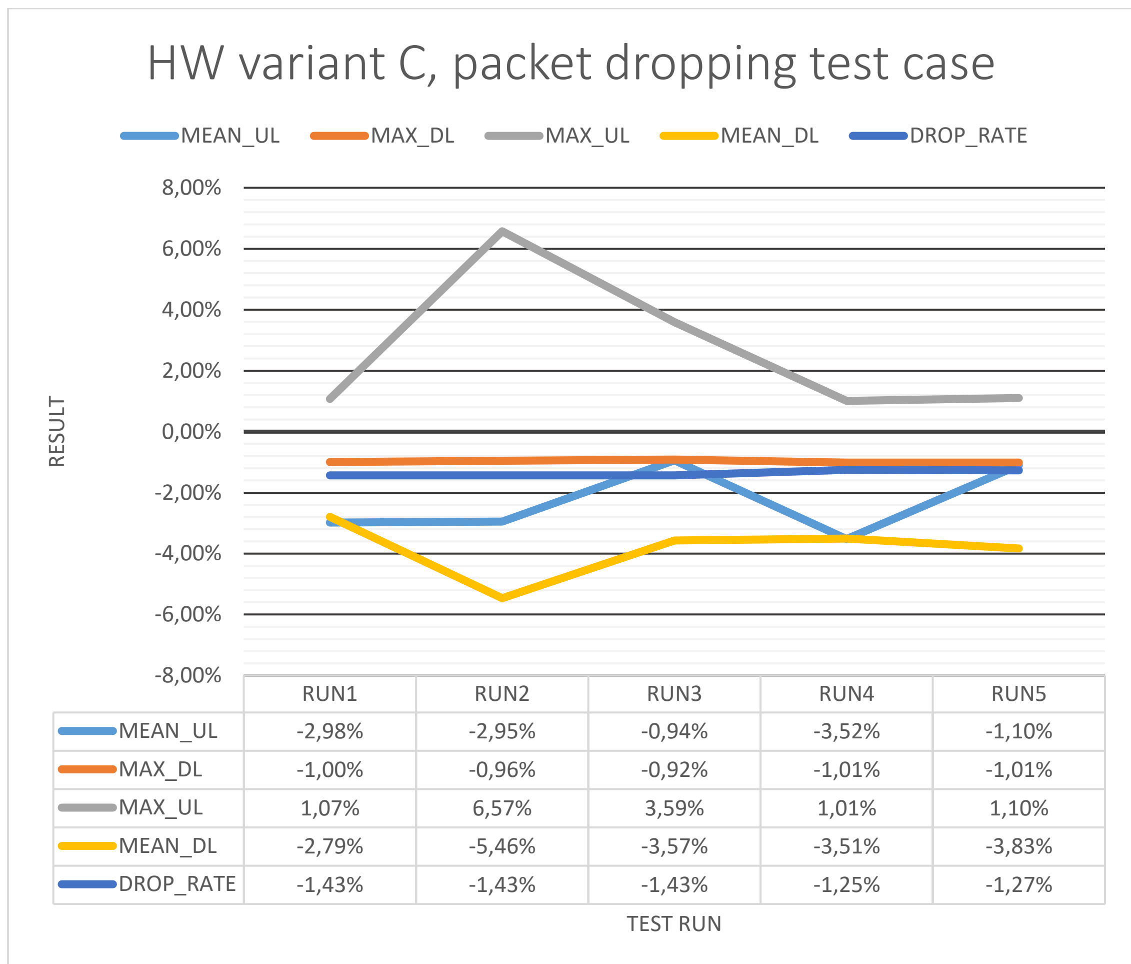



| | RUN1 | RUN2 | RUN3 | RUN4 | RUN5 |
|---|---|---|---|---|---|
| MEAN_DL | -6,71% | -3,81% | -6,69% | -6,85% | -3,90% |
| MEAN_UL | -3,25% | -3,24% | -3,23% | -0,26% | -3,34% |
| MAX_DL | -0,91% | -0,92% | -0,91% | -1,00% | -0,97% |
| MAX_UL | 5,86% | 5,27% | 5,45% | 4,76% | 4,29% |
| DROP_RATE | -0,66% | -0,66% | -0,66% | -0,66% | -0,66% |

**Figure 17.** HW variant A's packet dropping test case.

Figure 17 presents the HW variant A's test case, which is multi UE test case, where part of the packets is dropped during the test case. Test case was run five times to gather material for validation. RUN with a running number tells which test run is in question. In figure 17, MEAN_DL and MEAN_UL represent average DL and UL cell PDCP SDU bit-rate, MAX_DL and MAX_UL represent maximum DL and UL cell PDCP SDU bit-rate and DROP_RATE illustrates DL PDCP SDU drop rate counter. Overall value estimations were very accurate in this test case, when the biggest difference between calculated and actual value was -6.85 %, with counter MEAN_DL. MEAN_DL was three times out of the bounds, and two times within the limits. MAX_UL happened to be three times out the bounds, with the highest difference of 5.86 %. MEAN_DL and MAX_UL were the only counters which had values that did not stay within the limits. The rest of the counters did stay within the limits with all the test runs. DROP_RATE stayed closest to the ideal zero value, and all the runs had only -0.66 % difference to zero value. MAX_DL counter's biggest difference to zero was -1.00 %. MEAN_UL stayed also within the limits, with having a little bit of variance between the test runs, and had -3.34 % difference at the worst.

Like figure 17 represents, in HW variant A's packet dropping test case counters MEAN_DL and MAX_UL did not stay within the 5 percent range. Thus, those counters cannot be taken as part of this test case's automatic counter verification. Reasons for these failing counters are still unclear and they should be investigated. The rest of the counters worked properly, so they can be activated in this test case.




| | RUN1 | RUN2 | RUN3 | RUN4 | RUN5 |
|---|---|---|---|---|---|
| MEAN_UL | -2,98% | -2,95% | -0,94% | -3,52% | -1,10% |
| MAX_DL | -1,00% | -0,96% | -0,92% | -1,01% | -1,01% |
| MAX_UL | 1,07% | 6,57% | 3,59% | 1,01% | 1,10% |
| MEAN_DL | -2,79% | -5,46% | -3,57% | -3,51% | -3,83% |
| DROP_RATE | -1,43% | -1,43% | -1,43% | -1,25% | -1,27% |

**Figure 18.** HW variant C's packet dropping test case.

Figure 18 presents the HW variant C's test case, which is multi UE test case, where part of the packets is dropped during the test case. Test case was run five times to gather material for verification. RUN with running number tells which test run is in question. In figure 18, MEAN_DL and MEAN_UL represent average DL and UL cell PDCP SDU bit-rate, MAX_DL and MAX_UL represent maximum DL and UL cell PDCP SDU bit-rate and DROP_RATE illustrates DL PDCP SDU drop rate. Like with HW Variant A's packet drop test case (figure 17), in this case also two counters had values that were out of the range. MEAN_DL had value difference of -5.46 % and MAX_UL had 6.57 % difference from expected value. The rest of the counters stayed within the limits on all runs. MAX_DL was the most accurate counter, having 1.01 % difference at its worst. DROP_RATE was also very accurate, having 1,43 % difference at its worst and -1.25 % difference at its best. MEAN_UL was the third counter that stayed within the acceptable limits. It had some variation on results between the test runs, having difference range from -3.52 % to -0.94%. MAX_UL was the only counter which had positive different % in all runs, when others had negative values during the tests.

Figure 18 illustrates HW variant C's packet dropping test case and the functionality of the counters that were automatically verified with it. Counters MEAN_DL and MAX_UL did stay within the 5 percent range on all runs, so they cannot be taken as part of the automatic verification. Reasons for these failures should be taken under further investigations. Other counters worked properly in this test case. Thus, they can be taken as part of automatic counter verification in this test case.

### 5.1.3 Summary of the results

Table 6 gathers data from all the test runs, from the perspective of the counters. Mean DL and Mean UL represent average DL and UL cell PDCP SDU bit-rate, Max DL and Max UL represent maximum DL and UL cell PDCP SDU bit-rate and Drop Rate illustrates DL PDCP SDU drop rate. Max DL, Max UL and Drop Rate were the only counters that were verified in all the test runs. Mean DL and Mean UL were only tested in packet dropping test case, since it had the needed over 10 second duration that average counters need to work properly. Obviously, it would have been possible to run average counters in all 55 runs, despite that fact they did not work like they should have, but when the duration issue was noticed, it was decided that this research should consider only the counters that have fair opportunity to behave correctly. That makes the sample of the average counters smaller than expected. With leaving the average counters out from the short test runs, average counters have now valid data from packet dropping test cases, and the complete results are comparable to other counters, too.

In table 6, value over range (VOR) illustrates the number of times that value was not between the limits. Max UL had the greatest number of VOR, when 12 test cases failed on Max UL counters. Value over range % tells the percent of VOR from all the test runs. In other words, it shows how many percent of test runs failed to stay in the 2 or 5 percent range, where percent value depends from type of the test case. Mean UL had also the highest VOR %, and even 30 % of the test runs had problems with Mean UL counters.

Diff average in table 6 describes the average difference between the expected and result value. All the counters had average, that meets the 5 percent range, and all counters had negative diff average. That means on average result values were smaller than expected.

**Table 6.** Total counter statistics with different measures.

| | Max DL | Max UL | Mean DL | Mean UL | Drop Rate |
|---|---|---|---|---|---|
| Number of test runs | 55 | 55 | 10 | 10 | 55 |
| Value over range | 8 | 12 | 1 | 3 | 0 |
| Value over range % | 14.55 % | 21.82 % | 10.00 % | 30.00 % | 0 % |
| Diff average | -1.70 % | -2.05 % | -3.25 % | -3.95 % | -0.18 % |
| Diff min | -0.16 % | 0.06 % | -0.26 % | -0.94 % | 0.00 % |
| Diff max | -12.80 % | -27.83 % | -5.46 % | -6.85 % | -1.43 % |
| Diff median | 0.26 % | 1.10 % | 3.30 % | 3.66 % | 0.00 % |

Diff min in table 6 describes the smallest difference that was achieved between the expected and result value. All the counters had under one percent difference at their best. Naturally, drop rate counter from other than packet dropping test cases reached that zero value, because packet discarding was not part of those test cases and drop rate stayed at zero. Max UL was very close to reach the expected value (0 %), by having only a 0.06 % difference at its best.

In table 6, diff max shows the biggest difference that a counter had between result and expected value. Max DL and Max UL had the biggest differences on values, where Max DL had 12.80 % and Max UL -27.83 % difference to the expected value.

Diff median in table 6 shows the median difference that counter had to expected value. Drop rate had median of 0.00 %, which can be explained with the amount of test cases where packet dropping did not happen. All the medians were inside 5 % range, and mean UL had the biggest diff median (3.66 %).

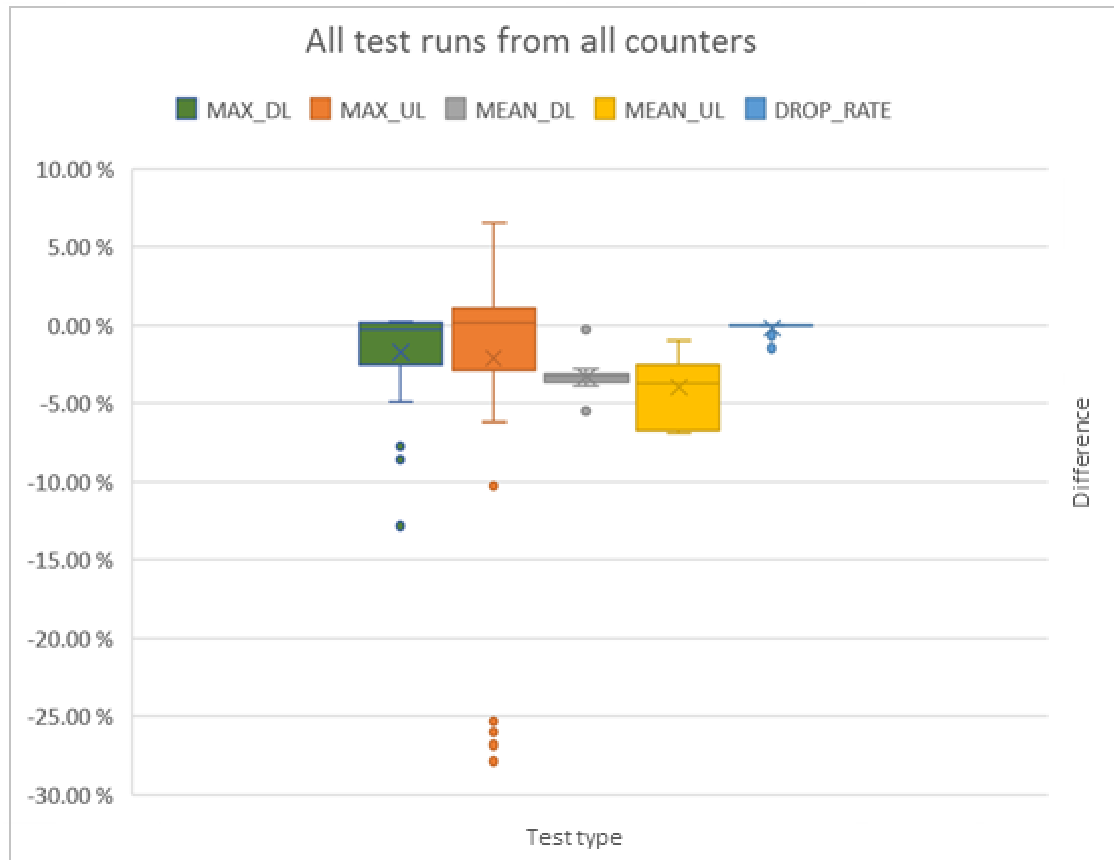



**Figure 19.** Box plot of all the data that was gathered on test runs.

Figure 19 shows the same information than figure 18 earlier, but it is illustrated as box-plot chart. From this illustration, it can be seen, how supremely accurate drop rate counter's estimations were, compared to other counters. Of course, this can be partly explained with the fact that only two the of test cases included PDCP SDU discarding, but from the test cases where PDCP SDU discarding was included, drop rate counter worked enormously well. Drop rate was the only counter that had all the result values within the 5 percent range. Mean counters were activated only in 10 test cases, since they did not work in other than longer discarding test cases, so the sample is smaller than with other counters. Mean DL has small deviation, with a few exceptions. Other counters than drop rate had bigger deviations, and especially MAX_UL counter had a dramatically wide range in its results, varying from 6.57 % to -27.83 %, which makes the counter's total range to 34.4 %. Max DL has the second most variation during test runs, -12.99 % in total. Obviously bigger variations with max counters can be partly explained with bigger sample, since max counters were part of 55 different test runs.

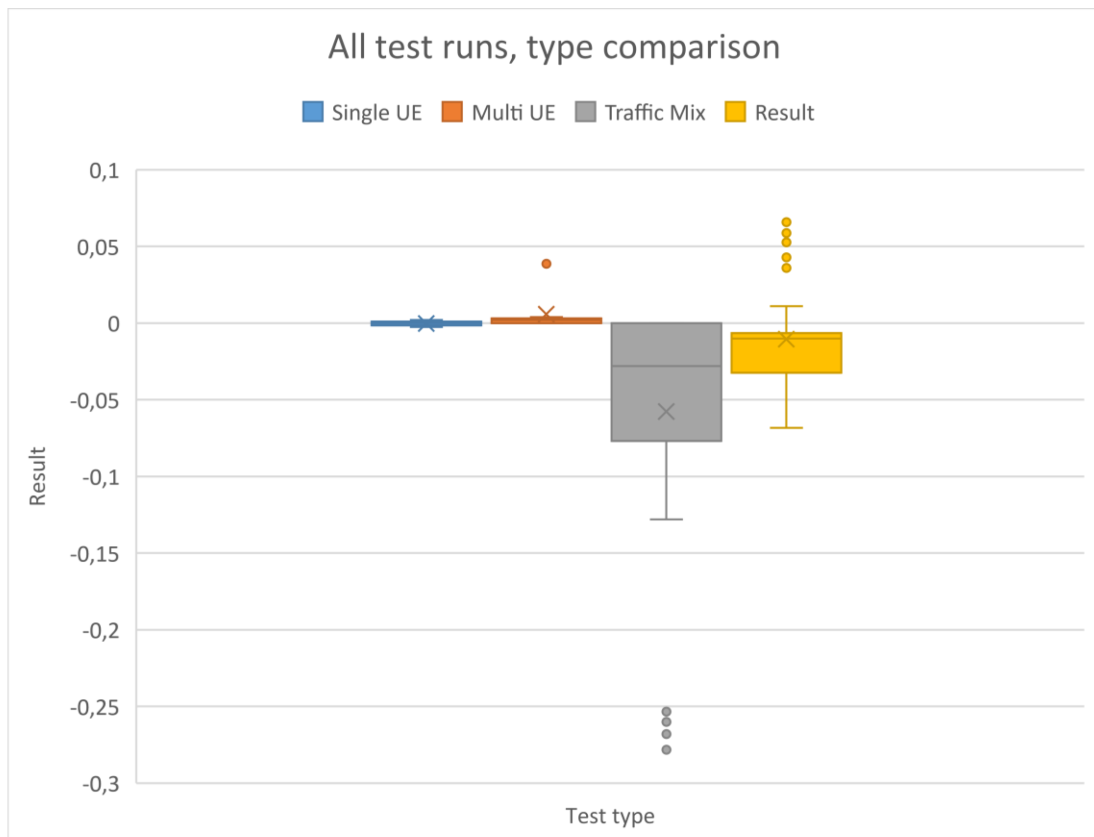



**Figure 20.** Box plot illustration of test type comparison, from all the data that was gathered from test runs.

Figure 20 illustrates the data from the perspective of test type. “Single UE” consists from all the HW variants (A, B & C) single UE test cases, “multi UE” from all HW variants’ multi UE test cases and “traffic mix” from traffic mix test cases of all the HW variants. “Packet drop” describes the packet dropping test cases from both the HW variants that had packet dropping test cases (HW variants A & C). From figure 20, it can be noticed that only Single UE and multi UE test cases stayed within the 5 percent limit on all runs. Traffic mix and packet drop test cases did not stay within the range on all runs. From figure 20 it can be noticed that single UE and multi UE are the only test types, where counter verification works reliably. Traffic mix and packet mix cases should be taken into further investigations, and it should be researched, what causes errors in those two test types.

**Table 7.** Data from all the test runs from the test type point of view.

| | Single UE | Multi UE | Traffic mix | Packet drop |
|---|---|---|---|---|
| Total test runs | 45 | 45 | 45 | 50 |
| Failure rate | 0 % | 0 % | 31,11 % | 14 % |
| Num. of > 5 % | 0 | 0 | 0 | 4 |
| Num. of < -5 % | 0 | 0 | 14 | 3 |
| Average diff | -0.03 % | 0.57 % | -5.77 % | -1.05 % |
| Min diff | 0.00 % | 0.00 % | 0.00 % | -0.66 % |
| Max diff | -0.26 % | 3.93 % | -27.83 % | -6.85 % |

Single UE, multi UE and traffic mix cases had all 45 test runs each, since they all had 3 test cases where they were run (HW variants A, B & C) and 3 counters each (maximum

DL and UL cell PDCP SDU bit-rate and DL PDCP SDU drop rate) and they were all run 5 times (3*3*5 = 45). Packet drop test cases were run in two HW variants (A & C), and each run had 5 counters, and was run 5 times (2*5*5 = 50). In table 7, "total runs" describes the amount of test cases run on each type of test.

Like it can be seen from table 7, single UE was the most successful test type in the means of limit value calculations. Single UE test cases with all HW variants had a very little variation between the expected and result values. Average difference between the result and expected value was just -0.03 %. Single UE test cases had a very small variation in the difference between result and expected value. The smallest difference was 0.00 % and the biggest -0.26 %, and the range of difference was as small as 0.46 %. Range of difference is calculated by summing up the absolute values of the greatest and the smallest value. With single UE test cases, all the runs stayed within the 5 percent range, which means there was no failed test runs in single UE test cases.

Multi UE test type was the second most successful from all the test types. Average difference between the expected and actual result value was 0.57 %, so estimations were pretty accurate with multi UE test cases. The smallest difference was 0.00 %, and the biggest was 3.93 %, and the range of difference was 3.93 %. It is also notable that in multi UE test cases, all the results were greater than or equal to zero. Similar trend was not seen in other test types. Multi UE test cases did not have any results that were over the 5 percent range, so there was no failed test runs in multi UE test cases.

Traffic mix test cases were the least successful compared to other test types. Average difference between result and expected value was -5.77 %, which is out of the acceptable range. The biggest difference between the result and achieved value was as great as -27.83 %. The smallest difference was 0.00 %, and range of difference was 27.83 %. In traffic mix cases, all the results were less than equal to zero. 14 out of 45 test cases had difference that was smaller than -5 %, which means that 14 test cases failed. Failure rate with traffic mix test cases was 31,11 %. From all the test types, traffic mix cases had the most out of range failures and the biggest difference range on results.

Packet drop was the third successful from all the test types. Its average between the actual and expected result value was -1.05 %. The biggest difference was -6.85 %, and the smallest -0.66 %, and the range of difference was 13,42 %. The difference between expected and result value was greater than 5 % on 4 test runs, and less than -5 % on 3 runs, which means that there were 7 failures on packet drop test cases, out of the 50 test runs. Failure rate with packet drop test cases was 14 %.

## 5.2 Comparison against the user stories

The implemented system was evaluated not only by the accordance of the limit value calculations, but also against the system requirements, which were presented as user stories. User stories were presented in chapter 4.5. In this chapter, each of the user stories are evaluated. Each user story was evaluated against its DoD (Definition of Done). Colour green represents *done* (D) user story, yellow *partly done* (P) user story and red *not done* (N) user story. At the end of the chapter, user story evaluation is summarized.

**ID:** US-1 (D)
**User Story:** As a developer, I want to enable counter verification in any test case in capacity tests, so that I can verify counter functionality in capacity tests.

**DoD**: Counter verification can be enabled in any capacity test case.
**Evaluation:** This research included selected test cases, from three most important HW variants, and four different types of tests (single UE, multi UE, traffic mix and PDCP SDU discarding). Counter verification system functioned properly in all these test cases. Although there is 500+ test cases in capacity tests, all test cases are triggered similarly, so as a conclusion it can be said that counter verification system works in any test case on capacity tests.

**ID:** US-2 (D)
**User Story:** As a developer, I want that counter verifier system is written in Python, so that I can get quicker feedback from capacity tests (since Python is faster than Robot Framework). .
**DoD:** System is implemented with Python.
**Evaluation:** System was implemented with Python. There was a couple of lines of Robot Framework code in implementation. Robot Framework was used to trigger the counter verifier.

**ID:** US-3 (D)
**User Story:** As a developer, I want that counter verifier system gathers counter group automatically, so that I do not need to know the counter group when I activate counter verification**.**
**DoD:** Counter data is available without knowing the counter group.
**Evaluation:** System gathers counter group data automatically based on counter name. Developer does not need to know anything about the counter group, she can just describe the desired counter names in the test case description (see chapter 4.6).

**ID:** US-4 (D)
**User Story:** As a developer, I want that value comparison is made with range values, so that I can set more precise limits to expected values.
**DoD:** Value comparison is done with range (e.g. min limit <= result <= max limit).
**Evaluation:** Value comparison is done with range values. It is mandatory to use min and max limits, or percentual range and limit value, which calculates the min and max limits automatically. Value comparison cannot be done if the min or max value is missing, they both are required.

**ID:** US-5 (D)
**User Story:** As developer, I want that expected value can be set as a value with percentual margin, so that I can set changeable percentual limits to counter verification.
**DoD:** Min and max limits can be set as percent from expected value (e.g. min limit = expected value – expected value * - range %).
**Evaluation:** Percentual limits can be set for each counter. Percent needs to be positive integer or float value, and between 0 and 100. Min and max range are calculated by multiplying range value with percentage (e.g. if percentage is 5, min limit = range value * 0.95 and max limit = range value * 1.05)

**ID:** US-6 (D)
**User Story:** As a developer, I want that expected value can be set as minimum and maximum value, so that I can set range for counter verification.
**DoD:** Min and max limits can be set as two integer values.
**Evaluation:** Minimum and maximum can be set as range values. Min and max can be integer or float values. Example of min and max value declaration can be found from chapter 4.6.

**ID:** US-7 (D)
**User Story:** As a developer, I want that test case is failed, when value is not between expected limits, so that I know when counter verification fails.
**DoD:** When result is not between limits, test case is failed.
**Evaluation:** Test case fails, if value is not in the range and "fail test case if not in range" is set in test case parameters. Example of setting "fail test case if not in range" to test case description, can be found from chapter 4.6.

**ID:** US-8 (D)
**User Story:** As a developer, I want to have options when expected counter value is not met, so that I have better debugging abilities in such a situation.
**DoD:** If it is declared in test case description, it is possible to not meet the limit values, and not fail the test case.
**Evaluation:** If "console print" or "console print if not in range" is set in test case parameters, test case is not failed when value is not in limits, and counter value is printed to console (only when counter has out of range value, when "console print if not in range" is selected).

**ID:** US-9 (D)
**User Story:** As a developer, I want that counter printing is possible to do for each cell value, so that I have better debugging capability.
**DoD:** Counter value is possible to be printed for each cell.
**Evaluation:** Each cell's counter values are printed to console, if "print cell values" is set in test case parameters, so counter value is possible to be printed for each cell.

**ID:** US-10 (D)
**User Story:** As a developer, I want that system is extendable for counters that need extra calculations, so that I can verify all kinds of counters.
**DoD:** Custom counters can be verified in system.
**Evaluation:** DL PDCP SDU discarding rate is custom counter, and it was verified as part of this research, so the system can be extended with "custom counter", which is not directly calculated in application.

**ID:** US-11 (D)
**User Story:** As a developer, I want that value can be verified from specific pool(s) or cell(s), so that I have better options on verification.
**DoD:** Counter values can be verified from specific pool(s) or cell(s).
**Evaluation:** Desired pool(s) or cell(s) can be determined in test case description (see example in chapter 4.6). Verification is possible to make for specific pool(s) or cell(s).

**ID:** US-12 (P)
**User Story:** As a developer, I want that counter verification system is replicable to another than capacity tests, so that I can perform counter verification in other types of tests.
**DoD:** Counter verification system can be adapted to other than just capacity tests.
**Evaluation:** Within the limits of this research, there was no time for test counter verification system in other than capacity tests. However, the system was designed in such a way that it should be easy to adapt to other than capacity tests, too. System should be usable in other than capacity tests, but since there was no time to verify that, this user story cannot be evaluated as "done".

**Table 8.** User story evaluation results.

| User story evaluation results | | | | | | | |
|---|---|---|---|---|---|---|---|
| US-1 | US-2 | US-3 | US-4 | US-5 | US-6 | US-7 | US-8 |
| US-9 | US-10 | US-11 | US-12 | | | | |

Like table 8 illustrates, majority of the requirements were met. Only one out of twelve user stories was "partially done", and none of the user stories were evaluated as "not done". Eleven user stories were "done", which means that they fulfilled the condition that was defined in DoD.

# 6. Discussion and implications

The goal of this research was to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. To achieve this goal, design science research methodology was applied for this research. This methodology was applied by creating an artifact (counter verification system), and by evaluating it rigorously. The implemented counter verification system was evaluated based on the accuracy of the limit value calculations, and by evaluating the user stories, that formed the requirements for the system. Results of the evaluation were desirable, and the research questions that were declared at the chapter *2. Research problem and method* can be answered with the help of the results. Research questions are answered and discussed below.

Research question 1: *How a dynamic, easily configurable counter verification system can be implemented to capacity tests?*

Counter verification system can be implemented to capacity tests, by following the steps of this research. At first, literature review is performed, which includes the product in question (LTE), 3GPP throughput counters and the key components of the case company's capacity testing environment. When the literature review is done, and basic knowledge about the LTE, counters and test environment is gained, suitable counters and test cases are chosen. There are several hundred of 3GPP counters in PDCP/RLC/MAC layer and over 500 test cases in case company's capacity tests. Not all of them are necessary to cover at once, but smaller subset is enough to verify the functionality of the system. After test case and counter subsets were selected, System Component Testing (SCT), which is an operational environment for capacity tests, is taken into further examination, which is followed by study of the counter verification process in SCT. When fundamentals of testing environment are understood, design can be started. In system's design phase, requirements for the system are written down. Then system can be implemented. After the system is ready, a way to calculate limit values for each counter needs to be created. Finally, system can be tested against the requirements that were done in design phase, and the accuracy of limit value calculations can be estimated.

Research question 2: *What factors affect to limit value calculation accuracy in simulated test environment?*

Like it was seen in chapter *5. Evaluation,* all the test runs did not stay within the acceptable (2 or 5 percent) limits. Especially traffic mix test cases were problematic, since 31,11 % of the traffic mix cases failed to stay within the limits. Packet drop test cases failed in 14 % of the test runs. Single UE and multi UE test cases were within the limits in all the test runs.

Single UE test cases are test cases, where only one UE transfers data. Multi UE test cases have multiple users, and test case consists only from data transmission. Traffic mix cases have multiple different kinds of users, some transferring data, and others making VoLTE calls. Packet drop test cases are multi UE test cases, where part of the PDCP SDUs are dropped during transfer. It was noticed that test case complexity has negative impact to success of counter verification, and to limit value calculations.

All HW variants were successful with single UE and multi UE test cases. HW variant C failed only in packet dropping test case, and was only HW variant that had no failures in traffic mix test case. Other HW variants failed in traffic mix and packet dropping test cases. Thus, it can be deduced that HW variant has effect to the accuracy of limit value calculations. From the HW variants that were part of this research, HW variant C seems to be the most suitable for counter verification.

Counter average DL PDCP SDU bit-rate was supposed to be part of every test case, which were selected to the scope this research. But like explained in chapter *4.7. Limit value calculations*, it was not possible to verify average counters in short (< 10 second) test cases. Majority of test cases in this case company's capacity tests are 5 second test cases, so this was an important finding. Thus, duration of the test case is a crucial factor for average counters' limit value calculation.

Like said in chapter *4.7. Limit value calculations,* calculating limit values was not an easy process. There are hundreds of different 3GPP counters in PDCP/RLC/MAC layer. Every counter collects different data, so calculations for each counter needs to be done separately. Of course, once the formula for limit value calculation is created and verified for a counter, it needs to be documented, to ease to process in future. Complexity of limit value calculations means that extremely high expertise of SCT and from the application are required. Thus, one thing that affects to limit value calculation accuracy, is the knowledge of the system and application.

Several things affected to the accuracy of limit value calculations in simulated test environment. Short test case duration had a restricting impact for average counters. If test case was too short, average counters did not work properly, and limit value calculations did not provide comparable results. In longer test cases, average counters worked like expected. Complexity of the test case was other factor that affected dramatically to limit value calculation accuracy. Number of test case failures increased dramatically, when complexity of the test case increased. Traffic mix and packet drop test cases had all the failures that occurred in test runs. HW variant was one of the factors that influenced to the accuracy of limit value calculations. HW variant C had the smallest number of failed test runs. Last thing that affected to the accuracy of limit value calculations, was the knowledge of the system. Limit value calculations turned out to be surprisingly complicated to execute. Accurate limit value calculations require deep understanding from both the SCT and the application.

## 6.1 Evaluation against the design science research guidelines

**Guideline 1: Design as an Artifact**
In this research, the produced artifact was to create automatic counter verification system for LTE PDCP/RLC/MAC layer's capacity tests. The system was verified in total of 185 test runs, and system was capable to verify counter functionality in new and exciting ways. Construct, models and methods used in the building of the artifact, were described in a detail.

**Guideline 2: Problem Relevance**
The research problem was relevant, since there was a need for counter verification system in case company's PDCP/RLC/MAC layer's capacity tests. This need was answered by building this artifact.

**Guideline 3: Design Evaluation**

Evaluation of the system was done rigorously based on case company's requirements, and on the accuracy of the limit value calculations.

**Guideline 4: Research Contributions**
Contribution of this research, was the implemented system itself and all the benefits it brings for the case company. The automatic counter verification system was able to solve the problems that were not yet solved, and benefit from the existing knowledge by creating something new out of it.

**Guideline 5: Research Rigor**
This research was conducted by following the guidelines Hevner et al's (2004) design science research framework. Rigorous methods were followed on evaluation and construction of the automatic counter verification system. The system was described with the mathematical formalism.

**Guideline 6: Design as a Search process**
Case company had a problem, since they did not have effective solution for counter verification in capacity tests. The best design for the system was developed as the combination of the knowledge in existing literature and from the fundamentals of SCT.

**Guideline 7: Communication of Research**
Artifact of this research is described in such a detail that it provides beneficial knowledge to the technology-oriented people of case company. This research also has practical information that can benefit management-oriented audiences.

# 7. Conclusions

The objective of this research was to create a mechanism, which makes it possible to easily enable automatic counter verification in any automated capacity test case. This counter verification system was piloted in large Finnish telecommunications company. The system was created by applying the design science research guidelines. The system was evaluated against the user stories, that were created on design phase, and based on the accuracy of limit value calculations. Implementation of the system went smoothly as expected. Limit value calculations were problematic, since the average counters were not verifiable in majority of the test cases. Also, the calculations turned out to be extremely complicated, and they required a broad expertise in both LTE and SCT. Counter verification system provided some interesting results, that require further investigations in case company. In total, 185 test case – counter combinations were verified, and in almost 13 % of them, counter verification failed the test case. Especially more complicated traffic mix and packet drop test cases were problematic for counter verification, and they should be examined carefully, to find a root cause for these problems.

## 7.1 Limitations

Complexity of the both product in question and SCT, made an insecure feeling to limit value calculations. This research was able to gather data of counter verification from different capacity test cases, but it was left undefined, why such failures occur in different test cases and HW variants. Although this research found the test cases, which had malfunctioning 3GPP counters, in the scope of this research it was not possible to figure out what are the root causes for the problems. There might be problems in SCT or application that causes the failures, and further analysis is needed to find the reasons for failures. It is also possible that limit value calculations are not done correctly for complicated test cases.

Average counters were not possible to be verified in short 5 second test cases. 9 out of 11 test cases that were part of this research, were 5 second test cases. That reduced the average counters' number of test runs from 55 to 10. Sample material for average counters was smaller than expected, and more test runs are needed to verify average counters' functionality.

Verification of limit value calculations was slow. To verify limit values, tests needed to be run on real eNodeB, and sometimes there was queues for test PCs, which meant that over half an hour was needed for running one test case. That made a limit value calculation a time-consuming process, and feedback from calculations was not as quick as wished.

There was no similar or same kind of research in existing literature, which limited this research. It would have been beneficial to see, how same kind of systems have been built by the other researchers. Now everything was made based on the researcher's earlier experiences and to the solutions in existing SCT environment. Peer review to other similar researches could have enriched the system that was implemented.

## 7.2 Future work

This research provoked a various improvement ideas. There was a couple of problems, that could not be solved in the scope of this research. Only single UE and multi UE test cases worked reliably, when traffic mix and packet drop test cases led to multiple failures. Although these findings were important, the root causes for failures were not solved. It is possible that counters do not work correctly in application, or there are some issues with SCT environment. The most important thing was that these findings were made, so that case company can focus on them in future and make the necessary corrections. This issue should be taken under further investigation.

This system was designed in a way, that it should be easy to take use in other PDCP/RLC/MAC layer’s tests than capacity tests. Unfortunately, in the scope of this research, there was no time to test automatic counter verification system in other than capacity tests. The system that was implemented as a part of this research, would most likely be beneficial in other tests as well, and it should be taken into use more widely in SCT.

Like stated earlier, limit value calculation was a time consuming and complicated process. It was slow to verify the values for each test case and HW variant. One of the most important knowledge that this research provided, was the limit value calculation formulas for each of the counters, which were part of this research. In future, it would be convenient to have a tool, which would help with limit value calculations. Even better would be an extension to counter verification system, which calculates limit values automatically from test case parameters. Then it should be only necessary to know, what counters are going to be verified, and limit value calculation could be totally skipped, since the system could calculate them automatically.

Average counters should be tested more with the longer test cases. The sample size for average counters in this research was too small, to make assumptions of their functionality. Perhaps it could be considered, if those 5 second test cases could be extended to 10 second test cases, to enable counter verification in those test cases. That would open a lot of new test cases, where average counters could be tested. Other solution would be to fix SCT to support average counters in shorter test cases.

# References

3GPP, About 3GPP. Retrieved January 12, 2017, from http://www.3gpp.org/about-3gpp/about-3gpp

3GPP TS 32.425 version 13.5.0 release 13 (2016) *telecommunication management; performance management (PM); performance measurements, evolved universal terrestrial radio access network, (E-UTRAN).*

3GPP TS 32.450 version 13.0.0 release 13 (2016) *LTE evolved universal terrestrial radio access (E-UTRA) key performance indicators (KPI) for evolved universal terrestrial radio access network, (E-UTRAN): Definitions*

3GPP TS 36.101 version 13.1.0 release 13 (2015) *Evolved universal terrestrial radio access (E-UTRA); User Equipment (UE) radio access capabilities (Release 13), user equipment (UE) radio transmission and reception*

Agile manifesto. Retrieved February 16, 2017, from http://agilemanifesto.org/

Beizer, B. (1990). Software testing techniques, van nostrand reinhold. *Inc, New York NY, 2nd Edition. ISBN 0-442-20672-0,*

Bianchi, G. (1998). IEEE 802.11-saturation throughput analysis. *IEEE Communications Letters, 2*(12), 318-320.

*BS ISO/IEC 26515:2011: Systems and software engineering. developing user documentation in an agile environment* (2012). British Standards Institute. Retrieved March 3, 2017, from https://bsol.bsigroup.com/en/Bsol-Item-Detail-Page/?pid=000000000030236632

Cao, J., Li, L. E., Bu, T., & Sanders, S. W. (2012). *System and method for root cause analysis of mobile network performance problems* Google Patents.

Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021. Retrieved February 26, 2017, from http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

Code like a pythonista: Idiomatic python. Retrieved April 4, 2017, from http://python.net/~goodger/projects/pycon/2007/idiomatic/handout.html.

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Dahlman, E., Parkvall, S., & Skold, J. (2013). *4G: LTE/LTE-advanced for mobile broadband* Academic press.

Dahlman, E., Skold, J., & Parkvall, S. (2011). *4G: LTE/LTE-advanced for mobile broadband*. Amsterdam: Academic Press.

Duvall, P. M. (2007). *Continuous integration* Pearson Education India.

Fewster, M., & Graham, D. (1999). *Software test automation* Addison-Wesley Professional.

Goldwasser, M. H., & Letscher, D. (2008). Using python to teach object-oriented programming in cs1. *Innovation and Technology in Computer Science Education. June*, 30-32.

Gordejuela-Sanchez, F., & Zhang, J. (2009). LTE access network planning and optimization: A service-oriented and technology-specific perspective. *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE,* 1-5.

Graham, D., & Fewster, M. (2012). *Experiences of test automation: Case studies of software test automation* Addison-Wesley Professional.

H. X. Nguyen, & B. Northcote. (2016). User spectral efficiency: Combining spectral efficiency with user experience. *2016 IEEE International Conference on Communications (ICC),* 1-6. doi:10.1109/ICC.2016.7511204

Heracleous, L., & Papachroni, A. (2012). Strategic leadership and innovation at apple inc. *Case Study. Coventry: Warwick Business School,*

Hoikkanen, A. (2007). Economics of 3G long-term evolution: The business case for the mobile operator. *Wireless and Optical Communications Networks, 2007. WOCN'07. IFIP International Conference On,* 1-5.

Holma, H., & Toskala, A. (2011). LTE for UMTS: Evolution to LTE-advanced: Second edition. *LTE for UMTS: Evolution to LTE-advanced: Second edition* () doi:10.1002/9781119992943

Humble, J., & Farley, D. (2010). *Continuous delivery: Reliable software releases through build, test, and deployment automation (adobe reader)* Pearson Education.

*IEEE standard glossary of software engineering terminology* (1990). doi:10.1109/IEEESTD.1990.101064

J. A. Fernandez-Segovia, S. Luna-Ramirez, M. Toril, & C. Ubeda. (2015). Computationally-efficient estimation of throughput indicators in heterogeneous LTE networks. *2015 IEEE 81st Vehicular Technology Conference (VTC Spring),* 1-5. doi:10.1109/VTCSpring.2015.7145756

L. Pierucci. (2015). The quality of experience perspective toward 5G technology. *IEEE Wireless Communications, 22*(4), 10-16. doi:10.1109/MWC.2015.7224722

L. Tratt, & R. Wuyts. (2007). Guest editors' introduction: Dynamically typed languages. *IEEE Software, 24*(5), 28-30. doi:10.1109/MS.2007.140

Lai, S., & Leu, F. (2015). Applying continuous integration for reducing web applications development risks. Paper presented at the 386-391. doi:10.1109/BWCCA.2015.54

Larman, C., & Vodde, B. (2010). Acceptance test-driven development with robot framework. *Robot Framework,*

Martin Fowler, Test Driven Development. Retrieved April 12, 2017, from https://martinfowler.com/bliki/TestDrivenDevelopment.html

Martin Fowler, Continuous Integration. Retrieved February 2, 2017, from https://www.martinfowler.com/articles/continuousIntegration.html

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

McGrath, M. (2014). *Python*. Leamington Spa, Warwickshire, U.K.: In Easy Steps.

Mishra, A. R. (2007). *Advanced cellular network planning and optimisation: 2G/2.5 G/3G... evolution to 4G* John Wiley & Sons.

Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* John Wiley & Sons.

Nakazawa, M., Okamoto, S., Omiya, T., Kasai, K., & Yoshida, M. (2010). 256-QAM (64 gb/s) coherent optical transmission over 160 km with an optical bandwidth of 5.4 GHz. *IEEE Photonics Technology Letters, 22*(3), 185-187.

Nunamaker Jr, J. F., Chen, M., & Purdin, T. D. (1990). Systems development in information systems research. *Journal of Management Information Systems, 7*(3), 89-106.

Nygard, M. (2007). *Release it!: Design and deploy production-ready software* Pragmatic Bookshelf.

Ostrand, T. (2002). White-Box testing. *Encyclopedia of Software Engineering,*

P. Brooks, & B. Hestnes. (2010). User measures of quality of experience: Why being objective and quantitative is important. *IEEE Network, 24*(2), 8-13. doi:10.1109/MNET.2010.5430138

Pajunen, T., Takala, T., & Katara, M. (2011). Model-based testing with a general purpose keyword-driven test automation framework. *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference On,* 242-251.

Python. Retrieved January 27, 2017, from https://www.python.org/about/

Python, data structures. Retrieved March 27, 2017, from https://docs.python.org/2/tutorial/datastructures.html#dictionaries

Robot framework. Retrieved January 7, 2017, from http://robotframework.org/

S. Jovalekic, & B. Rist. (2008). Test automation of distributed embedded systems based on test object structure information. *2008 IEEE 25th Convention of Electrical and Electronics Engineers in Israel,* 343-347. doi:10.1109/EEEI.2008.4736543

Seth, N., & Khare, R. (2015). ACI (automated continuous integration) using Jenkins: Key for successful embedded software development. Paper presented at the 1-6. doi:10.1109/RAECS.2015.7453279

Software and systems engineering software testing part 1: Concepts and definitions (2013). doi:10.1109/IEEESTD.2013.6588537

Stresnjak, S., & Hocenski, Z. (2011). Usage of Robot Framework in automation of functional test regression. *ICSEA 2011 the Sixth International Conference on Software Engineering Advances,*

V. Buenestado, J. M. Ruiz-Avils, M. Toril, S. Luna-Ramrez, & A. Mendo. (2014). Analysis of throughput performance statistics for benchmarking LTE networks. *IEEE Communications Letters, 18*(9), 1607-1610. doi:10.1109/LCOMM.2014.2337876

Van Rossum, G., Drake, F. L., & Kuchling, A. (1999). *Python tutorial*. Lincoln, Neb.: Open Docs Library.

Wireless network data traffic: Worldwide trends and forecasts 2016-2021. *Ananylys Mason*