Faculty of Science and Technology
Department of Computer Science

Exploring the inner mechanisms of 5G networks for orchestrating container-based applications in edge data centers

Fredrik Hagen Fasteraune

Master thesis in Computer Science June 2023

I would like to dedicate this thesis to my supervisors Anders and Arne.

“Simplicity is prerequisite for reliability”
–Edsger Dijkstra

“Premature optimization is the root of all evil”
–Donald Knuth

“I apologize for such a long letter — I didn’t have time to write a short one.”
–Mark Twain

Abstract

One of the novel new features of mobile 5G networks is what is commonly known as "Ultra Reliable Low Latency" communication. To achieve the "Low Latency" part, it is necessary to introduce processing and storage capabilities closer to the radio access network, thus introducing Edge data centers. An Edge data center will be capable of hosting third-party applications and a user of these applications can access them using the cellular mobile network. This makes the network path between the user equipment (UE) and the application short in terms of physical distance and network hops, thus reducing the latency dramatically.

This thesis looks into these new features of the 5th-generation mobile networks to establish if, and how they can be used to orchestrate container-based applications deployed at edge data centers. The orchestration mechanism suggested will be described in more detail in the thesis body but as an overview, it involves using the user's positions and the knowledge about which applications the users are accessing and information about where these applications reside to move applications between edge data centers.

One of the 5G exploration findings was that the location of users in a 5G network can be determined using the Network Exposure Function (NEF) API. The NEF is one of the new 5G network functions and enables trusted third-party actors to interact with the 5G core through a publisher-subscriber-oriented API. The proposed orchestration strategy involves calculating the "weighted average location" of 5G users who have accessed the specific application residing in the Edge within a specified time frame.

A live 5G network with a stand-alone (SA) core was not available at the time of writing and part of the thesis work has therefore been to identify if there exist network emulators with the functionality needed to reach the goal of this thesis, i.e. design and implement the orchestrator based on interaction with the network. More specifically: can we find a NEF emulator that can be configured to give us network data related to user equipment location? Unfortunately, the three alternatives considered: Open5Gs, NEF_emulator, and Nokia's Open5Glab do not fully meet our requirements for generating user events. Open5Gs an open source 5G network implementation lacks the whole NEF north-bridge implementation, NEF_emulator has limited implementation and integration complexities, and Nokia's Open5Glab's simulated users are inactive and thus do not generate sufficient data.

Given the absence of suitable emulators to generate the needed data, the thesis pivoted to also include the design and implementation of a mobile network emulator with the following key components: a mobile network abstraction that encompasses crucial elements from 5G, such as users and radio access nodes, allowing users to connect to the mobile network; a network abstraction that hosts emulated edge data centers and the corresponding applications accessible to connected users; and mobile network exposure that exposes mobile network core events through a simplified NEF north-bound API implementation.

Finally, the thesis concludes by implementing the proposed orchestration strategy using the mobile network emulator, demonstrating that orchestrating can effectively reduce the end-to-end latency from users to applications, as evidenced by the obtained results.

Acknowledgements

I would like to give a special thanks to both of my supervisors Anders and Arne, as well as thanks to my family and peers who have helped me through this difficult period. And a special thanks to my girlfriend Ida Marie who is the most awesome of all.

Contents

List of Figures

1 Introduction

Traditionally, the evolution of telecom networks has occurred at a relatively slow pace. However, a significant transformation is underway with the introduction of 5G networks. These networks are in an excellent position to revolutionize the structure and functionality of network systems, representing a remarkable leap forward. 5G networks are expected to incorporate edge computing on a large scale to achieve extremely low latency. Papers such as [1] and [2] help build up and explore some possibilities related to edge computing and 5G networks.

This shift in telecom networks signifies a departure from traditional approaches and opens up new possibilities for enhanced connectivity, real-time applications, and improved user experiences. As 5G continues to unfold, it is anticipated that these advancements will shape the future of telecommunications and enable a wide range of innovative services and applications.

Consider an infrastructure provider that has data centers in various locations. Since these data centers are very close to the potential users we refer to them as edge data centers, as they are relatively close to the edge of computational resources. At least when considering the infrastructure provider's point of view. Suppose the infrastructure provider wants to offer Infrastructure as a Service (IaaS), Containers as a Service (CaaS), and Functions as a Service (FaaS) using these edge data centers, much similar to other cloud providers. This implies that edge data centers can host applications, which can be everything from micro-services to fully blown web-services. This is valid as long as the application can be containerized and managed by tools like Docker or Kubernetes. An illustration of this can be seen in Figure 1.1

We refer to the clients of these applications as users, and they can connect either via the Internet or 5G networks. This thesis will focus on the users connecting via 5G networks. We envision that all these applications can be placed on any edge data center and thus be used by users. We strike a problem when we let users freely choose which edge data center to place applications. In the worst-case scenario, all applications will be placed in the same edge data center. This will fatally throttle the performance, considering parameters like latency, bandwidth and available resources on the said edge data center. This is illustrated in Figure 1.2.

To handle this we want to optimize the placement of these applications to the edge data centers in such a way that we minimize the aforementioned parameters. This is in other words an optimization problem where we have many users that want to use applications that are placed in edge data centers.

We want to find an optimal distribution of applications in these edge data centers, so that we minimize the latency to the users, have the most computational

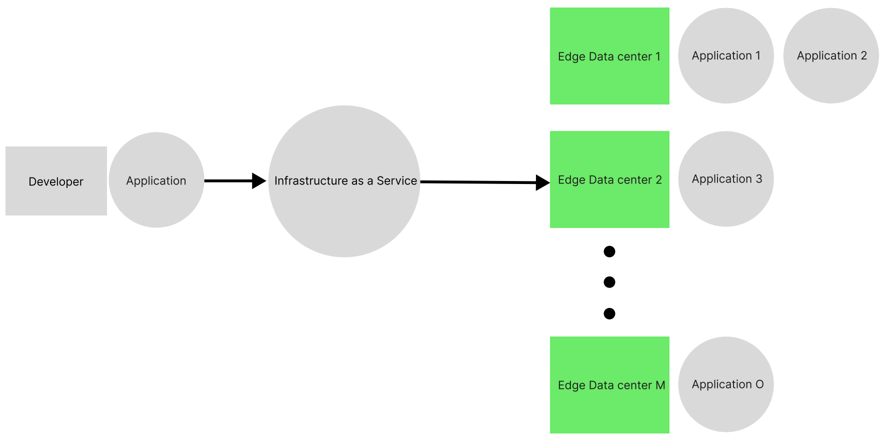


Figure 1.1: Illustration of an initial situation where developer deploys applications

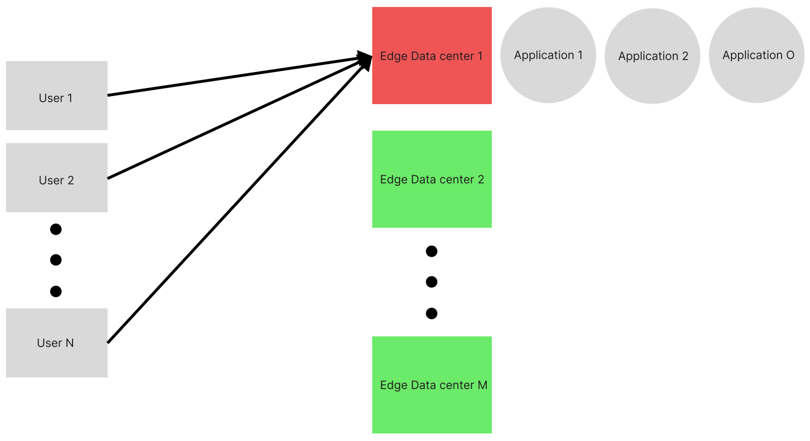


Figure 1.2: Illustration of a worst-case scenario where users choose where to place applications

resources available at the edge data centers, and overall save bandwidth in the network by minimizing the network routes between users and edge data centers. However, there are some concerns with finding an optimal distribution. Firstly an optimal distribution at one point in time is not optimal at a later time. This is due to the user traffic pattern to the application changing over time.

Another concern is that the set of applications also constantly changes.

To solve this problem this thesis proposes an application orchestrator, which has the goal of continually finding the optimal configuration of applications on edge data centers. The application orchestrator will solve this by looking at the application usage patterns of the users, and the locations of edge data centers relative to the users who are using applications. Based on this information the application orchestrator should be able to continually find an optimal configuration of applications such that it minimizes latency between users and applications, saves bandwidth by leveraging shorter routes and finally improve resource availability. An Illustration can be seen in Figure 1.3

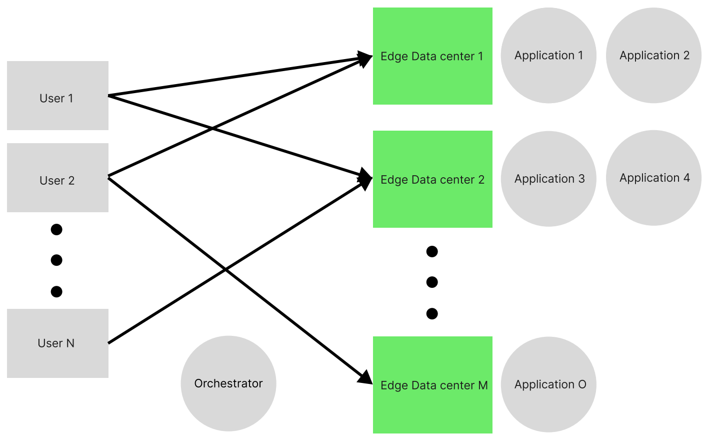


Figure 1.3: Illustration of a best-case scenario where the orchestrator help choose where to place applications

A previous capstone project has shown that introducing a mechanism to do this precisely would be beneficial. The project's results showed that it reduced all users' average latency and standard deviation. One of the significant disadvantages of the previous work is that it did not address how to properly use and leverage 5G networks to achieve this task.

The main contribution of this thesis will be: Investigate 5G networks and their capabilities to offer application migration. This includes providing an overview of the architecture, services, and select functionality of a 5G standalone core cellular network. Then if possible, we will leverage this insight to propose an application orchestration mechanism. With the application orchestration mechanism in place, we will investigate if there is possible to use some existing projects to generate realistic user data and traffic for implementing the proposed strategy. Based on the results, some experiments will be conducted to provide proof of concept.

The thesis will include a chapter consisting of a literature review, related work and projects, which will cover the necessary technical background and related work. It will then cover and suggest an application orchestration strategy using user locations and traffic patterns to applications. Finally, the chapter will look into existing projects where we will try to conduct some practical work to see if there is possible to create a proof of concept. However, due to the projects lacking features, we will design and implement a custom mobile network emulator which will be described in the result chapter.

The literature review, related work and projects will be followed by a method chapter outlining the problem of the thesis, the hypothesis of how to solve the problem and, how the work process has been during the thesis.

The next chapter will be the result chapter which will go through the design and implementation of the mobile network emulator and orchestrator. In addition to experiments and discussions around the mobile network emulator and orchestrator. The experiments section will cover a simple experiment using the emulator and orchestrator as proof of concept. Finally, in this chapter, there will be a discussion, which covers some issues and limitations of this thesis.

Finally, this thesis will include a conclusion, which recaps this thesis and proposes some future work from the previous chapter.

2 Literature review, related work and projects

To realize application orchestration using 5G it is appropriate to look into the technical side of 5G. This chapter will look at selected technical details within 5G networks that we believe are useful to achieve application orchestration. To be more specific we need to know how a third-party application can interact with 5G networks to retrieve information about users and how that effect the network and its users.

This section will be followed by previous work conducted and its results and limitations. And as mentioned in the previous chapter there have been performed some previous work that showed that application orchestration is beneficial.

This will be followed by a section that will look into related work and projects that are either, useful to establish the need for 5G networks and edge computing, orchestration mechanisms to solve this problem, papers that propose technologies that can be used to optimize edge data centers, and projects that we find useful for this thesis.

Finally, we have a section that looks into and proposes an orchestration strategy based on the literature review, related work and projects. This includes proposing an orchestration strategy, in addition to practical work where we investigate existing 5G-related solutions to see if any of them are viable for performing practical experiments validating the orchestration strategy.

2.1 5th generation mobile networks

5th generation mobile cellular network, or 5G, is the fifth iteration of mobile cellular networks. There are many advantages when using 5G networks compared to the older generations. These are some of the advantages listed in Qualcomm's[3] marketing pitch:

One argument for adapting 5G compared to the older generations is that 5G offers higher data rates. For instance, based on Qualcomm's marked pitch, they claim that 5G can offer up to 20 gigabits per second in peak data rate and an average of over 100 megabits per second, compared to 4G, which only offers up to 35 megabits per second in peak data rate and an average of 15 megabits per second. Another point of using 5G compared to the older generations is that 5G offers much more capacity. According to Qualcomm, 5G is designed to handle 100 times more traffic concerning capacity and network efficiency. 5G networks offer lower latency; based on Qualcomm, it provides a ten times decrease down to 1 ms. 5G utilizes the radio spectrum much better than 4G. The last point is that the 5G platform and core are based on one unified platform

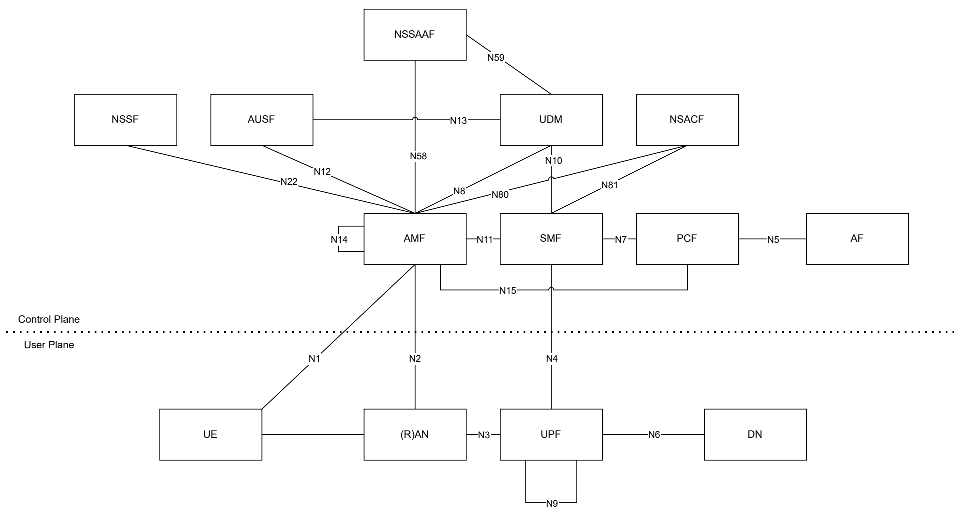


Figure 2.1: Simplified 5G standalone core architecture with emphasis on *N reference points*

with more capabilities and features than 4G. It is this core that this thesis will be in focus.

In the telecommunications domain, most components in 5G are referred to as functions, this is because 5G is based on a MiMo (Multiple input Multiple output) model. Where each component is capable to be replicated so the core network is capable of massive horizontal scaling.

The 5G system logically divides into two parts: the user and control plane. The user plane consists of users with 5G-capable equipment who can connect to the 5G network and have their traffic sent to the public internet or other data networks. The other part is the control plane, based on a service architecture where the components are split into parts with different responsibilities. In 5G these components are called functions. For instance, the component which is responsible for handling user policies within the control plane is called the Policy Control Function. A simplified figure of the 5G architecture can be seen in Figure 2.1

The user plane consists of four components: The clients with 5G capable user equipment or for short UE, gNodeBs, which is the Radio Access Node (RAN) and is responsible for forwarding data to the user plane function and handling the communication between the control plane and UE. The next component is the UPF (User Plane Function), which is a service-based component that is responsible for handling the user data so that it reaches the data network. The last component is the Data Network, which can be everything from the Internet service provider's private network, services, and, public internet.

The control plane consists of many service components, and each component has its responsibilities; many components are strictly required for the network to function, and some are not required but can provide useful functionality. There are many components in the control plane, many can be seen in Figure 2.1.

Most of the components in the control plane are interconnected, and most parts offer a *Network Function Service*, which they use to communicate internally. The Network Function Service behaves like an interface to the other network functions. The interface functions like a publisher-subscriber mechanism, so if a network function wants to interact with another network function, it does this through the Network Function Service. It is also possible to send "commands" through certain network function services. For instance, in the NEF (Network Exposure Function), which is a network function that lets external application functions communicate with the control plane, it is possible to fetch the current location of a UE. The most crucial notion here is that most network function services behave like publisher-subscriber interfaces. The network function services are defined over an HTTP (Hypertext transfer protocol). Figure 2.2 shows a simplified 5G core with an emphasis on how the network functions expose network function services and how they communicate.

The other way components communicate in the 5G network is over other protocols and interfaces. In the 3GPP specification[4], they refer to these as reference points. These are more formally defined protocols for communicating between network functions and other components. In 3GPP they call these reference points N and a number. For instance, the N2 reference point describes strictly how the (R)AN communicates with the AMF. In the 3GPP 5G system architecture, they defined up to 95 reference points. Note that most reference points between network functions are realized through the network function service they provide. The exceptions are the refrence points *N1* to *N4*, *N6*, and *N9*. Figure 2.1[1] shows a simplified 5G core with an emphasis on how the reference points are used. All the reference points can be found in the 3GPP specification.

This thesis does not require to have in-depth knowledge of all components in the control plane. Therefore will this literature review will focus on the user plane and select components from the control plane. The components which are selected from the control plane are chosen because they provide the relevant functionality for achieving the application orchestration.

1. Note that Figure 2.2 and 2.1 are recreations of simplified figures from the 3GPP 5G system architecture specification[4, Section 4]. They aim to show a simplified 5G network with a minimum of network functions.

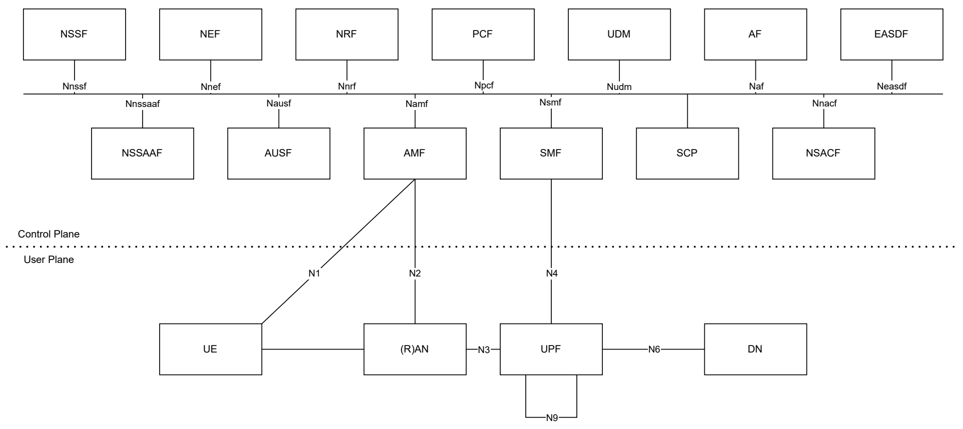


Figure 2.2: Simplified 5G standalone core architecture with emphasis on *Network function services*

The components we will look into are the following:

- Access and Mobility Management Function (AMF).
- Policy Control Function (PCF).
- Session Management Function (SMF).
- Unified Data Management (UDM).
- Unified Data Repository (UDR).
- Network Data Analytics Function (NWDAF).
- Analytics Data Repository Function (ADRF).
- Service Communication Proxy (SCP).
- Network Exposure Function (NEF).

The following sections will go through the user plane and select functions from the control plane. The intention is to show some understanding of how 5G functions internally. The technical details in the following sections are based on the technical specification published by ETSI and 3GPP[4].

2.1.1 User plane

User Equipment

User equipment (UE) refers to a device capable of connecting to 5G. UEs encompass all devices that can connect to the 5G network using New Radio (NR), the protocol that replaces the older communication protocol EUTRA used for 4G.

In broad terms when a user equipment wants to connect to a data network it contacts a Radio Access Node through a protocol called Non-Stratum Signaling (NAS) which contacts the control plane and establishes a PDU-Session which, finally enables the UE to access the internet.

Radio Access Nodes or gNodeBs

Radio Access Node (RAN) or in the 5G domain gNodeBs are the first point of entry into the 5G network for UEs. In 5G gNodeBs base stations act as the access points which the UEs can communicate with. They serve as interfaces to communicate with the control plane and user plane. The gNodeB base stations are physical devices placed in a location where the telecommunications provider wants to offer 5G.

User Plane Function

The User Plane function (UPF) is a component in the 5G core network and is responsible for managing user sessions and directing traffic to the intended destination. Although technically part of the control plane, it makes more sense to classify the UPF as part of the user plane since it handles user traffic from user devices into the data network.

The most interesting features the UPF offers are: Allocating UE IP address/prefix, acting as an external PDU session point of interconnect when connecting to the data network, routing and forwarding packets in connection with PDU sessions, enforcing policy rules concerning gating, redirection, and traffic steering, performing lawful interception, handling Quality of Service for the user plane, performing traffic verification, transport level packet marking, and finally, packet buffering. In addition, the UPF can respond to Address Resolution Protocol (ARP) and IPv6 Neighbor solicitation requests, and provide MAC addresses corresponding to IP addresses. It can also expose network information, such as QoS monitoring data. However, the UPF does not provide a network function service interface, and the control plane interacts with it through other interfaces, particularly the N4 interface.

Data Network

The data network in 5G can either be the public internet, private closed networks and everything in between. The data network, which will be in focus in this thesis is the public internet.

2.1.2 Control plane

Access and Mobility Management Function

The Access and Mobility Management Function or AMF is one of the most central and critical components of a 5G standalone core network. It is the first component that the UE will communicate within the control plane and the AMF relays most of the UE requests to other components.

The AMF is responsible for signaling and mobility of user equipment (N1), it also relays communication from RAN into the network (N2). It serves as the endpoint for communication interfaces in the Radio Access Network and handles signaling Non-Stratum Signaling NAS termination (N1) and ciphering/integrity protection in the Non-Stratum context.

One of the primary roles of the AMF is to manage registration, connection, reachability, and mobility for user equipment seeking to utilize the network. This entails keeping track of the gNodeBs to which a UE is connected and determining its reachability. The AMF is responsible for handling access authentication, authorization, and slice-specific (which part of the radio frequency the UE is using) networks for UEs. Additionally, the AMF facilitates lawful intercept capabilities. The AMF enables transport from the UE to the core network, facilitating the transmission of session management messages to the session management function and SMS messages to the SMS function. The AMF acts as a conduit for transporting messages between the core network components and the UE. It provides notable transport and proxy services, including the transport of session management messages from the UE to the session management function and SMS messages from the UE to other users. The AMF also facilitates location message transport between the UE and the LMF (Location Management Function) as well as between the RAN and LMF.

The AMF supports cloud IoT optimizations in both the control and user planes. It also includes features for restricting enhanced coverage provisioning based on external parameters, which define the expected behavior of UEs or network configuration parameters. The AMF also offers various features to support non-3GPP access networks and roaming capabilities. To interact with the AMF from the other services, the AMF provides a network function service and it can be found in[4].

The AMF is important in our case because it provides many of the critical features of how the UE is managed related to location and connectivity, and it provides communication from the UE to the other control plane components.

Policy Control Function (PCF)

The Policy control function (PCF) is responsible for providing Quality of Experience (QoE) and Quality of Service (QoS) profiles to the session management function. The PCF does this by supporting a unified policy framework that makes it possible to govern network behavior. This includes the PCF providing policy rules to the other control plane functions so that they can enforce them. Lastly, it uses the subscription information of users found in the Unified Data Repository to make policy decisions.

The PCF provides a network function service that the other network functions use to interact with the PCF. The network function services can be found in the 3gpp specifications[4].

Session Management Function (SMF)

The session management function (SMF) is responsible for setting up the UDM sessions in the core network. It is also managing the data path for a PDU session.

The SMF has several key responsibilities as defined by the 3GPP specification[4]. It is responsible for establishing, modifying, and releasing tunnels between the UPF and AN nodes. Furthermore, the SMF handles the allocation and management of UE IP addresses, which may involve optional authorization. The UE IP address can be obtained from either a UPF or an external data network. In addition, the SMF provides DHCP (Dynamic Host Configuration Protocol) services for both IP versions 4 and 6. When it comes to Address Resolution Protocol (ARP) requests and IPv6 neighbor Solicitation requests, the SMF promptly responds based on locally stored cache information for Ethernet PDUs. It fulfills these requests by providing MAC addresses corresponding to the IP addresses included in the requests.

Another crucial responsibility of the SMF is the selection and control of the user plane function. This includes managing the UPF to perform proxy ARP or IPv6 neighbor discovery. Additionally, the SMF can handle forwarding all the aforementioned traffic to the SMF, specifically for Ethernet PDU sessions.

It is important to note that the SMF is the network service in 5G that configures the traffic steering to the UPF, ensuring the traffic reaches the correct destination.

The SMF like most other functions provides a network function service to the other network services. The network function service specification can be found in the 3gpp specification[4].

Unified Data Management (UDM)

Unified Data Management (UDM) is a 5G core system component responsible for handling structured data in the 5G core network. In short, the UDM generates 3GPP AKA authentication credentials. This means that the UDM most interestingly handles user identifications, which includes storage and management of SUPI (Subscription permanent identifier) for each subscriber in the 5G system, de-concealment of privacy-protected subscription identifier (SUCI), access authorization based on subscription data, UE's Service network function registration management, support service/session continuity, MT-SMS delivery support, lawful intercept, Subscription management, SMS management. Finally, support for external parameter provisioning and disaster roaming.

The UDM provides a network function service that enables other services to communicate with the component, and its definitions can be found in the 3gpp specification[4].

Unified Data Repository (UDR)

The unified data repository(UDR) is a component in the 5G system responsible for storing and retrieving data. More specifically, subscription data to the UDM, policy data to the PCF, and structured data for exposure. It also handles application data, which includes Packet flow descriptions, Application function request information for multiple UEs. To make these mechanisms accessible for other network functions, the UDR has a network function service, which can be found in the 3gpp specification[4].

This component is relevant for application orchestration because it handles much of the data that we need.

Network Data Analytics Function (NWDAF) and Analytics Data Repository Function (ADRF)

Network Data Analytics Function (NWDAF) can support these features: Data collection from other network functions, application functions, and OAM. The NWDAF can handle service registration and meta-data exposure to other network and application functions. The NWDAF can support analytics information provisioning to other network and application functions. Finally, it can support machine learning model training and provisioning to other NWDAFs, given that they contain the analytics logical function. The NWDAF is accessible to other network functions through its network function service, which can be found in the 3gpp specification[4]:

The NWDAF retrieves and stores its data in another component called the Analytics Data Repository Function ADRF. The ADRF provides a network function service to achieve this.

Service Communication Proxy (SCP)

The Service communication proxy(SCP) is a service that provides indirect communication, delegated discovery, message forwarding, and routing between network functions and other SCPs. It also provides communication security, load-balancing monitoring, and overload control. In addition, it can optionally interact with the UDR to resolve identities.

The SCP does not provide a network function service interface, so other network functions can not communicate with it.

Network Exposure Function (NEF)

The network exposure function (NEF) is a service component that exposes internal network function services to external services outside the 5G networks, such as a potential application orchestrator mechanism.

Based on the 3GPP specification[4] the NEF provides the following functionality: The NEF exposes capabilities and events in the 5G core network to 3rd party applications, external application functions (AFs), and edge computing. It stores and retrieves information as structured data using the UDR(Unified Data Repository).

The NEF is capable of secure provisioning of information from an external application function to the network described by 3GPP[4]. This means that

Application functions (Applications in general) have a secure way of providing information and data to the 5G core network, including but not limited to expected UE behavior, time synchronization service information, and service-specific information. Note that the NEF can authenticate and assist in throttling the application function providing the information.

The NEF can provide translation of internal-external information, meaning it can translate information exchanged with external application functions and information exchanged with the internal network function. An example of this is provided in the 3GPP specification[4], which states that the NEF is capable of translating information between an Application Function Service Identifier and internal 5G core information such as S-NSSAI. Meaning the NEF is capable of masking network information and user-sensitive information to external applications according to the core network provider's network policy, which is relevant for Section 4.3.2.

The NEF can redirect application functions to more suitable NEF/L-NEFs when for instance serving an Application Function request for local information exposure and detecting there is a more appropriate NEF instance to serve the Application Function request.

The NEF collects information from other network functions based on the exposed capabilities of other network functions. It stores this information as structured data using a standardized interface to Unified Data Repository. When this is done, the information can be accessed through a re-exposed interface to other network functions, application functions and, external applications. This is a feature highly relevant for retrieving location information about users.

The 5G core provider may have a single NEF to support the functionalities described above or a subset of the functionalities. So it is assumed that it provides most of the features that are listed.

The NEF has quite many network function services, and they are defined in 3GPP specifications[4]:

Finally, for third-party developers who want to develop third-party applications to using the NEF, 3GPP has provided a north-bound API that is intended for third-party developers to interact with. The north-bound API will be used for this thesis.

2.2 Previous Work

This thesis builds upon prior research conducted in a capstone project. The capstone project assessed the feasibility of implementing an application migration mechanism to automatically move third-party applications between edge data centers, where the goal was to minimize latency accessing the applications.

The capstone project accomplished this by designing and running a simulation of a 5G-like network environment with multiple edge data centers. The simulation featured an application orchestrator that monitored usage patterns and network delays to determine the optimal location for an application.

The simulation consisted of 5 components:

Client simulator that represents users that wanted to use an application.

Base station simulator which was intended to represent a gNodeB(RANs).

UPF simulator which is a simplified abstraction for the 5G network where its main responsibilities were to find and forward user traffic to the right edge data centers.

Edge data center which acts as a simplified data center that contained simplified applications.

Orchstrator which was a component that checked the traffic patterns of the users accessing the applications, and moved the applications to as optimal locations on the edge data centers as possible.

The simulation results showed that, on average, an application orchestration mechanism led to lower latency for most users and reduced standard deviation. This thesis expands upon these findings to further explore the potential benefits of the application migration mechanism.

The limitations of the previous work were that the UPF simulator was that it was poorly designed and needed to be constantly updated to know where each application was located in each instance. Additionally, the simulation only focused on the orchestration of one container-based application at a time.

2.3 Related Work and Projects

2.3.1 Consolidate IoT Edge Computing with Lightweight Virtualization

[1] focuses on presenting an in-depth analysis of the current requirements of edge computing and light virtualization, where it focuses on harnessing the power of the Internet of Things. In this context edge computing is a compliment to the powerful centralized data centers with a large number of nodes that provides virtualization close to the data source and end users. It discusses and compares the applicabilities of containers and uni-kernels as light virtualization technologies, where the focus is on enabling scalability, security and manageability. Finally, the article tries to inspire further research by identifying open problems and highlighting future direction to serve as a road map for industry and academia.

This article is included to build a basis on why we are researching the topic that we are. The article argues that mobile operators and containers are a solution for moving applications and data closer to the users. This implies a greater need for faster deployment and better infrastructure.

2.3.2 Edge-computing-driven Internet of Things: A survey

In [2] presents a survey that looks at the impact of edge computing on the development of the Internet of Things and it points out why edge computing is more suitable for IoT than other computing paradigms. In this context, the paper presents edge computing as placing data processing in near-edge devices instead of in remote cloud servers. They claim it is promising to build a more scalable, low-latency IoT system. The main idea of edge computing is placing data processing in near-edge devices instead of remote cloud servers. It is promising to build more scalable, low-latency IoT systems. Many studies have been proposed on edge computing and IoT, but a comprehensive survey of this crossover area is still lacking. The paper categorizes recent advances in the field from top to bottom covering six aspects of edge computing-driven IoT, where it concludes with the lessons learned and proposes some challenges.

[2] is included because it helps to establish a basis for why we would want application migration and edge-driven computing for areas such as IoT.

2.3.3 Service-Oriented MEC Application Placement in a Federated Edge Cloud Architecture

[5] addresses the challenging question of where to deploy a set of Multi-access Edge Computing(MEC) applications on a federated edge infrastructure accessible by 5G networks. Especially when they need to meet the requirements of the applications, considering both computational resources and latency while maintaining availability. The paper formulates the service placement as an integer linear problem, intending to balance the computational load between the available Mobile Edge Platforms (MEP) while respecting the application latency requirements and MEP service availability. They conclude that this is an NP-hard problem, and to solve this problem efficiently, they propose an algorithm based on Tabu-Search (TS) meta-heuristics.

This paper is included because it suggests a MEC application placement strategy and is highly relevant to our use case. Unfortunately, the paper is quite heavy and does not quite go into the specific implementations of how to do their suggested algorithm. Due to time constraints, it is unfortunately only included as an alternative orchestration mechanism and mentioned in Section 4.3.6.

2.3.4 End-to-end simulation environment for mobile edge computing

[6] looks into and analyses end-to-end latency in simulated 5G network architectures. It looks at proven simulators such as SUMO, OMNeT++ and CloudSim, where the goal is to provide large-scale urban mobile simulation environments. The paper discusses the benefits and challenges of such an approach and provides example results that showcase some aspects of the potential end-to-end delay analysis by collecting and process gathered simulation output.

[6] is included because it provides alternatives to the previous work and some useful lessons for this thesis as well.

2.3.5 Open5GS

Open5GS is an actively developed open-source implementation of a 5G standalone core and EPC + 5GSC combination An overview of its architecture can be seen in Figure 2.3.

It implements most of the features specified in 3GPP release 17[4]. The implementation itself is written in C. This project is included in this thesis because it

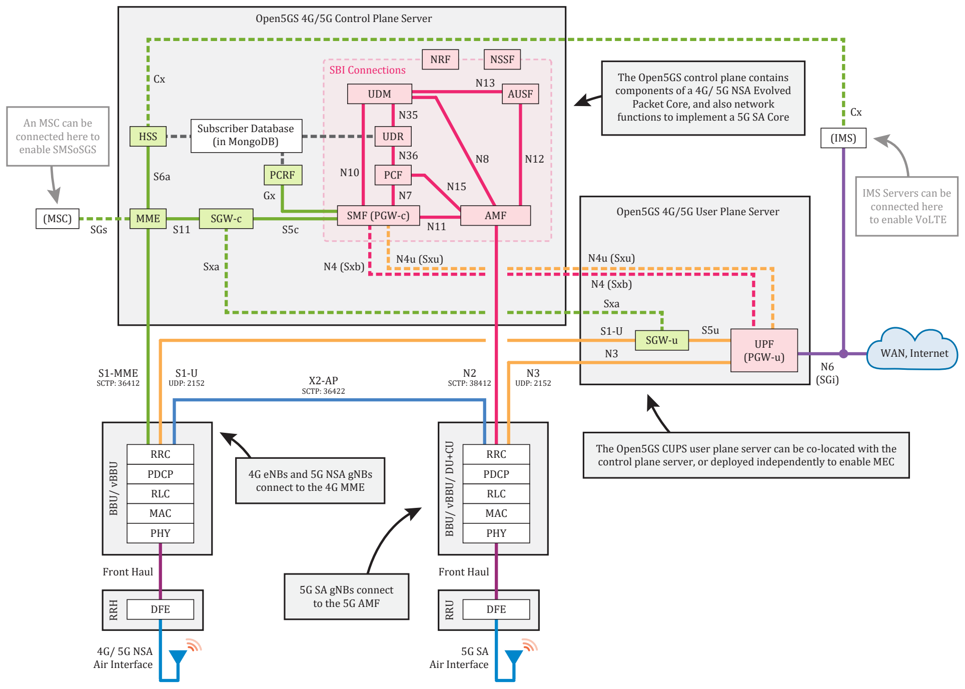

Open5GS 4G/5G Control Plane Server
SBI Connections
NRF
NSSF
UDM
N13
AUSF
N35
UDR
N36
PCF
N10
N8
N12
N15
N7
SMF (PGW-c)
N11
AMF
HSS
Cx
Subscriber Database (in MongoDB)
PCRF
Gx
S6a
MME
SGW-c
S11
S5c
SGs
(MSC)
An MSC can be connected here to enable SMSoSGS
Sxa
N4 (Sxb)
N4u (Sxu)
The Open5GS control plane contains components of a 4G/ 5G NSA Evolved Packet Core, and also network functions to implement a 5G SA Core
(IMS)
IMS Servers can be connected here to enable VoLTE
Open5GS 4G/5G User Plane Server
S1-U
SGW-u
S5u
N3
UPF (PGW-u)
N6 (SGi)
WAN, Internet
S1-MME SCTP: 36412
S1-U UDP: 2152
X2-AP SCTP: 36422
N2 SCTP: 38412
N3 UDP: 2152
The Open5GS CUPS user plane server can be co-located with the control plane server, or deployed independently to enable MEC
BBU/ vBBU
BBU/ vBBU/ DU+CU
RRC
PDCP
RLC
MAC
PHY
4G eNBs and 5G NSA gNBs connect to the 4G MME
5G SA gNBs connect to the 5G AMF
Front Haul
RRH
RRU
DFE
4G/ 5G NSA Air Interface
5G SA Air Interface

Figure 2.3: Open5GS architecture overview

is a 5G core implementation and it is investigated more in Section 2.4.2

2.3.6 srsRAN

srsRAN [7] is a 4G and 5G software radio suite developed by Software Radio Systems.

From the documentation, they deliver a complete RAN solution that is compliant with 3GPP and O-RAN Alliance specifications. This means that the project delivers a full L1/2/3 stack with minimal external dependencies. This makes it possible to set up UEs, gNodeBs, and eNodeB in software and connect them to 4G and 5G cores networks.

This project is included in the thesis because Open5GS references this project as a proof of concept in its documentation. It is also elaborated more in Section 2.4.2

2.3.7 medianetlab: NEF_emulator

NEF_emulator[8] is a emulator that simulates UEs and gNodeBs. The *NEF_emulator* exposes a small subset of the 3GPP Northbridge OpenAPI specification, which makes it possible to fetch information about the location and connectivity of the simulated UEs.

This project is included because it looks promising to achieve application orchestration due to having a minimal NEF API implementation. This project is explored in Section 2.4.2

2.3.8 Nokia: Open5Glab

Nokia's Open5Glab is a 5G network test suite that enables users to develop applications that interact with a simulated 5G network through their NEF API implementation. Their NEF implementation offers support for the following functionalities:

- Obtaining monitoring events from users.
- Changing and interacting with the chargeable party.
- Allowing application developers to establish an AS session with QoS parameters.
- Setting up NIDD sessions with UEs.
- Configuring network parameter settings for UEs.
- Managing PFDs (Packet Filter Data).
- Configuring resource management for background data transfer in UEs.
- Performing traffic influence on UEs.
- Obtaining analytics exposure of UEs and 5G networks in general.
- Configuring enhanced converged restriction control.
- Exposing an API for group message delivery via xMB for MBMS (Multimedia Broadcast Multicast Service).
- Exposing an API for notifications of UE-updated location information.

- Exposing a telephony API that facilitates interaction with call notifications, third-party call control, user interaction, and SMS.

In short their offer parts of the NEF API with some additional services. This project is included because it looks promising to achieve application orchestration due to having seemingly complete NEF API implementation. This project is explored in Section 2.4.2

2.3.9 Camara project

CAMARA[9] is an open-source project that aims to define, develop and test telco network capabilities. Meaning they intend to define usable APIs which are easy and ergonomic to use. The CAMARA project is part of the Linux Foundation and works closely with the GSMA Operator platform group to define these APIs.

The CAMARA project is included in this thesis because it tries to address the complex API definitions in 3GPP. This project is mentioned in Section 4.3.7

2.3.10 CDN: Content Distribution Networks

This paper[10] shows and presents Content Distribution Networks which is an effective approach to improving Internet service quality. CDNs in general are about replicating content and resources so that they are as close to the users as possible. The paper gives an overview of CDNs, including the critical issues which are involved in designing and implementing effective CDNs, and some approaches to handle these issues. Finally, the paper presents a fast service location for peertopeer systems.

CDNs are included in this thesis because of the similarity of application orchestration in terms of replication and most importantly resource management. It will be used in Section 4.3.6

2.4 Investigation of orchestration strategy using 5G

In the real world, humans or machines connected to a 5G mobile network are capable of moving around, i.e. changing their geographical location. In a (future) 5G network, there may be many edge data centers spread around at different geographical locations. All these edge data centers are capable of hosting web-based services with APIs that can be used by humans or machines while they potentially over time change their location.

If we assume that it is possible to deploy different applications to specific edge data centers and assume that it is possible to change the edge data center where specific applications are hosted, then we have a constantly changing optimization problem when it comes to the distribution of network traffic and edge data center load. We can also envision that it could be possible to distribute (duplicate) services to more than one edge data center on demand, but this part of the optimization problem has not been the focus of this thesis and will be left for future work. The entity that resolves and handles this optimization problem will in this thesis be called the orchestrator. The orchestrator needs information to be able to solve the optimization problem. For the orchestrator created in this thesis, the available information is assumed to be:

1. Information from the network about the geographical location of the User Equipment (used by humans or Iot(Internet of Things) devices).

2. Information from the edge data centers about which UEs are at any given point in time have accessed an application hosted in a specific edge data center.

Using this information, the goal of this thesis is to design and develop an orchestrator that solves the optimization problem that arises.

2.4.1 Location and traffic pattern monitoring

The goal is that we want to design an orchestration mechanism that picks up the locations of the UEs and their traffic patterns. And with traffic patterns, we refer to the number of times users are accessing available applications on the open internet. This section aims to describe a possible solution on how to achieve this, with some selected technical details.

Firstly we will look into how to retrieve the information we need from the 5G network and how to process it. From Section 2, we can conclude that there is possible to access a 5G standalone core network for an external application function, which is through the NEF. In 3GPP the API defined to interact with the NEF for third-party applications is called the north-bound API From here on we will refer to this as the NEF API.

With this in mind, this is the architectural proposition for the orchestration mechanism: We assume that the orchestrator knows about all the applications that the cloud provider wants to orchestrate. Next, we also require that we have access to the telecom providers' 5G network and can interact with the network through the NEF API.

In the data network, we have edge data centers at various locations, where the orchestrator can deploy and remove the user-submitted applications. We assume that the applications are accessible to all users. We also assume that the network route is updated when we move an application from one site to another. This is so we can assure that the application is available for the users most of the time.

For the orchestrator service to orchestrate an application so it is closest to the users, this thesis proposes to do it in the following way:

Keep track of all the relevant users that uses the application, and move the application so that they are closest to most of the users. To do this we need the locations of the users that use the applications, including their associated IP address. To collect this information the orchestrator needs a service that subscribes to the 5G network `3gpp-monitoring-event`, which is a functionality provided by the NEF API. The aforementioned service logs the events down to a database.

The 3gpp-monitoring-event is a service exposed by the NEF API, and it lets consumers of the API subscribe to events happening in the 5G network. The monitoring event has many different events defined. However, for achieving application orchestration the most interesting events for orchestrating are the LOCATION_REPORTING and PDN_CONNECTIVITY_STATUS events. The former gives us updates on where a specific UE is located. The latter gives notifications when a UE establishes a PDU session and when the PDU session is released. In the PDN_CONNECTIVITY_STATUS event, it states if the UE establishes an IP-based PDU-session. It can also establish Mac-based and NIDD-based sessions as well. We are interested in the IP-based PDU-session as they are the ones we can use to correlate with the traffic at the application. And thus find an approximation of the location of a user.

Next, we need to log who is accessing the applications on the data network side. This can be as simple as the edge data center logging the IP and timestamp of the user accessing the application.

The orchestrator can then group the information logged from the edge data center and the 5G network to suggest an optimal placement for all applications. In this thesis, we base this on the weighted average location of all the active users of each application, where we minimize the distance between the edge data center and the mean location of the users. We want to do it in this way because the distance is closely related to the time used to traverse the distance. By minimizing the distance, we are also minimizing end-to-end latency for the users. The end-to-end latency is the duration from when a user sends a request until it is processed and returned by the application. The weighted average location is calculated by taking the location of an IP multiplying it with the times it has been accessed, and finally dividing the result by the sum of the weights.

2.4.2 Investigating exsisting solutions

To provide something more valuable to the previous Section 2.4. It is appropriate for some practical work to verify the orchestrator strategy proposed. The goal of this practical work is to investigate if there are some already existing 5G solutions that implement the NEF API and can produce data so that it is possible to implement an orchestrator.

The existing solutions investigated and tested during this thesis were:

1. The open source projects Open5Gs[11] in combination with srsRAN[7]
2. Medianetlab's NEF_emulator[8]
3. Nokias 5G NEF simulator: Open5Glab[12]

These three projects were chosen on the basis that they looked promising and had seemingly many of the features we are interested in.

Firstly the Open5Gs and srsRAN were interesting because Open5Gs offer a mostly complete 5G core. This means most of the components we can see in Figure 2.3. Based on the documentation of Open5Gs it would be possible to use it together with srsRAN, which is a project that lets us create and set up UE's and gNodeBs both with and without hardware that provides trancievers[2].

The idea of practical work here was to use a 5G core running Open5Gs and with multiple gNodeBs and UEs, this would give us full control over the environment and we could easily interact with our 5G core. With this, we could then implement an orchestrator that uses the information available from the 5G core to orchestrate and move applications between edge data centers with the strategy proposed above.

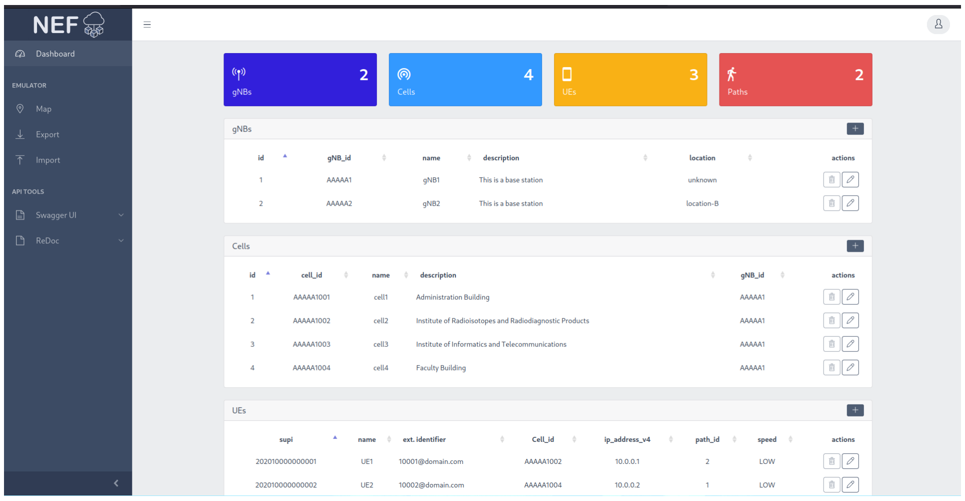
Unfortunately, this is not possible due to Open5Gs not implementing the NEF

2. Examples can be found in the Open5Gs and srsRAN documentation

API[3]. Another limitation is that time, resources and funds are limited to explore this variant of an experiment.

The next consideration was Medianetlab's *NEF_emulator*. As mentioned in Section 2.3, *NEF_emulator* is a 5G emulation suite that offers a limited set of the NEF API.

The simulation is unique because it is possible to define and, dynamically add UEs and gNodeBs (RAN) to the simulation. One of the most promising features is to add tracks to the users so that they can move in and out of the range of the RANs. This would create monitoring event notifications that are retrievable using the NEF API. Finally, the emulator provides a great web user interface for us to interact with, which can be seen in Figure 2.4 and 2.5



gNBs

id	gNB_id	name	description	location	actions
1	AAAAA1	gNB1	This is a base station	unknown	
2	AAAAA2	gNB2	This is a base station	location-B	

Cells

id	cell_id	name	description	gNB_id	actions
1	AAAAA1001	cell1	Administration Building	AAAAA1	
2	AAAAA1002	cell2	Institute of Radioisotopes and Radiodiagnostic Products	AAAAA1	
3	AAAAA1003	cell3	Institute of Informatics and Telecommunications	AAAAA1	
4	AAAAA1004	cell4	Faculty Building	AAAAA1	

UEs

supi	name	ext. identifier	Cell_id	ip_address_v4	path_id	speed	actions
202010000000001	UE1	10001@domain.com	AAAAA1002	10.0.0.1	2	LOW	
202010000000002	UE2	10002@domain.com	AAAAA1004	10.0.0.2	1	LOW	

Figure 2.4: NEF emulator WebUI

The idea here for practical work is to create many RANs and users. Then, have the users move between the RANs to create monitoring event notifications. With this in place, design and implement a data network abstraction that has edge data centers that contain applications. This is to let the users who are connected to the 5G network connect and generate traffic on the applications contained within this data network. Finally with this in place, provide application orchestration by implementing an orchestrator.

The *NEF_emulator* falls short of what we want because it only offers a limited set of the defined NEF north bridge functionality. In addition to that fact that we would have to implement much functionality around the *NEF_emulator* to have applications and edge data centers, that can host the applications. Another

3. However, as of writing this thesis these features are planned, not implemented

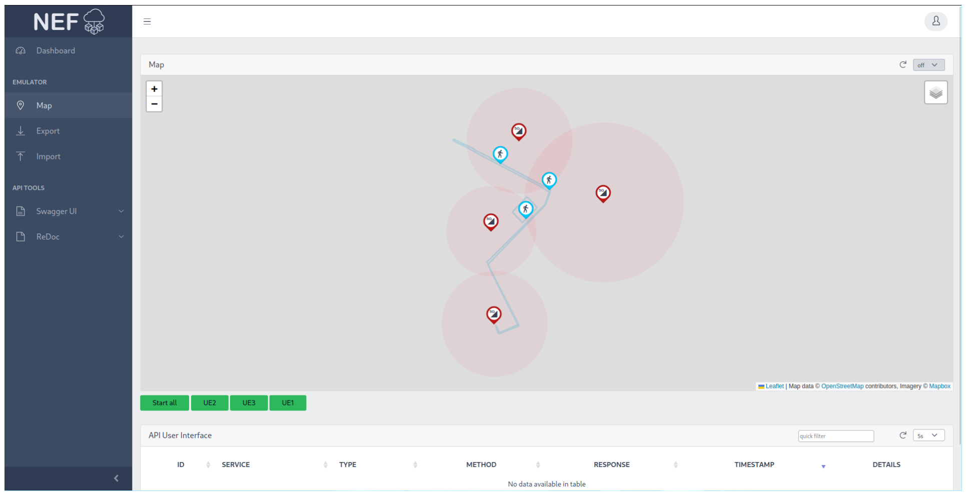


Figure 2.5: NEF emulator WebUI showing the map

complication was to facilitate the users using the data network to access and use the simulated applications located at the edge data centers. The overhead of integrating with the *NEF_emulator* would be too time-consuming and we would create a dependency on this project and it could suddenly change and break our practical work as it is being actively developed.

The third and final project considered as a baseline for the practical work was the Open5Glab from Nokia. The Open5Glab from Nokia exposes a mostly complete NEF API implementation with some additions. They offer the 4G compatible part of the NEF API, which means that the 5G network they use is in a combination with 4G. During this thesis, we applied to get access to this project to test and develop the orchestrator.

While Nokia's NEF API was the one which was most feature complete, the users we were granted access to within this system did not move so they never established any PDU-sessions, which would trigger the PDN_CONNECTIVITY_STATUS monitoring event or any meaningful updates for the LOCATION_REPORTING event. This results in that we would only get a feel of how the orchestrator would work in a static environment, which is not ideal. As it becomes difficult to validate that the orchestrator working properly.

From the existing projects, all of them offer features we are interested in. *Open5Gs* and *srsRAN* offer a mostly complete 5G core, however, they lack the NEF API functionality, which is crucial to do the orchestration. *NEF_emulator* offers most of the features we are looking for. However, they lack the edge data center and application abstractions, which is cumbersome and time-consuming to implement and integrate with this system. The same applies to Nokia's

Open5Glab, They offer a mostly complete NEF API interface. However the UE's we have access to never move, which is a critical part we need to implement orchestration. To address these issues in this section, this thesis will shift to design and implement a simplified mobile network emulator that addresses the lacking features of the aforementioned projects, and try to complete a feature set that we need for application orchestration. And if the first draft is satisfactory we will also implement the application orchestration strategy proposed in this chapter.

3

Methodology

This chapter will explain the problem, the hypothesis and, the work process in detail.

This thesis is based on the following *problem*. If an edge data center at one given location has too many applications this will result in the edge data center throttling and the performance and latency of the applications suffer. Ultimately the end users suffer. To solve this *problem* we propose this *hypothesis*: If the users are connected to 5G and use the applications, is it possible to use information about the user's locations and their usage pattern at each application to suggest a more optimal distribution of all applications such that the overall end-to-end latency gets lowered and we get more capacity at each edge data center?

This thesis is solved by using an agile-based[13] work method. Do note that not everything stated here has been followed, for instance, there have been no teams due to this being an individual master's thesis. Agile software development is an approach to building software that focuses on collaboration, adaptability, and delivering value to the end customer. It is a great alternative to traditional, sequential development methods like Waterfall.

Agile projects are divided into short time frames called iterations or sprints. Each iteration typically lasts from one to four weeks and involves planning, development, testing, and review. For this thesis use case, the time frame for each iteration is about 1 week which includes planning, development and testing. With a meeting with supervisors every two weeks to act as reviews.

Popular Agile methodologies include Scrum, Kanban, Extreme Programming (XP), and Lean Software Development. Each methodology has its specific practices and guidelines, but they all share the core principles of Agile software development. Overall, Agile software development is characterized by its flexibility, collaboration, and customer-centric approach, allowing teams to deliver high-quality software iteratively and adaptively.

The most applicable agile method used in this thesis is Kanban[14]. In short, Kanban is an Agile methodology that focuses on visualizing and managing work in progress. It originated from Toyota's manufacturing practices and has gained popularity in software development.

In Kanban, work items are represented by cards or sticky notes on a visual board. The board is divided into columns representing different stages of the workflow. It helps visualize the flow of work and maintains transparency within the team.

At the beginning of the thesis, the board was replaced with notes and to-do lists. Later in the thesis, a board is in place and an excerpt can be seen in

Figure 3.1. The board itself has six columns which contain:

todo which are items in the backlog.

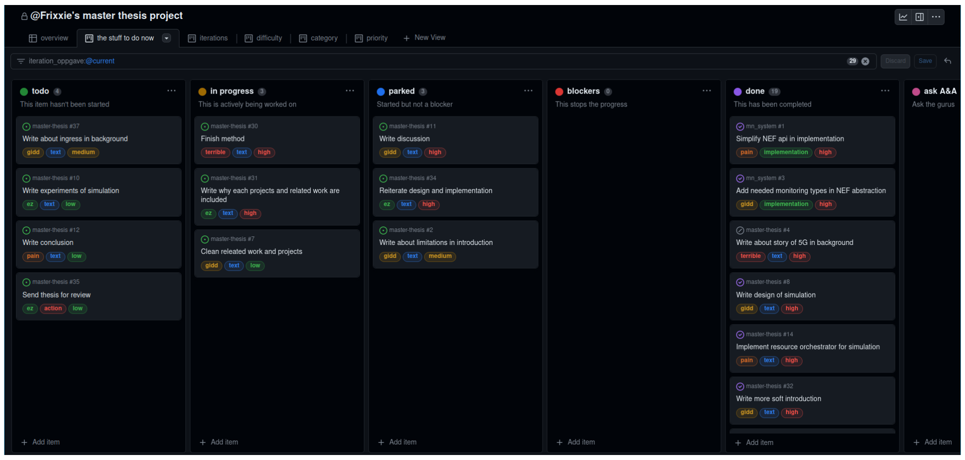
in progress which are the items currently being done.

parked which are items started but not actively being worked on.

blockers which are items that halt progress.

done which are finished items

ask A&A which are items that I need to ask supervisors about.



todo 4
This item hasn't been started

master-thesis #37
Write about ingress in background
gidd text medium

master-thesis #10
Write experiments of simulation
ez text low

master-thesis #12
Write conclusion
pain text low

master-thesis #35
Send thesis for review
ez action low

in progress 3
This is actively being worked on

master-thesis #30
Finish method
terrible text high

master-thesis #31
Write why each projects and related work are included
ez text high

master-thesis #7
Clean releated work and projects
gidd text low

parked 3
Started but not a blocker

master-thesis #11
Write discussion
gidd text high

master-thesis #34
Reiterate design and implementation
ez text high

master-thesis #2
Write about limitations in introduction
gidd text medium

blockers 0
This stops the progress

done 19
This has been completed

mn_system #1
Simplify NEF api in implementation
pain implementation high

mn_system #3
Add needed monitoring types in NEF abstraction
gidd implementation high

master-thesis #4
Write about story of 5G in background
terrible text high

master-thesis #8
Write design of simulation
gidd text high

master-thesis #14
Implement resource orchestrator for simulation
pain text high

master-thesis #32
Write more soft introduction
gidd text high

ask A&A
Ask the gurus

Figure 3.1: This is a screenshot of the Kanban board during the thesis

Kanban introduces work-in-progress (WIP) limits to prevent overloading. Each column has a limit, encouraging the team to focus on completing tasks before starting new ones. This reduces multitasking and improves flow. This has been done in a somewhat capacity, as it helps prioritize however it could have been done better.

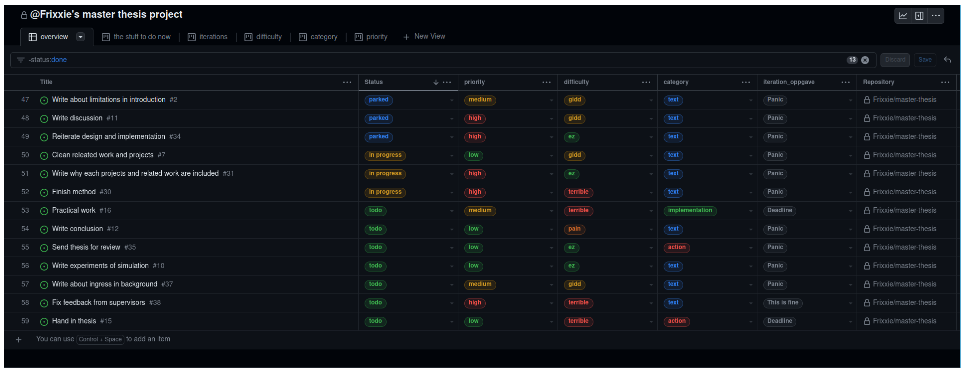
Kanban operates on a pull-based system, where team members pull work from the previous column when they have capacity. This ensures a continuous flow of work and promotes collaboration to resolve constraints.

Continuous improvement is a key aspect of Kanban. By monitoring metrics like cycle times and lead times, the team can identify areas for improvement and implement changes gradually. For this thesis, this has not been a priority.

Kanban is customer-focused, prioritizing work based on customer needs. This ensures that the most important tasks are addressed first. This has been important in our case as we want to design and implement orchestration strategies.

Kanban allows for evolutionary change, making it flexible for teams to introduce and adjust based on their needs. It offers transparency, efficiency, and continuous improvement by visualizing workflow, limiting work in progress, and focusing on customer value. For this thesis, this Kanban has been used loosely.

Other metrics that have been used in the thesis are difficulty from easy to difficult, priority from low to high, finally, each task has been categorized into text, thought, implementation and action. An excerpt of this can be seen in figure 3.2



	Title	Status	priority	difficulty	category	iteration_oppgave	Repository
47	Write about limitations in introduction #2	parked	medium	gidd	text	Panic	Frixxie/master-thesis
48	Write discussion #11	parked	high	gidd	text	Panic	Frixxie/master-thesis
49	Reiterate design and implementation #34	parked	high	ez	text	Panic	Frixxie/master-thesis
50	Clean releated work and projects #7	in progress	low	gidd	text	Panic	Frixxie/master-thesis
51	Write why each projects and related work are included #31	in progress	high	ez	text	Panic	Frixxie/master-thesis
52	Finish method #30	in progress	high	terrible	text	Panic	Frixxie/master-thesis
53	Practical work #16	todo	medium	terrible	implementation	Deadline	Frixxie/master-thesis
54	Write conclusion #12	todo	low	pain	text	Panic	Frixxie/master-thesis
55	Send thesis for review #35	todo	low	ez	action	Panic	Frixxie/master-thesis
56	Write experiments of simulation #10	todo	low	ez	text	Panic	Frixxie/master-thesis
57	Write about ingress in background #37	todo	medium	gidd	text	Panic	Frixxie/master-thesis
58	Fix feedback from supervisors #38	todo	high	terrible	text	This is fine	Frixxie/master-thesis
59	Hand in thesis #15	todo	low	terrible	action	Deadline	Frixxie/master-thesis

Figure 3.2: Screenshot of project overview showing items with all categories

The thesis can broadly be explained in several phases; The first phase included researching and writing about 5G networks, the second phase included investigating orchestration possibilities and testing 5G environments if they provide what we need, and the third phase included designing, developing, testing and writing about the findings from the previous phase, the final phase was about writing and closing out this thesis.

It must be disclosed that ChatGPT[15] has been used in this thesis. ChatGPT is a generative AI tool that has been used to restructure and rephrase some sections in this thesis. ChatGPT in short functions as a chatbot service where it generates output based on the input passed in. It has been used to rephrase some sections by passing them in asking it to rephrase and improve working in those sections. If the wording was better than the initial section it would replace the section, while correcting the technical content.

Finally, the thesis and Mobile Network Emulator implementation has been version-tracked using Git and GitHub.

Here are some relevant statistics for the thesis and emulator: Figure 3.3 and 3.4 show the frequencies of commits for each project. Notice that when there are few commits in the master-thesis project there are quite many commits in the Mobile Network Repository repository. In the master thesis repository, there are about 232 commits, and for the Mobile Network Emulator repository, there are 91 commits and counting. For the Mobile Network Emulator, there are about 2700 lines of code in the implementation.

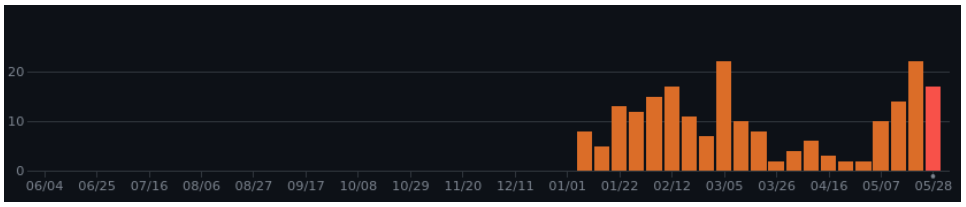


Figure 3.3: Screenshot of commit frequency on GitHub for the master-thesis

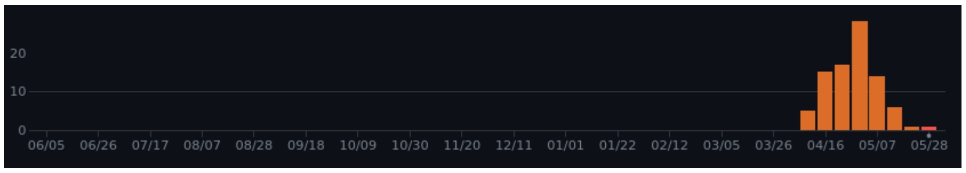


Figure 3.4: Screenshot of commit frequency on GitHub for the emulator

4 Results

4.1 Design and Implementation

The design and implementation of an emulator play a crucial role in the development and evaluation of the proposed orchestration strategy. In the context of this thesis, an emulator is required to address the limitations and gaps found in existing 5G solutions, specifically in terms of implementing the NEF API, generating suitable data and achieving application orchestration. This section presents the design and implementation of *Mobile Network Emulator*, which will be developed as part of the practical work in this thesis.

The primary objective of the emulator is to create a flexible and extensible platform that emulates select functionalities of a 5G network. This includes key elements such as UEs, RANs, a publisher-subscriber interface that resembles the NEF API and finally, a data network abstraction containing edge data centers that can host simplified applications. The emulator should provide the necessary infrastructure to facilitate the movement of UEs within the network, generating monitoring event notifications and enabling seamless interaction with the emulated applications hosted by the edge data centers.

One of the main considerations in the emulator is to ensure compatibility with the NEF API. This allows for the effective management and orchestration of applications and resources within the emulated network environment. By adhering to the 3GPP standards[4], the emulator can serve as a reliable testbed for evaluating the proposed orchestrator and its effectiveness in emulated 5G network scenarios.

Additionally, the emulator aims to address some of the limitations identified in the existing projects, namely in *NEF_emulator* and Nokia's Open5Glab. While these projects offer various features of interest, they lack critical functionality required for comprehensive application orchestration, such as complete interaction with emulated applications and edge data centers and, in Open5Glab's case movement of UEs

To overcome these limitations, the emulator design will incorporate a simplified emulation suite. This suite will encompass the necessary NEF API functionality, enable dynamic movement of users within the network, emulate monitoring event notifications triggered by user mobility, and provide an interface for enabling interaction with applications hosted in the edge data centers. By combining these features, the emulator will offer a powerful tool for evaluating and refining a potential application orchestrator.

The following sections will cover the design and technical implementation of the Mobile Network Emulator, its components, and an orchestrator. The first section will try to give an overview of functionality within the Mobile

Network Emulator. The next subsections will go into more detail about how its components are designed and implemented. Finally, the next section will cover the design and implementation of an application orchestrator that will use this emulator. The subsections are built up this way due to the implementation being close to the logical design.

4.1.1 Mobile Network Emulator

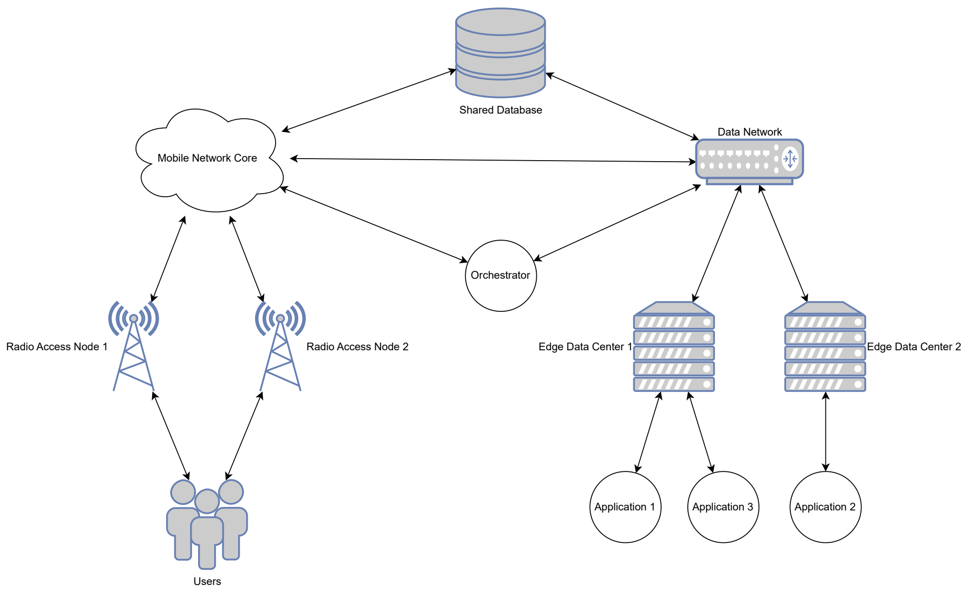


Figure 4.1: Figure that shows the design of the mobile network emulator

Firstly is the mobile network emulator. An illustration of the design can be seen in Figure 4.1. The intention behind the illustration is to show how the logical design of the emulator on a high level.

The big picture is to have the emulator organized mainly into three parts: The first component is the Mobile Network Core, which is intended to emulate the most important parts of the control and user plane of a 5G network. This implies the most critical functionality of the AMF, SMF and UDM. It also emulates user equipment(UEs) that are capable connect to radio access nodes(RANs). When the UEs are connected to RANs they can use the data network that contains edge data centers and applications.

The next component is the Network. The Network component is intended to emulate a data network containing edge data centers and applications. One of the main responsibilities of the network is to handle the application traffic from the users and forward them to the applications hosted on the edge data centers.

This functionality is very similar to the UPF in terms of forwarding traffic to the Data Network which is intended to be the Internet. So when a user wants to access an application it accesses the network which routes the user request to the appropriate edge data center containing the application.

The third component in the emulation is the Mobile Network Exposure. It is intended to emulate the publisher-subscriber pattern which the NEF API uses. The Mobile Network Exposure lets clients subscribe to well-defined Mobile Network Core events and will notify all subscribers of new events. In Figure 4.1 Mobile Network Core and Mobile Network Exposure are merged as the mobile network.

Mobile Network emulator also has frontend visualization, which shows live the current locations of edge data centers, RANs and, users. A screenshot can be seen in Figure 4.3.

The Mobile Network Emulator is implemented using the Rust programming language[16]. The main reason for this is that Rust is a systems programming language that is intended to be fast and memory-efficient. In addition, it has a strong type system and a good package manager called Cargo. For an emulation like this where we want to have a lot of control over the memory and performance, Rust is an excellent choice.

A detailed illustration of the architectural implementation can be seen in Figure 4.2. The figure represents all the data structures and API endpoints defined for Mobile Network Emulator and their relationship with each other. The figure itself is based on a UML diagram however, there has taken some liberties. This is to best represent the implementation as it is not written in an object-oriented language which UML works great to represent.

This figure intends to show how the three main different components interact with each other, where the three different components are marked by different colors. Clients can interact with the components through the defined API endpoints, which act as the point of entry in this figure. This means that there are three points of entry to interact with the emulation, namely Mobile Network Core API Endpoints for internal mobile network features, Mobile Network Exposure API endpoints for NEF API functionality, and Network API endpoints for interacting with the data network. The figure itself will also be handed in as an extra appendix outside this thesis due to its size.

The main function of the Mobile Network Emulator uses all the components found in Figure 4.2. Currently specifying the number of users, RANs, edge data centers and a starting number of applications is done here. When the program itself is started it runs as a state machine where the RESTful APIs are

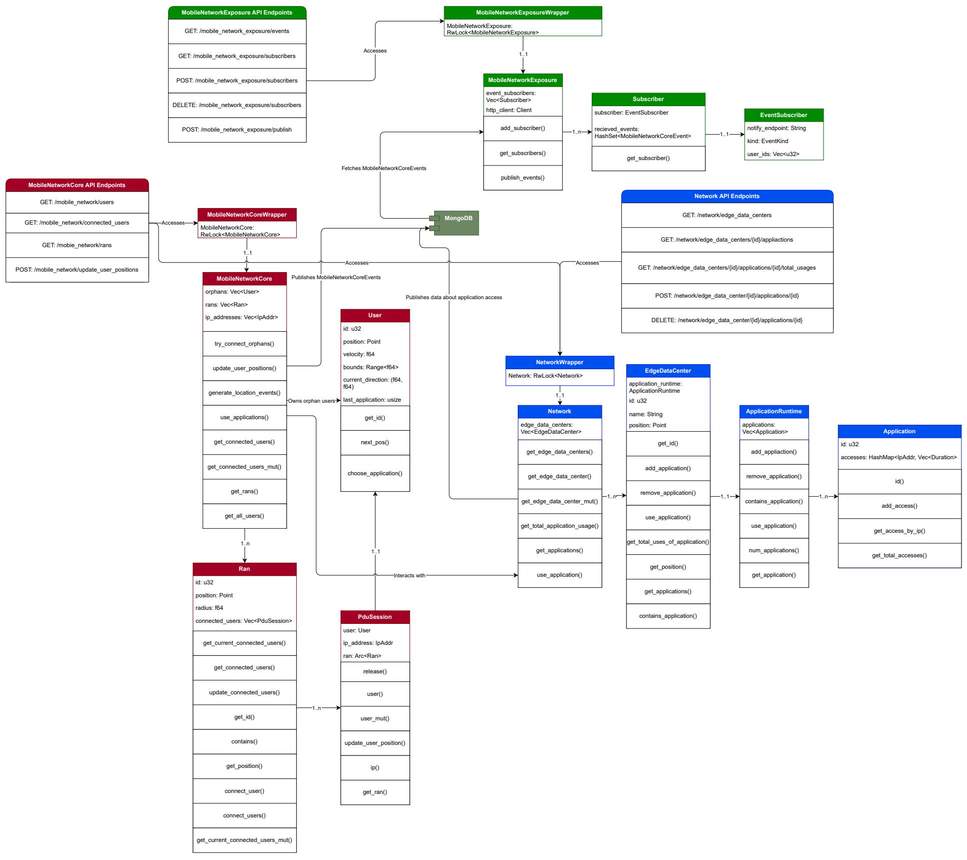

MobileNetworkExposure API Endpoints
GET: /mobile_network_exposure/events
GET: /mobile_network_exposure/subscribers
POST: /mobile_network_exposure/subscribers
DELETE: /mobile_network_exposure/subscribers
POST: /mobile_network_exposure/publish
Accesses
MobileNetworkExposureWrapper
MobileNetworkExposure: RwLock<MobileNetworkExposure>
1..1
MobileNetworkExposure
event_subscribers: Vec<Subscriber>
http_client: Client
add_subscriber()
get_subscribers()
publish_events()
1..n
Subscriber
subscriber: EventSubscriber
recieved_events: HashSet<MobileNetworkCoreEvent>
get_subscriber()
1..1
EventSubscriber
notify_endpoint: String
kind: EventKind
user_ids: Vec<u32>
Fetches MobileNetworkCoreEvents
MongoDB
MobileNetworkCore API Endpoints
GET: /mobile_network/users
GET: /mobile_network/connected_users
GET: /mobile_network/rans
POST: /mobile_network/update_user_positions
Accesses
MobileNetworkCoreWrapper
MobileNetworkCore: RwLock<MobileNetworkCore>
1..1
MobileNetworkCore
orphans: Vec<User>
rans: Vec<Ran>
ip_addresses: Vec<IpAddr>
try_connect_orphans()
update_user_positions()
generate_location_events()
use_applications()
get_connected_users()
get_connected_users_mut()
get_rans()
get_all_users()
Publishes MobileNetworkCoreEvents
Accesses
Publishes data about application access
Network API Endpoints
GET: /network/edge_data_centers
GET: /network/edge_data_centers/{id}/applications
GET: /network/edge_data_centers/{id}/applications/{id}/total_usages
POST: /network/edge_data_center/{id}/applications/{id}
DELETE: /network/edge_data_center/{id}/applications/{id}
Accesses
User
id: u32
position: Point
velocity: f64
bounds: Range<f64>
current_direction: (f64, f64)
last_application: usize
get_id()
next_pos()
choose_application()
Owns orphan users
NetworkWrapper
Network: RwLock<Network>
1..1
Network
edge_data_centers: Vec<EdgeDataCenter>
get_edge_data_centers()
get_edge_data_center()
get_edge_data_center_mut()
get_total_application_usage()
get_applications()
use_application()
1..n
EdgeDataCenter
application_runtime: ApplicationRuntime
id: u32
name: String
position: Point
get_id()
add_application()
remove_application()
use_application()
get_total_uses_of_application()
get_position()
get_applications()
contains_application()
1..1
ApplicationRuntime
applications: Vec<Application>
add_application()
remove_application()
contains_application()
use_application()
num_applications()
get_application()
1..n
Application
id: u32
accesses: HashMap<IpAddr, Vec<Duration>>
id()
add_access()
get_access_by_ip()
get_total_accesses()
1..n
Ran
id: u32
position: Point
radius: f64
connected_users: Vec<PduSession>
get_current_connected_users()
get_connected_users()
update_connected_users()
get_id()
contains()
get_position()
connect_user()
connect_users()
get_current_connected_users_mut()
1..1
Interacts with
1..n
PduSession
user: User
ip_address: IpAddr
ran: Arc<Ran>
release()
user()
user_mut()
update_user_position()
ip()
get_ran()

Figure 4.2: Figure that shows the detailed implementation of the mobile network emulator

used to fetch information about the emulator as well as to progress (step) the emulation forward.

The emulator has an associated MongoDB as the shared database as seen in both Figure 4.1 and 4.2. The mobile network core and network store data in the database, and mobile network exposure fetches data from the database.

The following subsections will cover how the MobileNetworkCore, Network and, MobileNetworkExposure are implemented.

MobileNetworkCore

The MobileNetworkCore is divided into four main components; MobileNetworkCore, User, RAN, and PduSession. In Figure 4.2 the components marked in red are all the components related to MobileNetworkCore.

We will begin looking into the implementation and features of the user(s). A user is composed of an id, position, velocity, current direction, bounds to make sure that they are within bounds, and finally a reference to the last application it has used. The main features of the users are that they can move around on a map using continuously generated paths. This will bring them within and outside the range of RANs.

The RAN is a data structure that contains an id, position, radius, and a list of users that are currently connected to it. RAN provides many utility functions for managing the users and itself. The RAN is capable of updating the positions of the connected users, and if they go out of its RANge it will release the user.

This means that a user can be either connected to or disconnected from the mobile network. If the user is connected or disconnected is managed by the MobileNetworkCore and will be explained later in this section.

When a user is connected to the mobile network it has an established PDU session, much similar to how UEs have PDU sessions when connected to 5G networks. A PduSession is a data structure that encapsulates a user, an IP address and a reference back to the RAN that the PDU session is associated with. It is a utility structure the MobileNetworkCore uses to associate the users that are connected to its RANs. It is mainly a mechanism to associate an IP address with a User.

During each update in the emulator, a connected user will pick an application and send a “use request” via the Network. This means that a user connected to the mobile network can access all applications hosted in different edge data centers. The user also remembers the last application it has used so it can be configured to have a tenancy to reuse the current application it has stored in memory.

Finally, we have the MobileNetworkCore, which is the data structure that manages all users, RANs and, PduSessions. The data structure itself contains a list of all users that are not connected to any RANs, namely orphans, a list of all RANs, and a list of all preallocated IP addresses it has available. The MobileNetworkCore has a couple of features. The first feature is that MobileNetworkCore has a mechanism for connecting users to RANs. This is done by checking which orphans are within the range of a RAN. If this is the case the MobileNetworkCore creates a PDUSession with one of its preallocated ip and places the PDU session encapsulation the user and the ip into the RAN. This is the functionality that is similar to the session management function (SMF) with creating and releasing PDU sessions.

The other feature of the MobileNetworkCore is that it has a mechanism for updating all positions for all users. If any user goes out of range of a RAN the mobile network will check if it belongs to any other RANs, and if this is not the case the MobileNetworkCore will release the PDU session and keep track of where the user is located. This functionality is related to the Access and mobility function (AMF).

For our orchestrator to manage applications, we need some data from the mobile network. To facilitate for this the MobileNetworkCore generates events the users connect and disconnect from the mobile network. As well as when users move in, out and, within range of RANs. These events are defined as MobileNetworkCoreEvent(s) and are currently implemented as two variants:

PdnConnectivityStatus for when PDU sessions are established and released.

LocationInformation for users when they enter, leave or move within RANs.

These events are designed to be as close as possible to the real NEF API definitions found in the 3GPP OpenAPI specification[4]. These also correlate to the events described when proposing an application orchestration strategy earlier in this thesis. The MobileNetworkCoreEvent(s) are stored by the MobileNetworkCore in the shared database.

To interact with the mobile network core, it provides a REST API that can be used to get information about the users, RANs, and PDU sessions. It also exposes an endpoint that updates and generates events. The intention is that the API is only available for internal tools like the frontend and other smaller tools used in the experiments. The specific API endpoints can be seen in Figure 4.2

Network

The network component plays a crucial role in the mobile network emulator as it represents the data network infrastructure. It encompasses the management of edge data centers that host application runtimes containing various applications. This is the component that the users within the MobileNetworkCore use to access the edge data centers and their applications. In Figure 4.2 the components related to the network are marked blue.

The Applications within the emulator are intended to emulate container-based applications. The Application is a data structure that is composed of an id and a HashMap where the key is an IP and the values are a list of timestamps that indicates when an IP has accessed the application. This is to log the accesses from users within the MobileNetworkCore. When a user access an application, it logs the IP address and timestamp when the application was accessed by the IP.

All applications are managed by an application runtime, which is a data structure managed by the EdgeDataCenter. The application runtime is intended to be container orchestrators like Docker or Kubernetes. It is implemented as a data structure containing a list of applications and has functions for adding, removing and using applications.

The application runtime is contained within an EdgeDataCenter. The EdgeDataCenter contains an id, the aforementioned ApplicationRuntime, a name and, a position. The EdgeDataCenter has functions for getting, using, adding, and removing applications from its ApplicationRuntime, and each EdgeDataCenter has a different ApplicationRuntime hosting different applications.

All EdgeDataCenters are connected to a network. The Network manages and exposes an interface to interact with the EdgeDataCenters both internally for mobile network users and externally. The Network has the functionality to let users use applications, as well as add and remove the applications. When a user accesses an application it is the network that is responsible for measuring "the access time" between the user and the application. This is done by measuring the Euclidian distance between the RAN the user is connected to and the edge data center that is hosting the application. This distance value will be used as an emulated latency measure for the experiment in Section 4.2.

To interact with the network, it provides a RESTful API that can be used to get information about the edge data centers and the applications. These endpoints are intended to be used by the orchestrator to get information about the applications and edge data centers. The specific endpoints can be seen in Figure 4.2.

Mobile Network Exposure

Mobile Network Exposure is a component intended to emulate the NEF API functionality found in the 5G networks. It does this by offering a publisher-subscriber service where it enables clients to subscribe to well-defined MobileNetworkCoreEvents. The current events supported are the PdnConnectivityStatus and LocationInformation events, which are corresponding to the events presented in the previous section.

This component fetches the events from the same database where the MobileNetworkCore stores its events. In Figure 4.2 the components marked in green is related to the MobileNetworkExposure functionality. The goal of implementing this component is to have the application orchestrator subscribe to events happening in the emulator.

The MobileNetworkExposure is organized in the following way: The MobileNetworkExposure is the data structure that organizes and keeps track of all subscriptions. A subscription is logically a client who has requested to subscribe to events that we have defined. The MobileNetworkExposure keeps track of all the subscribers by storing them in a list. It also has an HTTP client which it uses to publish new events to the subscribers

A subscriber in this context is a data structure that contains the details of the subscription, namely an EventSubscriber. The subscriber also contains a set of all received events that match the details within EventSubscriber.

An EventSubscriber is a data structure that specifies which kind of events the client subscribes to. Internally the data structure contains the URL the MobileNetworkExposure is going to send new events. It also contains which event type the client is interested in (which currently is either PdnConnectivityStatus or LocationInformation events), and finally, a list of all user ids the client wants events for.

The MobileNetworkExposure has functionality for publishing events. This is done by fetching all events from the database, then iterating through the subscriptions and sending new events that they have not received.

MobileNetworkExposure exposes a RESTful API that lets clients use this service. The users can subscribe to an event by sending a POST request with a JSON formatted body specifying the event type, which UEs, and the URL to which the Mobile Network Exposure should post the events in the body of the request, this data is expressed as the EventSubscriber. This is in principle done the same way as in the NEF API. The MobileNetworkExposure also exposes some utility endpoints where it is possible to fetch all subscribers, and events and finally publish all new events. The specific API endpoints can be seen in Figure 4.2

Frontend

In the mobile network emulator, there is also provided a frontend visualization that shows the edge data centers, RANs and users. In the visualization, the users are defined as blue points, which change colors when they are connected to the data network. The RANs are defined as yellow points with green circles around them to indicate the radius of connectivity. Finally, we have the red squares which are used to indicate where the edge data centers are located.

Figure 4.3 shows a screenshot of this.

The frontend uses the MobileNetworkCore API to fetch information about the RANs and users and the Network API to fetch information about the edge data centers.

The frontend is implemented using Svelte[17] and svelte-p5[18]. It functions by first fetching the RANs from the MobileNetworkCore, and the edge data centers from the Network. It then fetches both the users and connected users from the MobileNetworkCore and draws them on the map, and then repeats this process.

4.1.2 Orchestrator

This section will cover the design and implementation of an orchestrator that orchestrates applications to reduce the end-to-end latency for the users in the Mobile Network Emulator. The main intention behind the orchestrator mechanism is to collect useful data from the 5G network and the edge data centers and use this information to orchestrate the applications.

The orchestrator fetches the IP address access log from the edge data centers and their applications. So for one application, the orchestrator fetches all IP addresses that have accessed it together with the timestamp when the IP was accessed. To find the approximate location of the IP address the orchestrator fetches MobileNetworkCoreEvent(s), which are made available through the MobileNetworkExposure component. The orchestrator has currently two types of events it uses, which are the PdnConnectivityStatus for relating IP address to user id, and LocationInformation for relating user id to location. Both of these events are defined in Mobile Network Emulator and also defined in the 5G NEF API.

When the orchestrator has found the approximate location of an IP address it tries to find the appropriate edge data center for the application. Currently, the orchestrator uses the weighted average location of users to find an edge data center to place the application in. The weighted average location is calculated by taking the average location of all IPs that have accessed the application, where the number of times an IP has accessed the application acts as the weight for each location.

The orchestrator is implemented using the Rust programming language[16] and is designed to run continuously and check for the optimal placement at given intervals, which can be specified.

4.2 Experiment

To verify that the orchestrator reduces the emulated time when users are accessing applications in the emulator. It is therefore appropriate for an experiment. The experiment in this case is to run the emulator with a set amount of users, RANs, edge data centers and applications and see if the orchestrator can reduce the emulated time when users are accessing the applications.

In the experiment, we are measuring the emulated time the takes for a user to access the application. This emulated time is the end-to-end latency from the user sending an "application use request" until it is completed. This is done in the emulation by measuring the Euclidian distance between the RAN the user is connected to, and the edge data center. To penalize users who are connecting to applications hosted far away the network multiplies a factor to each latency. The Network logs these end-to-end latencies down to the shared database. We are also monitoring the application locations in the edge data centers. This is done by using the Network API. We are measuring the application locations because we want to verify that the orchestrator is behaving properly.

The emulation is run with 128 simultaneous users that have a 98% chance to reuse the application it is currently using. The emulation has 8 applications for the orchestrator to manage. The edge data centers and RANs are placed using the Poisson disk sampling[19], where the RANs have on average 150 units of space in between each other and the edge data centers have on average 200 units of space in between. A snapshot of this experiment can be seen in Figure 4.3.

The experiment ran for 5 minutes where in the first half the orchestrator is inactive. Initially, every application is placed in the edge data center with the id 0. This is to create a baseline latency so we can compare when the orchestrator is activated. The orchestrator tries to orchestrate every 10 seconds after it is activated.

Since there are many applications and quite many users accessing the same applications we will calculate the average end-to-end for the last 15 seconds at each data collection. This is to reduce the amount of data points and plots we have to use in this experiment.

4.2.1 Results

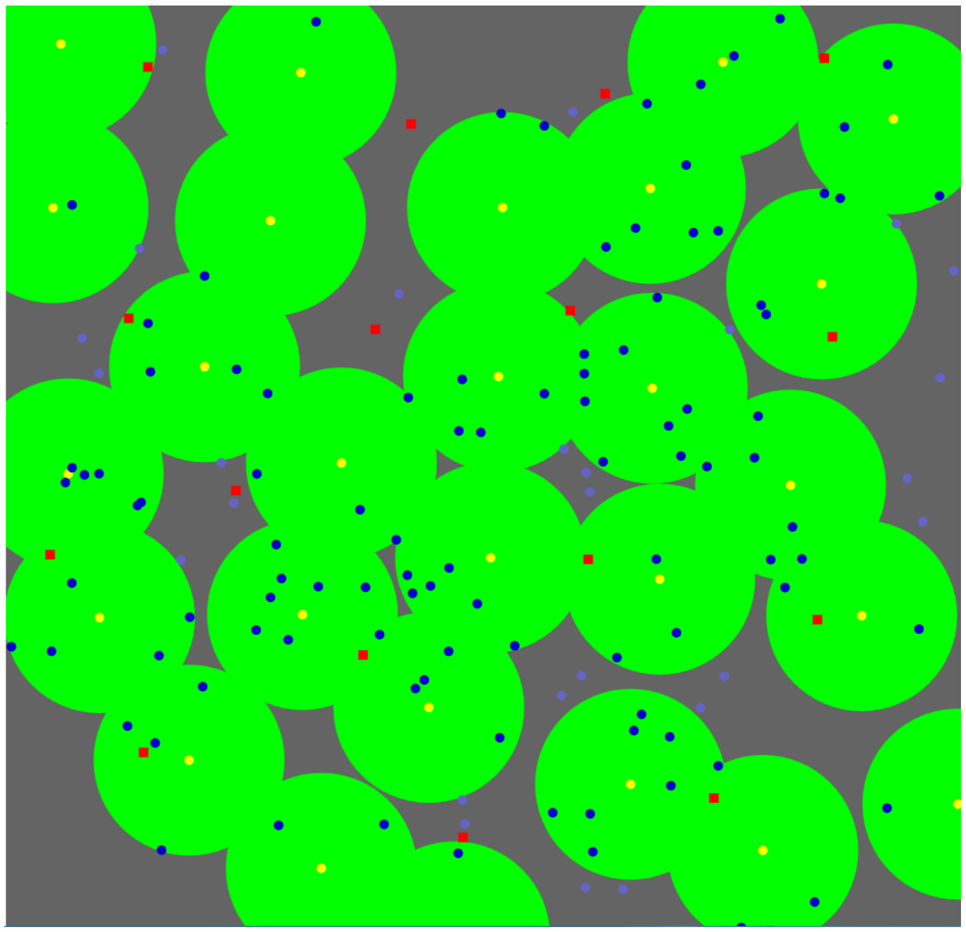


Figure 4.3: Snapshot of the WebUI during the experiments

Plot 4.4 and 4.5 show the average end-to-end latency of the last 15 seconds for applications with id 0 and 1. The x-axis shows the time in seconds when the data was collected, and the y-axis shows the mean end-to-end latency for the last 15 seconds. The bars in the plot are the standard deviation during the last 15 seconds. Notice when the orchestrator is activated the end-to-end latency decreases, which is the goal, and an indication that the orchestrator with the weighted average location works as intended.

Plot 4.6 shows which edge data center hosts which application at a given time. The x-axis shows the time when the data was collected, the y-axis shows which application we are following and the z-axis shows the edge data center the application is hosted on. Notice when the orchestrator is activated the applications get moved from their initial configuration, which is the goal.

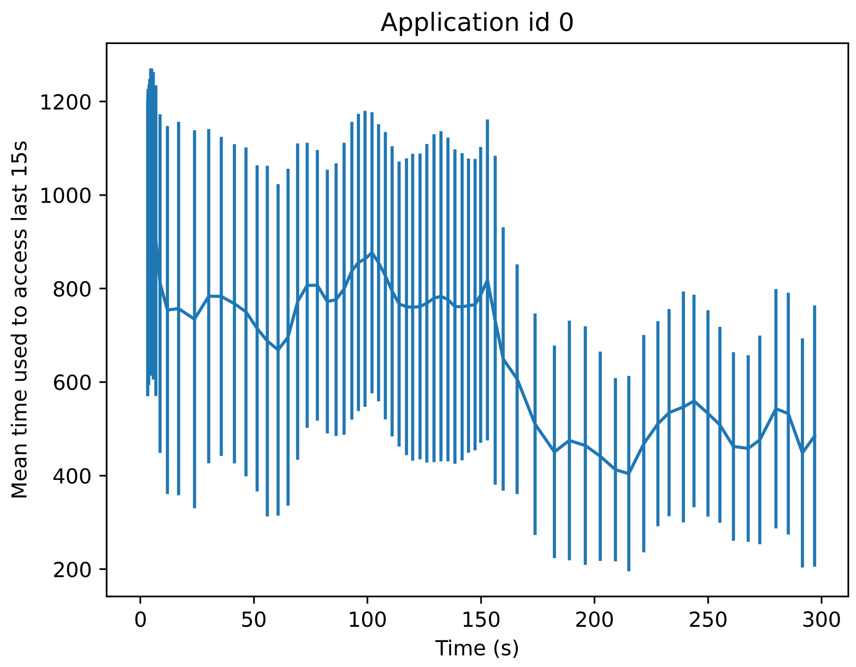


Figure 4.4: Plot showing the average end-to-end latency for appliaction 0

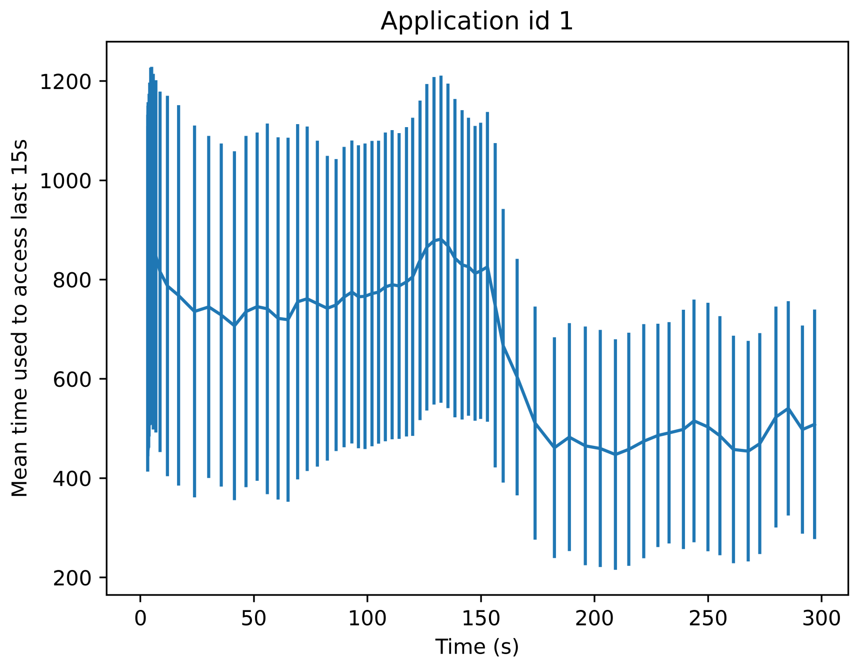


Figure 4.5: Plot showing the average end-to-end latency for appliaction 1

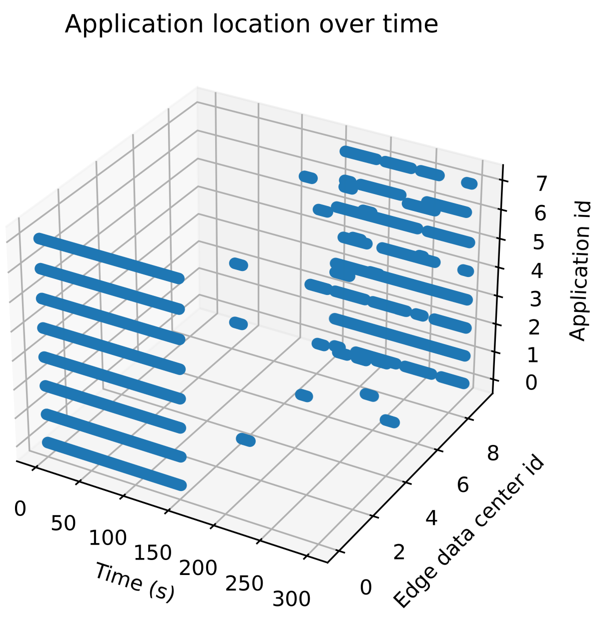


Figure 4.6: Plot showing where every application is located

4.3 Discussion

The following sections will be used to discuss and highlight relevant topics in this thesis. This includes discussing the results and highlighting the limitations of both the emulator and orchestrator.

4.3.1 Results

Firstly the experiments are a success as they achieve the goal of reducing the end-to-end latency from user to application by moving the application between sites. This is evident because we see in plots 4.4 and 4.5 that the latency drops when the orchestrator is activated.

However, a glaring result from the experiments is that the average end-to-end latencies and their trends for all applications are very similar. This is likely due to how the users move and choose applications. In the current state of the emulator, users are evenly distributed over the space that they are moving in, and this affects results quite greatly, This is evident in both plots 4.4, 4.5 and, 4.6.

This can be fixed by changing the movement patterns of the users, and mechanisms of how the users choose which application to use. If the movements and application patterns are more skewed the more effective the orchestrator should be.

4.3.2 Privacy concerns

The proposed solution and approach described in Section 2.4.1 raise privacy concerns. Specifically, the strategy suggested resembles a form of mass surveillance, despite the existence of data aggregation and anonymization mechanisms in 5G. Although the possibilities to make efforts to protect privacy, the substantial amount of data collected still poses notable privacy risks, particularly if the data were to fall into the wrong hands.

Another perspective to consider is that of the edge data center. The orchestrator necessitates access to the IP access logs of these centers. When the data from both the edge data centers and 5G is combined, the user's location and traffic patterns become vulnerable. To address this issue, the orchestrator must either only operate on anonymized data or be operated by a trusted entity, and users must provide explicit consent for their data to be utilized in this manner.

It is essential to acknowledge that these privacy concerns are substantial and require extensive efforts to develop a viable solution. However, addressing these issues falls beyond the scope of this thesis. Nevertheless, it is imperative to resolve these concerns before implementing this solution in real-world scenarios.

4.3.3 Alternative practical work

While Section 2.4.2 presents several ideas for alternative practical work, it is important to discuss additional options.

One obvious alternative for practical work would be to solely focus on utilizing data provided by the NEF API and design and implement an orchestrator based exclusively on the form of this data. However, there are certain drawbacks to this approach. As mentioned in Section 2.4.2, there is a scarcity of platforms that generate relevant based on the NEF API data. This poses challenges in terms of validating the proper behavior of the orchestrator. Additionally, implementing an edge data center abstraction with users and defining patterns for accessing applications would still be necessary, regardless of relying solely on the NEF API data.

Therefore, exploring additional alternatives and considering a more comprehensive approach is crucial to address these limitations and ensure the effective implementation of the orchestrator.

Another alternative would be to look closer at the emulators discussed in [6]. It could be an idea to see if it is possible to implement application orchestration using these emulations, or take aspects from those emulations and integrate them with our emulation. However, this will be left for future work.

4.3.4 Limitations of the emulator

The emulator possesses certain limitations and weaknesses that need to be addressed. This section will outline these limitations, explain their causes, and propose potential solutions.

Firstly, the movement patterns of users within the emulator are not realistic. Currently, users follow a random walk pattern, which does not accurately reflect real-life movement, where users group to several clusters of locations and move in between them. To rectify this, an alternative approach would involve incorporating user patterns from experiments or similar studies where individuals are tracked using technologies like GPS. Or use emulation strategies like the boid flocking algorithm[20] to get clustering behavior.

Another issue pertains to the users' access to applications, which could be improved. Currently, users randomly access applications during each update of the emulator, which deviates from realistic behavior. In reality, users tend to repeatedly use the same set of applications. Although measures have been taken to address this problem by having users behave like infinite-state machines when selecting applications, where the users repeatedly reuse the same application and occasionally change the application to use, further improvements can be explored.

These two points heavily affect the effectiveness of the orchestrator, as highlighted in Section 4.3.1.

The placement of RANs and Edge data centers is another area that could be enhanced. Currently, the Edge data centers and RANs are placed using Poisson disk sampling[19]. However, a more realistic approach would involve comparing the placement with actual 5G coverage maps to achieve better results.

There is currently no strong correlation between IP addresses and User IDs in the emulation. The allocation of IPs in the emulator follows a First In First Out list, leading to a high probability of immediate reuse when a user disconnects. This limits the association between IPs and users. To address this, alternative IP allocation schemes or utilizing a First In Last Out list could be explored to increase IP circulation. This would also enhance the orchestrator's ability to determine user location based on IP.

The emulator's optimization is inadequate, particularly regarding the I/O operations involved in writing mobile network core events and user statistics to the database during each update loop. Initially, these events are stored in memory, which consumes space and made retrieving the data challenging. Improving the optimization of these operations is necessary if we want to scale up the number of users, rans, and edge data centers. Running the emulator over time hurts the performance.

Additionally, there is a lack of diverse orchestration strategies implemented in the emulator. Currently, only one strategy the weighted average position of users is utilized. Expanding the range of orchestration strategies should be considered and further discussed in Section 4.3.6.

Lastly, the implementation of NEF functionality is limited. Although the current implementation suffices for the present use case, there is room for improvement. Section 4.3.7 provides a further discussion on enhancing NEF functionality.

4.3.5 Limitations of the orchestrator

The current implementation of the orchestrator has some limitations, this section aims to give some insights into the limitations and propose some solutions to fix them.

The IP to location scheme can be inaccurate The root cause of this is because of the IP allocation issue described in Section 4.3.4. The orchestrator tries to address this by finding the user id which is the closest associated timestamp when the application was accessed.

No predictions of application usage patterns A fix for this is suggested in Section 4.3.6, where it is suggested that the orchestrator uses time series-based forecasting for the weighted average position of the users.

Currently fetches directly from the MongoDB Due to limited time, the orchestrator currently fetches the `MobileNetworkCoreEvents` directly from the database, the intention was for the orchestrator to use the `MobileNetworkExposure` functionality. However, to save time we fetch directly from the database instead. Note that the mobile network exposure and the events from the database would be the same. So this is a fix for future work.

Available resources at each edge data center is not considered Currently the orchestrator does not consider the resources available at each edge data center. This is only due to insufficient time. The other schemes were prioritized as fetching data from the 5G network and usage patterns from the edge data centers are more critical for the orchestrator to function properly in the first place. Works like [10] could be considered to be used as an inspiration for developing replication mechanisms.

4.3.6 Alternative orchestration strategies

The objective of this section is to propose alternative orchestration strategies that can be employed in the orchestrator.

Firstly we could and probably should implement the tabu search metaheuristics algorithm proposed in [5], however due to time constraints and the fact that the current orchestration strategy is much simpler we prioritized the current strategy

One potential improvement is to utilize time-series-based prediction of user patterns. Given the availability of present and up-to-date user data, an alternative approach could involve employing techniques such as weighted moving average forecasting to predict the average location of users. This enhancement could reduce the frequency of application analysis by the orchestrator, leading to improved resource utilization.

Another alternative orchestration tactic is machine learning-based prediction of optimal location. For instance, a neural network trained on existing orchestration strategies could be utilized. The neural network could take into account the number of times each application has been used from each RAN as input and provide suggestions for the optimal edge data center placement based on this information, incorporating hidden layers and an output layer, where the output layer is an indication on where the application should be placed.

Employing a machine learning-based solution is desirable, especially when handling large amounts of data, as it becomes computationally expensive to calculate the precise edge data center for application placement. However, this approach has drawbacks. The edge data centers must remain static since any changes would require retraining the model to account for the new configuration. Furthermore, the machine learning model may struggle when encountering user patterns it has not seen before, potentially leading to suboptimal decisions regarding application placement.

Other machine learning models could be explored, such as naive multinomial Bayesian classifiers, logistic multinomial classifiers, and multinomial decision trees. The fundamental concept would remain similar to the neural network approach, where a feature vector consisting of the frequency of application usage from each RAN is associated with the optimal edge data center. Determining the most suitable model would require implementation and testing, as different supervised models possess unique characteristics and quirks. However, these options will be left for future work due to time constraints.

4.3.7 The 5G API and its definitions are terrible!

While this section is related to this thesis and is subjective, it is related to the solution and the 5G API and is important to discuss for future work.

In general, a publish-subscribe model is fine to use, however how the data structures are defined and used in the 3GPP specification. It results in an over complicated interface which hard to use, especially if you use strongly typed programming languages such as Rust[16].

In general, the 3GPP OpenAPI specification uses a lot of polymorphic data structures in their APIs. For instance, the “MonitoringEventReport” data structure, is highly used in this thesis. It has only one required field, which is the "MonitoringType". Every other field in the data structure is optional and this implies that you may get a different answer from the API between each request. This makes it very hard to work with the API.

There are projects like the Camara project[9] mentioned in Section 2.3 that try to simplify and standardize the 3GPP specification to be easier to use. The emulator tries to address this by representing the data structures as strongly typed sum types for the different monitoring types. A more elegant solution would be to split the data structure up into smaller and more well-defined data structures and define the domain more clearly using methods like domain-driven design. This would make it more approachable to develop applications functions and services using 5G directly, and not through another standard.

/5 Conclusion

5.1 Future work

Several areas of future work can be explored. This section will identify some of these areas.

Improve emulator as most of the current issues have been discussed in section 4.3.4.

Implement more orchestrataion strategies . As discussed in section 4.3.6, several alternative orchestration strategies can be explored and tested.

Continue to improve the NEF data types . As a continuation of section 4.3.7, the NEF data types can be improved to be more approachable and easier to use.

Investgate privacy issues . As discussed in section 4.3.2 there must be done some form off evaluation if this project should be used. This thesis suggests a risk analysis.

5.2 Conclusion

In this thesis, we have presented a new approach to the problem of designing and implementing application orchestration leveraging 5G networks and its NEF API. While the goal was to only look into the application orchestration using 5G networks, and due to lackluster data generation this thesis shifted towards solving the problem of data generation. The data generation problem was solved by designing and implementing a combination of a 5G network emulator and a data network emulation containing edge data centers and applications. With the help of this emulation, we were able to generate data for the application orchestration problem and suggest an orchestration algorithm that can be used to orchestrate applications in 5G networks, and based on the experiments conducted in this thesis we have achieved the goal of minimizing end-to-end latency for users.

The thesis consists of a literature review, related work and projects chapter that looks into how 5G networks are designed with some related projects and papers. Some of the papers are used to establish the need for more development in this field, some are projects considered for future work, and some are investigated closer in this thesis. The reviewed literature, related work and projects chapter ends with research and provides suggestions to do application orchestration and route optimization using 5G networks. It is then followed by practical work where the thesis looks into how to generate data for the orchestration problem. This is done by looking at three already existing projects Open5Gs + srsRAN, NEF_emulator and Nokias Open5Glab. We found that none of these projects could be used to generate data for the orchestration problem fully, and therefore we decided to create our emulation.

The methodology chapter states the problem, hypothesis and, how work has been conducted during this thesis by using an adaption of Agile and Kanban.

The result chapter looks into the design and implementation of the emulator with some experiments, results and discussion. The results show that deploying the orchestrator in this limited emulator lowers the end-to-end latency on average for most users. The discussion section looks into the results, privacy issues, alternative practical work and, limitations of the emulation and orchestrator, and discusses the 5G NEF API.

The thesis ends with a conclusion chapter that suggests future work and summarizes this thesis.

Bibliography

[1] Roberto Morabito et al. “Consolidate IoT Edge Computing with Lightweight Virtualization.” In: *IEEE Network* 32.1 (2018), pp. 102–111. DOI: 10.1109/MNET.2018.1700175.

[2] Linghe Kong et al. “Edge-Computing-Driven Internet of Things: A Survey.” In: *ACM Comput. Surv.* 55.8 (Dec. 2022). ISSN: 0360-0300. DOI: 10.1145/3555308. URL: https://doi.org/10.1145/3555308.

[3] Qualcomm. *What is 5G?* 2023. URL: https://www.qualcomm.com/5g/what-is-5g.

[4] *5G; System architecture for the 5G System*. 3GPP TS 23.501 version 17.7.0 Release 17. ETSI and 3GPP. Jan. 2023.

[5] Bouziane Brik, Pantelis A. Frangoudis, and Adlen Ksentini. “Service-Oriented MEC Applications Placement in a Federated Edge Cloud Architecture.” In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9148814.

[6] Katja Gilly et al. “End-to-end simulation environment for mobile edge computing.” In: *Simulation Modelling Practice and Theory* 121 (2022), p. 102657. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2022.102657. URL: https://www.sciencedirect.com/science/article/pii/S1569190X22001277.

[7] Open Source Software. *SrsRan*. 2023. URL: https://www.srsran.com/.

[8] Open Source Community. *NEF_emulator*. 2023. URL: https://github.com/medianetlab/NEF_emulator.

[9] Linux fundation projects. *CAMARA*. 2023. URL: https://camaraproject.org/.

[10] Gang Peng. “CDN: Content Distribution Network.” In: *CoRR* cs.NI/0411069 (2004). URL: http://arxiv.org/abs/cs.NI/0411069.

[11] Open Source Software. *Open5Gs*. 2023. URL: https://open5gs.org/open5gs/.

[12] Nokia. *Nokia Open5GLab*. 2023. URL: https://www.nokia.com/developer/open5glab.

[13] Kent Beck et al. *Agile Manifesto*. 2023. URL: https://agilemanifesto.org/.

[14] Jira. *What is Kanban?* 2023. URL: https://www.atlassian.com/agile/kanban.

[15] OpenAI. *ChatGPT*. 2023. URL: https://chat.openai.com/chat.

[16] Steve Klabnik, Carol Nichols, and Rust Community. *The Rust Programming Language*. URL: https://doc.rust-lang.org/book/index.html.

[17] Open Source Software. *Svelte*. 2023. URL: https://svelte.dev/.

[18] tonyketcham and Open Source Software. *svelte-p5*. 2023. URL: https://github.com/tonyketcham/p5-svelte.

[19] Robert Bridson. "Fast Poisson Disk Sampling in Arbitrary Dimensions." In: *ACM SIGGRAPH 2007 Sketches*. SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, 22–es. ISBN: 9781450347266. DOI: 10.1145/1278780.1278807. URL: https://doi.org/10.1145/1278780.1278807.

[20] Craig W. Reynolds. "Flocks, Herds and Schools: A Distributed Behavioral Model." In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 25–34. ISSN: 0097-8930. DOI: 10.1145/37402.37406. URL: https://doi.org/10.1145/37402.37406.

A
Source code for Mobile Network Emulator

While a code listing would be natural for showing the source code of this thesis. The number of lines in the source code exceeds 2700, so instead a link to the GitHub repository is included here: `https://github.com/Frixxie/mobile_network_emulator`

B Additional plots

This appendix will show the additional plots for all other applications during the experiment.

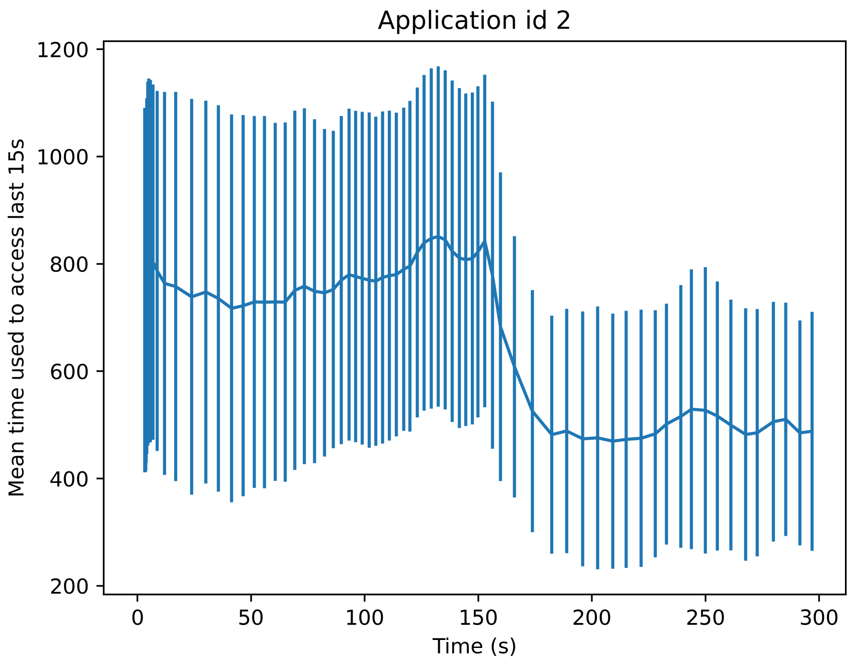


Figure B.1: Plot showing the average end-to-end latency for appliaction 2

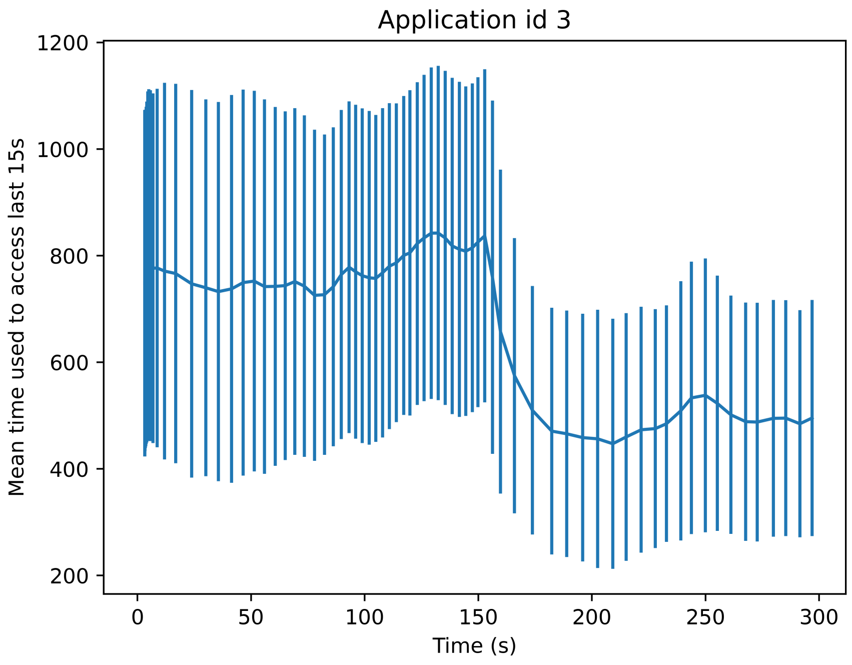


Figure B.2: Plot showing the average end-to-end latency for appliaction 3

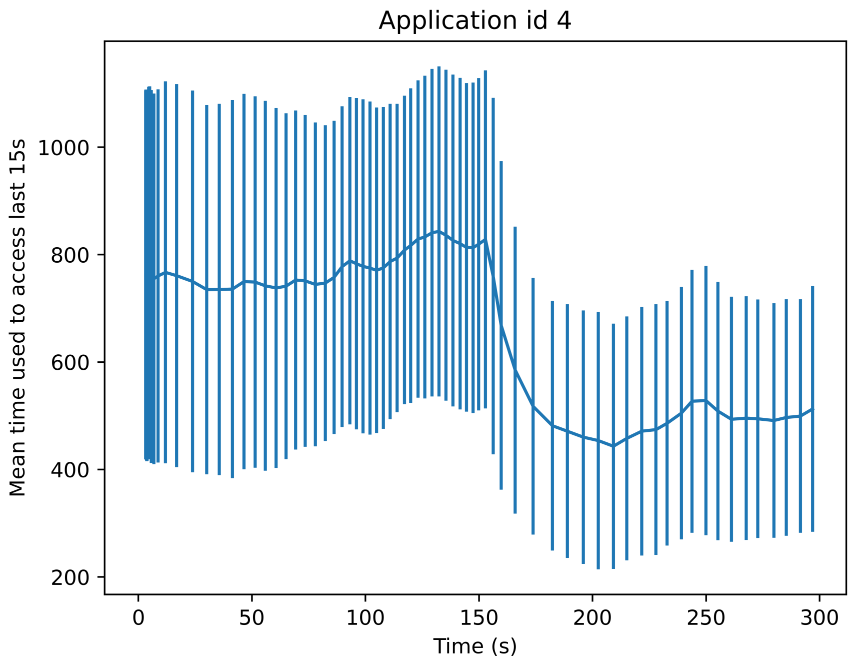


Figure B.3: Plot showing the average end-to-end latency for appliaction 4

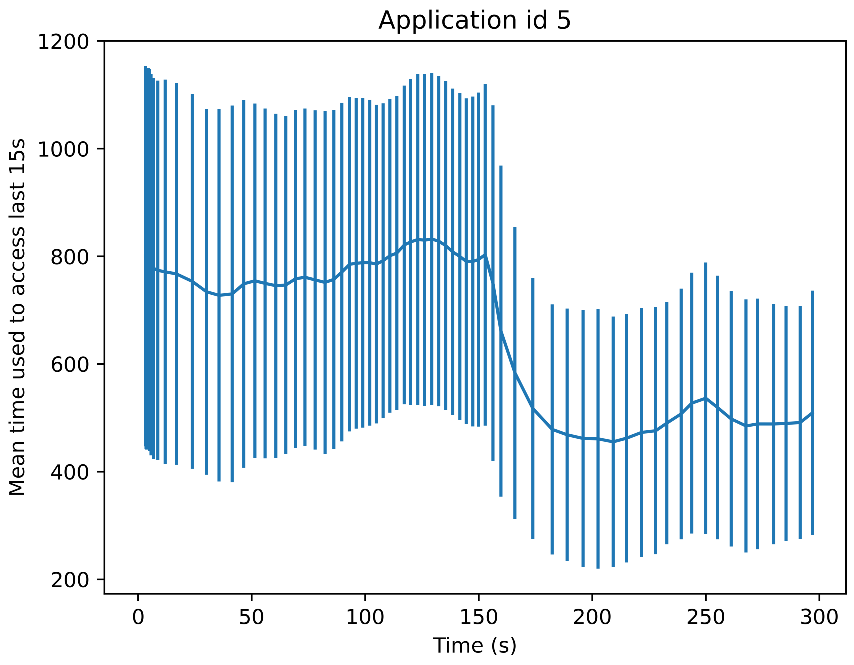


Figure B.4: Plot showing the average end-to-end latency for appliaction 5

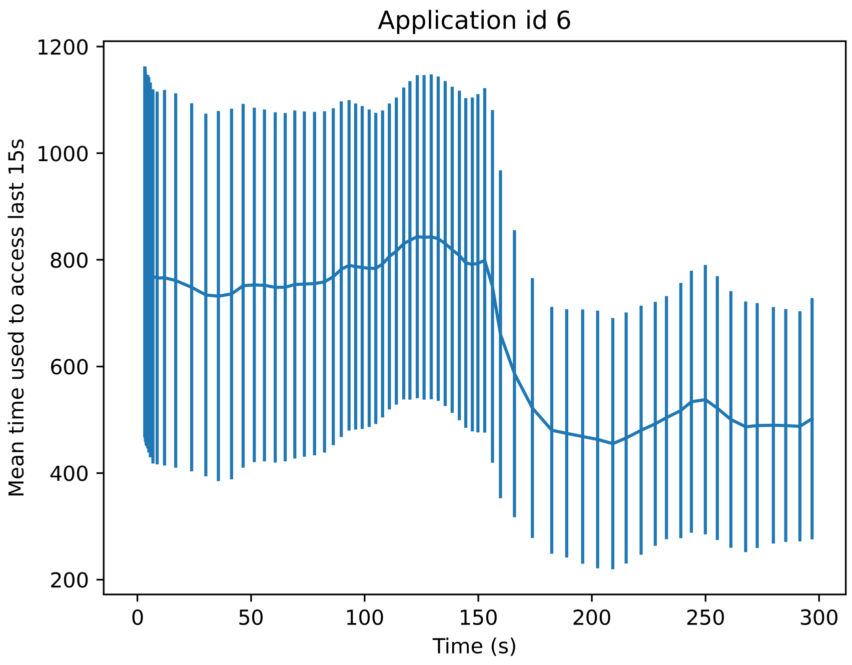


Figure B.5: Plot showing the average end-to-end latency for appliaction 6

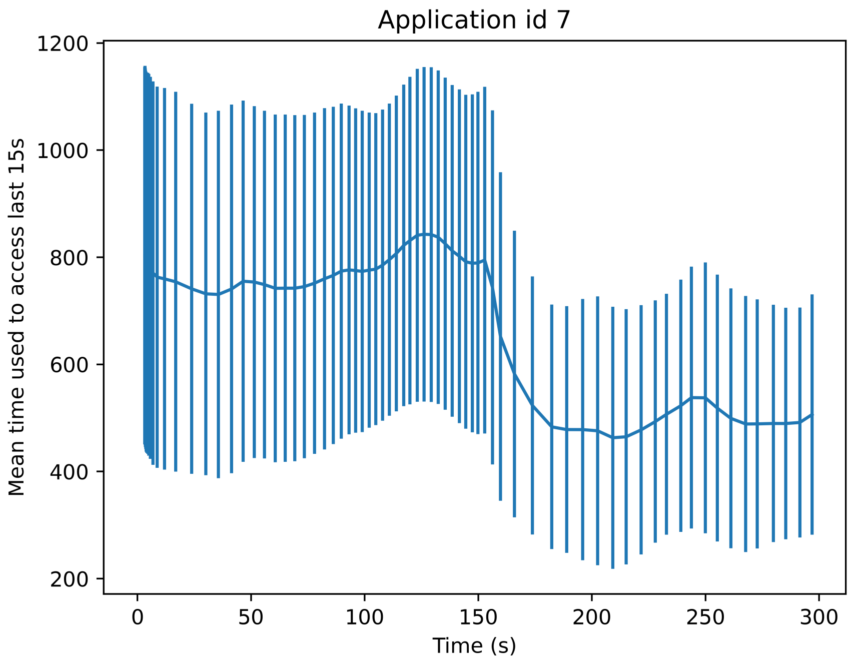


Figure B.6: Plot showing the average end-to-end latency for appliaction 7