CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

5G User Plane Load Simulator

Master's thesis in Computer science and engineering

OLOF DÜSTERDIECK
TASDIKUL HUDA

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

5G User Plane Load Simulator

OLOF DÜSTERDIECK
TASDIKUL HUDA

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

5G User Plane Load Simulator
OLOF DÜSTERDIECK
TASDIKUL HUDA

Supervisor: Romaric Duvignau, Department of Computer Science and Engineering
Advisor: Patrik Nyman, Ericsson
Examiner: Marina Papatriantafilou, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

5G User Plane Load Simulator
OLOF DÜSTERDIECK
TASDIKUL HUDA
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As the commercial introduction of 5G networks is getting closer, the deployment of testing technology able to perform at the requirements specified for 5G, is of utmost importance. As a company on the leading edge of 5G deployment, Ericsson manages mobile traffic by developing a gateway known as the **Evolved Packet Gateway (EPG)**.

EPG is the main component responsible for bridging the gap for data packets between 5G base stations and servers on the Internet. Following the 5G design, EPG is split into a **Control Plane (CP)** and a **User Plane (UP)**. The purpose of this thesis is to implement a 5G network simulator that is able to load test a physical 5G User Plane which is used to route traffic in a mobile network.

The simulator is able to simulate traffic in both the uplink and downlink direction as well as sending traffic on several different user sessions. To simulate traffic, three different traffic models, **steady-rate**, **step-wise** rate and **Poisson processes**, are used. In each simulation, the traffic is stateless and follows a client-to-server architecture. All parts included in this architecture is simulated, except for the UP.

To analyze the results of a test, we define several different performance metrics such as **throughput** and **latency**. These are evaluated using evaluation techniques such as **Control Charts**. Several sets of experiments are performed in which we verify the implemented load types and measure the maximum rate of both the simulator and the UP.

Using these results, it is possible to analyze the scalability of the UP and our simulator. We conclude that our simulator performs above the required rate to load test the UP for most scenarios with differing number of users and flows per each user. Where this is not the case, ideas and suggestions on how the simulator can be enhanced additionally are given. We also conclude that the UP scales very well with an increased number of users and that it performs above the proposed 5G requirements.

Keywords: 5G, EPG, User Plane, load testing, mobile networking, network simulator, traffic modeling.

Acknowledgements

First of all, we would like to express our gratitude to Romaric Duvignau (supervisor at Chalmers) for his guidance throughout this project from an academic point of view. Next, we would like to thank Patrik Nyman (supervisor at Ericsson), for the time and effort he invested in us and in the project to make sure it always progressed towards the right direction. Finally, we would like to thank Eric Nordström (technical advisor from Ericsson) for helping us through the technical parts of the thesis and Marina Papatriantafilou (examiner at Chalmers), for providing feedback and helping us enhancing the quality of this thesis.

Olof Düsterdieck, Gothenburg, October 19, 2020
Tasdikul Huda, Gothenburg, October 19, 2020

Contents

1

Introduction

During the past few years, technology has played a major role in providing sustainable solutions to everyday life. As society becomes more and more dependent on services provided by modern technology, ever more enterprises are gaining increased interest in investigating new technological areas. Significant examples of areas that have had a growing potential in recent years are autonomous driving and the **Internet of Things (IoT)**. Some requirements for these technologies to flourish are reliable network communication with low latency and high throughput. These requirements are expected to be met by the introduction of the 5G mobile network.

As 5G is approaching commercial roll-out, there is a growing need for test environments in this area. The purpose of this thesis is to address this need by implementing a 5G network simulator that can load test specific network components. Examples of a situation where the simulator will provide an additional value is when there is a need to investigate the behaviour of a 5G network with specific requirements, without actually having to set up the network physically.

1.1 Background

A company that has shown a great initiative in driving the potentials of 5G forward is Ericsson. One of the main areas that Ericsson focuses on is the management of mobile traffic. This is done by maintaining a robust gateway known as the **Evolved Packet Gateway (EPG)**. EPG is placed on a network called **Evolved Packet Core (EPC)** and it is used to route 3G and 4G data [11].

When transitioning to 5G, EPG will be split into a **Control Plane (CP)** and **User Plane (UP)**. The main purpose of CP is to handle functions associated with session management such as the allocation of IP addresses. The CP is further composed of different components called **AMF** and **SMF**. However, this thesis will focus on the UP and therefore, AMF and SMF will not be explained in more detail but rather only referred to as a combined CP. UP, on the other hand, is needed to inspect packets and route them towards the correct destination [10].

An overview of the 5G network environment is illustrated in Figure 1.1. It can be seen that the network environment follows a *client-to-server architecture*. Clients,

which in this context are denoted **User Equipment (UE)** communicate with servers on the Internet, through the UP. These servers are located on a network referred to as **Packet Data Network (PDN)**. During any client-to-server communication, data can travel in two different directions. The direction of the data sent from UE to PDN is referred to as **uplink** and correspondingly, the direction when a server sends data to UE is called **downlink**.

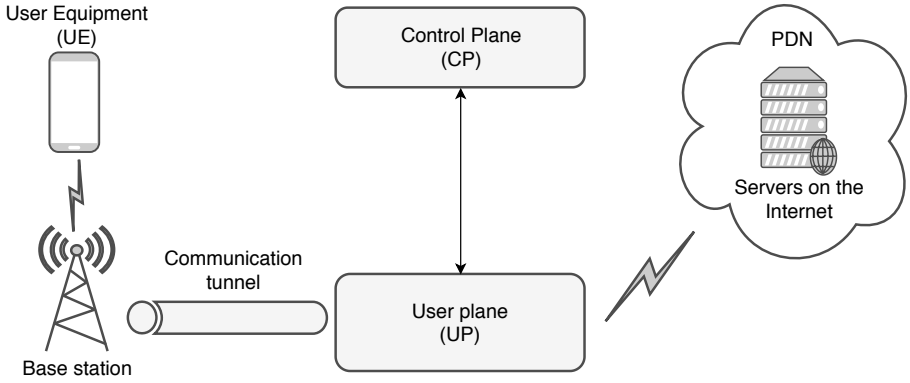

Figure 1.1: Simplified illustration of the 5G network.

1.2 Aim

The goal of this thesis is to implement and analyze a network simulator capable of load testing the UP part of the 5G mobile infrastructure. In the context of this thesis, the only part of the 5G network that is physical is the UP. The remaining part of the network such as UE, PDN and CP, are created by our network simulator when it starts to run. This setup is shown in Figure 1.2.

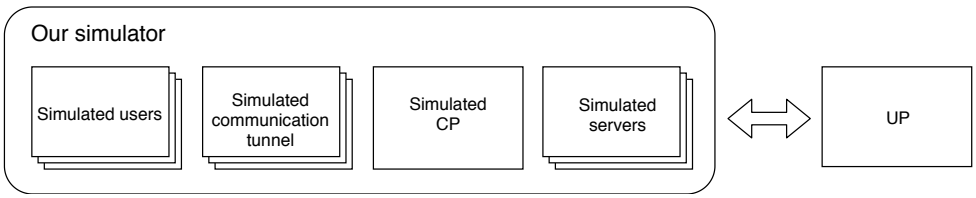

Figure 1.2: Simulated components and a physical UP.

The setup of the environment is described in more detail in §3.3.1 and once this setup is done, it is possible to simulate 5G traffic in both the uplink- and the downlink direction. Regardless of which direction traffic is generated in, it will always go through the UP in order to be routed to the correct destination.

Simulating traffic can be performed with different settings and the goal of each simulation is to be able to load test and analyze the performance of the UP. The two most important performance characteristics are **latency** and **throughput**. Latency describes the time it takes for a packet to traverse a system and throughput describes the number of packets per second or bits per second that can be processed

by the system. Rigorous definitions of these characteristics are given in §2.1. Furthermore, a detailed description of different approaches to run simulations and test performance is discussed in §2.3.

Research questions

Using the implemented simulator, the following research aspects are going to be investigated throughout this project:

1. **Performance analysis:** investigate basic performance metrics of a 5G simulator, for instance, investigate accurate approaches to determine the throughput and latency of different setups and compare them.

2. **Saturation:** determine the maximum throughput at which the UP and the simulator can send traffic.

3. **Scalability:** how much does an increased number of UE and connections per UE impact the performance of UP?

4. **Comparison:** is it possible to notice a clear improvement in the 5G setup compared to previous 4G results regarding chosen performance metrics?

Required properties

The features that will be implemented in the simulator are listed below:

1. **Low overhead**: the simulator needs to maintain a low simulation overhead that is able to cover the 5G requirements.

2. **Smooth deployment**: the simulator should follow a cloud-native approach so it becomes easy to deploy it on different operating systems.

3. **Bidirectional communication**: it should be possible to generate 5G communication in both the uplink- and the downlink direction.

4. **Multiple user sessions**: an arbitrary number of UE should be able to establish communication through the UP to an arbitrary number of PDN.

5. **Simple configuration**: it should be easy to set session parameters such as IP addresses for network components, rate limits of packets and different load types for the traffic.

6. **Practical simulation-types**: implementation of different traffic models to analyze how the system reacts when the communication behaviour changes at run-time.

1.3 Scope

In §1.2, the goal of the simulator was defined as performance testing of the UP in a 5G environment. So far, this goal has been described at a high level. In this section, this definition will be put into the scope of our thesis.

Firstly, focusing on performance testing means that the evaluation of the test will only take the UP's behavior during load into account and not if the load is processed correctly. More specifically, no verification will be done by the simulator on whether the UP follows the protocols used during load correctly. Instead, only performance metrics will be analyzed.

Secondly, 5G is referred to as the environment in which the simulator will be used. 5G as a concept is used to describe a wide range of areas related to telecommunication. However, in the scope of this thesis, the 5G environment considers these following parts:

- A 5G system-level architecture.
- Performance expected of the next generation.

A system-level architecture in 5G requires using protocols such as **PFCP** and component separation as suggested by the **CUPS-architecture**. A more complete description of this architecture is given in §2.2.3.

One of the main goals of 5G is increased performance. As a testing tool for a 5G application, our simulator needs to meet the requirements expected from the next generation of telecommunication networks. These requirements are presented in 3.1.

1.4 Outline

Chapter 2 is divided into three main parts. First, §2.1 provides explanations of necessary definitions and nomenclature needed to understand the rest of this thesis. Second, §2.2 gives an explanation of the current mobile network architecture followed by a description of how the network architecture is reconstructed when transitioning to 5G. Moreover, a description of the network environment in which the simulator load tests the UP-component is explained. Third, §2.3 provides a detailed discussion of load testing, load designing and traffic modelling. Thereafter, the implementation of the simulator and its features are presented in Chapter 4. Chapter 5 presents our main ideas on how to analyze the performance of the simulator as well as how different simulation scenarios are designed and evaluated. Next, chapter 6 elaborates on the most important results achieved in this thesis followed by Chapter 7 and Chapter 8 which provide a discussion of the results and a conclusion of the entire thesis.

2

Background

This chapter is divided into three main parts. The first part emphasizes important definitions and nomenclatures. The second part of this chapter explains how mobile networking works today followed by how mobile networking will be in 5G. The third part elaborates on a discussion of current research regarding load testing and traffic modeling. Furthermore, necessary concepts related to load testing and traffic modeling are rigorously defined by the end of this chapter.

2.1 Preliminaries

This section presents how different terms related to computer networks are defined in our context.

2.1.1 Definitions

Uplink

The direction of the communication when traffic is generated by UE and sent to PDN is referred to as **uplink**.

Downlink

When traffic is generated by PDN and sent to UE, the direction of the communication is said to be in **downlink**.

Throughput

In the context of this thesis, there are two different measurements of throughput. The first one measures how fast our simulator is able to send traffic to the UP. From now on, this will be referred to as the **simulator throughput** or the **incoming rate**. The second one measures how fast the UP can send out traffic that is passing through it and this will be denoted as the **UP-throughput** or the **outgoing rate**.

Both the simulator- and the UP-throughput are measured in either packets per second (pkt/s) or bits per seconds (bit/s). The former might, in some cases, be preferable and easier to interpret whereas the latter provides a more accurate approach when handling packets of differing sizes.

Latency

Usually, latency is defined as the time elapsed from when a packet is sent from a source device until a response is received by that same device (RTT-latency). However, this thesis is only concerned with testing the UP and does not handle matching requests with the corresponding response. Therefore, latency in this context is defined as the difference in time from when the UP receives a packet, processes it, and sends it out. From now on, this definition will be referred to as **UP-latency**.

UP-latency can be split into two versions, UP-latency in the uplink (UP-latency$_{up}$) and downlink direction (UP-latency$_{down}$), and these two versions could potentially be different values. UP-latency$_{up}$ together with UP-latency$_{down}$ is a lower bound of the RTT latency.

Packet loss

Packet loss refers to the event of a packet being dropped or becoming malformed while traversing through the network. Packet loss is measured in the number of occurrences of such events divided by the total number of packets sent. A packet is deemed lost if it does not arrive within a specified timeout window of a simulation.

Sessions

A connection C, between UE and PDN, is a 4-tuple that is described by:

$$C = \langle \text{IP}_{\text{Source}}, \text{IP}_{\text{Destination}}, \text{Port}_{\text{Source}}, \text{Port}_{\text{Destination}} \rangle$$

All connections that share the same $\text{IP}_{\text{Source}}$ belongs to the same session. This implies that the total number of sessions established during a simulation is equal to the total number of unique $\text{IP}_{\text{Source}}$ for the UE devices. From now on, a session will be referred to as a **user session**.

Flows

All existing connections during a simulation are called **flows** and any user session can have an arbitrary number of flows associated with it. For each existing flow, it is possible for a packet to travel in both the uplink- and downlink direction with a specific rate.

Overhead

Overhead is defined as the speed at which the simulator runs and it is measured as the time between any action performed by the UP until the simulator is able to react to that action. The main reason why low overhead is of utmost importance is to be able to simulate a system that fulfills the strict requirements of 5G. To achieve this, the simulator will be developed by utilizing low-level network APIs.

2.1.2 Cloud-native architecture

The term **Cloud native** has grown in conjunction with the rise of cloud computing. It refers to the possibility of running an application on any system regardless of the **Operative System (OS)** or architecture underneath. This is achieved by providing a standardized API to the application so that it does not have to rely on OS-specific features. The **Cloud Native Computing Foundation (CNCF)**, which is an open source software foundation dedicated to the growth and adoption of cloud-native computing [5]) defines the term as follows:

> *"Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds."* [4]

This is done using something called **containers** along with other technologies. These containers can be deployed on a range of architectures and OSs and can thereafter be configured to run applications. In this way, containers are very close to **Virtual Machines (VMs)**. VMs do also allow an application to run inside their environment, regardless of what OS the VMs themselves are running on. The biggest difference between the two approaches is that a container tries to be as thin as possible on top of the underlying OS.

While there are different container systems on the market e.g., **Linux Containers (LXC)**, the most well-known implementation is probably **Docker**. Even though Docker is a for-profit company, most of its software is free and provided under the Apache 2.0 license [8][7] together with an End User License Agreement [9]. Besides offering containerization, Docker also provides support for some cloud management features.

One of the main components of a Docker instance is an **image**, a concept that is also present in VMs. An image can be defined as a snapshot of the state of the system, at any moment in time. It can contain files, executables and other statics on the system.

It is possible to configure a Docker instance either by direct command-line instructions or by using a **Dockerfile**. The Dockerfile is a document in which settings for a Docker-image is declared. These settings can, for instance, include behaviour such as which port should be open or if directories from the host system should be shared with the image.

2.1.3 Configuration files

To simulate 5G communication, one requirement is to configure the network components with certain parameters. An example is the IP addresses of devices that might need to be changed between different test runs. However, these addresses need to be set at the beginning of a simulation. Thus, it can become tedious to search through the entire source code and modify parameters before each simulation. A more efficient approach is to have a configuration file where one can easily

change values of necessary parameters. These chosen values from the configuration file are read and mapped to the corresponding device or network component when a simulation starts.

The configuration format that has been chosen for this project is **YAML**. The abbreviation stands for "YAML Ain't Markup Language" which suggests that it differs from other formats for configuration such as XML. The official YAML specification [37] describes the language as:

> *"YAML™ (rhymes with "camel") is a human-friendly, cross language, Unicode based data serialization language designed around the common native data types of agile programming languages."*

Compared to XML, YAML focuses on data specification and as such omits most of the markup used in XML. This is done to keep the data as human-readable as possible. A simple YAML file can be seen in Listing 1. On the first two lines, simple declarations are shown. The definition on line 3 is a sequence which will map onto a list in most programming languages. Lastly, starting on line 4 is a map and within this structure, each subsequent definition will be mapped as a key and value pair of a map. However, in C, this structure is often made into a struct.

The YAML file illustrated in Listing 1 would in our program parse into the C data structures seen in Listing 2.

```
1 load_rate: 1000
2 payload_length: 32
3 pfcp_addresses:
4     up: 5.5.5.5
5     cp: 6.6.6.6
6 vlan_config:
7     sgw_u: 200
8     pfcp:  201
9     sgi:   202
```

Listing 1: Example of a very basic YAML file.

```
1  uint32_t load_rate;
2  uint32_t payload_length;
3
4  struct vlan_config {
5   uint16_t sgw_u;
6   uint16_t pfcp;
7   uint16_t sgi;
8  };
9
10 struct pfcp_addresses {
11  ipaddr_t up;
12  ipaddr_t cp;
13 };
```

Listing 2: C data structures corresponding to the YAML file.

2.2 Network architecture in 4G and 5G

The simulator that is implemented during this project is concerned with traffic generation in a 5G network. However, the 5G network will be a transition from the current 3G/4G. Thus, in order to understand the network architecture of 5G and the components deployed in such an environment, it is necessary to first understand the network setup of 3G/4G.

Note that the content of this section will be a simplified version of the systems used in the real world. Only the parts of the mobile network that are relevant to this thesis will be included.

2.2.1 Current network architecture

An overview of the 3G/4G network architecture is illustrated by Figure 2.1. From this figure, it can be seen that the architecture of a 3G/4G network is divided into three parts [26]:

1. **LTE Network** which includes UE and base station(s).
2. **Evolved Packet Core (EPC)** which deploys different network components to manage uplink- and downlink traffic.
3. **Packet Data Network (PDN)** which refers to devices on the Internet with IP connectivity.

A UE with mobile connectivity is associated with a radio station. This station maintains an access network denoted **eNodeB (Evolved Node B)** and its purpose is to connect a UE to an EPC network [26]. EPC is a core network architecture for 3G and/or 4G traffic and the three main components deployed in an EPC Network

are [30]:

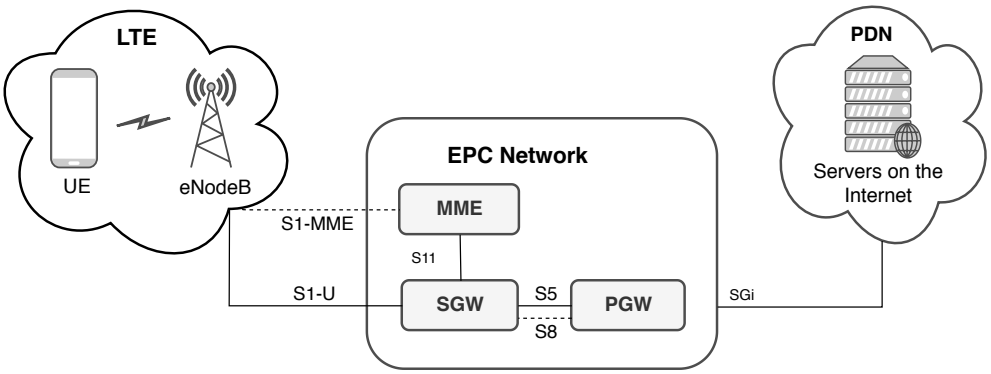

Figure 2.1: A brief illustration of 3G/4G network.

- **Mobility Management Entity (MME)**: responsible for managing UE, for example, authenticating and choosing suitable PGW and SGW (described below) for a certain UE. Such information is obtained by eNodeB.
- **Serving Gateway (SGW)**: either receives packets from UE through a base station (eNodeB) or from PGW. If the incoming packet is sent by a UE, it will be forwarded to an appropriate PGW. If the received packet, on the other hand, is sent by a PGW, it will be handed off to a UE.
- **Packet Data Network Gateway (PGW)**: awaits packets from SGW. Once a packet is received, IP-connectivity is provided so they can be forwarded to PDN. PGW can also receive packets from PDN and hand it off to an SGW.

As seen in Figure 2.1, there are several interfaces used to connect the different components and enable communication between them. A short explanation of these interfaces is given below:

- **S11** is the interface that connects MME to SGW. This interface provides several important functions such as creation and deletion of connections to PDN, downlink data acknowledgments and path management, i.e., updating UE's IP addresses in case it changes during a communication session.
- **S1-U** is the interface used to manage communication e.g., forwarding payload in the uplink- and the downlink direction between SGW and the base station. S1-U is based on the specification of **GTP-U**. GTP is a protocol used to transfer payload packets from the LTE network to the EPC network. GTP-U is an extension of the GTP protocol in order to make it possible to transfer packets through a tunnel to a UP.
- **S1-MME** is used to connect UE to EPC networks. Dotted lines represent signaling messages and unlike packets with payload, **signaling messages** only include command codes to coordinate the communication between eNodeB and MME.

- The **S5** interface is used to transfer payload data between SGW and PGW. The implementation of this protocol is based on GTP which makes it possible to relocate SGW. This can be necessary since UE is not assumed to have the same position during mobility.
- **S8** interface is also based on GTP and it is used to connect SGW to PGW and send signaling messages between them.
- **SGi** is the interface by which communication between PGW and PDN is done.

2.2.2 Evolved Packet Gateway

The services provided by PGW and SGW are based on **Evolved Packet System (EPS)**. EPS is a network architecture to support mobile communication between UE and PDN. For this purpose, EPS maintains a robust gateway known as **Evolved Packet Gateway (EPG)** which can act as either PGW, SGW or both [11]. An illustration of this is shown in Figure 2.2.

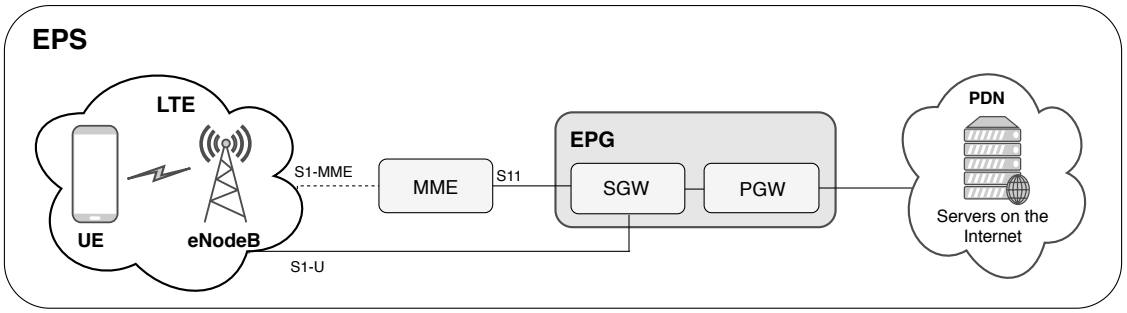

Figure 2.2: Overview of EPG in an EPS network architecture.

Some fundamental features provided by EPG are:

- **Session management**: responsible for managing and maintaining sessions between UE and PDN. In order to manage sessions, EPG deploys pipelines referred to as **EPS-bearers** and they follow the GTP protocol. These bearers are associated with user sessions and keep track of relevant information for IP connectivity such as IP addresses.
- **User packet handling**: when a session is established, EPS-bearers are used to transfer user packets between the components in an EPS.
- **Routing**: the EPG handles routing of all IP packets received from and sent to all connected networks. This mainly consists of finding the optimal routing paths and transferring packets through the network.

In the current network architecture, SGW and PGW are split into two separate parts. These parts are referred to as SGW-U, SGW-C, PGW-U and PGW-C. U and C in the respective part denotes functions of **User Plane (UP)** and **Control Plane (CP)**. The CP is responsible for e.g., session management whereas UP is

deployed to inspect and transfer user packets (payload data). An illustration of this can be seen in Figure 2.3.

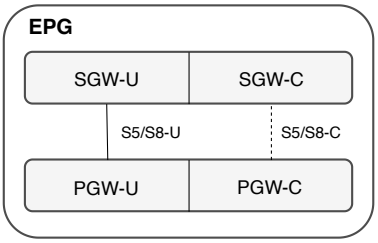

Figure 2.3: SGW and PGW in a 3G/4G network.

2.2.3 CUPS architecture

As 5G is becoming publicly available, a reconstruction of the current EPS architecture will be necessary in order to fulfill the specifications of the new generation of mobile networking. This section will explain how the architecture of the current mobile network will be changed when transitioning to 5G.

The EPG is one of the network components that will be reconstructed in 5G. As explained earlier, EPG consists of two parts, PGW and SGW. Recall that each part is split into a UP and CP. When transitioning to 5G, EPG will still remain in two parts but the UP part from SGW and PGW are going to be merged into one part and likewise, the CPF from SGW and PGW are going to be combined into the second part. An example of this is shown in Figure 2.4.

Figure 2.4: Reconstruction of EPG in a 5G environment.

The concept of combining all functions of UP into one part and all functions of CP into another is referred to as Control- and User Plane Separation, abbreviated as the **CUPS architecture**. An advantage of this architecture is that it makes it possible for CP and UP to scale independently, e.g., reconfigure an existing CP or UP without affecting the functionality of currently existing nodes.

Communication between the components in the CUPS architecture is managed by two interfaces, $\mathbf{Sx_a}$ and $\mathbf{Sx_b}$. $\mathrm{Sx_a}$ is the interface used to connect UP with CP for SGW whereas $\mathrm{Sx_b}$ is used to connect UP and CP for PGW. Both interfaces follow the implementation of the **Packet Forwarding Control Plane (PFCP)** protocol.

PFCP is defined by 3GPP CT4 and its purpose is to reliably deliver UDP messages between the CP and UP. Note that the messages that are sent using PFCP do not

contain any payload data. They are purely controlled messages that are used to manage sessions.

2.3 Load Generation

As this thesis will cover load testing of software, it is important to give an introduction of current research related to this kind of testing. Therefore, this section will start by providing background on load testing. Next, a description of different approaches that can be taken to design load for testing is discussed. Finally, a brief overview at the literature on network traffic modelling is given.

2.3.1 Load testing approaches

The main reason to perform load testing is to test the performance of a system in the presence of increased utilization. This is an crucial part of testing as it has been proven that large scale systems are more frequently prone to failure due to their inability to scale as opposed to bugs in the actual system [35].

In the context of this thesis, **load** is defined as the number of packets processed by a UP during a simulation. A **load test** is the process of monitoring a UP's performance in the presence of a certain load under varying, controlled parameters [16].

Contrary to functional testing, load testing usually has no clear requirements on a pass or fail criteria. In literature, this is referred to as "lacking a test oracle"[16]. Because of this, the criteria for load tests are usually derived from the *no-worse-than-before* principle. This principle states that the performance of a version in testing should be at least as good as the previous versions [16]. In literature, the previous version is usually called the **baseline** and the version in testing is usually called the **target run**.

To compare with the baseline, it would be necessary to model the performance of both in some way. Gao *et al.* [13] proposed a framework for comparing and analyzing different load testing analysis techniques. Eight different performance models were evaluated falling into three different method categories. These categories are denoted as **Queuing Theory-based methods**, **Data Mining-based methods** and **Rule-based methods**. The methods that are based on Queuing Theory and Data Mining involves performance prediction and are therefore outside the scope of this project. Furthermore, two out of the three methods that the authors concluded "most effective" are Rule-based. These two are called **Control Charts (CtrlChart)** and **Descriptive Statistics (DescStats)**.

DescStats is a very basic evaluation method. It simply compares the performance of different test runs using statistics such as the mean or median [12]. One might compare the absolute relative difference between the mean latency between two tests.

When using CtrlCharts, it is first necessary to define a **Control Line (CL)**. The

control line is used to represent characteristics such as the mean- or median value of a certain performance parameter obtained from different load tests. From the control line, it is possible to define an upper and lower control limit (UCL and LCL respectively) by e.g., calculating the standard deviation. The UCL can, for instance, be set to the median value plus a set number of standard deviations while the LCL is the median minus the same number of standard deviations. The idea of defining these threshold values is to be able to analyze and draw a conclusion on whether the model applied to design load is satisfying or not.

The previously mentioned definition of load testing overlaps somewhat with what is generally defined as **performance testing** [18]. The difference is that performance testing usually focuses specifically on monitoring performance metrics such as throughput, latency and resource utilization. Load testing, on the other hand, often addresses metrics such as reliability and robustness.

One normally differentiates between load testing and **stress testing**. Unlike load testing, stress testing is applied to determine the maximum load a system can realistically handle. With this approach, tests are usually run with increasing traffic until the system is saturated. Using stress testing, it is possible to determine maxima of system statistics such as throughput, sessions, flows and rate.

2.3.2 Load designing approaches

When performing load testing, some traffic is generated and sent through the system. There are many different ways of designing load and the decision of what load model to use can differ depending on the objectives of the test. Generally, the two schools of thought around modeling load are designing **realistic load** and **fault including load**. Designing realistic load tries to resemble expected usage whereas fault including loads are most likely to cause problems in the system [18].

This thesis will focus mainly on testing using realistic loads and thereby, only concepts connected to this load designing model will be discussed. In a survey of load testing on large scale systems, Jiang *et al.* found two approaches proposed in the literature for designing realistic goals. These two approaches are **aggregate-workload based load** and **use-case based load** [18].

Aggregate-workload based load design focuses on generating requests at set target rates. A system might, for example, be required to handle a certain kind of traffic at a certain traffic rate. When designing these kinds of loads, two dimensions need to be considered. The rate of the incoming traffic, called **workload intensity** and the ratio of a different kind of traffic referred to as **workload mix**. These two dimensions can either be kept constant during the entire test i.e., *steady load* or vary over time i.e., *step-wise load*. The values to use for workload intensity and workload mix could be inferred from past data or known operational values, for instance, normal expected usage or peak time usage.

However, fixed target rates might not be realistic. To better reflect an actual user, using the system, load that resembles different use cases could be designed. This

is called a **use-case based load design**. These techniques try to model users' behaviour in a variety of ways in order to create load. These user models could be UML Activity Diagrams, Markov chains or other stochastic/probabilistic models.

In the specific case of a network component, a use-case based model might need to create data traffic similar to what would be observed in the real world. Generating realistic traffic using probabilistic models is a large research area and incorporates a lot of concepts independent of load design as a whole. Because of this reason, such models will be discussed in more detail in §2.3.3.

Pseudo or hybrid methods in which a combination of these different strategies are used can also be employed. Varga, Pál, and Olaszi [34] used a step-wise load in which subscribers moved in steps between several different use-case based load designs, each with a separate traffic model. This was done to simulate users moving between activities such as *"using a website"* to *"call someone using Voice-over-IP (VOIP)"*.

2.3.3 Traffic modeling approaches

The literature for traffic modeling is extensive and has evolved ever since the introduction of computer networks. Examples of surveys that have documented the evolution of this field are the surveys by Wilson [36], Hlavacs *et al.* [15] and Karagiannis *et al.* [19]. In this section, the traffic modeling approaches considered most relevant for this thesis will be presented.

Poisson processes

The application of **Poisson processes** is a common approach to model traffic. This depends on the fact that they are easy to both understand and analyze. Moreover, Poisson processes are often close to natural processes and offer some useful analytic properties that are discussed thoroughly later in this section.

Poisson processes are processes in which one observes discrete events at irregular times, over a continuous time interval. In the case of network traffic, these events could be arrival times of packets [23]. These events are statistically independent, meaning that the timing of one event does not depend on the timing of any previous event.

Poisson processes are (as the name implies) closely related to Poisson random variables and the Poisson distribution. A Poisson random variable X is a discrete random variable such that

$$P\{X = k\} = \frac{e^{-\lambda}\lambda^k}{k!}, \quad k = 0, 1, 2, \ldots \tag{2.1}$$

where $\lambda > 0$ is a fixed parameter of the Poisson distribution and corresponds to the average value of X.

In a Poisson process with parameter λ, the number of events occurring within a time interval $[0, t]$ (usually denoted $N(t)$) is distributed following the Poisson distribution with parameter λt. The previous equation 2.1 then instead becomes:

$$P\{N(t) = k\} = \frac{e^{-\lambda t}(\lambda t)^k}{k!}, \quad k = 0, 1, 2, \ldots \tag{2.2}$$

Poisson processes offer two significant analytical properties for traffic modeling and simulation. First of all, it is *memoryless*. This ensures that future behaviour is independent of past behaviour. Each new generation can be done without any preserved state of prior execution. Secondly, the aggregation of several Poisson processes' streams simply generate a new Poisson process with the resulting λ equal to the sum of each individual stream's λ'.

In spite of the advantages offered by Poisson processes, research has shown that pure Poisson processes are only suitable for modeling traffic over short time periods [20][28]. These models tend to deviate from reality when burstiness is taken into account. Burstiness describes the fact that network packets usually arrive in bursts and not with a mean arrival rate as Poisson processes model them.

Markov Modulated Poisson Processes

Markov Modulated Poisson processes (MMPP) combine Poisson processes together with Markov chains to create a model with more realistic burstiness than the simple Poisson processes. The fundamental concept is to have several different Poisson processes where each is represented by the different states in a Markov chain. The common choice is to use a two-state chain where each state is associated with different rates λ and a *sojourn time* r representing the mean time the stream stays in that state [14]. For example, a two-state MMPP can be represented with the following 4-tuple:

$$\langle \lambda_1, \lambda_2, r_1, r_2 \rangle$$

To determine these parameters for each state, the original paper [14] makes use of the following values measured from an existing packet trace:

- Mean arrival rate.
- Variance-to-mean ratio of the number of arrivals over a period of time, denoted t_1.
- Long-term variance-to-mean ratio of the number of arrivals.
- The third moment of the number of arrivals in a period t_2.

An underlying characteristic of MMPP is that it preserves most of the analytical properties and simplicity of the Poisson process model. For example, the simplicity of aggregate streams still holds. If a new stream with rate λ' is added to a normal

two-chain MMPP, the resulting new stream could be represented with the following 4-tuple:

$$\langle \lambda_1 + \lambda', \lambda_2 + \lambda', r_1, r_2 \rangle$$

Furthermore, even though the Markov chains add further states to be kept at runtime, individual Poisson processes are still memoryless. The simplicity of the original model is therefore preserved to a certain extent.

Because of these properties, MMPP is often used in literature, existing traffic models, generators and simulators. An example is the *ns* simulator which made use of MMPP when simulating HTTP traffic [38].

Self-similarity

As mentioned in the previous sections, models based on Poisson processes lack burstiness in its generated traffic. The first evidence of this missing property was properly shown by Leland and his colleagues in [20]. When comparing distributions of arrival times for traffic over a long enough time period, real-world traffic exhibits something Poisson processes clearly lack, namely **self-similarity**.

Poisson models follow the **Central Limit Theorem**. This theorem states that any distribution involving independent random variables tend towards being normally distributed [23]. In more descriptive terms, it means that, as the amount of measurements increase, the distribution will smooth to a line around a common mean. This will never occur with a truly self-similar distribution. If a self-similar distribution is bursty at a certain time-scale, it will still be bursty at any time-scale [19]. Mathematically, this is described by the following formula:

$$X(at) = a^H X(t), a > 0 \tag{2.3}$$

where a is the scaling factor, $X(t)$ is the distribution over time and H is the Hurst exponent. If the Hurst exponent is within the interval $0.5 < H < 1$, then the time series is said to be **Long Range Dependant (LRD)**. On the other hand, if the exponent is below 0.5, then the time series is called **short range dependence**.

Calculating the Hurst exponent has been found to be a hard problem. The exponent can only be estimated and even though several different models have been developed to estimate the exponent, they might give differing results. The models fall into two categories, those operating on the time domain and those operating on the frequency or wavelet domain. All of them are quite complex and it is not always clear which gives the most accurate result [19].

LRD and self-similarity are not always important to take into account. Its effects are most noticeable on very large time scales. Furthermore, LRD has been shown to correlate with heavy-tailed distributions that exist, for instance, in large file transfer requests [19]. Given the complexity of using a self-similar model depending on the specific use case, one might be able to avoid taking LRD into account.

Some studies indicate that simpler models could still be applicable in certain situations where the number of sources is very large [2]. As another example, MMPP could be extended to better hold up to self-similarity requirements. With added states to the model, MMPP has been theoretically shown to simulate a self-similar process up to any set time-scale [32][31].

3

Problem Analysis

The purpose of this chapter is to elaborate on the network simulator that we are going to implement. To achieve this purpose, we will start by presenting the requirements for throughput and latency for different 5G applications. After that, a brief discussion of why existing network simulators cannot achieve these requirements will be given. Finally, an analysis of the network simulator that we propose will be provided and the goals as well as the challenges introduced by creating the simulator will be discussed.

3.1 5G requirements

As previously stated, 5G promises high throughput and low latency. This makes it possible for new technologies with strict performance requirements to emerge. While 3GPP has set certain requirements for 5G performance [1], different applications will require different values of the performance metrics.

There is a large spectrum of applications that are expected to benefit from 5G. Some applications have heavy requirements on low- and reliable latency while others take advantage of the higher throughput. In an analysis report, GSMA (trade body representing mobile network operators worldwide) tried to specify and visualize some of these application requirements [17]. They mapped several 5G applications on a scale of 1-1000 Mbps in throughput and 1-1000 ms in latency. In table 3.1, a sample of these applications along with their requirements from all of the spectrum have been summarized.

Augmented Reality (AR) and Virtual Reality (VR) applications using network communication are examples of services in need of both high throughput as well as low latency. Especially very low latency has been stated as a large requirement [27] as this can otherwise highly impact the perception of virtual objects. More specifically, an RTT latency around 1-10 ms is said to be required, while throughput requirements range from 100-5000 Mbps [6].

In addition to the application requirements, there are also some performance requirements specifically set on network components such as UP. Going through the requirements and technology needed for 5G, Vannithamby *et al* [33] calculated the typical latency for a UP component in LTE at around 4.8 ms. As 5G is supposed to

Applications	Latency	Throughput
AR/VR	Low	High
Autonomous driving	Low	Medium
Realtime gaming	Medium	Medium
Video streaming	High	High
Disaster alert	Medium	Low

	Latency	Throughput
Low	< 10	< 10
Medium	10-100	10-100
High	> 100	> 100

Table 3.1: Tables describing 5G application requirements.

be an improvement on existing systems, this could be considered an upper bound for UP-latency in a 5G environment.

3.2 Existing simulators

There are already several network simulators that exist today. However, to our knowledge, none of them fulfill the requirements set by this thesis. Some simulators focus solely on the physical layer and analyze complications arising in 5G during interactions between mobile users and base stations [25][29][22]. Since the performance analysis of these simulators never go deep enough in the network stack to handle EPC or specifically UP routing, the values that these simulators provide complement the results obtained in this thesis.

Another group of simulators does take either UP- or EPC performance into account [24][3]. These simulators are able to simulate some parts of the network used in this thesis but are interested in evaluating other aspects. For example, they are not focusing on the performance of the EPC components on their own but together as a larger system. The focus of these simulators are, for instance, investigating the latency introduced by adding additional UP components to route incoming traffic [3] or the impact of having certain network functions performed at either the edge or at the core of the communications network [24]. Because of this, these simulators do not have the features needed to answer the research questions put forth by this thesis.

In addition to the existing simulators, there is a basic implementation of a 5G simulator at Ericsson that can be used to perform simple tests on a physical UP. This simulator is only single threaded but has been shown to still handle the data rate of 5G. Nevertheless, the existing simulator is very restricted and can only handle data streams in the downlink direction. Moreover, it is not possible to establish more than one single session with the current simulator. To answer the research questions presented in §1.2, the current simulator will be expanded by implementing the set of features described in §1.2.

3.3 Solution description

To fulfil the goals stated in 1.2 and meet the requirements specified in 3.1, we propose a system level simulator aimed to test the performance of the UP at 5G appropriate rates. The simulator we implement will be able to connect to the UP on different interfaces. On one hand, it will act as a CP in order to set up user sessions on the UP. On the other hand, it will act as an SGW and PDN to send payload through the system.

Since the focus is mainly on performance testing, it is unnecessary to use stateful traffic as this would only be needed for functional testing. The traffic generated by the simulator will instead only vary based on different load models.

The following sections will elaborate on what is expected by the simulator. Firstly, the setup of the network environment in which the simulator will run is presented. Secondly, a discussion of the challenges that needs to be addressed when creating the simulator is given. Finally, an analysis of the requirements on the load models that the simulator will support is presented.

3.3.1 Network architecture

This section will go into more detail on how the network environment will look like when 5G traffic is simulated by our cloud-native load simulator.

Figure 3.1 shows the position of the network components within a 5G client-to-server communication. When a UE sends data in the uplink direction, the data is first received by a radio station that is close to the client's position. The radio station assigns a **Tunnel Endpoint Identifier (TEID)** to the client's data and the reason for doing so is to be able to transfer the data through a tunnel to an appropriate UP. Furthermore, to connect with the UP, the tunnel needs to send an **ARP request** to it. ARP is a protocol for mapping link level MAC (physical) addresses to the IP (logical) addresses of the transport layer.

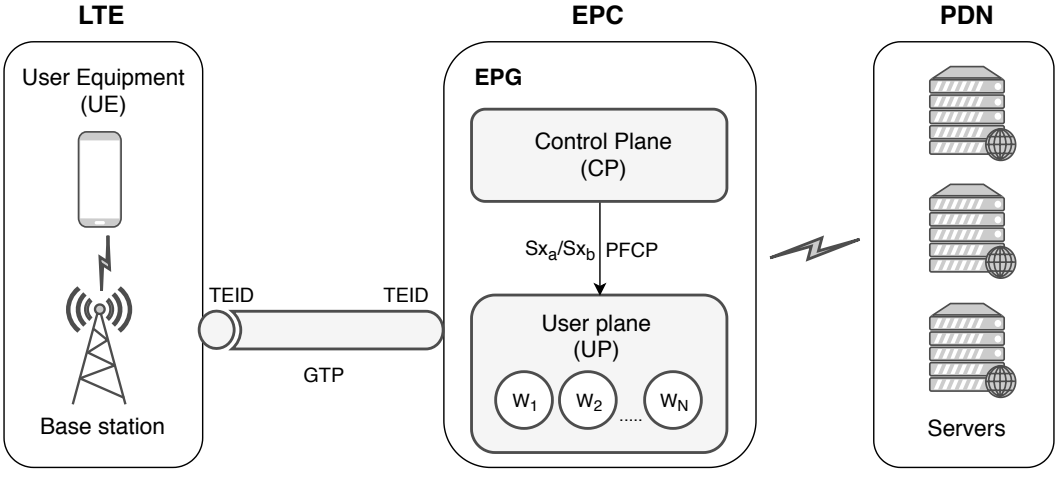

Figure 3.1: Illustration of the simulated 5G network.

Transferring the data is done by following the **General Packet Radio Service Tunneling Protocol (GTP)**. The UP associates traffic to and from each user with a specific user session. One UP can maintain several user sessions at a time but the tunnel that is used for this purpose is unique per each session that is established. It is important to note that there can exist several UPs between the radio station and the UP that forwards packets to the Internet. Figure 3.1 only illustrates the simplest scenario.

The UP can run on virtual machines and containers deployed on different platforms and the major role of this component is to forward the received data to the destination, via the Internet. Shown in Figure 3.1 is a single UP that consists of several worker threads where each thread maps to a separate CPU core on the UP card. The main task of these workers is to await packets and process them in parallel. For every received packet, several possible routes can exist to the destination. Thus, in order to be able to choose an efficient path, the UP receives routing information from the CP. The communication between the UP and CP is done by PFCP using either the Sx_a or Sx_b interface.

The communication in the downlink direction works in a similar manner as the uplink direction. However, forwarding the data from the UP to the Radio Station is not always straightforward. The reason depends on the fact that the Radio Station associated with the client might change if the client's position changes during the time when data is being forwarded from the tunnel. Thus, to direct packets to the correct destination UE, in the downlink direction, MME is needed.

3.3.2 Challenges

The main challenges in creating a simulator as proposed mostly stems from maintaining high rates and setting up a consistent network environment. These challenges are discussed in more detail below.

Optimization challenges

Maintaining the high rates specified for 5G gives rise to optimization challenges. To address the optimization challenges, there are several aspects that need to be considered. First of all, it is of utmost importance to have a well optimized main loop since most of the runtime will be executed here. The main loop refers to the loop in which the simulator sends and receives packets. Any reduction in calculations might have a large impact in performance since the main loop will be executed many times.

Another optimization is to avoid executing costly instructions as much as possible. This would include any system call, as the operative system adds unnecessary overhead to these. The same goes with the normal system calls used for sending network traffic. Preferably, one would like to avoid this overhead when sending packets. As we are targeting a cloud native deployment, these overhead-factors are even more pronounced as a container introduces further layers to send traffic.

Storing the output correctly does also become a challenge at such high rates. One aspect to consider is how to store the output in memory during runtime. Time paused to allocate more memory should be kept at a minimum level. Another aspect is how to store the output in the file system and when to do so. It is not desirable for either of these aspects to interfere too much with the normal procedure of generating the load.

Network challenges

There are also several challenges connected to the network environment and setting up a working test environment. Even though our simulator will not perform functional testing or use stateful traffic, it still needs to follow some standard protocols.

One protocol that must be followed in order to set up user sessions is PFCP. For the UP to forward traffic to and from different UE addresses, it needs to have a user session configured for that address. The simulator needs to set up these sessions and store certain values about each session to be used in traffic.

For example, the simulator needs to store the TEID associated with each user session. This is to be used whenever traffic is being sent from that particular UE in the uplink direction. In this direction, the simulator is simulating an SGW and needs the TEID to communicate with the UP using the GTP protocol.

3.3.3 Load model analysis

When discussing different traffic models to support in the simulator, there are a set of criteria that need to be considered. These are stated below, in no particular order:

- Simple parameters
- Realistic traffic
- High generating speed
- Low resource usage
- Usefulness

Simple parameters are needed since one of the overall goals of the simulator is to be easily configurable. This means that it should be easy to both specify the behaviour of a load model and change the behaviour between each simulation. This will be achieved by providing a configuration file that is parsed before each run of the simulator.

Realistic load is one of the main kinds of load used in load testing and this is discussed in Chapter 2.3.2. Generating *realistic traffic* is an important criteria and how realistic some load models are compared to normal network communication has been discussed in §2.3.3.

Another one of our overall goals for the simulator is to have low overhead. Therefore, the load types that will be supported by our simulator needs to have *high generating speed* while maintaining *low resource usage*.

Lastly, *usefulness* is arguably the most obvious criteria but also much harder to define. We will argue for a load type's usefulness from use cases found in this thesis and potential future use cases.

The simulator we implement will provide support for three different load models: *Steady rate load model*, *Step-wise rate load model* and *Poisson processes*. Each of these load models will be evaluated on how well they fulfill the criteria presented in this analysis.

4

Implementation

This chapter presents the implementation of our simulator, its runtime structure and how it is used to execute load with different load types. The first section will provide a brief overview of how the simulator works and it is supposed to give enough understanding for a potential user to be able to use the simulator and run tests. The second section will elaborate on the different load types that are supported by our simulator. The third section will present the runtime phases of the simulator which includes everything from setting up a simulation to executing it.

4.1 Simulator overview

A test that is run by the simulator can be split into two phases: the **setup phase** and the **load execution phase**. To control the simulator behaviour during these phases, a user specifies different configurations in a YAML file. In this file, it is possible to set variables such as IP addresses for different network devices and the payload length of packets. Moreover, a user can specify the way that the simulator will send packets during a test. Note that this file is only read by the simulator in the setup phase and because of this, a modification of the configuration file during a test will not change any behavior of that test.

How traffic is sent to and from a UE address during a test is set by what is called *load-types*. In the configuration, these consist of a load-type name and a set of options that differ depending on the specific load-type used. For example, you can specify the rate of traffic and for how long traffic will be sent with that specified rate. A configuration file can have an arbitrary number of load types in it. More details on which different load-types are supported and what options each one has is described in §4.2.

How many UE addresses there should be during a test is specified by *user groups*. A user group describes how many users it consists of, how many flows each of those users have connected to them and what load type characterizes the traffic on each of those flows.

The simulator runs until all load-types have finished sending traffic. During the run, the simulator can be configured to output some runtime statistics every five seconds. After that, it will go through the data from the test, calculate some statistics and

finally output the results as CSV files. These CSV files can then be used to create charts that might be useful for visualization of the simulation result.

4.2 Supported load models

Load types are specified in the configuration file and it is used to determine the nature of the traffic during the simulation. The parts described in this section are executed during the sending step in the load execution phase (as described in §4.3.2).

A common property that all of the supported load types make use of is a timestamp, Δt, that specifies the time between each iteration of the load execution loop. During load execution, traffic is sent over flows and each flow is associated with a load type. This makes it possible to calculate how much traffic should be sent on a flow during the time specified by Δt.

In addition to the shared timestamp, all load types also calculate how many packets that need to be sent on the flow in the uplink- as well as the downlink direction. However, this calculation differs depending on the load type that is being used. For that reason, this will be described in connection with the respective load types below. For simplicity, packets needed to be sent on each flow will be referred to as *required-packets*.

Note that in order to handle very low rates, required-packets have to be aggregated as a floating point value. Example of a scenario where this is necessary is when the rate times Δt becomes less than one. In that case, the required-packets would get rounded down to zero. Thus, if an aggregated floating point value is not used, a low rate would most likely never be able to send a packet.

4.2.1 Steady-rate load types

The steady-rate load type describes traffic holding a constant rate of packets over a set time interval. There are three different attributes that can be specified for steady-rate in the configuration file. The first two denote how many packets that should be sent per second in the uplink- and downlink direction whereas the third attribute specifies the duration of the load type.

Each time a flow uses steady-rate to send traffic, the simulator first checks whether the time for the entire simulation has expired. In that case, it sets the rate as well as packets that need to be sent in both directions to zero, indicating that the simulator should no longer be allowed to send traffic on flows associated with this load type.

On the other hand, if the simulation time has not expired, the simulator calculates the required-packets. This calculation is simply the rate in the given direction times the time it would be sent over, in this case, Δt.

4.2.2 Step-wise rate load types

Expanding on steady-rate, the step-wise load type describes traffic whose rate changes at set times during a test. Similar to steady-rate, this load type is also associated with three attributes: uplink rate, downlink rate, and time interval. However, this load type allows each of these to be specified in a sequence. Each sequence describes how traffic should be sent during the current time interval.

Initially, this load type compares the simulation time so far to determine whether the timer is outside the duration of the load type. In that case, the rates in both directions, as well as required-packets, are set to zero. If the simulation time, on the other hand, has not expired then the current time interval is decided. Lastly, the rates in both directions and required-packets are calculated for the current time interval, in a similar way as described in §4.2.1.

4.2.3 Poisson processes

The Poisson process load type describes traffic modeled by a set mean rate, but varying sending times, as described under Poisson processes in §2.3. Our implementation of Poisson processes follows closely to the version presented by McQuighan [21] but differs by only simulating over finite time intervals.

Poisson processes in our context are specified by three attributes in the configuration file: mean uplink and downlink rate as well as a time interval. The implementation of this load type uses a uniformly distributed random number u between [0, 1] in order to calculate a timestep t, that is used to determine the time that a packet should be sent. This is done in the following way:

$$t = \frac{-log(1-u)}{\lambda} \tag{4.1}$$

where λ is the mean rate, either in uplink- and/or downlink direction.

This process is repeated in both directions until the sum of the timesteps for the respective direction exceeds Δt. Finally, the required-packets is set to the number of timesteps able to fully fit inside Δt. Consequently, the last timestep will not necessarily be counted as a full packet. To solve this, it has to be added as an aggregated part that is equal to the percentage of the last timestep that was able to fit inside of Δt.

4.3 Simulator runtime phases

This section will walk through the implementation of the simulator from when it is setting up a test until the test has finished executing. This entire procedure of a test is divided into two phases: the **setup phase** and the **load execution phase**. The implementation specifics of these phases are presented below.

4.3.1 Setup phase

At the setup phase, all basic steps needed for the simulator to run a test are performed. This starts by building the configuration file and the Docker image, sending them to the test server, loading the image on the server, making adjustments to the configuration file if necessary and finally, running the program. Once the program starts to run, it goes through different stages and performs necessary calculations to be able to run the actual simulation. These stages are shown in Figure 4.1.

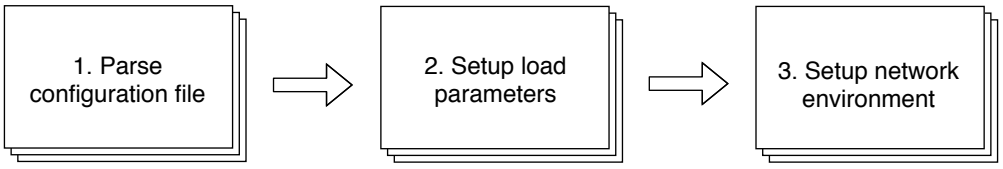

Figure 4.1: Overview of the steps included in the simulation setup.

Parse configuration file

When the program enters its execution state, the first thing it does is to read the configuration file that was sent to the test server. This file is written in YAML and it can contain values of different data types. Some values might be a single integer or string while other values can be a sequence of different data types.

In order to properly parse values written in YAML to C, functions provided by the official library of YAML is utilized. Once the entire configuration file has been parsed correctly, the program continues with the next step in the setup phase which is to *setup the load parameters*.

Setup load parameters

Using the values specified in the configuration file, the program calculates most of the parameters needed to run a simulation. Initially, values such as *number of user groups*, *users per group* and *flows per user in a group* are used to calculate the total number of users u, and flows f, using 4.2 and 4.3 in the following way:

$$u = \sum_{i=1}^{g} u_i \tag{4.2}$$

$$f = \sum_{i=1}^{g} u_i f_i \tag{4.3}$$

where g and u_i in 4.2 and 4.3 denotes the total number of user groups and number of users u in group i respectively. f_i in 4.3 specifies the number of flows per user in group i.

A known value for u makes it possible to create simulated UE with IP connectivity. Packets that are sent to and received by IP addresses corresponding to a UE are always encapsulated inside GTP. This makes it possible to choose the IP addresses for UE arbitrarily. In addition to creating the simulated UE, the total number of users does also imply how many user sessions that should be established during the simulation since the number of sessions are equal to the number of UE.

Calculating f, on the other hand, makes it possible to determine how many PDNs that need to be created for the simulation. However, there are some restrictions that one needs to adhere to. For instance, the IP address for the PDN needs to be within the same subnetwork as the intermediate device that routes packets from a specific PDN to the UP. Therefore, it is important that the correct network address is specified in the configuration file.

Furthermore, having determined the total number of flows, it becomes possible to iterate through each of them and associate flows with values that will be used during the simulation. One example of an important value is which load type the flow belongs to. Several different load types with particular settings can be specified in the configuration file. One important attribute that is included in each load type is which user group they belong to. All users in a certain group are assumed to have the same load type. As a result, it becomes possible to associate each flow with a specific load type, when iterating through the flows.

As mentioned in §4.2, all load types store information regarding rate, i.e., the number of packets that should be sent in the uplink- and the downlink direction each second. The load types do also store information about time intervals that specifies the duration of the entire simulation that these rates should hold. Having these values stored, it becomes possible to calculate the total number of packets that should be sent during an entire simulation.

Number of packets n, when using either steady-rate or poisson processes as the load type is calculated by the following equation:

$$n = \sum_{i=1}^{\tau} k_i \cdot d_i \cdot (\Delta_{\uparrow i} + \Delta_{\downarrow i}) \tag{4.4}$$

where τ specifies the total number of load types, k_i denotes the number of times the same load type occurs, d_i specifies the duration of the load type i, i.e., for how long traffic should be simulated with the given uplink rate $\Delta_{\uparrow i}$ and the downlink rate $\Delta_{\downarrow i}$.

If the load type, on the other hand, is step-wise rate, then the uplink and the downlink rate can vary during different time intervals. Taking this into consideration,

formula 4.4 above is modified to:

$$n = \sum_{i=1}^{\tau} \sum_{j=1}^{t} k_i \cdot d_{ij} \cdot (\Delta_{\uparrow ij} + \Delta_{\downarrow ij}) \tag{4.5}$$

where t denotes the number of time intervals for each load type that are step-wise, d_{ij}, $\Delta_{\uparrow ij}$ and $\Delta_{\downarrow i}$ respectively denotes the duration and rates of load type i and time interval j. The total number of packets to be sent during an entire test is obtained by calculating the sum of the number of packets for each load type respectively.

Setup network environment

This phase of the setup involves steps where the program needs to communicate with its connected network. To carry out this communication, a **Data Plane Development Kit (DPDK)** session is established and this makes it is possible to interface with the network. The motivation of doing this is to bypass the kernel and avoid context switches introduced by the operating system when sending and receiving messages. Avoiding context switches leads to less overhead for the simulator when sending packets to and receiving packets from the UP. This reduction in the overhead is not specifically needed to set up the communication that follows in this section, but will be a contributing factor to the simulator performing as fast as required during the remainder of the test.

To be able to communicate with the UP during the test, the program needs to know the MAC addresses for certain intermediate devices between the UP and itself. These addresses are unknown to the program and needs to be discovered from the network environment. To do this, it performs ARP requests to the network and when a response is obtained, the corresponding address is mapped to an ARP-table for future use.

The last thing the program does before actually running the test is establishing a PFCP session with the UP. Using this session, the program is able to communicate with the UP, as if it was a CP, and set up user sessions between the simulated UE and PDNs needed for the test. One important parameter that is used when setting up a PFCP session is a unique identifier (TEID) for the GTP tunnel. This needs to be known by the UP so it can figure out which tunnel that needs to be used to forward traffic that belongs to a certain user session.

4.3.2 Load execution phase

When all the configuration- and environment setup is properly finalized, the simulator goes into its main operating mode which is load execution. The load execution phase mainly consists of a loop which is based on a timer that specifies the duration of the test. The loop itself consists of two fundamental steps: sending packets to and receiving packets from the UP. A flow chart of this phase can be seen in Figure 4.2.

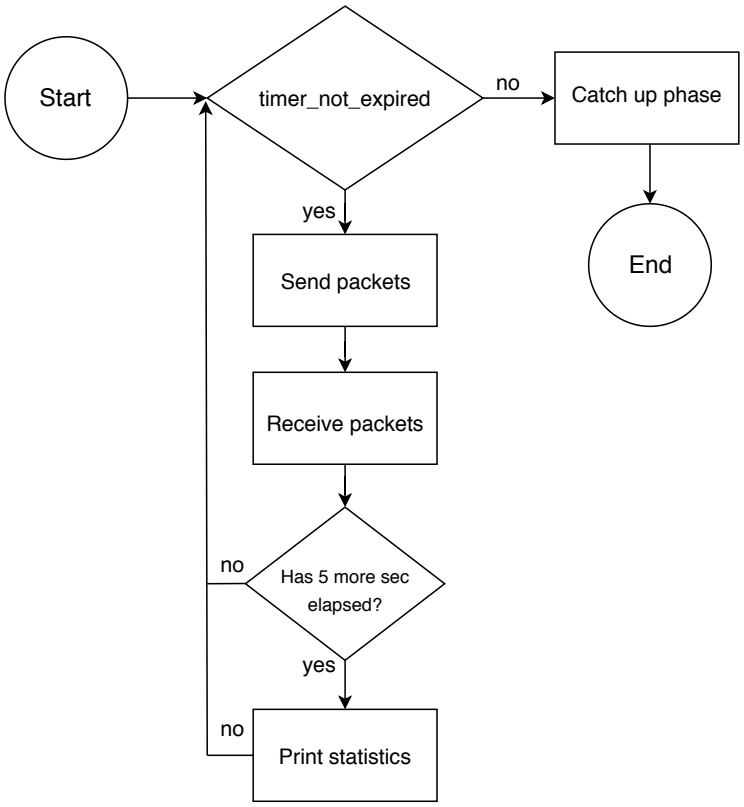

Figure 4.2: Load execution illustrated in a flow chart.

Sending packets is done in bursts and each burst can contain a maximum of 32 packets at a time. The simulator goes through all existing flows and checks whether packets should be sent on them. By keeping track of the time that has passed since the last repetition of the loop, the simulator calculates how many packets it will need to send to maintain the correct rate in either direction. If the simulator is not able to send all packets needed to keep a given flow's rate within the 32 packet-burst limit, the simulator will store the number of leftover packets and send them in future iterations of the execution loop. The exact algorithm for calculating the number of packets needed to be sent on a flow depends on the load type associated with the flow as described more specifically in §4.2.

The structure of a packet to send differs depending on the direction that the packet is traversing in. Since packets traveling towards the uplink direction would reach the UP through a GTP-tunnel, these need to be specifically encapsulated inside a GTP message holding the correct TEID. Other than that, packets in either direction are UDP packets with source or destination set to corresponding UE or PDN address, depending on direction. Moreover, necessary fields in the Ethernet header are set in order to correctly route to the UP. The length of the packet's payload is fixed and the first eight bytes are always used to distinguish packets with a unique identifier. Lastly, before sending a packet, a timestamp indicating the send time is recorded.

The second step in the load execution loop receives packets incoming from the UP. Once a packet is sent out by the UP and received by the simulator, it is decapsulated

and assigned a receiving timestamp. Similar to the sending phase, receiving is done in bursts of 32 packets. Though, differing from sending, receiving packets is repeated in successive bursts until all current packets received from the UP have been handled.

On top of these two steps, the simulator is also able to print a summary of some performance metrics once every five seconds. This includes the current rate in both directions in bits per second as well packets per second, a load ratio that indicates the CPU utilization for the time being and the average time it takes to send packets in one iteration of the loop. Nevertheless, this output is meant to make it easier to follow the execution of a test during its runtime. A more detailed evaluation of the performance of the test will be described in connection with Chapter 5.

After the test is done, i.e., the specified timer for the test has expired, the simulator stops sending packets to the UP and performs a small *catch-up phase* of approximately five seconds. In this phase, the simulator only receives packets from the UP. The main purpose of executing this step is to give the last packets sent to the UP some time to pass through the system correctly instead of directly marking them as lost packets.

5

Evaluation

In order to properly understand and accurately validate the results from a simulation, we need to analyze different performance metrics and set up criteria for evaluating the test results. This chapter describes how different simulations are designed, what results that can be obtained from a simulation, how these results are interpreted and how they are evaluated.

5.1 Evaluation metrics

After a simulation has finished executing, the program starts to evaluate the recorded packets that are received back from the UP. Statistics such as means and aggregated values about the test are calculated. Most statistics are calculated over a number of equally sized sampling steps. To clarify, assume that a simulation is run for 60 seconds and the number of sampling steps is set to 6000. This means that the output of the program will contain data points for every 10 milliseconds.

As a result of calculating statistics over defined sampling steps, it becomes possible to analyze measurement differences over specific parts of a test, such as before and after the load is drastically increased. If a statistic over a full test is desired, one can simply calculate it using a single sampling step. Once all statistics have been calculated, they are written to CSV files and scripts are provided to output the data from CSV files to graphs.

Latency

There are two different notions to present the latency over a test. The first one is **individual packet latency** which simply presents the latency of each packet at the point of arrival. Since all packets are associated with timestamps during the load execution phase, one can determine the latency of each packet p by subtracting the sending time $t_s(p)$ from the receiving time $t_r(p)$ as:

$$L(p) = t_r(p) - t_s(p) \tag{5.1}$$

The second one is **bucket latency** which is defined as the average latency over a sampling step. This splits the entire test into sampling steps in order to present the

latency. This approach starts by determining the *duration* of the entire simulation which is calculated by subtracting the start time from the end time. This duration d is then used to determine the length of each sampling interval ℓ by:

$$\ell = \frac{d}{s} \tag{5.2}$$

where s is the number of sampling steps.

All packets that are received during a simulation can be associated with one of the existing sampling intervals s_a. The receive time for each packet is divided with ℓ to calculate which interval a particular packet p should be in. This is shown by Equation 5.3

$$s_a = \{p \mid \lfloor \frac{t_r(p)}{\ell} \rfloor = a\},\ a \in [0, s] \tag{5.3}$$

Finally, the mean latency of the packets in each sampling interval is calculated. This is done by calculating the sum of the latency for each interval and divide it by the number of packets in that interval, as shown in Equation 5.4

$$L_s(a) = \frac{\sum_{i=1}^{n} L(p_i)}{|s_a|},\ \forall p \in s_a \tag{5.4}$$

where n is the total number of packets sent during the test.

Throughput

Besides latency measurements, the simulator does also measure the incoming and outgoing rate. Each time the simulator receives a packet from the UP, the size of the corresponding packet is saved. Once a simulation has finished executing, the entire simulation time is divided into several time intervals that are specified by the user. Since each packet is associated with a sending and receiving timestamp, it is possible to determine which time interval every packet should belong to. This is done in a similar manner as shown by Equation 5.2, Equation 5.3 and Equation 5.4.

When the sampling interval for each packet has been decided, the incoming and outgoing rate for the respective interval can be calculated by:

$$\text{rate} = \sum_{i=1}^{s} \frac{n_i}{\ell} \tag{5.5}$$

where s is the number of sampling intervals, ℓ denotes the length of an interval and n_i denotes the number of packets in sampling interval i.

Note that when the rate is calculated, it is important to differentiate between the incoming and the outgoing rate. For the incoming rate, it would be necessary to look at the sending timestamp and for the outgoing rate, it would be necessary to look at the receiving timestamp.

Packet loss

After a simulation, all packets without a receiving timestamp are recorded as packet loss. The program does not only show the total number of packets lost during the entire simulation, but more importantly, it is able to show how many packets that were lost during specific time intervals. In order to do this, the program once again decides which time interval each packet falls into. This is done in the same way as described in §5.1. Since the lost packets by definition do not have a receive timestamp, they can only be presented according to their sending time.

Moving average

The moving average can be calculated for both latency and throughput and it is used to smooth data when the output is too erratic to be properly analyzed. The calculation uses a window whose size can be specified in the configuration file. When calculating the moving average, the specified window is moved over the input values, one at a time. For each time the window is moved, an output value is produced and this value presents the mean of the values in the corresponding window.

Simulation overhead

In order to analyze the overhead introduced by simulating 5G communication, the simulator performs two different measurements. The results provided by both of these measurements are presented in nanoseconds.

First of all, the simulator keeps track of the time it takes for each loop (as seen in 4.2) to iterate once. This data is presented as a histogram over loop times, showing the number of loops with corresponding loop time inside each time interval.

Secondly, the simulator is able to compute the average time it takes for a loop to send packets on the associated flow. This is calculated as the time it takes for the loop to iterate, divided by the number of packets that were successfully sent. If no packets are sent on a loop, that loop is skipped and not accounted for in the measurement.

Verification of load test

In order to validate the results of a complete load test i.e., a target run, the program compares the result of that test with a baseline test. The target run is then scored depending on the results. This is done by using the CtrlCharts technique as described in §2.3.1. The verification process is performed in several steps and these are described below.

As a starting point, a baseline test is defined. From this baseline test, the program is able to set the CL as well as the UCL and LCL. In our implementation, the CL is the median of the baseline test readings and the UCL and LCL are set as three standard deviations from this median. Using these values, the program can score any subsequent target run by outputting the ratio of data points outside these control limits. Once this ratio becomes high enough, the target run is considered to

have failed. Note that determining how high the ratio is allowed to be is somewhat arbitrary and needs to be decided from performing some experimental tests first.

5.2 Test design

This section is intended to discuss the design and configuration choices made for the tests that are presented in Chapter 6.

Uplink and Downlink ratio

When a test is run, the measured incoming rate sent to the UP could be a combination of packets in any direction. More specifically, it could be in uplink or downlink or any ratio in between. For example, one could see a test where rate is 2000 Mbps in uplink and 2000 Mbps in downlink direction, for a total of 4000 Mbps sent to the UP.

To have the most accurate presentation of our results, we show three different scenarios for most tests. One with the rate only sent in the uplink direction, one in only the downlink direction and one with the rate going in both directions. These will be called the UL, the DL and the COMBINED scenario respectively. In the COMBINED scenario, the rate will be equally split in each direction, so the rate in either direction will be half of what is presented in the result.

Packet size

During a test, the size of the packets sent by the simulator has a large impact on the throughput measurement. Because of this, it is important to keep this variable constant over all tests. We have chosen 750 bytes as the size of the payload for every packet, in all tests. However, note that the real packet size will be slightly larger than this and differ in size depending on the direction because of different header encapsulation, as discussed in §4.3.2.

Note that when the UP receives a packet traversing in the uplink direction, it will decapsulate the GTP header before forwarding it to the destination. As a result, the size of the entire packet will become smaller and thereby also the number of bytes that the UP sends out. On the other hand, if a received packet is traversing in the downlink direction, it will be encapsulated with a GTP header in order to be transported through a GTP tunnel. In this case, the size of the packet will become larger, and in connection with that, the number of bytes that are sent per second will increase.

Now, consider a scenario where packets are being sent by the simulator with a rate of 5 Gbps in the UL and the DL scenario respectively. As a consequence of the encapsulation and decapsulation made by the UP, the rate in the uplink direction will become a little less than 5 Gbps and the rate on the downlink direction will be a little more. Thus, the throughput results of the UP might seem to deviate from

what the simulator is sending, but in reality, they are actually sending the same amount of packets.

Architecture used at Ericsson

We have had access to two different UP-setups at Ericsson. Hereinafter, the first setup will be referred to as UP1 and the second one will be referred to as UP2. UP1 has a limited number of worker threads compared to UP2. Therefore, it has been chosen to only test the performance of the simulator and not to analyze the behaviour of the UP.

UP2, on the other hand is designed to handle much higher rates compared UP1 and is used to test the UP behavior. For this reason, UP2 is used in this thesis to perform load testing and to investigate the maximum rates of different configurations.

5.3 Simulation scenarios

This section aims to describe the different test scenarios we will use to present our results in chapter 6.

Load type verification

In order to properly load test the UP, it is first important to show that our implemented simulator behaves as expected. For this reason, some basic simulations are performed. The purpose of these simulations is to show that the simulator can generate traffic with the supported load models i.e., steady-rate, step-wise rate and Poisson processes, in the UL, DL, and COMBINED scenario. For each simulation, the number of UE is set to 1 and the number of flows per UE is set to 100. For Poisson processes, we will also present a distribution graph based on the results from the simulation to verify that the simulation correctly follows a Poisson distribution.

Maximum rate of the simulator

The implemented CtrlCharts technique for verifying a load test can be used to determine the maximum rate of the simulator with different setups. One can define the max rate of the simulator as the highest throughput rate where the simulator still holds a satisfying violation ratio. In the case of the simulator, the baseline would be created by calculating the difference between the rate specified for a test and the rate the simulator outputs, at a rate the simulator is known to handle.

In order to determine the maximum rate, we first run some experimental tests for the UL and DL as well as COMBINED scenario separately. This is done to observe the behaviour of the simulator when generating traffic with a different configuration (rate, number of flows, users, etc.). From these observations, a baseline is defined at a configuration where the rate is stable.

As previously mentioned, our simulator can generate traffic with different setups such as a single user with multiple flows, multiple users with single flows and multiple users with multiple flows. Therefore, there is no single notion for the maximum rate of the simulator, but rather a maximum rate for a certain setup. So, in order to determine the maximum rate for different setups, a number of different simulations are performed.

Nevertheless, for all the simulations, the verification technique using CtrlCharts were used in order to compare the different simulations with a baseline and calculate a violation ratio. Violation ratio is defined as the percentage of data points that falls outside the UCL and LCL. We define the maximum rate of the simulator as the rate at which the simulator no longer holds a smaller than 1 % violation ratio from the UCL and LCL of the baseline. We use separate baseline values for setups with different uplink and downlink rate ratios, but keep them the same for any changes in the number of flows and users.

Maximum rate of the UP

Having determined the maximum rate of the simulator for different setups, it becomes possible to load test the UP and determine its maximum rate for the corresponding setup. Assume that the maximum rate for the simulator, when generating traffic with 100 UE and 10 flows per UE is x. This means that we are able to test the UP up to rate x by e.g., performing simulations with step-wise rate and observe when the UP starts to deviate from x, i.e., when it no longer can hold the rate of the simulator.

When defining the maximum rate of the UP, we do not make use of the CtrlCharts technique used for the maximum rate detection of the simulator. This is because any deviation in the throughput of the simulator and the UP must either show up as an increment in latency or as packet loss and while latency could be tested using CtrlCharts, packet loss could not. It would not be possible to create a baseline for packet loss since during any stable test eligible as a baseline, there should not be any packet loss at all.

A consequence of not having any packet loss would not yield the deviation needed to create a UCL and LCL. For simplicity, only the packet loss ratio over a test will be used to define the maximum rate of the UP. We define the maximum rate as the highest rate the UP can output before 0.1 % of all packets are lost.

One last thing that is important to point out is that there might exist scenarios when the UP is load tested up to the maximum rate of the simulator, but still never shows any loss in packets. In these cases, it is not possible to specify an exact number or an approximation of the maximum rate of the UP. Instead, all we can state is that the UP is at least as fast as the simulator and can most likely be load tested with higher rates that our simulator cannot handle according to the verification process with CtrlCharts.

6

Results

This chapter will present the results achieved in this thesis in order to verify that the purpose of the thesis (presented in §1.2) has been achieved. The first section is intended to show that the load models supported by the simulator behave as expected. The second section will present the approach that has been taken to determine the maximum rate of the simulator as well as its results, followed by the third section that will show how the maximum rate of the UP is detected as well as its performance metrics. The simulations that are presented in the first two sections have been performed on UP1 whereas the results presented in the third section are obtained by running simulations on UP2.

6.1 Load types behaviour

There are three different load models that are supported by our network simulator. Each model was used several times with different setups to perform simulations and load test a physical UP. This section will present the behaviour of the simulator when running different tests with one single user that has 100 flows. All simulations that are presented in this section have a duration of 60 seconds except for the Poisson process whose duration is 120 seconds. The data points used to visualize the graphs have been sampled every 10 milliseconds. The default payload length for each packet is set to 750 bytes and the moving average presented for the throughput has a window of size 100.

6.1.1 Steady-rate behaviour

Figure 6.1 shows the throughput of the simulator when traffic is simulated for the UL scenario and Figure 6.2 shows the throughput for the DL scenario. In both of these simulations, a steady-rate of 5 Gbps is specified. The moving average which is represented by the orange line is very stable around the specified rate while the actual throughput that is represented by the blue line can be seen to vary with approximately ± 0.05 Gbps in the respective direction. Furthermore, to more accurately visualize that the specified steady-rate is kept steady, Figure 6.1 is zoomed zoomed in to cover 3 seconds and the result is presented in Figure 6.3.

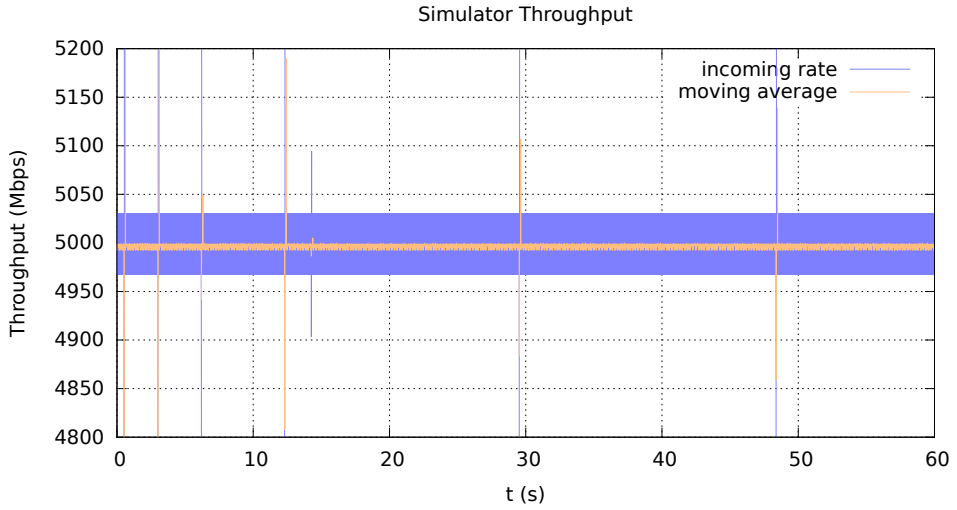

Figure 6.1: Simulator throughput for UL with steady-rate 5 Gbps.

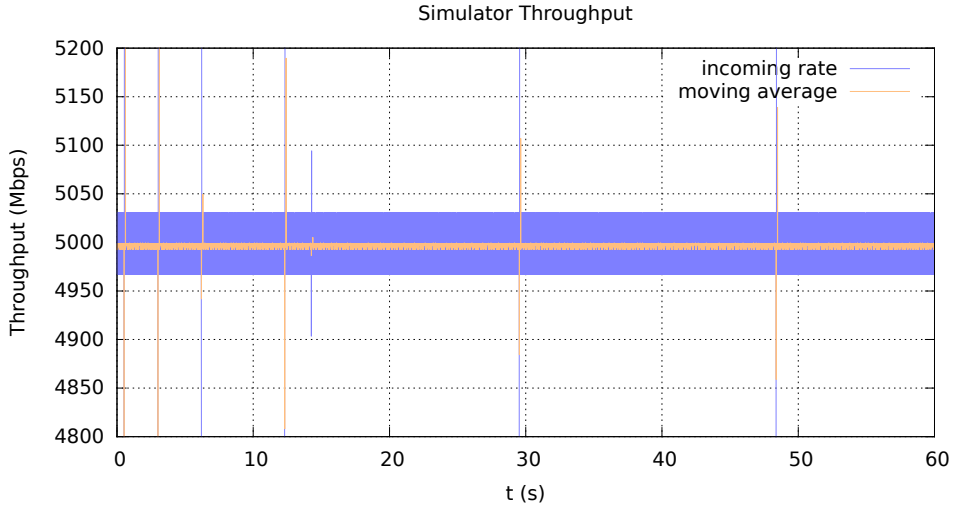

Figure 6.2: Simulator throughput for DL with steady-rate 5 Gbps.

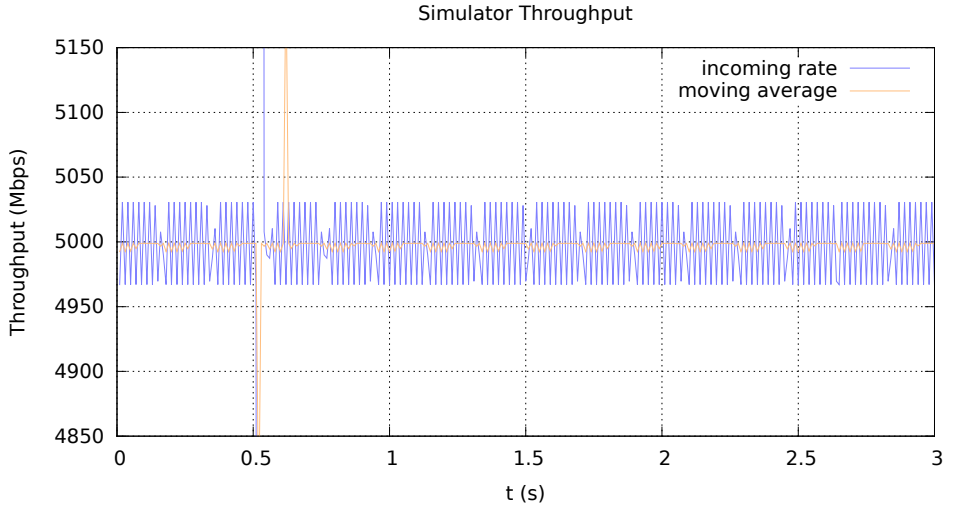

Figure 6.3: Steady-rate 5 Gbps UL, zoomed in over 3 seconds.

We have also performed simulations in the COMBINED scenario. Figure 6.4 shows the throughput of the simulator when traffic generated with 2.5 Gbps in the UL and DL scenario respectively. Similar to the simulations with only one direction, the moving average is very steady around the specified rate. However, the actual throughput is not maintained as steadily when combining the directions. It varies roughly around 0.1 Gbps above and 0.05 Gbps below the specified rate.

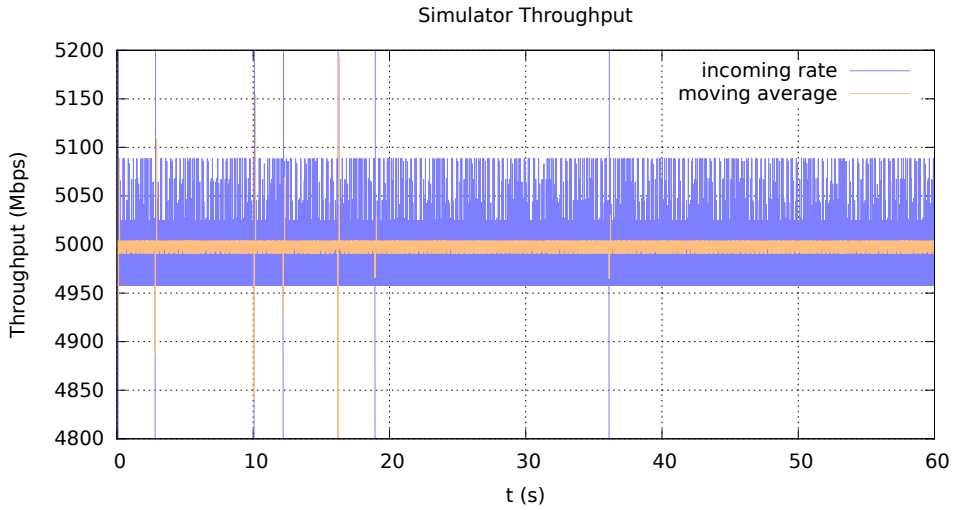

Figure 6.4: Simulator throughput with steady-rate 5 Gbps for COMBINED scenario.

6.1.2 Step-wise rate behaviour

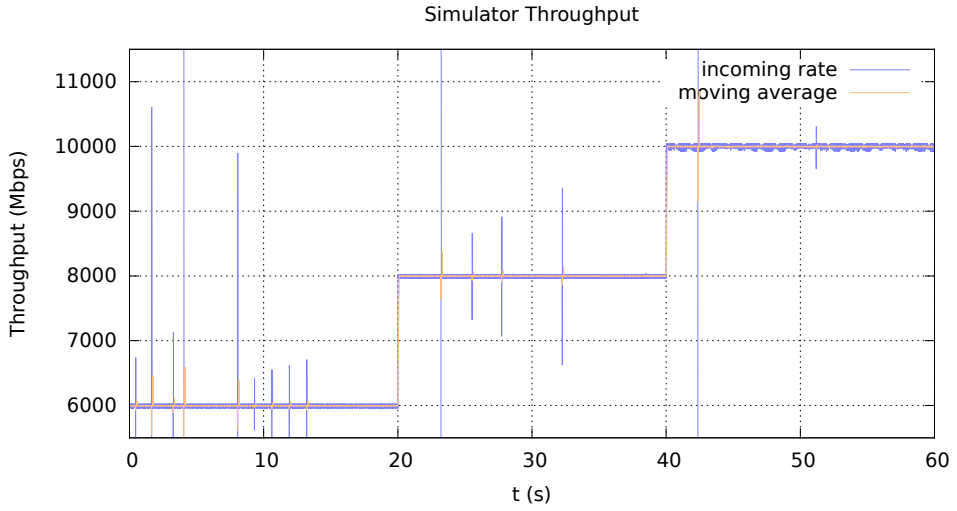

Figure 6.5: Simulator throughput when generating traffic with varying rates.

Figure 6.5 present the throughput of the simulator when generating traffic with step-wise rates. For this simulation, the entire duration of the test is split up into three equally sized time intervals of 20 seconds each. For the first 20 seconds, traffic is

generated for the UL scenario with 6 Gbps. The next 20 seconds, the rate is increased to 8 Gbps and the scenario is changed to DL. Finally, for the last 20 seconds, we generate traffic for the COMBINED scenario with 5 Gbps in UL and DL respectively. Similar to the results presented by steady rate simulations, we can see that the moving average is around the specified rate during all three time intervals and the actual throughput varies more when simulating in both the COMBINED scenarios.

6.1.3 Poisson processes behaviour

For Poisson processes, we will present the throughput of the simulator when the mean rate is specified to 5 Gbps for the uplink and downlink direction together. Figure 6.6 shows an example of this scenario and it can be seen that the actual throughput varies with roughly 0.1 Gbps above and 0.2 Gbps below the specified mean rate. Furthermore, it can be seen that the variations in both the moving average as well as the actual throughput are much more frequent compared to the corresponding simulation with steady-rate that is shown in Figure 6.4.

In addition to the variations in throughput, it can also be observed that the spikes that are distinct in Figure 6.1 - Figure 6.5 are not at all noticeable in 6.6. Our hypothesis for the reason of this behaviour is that the packet burst are varied by the Poisson process instead of being fixed as for steady-rate and step-wise rates.

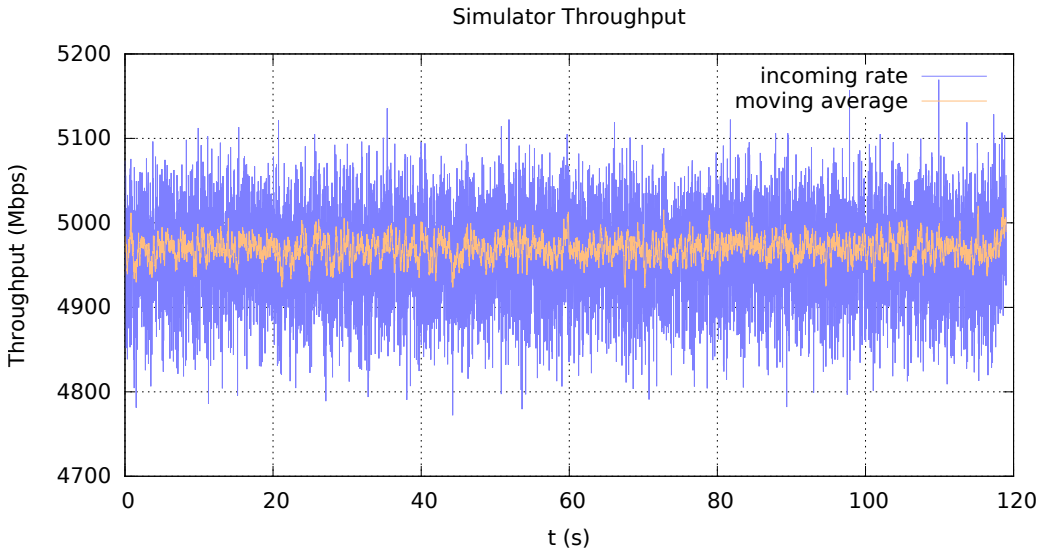

Figure 6.6: Simulator throughput when traffic is simulated using Poisson process.

To rigorously observe the variation in the throughput of the Poisson process compared to the steady-rate, Figure 6.6 and Figure 6.4 is zoomed in and visualized with a window of 1 second. This is presented by Figure 6.7 and Figure 6.8.

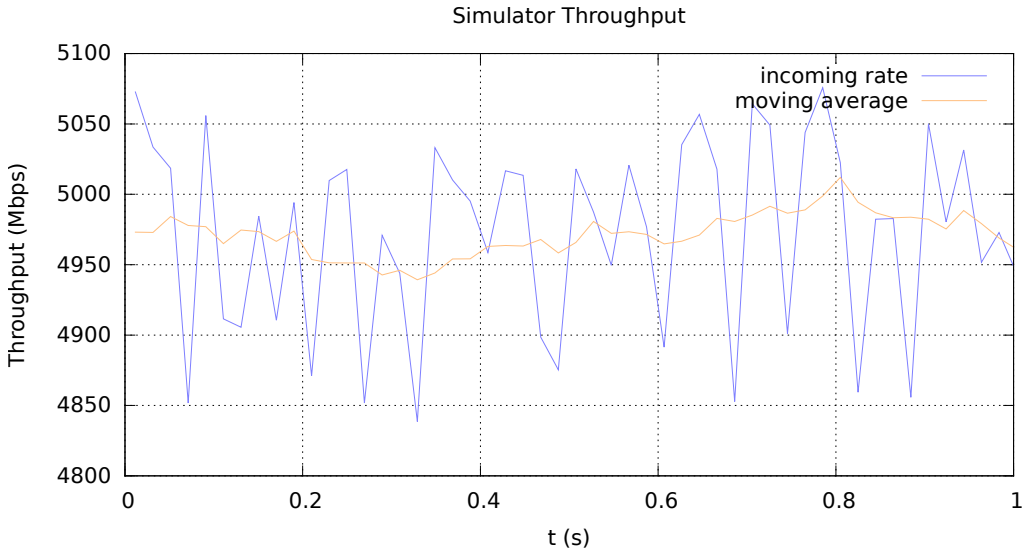

Figure 6.7: Poisson process zoomed in over 1 second.

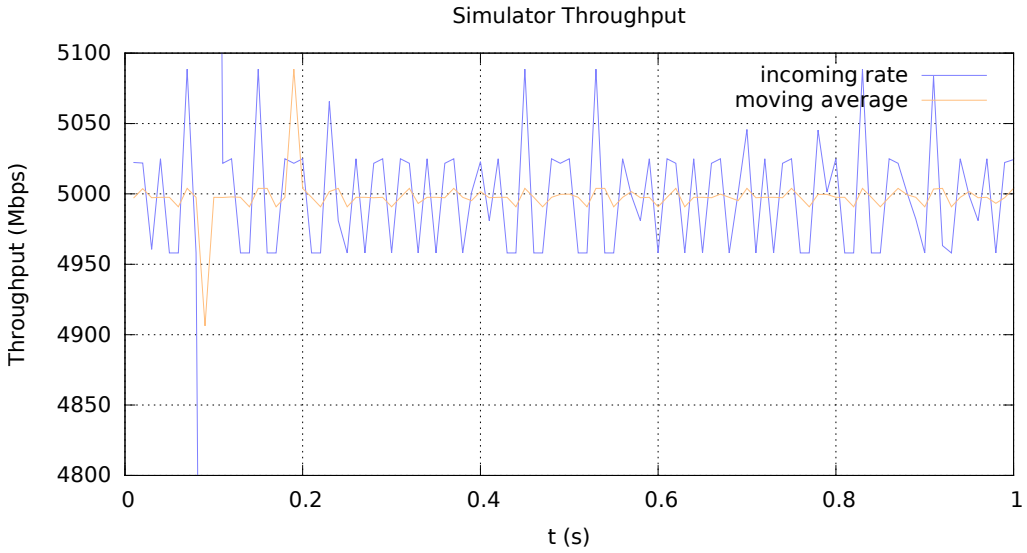

Figure 6.8: Steady-rate of 5 Gbps in both directions zoomed in over 1 second.

To more clearly show that the distribution of the packets sent by the simulator actually follows a Poisson process, we calculate the number of packets sent each 100 ms over a test of 60 seconds and show the distribution of this in a histogram. This is presented by Figure 6.9 below. For this experiment, a total rate of 5 Gbps was used. One-third of this rate was in the uplink direction while two-thirds were sent in the downlink. With an exact size of 836 bytes for packets in uplink and 796 bytes for packets in downlink (750 bytes payload + encapsulation), the mean rate

in packets per second should be 77224.

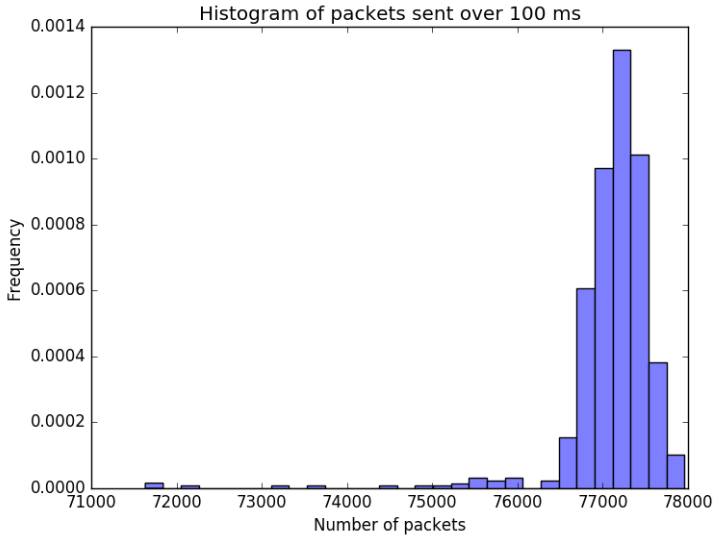

Figure 6.9: Distribution of packets sent over 100 ms for a 5 Gbps Poisson process.

6.2 Simulator maximum rate detection

This section will present the maximum rate of the simulator during different setups. For each setup presented in this section, several simulations have been performed separately for the UL, DL and COMBINED scenarios.

Baseline test

The first step towards detecting the maximum rate is to perform some experimental tests with different setups to determine a baseline. The point is to observe how well the simulator behaves during different circumstances. After some experiments, we have chosen the following configuration for the baseline tests:

Rate	10 000 Mbps
User sessions	1
Flows	100

Table 6.1: Configuration for baseline tests.

These configurations were chosen for simulations in both the UL, DL and COMBINED scenarios. It has some amount of flows, as to not be too easy for the simulator to handle, while not being over what the simulator can handle comfortably in all directions. Furthermore, 10 Gbps has shown to be a rate that is maintained very steadily for all three scenarios. An example of this is represented in Figure 6.10 for the UL scenario. Note that the actual duration of the simulation is 60 seconds. The

reason we present Figure 6.10 with 1 second is to more accurately show that the rate is kept steadily around the specified rate.

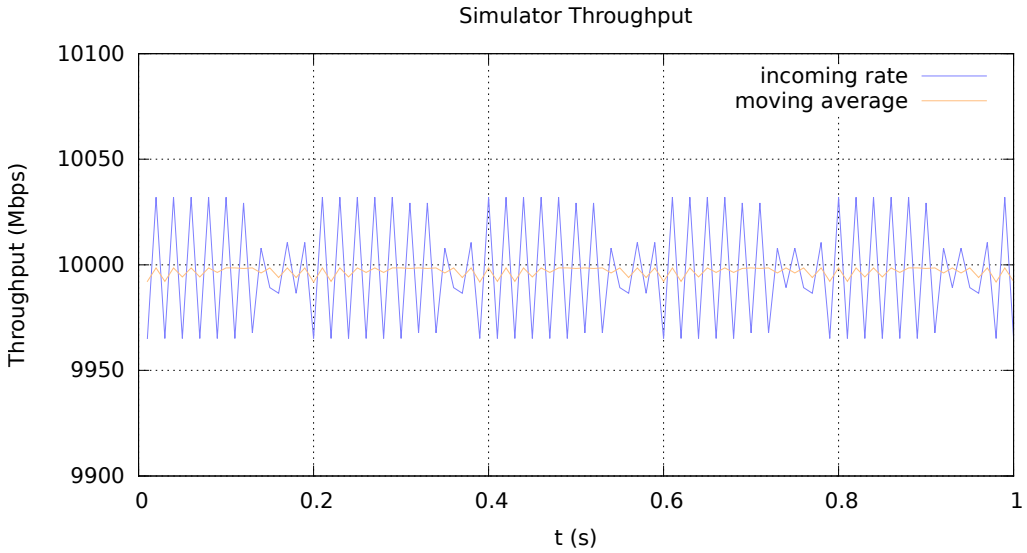

Figure 6.10: Baseline simulation with steady-rate 10 Gbps, zoomed over 1 second.

UL BASELINE	
CL	-0.00135
UCL	0.03876
LCL	-0.04146

DL BASELINE	
CL	-0.00032
UCL	0.03190
LCL	-0.03253

COMBINED BASELINE	
CL	0.00043
UCL	-0.01601
LCL	0.01687

Table 6.2: Tables of the baseline values to use for the simulator max rate detection.

Table 6.2 presents the baseline values that are calculated for each scenario. These values are rounded to the fifth decimal. As previously mentioned, CL is calculated by taking the difference between the expected simulation rate and the actual median rate that the simulator outputs. Therefore, a negative value for CL means that the median rate of the baseline test is higher than expected while a positive value means that the median rate was lower.

Maximum rate for varying number of sessions and flows

With a determined baseline value for every scenario, we were able to search for the maximum rate for different combinations of the number of UE and flows per UE on the respective scenario. This was done using the CtrlCharts technique as described in §5.1 and an accepted violation ratio of 1 %. The simulations performed to find the maximum rate used steady-rate as load type and increased the rate of the simulator with 1 Gbps between each run. Using 1 % as accepted violation ratio might seem restricting but from the performed experiments, it was observed that any ratio between 1-5 % would not have made any significant difference in the result.

However, if a higher violation ratio of, for instance, 10 % was allowed, the measured rate would have increased some of the measured maximum rates with a few Gbps. Though, the throughput at this violation ratio looks noticeably less stable at a glance. An example of this can be seen in Figure 6.11 and 6.12. These figures show the throughput when moving from 23 Gbps to 24 Gbps. Between these rates, the violation ratio increased from 0.1 % to 6 %. Even though the violation ratio only increases to 6 %, one can see that the actual rate deviates frequently from the specified rate. Because of this, we chose to keep the accepted violation ratio at 1 %.

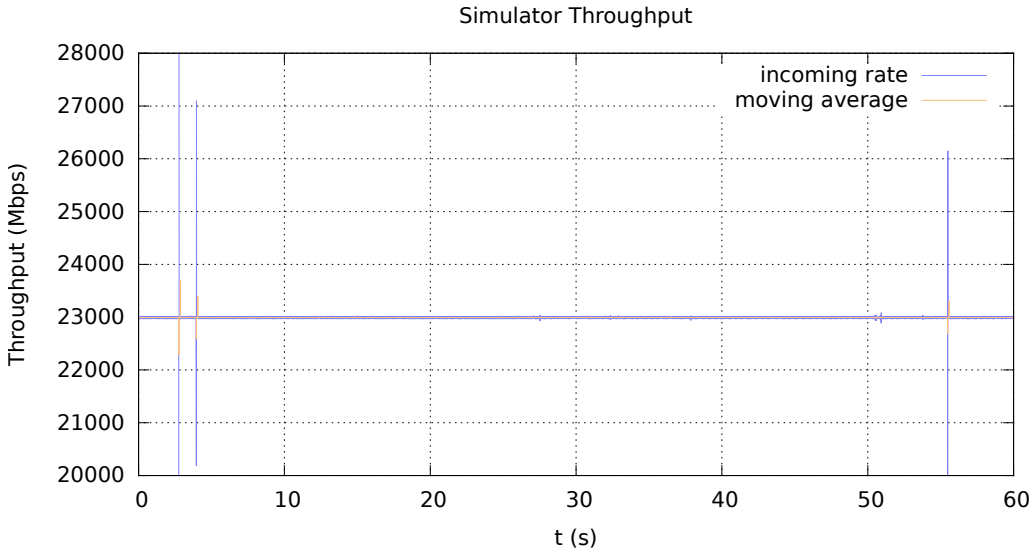

Figure 6.11: Simulator throughput at 23 Gbps for UL scenario.

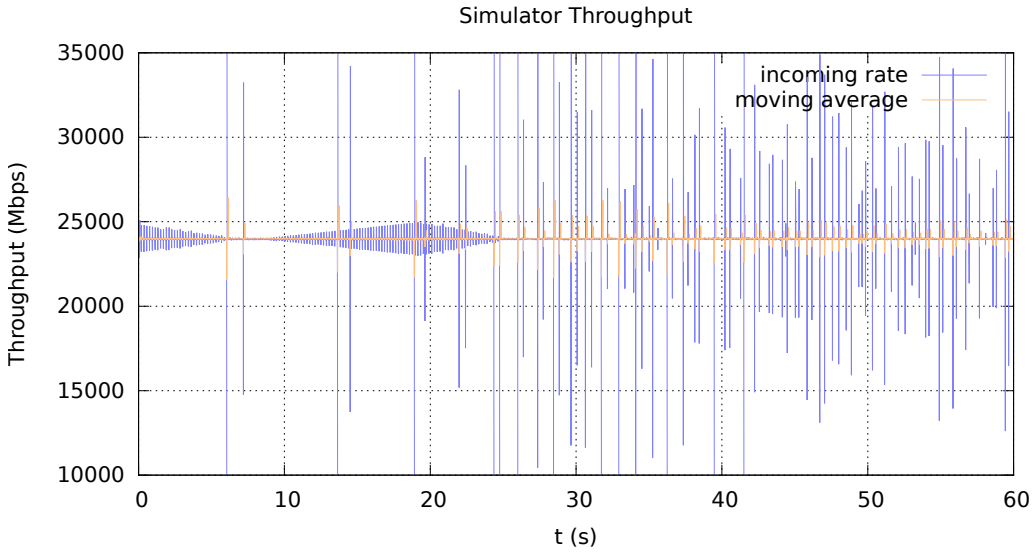

Figure 6.12: Simulator throughput at 24 Gbps for UL scneario.

In Table 6.3, 6.4 and 6.5, the found maximum rates of the simulator for different number of users and flows per user are shown and this is rounded down to the closest

whole Gbps. Each table shows the rates for the respective scenario. Cells in the table denoted with a "-" have not been tested as we confined our experiment to situations with a maximum of 1000 flows all in all.

The values denoted as "∗" means that our CtrlCharts technique was not able to find any rate such that the criteria of 1 % violation ratio were held. However, the simulator was still able to output a rate close to the specified one. One such example can be seen in Figure 6.13 where the specified rate is 9 Gbps, the number of UE is set to 100 and flows per UE is set to 10. As one can see, the moving average of the rate is still very stable at 9 Gbps but the span in which the actual rate is varying has become too large for the CtrlChart's limits to accept. This span for the throughput could be observed on all rates between 0.05 to 9 Gbps and therefore, no maximum rate was able to be reported.

Furthermore, it is important to note that the previously discussed potentially higher accepted violation ratio of 10 % would not allow the missing data points to be found. All rates in these scenarios had a much larger violation ratio, usually between 20-60 %. These missing data points were also the reason we restricted the search to a maximum of 1000 flows.

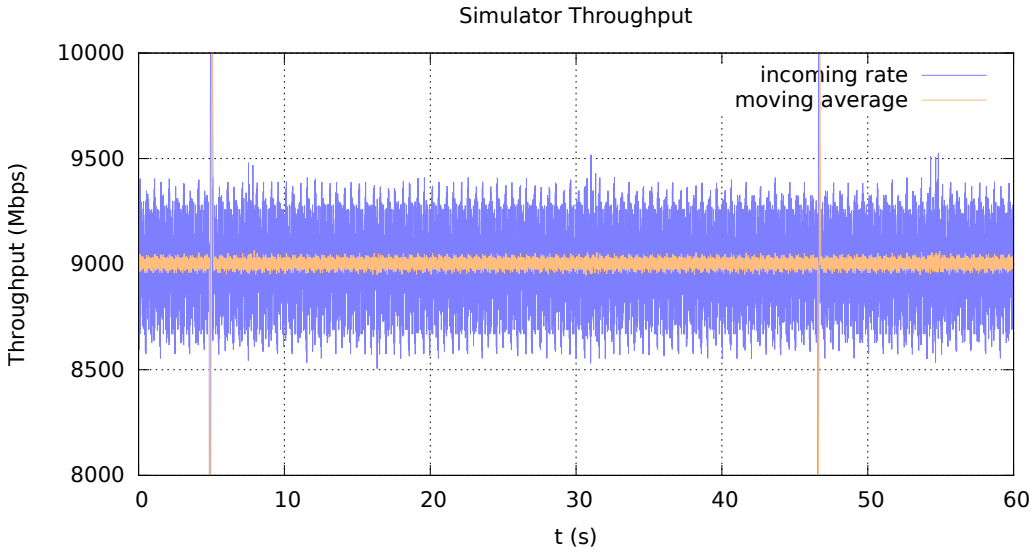

Figure 6.13: Simulator COMBINED throughput with 100 UE and 10 flows per UE.

Number of UE \ Flows per UE	1	10	100	500	1000
1	34	31	24	11	7
10	36	20	7	-	-
100	28	7	-	-	-
500	11	-	-	-	-
1000	7	-	-	-	-

Table 6.3: Table of max rates (Gbps) for the simulator in the DL scenario.

Number of UE \ Flows per UE	1	10	100	500	1000
1	40	33	30	17	12
10	32	23	12	-	-
100	30	12	-	-	-
500	20	-	-	-	-
1000	12	-	-	-	-

Table 6.4: Table of max rates (Gbps) for the simulator in the UL scenario.

Number of UE \ Flows per UE	1	10	100	500	1000
1	38	30	24	12	*
10	29	24	*	-	-
100	26	*	-	-	-
500	12	-	-	-	-
1000	*	-	-	-	-

Table 6.5: Table of max rates (Gbps) for the simulator in the COMBINED scenario.

6.3 UP maximum rate detection

Having determined the maximum rate of the simulator for different scenarios, it becomes possible to load test the UP for the corresponding scenarios. These load tests are used to analyze whether our implemented simulator is fast enough to detect the maximum rates of UP.

The approaches that are taken to determine the maximum rate of the UP for different scenarios follow a similar process. Therefore, in this section, we will start by presenting our approach for only two cases, which together can be seen as representative of all the tests ran. The first one is a DL scenario with step-wise rates, 500 UE and 1 flow per UE, where the maximum rate of the UP is detected. The second case is also a simulation with step-wise rates, 500 UE and 1 flow per UE but this is a COMBINED scenario where the simulator is not able to properly detect the maximum rate of the UP. Finally, at the end of this section, a summary of all performed simulations will be presented in tables, similar to Table 6.3, 6.4 and 6.5 in §6.2.

Case 1

To determine the maximum rate of the UP with 500 UE and 1 flow per UE in DL, we perform a step-wise simulation with increasing rates. We set the duration of the entire test to 20 seconds, set the initial rate of the simulator to 1000 Mbps and increase it with 1000 Mbps every two seconds. This is shown in Figure 6.14. It can be can see that the simulator is able to keep all the specified rates steadily.

Next, we analyze how the UP reacts when the simulator increases its rate every two seconds. This is illustrated in Figure 6.15. In this figure, it can be seen that the UP is able to hold the rate of the simulator until the last two seconds. Here, the simulator is still able to send traffic with 10 000 Mbps as shown in Figure 6.14, however, the UP starts to deviate from the simulator.

Since the specified rate is not held by the UP, it means that the UP starts to drop packets. For that reason, the packet loss in each interval of the step-wise rate is analyzed. Here, we notice that the only interval where packet loss occurs is within the last two seconds with a loss ratio of roughly 6 %. We can also see a correlation between packet loss and UP-latency by analyzing individual packet latency and bucket latency throughout the entire simulation. This is shown in Figure 6.16.

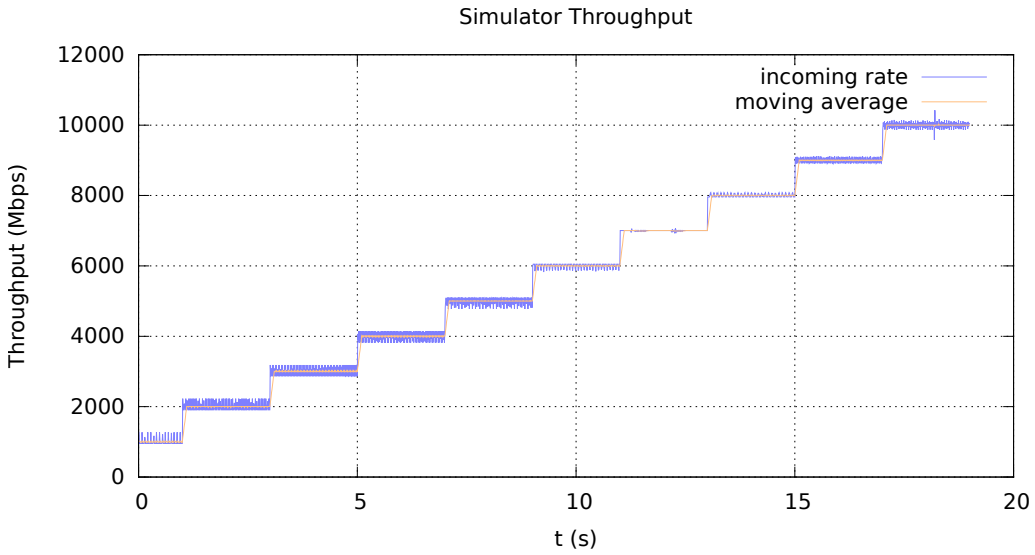

Figure 6.14: Simulator throughput with 500 UE and 1 flow per UE.

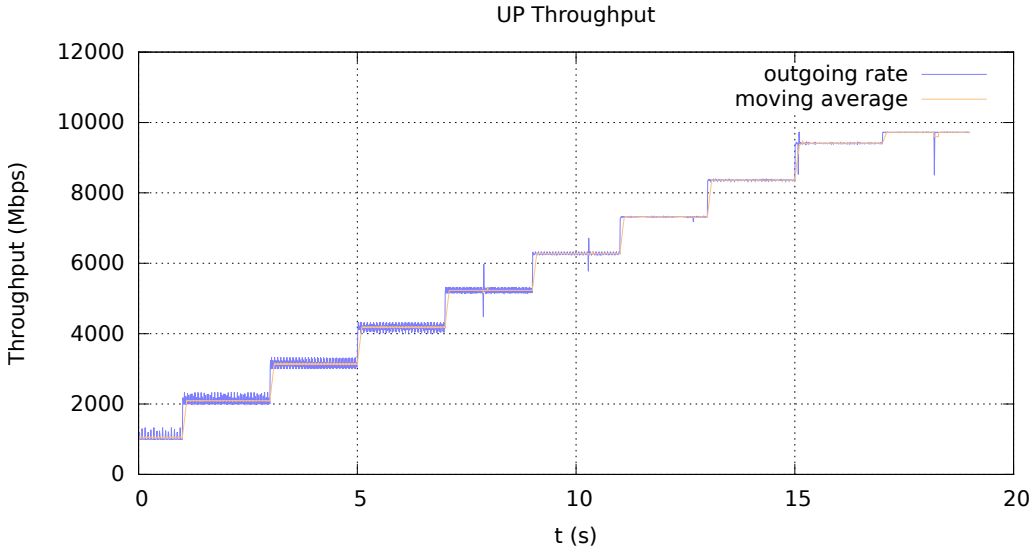

Figure 6.15: UP throughput with 500 UE and 1 flow per UE.

In Figure 6.16, the latency of each packet processed by the UP is represented with blue points whereas the bucket latency is represented with the orange line. It can be seen that both of these latency measurements are kept at a steady and low level (almost zero milliseconds) until the last two seconds of the simulation where the packet loss is introduced, thereby increasing the latency drastically. From this analysis, we state that the maximum rate of the UP, when simulating traffic in the downlink direction with 500 UE and 1 flow per UE, is around 9000 Mbps.

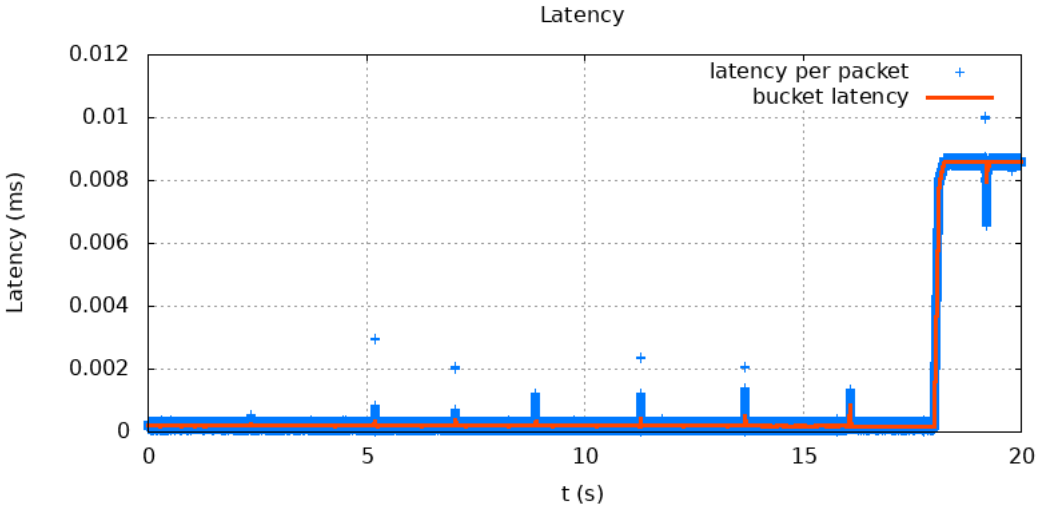

Figure 6.16: UP-latency for downlink step-wise traffic with 500 UE with 1 flow each.

Case 2

The maximum rate of the simulator for 500 UE and 1 flow per UE in the COMBINED scenario is 12 Gbps. In order to determine the maximum rate of the UP for this setup, we perform a step-wise simulation similar to Case 1, but this time, the initial rate of the simulator is set to 5000 Mbps and the rate at the last time interval is 12 000 Mbps.

Furthermore, unlike Case 1, the rate during the last four intervals is increased by 500 Mbps between the intervals. Figure 6.17 shows that the simulator is able to hold the specified rates and analyzing the throughput of the UP shows that the UP always manages to follow the rate at which the simulator sends traffic. This is illustrated by Figure 6.18.

Throughout the entire simulation, the UP manages to receive and process all packets sent by the simulator. As a result, the packet loss ratio is kept at 0 % during the entire test. Furthermore, the latency is kept at a very low and steady level as shown in Figure 6.19.

To summarize Case 2, the UP is load tested up to the maximum rate of the simulator. Throughout the entire test, it manages to follow the rate of the simulator while keeping the packet loss ratio at 0 % and the latency at the same level. This means that the maximum rate of the UP for Case 2 cannot be detected by our simulator. All we can say is that the UP can at least handle 12 000 Mbps.

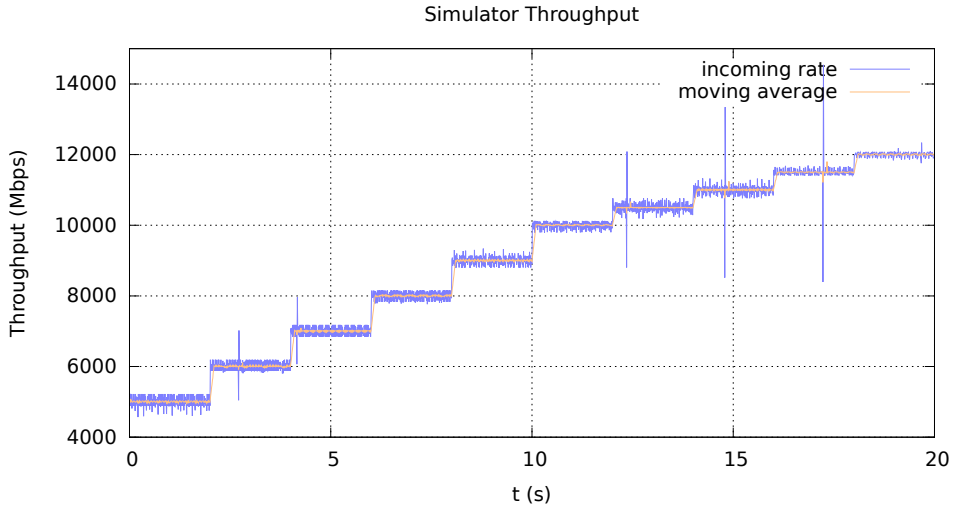

Figure 6.17: Simulator combined throughput with 500 users and 1 flow each.

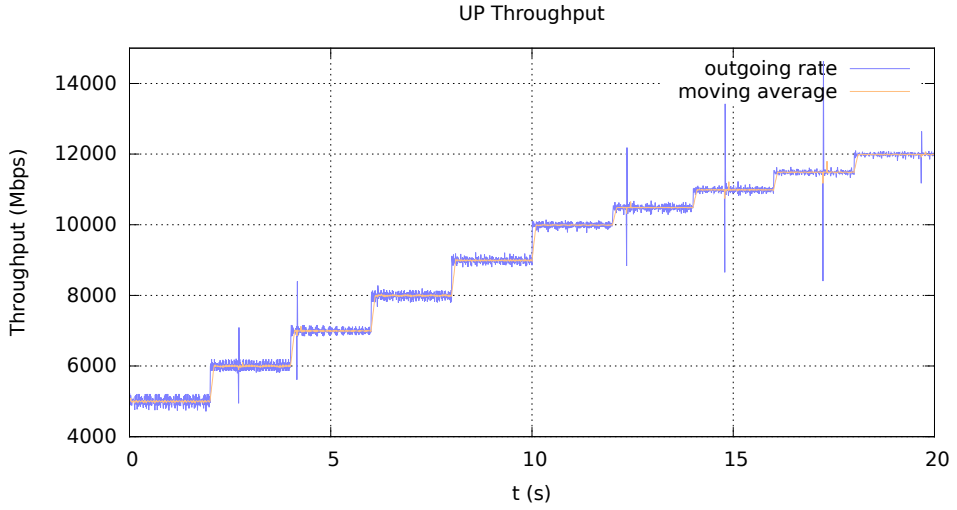

Figure 6.18: UP throughput with 500 users and 1 flow per user in combined direction.

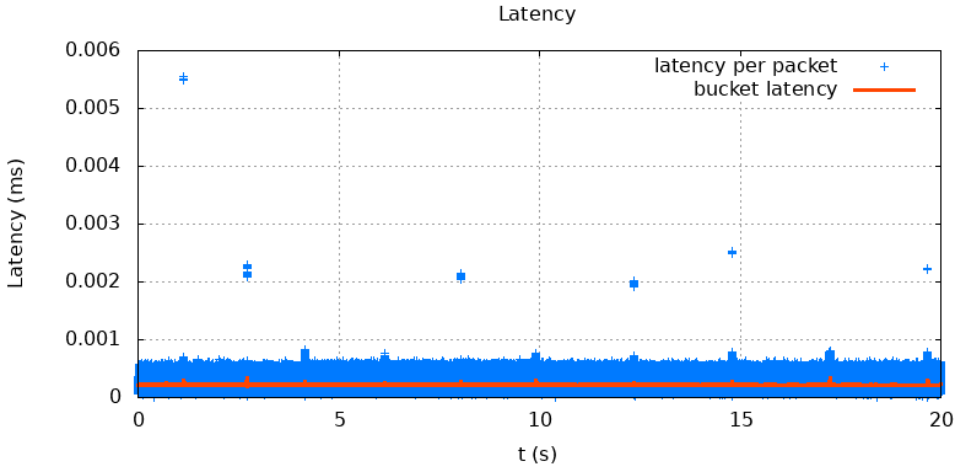

Figure 6.19: UP-latency with 500 users and 1 flow per user in combined direction.

Maximum UP rates for varying number of sessions and flows

The following tables (Table 6.6, 6.7 and 6.8) present the maximum rates for the UP and the average UP-latency when this rate is held. This is presented by *max_rate* | *latency* where *max_rate* is presented in Gbps and *latency* is specified in nanoseconds. The cases where cells are marked with "∗" describes situations where the UP handled our simulators maximum rate without any packet loss or where we did not have any reported maximum rate for our simulator. As with the tables of the simulator maximum rates, the values are presented rounded down to the closest whole Gbps.

Number of UE \ Flows per UE	**1**	**10**	**100**	**500**	**1000**
1	4 | 100	9 | 114	9 | 135	4 | 435	4 | 665
10	9 | 112	9 | 170	8 | 320	-	-
100	9 | 167	8 | 236	-	-	-
500	9 | 250	-	-	-	-
1000	9 | 260	-	-	-	-

Table 6.6: Table of max rates for the UP in the UL scenario.

Number of UE \ Flows per UE	**1**	**10**	**100**	**500**	**1000**
1	4 | 85	9 | 150	5 | 500	4 | 400	4 | 553
10	9 | 150	9 | 220	9 | 300	-	-
100	9 | 150	9 | 345	-	-	-
500	9 | 217	-	-	-	-
1000	9 | 226	-	-	-	-

Table 6.7: Table of max rates for the UP in the DL scenario.

Number of UE \ Flows per UE	**1**	**10**	**100**	**500**	**1000**
1	9 | 275	9 | 137	9 | 210	9 | 385	*
10	10 | 230	9 | 240	*	-	-
100	15 | 510	*	-	-	-
500	*	-	-	-	-
1000	*	-	-	-	-

Table 6.8: Table of max rates for the UP in the COMBINED scenario.

Data points of note from the tables would be the following:

1. The difference between the COMBINED scenario compared with both the UL and the DL scenario at one UE with one flow.
2. The steady maximum rate for both the UL and DL scenario as UE increases, while the COMBINED scenario has an increased maximum rate over the same data points.
3. The difference in the DL scenario from the UL and COMBINED scenario at one UE with 100 flows.
4. The steady maximum rate of the COMBINED scenario as the number of flows increase on one UE, where the UL as well as the DL scenario see a decrease in the maximum rate.

7

Discussion

7.1 Challenges solutions

When presenting our network simulator in §3.3, two main challenges were mentioned in §3.3.2. These challenges were concerned about optimizing the load execution loop and maintaining a consistent network environment.

To address the first challenge, we have implemented an algorithm for load execution with low time complexity. A detailed description of how this works is given in 4.3.2 but to summarize, it only needs to iterate over the total number of flows in order to generate traffic and execute load. This means that the time complexity for the proposed algorithm will always be $O(n)$.

Furthermore, in §3.3.2, it was stated that execution of costly instructions should be avoided. One approach that has been taken to address this challenge was to utilize the number of clock cycle and the frequency of CPU to estimate time. With this approach, it has been possible to avoid system calls and thereby bypass the kernel which would introduce overhead by performing context switches.

To avoid costly instructions connected to sending network traffic, our simulator made use of DPDK as described in 4.3.1. This solution bypasses most of the overhead added by both the Operative System as well as the Docker container.

To solve the challenges regarding the network environment, our simulator follows the PFCP protocol when setting up a user session. Once the user session has been established, a mapping of the TEIDs and IP addresses for each flow is stored. This makes it possible to send traffic in the correct GTP-tunnel whenever sending traffic in an uplink direction.

7.2 Load models

The verification that our load types behave properly is closely related to a more general discussion of their usefulness. The following sections will discuss how well each of the load types fulfill the criteria set up in §3.3.3 using the results presented in the previous chapter as evidence to support these discussions.

7.2.1 Steady-rate

Steady-rate is the most basic of our load types. It is easily configurable and while not being very realistic, it is very useful for load testing where a certain rate needs to be tested. Its usefulness has also been shown by its use for generating the results in this thesis. However, usefulness only exists if it fulfills what it is made to do - keep the rate constant and steady - while having as low overhead as possible.

The results show that the steady-rate load type holds its specified rate very accurately in all three scenarios with approximately 50 Mbps varying up and down from the specified rate. However, during the later experiments in finding the maximum rate, one can observe that the span in which the actual rate varies has increased. This was the reason why some maximum rates were not able to be reported for all of the COMBINED scenarios. The experiments also seemed to show this effect to get worse as the rate was decreased.

We have not been able to precisely point to the reason for this span but it might appear because of how packets are sent on all flows. During runtime, the amount of packets that are needed to be sent is calculated per flow and is also aggregated whenever anything less than a whole packet is calculated. That means that whenever a flow calculates that it needs to send a packet, all flows of the same load type would do exactly the same, resulting in a burst of packets sent at the same time. This would probably increase the sending rate momentarily. When there are more flows than what can be sent in a burst during one step, most flows would be sent in the first burst, and in the next one, only the ones left would be sent. This would result in an initial large burst of packets followed by a smaller one. A piece of evidence that would suggest this to be the case would be that the moving average over the rates always stays very stable, suggesting that the amount of packets sent is not calculated wrongly overall.

To alleviate this problem, one would have to spread out when flows send their packets more evenly over more timesteps. More experiments would have to be conducted to find the best way to do this but a possible solution might be to limit the number of flows that can send packets each loop and in that way, evening out the number of packets sent over a larger time. Nevertheless, this would possibly impact performance and would have to be implemented with caution. With this handled in a better way, our simulator might be able to report maximum rates in the COMBINED scenario where it was not able to.

7.2.2 Step-wise

The step-wise load type is an extension of steady rate and therefore, it does also fulfill the model goals in the same way. This load type was mostly used to gather results on the maximum rate of the UP. While steady-rate has been useful to observe and analyze the behaviour of the simulator with a specific rate, during a longer simulation, the contribution of step-wise rates has made it faster to load test lots of different rate scenarios on the UP with suitable parameters observed from steady-rate simulations.

As the main difference from steady-rate is the ability to change rate at specified times, how well this is performed forms the basis of its usefulness. From the experiments, one can see that the change in rate happens very quickly (usually over less than a millisecond) and the new rate is found and held steadily afterwards.

7.2.3 Poisson processes

Poisson processes fit most of the goals set. At short time spans, the model would, as mentioned in §2.3.3, still be realistic enough for most use cases. In addition, it has very understandable and simple parameters while at the same time needs very little computer resources to compute and generate load.

The problem would arise when using this model for generating larger traffic sets. At a certain point, the model would not be realistic anymore, as self-similarity and LRD (described in 2.3.3) would be present. If realism would still be desired after that point, the user should be asked to switch to a better model. Of course, it could still be the case that realism is not needed for a certain test case.

The results of running simulations with Poisson processes show that the Poisson load type produces a load with a more varied rate compared to steady-rate. Furthermore, the histogram over the distribution of a specified mean-rate varies correctly. One example has been presented in Chapter 6 by Figure 6.9 where the specified mean-rate is shown to be placed correctly over the distribution.

7.3 Scalability

To answer the research question of saturation, the results presented in §6.2 and §6.3 show that you can determine the maximum rate of both the simulator and the UP. Therefore, we consider that question mostly answered but in this section, we will discuss and analyze the results from determining the maximum rate of both the simulator and UP. It will cover what these results can say about the two components' behaviour.

Simulator scalability

In the tables presenting the measured maximum rate of the simulator, one can see that the simulator is able to output a very high rate on low amounts of UE and flows. In fact, when combining this with the maximum rates found for UP, one can determine that the simulator is, in these situations, able to generate traffic at a rate above of what the UP can handle.

However, the tables do also clearly show this high maximum rate decreases when additional flows are added. The fact that the performance is strongly connected to the number of flows is something we already knew from the implementation (since the simulator iterates over the total number of flows) but is also very obvious from these results.

Another observation possible from the simulator maximum rate tables is that the maximum rate in most cases is higher in the UL scenario than in the DL scenario. At first glance, we accredited this to the difference in the size of packets because of encapsulation. Since the packets sent from the simulator in the UL scenario is larger, it requires fewer packets sent each timestep to achieve the given rate.

However, if one calculates what the maximum rates in the DL scenario would be with the larger packet size (by multiplying each value with the ratio between the packet sizes), the mean difference between the measured max rates in either direction decreases from 4.17 to 3.23. This would imply that, while the packet size might account for some of the difference, it might not be the only contributing factor. Our best guess for where this additional increase in maximum rate comes from is that the decreased number of packets needed to be sent allows for better performance aside from the natural increase in throughput.

UP Scalability

The maximum rate of the UP in all scenarios shows clearly that the UP scales very well with the number of users. We were not able to find any decrease in the maximum rate for UP as the number of users increased. Additionally, in the COMBINED scenario, the maximum rate of the UP even increased with the number of users. This behaviour is intriguing. It is clear that UP is very good at handling traffic from different users optimally, but it also seems very adapted to handling traffic on the same user but in different directions.

This conclusion can also be drawn from the increased maximum rate for UP at a single UE with a single flow. In this situation, both the UL and DL scenario have decreased performance while the combined seem to handle it as well as in most other situations, seemingly able to handle the traffic better since it is split into both directions. Furthermore, the fact that the single user - single flow case lowers the maximum rate shows that the strength of UP is not in giving a high rate to a single user, but in handling a large number of users without performance impact.

Another data point of note from the tables is the way the maximum rate of the DL scenario diverges from the other scenarios in the case of one UE and 100 flows. This is quite remarkable. Especially since the same decrease cannot be seen in any other comparable cells where the same or more flows should have been. This data point remains from our view hard to explain.

A decreased maximum rate can also be seen in the UL and DL scenario as the number of flows increase while once again the COMBINED scenario does not show signs of changing performance. This would point to UP not handling flows the same way as users, leading to this impacting performance more than an increase in UE. One can infer that traffic from different flows cannot be split up as effectively as traffic from different UE.

Lastly, one can note from our experiments that as the UP reaches its maximum rate, its actual output rate still increases, as we continue to increase the incoming rate

from our simulator. One could hypothesize that this increase would continue further in connection with increasing the incoming rate until finally, a *plateau rate* is reached. This new maximum rate could be measured as another evaluation metric. However, we concluded to not research this plateau rate further as the amount of packet loss present for rates above the maximum would only become higher. Therefore, investigating the plateau rate would not be of interest since the UP would not be considered usable with that amount of packet loss.

7.4 Requirement comparisons

This section will provide a small discussion on how well our simulator and the UP in a 5G environment are able to handle the requirements for different applications mentioned in §3.1. We mentioned that a measurement in throughput of over 100 Mbps is considered high. Looking at the results provided for the maximum rate of our simulator and the UP in §6.2 and §6.3 respectively, we can say that these requirements are fulfilled for at least 1000 flows in total. The lowest rate we obtain for our simulator is 7000 Mbps and the lowest rate for the UP is 4000 Mbps.

In §1.2, it was stated that an investigation on whether a noticeable improvement in 5G can be seen compared to 4G will be performed. For that purpose, we used the UP-latency in a 4G environment as a base. It was stated in §3.1 that this latency was estimated to 4.8 ms. Looking at the UP-latency results provided in Table 6.6, 6.7 and 6.8, we see that the UP-latency never becomes larger than 4.8 ms. The highest UP-latency seen in these tables is 665 nanoseconds which is significantly below 4.8 ms.

7.5 Suggested future work

This section will propose some suggestions for future work to enhance the performance and possibilities of our simulator. Furthermore, additional load models that for instance circumvent the non-self-similarity of Poisson processes will be suggested.

Varying payload packet length

In reality, it is not necessarily true that the payload length of all packets is of the same size. Therefore, in order to better mimic reality, one suggestion is to implement support for varying length in payload when traffic is simulated. Right now, the payload length of packets is specified in the configuration file as a single value.

One approach to support varying lengths would be to allow a user to specify several values as a sequence. Say for example that the simulation is intended to observe traffic over HTTP, FTP and SMTP. Then, a user could specify the payload lengths as a sequence of three values and when connections are made between UE and PDNs, the program could automatically choose the payload length depending on if the PDN that a UE is communicating with is a HTTP, FTP or SMTP server.

Support for unbounded-length simulations

In addition to supporting varying payload lengths, one improvement is to associate sending and receiving timestamps with the payload. This is for example needed when calculating latency. Right now, the payload of packets does only consist of a single integer value denoting the identifier of the corresponding packet. One advantage of this is that look-up of packets can be performed quickly since packets are stored in an array and the index matches with the identifier. However, one consequence of this approach is that when a simulation is run with extreme rates and over a very long time period, it would not be possible to allocate an array that is large enough to store all the packets.

By sending the required timestamp along with the payload, it would be possible to avoid the allocation of arrays and instead determine the latency from the timestamps that each packet is carrying as payload information. Nevertheless, this would mean that no information about packets during a simulation is stored. One way to overcome this would probably be to write information to a file during runtime but that would be costly and lead to a slower simulation. Furthermore, depending on the implementation, this could also lead to a decreased precision as measurements for each packet would have to be summarized to be saved until each print.

Improved network protocols

In the context of this thesis, the network environment is assumed to have GTP-tunnel at only one side of the UP. This is between the LTE and the EPC network. However, in reality, there could exist situations where GTP-tunnels do also exist between the EPC and PDN. This could be simulated in a similar manner to how the current GTP-tunnel is simulated. The difference would be that when a PDN sends packets in the downlink direction, it would have to be encapsulated with GTP header as well and when a UE sends packets, the UP would need to encapsulate the packet with GTP header before sending it out towards the correct PDN.

Another improvement regarding network protocols is to more properly check the control messages that are sent by PFCP. When a user-session is about to be established, the UP and CP communicate with each other through PFCP. This communication consists of a request and a reply message containing information about the session e.g., whether the session is successfully established or not. The current implementation does only check if these messages went through correctly between the CP and the UP. An improvement would be to properly decode the control messages in order to check the exact status such as “ACCEPT” or “REJECT” and take actions accordingly.

A final suggestion on improvement regarding network protocols is to make the simulator behave as defined by ARP. Since all parts of the network except the UP is simulated, the simulator has a mapping for all MAC and IP addresses for all network devices. However, when an ARP request is sent by the UP, all devices that are part of the simulation send replies, even though the IP address of a certain device does not match with the IP address in the request. Moreover, there can exist cases

when the ARP request does not contain any of the IP addresses maintained by the simulator but a reply is performed anyways. Even though this is not a problem in our environment, it might give rise to ARP spoofing in a more open network.

Implementation of additional load models

Right now, the implementation of Poisson processes requires specifying a mean rate during a certain time interval. The program could be extended to support reading a sequence of mean rates from the configuration file where each rate is associated with a time interval. This would make it possible to perform simulations like step-wise rates but as Poisson processes instead, thus providing further realism to the step-wise traffic.

A problem with Poisson processes already discussed in this thesis is the non-realistic nature of this kind of traffic once the simulation runs long enough for self-similarity and LRD to play a part. This presents an obvious possibility for extensions and further load types. A possibility that is already discussed previously is MMPP.

An implementation of MMPP would not be a very large extension of the Poisson processes. The only thing needed would be to implement the way the model would switch between Markov-states. However, to provide a proper solution to self-similarity, the MMPP implementation would have to handle multiple states, somewhat complicating the implementation. Furthermore, to provide a good user experience for using the MMPP model, some kind of help tool for calculating the parameters used for the model might have to be implemented.

Support for multithreading

As already mentioned in §3.3.3, when discussing the usefulness of the steady-rate load, the measured maximum rate of the simulator has been shown to be well above what is needed to load test UP on low amounts of UE and flows per UE. However, this is not the case when these parameters increase. This observation implies that future improvements of the simulator should focus on finding ways for the simulator to split up the number of flows and make it possible to work and send traffic on them in parallel.

Following this intuition would paint the outlines of a strategy for how to introduce multithreading into the program in the future. Each thread could be made to handle a part of all the flows available and only send packets needed for those flows. The problem would be to synchronize the handling of received packets and measured statistics. Due to how the libraries used for sending and receiving traffic work, each thread would have to handle receiving packets and in a safe way communicate this data in whatever way the program chooses to output to a user. To accomplish this, several implementation possibilities exist and it is up to the programmer to decide on which approach is the better one. Inspiration could for example be taken from known distributed monitoring algorithms [11].

Further evaluation techniques

It would be interesting to perform a more in-depth and complete analysis of the UP performance and behaviour. For example the Quality of Service (QoS) maintained by the UP could have been investigated by e.g., measuring trade-offs between high throughput and latency for sending traffic through the UP.

To achieve this, further analysis techniques and methods would have to be developed or applied. In addition to CtrlCharts, evaluation techniques such as DescStats (described in §2.3.1) could be used.

Another analysis technique that could be applied is a metric for packet loss. The simulator can already output packet loss but it is not presented in any of the result in this thesis. We had the possibility to sample packet loss as a part of our experiments. However, in all of our simulations, we could observe a correlation between latency and packet loss or between throughput and packet loss. This is reasonable, since if the UP is not keeping up with the throughput sent to it, this has to show up as either increment in latency in outgoing packets, packet loss or both as its packet queues would fill up. This is the reason why measurements and investigation on packet loss has been omitted from this thesis.

However, packet loss as a metric would be incredibly useful in testing scenarios where a correlation between the throughput, latency and packet loss could not be observed. In these scenarios, the simulator could provide sample points of the packet loss during timing windows of a given length, thereby pinpointing the exact moment faults appeared.

8

Conclusions

During this thesis, we have implemented a network simulator that is able to generate 5G communication in a client-to-server-architecture. In this network architecture, clients (UE), servers (PDNs), Control Plane (CP) and other intermediate routers are a part of the simulation. The only part of the network that is physical is the User Plane (UP) and the goal of the implemented simulator is to simulate 5G traffic in order to load-test the UP and analyze its performance.

In order to reach this goal, several features were implemented. First of all, to generate 5G traffic that goes through the UP, support for bidirectional communication and multiple user sessions were implemented and this is described in §4.3.2.

To model the communication, we have implemented three different load types: *steady-rates*, *step-wise rates* and *Poisson processes*. The first load type tries to keep a specified rate as steadily as possible during a given time interval. The second load type varies the rate during different time intervals and the third load type follows a Poisson process and varies the specified rate randomly in order to behave stochastic. The implementation of these load types are described in §4.2.

When a simulation is run, traffic is generated using these load types. The rate at which the simulation should run, the duration of the simulation and other necessary parameters such IP addresses for UP, CP and intermediate routers are specified in a configuration file which is read by our program when it starts to execute. Furthermore, in order to be able to cope with the strict requirements of 5G, low-level networking API is utilized to bypass the kernel and maintain a low simulation overhead. The implementation of this is described in 4.3.1.

During a simulation, information about all packets that are sent and received are stored. This information does, among other values, include the sending and receiving timestamp of each packet. This makes it possible to evaluate all simulations and thereby analyze the behaviour of our simulator as well as the performance of the UP. For example, a method known as CtrlCharts is utilized to validate the performance of the simulator. We do also measure throughput, latency and packet loss to analyze the UP. A detailed description of how the UP is analyzed and how the simulations are evaluated is given in §5.

To show that the required properties are fulfilled and to answer the research ques-

tions set up in §1.2, several experiments have been carried out. First, in Chapter 6, we showed that the implemented load types behave as expected. After that, the maximum rate of both the simulator and UP in different scenarios were measured. This is presented in §6.2 and §6.3 respectively. Using these results, we were able to discuss as well as analyze the behaviour of both components and this is presented in §7.3.

From all the performed experiments, we conclude that our simulator performs above the required rate to load test the UP for most scenarios with differing number of UE and flows per UE. There are some cases where a maximum rate for the UP cannot be determined according to the CtrlCharts technique. Some ideas and suggestions on how the simulator can be enhanced to cover these cases have been given in §7.5. We can also conclude that UP scales very well with an increased number of users and that it performs above proposed 5G requirements.

Bibliography

[1] 3GPP. Ts22.261 - 5g; service requirements for next generation new services and markets. https://www.etsi.org/deliver/etsi_ts/122200_122299/122261/15.05.00_60/ts_122261v150500p.pdf, 2019. Last accessed 27 May 2019.

[2] Jin Cao, William S Cleveland, Dong Lin, and Don X Sun. On the nonstationarity of internet traffic. In *ACM SIGMETRICS Performance Evaluation Review*, volume 29, pages 102–112. ACM, 2001.

[3] Siyoung Choi, Junghwan Song, Junseok Kim, Suhun Lim, Sunghyun Choi, Ted Taekyoung Kwon, and Saewoong Bahk. 5g k-simnet: End-to-end performance evaluation of 5g cellular systems. In *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2019.

[4] CNCF Cloud Native Computing Foundation. Definition of cloud native. https://github.com/cncf/toc/blob/master/DEFINITION.md, 2018. Last accessed 7 February 2019.

[5] CNCF Cloud Native Computing Foundation. Homepage. https://www.cncf.io/, 2018. Last accessed 7 February 2018.

[6] Aaron Yi Ding and Marijn Janssen. 5g applications: Requirements, challenges, and outlook. *arXiv preprint arXiv:1810.06057*, 2018.

[7] Inc Docker. Apache 2.0 license. https://github.com/moby/moby/blob/master/LICENSE, 2019. Last accessed 2 May 2019.

[8] Inc Docker. Components & licenses. https://www.docker.com/legal/components-licenses, 2019. Last accessed 2 May 2019.

[9] Inc Docker. Docker software end user license agreement. https://www.docker.com/legal/docker-software-end-user-license-agreement, 2019. Last accessed 2 May 2019.

[10] Marcin Dryjanski. 5g core network functions. https://www.grandmetric.com/2018/03/02/5g-core-network-functions/, 2018. Last accessed 4th Februari 2018.

[11] Romaric Duvignau, Marina Papatriantafilou, Konstantinos Peratinos, Eric Nordström, and Patrik Nyman. Continuous distributed monitoring in the evolved packet core. In *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, pages 187–192, 2019.

[12] Ruoyu Gao. An empirical assessment on the techniques used in load testing, 2016.

[13] Ruoyu Gao, Zhen Ming Jiang, Cornel Barna, and Marin Litoiu. A framework to evaluate the effectiveness of different load testing analysis techniques. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 22–32. IEEE, 2016.

[14] Harry Heffes and David Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE Journal on selected areas in communications*, 4(6):856–868, 1986.

[15] Helmut Hlavacs, Gabriele Kotsis, and Christine Steinkellner. Traffic source modeling. *Technical Report No. TR-99101*, 1999.

[16] Frank Huebner, Kathleen Meier-Hellstern, and Paul Reeser. Performance testing for ip services and systems. In *Performance Engineering*, pages 283–299. Springer, 2001.

[17] GSMA Intelligence. Understanding 5g: Perspectives on future technological advancements in mobile. *White paper*, pages 1–26, 2014.

[18] Zhen Ming Jiang and Ahmed E Hassan. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering*, 41(11):1091–1118, 2015.

[19] Thomas Karagiannis, Mart Molle, and Michalis Faloutsos. Long-range dependence ten years of internet traffic modeling. *IEEE internet computing*, 8(5):57–64, 2004.

[20] Will E Leland, Walter Willinger, Murad S Taqqu, and Daniel V Wilson. On the self-similar nature of ethernet traffic. *ACM SIGCOMM computer communication review*, 25(1):202–213, 1995.

[21] Patrick McQuighan. Simulating the poisson process. Department of Mathematics-University of Chicago, Volume 23, 2010.

[22] Marco Mezzavilla, Sourjya Dutta, Menglei Zhang, Mustafa Riza Akdeniz, and Sundeep Rangan. 5g mmwave module for the ns-3 network simulator. In *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 283–290. ACM, 2015.

[23] J Susan Milton and Jesse C Arnold. *Schaum's Outline of Introduction to Probability & Statistics: Principles & Applications for Engineering & the Computing Sciences*. McGraw-Hill Higher Education, 1994.

[24] Marco Miozzo, Nikolaos Bartzoudis, Manuel Requena, Oriol Font-Bach, Pavel Harbanau, David López-Bueno, Miquel Payaró, and Josep Mangues. Sdr and nfv extensions in the ns-3 lte module for 5g rapid prototyping. In *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2018.

[25] Martin Klaus Müller, Fjolla Ademaj, Thomas Dittrich, Agnes Fastenbauer, Blanca Ramos Elbal, Armand Nabavi, Lukas Nagel, Stefan Schwarz, and Markus Rupp. Flexible multi-node simulation of cellular mobile communications: the vienna 5g system level simulator. *EURASIP Journal on Wireless Communications and Networking*, 2018(1):227, 2018.

[26] Magnus Olsson and Catherine Mulligan. *EPC and 4G packet networks: driving the mobile broadband revolution*. Academic Press, 2012.

[27] Jason Orlosky, Kiyoshi Kiyokawa, and Haruo Takemura. Virtual and augmented reality on the 5g highway. *Journal of Information Processing*, 25:133–141, 2017.

[28] Vern Paxson and Sally Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on networking*, 3(3):226–244, 1995.

[29] Stefan Pratschner, Bashar Tahir, Ljiljana Marijanovic, Mariam Mussbah, Kiril Kirev, Ronald Nissel, Stefan Schwarz, and Markus Rupp. Versatile mobile communications simulation: The vienna 5g link level simulator. *EURASIP Journal on Wireless Communications and Networking*, 2018(1):226, 2018.

[30] Zafar Ayyub Qazi, Melvin Walls, Aurojit Panda, Vyas Sekar, Sylvia Ratnasamy, and Scott Shenker. A high performance packet core for next generation cellular networks. In *Proc. of the Conf. of the ACM Special Interest Group on Data Communication*, pages 348–361. ACM, 2017.

[31] Ali Rajabi and Johnny W Wong. Mmpp characterization of web application traffic. In *2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 107–114. IEEE, 2012.

[32] Paulo Salvador, Rui Valadas, and António Pacheco. Multiscale fitting procedure using markov modulated poisson processes. *Telecommunication systems*, 23(1-2):123–148, 2003.

[33] Rath Vannithamby and Shilpa Talwar. *Towards 5G: Applications, requirements and candidate technologies*. John Wiley & Sons, 2017.

[34] Pál Varga and Péter Olaszi. Lte core network testing using generated traffic based on models from real-life data. In *2013 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, 2013.

[35] Elaine J Weyuker and Filippos I Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *IEEE transactions on software engineering*, 26(12):1147–1156, 2000.

[36] Michael Wilson. A historical view of network traffic models. *Unpublished survey paper. See* *https://www.cse.wustl.edu/~jain/cse567-06/traffic_models2.htm*, 2006.

[37] YAML. Official yaml specification. `https://yaml.org/spec/1.2/spec.html`, 2018. Last accessed 4 March 2018.

[38] M Yuksel, B Sikdar, KS Vastola, and B Szymanski. Workload generation for ns simulations of wide area networks and the internet. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, pages 93–98, 2000.

List of Figures

List of Abbreviations

ARP	Address Resolution Protocol
CL	Control Line
CP	Control Plane
CPF	Control Plane Function
CUPS	Control Plane and User Plane Separation
CtrlCharts	Control Charts
DescStats	Descriptive Statistics
DPDK	Data Plane Development Kit
eNodeB	Evolved Node B
EPC	Evolved Packet Core
EPG	Evolved Packet Gateway
EPS	Evolved Packet System
GTP	General Packet Radio Service Tunneling Protocol
LCL	Lower Control Limit
LRD	Long Range Dependency
MME	Moblity Management Entity
MMPP	Markov Modulated Poisson Processes
PDN	Packet Data Networks
PFCP	Packet Forwarding Control Protocol
PGW	Packet Data Network Gateway

RTT	Round Trip Time
SGW	Serving Gateway
SMF	Session Management Function
TEID	Tunnel Endpoint Identifier
UCL	Upper Control Limit
UE	User Equipment
UP	User Plane
UPF	User Plane Function
YAML	YAML Ain't Markup Language