Dynamic bandwidth control for improving 5G network resource utilization in Cloud Radio Access Networks

August Lidfeldt, Daniel Isaksson

ISSN 1650-2884
LU-CS-EX 2021-36

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-36

Dynamic bandwidth control for improving 5G network resource utilization in Cloud Radio Access Networks

August Lidfeldt, Daniel Isaksson

Dynamic bandwidth control for improving 5G network resource utilization in Cloud Radio Access Networks

August Lidfeldt
au4322li-s@student.lu.se

Daniel Isaksson
mte15dis@student.lu.se

June 14, 2021

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Erik Walles, erik.walles@ericsson.com
Markus Borg, markus.borg@cs.lth.se

Examiner: Per Andersson, per.andersson@cs.lth.se

Abstract

The fifth generation of mobile communications will undoubtedly change the way society uses and interacts with technology, offering increased speed, reliability and efficiency. However, several important capabilities of the next generation of mobile connectivity must be developed before this is achievable. One of these capabilities is efficient network resource utilization. If telecommunication companies could improve bandwidth utilization they could potentially sell more connections, save energy or reduce the needed infrastructure. In this thesis we have built a simulation environment to explore dynamic bandwidth allocation in the 5G transport network. The goal of the exploration was to create a more intelligent solution for allocating bandwidth, as compared to the overprovisioning common in telecommunications today. The final simulation environment combines existing cloud technologies such as Kubernetes, Docker and Open vSwitch. In addition, it contains a software defined networking controller with a proof-of-concept algorithm for optimizing bandwidth usage and prioritizing between traffic classes. Our simulation results show potential for dynamic bandwidth allocation, with bandwidth savings of up to 56%. Small and slow demand changes could be handled by the software defined networking controller without considerable packet loss, while larger accelerations resulted in increasing packet loss. Most bandwidth could be saved when traffic classes with different priority peaked at different times. If future research could prove our solution effective on real traffic patterns and implement it at scale, the days of static allocation might be over.

Keywords: 5G, Cloud RAN, transport network, software defined networking, dynamic bandwidth control, Kubernetes

Acknowledgements

Throughout this thesis, a lot of people have contributed and helped us achieve what is presented in this final paper. We would like to take this opportunity to express our gratitude and direct a special thanks to those people.

We would first of all like to thank our supervisor at the Department of Computer Science, Dr. Markus Borg. Your support, clear-sightedness and guidance enabled us to stay on the right track even when the way forward was uncertain. We could not have wished for a better mentor during this project.

We would also like to thank our supervisor at Ericsson, Erik Walles, for his great support, insightful tutorials, and for coming up with the idea for this master thesis. Without your expertise we would have been lost in the jungle of telecommunication. We are grateful for the opportunity to write this thesis at Ericsson with you as our supervisor. Furthermore, we would also like to thank Thomas Edwall for believing in us and bringing us on board. Also a special thanks to everyone else at Ericsson who have shared their knowledge and helped make our project better.

Finally, a big thank you to our friends and family for always supporting us. Specifically during this thesis but also during our studies in Lund as a whole.

Lund, June 2021

Contents

Abbreviations

3GPP 3rd Generation Partnership Project

API Application Programming Interface

BBU Baseband Unit

CBC Coin-or Branch and Cut

CI/CD Continuous Integration/Continuous Deployment

Cloud RAN Cloud Radio Access Network

CNCF Cloud Native Computing Foundation

CNF Cloud-native Network Function

COTS Commercial-Off-The-Shelf

CPU Central Processing Unit

CU Centralized Unit

DevOps Development and Operations

DU Distributed Unit

eMBB Enhanced Mobile Broadband

IoT Internet of Things

JSON JavaScript Object Notation

LTE Long-Term Evolution

mMTC Massive Machine Type Communications

NF Network Function

O-RAN Open Radio Access Network

OPEX Operating Expenses

OS Operating System

QoS Quality of Service

RAN Radio Access Network

RBAC Role-Based Access Control

REST Representational State Transfer

SDN Software Defined Networking

SDNc agent Software Defined Networking controller agent

UE User Equipment

URLLC Ultra Reliable Low Latency Communications

vCU virtual Centralized Unit

vDU virtual Distributed Unit

vLAN virtual Local Area Networks

VM Virtual Machine

Chapter 1
Introduction

The fifth generation of mobile communications will undoubtedly change the way we use and interact with technology. As a result of substantial technological improvements since the last generation of 4G networks, 5G will provide features such as ultra-high-speed connectivity at ultra-low latency. This will make it possible to stream HD videos on a high-speed train traveling at 500 km/h or reliably controlling a fleet of self-driving cars going down the highway [2]. Before this is achievable, several important capabilities of the next generation of mobile connectivity must be developed in order for the technology to deliver on its promises. These capabilities include efficiency, flexibility, reliability, scalability, and automation. Each vendor's ability to deliver on these features will affect their competitiveness in the new 5G market.

For each capability, strategies have been proposed to implement it, largely building on successful technological innovations from the software industry. In order to achieve high-efficiency cloud computing, hardware centralization has been envisioned. High flexibility could be achieved through the use of Commercial-off-the-shelf (COTS) servers, as opposed to custom telecommunication hardware. This would allow for easier maintenance and remote-upgrading of units. Furthermore, the same server hardware could be used regardless of 4G, 5G, or 6G functionality. Finally, reliability, scalability, and automation could be achieved via microservice architectures, where larger applications are divided into small subcomponents [16]. These subcomponents are made available through a virtualization technique called containerization, which will be a central theme throughout this thesis.

The concepts and technologies associated with cloud, COTS, and container orchestration have all matured since the last generation of mobile communication networks. For cloud technologies, the growth and development have been largely fueled by the cloud computing and data center boom, spearheaded by companies like Amazon, Google, and Microsoft [63]. The possibility to substitute custom specialized hardware with COTS has been made possible by large performance improvements for processors and other computational hardware. Lastly, the increase in the use of containerization can be attributed to the larger adoption of systems such as the container orchestration framework Kubernetes and the container run-

time Docker [13].

An important concept regarding the new 5G technologies is their increased reliance on cloud technologies. As the telecommunication industry is taking a step into the world of cloud computing, the industry has to decide which cloud features to implement, but also how these should be adapted to the telecommunication context. The integration of traditional telecommunication radio access networks (RAN) with the above-mentioned cloud technologies, has been coined Cloud RAN. Another concept that is growing in popularity is software-defined networking (SDN). SDN is a network management approach where the network infrastructure is dynamically and programmatically managed [45]. Using this approach infrastructure management can be performed with more flexibility and automation.

In this thesis we will explore how software-defined networking and cloud technologies can be deployed in a 5G context, concretely focusing on building a simulation environment for evaluating network resource management through an SDN controller. Specifically this thesis studies how network resource utilization can be improved compared to current solutions. The goal is to build the environment with technologies and concepts that have shown promise in research and telecommunication development. This includes selecting components that are scalable, flexible, hardware-agnostic, and open-source in nature. The hope is that such a simulation environment can help explore and find the incremental improvements that will shape the next generation of telecommunications. Only by drawing on the lessons learned from the success stories of cloud computing and software development can the potential of the next generation be unlocked.

1.1 Structure of Thesis

Background In the background the foundational knowledge needed to understand this thesis will be presented. This includes the basics of telecommunication and its evolution up to this day. After the basics have been covered the focus will shift to 5G, including use cases and associated technologies such as cloud computing and software defined networking. Lastly, important tools used during this thesis will be detailed.

This thesis work has two distinct parts: developing the simulation environment, and testing it. As these parts both have separate methodologies and results, they will be presented as such. The two parts are called Creation of the Simulation environment and System Assessment.

Creation of the Simulation environment This chapter starts off by presenting the methodology used to develop the simulation environment. Then three distinct development phases are introduced to the reader, under which the design choices are detailed and evaluated. Finally the resulting environment is presented and the overall functionality explained.

System assessment In this chapter the methodology for testing the simulation environment will be detailed. This includes an explanation of the experiment process including the scoping, planning, and operation. Thereafter, the experiment results will be presented. This includes the resulting use cases and their corresponding traffic scenarios, as well as the system's performance during these traffic scenarios.

Discussion In the discussion the results from both Creation of the Simulation environment and System Assessment are discussed in relation to their respective research questions. This includes implementation and design choices, as well as implications of experiment results. Thereafter, threats to validity are discussed, covering both external and internal validity. Lastly, future work is outlined, including areas in need of exploration, as well as a few interesting future directions.

Conclusion This chapter summarizes the main findings of the thesis and answers the research questions in a clear and concise way. The chapter concludes with implications of the results and reflects on the work of this thesis in the larger context of telecommunications.

1.2 Thesis Motivation

During the last decade of the cloud computing boom, a lot of effort has gone into making data centers run more efficiently, optimizing everything from resource utilization and data security to energy efficiency and cooling systems [68]. Some companies have even gone as far as moving their entire data center closer to the arctic circle in order to cut cooling costs [28]. The operations in data centers are often computationally intensive, and therefore attractive subjects of optimization. Approaches to improve operational performance have included using virtual machines and locating these to utilize the available hardware resources to the highest degree possible [75].

Common resource types in cloud computing are CPU, memory, disk, and network resources. When it comes to resource utilization in a containerized environment, scheduling decisions are natively done on the basis of CPU and memory. Constraints on other resource types must be handled separately. Increasing a data center's computational power by adding new hardware is relatively cumbersome due to the manual installation required, and improving the utilization of existing hardware has thus been subject to a great amount of research. Network resource utilization in data centers has also seen a lot of research during the last decade. Concepts such as SDN and network function virtualization have been explored as means to improve network performance. However, a lot of challenges remain and currently dynamic behavior such as limitation of link rate during low traffic load is not a native part of the hardware [46].

One challenge in data centers is the cloud provider's limited possibility to control the network outside of the data center. While there are examples of companies, such as Amazon [4], having their own global network infrastructure, most cloud provider's have limited control over the core internet equipment. With this hardware being outside control, a lot of networking optimization is made impossible. In addition, the service level agreements (SLAs) of public clouds today focus on computation and storage. Guaranteeing network performance with SLAs which include factors such as bandwidth and latency is rare and rather upheld at best effort [46].

When moving cloud technologies into the telecommunication space, both the conditions for network control as well as the performance requirements are different compared to ordinary data centers. While the telecommunications operators do not have access to core internet infrastructure they do have the possibility to control a larger part of the network infrastructure between the user and the server. Compared to data centers, telecommuni-

cation networks span much larger areas and contain a lot more network elements such as switches and routers. This means that more optimization possibilities exist both in geographical terms, but also counted as network elements under control.

An important difference in quality-of-service requirements between cloud providers and telecommunications operators is that telecommunications networks do have to uphold SLAs with guarantees of a certain bandwidth, latency, or packet drops. The way this is usually solved today is by overprovisioning network resources. Static bandwidth allocations are set well beyond the expected max requirement [50]. With growing traffic loads and more data being sent over mobile communication, this wasting of bandwidth by overprovisioning might soon become a cost efficiency target.

Research has addressed the problem of dynamically allocating bandwidth, and software-defined networking is one of the proposed solutions. However, combining dynamic network resource allocation with the cloud technologies envisioned for the 5G networks is in need of further exploration. One of the reasons that dynamic bandwidth control is not wide spread today is that overprovisioning bandwidth to accommodate the data volumes associated with previous generations has been relatively cheap. With 5G these data volumes are expected to increase significantly and as such increase overall cost. Another reasons is that the increased flexibility and control that cloud technologies offer has not been present in previous generations. This increased control opens up to optimization that has historically not been possible. However, the combination of telecommunication and cloud is new territory and in need of research. This need of research motivates us to further investigate the possibilities of combining a microservice architecture orchestrated by Kubernetes with an SDN controller. In order to explore this, the thesis will focus on building a simulation environment. In such an environment one could observe and control network traffic through abstracted Cloud RAN components. Furthermore, the simulation environment could make it possible to try different approaches to dynamic reallocation of bandwidth, as well as analyzing their potential network resource savings.

1.3 Academic Contribution

This section will present previous work within this field of study through the Related Work section and then detail the contribution of this thesis in the Contribution section. Lastly, the distribution of work between the two thesis writers will be detailed.

1.3.1 Related Work

Several approaches to improve network resource utilization have been put forth in academia. We have selected a subset of these that are most in line with our envisioned goal. These will be presented in the section below.

Wang and Shi [71] proposed and implemented a static bandwidth allocation schema through an Open vSwitch (OVS). They utilized an SDN controller to establish their bandwidth strategy on the OVS. This strategy was based on allocating different amounts of bandwidth depending on the priority level of the host. Finally, they verified that the OVS enforced their bandwidth strategy by measuring the throughput for each host.

Raza et al. [62] investigated the benefits of a dynamic resource allocation strategy between network slices. They compared this to a static resource allocation where each slice would be provided with enough resources to meet peak service requirements. To obtain new dynamic configurations they formulated a mixed-integer linear programming (MILP) problem. When the resources were dynamically allocated the researchers showed that they could decrease the rejection probability of incoming connections. This in extension means that network providers could allow more services into the current infrastructure.

Shifting focus towards the cloud aspect of the problem, Kim et al. [53] investigated the effects of network-intensive containers on the overall performance of the cluster when each pod is given a specified bandwidth. They found that even when pods are scheduled that collectively have lower bandwidth requests than the total available bandwidth performance is degraded. This is due to a kernel process called Softirq that uses high levels of CPU computations to handle the network workloads. Their conclusion was that CPU capacity must be considered even beyond what the pods request when scheduling a network-intensive pod. This is something that has to be kept in mind when evaluating the performance of the cluster in a telecommunications context.

Furthermore, Zhang et al. [76] propose using constraint programming to dynamically allocate virtual hardware resources to virtual machines operating under QoS requirements. In addition to meeting QoS requirements, they also take the cost of virtual cloud resources into account. Through simulations, they could show that their algorithm could both reduce QoS violations and lower resource usage costs.

As is evidenced by the research mentioned above progress has been made both within allocating resources in cloud environments and with regards to SDN controlled virtual switches. In this thesis, we aim to build upon this and explore how real-time load data can be used to dynamically allocate bandwidth to different traffic classes. In particular with respect to three 5G use cases defined by the telecommunication standards organization 3rd Generation Partnership Project (3GPP). These use cases will be further elaborated in Section 2.1.3.

1.3.2 Contribution

As one of the goals of a master thesis is to add to the current knowledge base within a certain research topic, this section will detail how this thesis will accomplish this. Building upon the research presented in the above section this thesis will contribute to the field within two areas:

- Investigating how a simulation environment for a 5G Cloud RAN can be implemented with the help of current cloud technologies.
- Analyze the possibility of dynamically prioritizing bandwidth access in a hierarchical fashion and what resource savings could be made through this approach.

To refine the areas presented above to questions that can be answered through this thesis we pose the following research questions.

Research Question 1.1 How can we develop a cloud-based simulation environment that represents the essential components of an SDN controlled 5G transport network?

Research Question 1.2 How can network traffic with different QoS levels be prioritized in an SDN controlled cloud-based simulation environment?

Research Question 2.1 Under what conditions can a dynamic network resource allocation improve the utilization of network resources in a 5G transport network, compared to static resource allocation?

1.3.3 Distribution of Work

In a master thesis spanning over 20 weeks with two students working together, you will inevitably find influences from both writers in all parts of the process. With this said, certain aspects could be attributed to a higher degree to one of the writers.

Regarding the implementation process, August contributed the most to the scripting and algorithmic parts of the environment. Daniel dedicated a larger effort to networking elements such as the Open vSwitch and Linux networking. The majority of programming done in Python was done through the pair programming technique, ensuring a high degree of collaboration.

With respect to the writing of the report, Daniel put more time into the theoretical foundations such as related work and methodology explanations. August worked to a larger extent on sections with a broader scope such as the discussion and thesis motivation. However, the majority of the report was written in collaboration and multiple iterations over all sections of the report ensured that both authors influenced the resulting document.

Chapter 2

Background

2.1 Telecommunications/Networking

To provide the reader with a clear context for this thesis, a high-level explanation of traditional telecommunication and the advances made through 5G will be put forth in this chapter.

2.1.1 Telecom Architecture and Evolution

At a very high level, a telecommunication network can be divided into two parts, the access network, and the core network. If a user wants to connect to the internet, their User Equipment (UE), e.g. a mobile phone, must first connect to the access network. The access network then provides connectivity between the UE and the core network. The core network's main function is to enable connectivity to a data network, oftentimes the internet [57]. The focus of this thesis will be on the access network, also referenced as the RAN).

Radio Access Network

As mentioned previously the role of the RAN is to provide connectivity between a UE and the core network. To do this a radio tower equipped with antennas is needed to connect the UEs to the RAN. The traffic moving both up and downstream in the network needs to be processed and forwarded. This processing was previously done in hardware units at the base of the radio tower but for the next generation networks, this architecture has evolved to a more centralized structure [57].

Switching and Routing

To enable traffic to flow through the network, forwarding packets to their intended destination is necessary. To achieve this, switches and routers are utilized. The purpose of a switch is to enable connectivity between multiple devices on the same network. A switch has multiple ports, each port is a pathway to a certain device. It controls the flow of data between the devices by forwarding packets between the ports on the switch. Configuration options such as adjusting speed and available bandwidth per port is part of the traffic management capabilities of a switch [9]. For the purpose of this thesis a router can be viewed as having the same function as a switch but connection networks instead of devices. An important aspect of a switch is the behavior exhibited when a switch becomes overloaded with network traffic, which is called congestion. Switches are configured to handle congestion through packet dropping, which occurs when the ques in the switch are full. Packet dropping simply means deleting the packages that cannot fit in the queue. How much traffic can be passed through each port before packet dropping occurs is determined by the quality of service (QoS) set for different traffic flows. A better quality of service could mean higher queue capacity or more available bandwidth [12].

2.1.2 5G Development and Capabilities

As mentioned above, parts of the envisioned 5G RAN architecture have been centralized with the goal of achieving better resource utilization. This concept will be further detailed at the beginning of the following section. This will be followed by a section explaining which network functions are executed in the network, lastly, the stakeholders in the 5G development will be detailed.

RAN Architecture

In previous RAN generations each Remote Radio Unit (RRU), where the radio waves are received and transmitted, was coupled with a Baseband processing unit (BBU). All of the BBUs processing functions were geographically co-located in the same physical unit. In 4G architecture the BBU is located at the foot of the radio tower [72]. In the 5G architecture the BBU is divided into two distinct parts, the Distributed Unit (DU), and the Centralized Unit (CU). This division is identified as the Cloud RAN architecture since it enables centralized cloud data centers where network functions can be deployed. When the functions running inside the DU and CU are virtualized, they are commonly refered to as virtualized DU (vDU) and virtualized CU (vCU) It should be noted that one CU is connected to one or more DU's, thereby enabling the benefits of centralized computing centers [57]. In Figure 2.1 the two different architectures are displayed.

Distributed Unit and Centralized Unit Depending on the vendor deploying the Cloud RAN, functions can be divided between the CU and DU in different ways. This division of functions affects which type of processing is done in which unit. To facilitate collaboration between vendors, the standards organization 3GPP has proposed a standardized division of functionality. This will give responsibility to the CU for functions such as QoS

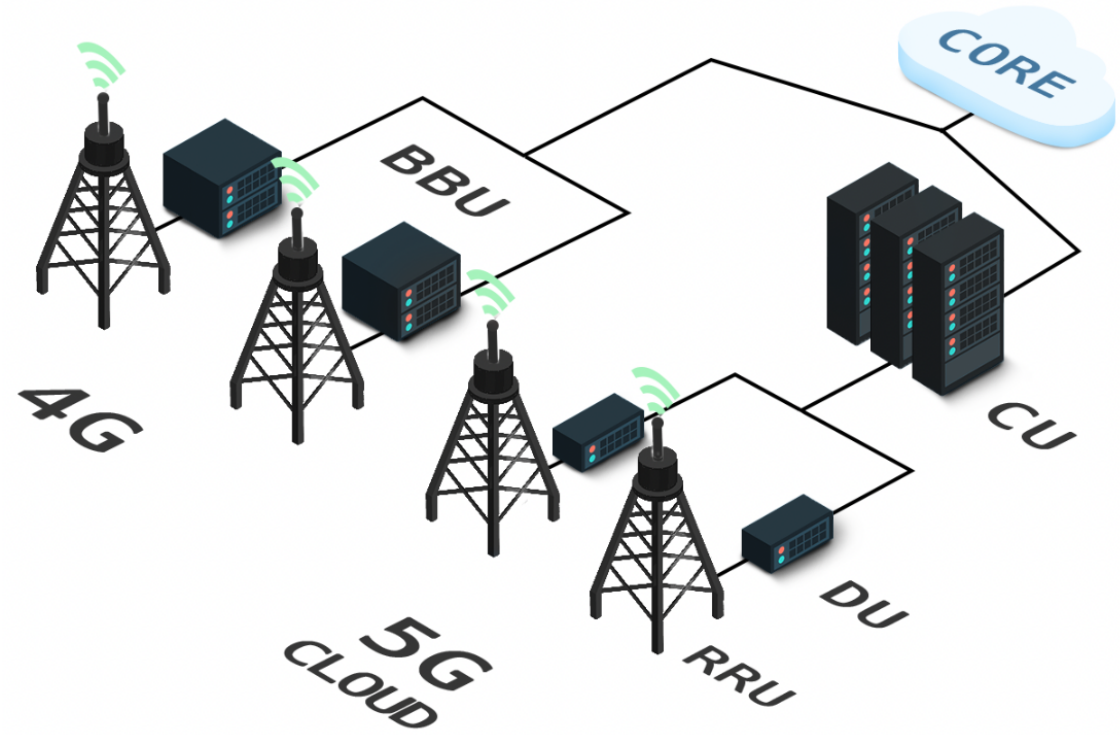

Figure 2.1: Architecture of 4G and 5G Cloud. Icons by [17].

Flow handling, connection establishment, and release. The DU assumes responsibility for protocol error detection, noise reduction, and lower-level encapsulation of data [57].

Network Functions

For a network to function properly, there are many processes that need to be carried out continuously. These are typically called Network Functions (NF), they are executed on dedicated hardware resources in a traditional network. Examples of NFs are firewalls, deep packet inspection, performance monitoring, and access routers [27]. In a Cloud RAN setting these are virtualized on high volume servers, a concept which will be detailed later in section 2.4.2 Cloud RAN.

Stakeholders, Open RAN and Vendor diversification

The stakeholders in the 5G development are numerous and range from network operators and telecommunications companies to policymakers and third-party businesses using the new technology [41]. In an effort to facilitate collaboration, flexibility, and ecosystem innovation between stakeholders multiple standards have been developed. The extent of this standardization effort marks a notable difference between the development of 5G and 4G networks, where the new generation focuses a lot more on compatibility between different vendors. Telecommunication systems have traditionally been sold as packages with the same vendor providing a full set of components, including software and hardware of the transport network infrastructure.

The standards organizations 3GPP and O-RAN Alliance seek to put an end to this vendor lock-in by building a common 5G framework for interface connectivity, coined Open Radio Access Networks (O-RAN). This framework would allow for new vendors to enter the market while simultaneously increasing competition between the traditional telecommunication vendors. In the face of this market democratization, the pressure to develop innovative and high-performing solutions increase [30].

2.1.3 5G Use Cases

As the development of fifth generation networks progressed three different service classes emerged to satisfy the projected usage requirements. These are the Enhanced Mobile Broadband (eMBB), Ultra Reliable Low Latency Communications (URLLC) and Massive Machine Type Communications (mMTC).[70]

eMBB (Enhanced Mobile Broadband)

The eMBB service could be characterized as an improved version of the 4G current broadband deployment. This entails high peak data rates and moderate reliability defined as a relatively low packet error rate [60] This will be the most commonly used service for regular cell phone users and allow for streaming of movies or online gaming even if the user is travelling on a high-speed train at 500 km/h [2].

URLLC (Ultra Reliable Low Latency Communications)

The URLLC service is the enabler of critical communications that need very high reliability and low latency to function properly. The rate of transmission is much lower than eMBB but calls for more scheduling to achieve the needed reliability. The packet drop rate is expected to be at a very low rate as compared to eMBB.[60] Examples of applications include remote surgery, drone control, and driverless cars [1].

mMTC (Massive Machine Type Communications)

mMTC differs from the above services mainly due to the number of devices envisioned for this traffic class. MMTC will allow for seamlessly connecting thousands of Internet of Things (IoT) devices. Oftentimes these IoT devices try to transmit small payloads at irregular intervals and to increase efficiency the devices could be coordinated to maximize the throughput per resource [60]. Concrete examples of applications are wearables, trackers and sensors [2].

2.2 Practices and technologies enabling 5G

In order to deliver the advanced functionalities promised by the 5th generation of mobile communications, the telecommunications industry will need to leverage best practices and technologies from the software industry. Since the launch of long-term evolution (LTE) 4G network in 2009 [36], the software industry has experienced large growth combined with an influx of new tools and ways of working.

A practice that has become increasingly important in delivering large complex software systems is the unification of development and operations activities, coined DevOps. As a part of these activities, the deployment method of choice has become to deploy in the cloud. [42] Both DevOps practices and cloud computing technology will be important components in the development of 5G. To better understand these concepts and how they serve a purpose in mobile communication, one needs to look at the core values of DevOps, as well as the technological components of cloud computing. In the following section a number of different tools will be discussed. In an effort to summarize these tools, Table 2.1 is provided at the end of the Background chapter.

2.2.1 Unification of Development and Operations

Previously separate, the unification of development and operations activities, has allowed for a holistic view of the software life cycle. DevOps practices strive to make software development and operations easier by building on core values such as automation, observability, scalability, and availability.

Automation Software application deployment consists of several different stages, such as integration, testing, infrastructure provisioning, and the deployment itself. Executing each stage manually repetitively during the development process is both time and resource-consuming, and can thus benefit from an automated solution. Creating a pipeline for continuous integration and deployment that can perform the actions of each stage through the push of a button is what is called automated deployment [66].

Observability Observability is a measure of how easy it is to understand what is happening inside of a system. This includes understanding why or why not your system is working, and where potential errors originate. To achieve high observability and gain insights into the system a combination of techniques such as metrics collection, logging and tracing are usually employed. By continuously collecting data the system operator can monitor metrics such as CPU usage or packet drops and understand the underlying causes for fluctuations or spikes. The data collection is vital to gain a deeper understanding of your system and its potential areas of improvement. What constitutes important data is however unique to each system and in the realm of observability there is, unfortunately, no one size fits all [42].

Scalability Scalability refers to the ease of increasing the capacity of a service to enable more users to access it or more data to be processed. This can be done either through creating another replica of the entire application or in the context of microservice architectures, one can scale only the bottleneck microservice [66].

Availability The concept of availability could be described as the portion of time your service is working as intended, available to the users. For example, your service might be available 99.999% of the time if it has high availability. To achieve high availability, concepts such as redundancy and distributed deployments are important [42].

2.2.2 Cloud computing

Since the introduction of cloud computing around 2007, the technology has seen tremendous growth. As of 2020, the market for public cloud computing was estimated at 236 billion U.S. dollars, 27 times larger than in 2009 [40]. Companies like Amazon, Microsoft, and Google are the largest cloud infrastructure service providers, together capturing 61% of the market [63]. Today cloud computing technology is used to run our favorite websites, store our personal data and deliver the work tools we use daily.

In the paper "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility" from 2008 Buyya et al. define Cloud Computing as:

> "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between the service provider and consumers."[47]

From this definition, one can deduce three main characteristics of cloud computing. These are sharing of infrastructure, dynamic provisioning, and network access.

Sharing of Infrastructure In a cloud setting each running application gets access to hardware resources such as computational power, storage, and networking capabilities. However, the resources allocated to each application only account for a portion of the available underlying physical hardware. This partitioning of the hardware resources is done through a concept called virtualization. From the perspective of the application, it is running on its own computer while in fact it is sharing the underlying resources with multiple other virtualized applications [47]. The virtualization enables decoupling of the application and the physical hardware, which makes it possible to move the application between different computers without disturbance to the application's operations. Virtualization will be described more in-depth in section 2.2.3

Dynamic Provisioning Dynamic provisioning is another important characteristic of cloud computing. The portion of resources that each virtualized application is receiving is not fixed and can easily be scaled up to meet an increase in demand. Likewise the resources can be scaled down when the demand is low. This dynamic provisioning makes for more efficient use of hardware resources as the physical hardware can be adapted to the average load of all running applications, instead of to the peak load of each application.[47]

Network Access The last key characteristic in cloud computing is the network access to the infrastructure. This is what makes the resource infrastructure available over the internet and to other parts of the network. Without the network making cloud services available from any device anywhere the appeal of cloud computing would quickly diminish.[47]

Deployment Models While there are many different components built into cloud computing solutions their importance differs depending on the deployment model. It is therefore important to define the scope of cloud computing in relation to telecommunications. There are several different deployment models discussed in cloud computing, but the

two most common are public and private clouds. Public clouds refers to cloud computing being offered to the public, individuals or companies, on a pay-per-use basis. The public cloud includes the offering of companies like Amazon Web Services or Google Cloud [47]. The private cloud is a cloud that is deployed and managed by a company for its internal use. This is the type of cloud that is the most common in the telecommunications industry today, where only the telecommunications company deploying the cloud would have direct access to it [49]. While the end users connecting to the radio access network will have their data and network traffic processed in the cloud, they will never directly interact with it. This is an important distinction to make as this means that only the technological components, and not the business model, is similar to the public cloud solution. While running a private cloud requires a lot more effort as opposed to buying cloud services, it lets the company retain control over the infrastructure and its optimization. This while still benefiting from the technological advantages of the cloud computing architecture.

2.2.3 Virtualization

To further understand the technology behind cloud computing one must understand the role of virtualization in the cloud. Virtualization has become an important technology in modern data centers and is being used extensively to increase operational efficiency, scalability, reliability, and flexibility. Through the adoption of these concepts, operational expenses (OPEX) generated inside a data center could be decreased. The two main types of virtualization are hardware-level virtualization and Operating System (OS) level virtualization. Virtualized hardware is commonly referred to as a virtual machine (VM), while virtualized applications on the OS level are commonly referred to as containers.

Hardware Virtualization – Virtual Machines In hardware-level virtualization physical hardware resources such as CPU, memory or networking interfaces are virtually partitioned and allocated to a VM. The VM runs a separate OS from the host and is unaware that it is part of a physical computer [67]. The combination of hardware and software components needed to run a VM is referred to as a virtual stack. The virtual stack in hardware virtualization can be seen in the left part of Figure **??**. In data centers hardware virtualization allows for multi-tenancy, meaning multiple VMs can share resources of a single underlying server. This means different processes and applications can run on the same physical hardware with minimal interference. The virtual machines can also be moved dynamically between different servers in the data center allowing for optimization of the hardware resource utilization. If two different servers are running workloads with low resource utilization the VMs used to run the workloads could easily be relocated to a single server, thus minimizing the overall number of servers needed [65].

OS Level Virtualization - Containers In OS-level virtualization each virtualized partition, called a container, is given a separate user space with access to a portion of the host operating system resources. While hardware virtualization has long been the predominately used virtualization paradigm, containers have seen massive growth in popularity in recent years. Containers are run on top of a host OS or directly in a virtual machine and contain all the dependencies and libraries needed to run its application. The virtual stack needed to run containers directly on a host OS is shown in the middle of Figure 2.2. To

run containers in a VM, a slightly different stack is needed, displayed on the right in the same figure. The advantage of containers over virtual machines is that containers are extremely lightweight. This is largely due to the fact containers only need to store the exact dependencies used for the containerized application. Different containers are also able to share common packages and dependencies, making them even smaller in size [64]. Containers are suitable for modularized applications, where small subcomponents of an application can exist in separate containers. If a certain subcomponent of the application is receiving a heavy workload this specific subcomponent can be replicated to increase the computational capacity, as opposed to having to replicate the entire application [38].

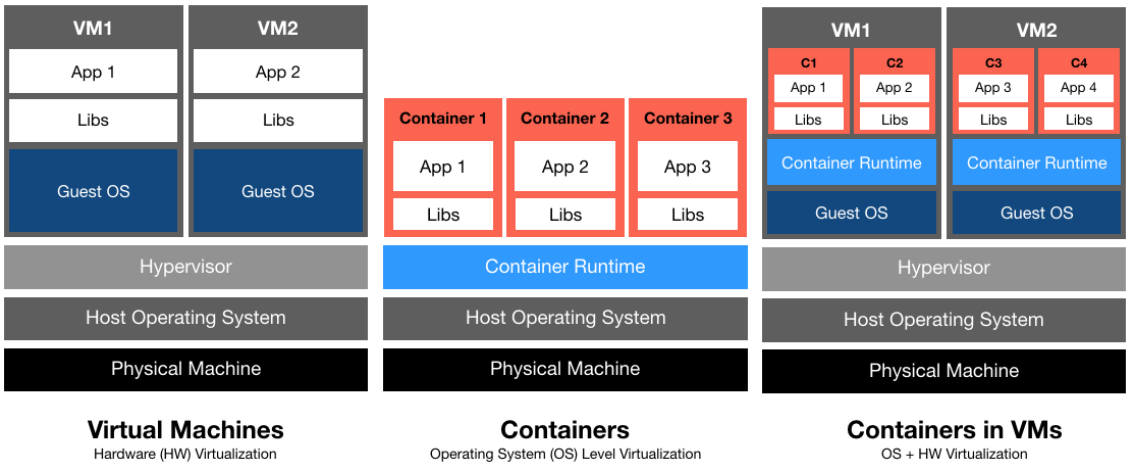

Figure 2.2: Virtualization stacks for virtual machines, containers and containers in virtual machines

Container Orchestration

When increasing the number of application components running in separate containers, the task of managing them quickly grows difficult. A running container can be fragile and it is not uncommon for containers to crash and die. If containers are run without an orchestration tool a system operator must, for example, respond to the death of a container by restarting it. This can get cumbersome with a large amount of containers, especially as container errors can propagate if the containers are dependent on each other's functionality. Sometimes the death of one container can cause others to crash. As a remedy against painful manual orchestration, container orchestration tools have been introduced. A container orchestration tool allows for the automation of tasks such as deployment, scaling, management, and networking of containers. The system operator can set a preferred state of the system, which will then be automatically maintained. If a container dies a new one will be automatically scheduled and deployed. While there are several different tools for container orchestration the most popular by far is the Google initialized open-source tool Kubernetes [69].

2.2.4 Software Defined Networking

As the worldwide network infrastructure has grown, network complexity has increased significantly. Not only have networks become a lot larger, but also more heterogeneous in terms

of equipment, applications, and services. Now the traditional networking architectures that were good for increasing performance in the early days of the internet, are making the networks inflexible and increasingly complex to manage. To fulfill the requirements of future networks, such as ubiquitous accessibility, high bandwidth, and dynamic management, new approaches are needed [74].

Software-Defined Networking (SDN) is a concept that has received a lot of attention in the ICT community in recent years. SDN is defined by the Open Networking Foundation as an emerging network architecture with two main characteristics. The first being a decoupling of control and data planes, the second being the programmability of the control plane [74]. In earlier architectural designs the control plane, associated with tasks like managing routing tables and storing network topology, and the data plane, where the user data is sent, were bundled together [54]. With the SDN design, more efficient configuration, better performance, and higher flexibility can be achieved [74]. By enabling programmability of the control plane, configuring network elements like routers and switches, can be simplified and done with increased dynamicity and automation. The control process in the control plane is achieved through the use of an SDN controller.

SDN Controller The SDN controller is the most important component of the SDN architecture. It centralizes the logic for controlling the network and uses a global network view as the basis for its decisions. The SDN controller will often deal with two types of tasks, network monitoring and network controlling. During network monitoring metrics about the network status are collected. These metrics are then used for network decision-making, where network controlling actions are decided with logical rules stored in the SDN controller. The SDN controller then updates the network by modifying the underlying infrastructure of the data plane. This infrastructure is made up of network elements such as switches and routers and can be modified by updating policing or routing rules in these elements [74]. If the network elements are virtualized, the infrastructure is referred to as a Virtualized Network Infrastructure.

Virtualized Network Infrastructure As mentioned previously in section 2.1.1, switching is an essential part of a network's functionality. To adapt to new virtualized environments, as described above, switches have also become virtualized. A commonly used virtual switch in production environments is the Open vSwitch (OVS). The OVS is supported by the Linux Foundation and is available freely to the public. It enables network providers to automate the behavior of their networks and each OVS is dynamically configurable through an SDN controller [31]. The OVS relies on Linux Traffic Control (tc) to provide QoS levels to different traffic flows. This means that traffic control features that are not available in tc, cannot be achieved through the OVS [34]. tc is a tool used to control network traffic on any operating system running a Linux kernel. For example, increase or decrease the available bandwidth on a network interface or simulate packet drops [24].

2.2.5 Control Engineering

Control engineering aims to enhance the performance of a system by adding control processes, sensors, and actuators. The control process captures signals from the sensors, analyzes these, and outputs instructions to the actuators, which in turn affects the behavior of the

system. An overview of a generic control system is shown in Figure 2.3 below. Due to the fact that the behavior that the controller process exhibits is tied to the sensed signals, this kind of system is called a closed-loop control system [56].

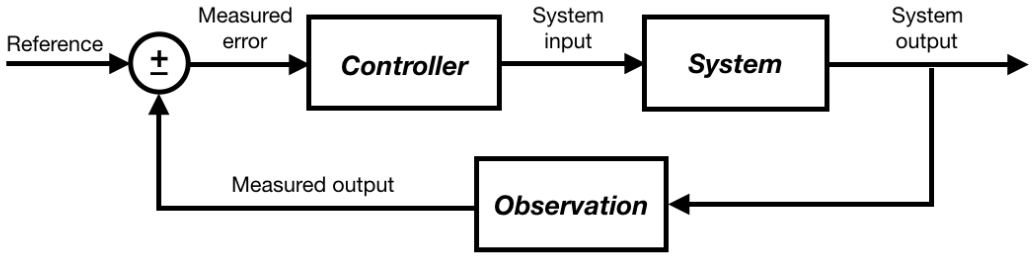

Figure 2.3: A generic closed-loop control system, adapted from [56].

2.3 Kubernetes

2.3.1 Kubernetes Introduction

The rise of containerization in the software industry brought with it an increasing need for container management. While running applications in containers solved many previous issues there was still a need for a system administrator to handle tasks such as automation, failover, monitoring. The more services a system contained, the more cumbersome it was to handle the overall operation and health of the containers [42].

As an answer to this problem, Google released the open-source container orchestration project Kubernetes in 2014. The open-source nature of Kubernetes proved to be a great advantage over other commercial orchestration solutions at the time. This made container orchestration highly available to all kinds of developers and the adoption rate of both containers and Kubernetes spiked. Not long after its release, the success of Kubernetes was a fact, the platform completely dominating the market [42]. Today cloud-service providers such as Amazon Web Services [3], Microsoft Azure[5] and Google Cloud Platform[18] offer Kubernetes orchestration natively.

In a 2020 software developer survey, by the popular question and answer site Stack Overflow, Kubernetes was ranked as the third most appreciated software technology outranked only by the containerization platform Docker and the operating system Linux [35].

2.3.2 The value of Kubernetes

Kubernetes is valuable both to developers and operations staff, providing solutions to many previously time-consuming and complex tasks. For developers, Kubernetes makes deployment easy by facilitating gradual rollouts of new applications. With the platform new functionality can be deployed to only one or a few containers in order to test new functionality on a few users before expanding to the entire user base. Other helpful features include autoscaling of applications to keep up with user demand. If Kubernetes discovers that the CPU usage

for a certain application is spiking it can automatically scale up the capacity by deploying more replicas of the application in question. Similarly, application instances not being used can be scaled down automatically in order to reduce the need for computational resources. If an application crashes the instance can be automatically restarted in order to maintain a predefined service level. This makes for a system with high availability only using the resources needed. In turn, this lowers the infrastructure costs through better resource utilization [42].

2.3.3 Kubernetes Architecture

A Kubernetes cluster is made up of several components, a schematic overview is provided in Figure 2.4. Observe that this is a highly simplified picture but it covers the parts that are important for a basic understanding of a Kubernetes cluster.

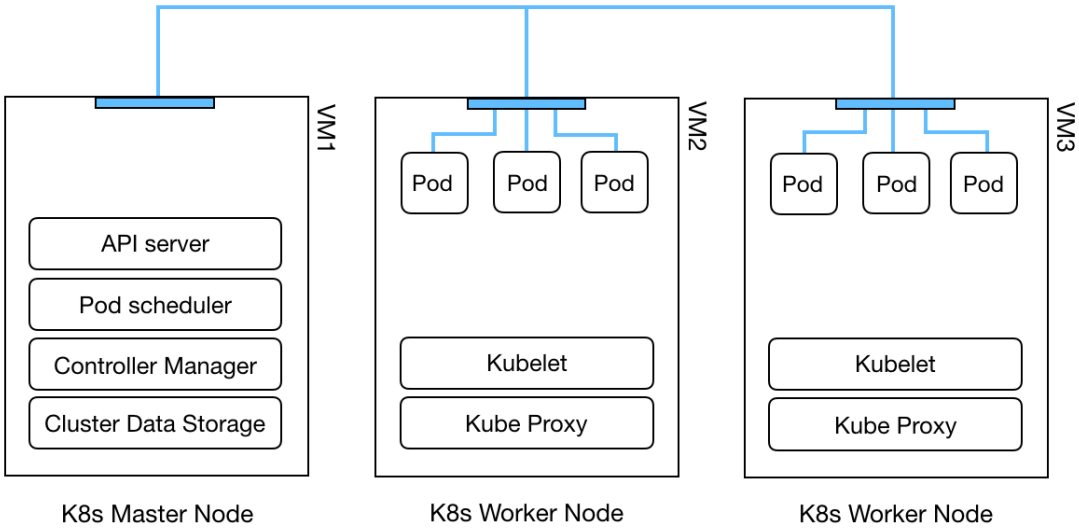

Figure 2.4: Kubernetes Architecture Overview with master and worker nodes, adapted from [21].

Nodes

A node in Kubernetes is a physical or virtual machine, a cluster usually has several nodes. There are two types of nodes, Master nodes, and Worker nodes. The key difference is that the Master node is part of the control plane of the cluster, meaning that it controls the worker nodes. The Worker nodes on the other hand runs the actual workload. To be clear, there is no inherent difference between a Master and Worker node, each of them is a physical or virtual machine, they just run different kinds of software on them [29].

Master Node The Master node is responsible for scheduling workloads through the Pod scheduler, serving application programming interface (API requests through the API Server, storing information about the cluster in the Cluster Data Storage, and running core control loops on the controller manager. Basically, it is the cluster brain making sure that everything works as intended and that the cluster can be accessed from the outside [22].

Worker Node Worker nodes on the other hand run the workloads that the Master assigns to them. It does this mainly through two processes, the kubelet and the kube-proxy. The kubelet makes sure that the intended containers are running and are healthy on the node. The kube-proxy upholds the network configuration on the node enabling communication with the pods [42].

Pods

A pod in Kubernetes is an object that contains one or more containers. These co-located containers share storage and network resources. The most common situation is for a single container to run in a single pod though. Why then bother with a pod at all one could ask? The answer is that in some situations a set of containers needs to run together to be effective. For them to be able to, for example, share storage, they can run on the same pod and still be abstracted as a single unit when controlling the cluster [42].

Container Networking To allow for more advanced networking inside a Kubernetes cluster, network plugins can be used. Flannel is a plugin that creates an overlay network on top of the standard Kubernetes network. This enables direct IPv4 communication between all pods in the cluster [14]. Furthermore Multus is a network plugin that lets the user create multiple network interfaces at each pod, which in turn enables segregation of traffic in and out of a pod [26]. This is an important capability to set up if one is to adjust the quality of service based on traffic class inside a cluster.

Jobs

To run and manage pods in Kubernetes you can use what is called a pod controller. A pod controller does things like making sure the container inside a pod is in good health, restarts the pod if the container would unexpectedly die, or load balances requests between different pods under the same controller. One common type of pod controller is a Job. A job is ideal to use for pods that have a specific task to complete and thereafter should exit. In the job specification, the number of times a pod should complete a task before exiting can be specified. If a pod would die before the tasks are completed the job will restart new pods until the job is finished.

2.3.4 Observability in Kubernetes

One of the requirements for observability is the availability of metrics on the observed system. To collect and compile system metrics in Kubernetes, Prometheus is a highly adopted solution.

Prometheus

Prometheus is a project by the Cloud Native Computing Foundation (CNCF) which is used for monitoring and alerting on its configured targets. The Prometheus program scrapes metrics at given intervals to evaluate the health and performance of these targets [32]. For container metrics, a common open-source metrics agent is the container advisor (cAdvisor).

cAdvisor exposes resource usage data about the running containers in the cluster, these can then be scraped by Prometheus [6]. The metrics can be accessed through the Prometheus database, either in table or graph format. In addition to this, alerts can be configured to warn the system administrator if certain system behaviors are observed [32]. Prometheus is so frequently used in combination with Kubernetes that Kubernetes components expose their metrics in Prometheus native format [25].

2.4 Cloudification and containerization of 5G

2.4.1 Cloud Native

An important concept in cloud computing is cloud-native. Cloud-native refers to the design and operating patterns that ensure that the developed software product takes full advantage of the cloud deployment model. More concretely this means employing practices such as continuous deployment, containerization, and division of monolithic products into microservices. These cloud-native practices allow for a product that has a high degree of automation, receives new functionality faster, and can scale elastically with demand [44].

The best practices associated with cloud-native stem from large tech companies like Netflix, Alibaba, and Facebook. Because of the large successes of cloud-native practices in the tech industry, cloud-native is now envisioned to become an important part of cloud development in telecommunications. It should be noted though that certain performance requirements specific to the telecommunications industry might not be supported by traditional cloud solutions. This means that telecommunications companies must develop new products and solutions to incorporate cloud technologies into their products. At Ericsson cloud-native practices are envisioned to help improve granularity and speed of software updates, automate virtual network function configuration, and adapt software architectures to make better use of cloud data center resources [48].

2.4.2 Cloud RAN

Cloud Radio Access Network (Cloud RAN) is the notion of 5G network compute functionality running in a cloud setting. Instead of network functions running on purpose-built hardware, they instead run on virtual machines or in containers on generic hardware. Such network functions, with examples in 2.1.2, are referred to as cloud-native network functions (CNFs). In this type of setup, the hardware functionality of traditional radio systems is emulated in software. The advantage of this is that the generic hardware, commonly referred to as COTS hardware, can be provided by a different vendor than the one providing the software. The Cloud RAN software being vendor agnostic gives increased flexibility in the network architecture. A carrier can even choose to combine CNF software from different software vendors [49].

Another advantage of the Cloud RAN setup is great scalability as the capacity of a CNF can be easily increased by deploying a new virtual machine. With virtualization, network functions can be dynamically scaled across all available hardware resources to adapt to the current network load. Previously, purpose-built hardware could only be used by compatible network functions. This type of setup can lead to poor resource utilization when unused

Tool	Description
Kubernetes	Container orchestration platform
Docker	Container runtime software
Open vSwitch	Virtualized production grade switch
Flannel	Plugin for the creation of an overlay network in Kubernetes
Multus	Enabling the creation of multiple network interfaces at each Kubernetes Pod.
Prometheus	Observation tool for Kubernetes cluster

Table 2.1: Summary of tools enabling 5G Transport Network Cloud functionality

Tool	Description
iPerf	A networking tool that generates traffic between two points on an IP network [20].
Flask	A lightweight web application framework used for Python web development [15].
Postman	A platform for API development, letting you requests to test your API [39]
Grafana	An open-source project for querying and visualizing time series data [19].
CBC	Coin-or Branch and Cut is an open-source mixed integer programming solver [7]

Table 2.2: Summary of tools supporting tools used in the thesis

hardware resources of one network function cannot be used by another network function. To avoid running out of hardware resources the solution would be to keep an overcapacity for each network function. In the future 5G network, Cloud RAN will complement purpose-built solutions increasing interoperability between vendors, improving resource pooling, and allow for cross-domain innovation [49].

2.5 Supporting Tools and Softwares

In addition to the tools that have been detailed in the previous sections and in Table 2.2, the simulation environment developed in this thesis uses a number of supporting tools. These are used both to run the simulation and facilitate development. They will be briefly covered below.

Chapter 3
Creation of the Simulation Environment

This chapter will detail the methodology and the resulting environment associated with RQ 1.1 and 1.2. It will start off with a brief description of the literature review process and then move on to the utilized methodology. After that, the design process itself will be detailed and finally, the resulting environment is presented.

3.1 Literature Review

After the establishment of RQs 1.1 and 1.2, the first step was an extensive exploration of relevant learning resources connected to 5G, networking, containerization, container orchestration, traffic control mechanisms, and optimization algorithms. Our main tools for gathering these resources were Google, LUBsearch, Google Scholar, ConnectedPapers, and educational material from Mprical. Oftentimes one resource led to another through investigating references to the current resource or using ConnectedPapers to find similar research. A small number of these were detailed in the Previous Work section. These were selected on the basis of similarity to the intended output of the thesis. It should be noted though that a lot of the learning material was also utilized to enable us to understand the tools and components used to build the simulation environment. The thorough review of documentation and tutorials made it possible to set up the simulation environment described below.

3.2 Design Science Methodology

To create a simulation environment representing RQ 1 in an accurate way, a workflow inspired by the Design Science methodology was adopted. In Design Science, design is viewed as both a process and a product. The process is divided into two iterative phases, developing and evaluating. This iteration is referred to as the Design Cycle. The product could be a model, method, software implementation, or a new problem formulation. The intended

output of this thesis was a software implementation of the simulation environment. Furthermore, two evaluating cycles are present in this methodology to ensure both Rigor and Relevance of the product. Rigor means applying existing theories and frameworks from the scientific community to the design process. This could be using a high-level theoretical framework to decide upon the software architecture of your solution. As an example, Control Theory was used as inspiration for the design of the control part of the simulation environment in this thesis. Relevance means making design choices that are implementable and applicable in the intended environment [52]. A schematic overview of the workflow inspired by this methodology is provided in Figure 3.1 below.

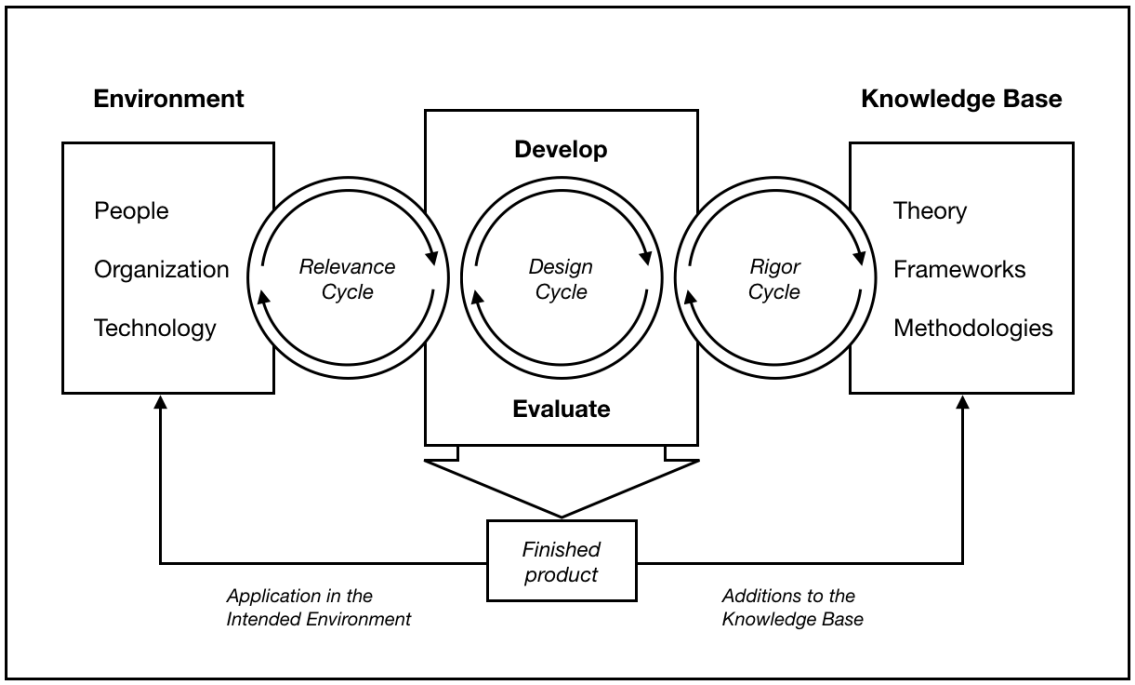

Figure 3.1: An overview of the Design Science process, adapted from [52].

The implementation process will be presented as three distinct phases in the chapter below. This is partly a simplification to make it easier to follow the design process and the choices made. Some overlap between the phases existed and sometimes we had to go back to a previous phase to alter the design due to newly acquired knowledge. The phases were named vDU and vCU, SDN Controller, and Network Traffic Generation, each one will be detailed in the following sections.

3.3 Envisioned System

To improve readability, a high-level concept of the intended simulation environment is presented in Figure 3.2. As was detailed in section 2.1.2 RAN Architecture, a Cloud RAN architecture envisions several vDUs being connected to a single vCU. The same concept will be mimicked in our simulation environment. Network traffic will be flowing from left to right in Figure 3.2, all of it passing through the virtual switch placed in the vCU. It is at this virtual

switch the SDN Controller is envisioned to allocate bandwidth dynamically to the different traffic classes. Observe that this is a highly simplified view of the system, more exhaustive figures will follow as each development phase is presented below. An overview of the three phases is shown in Figure 3.3

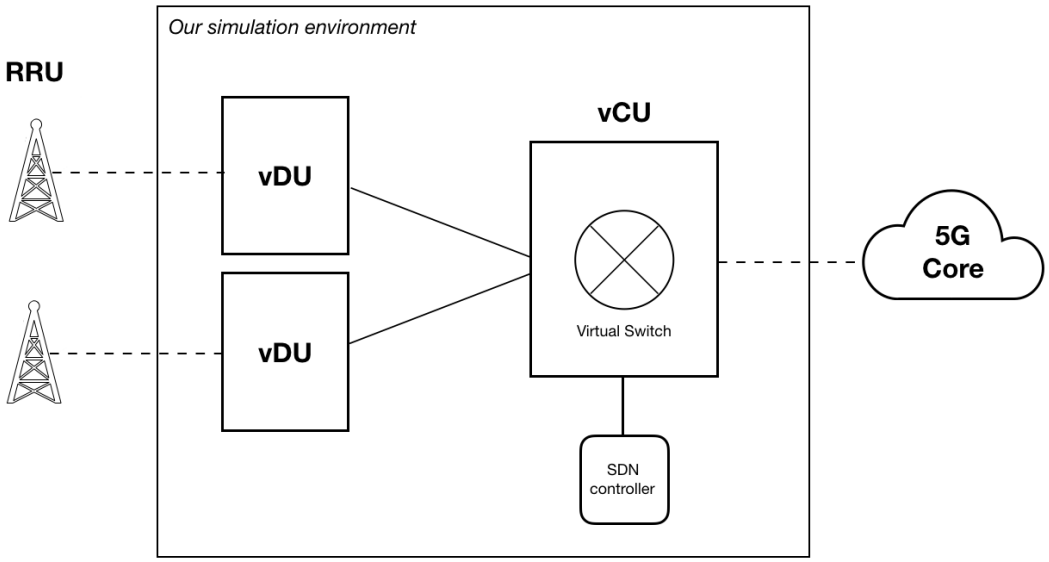

Figure 3.2: An overview of the envisioned system.

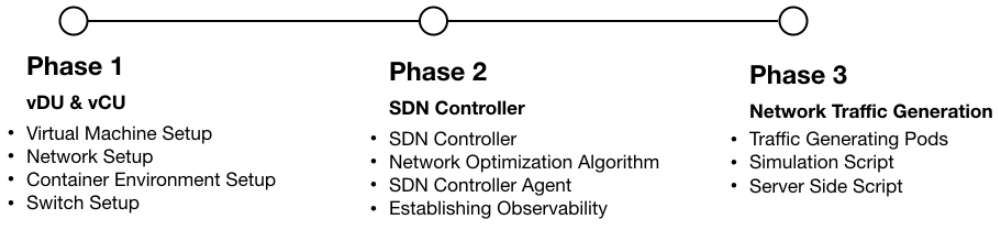

Figure 3.3: An overview of the implementation phases.

3.4 Phase 1: vDU and vCU

In this phase the goal was to build an abstraction of the network connecting the vDUs and the vCU in the 5G transport network. The environment should contain the essential infrastructure needed to enable simulation and observation of network traffic on this network. As the focus was put on the utilization of network resources, the network functions that are carried out in the vDU and vCU in a real RAN were intentionally left out.

3.4.1 Design

To be able to simulate servers on the vDUs communicating with a vCU, a virtual machine setup was deemed suitable. The virtual machines were instantiated on a single PC in the

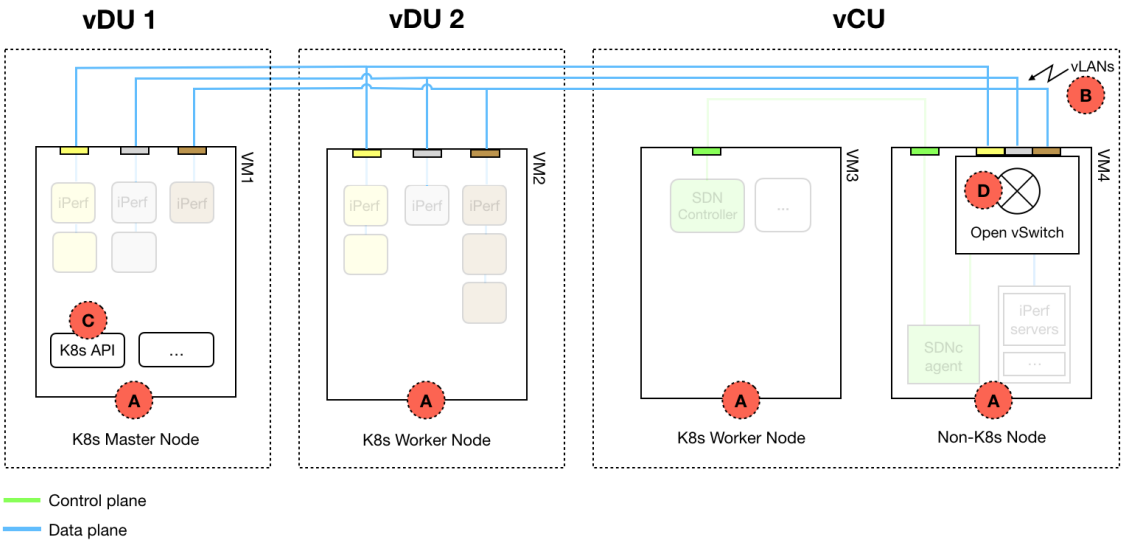

Figure 3.4: The system in phase 1, with VM:s, Kubernetes and virtual switch added.

virtualization environment VirtualBox. In a real world scenario each virtual machine would likely be placed on a blade server. For our setup we chose four virtual machines, two modeling two separate vDUs, and two modelling a single vCU, with a network switch placed in one. The four VMs in the simulation environment can be seen in Figure 3.4, with the label A. The virtualization stack with the specific components used can be seen in Figure 3.5.

The Networks Between the three virtual machines, virtual local area networks (vLANs) were established. These networks are an abstraction of the network infrastructure, such as cables and switches, that would connect an interface on a vDU with interfaces on a vCU. With the help of these vLANs, traffic flows could be separated into QoS levels. In our case, these were named “gold”, “silver” and “bronze”, where traffic sent over the gold network would receive the highest priority, while traffic sent over the bronze would receive the lowest priority. The networks are illustrated as the blue lines at the top of Figure 3.4, labeled B.

The Container Environment To create a container environment, Docker and Kubernetes were installed as container runtime and container orchestrator respectively. As Kubernetes runs on top of Docker, most interactions with the cluster were done through Kubernetes. In a real scenario each vCU and vDU would likely have both a Kubernetes Master Node and several Worker Nodes, but for the sake of our simulation vDU 1 was instantiated as a master node and vDU 2 was instantiated as a worker node. The vCU was instantiated as a worker node as well.

VM 1, VM 2 and VM 3 together made up our Kubernetes cluster, Kubernetes was not installed on VM 4 as this VM was intended for the OVS. In order to give certain pods privileges to control the cluster, Role-Based Access Control (RBAC) was set up. With RBAC, increased privileges such as starting pods or accessing cluster metrics can be given to a pod. This was essential during early development to enable observation and control of the cluster. Eventually, this component was no longer needed when the virtual switch that is described

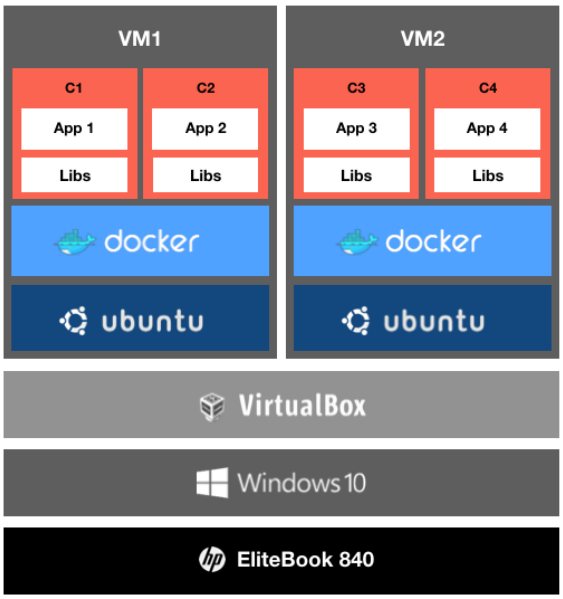

Figure 3.5: The virtualization stack used in the simulation environment.

below was implemented.

To connect pods to different vLANs, according to their traffic priority, custom Linux bridges were created. In essence, these bridges create a connection between a pod and a network interface on the VM that the pod is located on. To enable several Linux bridges to connect to each pod, one for each traffic class, the Multus network plugin was used. Through Multus three new network interfaces were created on each pod, the gold, silver, and bronze interfaces.

The Kubernetes master orchestrator functions and the other container environment functions can be seen in Figure 3.4 labeled C. Multus is not shown in the figure to avoid being overly detailed, but was part of the creation process of each pod.

Switch Setup An important component of a vCU would be a network switch forwarding data from the vDUs to the Core Network, and vice versa. While such a switch could be physical, a virtual version would allow for increased flexibility and reconfigurability. In order to create this virtual switch in our environment, the open-source switch OVS was chosen. The OVS was installed on VM4 and configured to connect to the vLAN networks that had previously been established. With this configuration, a vDU-pod connected to the gold network could send traffic to the virtual switch, where the traffic would be handled and prioritized according to the network it was sent on. The three interfaces of the OVS together share a bandwidth of 110 Mbit/s. When configuring the OVS, a certain amount of bandwidth is allocated to each interface. For example, the switch could be configured to give gold 50 Mbit/s, silver 30 Mbit/s, and bronze 30 Mbit/s. The switch can be seen in Figure 3.4, labeled D.

The Design Process

During the phase, a lot of configuration was required to make the different components work together. The virtual machines that were used had been pre-configured by Ericsson, specifying settings such as how much memory or CPU each VM was entitled to. Docker and Kubernetes were also pre-installed. This meant that the set up of these components went quicker and less manual configuration was needed. Throughout the entire phase, dummy pods were used, from which traffic was manually generated to evaluate and test the setup. While the process as a whole was incremental, each component that was added required a lot of iterative work in itself. These iterations consisted of activities such as making sure pods could connect to the right networks, giving pods the right privileges, and configuring the network to connect to Kubernetes pods and the switch.

3.4.2 Relevance

The choices made in this phase were mainly guided by Ericsson's explicit intentions of developing a Virtualized 5G RAN. They see significant benefits in virtualization, for example, the possibility of using the same type of server hardware throughout all parts of the network. Their emphasis on virtualization made using a virtual switch a natural choice. OVS was selected due to its widespread adoption and it being an open-source project, which is in line with Ericsson's goal of supporting open source [8]. Containers and container orchestration are relevant as tools since Ericsson also explicitly aims to increase the flexibility with which they can scale up and down resources in a Virtualized RAN [37]. For these tools the open-source projects Docker and Kubernetes were chosen due to their high adoption rate, detailed documentation, and large supporting communities.

3.4.3 Rigor

To complement the input from Ericsson with additional scientific research, research papers with similar objectives to our own were explored. The choice of Kubernetes was motivated by researchers in [61] investigating Kubernetes network plugin performance; they referenced the platform as the leading container orchestration tool among cloud service providers. They also emphasize the importance of container networking solutions to achieve a well-functioning cluster. Kubernetes was also corroborated by Kim et al. [53] who named it as the most utilized container orchestration platform. A paper where researchers were investigating bandwidth control through an OVS was found as well [71]. The researchers proposed a setup where three client VMs were connected to a server VM through two OVS instances. As our scope only demanded three client VMs connected to a single OVS VM, we modified their setup to look like Figure 3.4.

3.5 Phase 2: SDN Controller

In phase 2 the goal was to build an SDN controller which could modify the network infrastructure built in phase 1. In order to do this, the SDN controller would have to both

observe and control the infrastructure. The network element subject to control was the virtual switch, described above. In our simulation environment, the SDN controller will control a single switch and observe traffic from the vDU 1 and vDU 2. In reality, an SDN controller would likely be connected to a lot more switches and other network elements, as well as observing larger clusters. However, our abstraction was similar in principle while the complexity was significantly lower. To allow for flexible placement and increased scalability, the SDN controller was built to run in a Kubernetes pod.

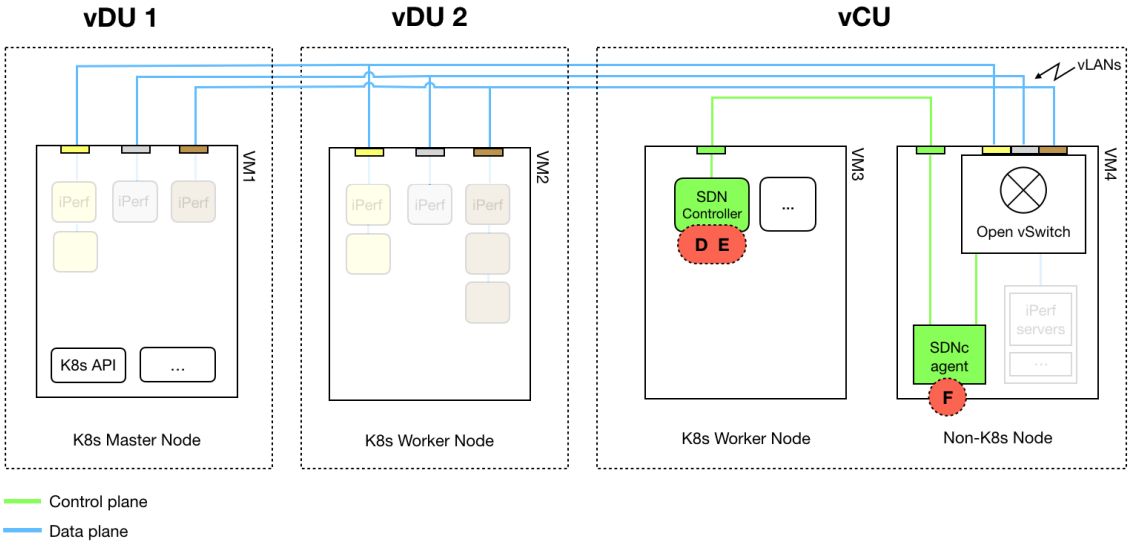

Figure 3.6: The system in phase 2, with the SDN controller (in green) integrated.

3.5.1 Design

SDN Controller To allow for easy interaction with the SDN controller it was built around a micro web framework called Flask. Based on the micro web framework, the controller continuously listens for incoming connections, allowing us to manually connect to the controller for configuration or data extraction. Both manual control and the SDN controller's interaction with other system components are done through an application programming interface (API) based on the style representational state transfer (REST). REST API is a very common style of machine-to-machine communication where each interaction can be expressed through simple operations such as GET, POST, PUT, DELETE.

The control loop of the SDN controller consists of observation, calculation, and configuration. In the observation step, the controller collects metrics from the virtual switch which returns data on sent and received bytes and packets. Using this data the metrics bandwidth, packets dropped and packet drop rate is calculated.

After the calculation of the above-mentioned metrics, the current bandwidth configuration settings for the OVS are compared to the current demanded bandwidth from each traffic class. Thereafter a new configuration is calculated using a linear programming algorithm, this algorithm is described in more detail in the following section. In the Figure 3.6 the SDN controller is labeled D, while the algorithm inside is labeled E.

To make the SDN controller run in a pod, it was compiled into a Docker image based on Ubuntu 18.04. In addition, packages for running Python and the Kubernetes Python API were installed.

Network Optimization Algorithm The solver used is called Coin-or Branch and Cut (CBC) which is an open-source mixed integer programming solver. Specifically, the implementation called PuLP [33]. The problem is constructed as is shown in Equation 3.1 below. This problem only has two relatively simple constraints due to the fact that we only consider bandwidth resources for a single switch in this thesis. In an extended scenario other factors and constraints, such as more virtual switches or other resource constraints, could be taken into account and could have easily been added to the problem construct.

$$\begin{aligned} \underset{z}{\text{maximize}} \quad & z = 3x_1 + 2x_2 + x_3 \\ \text{subject to} \quad & x_1 + x_2 + x_3 <= 100, \\ & 0 <= x_1 <= \textit{Requested bandwidth Gold}, \\ & 0 <= x_2 <= \textit{Requested bandwidth Silver}, \\ & 0 <= x_3 <= \textit{Requested bandwidth Bronze} \end{aligned} \tag{3.1}$$

The input to the solver is the estimated demand for the next iteration, shown as x_1 , x_2 and x_3 in Equation 3.1. The estimated demand is calculated as the current demand added with the current derivative of the demand curve. Given these values, the solver determines the prioritization between the traffic classes under the constraints given in Equation 3.1, and allocates a certain bandwidth to each class. However, it was deemed unrealistic to cut off a traffic class entirely and therefore the output from the solver was checked after each control loop. If the allocated bandwidth was lower than 5 Mbit/s for bronze or silver, they were given 5 Mbit/s connections anyway to uphold a small stream of traffic. This resulted in a maximum allocation of bandwidth of 110 Mbit/s if gold requested 100 Mbit/s, thereby ensuring a buffer of 10 Mbit/s if gold requests spiked. After the algorithm has calculated a new configuration, it is then sent to the virtual switch through the SDN controller agent (SDNc agent), of which an explanation follows below.

SDNc Agent In order for the SDN controller to be able to observe and then configure the virtual switch, an SDNc agent was developed. This SDNc agent was placed on the same VM as the OVS and acts as an intermediary between the SDN controller and the switch. During the observation step, the SDNc agent receives a metrics request from the controller and then parses the data that is extracted from the switch. This data is converted to JavaScript Object Notation (JSON) format before being returned to the SDN controller. During the configuration step, the SDNc agent receives a POST request with the new configuration to be applied and then converts this to commands which are executed in the OVS. The SDNc agent did not run in the Kubernetes environment through a pod, it was executed directly on the OS of VM4. In the environment depicted in Figure 3.6, the SDNc agent is labeled F.

Establishing Observability To observe the behaviour and resource utilization of the cluster the open-source instrumentation framework Prometheus was installed. Prometheus enables observability of a Kubernetes cluster on metrics such as the CPU, memory and network usage for each pod and node in the cluster. This provides a great overview of the system

and could be used by a system administrator to monitor the network load in a real world scenario. In our case Promtheus was initially envisioned as the metrics extractor for the data on which the SDN controller would base its control decision. However, upon closer inspection of the tool it was discovered that the precision was too low for the control loop. Mainly this issue revolved around discrepancies between the sent data measured at the pod level and the received data measured at the virtual switch. These discrepancies seemed to originate from Prometheus being a bit slow to update its metrics as compared to the OVS. As we needed a system where packet drop rates in the range of 1‰ would be important, the precision from Prometheus was deemed unsatisfactory.

In addition to Prometheus the open source analytics and visualization tool Grafana was installed. Grafana is mainly used to better visualize the Prometheus data and to make nice dashboards. While these tools are not used by the control system itself, they remain important to a system administrator as well as the development process. Furthermore they would provide the ability to monitor the SDN Controller and its performance in a real access network. However, as the two tools are not part of the control loop, they are not shown in the figure for phase three.

To simulate network traffic, the tool iPerf was utilized. iPerf is a networking tool that lets you generate traffic between two points on an IP network with configuration options such as protocols and traffic levels.

The Design Process

The second phase was the longest as the SDN controller involved several different interacting components. The developed API proved to be very useful in debugging and testing the functionality of the controller. This was used in tandem with a REST API testing program called Postman. With this we could observe the metrics that were returned by the SDNc agent at the virtual switch. In addition to enabling the collection of metrics it was quite time consuming to make sure these were stored correctly in the SDN controller's memory. Furthermore the algorithm development demanded a lot of iterations as its behaviour needed to be both observed, evaluated and tuned. To test the SDN controller traffic was manually generated and sent between the pods and the virtual switch.

3.5.2 Relevance

As in the previous phase, many of our design choices were influenced by Ericssons business context. Deploying our SDN Controller through the use of Docker and Kubernetes was a decision made together with engineers at Ericsson. In large part due to Ericsson already having adopted Kubernetes in their current products [10]. Regarding the other components, such as iPerf, Prometheus, Grafana and Flask, they were also validated through meetings with Ericsson to make sure they fit the intended scope. Some were tools that Ericsson already used internally and some were open-source software tools with high adoption rates.

The choice of the PuLP linear programming solver was found suitable based on both it being distributed by the open-source organisation Coin-or and it being maintained by an active community providing well explained examples [33].

3.5.3 Rigor

The high level behaviour exhibited by the SDN Controller is inspired by Control Engineering as presented in Section 2.2.5. The foundation of control engineering is utilizing sensors to observe the system, calculating adjustments and then applying those adjustments to the system. This corresponds closely to our control loop of observation, calculation and configuration.

Regarding the use of the Flask Micro Web Framework, researchers are citing it as simple, light and not dependent on many external libraries [58]. This was deemed a good fit to our relatively basic needs.

In [55] researchers argue that SDN Controllers do not scale well when the network infrastructure being controlled gets too large. To solve this they suggest building SDN Controllers with a micro-service architecture coupled with a container orchestration tool such as Kubernetes. Given that our network infrastructure is very small, we decided to not adopt the whole micro-service architecture but to still deploy our SDN Controller through Docker and Kubernetes to enable future development.

Researchers in [76] propose a virtual resource allocation model based on constraint programming. This approach enables the model to make trade-offs between performance goals and the cost of the required resources. This approach was considered a promising venue for our thesis on the basis of similar problem structure. However, as we sought to find an optimal solution rather than a feasible one, linear programming was deemed more suitable. Linear programming has a very similar problem structure to constraint programming, but puts more emphasis on optimization of the objective function. It should be noted that our own linear programming solution is more of a proof of concept model rather than an attempt at replicating the complexity of the model in [76].

3.6 Phase 3: Network Traffic Generation

In phase 3 the goal was to build a simulation script with which simulation of traffic scenarios according to our use cases could be automated. Thereby testing both our network infrastructure built during phase 1 and our SDN controller built during phase 2. Each simulated connection can be thought of as an abstraction of a user equipment (e.g. a mobile phone) connecting to the network. Then having its traffic forwarded throughout the network. A simulation run would entail many different user connections with different traffic classes connecting and disconnecting.

3.6.1 Design

Traffic Generating Pods To simulate traffic throughout the project a tool called iPerf has been used. iPerf is a tool for network performance measurement and tuning, written in C. With the tool installed, a server and client can be started from the command line. Upon starting a server, the iPerf instance will listen for incoming connections on a specified port. When starting the client a host address and port, connection duration, bandwidth and transport protocol type (UDP/TCP) has to be defined. Thereafter the client will send IP traffic to the specified host for the specified duration. With this tool we could manually start

connections between pods in phase 1 and 2, but during this phase the goal was to automate the setup and initialization of connections.

To automate the start up of an iPerf client connection from inside a pod, a python-script was developed. Upon starting the script inside a pod, it reads its local environment variables in which each iPerf argument has been specified. Thereafter the connection is started. This script was packaged in a Docker image, thus enabling it to run in a pod. The iPerf client pods are shown in Figure 3.7, labeled G.

The Simulation Script The purpose of the simulation script is that it shall start pods containing the iPerf python-script detailed above. The basic functionality of the simulation script can be broken down into configuration loading and Kubernetes job creation. When the simulation script is run from the command line on one of the vDUs a configuration filename is passed as an argument. From this configuration file the script will receive a number of pods to be started on each of the two VMs representing vDUs. Apart from the number of pods the configuration file also specifies bandwidth of the iPerf connection starting in the pod, the network ("gold", "silver", "bronze") to connect to and lastly the host IP of the iPerf server. This server is listening for connections on the other side of the virtual switch.

The script parses over the configuration file and starts jobs in Kubernetes with the Kubernetes Python API. The jobs in turn start pods running the iPerf script described in section 3.6.1. Once the iPerf script in a pod is finished, the job is done and the pod will be removed. In this way the behaviour is similar to that of a connecting user equipment. The simulation script is run from VM 1 and is shown in Figure 3.7, labeled H.

The Server Side Script Before the client pods can begin sending data a corresponding server instance must be started. A limitation of iPerf is that a server instance can only maintain a connection with one client at a time. For this reason a server instance for each client pod must be started. This is done with a Python script that runs on the same VM as the virtual switch, see Figure 3.7, label I. At startup the simulation configuration file is read and a server instance is started for each pod that will send data during the simulation run. In the script the IP address and port on which to listen for incoming connections is specified.

3.6.2 Relevance

Creating an aggregated network traffic stream through multiple small traffic streams was agreed together with Ericsson to be the most realistic way to portray the traffic in our network. Ericsson recommended the use of iPerf for this purpose. To create a multitude of different iPerf clients providing a specified amount of network load, Kubernetes Jobs was a natural choice due to our already available Kubernetes environment. A Kubernetes job creates a pod and keeps it alive until the task specified for that job is completed [23]. In our case a completed task was a particular network load applied for a certain duration of time.

3.6.3 Rigor

iPerf was used as the network performance tool partly due to Ericsson recommending it but also since it is frequently used in research investigating network performance, e.g. [51] [59]

[43]. Furthermore it is primarily developed by ESNet and the Lawrence Berkeley National Laboratory which lends it stable and continuous support.

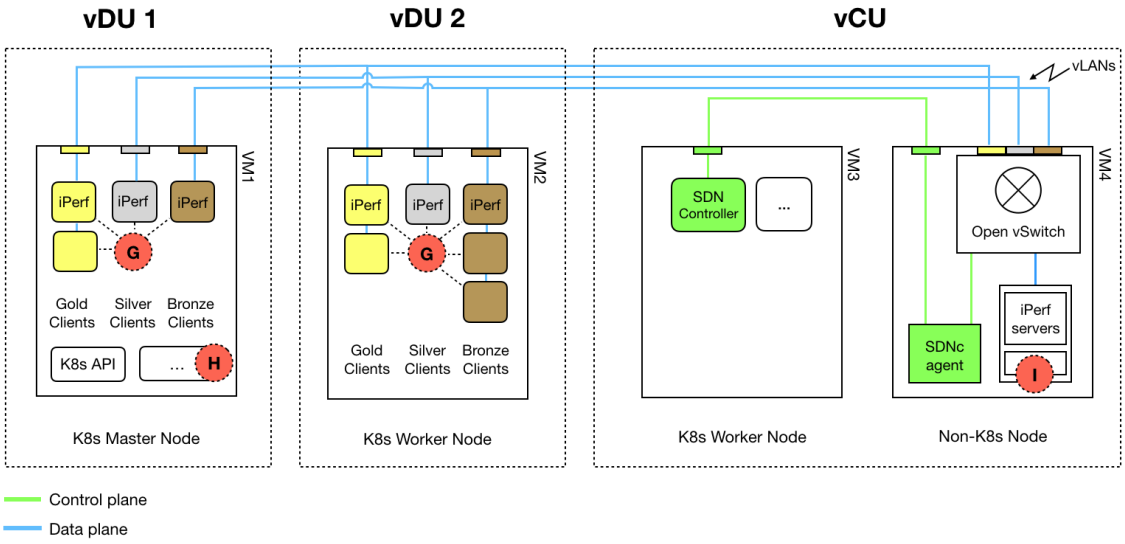

Figure 3.7: The system in phase 3, the iPerf-servers on the virtual switch VM and the iPerf pods on VM2 are added.

3.7 Design Evaluation

The design science process involves continuous evaluation of the design artifact. Inherent in this process is to ensure utility and effectiveness. Four different evaluation strategies are proposed in [52], these are Observational strategies, Analytical strategies, Experimental strategies and Testing strategies. For the evaluation of our simulation environment the Experimental strategy was adopted, specifically the subcategory Simulation. This strategy depends on the execution of the artifact with artificial data to judge its performance. Due to our iterative process of development, these simulations were carried out almost daily to assess the functionality of the system. If a newly added artifact did not meet requirements it was discarded in favor of another implementation. These requirements could be, for example, in the form of reliability or speed constraints.

3.8 Resulting Environment

After completing the three phases of development, the resulting simulation environment is shown in Figure 3.8. In the spirit of open source our implementation is available under an Apache License at Github through [11]. As have been mentioned earlier, the finished environment should be capable of simulating network traffic, observing the bandwidth requested per traffic class and then configuring the settings of the OVS to fit the specific traffic scenario. The steps of the simulation run are shown in both Figure 3.8 and Figure 3.9, with corresponding numbering in blue. Figure 3.8 shows the entire simulation environment, while Figure 3.9

shows the SDN controller in relation to the flow of traffic through the virtual switch. Before the control loop starts, the simulation script (1a) and the server side script (1b) are started. Thereafter the simulation script creates Kubernetes jobs and subsequently starts the traffic generating pods (2) in vDU 1 and 2.

The observation and configuration are part of the SDN controller's control loop, which can be divided into three steps: collect network metrics (3), calculate bandwidth configuration (4) and apply new configuration (5).

Collect Network Metrics The SDN Controller starts the control loop every three seconds, an interval chosen due to it being the lowest with which consistent system metrics could be obtained. The first step of the control loop is to find out the current network load. To do this an API call is sent to the SDNc agent (3a). The data is fetched from the OVS through a shell script, requesting the number of bytes and packets received per traffic class. When the SDNc agent has retrieved the data from the OVS, it sends it back to the SDN Controller (3b). In the example in Figure 3.9 one can see that the SDN controller observes (3) an increased demand in gold traffic, with a total demand surpassing the total available bandwidth.

Calculate Bandwidth Configuration When the SDN Controller receives the current bandwidth utilization from the SDNc agent it utilizes these metrics to start the allocation algorithm. The algorithm then provides each traffic class with bandwidth in a prioritized manner (4). The prioritization only comes into play if the aggregated request is larger than the total available bandwidth resources. Otherwise each traffic class is allocated what its current request is, plus the derivative over the last time step. This addition results in the predicted bandwidth demand for the next time step. In the example in Figure 3.9 the SDN controller calculates a new allocation from the observed increase in gold traffic.

Apply New Configuration When the configuration has been calculated it is sent back to the SDNc agent via an API call (5a). The SDNc agent receives a new OVS configuration from the SDN Controller. The configuration specifies exactly how much bandwidth each interface, belonging to a certain traffic class, shall be assigned. This assignment is done via yet another shell script modifying the OVS configuration database (5b). When this is done the SDN Controller and Agent have finished their task and will wait for the next control-loop iteration. In the example in Figure 3.9 one can see that the applied configuration (5) lets the gold traffic flow freely, while slightly limiting the throughput of silver and maximally limiting the throughput of bronze.

This concludes the creation of the environment and the description of its functions. However, evaluating its performance on challenging traffic scenarios will be detailed in the next chapter, System Assessment.

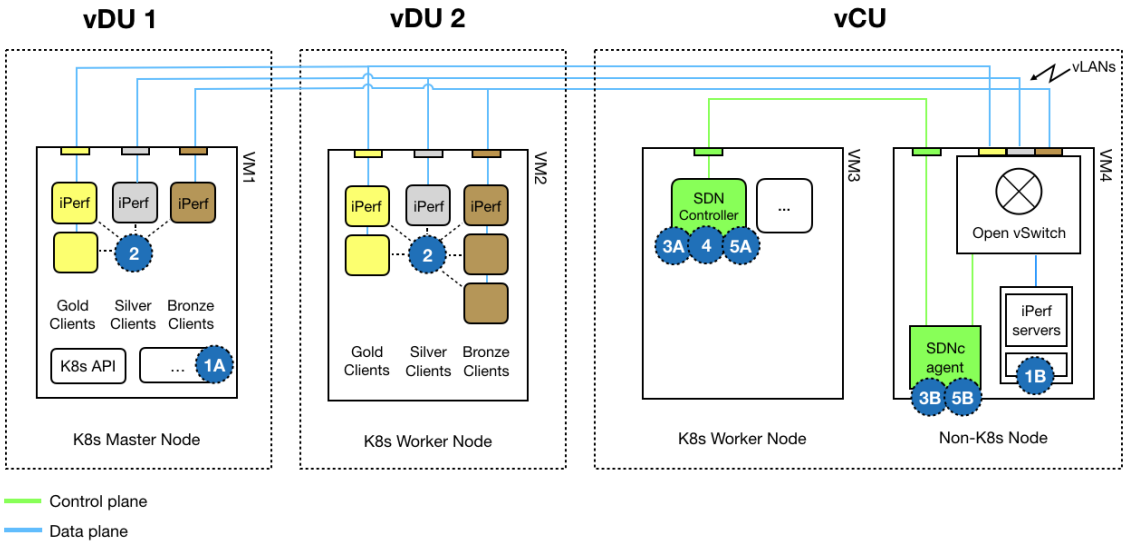

Figure 3.8: An illustration of the final simulation environment, with the steps of the simulation run labeled in blue.

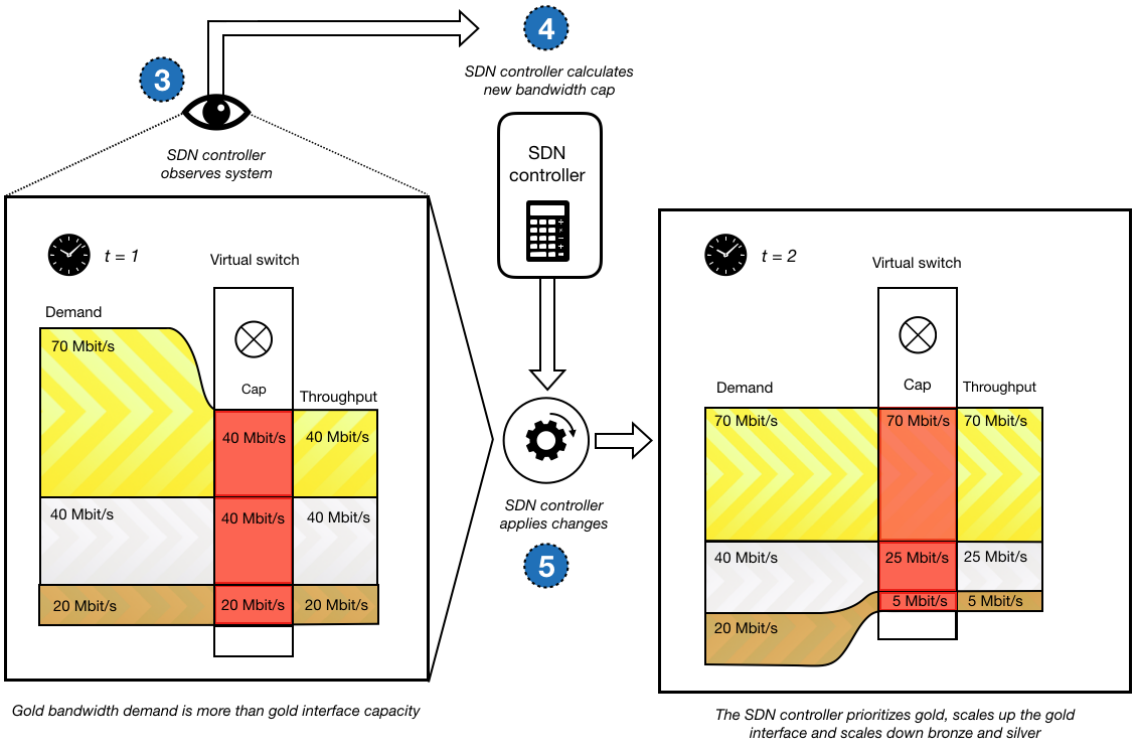

Figure 3.9: An illustration of the SDN controller's response to adapt to an increase in gold bandwidth demand, resulting in lowering of the cap for silver and bronze traffic while increasing the cap for bronze.

Chapter 4
System Assessment

To assess the SDN controller's performance and potential for network resource savings, a set of experiments were carried out. In order to do this a number of use cases were developed together with engineers at Ericsson. A use case is more concretely a specific traffic scenario. To construct experiments from these use cases we took inspiration from the experiment process described in Wohlin 2012 [73]. In the following sections the experiment methodology and process will be elaborated on.

4.1 Experimental Methodology

To assess the performance and function of our simulation environment an experimental methodology was adopted. Experiments can be conducted when it is possible to control the environment in which the experiment is to be performed. Thereby having the ability to manipulate certain variables to observe the changes in others. Extending this concept, there are some prerequisites that limit the choice of research methodology. These are Execution control, Measurement control, Investigation cost and Ease of replication. Execution control was in the hands of the writers and the supervisor at Ericsson. Since the simulation environment was created during this thesis, it was ensured that measurement control was satisfied. Investigation cost was mainly an investment of time where the timeframe was estimated to be sufficient for this project. Furthermore, given that the environment would be available to someone trying to replicate these experiments, the ease of replication would also be high. Through the assessment of these four factors the experimental research strategy was solidified. In addition to this, it was determined that the experiments would be best suited to quantitative research. This is due to the intention of analyzing the effect of the manipulation of an independent variable.

The experiment process can be illustrated as is shown in Figure 4.1 below. The main steps are Scoping, Planning, Operation, and then Analysis Discussion. These steps were adopted from the experiment process of Wohlin et al. [73]. The final step of Analysis Discussion

will be detailed in chapters 4.1.4 System Assessment Results and in the Discussion chapter. These remainder of the steps will be further detailed in the following sections explaining our choices and processes in depth.

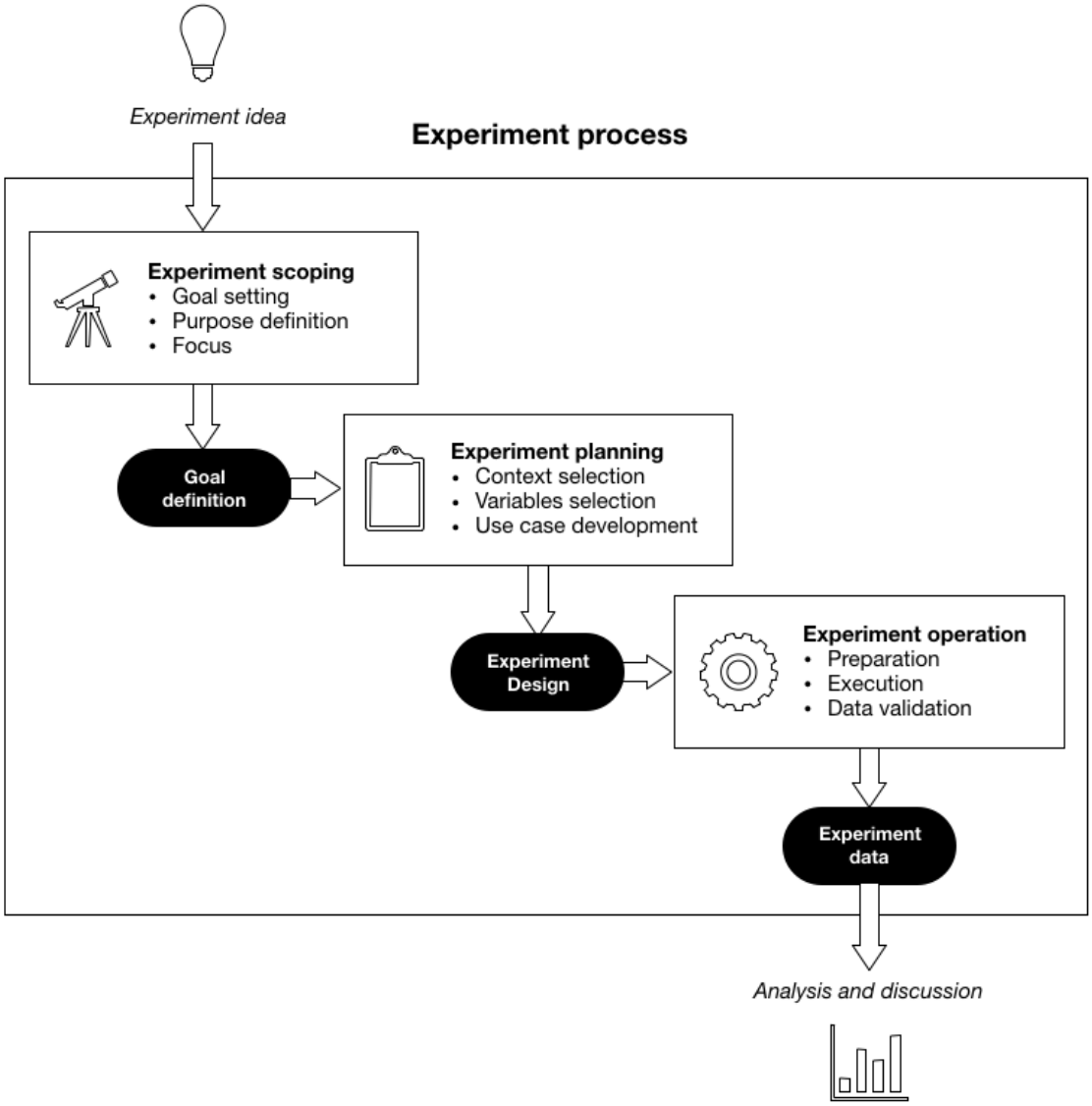

Figure 4.1: The steps of the experiment process.

4.1.1 Experiment Scoping

The purpose of the scoping phase was to define the goal and purpose of the experiments. The phase also included defining the limitations of the experiments in order to utilize time and resources as well as possible. The goal of the experiments was to find the scenarios in which our dynamic SDN controller shows the greatest potential. This was done for the purpose of assessing the simulation environment's usability in different scenarios and identifying components of our system in need of further research. The context of the study was the prepared simulation environment, while the object of study was the SDN controller. The focus of the

Operating system:	Ubuntu 18.04 (64-bit)
Processor cores:	2
Memory (RAM):	2048 MB
Storage:	50 GB

Table 4.1: Configuration for each of the virtual machines

Operating system:	Microsoft Windows 10 Enterprise
Make:	HP EliteBook 840 G5
Processor:	Intel Core i7-8650U CPU 1.90GHz
Memory (RAM):	32 GB
Storage:	1 TB
VirtualBox:	Version 6.1.22

Table 4.2: Laptop configuration

experiment was to observe how traffic variations and combinations of demand from different traffic classes affect the SDN controller's performance. This performance was measured in packet loss and saved bandwidth, with lower numbers being better for packet loss and higher numbers being better for saved bandwidth.

4.1.2 Experiment Planning

In the planning phase it was defined how the experiments would be conducted. This included specifically defining context, variables and subjects. Selection of subjects included defining our use cases together with the engineers at Ericsson. The experiments were designed to quantitatively evaluate the SDN controllers performance in our specific simulation environment.

Firstly the context was defined. The constructed simulation environment needs four different VMs to operate: one for the vCU, two for the vDUs and one for the virtual switch. Each VM was configured according to table 4.1. The VMs in turn ran in VirtualBox on a high performance laptop provided by Ericsson. The basic hardware and software specification for the laptop can be seen in table 4.2.

The SDN controller in the simulation environment was set to collect metrics and reconfigure the system once every third second, as this was the lowest control interval that the hardware could handle while still providing consistent results. The simulations were all conducted in real-time, as such, simulating 60 seconds of user connections takes 60 seconds.

Next the relevant experimental variables were defined. The independent input variable is the traffic patterns representing the use cases. The dependent output variables are the Packet Loss Rate, Average Allocated Bandwidth, and Average Required Static Bandwidth.

Packet Loss Rate is the share of packets lost as compared to the number of packets sent. It can therefore assume values between zero and 100%. Packet loss is presumed to occur only as an effect of control system delay causing congestion, i.e. when a traffic class demand rapidly changes and the amount of allocated bandwidth is trailing behind. In a real system, packet loss can also depend on error in data transmission. In our virtual simulation environment packet loss due to transmission error was assumed to be zero.

Use case 1:	The silver traffic class peaks and then decreases again
Use case 2:	The silver traffic class peaks slowly and then decreases again
Use case 3:	All traffic classes spike at different times
Use case 4:	Gold increases which throttles bronze and then silver

Table 4.3: The use cases

The Average Allocated Bandwidth metric is the amount of total bandwidth resources assigned by the SDN controller to handle traffic requests from each traffic class. So if the SDN controller allocated 10 Mbit/s to the gold traffic class for 10 seconds and then 5 Mbit/s for 10 seconds after that, the Average Allocated Bandwidth for the gold class would be 7,5 Mbit/s.

Finally the Average Required Static Bandwidth shows how much bandwidth the system would need to handle the peak load for each traffic class, with a static bandwidth allocation strategy. The peak load static bandwidth allocations were rounded up to the nearest multiple of five to ensure that the peak load could have been handled. As an example, if the peak load for gold, silver, and bronze was 29 Mbit/s each, they would all be rounded to 30 Mbit/s and then summed up to 90 Mbit/s.

Defining Use Cases As information about real 5G transport network loads was unavailable, relevant and interesting use cases were crafted with the supervisor at Ericsson. Real world traffic patterns could likely depend on factors such as radio tower placement or time of day. For example the bandwidth demand for different traffic classes would probably differ a lot between urban and rural areas, or between residential or business areas. Such variance was not captured in the use cases.

The scenarios were instead selected for their qualitative characteristics where emphasis was laid on capturing isolated dynamic behaviours that could occur in a real system. Certain scenarios focused on bandwidth demand that shifted between different traffic classes over time. Others were focused on scenarios where the total demanded bandwidth surpasses the capacity of the network infrastructure. The discussions resulted in the use cases specified in 4.3. A detailed explanation of each use cases can be found in Appendix A

The number of use cases was decided through discussion with our supervisor at Ericsson. It was concluded that these four scenarios captured the key dynamic behaviours we wanted to investigate, e.g. fast and slow traffic acceleration, multiple traffic classes accelerating after one another and exceeding the available network infrastructure. Each use case was defined with a starting bandwidth demand for each traffic class. Thereafter it was specified how the bandwidth demand should vary over time, including an expected system response. The max bandwidth demand was limited well below the capacity of our hardware resources, in order to avoid the interference problems discovered by Kim et al. in [53], as described in Section 1.3.1.

4.1.3 Experiment Operation

The operational phase of an experiment starts with the preparation of the experiment, this is based on the above detailed experiment design. It then proceeds to the execution of the experiments and ends with validation of the obtained data.

The largest part of the preparation stage consists of setting up the configuration of the simulation environment. This entails starting the VMs, configuring the ports of the OVS to enable traffic separation and starting the SDNc agent. Furthermore a server side iPerf-script is started on the VM that contains the OVS and SDNc agent, this script is use case specific to account for all of the intended traffic streams.

The execution stage is divided into three parts. The first step is that the SDN controller is deployed in a pod and connectivity with the SDNc agent is established. The second step consists of starting the client side iPerf-script, which is also use case specific. This script dictates what traffic streams are generated and for how long. The third and final step is to extract the experiment data from the SDN Controller after the experiment is finished. To do this the csv file which contains the simulation data is copied from the SDN Controller pod's file system to the VM file system. Then this file is finally transferred to the host system computer file system.

Data validation must then be performed on the data. This was done through importing it to an Excel workbook and reviewing the raw data for possible defects. Via this procedure faults such as incorrect traffic scenarios or inaccurate OVS settings could be detected.

All the scripts described in this section are available at Github through [11].

4.1.4 System Assessment Results

To analyze and draw conclusions from the data obtained through the execution of the experiments, the data was visualized in time series graphs. This enabled interpretation of the SDN controller's behaviour in relation to the different use cases. The time series variables included in the output data were requested bandwidth per traffic class, throughput per traffic class, bandwidth allocated per traffic class and packet drops per traffic class.

Experiment Results

Use Case 1: The silver traffic class peaks rapidly and then decrease again As have been mentioned above, the detailed specification of the use case can be found in Appendix A. The essence of this use case is that each traffic class has a stable traffic flow to begin the test and then silver-class traffic starts to increase rapidly. This is to purposefully test how the SDN controller performs when traffic demand increases sharply. The throughput, meaning the amount of bandwidth transmitted through the system, is visualized in Figure 4.2. There is a clear spike of silver-class traffic in the middle of the test.

To complement this view another plot is provided, Figure 4.3 shows the SDNC Allocated Bandwidth Resources assigned to the silver traffic class during the test. This is shown as the green area in the plot. It also shows the requested bandwidth by the "users" as the blue line and the throughput that actually passed through the system as the orange line. It is apparent in the plot that the bandwidth cap assigned by the SDN Controller cannot keep up when traffic demand increases suddenly. This results in packet loss, which is illustrated by the vertical distance between the blue and orange line in the plot. As both the gold and bronze traffic demand was stable during the use case no SDN controller intervention was needed regarding them. Therefore these are not shown as figures in the same way as the silver traffic class.

The key metrics described above are presented in Table 4.4.

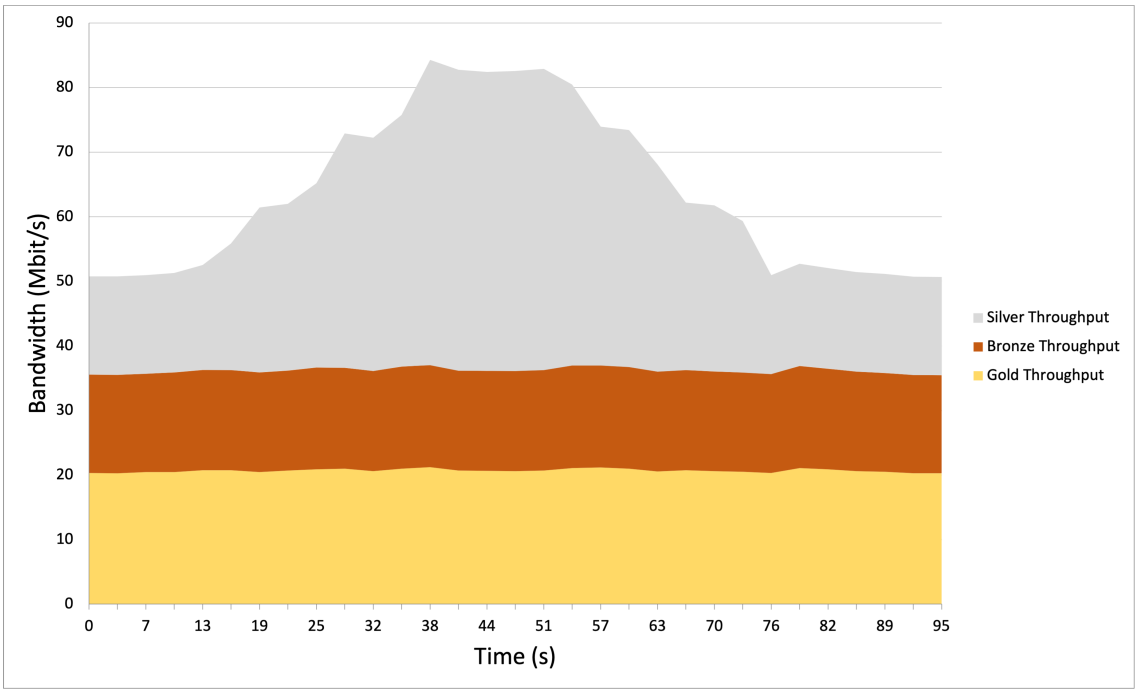

Figure 4.2: Throughput per traffic class for use case 1.

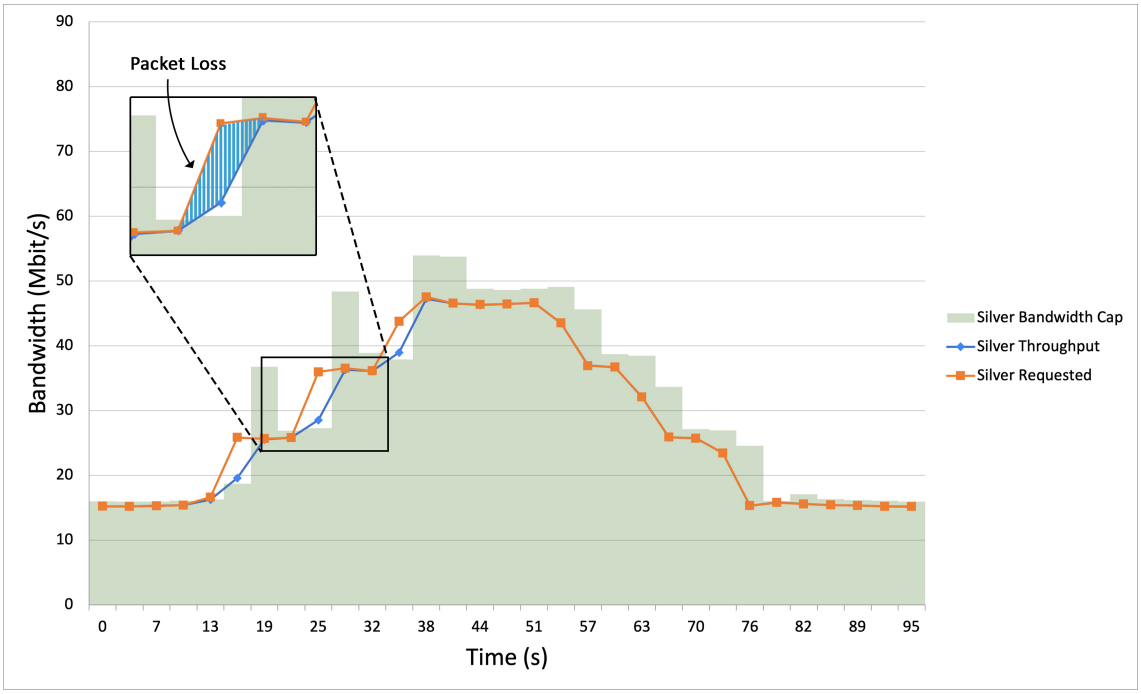

Figure 4.3: The exhibited behaviour SDN controller concerning the silver traffic class in use case 1.

Total amount of data sent	5980 Mbit
Average Required Static Bandwidth	95 Mbit/s
Average Allocated Bandwidth	67 Mbit/s
Average Saved Bandwidth	28 Mbit/s
Percentage of saved bandwidth	29%
Packet Loss Rate Gold	0%
Packet Loss Rate Silver	2.3%
Packet Loss Rate Bronze	0%

Table 4.4: Use case 1 metrics

Use case 2: The silver traffic class peaks slowly and then decreases again This use case has a very similar structure to the one above, the silver traffic class increases while the gold and bronze traffic is stable. The difference between this use case and the previous one is that the acceleration of the increase has been halved. The point of this is to investigate the effect of traffic acceleration on the performance of the SDN Controller. The throughput for each traffic class over time is shown in Figure 4.4 below.

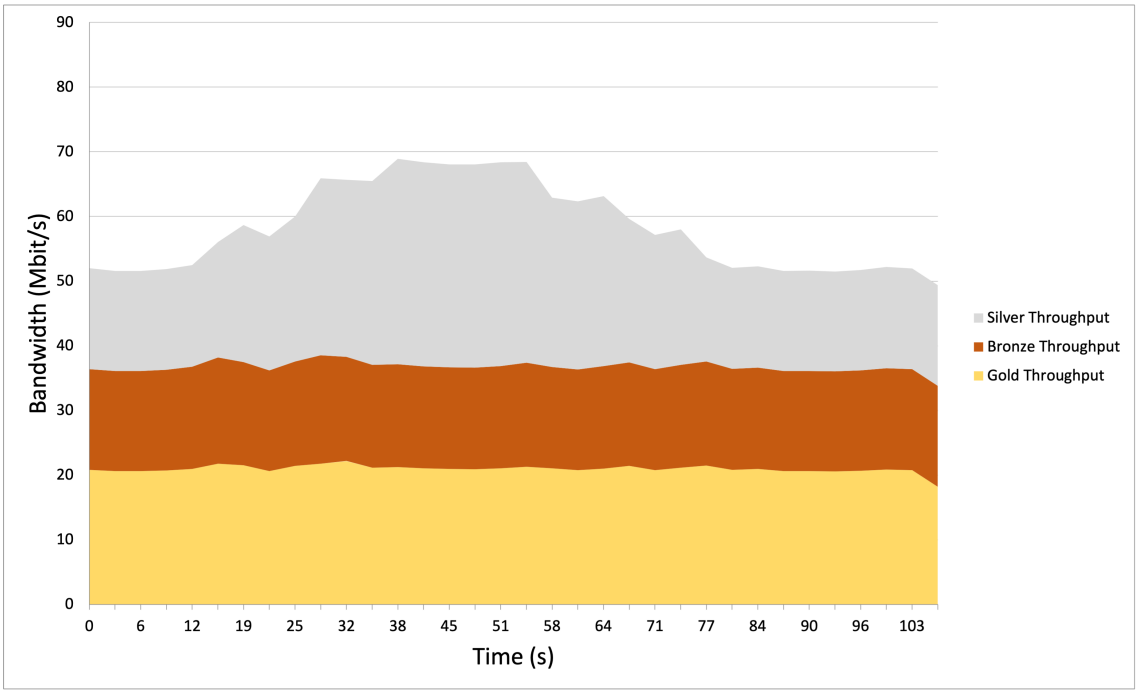

Figure 4.4: Throughput per traffic class for use case 2.

As in the previous use case, Figure 4.5 shows the SDNc Allocated Bandwidth Resources assigned to the silver traffic class during the test, the requested bandwidth and the throughput.

The vertical distance between the requested bandwidth and the throughput is smaller in this use case. This also results in a lower Packet Loss Rate for the silver traffic class, as can be seen in Table 4.5 below.

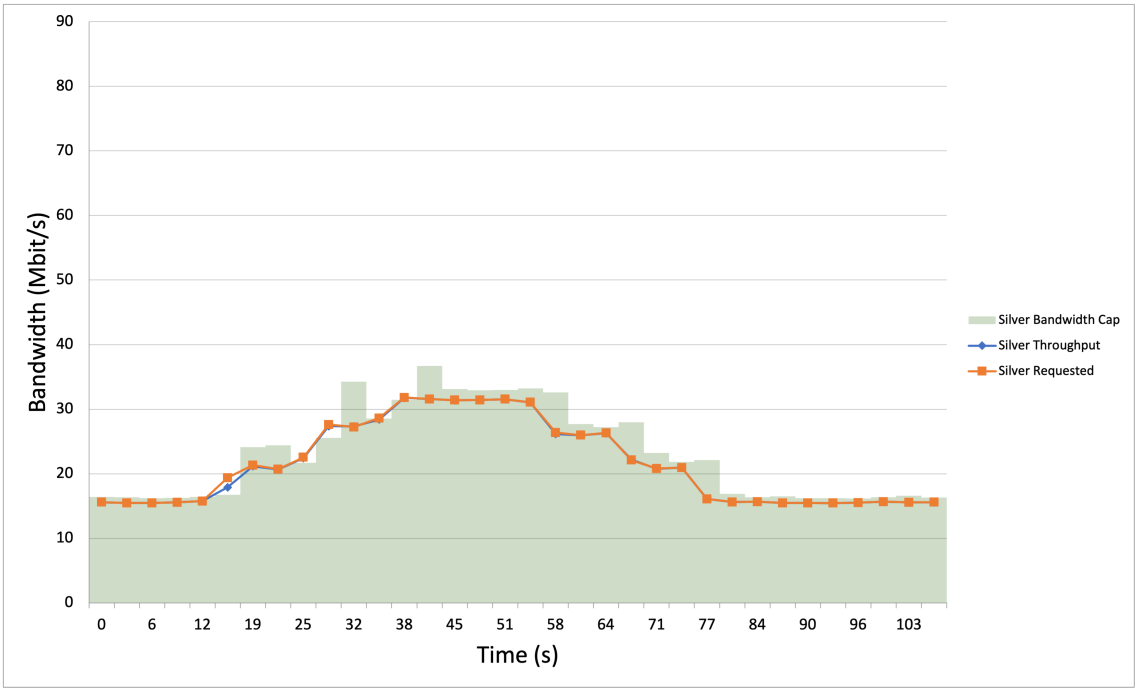

Figure 4.5: The exhibited behaviour SDN controller concerning the silver traffic class in use case 2.

Total amount of data sent	5950 Mbit
Average Required Static Bandwidth	80 Mbit/s
Average Allocated Bandwidth	61 Mbit/s
Average Saved Bandwidth	19 Mbit/s
Percentage of saved bandwidth	24%
Packet Loss Rate Gold	0%
Packet Loss Rate Silver	0.4%
Packet Loss Rate Bronze	0%

Table 4.5: Use case 2 metrics

Use case 3: Traffic classes spike at different times Use Case 3 exhibits a different traffic pattern than the two previous use cases. The fundamental behaviour of this traffic scenario is that each traffic class peaks after the other. Starting with the gold traffic class, then silver, then bronze. This scenario was created to see how the SDN controller could re-allocate bandwidth when demand shifted. The traffic pattern is visualized in Figure 4.6.

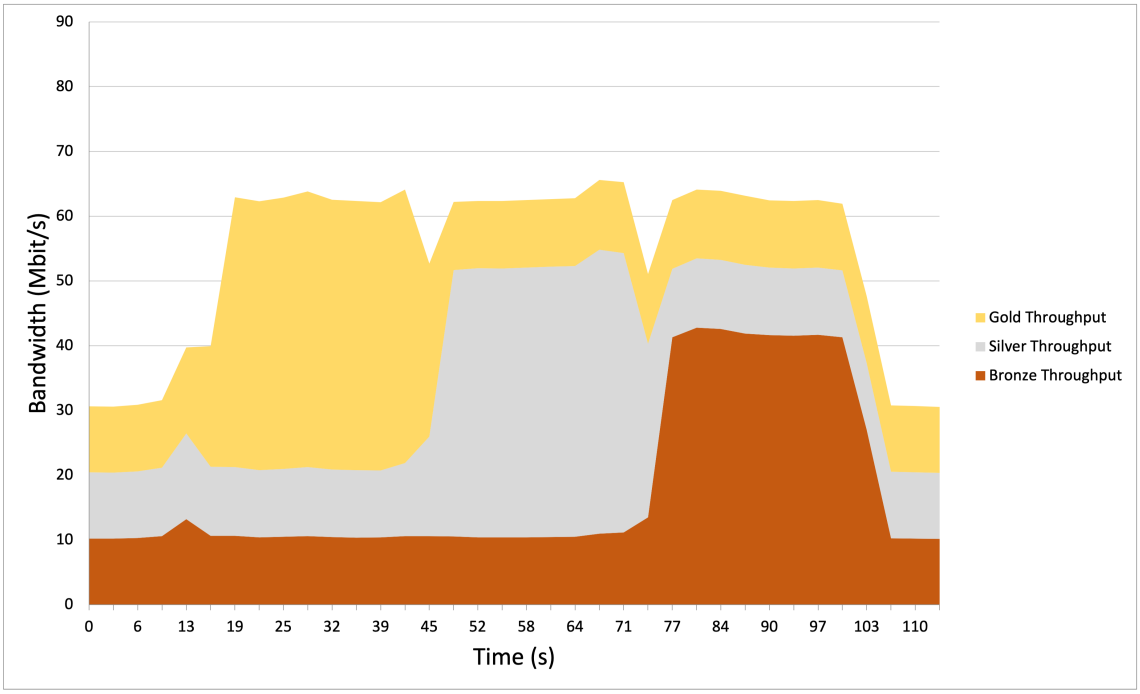

Figure 4.6: Throughput per traffic class for use case 3.

To show the behaviour of the SDN Controller in this scenario Figure 4.7 is provided. It displays clearly the tendency of the SDN Controller to overshoot the requested bandwidth when there is a sudden traffic surge. This is apparent through the sharp peak of the assigned bandwidth cap at around 19 seconds in the plot. There is also a noticeable vertical distance between the requested bandwidth and the throughput at around 16 seconds, which indicates packet loss. Only the gold traffic class is displayed in this figure, this is due to both the other traffic classes exhibiting almost exactly the same behaviour.

As in the previous use cases, the key metrics for Use Case 3 are presented in Table 4.6.

Total amount of data sent	6070 Mbit
Average Required Static Bandwidth	135 Mbit/s
Average Allocated Bandwidth	59 Mbit/s
Average Saved Bandwidth	76 Mbit/s
Percentage of saved bandwidth	56%
Packet Loss Rate Gold	3.0%
Packet Loss Rate Silver	3.2%
Packet Loss Rate Bronze	3.7%

Table 4.6: Use case 3 metrics

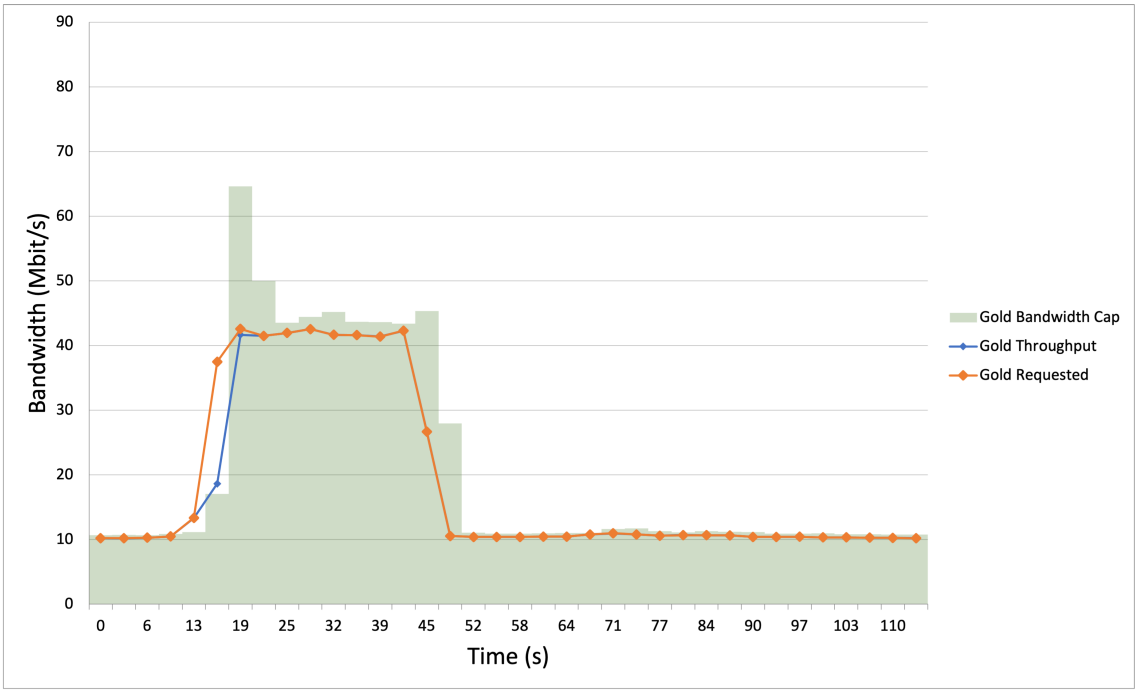

Figure 4.7: The exhibited behaviour SDN controller concerning the gold traffic class in use case 3.

Use case 4: Gold increases which throttles bronze and then silver

In this use case the gold, silver and bronze traffic classes all request a bandwidth well within the infrastructure capacity to begin with. Then the gold traffic increases to the point where first bronze is throttled down, meaning that it receives less bandwidth than it has requested, and then silver traffic is also throttled down. This is due to the fact that total bandwidth demand from the three traffic classes exceeds the available network infrastructure.

Both the silver and bronze traffic class requests around 20 Mbit/s for the duration of the simulation. As can be seen in Figure 4.9 the SDN Controller down throttles the allowed bronze throughput at around 40 s. This can be seen as the purple line in Figure 4.9 declines to around 5 Mbit/s when the gold traffic request increases.

In the results for this use case, bandwidth metrics were intentionally excluded. This is due to the fact that the use case was constructed to test the SDN controller behaviour when bandwidth resources were depleted. In such a scenario presenting the saved or used bandwidth would be misleading, as choking a traffic class is not synonymous with saving that bandwidth. Therefore the results presented in Table 4.7 only include Packet Loss Rate to show the effect of traffic class prioritization.

Packet Loss Rate Gold	1.6%
Packet Loss Rate Silver	30%
Packet Loss Rate Bronze	93%

Table 4.7: Use case 4 metrics

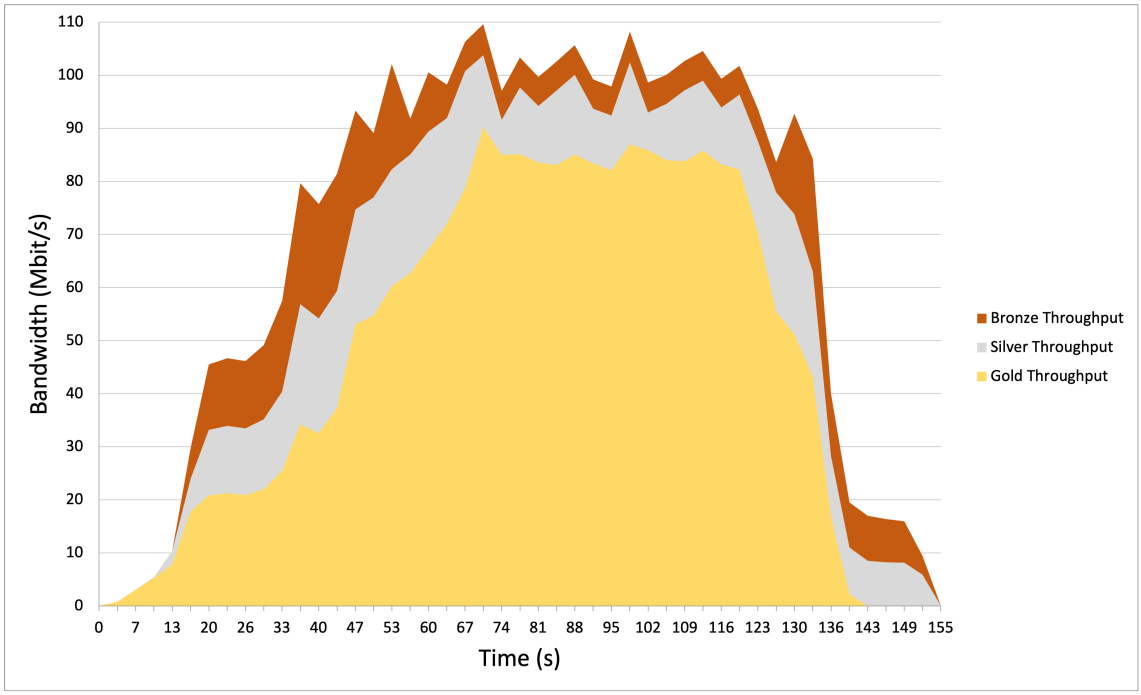

Figure 4.8: Throughput per traffic class for use case 4.

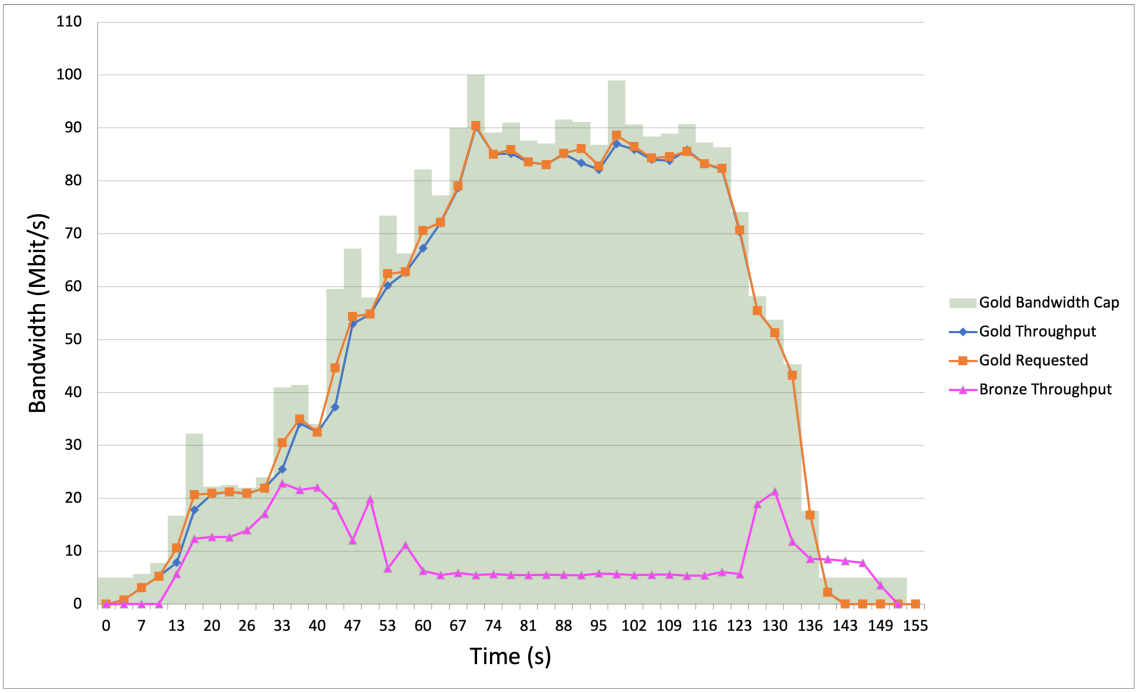

Figure 4.9: The exhibited behaviour SDN controller concerning the gold and bronze traffic classes in use case 4.

4.1.5 Summary of Results

Comparing the results from the different use cases, a couple of important trends can be summarized as follows. Use case 1 and 2 have a very similar structure, the main difference being that the silver traffic demand increases more quickly in Use case 1. This results in a higher packet loss rate for Use case 1 of 2,6% versus 0,8% for Use case 2. This indicates that a rapid traffic demand increase results in more dropped packets.

Furthermore Use case 3 exhibits the largest bandwidth savings by far. This is due to the fact that the demand of all three traffic classes peak over the duration of the simulation. If the system would statically allocate bandwidth for the interfaces “gold”, “silver” and “bronze”, each interface would need to accommodate the peak bandwidth demanded by their respective traffic class. This would result in a large amount of overprovisioning, compared to dynamically allocating bandwidth to each traffic class from a common resource pool.

Use case 4 exhibits a down throttling of the lower priority traffic classes, which is indicated by the large percentages of packet loss for the silver and bronze class. This was the intended behaviour of the system as bandwidth resources were exhausted. If such a scenario occurs in a real system, the infrastructure capacity has probably been underestimated.

Chapter 5
Discussion

In this master thesis we set out to explore how cloud technologies and software defined networking could be combined to improve the 5G transport network. To approach this complex topic within the given timeframe the research effort was limited to dynamic network bandwidth allocation. This focus was selected based on the fact that today's telecommunication networks mostly rely on static overprovisioning of bandwidth. While this strategy is good for providing high quality of service, it also leads to underutilization of network resources such as routers and switches. Looking into the future, intelligent utilization of the available network resources might become increasingly important as both the number of users and average used bandwidth per user increases.

As has been evidenced by the structure of this thesis, two distinct areas of investigation have been put forth. The first part focused on building a cloud based simulation environment, the second part focused on testing such an environment. The discussion will be partitioned in the same way, discussing our results in relation to RQ 1.1 and 1.2 first, then RQ 2.1.

5.1 Creation of the Simulation Environment

5.1.1 RQ 1.1 Building the Simulation Environment

The first research question, RQ 1.1, focused on how we could develop a cloud based simulation environment representing the essential components of a SDN controlled vDU to vCU transport network. While some technologies, such as the container orchestrator Kubernetes and virtual machines, were already determined to be important components, little had been decided on how these would interconnect. Deciding the rest of the components and ensuring their functional interaction was our main focus for this research question. The implementation process has been detailed in Chapter 3 and the resulting environment presented in 3.8.

The following discussion will center around the strengths and possible improvements of the resulting environment and its design process.

Implementation

The use of open-source components in this thesis is considered as a strength as it enables anyone to build upon our work and pursue further research. Furthermore it is in line with Ericsson's wish to contribute to the open-source community. Another strength that the resulting simulation environment exhibits is the ease of which new algorithms and use cases could be introduced.

The SDN controller algorithm is modularly separated from the rest of the code base and could be easily exchanged for another solution. The only requirement is that the algorithm takes the current bandwidth load as its input parameter and outputs a new switch configuration. Adding another parameter such as packet drops would not be very time consuming since the metric is already stored in the SDN Controller. On the other hand integrating latency as a parameter in the algorithm would take more effort due to it not being a part of the observation process in the current environment.

The construction of new use cases only requires a new configuration file in JSON format, where any type of traffic flow could be specified. This enables quick adjustments and the possibility of exploring new unseen traffic scenarios with little additional effort. If one would obtain real traffic data the use cases could also be configured to mimic the traffic patterns, and as such test the SDN controller under more realistic conditions.

Furthermore the use of Kubernetes and Docker enables flexibility and scalability if a larger simulation is desired. As both these softwares are part of many DevOps toolboxes, their characteristics would be familiar and if desired they could be integrated into a continuous integration/continuous deployment (CI/CD) pipeline for quicker development of the simulation environment.

In addition to the specific implementation details mentioned above, an interesting aspect of the development process was the Design Science methodology. The structure and the different evaluation cycles the methodology provided were beneficial to our overall development process. It prompted valuable rigor and relevance discussions regarding individual components and overall design choices. The fact that iterations were an integral part of the process also contributed to quick feedback loops and thereby enabling adjustments.

An aspect of the development process that could have been improved was an earlier investigation of comparable solutions. The reason this was difficult to achieve was that our knowledge of the solution space was limited to start with. This issue was alleviated partly by our connection to Ericsson but it could still have been an improvement to put more effort into it. Furthermore more emphasis could have been placed on the Rigor cycle incorporated in the Design Science methodology. Due to our close connection to Ericsson a lot of focus was put on the Relevance cycle. A more evenly distributed attention could have saved time and effort in hindsight.

Comparison to Previous Work

Examining our resulting simulation environment a lot of influences can be seen from previous work within the field. In line with researchers in [71] we pursued a SDN Controlled band-

width allocation strategy through an OVS, though in contrast our allocation was dynamic instead of static. This implementation was more aligned with [62] where dynamic allocation was implemented through Mixed Integer Linear Programming. They did not however utilize an OVS to carry out their calculated allocation. Furthermore inspiration was drawn from [76] as they employed constraint programming to solve complex quality of service trade offs. In hindsight it would have been interesting to try to incorporate more quality of service aspects in our environment. Unfortunately time was limited so bandwidth and packet losses were the ones included. Furthermore we had a larger focus on containerization and container orchestration than closely related work in the field. This was partly due to Ericsson's vision of implementing these technologies in the Cloud RAN and partly due to researchers in [55] advocating for a containerized SDN Controller. Our environment took this one step further and used containerization techniques to generate the traffic flows as well. Many parts of our solution were inspired by previous research but the combination of components presented in this thesis does not seem to exist in related literature at the moment.

5.1.2 RQ 1.2 Algorithm and QoS Traffic Class Prioritization

The second research question concerns how the network resources in the simulation environment could be prioritized between different QoS traffic classes. To make the QoS prioritization possible three parts were needed: an easily reconfigurable network switch, reliable metrics collection on current network demand, and an algorithm for calculating new configuration according to QoS priority.

Open vSwitch was singled out as one of the most capable open-source virtual switches. The switch could be reconfigured with very short time intervals, which was a requirement for the near real-time prioritization of QoS traffic. It also provided reliable metrics on received and transmitted bytes that could be used in the control loop. This metrics collection at switch level was superior in precision as compared to the Kubernetes tool Prometheus that was initially envisioned.

In a real world scenario multiple QoS requirements would have to be considered simultaneously when modifying a network element such as a switch. The configuration of the switch would affect factors such as bandwidth, packet drop and latency. The allowed configuration options would in turn be governed by constraints defined in the current service level agreement between the teleoperator and the telecommunications company. If a "gold" service tier guarantees 100Mbit/s connections with a maximum latency of 50 ms and a packet drop rate of 1‰ the switch should not be modified in such a way that the service level is at risk of being broken.

With QoS requirements that could be expressed as constraints, linear programming was deemed a good fit for calculating the optimal switch configuration. However, the goal for this part of the project was to develop a proof-of-concept algorithm rather than an optimal solution. Algorithm development is a very complex topic, to the point that algorithm development for this type of environment would be worthy of its own thesis. Thus the constraints in our algorithm were limited to bandwidth only. While the problem could be solved without using linear programming, implementing the linear programming solution lay the foundation for easier algorithm improvements in future research work. In a real scenario the

number of different QoS traffic classes is also far greater than the three used in our simulation, with 26 different standardized by 3GPP [57]. In such a context the usefulness of linear programming is also further increased.

5.2 System Assessment

5.2.1 RQ 2.1 Under what conditions can a dynamic network resource allocation improve the utilization of network resources in a data center compared to static resource allocation?

For RQ 2.1 the conditions under which one could dynamically allocate bandwidth with an SDN Controller, without breaking service level agreements, were examined. In the simulation environment system performance would be affected by both the implementation of the SDN Controller and the simulated traffic patterns. Both these variables determine how well the SDN Controller manages to allocate the demanded bandwidth to each traffic class. For the sake of the analysis the SDN controller implementation was therefore assumed to be fixed. As such, the experiment more specifically examined under which conditions our SDN controller managed to control bandwidth without generating packet loss.

Packet Loss

From the plots in 4.1.4 one can see how the SDN controller manages to follow demand through reconfiguration of the interfaces of the virtual switch. Overall the SDN controller follows the demand closely. Most challenging are the sudden accelerations when a new simulated user connection is established. Upon the sudden increase in traffic, packet loss occurs, which can be seen as the difference between the requested bandwidth (blue line) and the throughput bandwidth (red line). The reason for this packet loss is that the allowed throughput is set too low until the SDN controller detects the change. When the traffic volume continues to increase the SDN controller picks up the slack with its derivative component. This derivative component can be more clearly noticed when the increase in traffic stops. Here the allocated bandwidth continues to increase for one time step and thus creates a small peak. This peak can be seen as wasted bandwidth, i.e. bandwidth that was allocated in excess of what was needed.

By comparing use case 1 and use case 2, we can see that faster acceleration of silver's demand results in higher packet loss. From a control theory perspective this is natural, as reacting to faster and larger changes is more challenging. If the real traffic patterns would contain a lot of fast and large changes it would thus be harder to avoid packet loss with dynamic bandwidth allocation, as it would be challenging for the SDN controller to keep up.

Bandwidth Savings

The amount of bandwidth that can be saved with dynamic bandwidth control depends on how the traffic fluctuates. If the network traffic on the "gold" net on average is around 20

Mbit/s but then occasionally peaks to 100 Mbit/s, it would be wasteful to always statically allocate 100 Mbit/s for "gold" traffic. With dynamic bandwidth allocation a lot of bandwidth could be saved in this type of scenario. However, if the average bandwidth on the "gold" net is around 90 Mbit/s with occasional peaks of 100 Mbit/s the bandwidth that would be saved with dynamic bandwidth allocation is significantly less.

Another factor important for determining if dynamic bandwidth allocation is sensible is whether the traffic peaks for different traffic classes occur at the same time or not. The dynamic bandwidth allocation strategy would benefit most if the peaks are spread out in the time domain. This can be seen in use case 3. In this use case "gold", "silver" and "bronze" peak one at a time and each traffic class can achieve a bandwidth of 40 Mbit/s, while the total used bandwidth never exceeds 65 Mbit/s. In this case less switch capacity would be needed with the dynamic allocation strategy, as compared to statically allocating 40 Mbit/s for each traffic class which would require 120 Mbit/s. However, if the traffic classes peak at the same time, with max total peaks of 120 Mbit/s, the dynamic allocation strategy would need as much switch capacity as the static allocation strategy. Concretely this means that the number of switches and corresponding hardware could be reduced with dynamic allocation, under the condition that the peaks of "gold", "silver" and "bronze" don't occur at the same time.

Implications of Saving Bandwidth

The possibility to save bandwidth will depend a lot on the real traffic patterns in the network, but assuming these would be suitable for saving bandwidth with dynamic control, what would the implications be? With less bandwidth needed for the same number of connections, three different scenarios could be envisioned. Telecommunications companies could 1. sell more connections, 2. turn off network devices to save energy, 3. reduce the amount of physical network hardware.

To sell more connections unused bandwidth could be sold cheaply at best-effort, meaning no guarantees to bandwidth, packet loss or latency. A best-effort subscription could be suitable for customers who are not depending on reliable connections and would rather save money than pay for guarantees.

Turning off network devices to save energy could have an impact on the carbon footprint of mobile communications. As Buyya et al. discussed in [46] the general practice for data centers is to leave all networking devices always on. The majority of network elements such as switches, hubs and routers are not energy proportional to the traffic flow and thus consume energy even with no traffic. At times throughout the day when network bandwidth demand is low, dynamic bandwidth control could allow for turning off network elements not in use.

Lastly, reducing the amount of physical network hardware would make it possible to provide the same service at a lower cost to the telecommunications operator. The more hardware units that could share bandwidth the more optimization possibilities there would be to consolidate network connections to fewer switches.

Balancing Bandwidth Savings and Service Level Agreements

Determining whether to use dynamic bandwidth control or not will ultimately come down to the trade off between bandwidth savings and risk of breaking the service level agreement.

To learn more about this trade off the SDN controller's performance on real traffic data must be examined. From this data one could estimate both the potential bandwidth savings and the risk of packet loss under different traffic patterns.

Preferably the cost of both wasted bandwidth and packet loss would be quantified. Then one could compare the cost of one wasted Mbit of bandwidth and the cost of one dropped packet, and as such determine the strategy for different traffic patterns.

Trends in Mobile Communication

When deciding whether to move forward with dynamic bandwidth control or not, the future growth of mobile communication also has to be considered. Examples of questions that need answers are "How much will the bandwidth usage per user grow?", "How much will the number of users grow?", and "How much increase in bandwidth could the current infrastructure handle with the current static allocation strategy?". While some things will depend on usage forecasts, a lot can also be learned by improving the current simulation environment. A few potential improvements will be discussed in the Future Work.

5.3 Threats to Validity

As most research is, this thesis is also subject to threats to validity. These threats can be divided into two types, external validity and internal validity.

External validity concerns the generalizability of the results to other situations. From the experiment results the bandwidth savings and packet loss metrics are subject to this type of validity threat. These metrics were obtained from simulation runs with traffic based on our use cases. As such, how well these results generalize to a real 5G network is uncertain. While the results in the simulation environment were promising, further research on real traffic scenarios would be needed to draw definite conclusions on the potential of the system. Regarding the architecture of the simulation environment, abstractions were made to simulate the essential parts of a Cloud RAN transport network. These abstractions were decided upon together with engineers at Ericsson, but with this said, how similar the behaviours of the abstracted components are to those of a real 5G transport network is hard to deduce. Further research into the generalizability of these component behaviours should be conducted.

Internal validity concerns the validity of cause-and-effect relationships and the presence of any confounding factors. In this thesis there were two main parts of the results, the simulation environment itself and the experiments conducted on certain traffic scenarios. The behaviour of the SDN Controller and its effectiveness may be closely tied to our particular simulation environment. Are there unknown factors in our simulation environment that affect the relationship between the use cases and the resulting metrics? Do the programs running on the laptop, in or outside our simulation, cause interference? Here future work on more complex simulation environments would provide a broader picture of the capabilities of the SDN controller.

5.4 Future Work

The developed environment successfully achieved the goal of simulating a 5G transport network. The experiments conducted on the system enabled observation of the SDN controller performance under different traffic patterns. Through this observation knowledge was acquired on the SDN controller's potential in different traffic scenarios. However, the work on exploring dynamic bandwidth control has only just begun, and there are several improvements that could be made with more time, resources and knowledge. A few of these improvements will be discussed below.

5.4.1 Increase Network Complexity

In this thesis the focus was on the bandwidth allocation problem with a single OVS connected to two vDUs. When solving complex problems, a common approach is to solve it for a single entity, then for two entities and then for N entities. Therefore the natural next step would be to add another OVS and try to balance the bandwidth allocation between these switches. Further into the future the goal could be to be able to control all the switches in an access network. This expansion could also include adding more vDUs to each vCU to simulate a more realistic large-scale infrastructure. As the network grows one would also have to consider how control shall be maintained with increased complexity. One could imagine multiple SDN controllers where each is responsible for a smaller part of the network. This setup naturally raises the question of how these SDN controllers would communicate and collaborate across a large network.

5.4.2 Improve Algorithm

As has been mentioned earlier in the report, the implemented algorithm was intended as a proof of concept and a groundwork for future development. Discussed below are a few interesting ways that the base algorithm could potentially be improved in future work.

One area of improvement that was investigated during the thesis was to add a bandwidth margin to each allocation. This means taking the requested bandwidth from users and adding a certain percentage to this number. By doing this the algorithm is given extra time to react to changes in the traffic flow. During these investigations both dynamic and static margins were examined. The preliminary results were promising in showing decreased packet loss in some traffic scenarios. However, this came at the price of wasted bandwidth resources as the margin is equivalent to allocating more resources than needed. A thorough study of how wasted bandwidth and packet loss can be balanced in economic terms would be an interesting target for future work. Unfortunately time constraints hindered the intentions of including these results in the thesis.

Another way the algorithm could be improved is by incorporating more ideas from control theory into the control loop. An integral term could for example be added, in addition to the proportional and derivative term the control loop currently has. This could smoothen out the acceleration and reduce the peaks when the acceleration stops. Another path that could be explored is developing the objective function through the addition of new resource constraints, such as latency or economic factors.

Experimenting with different control intervals would also be interesting. The current control and observation interval is set to three seconds; an interval that could not be decreased further due to hardware limitations. A shorter interval could bring the system closer to real-time adaption. However, a longer interval could also be a possible way forward, with bandwidth limitations set through predictions. To make accurate predictions both time series analysis and machine learning could be envisioned.

Through exploring Control Theory, Linear Programming and Optimization, one could surely create a more reliable and accurate algorithm for bandwidth allocation. This would be fascinating to delve into and improving the algorithm could be a suitable subject for future thesis workers. By providing an open source environment, our work enables future work on algorithm development significantly.

5.4.3 Economic Considerations

Another intriguing aspect that could be further looked into would be the economic implications of applying a dynamic bandwidth allocation schema on an entire access network. When would it be prudent to overprovision network resources versus when it would make economic sense to implement a dynamic allocation system? This would likely come down to the marginal cost of production for a certain amount of bandwidth. It would be interesting to understand at what price point the scale tips in favor of one strategy or the other.

5.4.4 OpenFlow Protocol

Another future enhancement that could be made is to adopt the OpenFlow communications protocol. This is a protocol which enables an SDN Controller to implement changes on an OVS via an API, meaning that the SDNc agent in our setup would be made redundant. The reason that the OpenFlow protocol was not added to the current setup is that it would add complexity and require good knowledge of the protocol, while barely altering the simulation behaviour. However, it could serve a purpose in a more complex future version of the simulation environment, as it could potentially free up both memory and CPU resources on the VM hosting the OVS.

Chapter 6
Conclusion

In this thesis we have explored dynamic bandwidth control in the 5G transport network by developing a cloud-based simulation environment. The simulation environment combines cloud technologies and software defined networking with the goal of improving bandwidth resource utilization. This objective stems from an interest to explore alternatives to the bandwidth overprovisioning common in telecommunication today. As a basis for this exploratory research layed three research questions.

Through RQ 1.1 it was investigated how one could develop a cloud-based environment which represents the essential component of the 5G transport network. Using design science an envisioned environment was broken down into components, where each building block could be motivated through a rigor and relevance analysis. This process resulted in an environment built on both insights from the scientific community and from the business context of Ericsson. The final simulation environment models a small 5G transport network containing two distributed units and one centralized unit. The centralized unit contains a virtual switch that forwards traffic from the distributed unit to the core network. To enable dynamic bandwidth control a SDN controller monitors the network traffic and continuously reconfigures the switch to adapt to the current load.

A key characteristic of the environment is its emphasis on open-source components, such as Kubernetes, Docker, and Open vSwitch. Not only are these components very capable and well known, but they are also maintained by strong communities and under much scrutiny. The use of open-source components also facilitates further research on the environment created in this thesis. Other characteristics of high priority in the development work were scalability and flexibility. As such the environment is built to be modular and easily reconfigurable for further testing. This ensures that minimal code has to be written to simulate new traffic scenarios. The simulation environment is available under an Apache License at Github through [11].

In RQ 1.2 ways of prioritizing network traffic with different QoS levels in the simulation environment were investigated. To achieve this goal, the traffic had to be separated into classes in order to enable resource allocation based on priority. The separation of the traffic

originating from the Kubernetes pods was achieved through a network plugin called Multus, which allows for multiple network interfaces per pod. Each interface corresponds to a QoS level. Separated traffic enabled prioritized allocation of bandwidth in the virtual switch. Then an algorithm for calculating the allocation of bandwidth was developed. This was done through the implementation of a proof-of-concept linear programming algorithm. The SDN controller containing the algorithm was built to be modular, where the proof-of-concept algorithm could be easily swapped for another bandwidth allocation algorithm of choice.

In the third research question, RQ 2.1, the conditions under which the SDN controller could improve the utilization of network resources were examined. This was done with the help of use cases that had been developed together with engineers at Ericsson. Each use case represents a dynamic network traffic behaviour. In these experiments the focus was on evaluating the SDN controller's ability to minimize bandwidth usage while avoiding packet loss. Our simulation results show potential for dynamic bandwidth allocation, with bandwidth savings of up to 56%. The highest savings were achieved when traffic classes with different priorities peaked at different times, as was shown in use case 3. Packet loss was heavily affected by the speed of the demand increase. When traffic acceleration was doubled between use case 1 and 2, packet loss went from 0.4% to 2.3%. Use case 4 showed that the system could prioritize the traffic classes when demand exceeded the total available bandwidth. One of the conclusions from these experiments was that large and fast demand changes led to the highest packet loss. However, these large increases in network traffic also increased the potential to save bandwidth with dynamic allocation as compared to static allocation. This means that the choice on whether to implement dynamic bandwidth allocation or not, would have to weigh the potential bandwidth saving against an increased risk of breaking service-level-agreements. To learn more about this trade-off, experiments on real traffic data with larger simulation networks are necessary.

The work in this master thesis has been fundamental research, with the overarching goal of acquiring more knowledge about dynamic bandwidth control in the Cloud RAN. While a lot more research is needed before an actual product prototype could be built, our initial results show promise. If telecommunication companies could make use of the overprovisioned bandwidth of today's network, it could potentially enable them to sell more connections, save energy or reduce the needed infrastructure.

The fifth generation of mobile communication will bring entirely new possibilities and potentially change the way we interact with technology. However, this next generation will also entail more competition through open-sourced designs and vendor diversification. In this new telecommunication landscape optimal bandwidth usage might be one of the many components that will become important to stay competitive. If the potential of dynamic bandwidth allocation can be proven on real traffic patterns and implemented at scale, the days of static allocation might be over. On the journey towards the communication networks that will shape the societies of tomorrow, the search for novel solutions must be continued. This could be but a small step towards the future of mobile communication, but an exciting one nonetheless. What a time to be alive.

References

[1] 12 use cases for 5g ultra reliable communications. https://www.linkedin.com/pulse/12-use-cases-5g-ultra-reliable-communications-sebastian/. Accessed: 2020-05-27.

[2] 5g wireless access: an overview. https://www.ericsson.com/en/reports-and-papers/white-papers/5g-wireless-access-an-overview. Accessed: 2020-05-27.

[3] Amazon elastic kubernetes service. https://aws.amazon.com/eks/. Accessed: 2020-03-05.

[4] Aws global network. https://aws.amazon.com/about-aws/global-infrastructure/global_network/. Accessed: 2020-05-31.

[5] Azure kubernetes service. https://azure.microsoft.com/sv-se/services/kubernetes-service/. Accessed: 2020-03-05.

[6] cadvisor. https://github.com/google/cadvisor. Accessed: 2020-03-25.

[7] Cbc. https://github.com/coin-or/Cbc. Accessed: 2020-05-11.

[8] Championing the open source networking ecosystem. https://www.ericsson.com/en/open-source-networking. Accessed: 2020-05-27.

[9] Cisco networking academy's introduction to basic switching concepts and configuration. https://www.ciscopress.com/articles/article.asp?p=2181836&seqNum=3. Accessed: 2020-05-10.

[10] Cloud container distribution. https://www.ericsson.com/en/portfolio/digital-services/cloud-infrastructure/cloud-container-distribution. Accessed: 2020-06-01.

[11] Cloud-ran-simulation-environment. https://github.com/danielisak/Cloud-RAN-Simulation-Environment. Accessed: 2020-06-03.

[12] Congestion avoidance overview. https://www.cisco.com/c/en/us/td/docs/ios/qos/configuration/guide/12_2sr/qos_12_2sr_book/congestion_avoidance.html. Accessed: 2020-05-10.

[13] Containerization. https://www.ibm.com/cloud/learn/containerization. Accessed: 2020-05-31.

[14] flannel. https://github.com/flannel-io/flannel. Accessed: 2021-03-05.

[15] Flask. https://github.com/pallets/flask/. Accessed: 2020-05-31.

[16] The four components of cloud ran. https://www.ericsson.com/en/blog/2020/8/the-four-components-of-cloud-ran. Accessed: 2020-05-31.

[17] Freepik wireless technology isometric flowchart, designed by macrovector. http://www.freepik.com. Accessed: 2020-05-27.

[18] Google kubernetes engine. https://cloud.google.com/kubernetes-engine. Accessed: 2020-03-05.

[19] Grafana. https://grafana.com/oss/grafana/. Accessed: 2020-05-04.

[20] Iperf. https://iperf.fr/. Accessed: 2021-03-17.

[21] Kubernetes components. https://kubernetes.io/docs/concepts/overview/components/. Accessed: 2020-05-31.

[22] Kubernetes components. https://kubernetes.io/docs/concepts/overview/components/. Accessed: 2020-03-25.

[23] Kubernetes jobs. https://kubernetes.io/docs/concepts/workloads/controllers/job/. Accessed: 2020-05-13.

[24] Linux traffic control manual. https://man7.org/linux/man-pages/man8/tc.8.html. Accessed: 2021-03-17.

[25] Metrics for kubernetes system components. https://kubernetes.io/docs/concepts/cluster-administration/system-metrics/. Accessed: 2020-03-25.

[26] Multus-cni. https://github.com/intel/multus-cni. Accessed: 2021-03-05.

[27] Network functions virtualization (nf); use cases. https://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf. Accessed: 2020-05-10.

[28] The node pole: inside facebook's swedish hub near the arctic circle. https://www.theguardian.com/technology/2015/sep/25/facebook-datacentre-lulea-sweden-node-pole. Accessed: 2020-05-21.

[29] Nodes. https://kubernetes.io/docs/concepts/architecture/nodes/. Accessed: 2020-03-25.

[30] Open ran explained. https://www.ericsson.com/en/openness-innovation/open-ran-explained. Accessed: 2020-05-11.

[31] Open vswitch. https://www.openvswitch.org/. Accessed: 2020-04-16.

[32] Prometheus. https://github.com/prometheus/prometheus. Accessed: 2020-03-25.

[33] pulp. https://github.com/coin-or/PuLP. Accessed: 2020-05-13.

[34] Quality of service (qos). https://docs.openvswitch.org/en/latest/faq/qos/. Accessed: 2020-05-11.

[35] Stack overflow developer survey 2020. http://https://insights.stackoverflow.com/survey/2020. Accessed: 2020-03-05.

[36] Telia – first in the world with 4g. https://www.teliacompany.com/en/about-the-company/history/first-in-the-world-with-4g/. Accessed: 2020-03-18.

[37] Virtualized 5g ran: why, when and how? https://www.ericsson.com/en/blog/2020/2/virtualized-5g-ran-why-when-and-how. Accessed: 2020-05-27.

[38] What is a container? https://www.docker.com/resources/what-container. Accessed: 2020-05-31.

[39] What is postman? https://www.postman.com/. Accessed: 2020-05-17.

[40] Control The Cloud, Before The Cloud Controls You. Technical report, 01 2016. Accessed: 2020-03-05.

[41] 5G-PPP. 5g-ppp stakeholders survey. https://5g-ppp.eu/5g-ppp-stakeholders-survey/. Accessed: 2020-05-10.

[42] John Arundel and Justin Domingus. *Cloud Native DevOps with Kubernetes: building, deploying, and scaling modern applications in the cloud*. O'Reilly Media, 2019.

[43] Saleh Asadollahi, Bhargavi Goswami, and Mohammed Sameer. Ryu controller's scalability experiment on software defined networks. *2018 IEEE International Conference on Current Trends in Advanced Computing (ICCTAC), Current Trends in Advanced Computing (ICCTAC), 2018 IEEE International Conference on*, pages 1 – 5, 2018.

[44] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. Microservices architecture enables devops: Migration to a cloud-native architecture. *Ieee Software*, 33(3):42–52, 2016.

[45] Kamal Benzekki, Abdeslam El Fergougui, and Abdelbaki Elbelrhiti Elalaoui. Software-defined networking (sdn): a survey. *Security and communication networks*, 9(18):5803–5833, 2016.

[46] Rajkumar Buyya, Satish Narayana Srirama, Giuliano Casale, Rodrigo Calheiros, Yogesh Simmhan, Blesson Varghese, Erol Gelenbe, Bahman Javadi, Luis Miguel Vaquero, Marco AS Netto, et al. A manifesto for future generation cloud computing: Research directions for the next decade. *ACM computing surveys (CSUR)*, 51(5):1–38, 2018.

[47] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[48] Ericsson. Cloud native by ericsson. `https://www.ericsson.com/en/cloud-native`. Accessed: 2020-05-10.

[49] Ericsson. Cloud ran by ericsson. `https://www.ericsson.com/en/ran/cloud`. Accessed: 2020-03-18.

[50] Matteo Fiorani, Paolo Monti, Björn Skubic, Jonas Mårtensson, Luca Valcarenghi, Piero Castoldi, and Lena Wosinska. Challenges for 5g transport networks. In *2014 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pages 1–6, 2014.

[51] Christopher B. Hauser and Santhosh Ramapuram Palanivel. Dynamic network scheduler for cloud data centres with sdn. In *Proceedings of The10th International Conference on Utility and Cloud Computing*, UCC '17, page 29–38, New York, NY, USA, 2017. Association for Computing Machinery.

[52] Alan Hevner, Alan R, Salvatore March, Salvatore T, Park, Jinsoo Park, Ram, and Sudha. Design science in information systems research. *Management Information Systems Quarterly*, 28:75–, 03 2004.

[53] E. Kim, K. Lee, and C. Yoo. On the resource management of kubernetes. In *2021 International Conference on Information Networking (ICOIN)*, pages 154–158, 2021.

[54] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[55] Carlos Manso, Ricard Vilalta, Ramon Casellas, Ricardo Martinez, and Raul Munoz. Cloud-native sdn controller based on micro-services for transport networks. *2020 6th IEEE Conference on Network Softwarization (NetSoft), Network Softwarization (NetSoft), 2020 6th IEEE Conference on*, pages 365 – 367, 2020.

[56] Iven Mareels. Linear controller design limits of performance : Stephen p. boyd and craig h. barratt. *Automatica*, 29:805, 05 1993.

[57] Mpirical. Exchanging data in 5g. Technical report, Mpirical Limited, 2018.

[58] Mohammad Robihul Mufid, Arif Basofi, M. Udin Harun Al Rasyid, Indhi Farhandika Rochimansyah, and Abdul rokhim. Design an mvc model using python for flask framework development. In *2019 International Electronics Symposium (IES)*, pages 214–219, 2019.

[59] Lance James Natonski, Tae-Hoon Kim, and Bethany Hadden. The effect of relay node and power control on performance in multi-hop wireless network. In *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 540–544, 2017.

[60] P. Popovski, K. F. Trillingsgaard, O. Simeone, and G. Durisi. 5g wireless network slicing for embb, urllc, and mmtc: A communication-theoretic view. *IEEE Access*, 6:55765–55779, 2018.

[61] Shixiong Qi, Sameer G Kulkarni, and K. K. Ramakrishnan. Understanding container network interface plugins: Design considerations and performance. In *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pages 1–6, 2020.

[62] Muhammad Rehan Raza, Matteo Fiorani, Ahmad Rostami, Peter Öhlen, Lena Wosinska, and Paolo Monti. Benefits of programmability in 5g transport networks. 2017.

[63] F. Richter. Amazon leads $130-billion cloud market. `https://www.statista.com/chart/18819/worldwide-market-share-of-leading\-cloud-infrastructure-service-providers/`. Accessed: 2020-03-18.

[64] Jyotiprakash Sahoo, Subasish Mohapatra, and Radha Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *2010 Second International Conference on Computer and Network Technology*, pages 222–226. IEEE, 2010.

[65] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and Y. C. Tay. Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, New York, NY, USA, 2016. Association for Computing Machinery.

[66] S. Sharma, R. RV, and D. Gonzalez. *Microservices: Building Scalable Software*. Packt Publishing, 2017.

[67] James E Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, 2005.

[68] Xiang Sun, Nirwan Ansari, and Ruopeng Wang. Optimizing resource utilization of a data center. *IEEE Communications Surveys & Tutorials*, 18(4):2822–2846, 2016.

[69] T4. Container platform market share, market size and industry growth drivers, 2018 - 2023. `https://www.t4.ai/industries/container-platform-market-share`. Accessed: 2020-03-25.

[70] International Telecommunication Union. Minimum requirements related to technical performance for imt 2020 radio interface(s), November 2017. ITU-R M.2410-0.

[71] Junshe Wang and Pengzhou Shi. Research on bandwidth control technology based on sdn. In *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*, pages 705–708, 2018.

[72] Kimio Watanabe. Outdoor lte infrastructure equipment (enodeb). 2012.

[73] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer-Verlag Berlin Heidelberg, 2012.

[74] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.

[75] Fei Zhang, Guangming Liu, Xiaoming Fu, and Ramin Yahyapour. A survey on virtual machine migration: Challenges, techniques, and open issues. *IEEE Communications Surveys Tutorials*, 20(2):1206–1243, 2018.

[76] Long Zhang, Yi Zhuang, and Wei Zhu. Constraint programming based virtual cloud resources allocation model. *International Journal of Hybrid Information Technology*, 6(6):333–344, 2013.

Appendices

Appendix A
Use Cases

The network traffic will be achieved by several smaller traffic streams that together aggregate to the below specified values. These streams will be in the size range of 0-10 Mbit/s.

A.1 Use case 1: The silver traffic class peaks and then decreases again

Prerequisites:

Available bandwidth is set to:
Gold: 20 Mbit/s
Silver: 15 Mbit/s
Bronze: 15 Mbit/s

Traffic flow:

At time zero seconds, network traffic is set to:
Gold: 20 Mbit/s static
Silver: 15 Mbit/s static
Bronze: 15 Mbit/s static

Traffic increase:

Silver requests + 10 Mbit/s every 10s three times, then back down to 15 Mbit/s total.

Expected bandwidth allocation:

Midpoint:
Gold: 20 Mbit/s
Silver: 55 Mbit/s
Bronze: 15 Mbit/s
End:
Gold: 20 Mbit/s
Silver: 15 Mbit/s
Bronze: 15 Mbit/s

A.2 Use case 2: The silver traffic class peaks slowly and then decreases again

Prerequisites:

Available bandwidth is set to:
Gold: 20 Mbit/s
Silver: 15 Mbit/s
Bronze: 15 Mbit/s

Traffic flow:

At time zero seconds, network traffic is set to:
Gold: 20 Mbit/s static
Silver: 15 Mbit/s static
Bronze: 15 Mbit/s static

Traffic increase:

Silver requests + 5 Mbit/s every 10s three times, then back down to 15 Mbit/s total

Expected bandwidth allocation:

Midpoint:
Gold: 20 Mbit/s
Silver: 30 Mbit/s
Bronze: 20 Mbit/s
End:
Gold: 20 Mbit/s
Silver: 15 Mbit/s
Bronze: 15 Mbit/s

A.3 Use case 3: All traffic classes spike at different times

Prerequisites:

Available bandwidth is set to:
Gold: 10 Mbit/s
Silver: 10 Mbit/s
Bronze: 10 Mbit/s

Traffic flow:

At time zero seconds, network traffic is set to:
Gold: 10 Mbit/s static
Silver: 10 Mbit/s static
Bronze: 10 Mbit/s static

Traffic increase:

Gold requests + 40 Mbit/s after 30s, then down to 10 Mbit/s again.
Silver requests + 40 Mbit/s after 30s then down to 10 Mbit/s again.
Bronze requests + 40 Mbit/s after 30s then down to 10 Mbit/s again.

Expected bandwidth allocation:

At Gold spike:
Gold: 72 Mbit/s
Silver: 11 Mbit/s
Bronze: 10 Mbit/s
At Silver spike:
Gold: 12 Mbit/s
Silver: 66 Mbit/s
Bronze: 10 Mbit/s
At Bronze spike:
Gold: 12 Mbit/s
Silver: 11 Mbit/s
Bronze: 60 Mbit/s

A.4 Use case 4: Gold increases which throttles bronze and then silver

Prerequisites:

Available bandwidth is set to:
Gold: 5 Mbit/s
Silver: 5 Mbit/s
Bronze: 5 Mbit/s

Traffic flow:

At the beginning of the use case, network traffic is set to:
Gold: 20 Mbit/s static
Silver: 10 Mbit/s static
Bronze: 10 Mbit/s static

Traffic increase:

Gold requests + 20 Mbit/s every 10s three times
Silver requests + 10 Mbit/s one time
Bronze requests + 10 Mbit/s one time

Expected bandwidth allocation:

Midpoint:
Gold: 80 Mbit/s
Silver: 20 Mbit/s
Bronze: 5 Mbit/s

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-06-03

EXAMENSARBETE Dynamisk bandbreddsstyrning för att förbättra nätverksresursanvändning i molnbaserade 5G Radio Access Nätverk
STUDENT August Lidfeldt och Daniel Isaksson
HANDLEDARE Markus Borg (LTH) och Erik Walles (Ericsson)
EXAMINATOR Per Andersson (LTH)

Nya sätt att förbättra 5G-nätverk med teknik från molnet

POPULÄRVETENSKAPLIG SAMMANFATTNING **August Lidfeldt och Daniel Isaksson**

Medan de flesta är medvetna om att 5G är nära förestående är det förmodligen få som vet vad som händer bakom kulisserna till nästa generationens mobilkommunikation. I samband med att Ericsson förbättrar sina 5G-nätverk med molnteknologier uppstår nya utmaningar och möjligheter.

En viktig skillnad mellan 4G och 5G är att den nya generationen använder sig av molnteknologier. Medan företag som Amazon, Google och Microsoft under det senaste decenniet har bemästrat drift och underhåll av storskaliga moln, är kombinationen av telekommunikation och molnet ny mark. I detta examensarbete har vi utvecklat en simuleringsmiljö baserad på molnteknologier för att undersöka hur programvara kan användas för att intelligent styrning av bandbredd i 5G-nätverk. Med den utvecklade miljön har vi simulerat olika typer av trafikflöden för att utforska de scenarier där dynamisk mjukvarustyrning av bandbredd har störst potential. I mobilkommunikation förbrukas bandbredd varje gång en användare skickar data genom nätverket. De tillgängliga bandbreddsresurserna är ofta begränsade av antalet nätverkskablar, routrar och annan nätverksutrustning. För att undvika fördröjning och långsamma nätverkshastigheter har bandbreddsresurser historiskt sett tilldelats i överflöd. Detta då utbyggnad av antalet nätverkskablar och annan utrustning garanterade hög servicekvalitet samtidigt som det var relativt billigt. I 5G-nätverk förväntas dock datavolymerna som skickas öka avsevärt, en förändring som kommer kräva intelligentare användning av bandbreddsresurser. Förutom vanliga användare som streamar Ultra-HD-filmer eller spel, kommer 5G också att koppla upp självkörande bilar och massiva nätverk av IoT-enheter. Dessa olika typer av nätverksanslutningar måste ha olika prioritet i nätverket; en IoT-termometer i en fabrik kan tolerera en fördröjning på någon sekund, en självkörande lastbil kan endast tolerera en fördröjning på några millisekunder. Den simuleringsmiljö som utvecklats i detta examensarbete adresserar både behovet av dynamisk bandbreddsstyrning och trafikprioritering. Den slutliga simuleringsmiljön modellerar en liten del av ett 5G-nätverk och möjliggör simulering av olika användare som ansluter till internet. Användarna är indelade i tre klasser och får på så sätt olika prioritet när deras trafik vidarebefordras i nätverket. Simuleringen av trafikscenarier visar att det finns potential för dynamisk bandbreddsstyrning då den nödvändiga bandbredden skulle kunna minskas jämfört med det överskott av bandbredd som tilldelas i traditionella telekommunikationsnät. Däremot krävs mer forskning på verkliga trafikmönster för att ta reda på den faktiska potentialen i lösningen. Om dynamisk bandbreddsstyrning skulle kunna bevisas effektiv på verklig trafik och implementeras i stor skala kan bandbreddsslösandets dagar vara räknade.