DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Application Server Mobility and 5G Core Network

ALI SYMERI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Application Server Mobility and 5G Core Network

Ali Symeri

2019-06-18

Master’s Thesis

Examiner
Markus Hidell

Academic adviser
Peter Sjödin

Industrial advisers
András Zahemszky and Vinay Yadhav

KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science (EECS)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

Abstract

With advancements in the mobile network architecture, from the Fourth Generation to the Fifth Generation, a vast number of new use cases becomes available. Many use cases require cloud-based services, where a service is deployed close to the user. For a user to communicate with a service, it connects to the mobile network base station, Fifth Generation Core network and then to the service. When the user changes physical location, the mobile network and the service must apply mobility techniques. This is to prevent tromboned traffic and provide low latency between user and service. When a handover occurs, so that a user's attachment point to the mobile network is changed from the one base station to another and the User Plane Function changes, the cloud-based service may have to seamlessly move from one cloud to another as well. In this thesis, a Service mobility framework is proposed and implemented, which enables service live migration between edge clouds and it provides simple RESTful APIs. The evaluation of the framework shows that the proposed implementation adds low delays to the total migration time and the service downtime is also shown to be low in the case of video streaming with no service interruption.

Keywords
Fifth Generation Core, Handover, Network Function, Edge cloud, Service mobility, Live migration.

Sammanfattning

Med framsteg i det mobila nätverkets arkitektur, sett från den Fjärde Generationen till den Femte Generationen, så blir nya användningsområden tillgängliga. Bland de nya användningsområdena inkluderas molnbaserade tjänster, där tjänster är placerade nära användare, dessutom har vissa områden behov av dessa molnbaserade tjänster. För att en användare ska kunna kommunicera med en tjänst så måste den först ansluta till det mobila nätverkets basstationer och sedan till Femte Generationens kärnnätverk, för att sedan kunna kommunicera med tjänsten. När användaren förflyttar sig från en plats till en annan, så måste det mobila nätverket och tjänsten tillämpa rörlighetstekniker, som förflyttning av tjänsten. Förflyttningen är för att förhindra trombonerad trafik och att förse låg latens mellan användare och tjänst. När en överlämning sker, d.v.s att en användares kopplingspunkt till det mobila nätverket ändras, från en basstation till en annan, och att User Plane Function ändras, så kan även den molnbaserade tjänsten förflytta sig sömlöst från ett moln till ett annat. I denna avhandling presenteras ett tjänströrlighetsramverk som möjliggör tjänströrlighet mellan moln och erbjuder enkla RESTfulla API:er. Evaluering av ramverket visar att implementationen bidrar med låga fördröjningar till den totala migrations tiden samt att tjänster med videoströmming har lågt driftstopp utan tjänstavbrott.

Nyckelord
Femte Generationen Kärna, Överlämning, Nätverksfunktion, Moln, Tjänströrlighet, Live migration.

Table of Contents

List of Abbreviations

3G	Third Generation
3GPP	3rd Generation Partnership Project
4G	Fourth Generation
5G	Fifth Generation
5GC	5G Core
AF	Application Function
AMF	Access and Mobility Management Function
API	Application Program Interface
AUSF	Authentication Server Function
CAPEX	Capital Expenditure
CN	Core Network
CRIU	Checkpoint/Restore In Userspace
DL	Downlink
DMTCP	Distributed MultiThreaded CheckPointing
DN	Data Network
DNS	Domain Name System
HTTP	Hypertext Transfer Protocol
IAP	Internet Protocol Advertisement Point
ITU	International Telecommunication Union
IoT	Internet of Things
LBO	Local Break Out
LR	Location Registry
LTS	Long Term Support
LXC	Linux container

NAT64	Network Address Translation
NEF	Network Exposure Function
NF	Network Function
NFV	Network Function Virtualization
NG-RAN	Next Generation RAN
NRF	Network Repository Function
NSSF	Network Slice Selection Function
OF	OpenFlow
OPEX	Operation Expenditure
OS	Operating System
OVS	Open vSwitch
PCF	Policy Control Function
PDU	Protocol Data Unit
PLMN	Public Land Mobile Network
PaaS	Platform as a Service
QUIC	Quick UDP Internet Connections
RAN	Radio Access Network
RTMP	Real-Time Messaging Protocol
SBA	Service-Based Architecture
SBI	Service-Based Interfaces
SDN	Software Defined networking
SMF	Session Management Function
SMO	Service Mobility Orchestrator
TCP	Transmission Control Protocol

UDM	Unified Data Management
UL	Uplink
UPF	User Plane Function
URI	Uniform Resource Identifier
VM	Virtual Machine
VoLTE	Voice over LTE
cgroups	Control Groups
gNB	Next Generation NodeB
mMTC	Massive Machine-Type Communications
uMTC	Ultra-reliable Machine-Type Communications
veth	Virtual Ethernet Device
xMBB	Extreme Mobile Broadband

Acknowledgement

I would like to thank my two supervisors at Ericsson, András Zahemszky and Vinay Yadhav, for their continuous support and guidance during the period of this thesis. I would also like to thank my examiner Markus Hidell for his feedback and supervisor Peter Sjödin at KTH. Lastly, I would like to thank the Ericsson research team where I was working, for allowing me to join their environment and use the 5GC prototype and cloud resources to realize the thesis.

Stockholm, June 2019
Ali Symeri

1 Introduction

During recent years, a rapid expansion of the total number of mobile users has occurred and it is expected to continue increasing with the coming years. The estimated number of new mobile subscriptions in the third quarter of 2018 was 120 million, yielding a total of 7.9 billion subscribers [1]. However, a fast-paced increase in mobile users is not the reason for deploying a new mobile network today. Previously, the Third Generation (3G) mobile network applied standards to meet the demand of the Smartphone era, i.e. web browsing, reading email and sharing content, all on a mobile device. The Fourth Generation (4G), boosted the 3G by increasing the network speed, performance and added features such as Voice over LTE (VoLTE), meeting the demand of a major increase in mobile users. Finally, the time for a new era, where services, companies, etc. move to cloud-based hosting, has arrived and a variety of new use cases that could not be achieved before, have been introduced. New use cases lead to the demand of increased bandwidth and reduced latency. So, companies such as Ericsson work on designing and building a new mobile network, Fifth Generation (5G), to co-exist with the old, 4G, until replaced entirely [1] [2].

To successfully deploy 5G, a standard has been worked on and envisioned, most notably by the 3rd Generation Partnership Project (3GPP) [3] and International Telecommunication Union (ITU) [4]. The standard includes a variety of use cases; Extreme Mobile Broadband (xMBB), Massive Machine-Type Communications (mMTC) and Ultra-reliable Machine-Type Communications (uMTC).

1.1 Background

As 5G is expected to accommodate a diverse set of new use cases, i.e. xMBB, mMTC and uMTC, there will be an increase of users that are not only humans anymore, but machines, Internet of Things (IoT) devices and vehicles. Therefore, it is required to be flexible, scalable and reliable. Specifically, the uMTC use case, requires that remote operations and applications should be ultra-reliable and have extremely low latency. The latencies should be in the spectrum of 1 ms or less for the user plane and 20 ms or less for the control plane [4]. In essence, this means that the latency between an application and a user should provide uninterrupted service. This is where mobility in the 5G network can be defined and improved.

In the 5G core network, mobility concerns the process of changing the point of attachment in the network, to ensure that the end user does not experience any interruption in the service that is ongoing [5]. This conveniently infers seamless changes in the point of attachment in the Core Network (CN) while yielding little to no decline in the network quality and signal. The purpose of mobility in such networks, is to effectively provide users with high availability, faster networking speeds, lower latency and keeping track of users attachment points in the network.

In Figure 1.1, the general 5G network architecture, based on the 3GPP specifications for 5G system architecture [6] [7] [8], is presented. It shows that the network is based on services that were introduced by utilizing Software Defined networking (SDN) and Network Function Virtualization (NFV) for 5G, so that elements in the architecture are mostly defined as Network Functions (NF). This means that the CN has been split into a user- and control plane respectively, in regard to SDN. The NFV aspect implies the possibility to run functions on virtualized platforms. Additionally, the specifications state the key enabler for 5G as SDN, as well as the key concept being network slicing. However, for the purpose of this thesis, the components Session Management Function (SMF), Access and Mobility Management Function (AMF) and User Plane Function (UPF) in Figure 1.1, are in focus. That is where the mobility management solution is considered, and handover is signaled and processed.

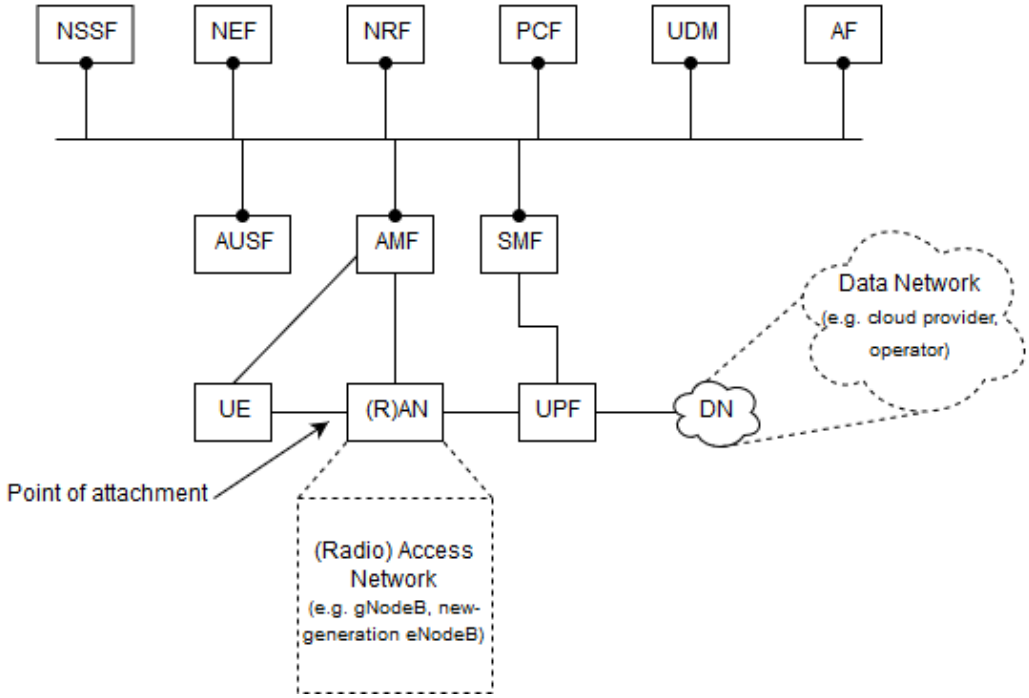

Figure 1.1: 5G system architecture.

The session management component, SMF may send notifications to subscribed Application Functions (AF) regarding certain events, i.e. user plane path change [7], when users change physical locations. The solution for this has to consider the aspects of propagating relevant notification data to the AFs so that they can efficiently act upon it. The notifications will assist the AF to determine if, for example, a service migration should occur. It will then be up to the AFs to decide whether it is a proper action to take or not. In addition, in the case of handover, the switch of UPF may occur in some cases. This has to consider the aspects of failure when switching UPF, since packets going to users are sensitive to being dropped and latencies have to be low. However, it is not always that UPF switch occurs. In essence, the 5G session management combined with mobility management, can handle the migration of user traffic by switching source and target UPFs. However, the problem lies at the service

side, as it does not currently migrate. When the user is connected to some service or application server in the network, i.e. streaming a video, the traffic will still be moved to another UPF and attachment point but the service or application server itself will not be migrated. This consequently leads to the traffic being directed all the way back to where the service is hosted even if the UPF and attachment point has changed.

Additionally, to the problem of application server migration, the drawback of not migrating the service yields tromboned network traffic, increased latency, non-optimal routing and performance and resource wasting. This means that operators deploying 5G and cloud providers, will have increased Capital Expenditure (CAPEX) and Operation Expenditure (OPEX).

To potentially solve the aforementioned problem, this thesis proposes a solution in which services or application servers are deployed in lightweight containers. The containers can be hosted at any cloud provider, Data Network (DN), that is accessible from the 5G Core (5GC) Network. This means that the containers can be placed at the edges of the network closer to the user. Moreover, when the user changes physical location, e.g. changes Radio Access Network (RAN) which is the attachment point, a handover signal is triggered in the 5GC. This may result in the change of UPF in the network, the container can then be live migrated from one edge cloud to another, if necessary, as seen in Figure 1.2. Both the UPF and container are not required to be switched or migrated, it is decided based on certain conditions in the 5GC. The figure shows that an edge cloud can have multiple containers, e.g. A and B. A user is watching a video stream that is hosted as a service in a container, A, in one edge cloud and container B is not used by any user. As container A is being used and the user changes physical location, requiring a handover, it is live migrated to the target edge cloud. This means that the Transmission Control Protocol (TCP) connections and sockets that are attached to the video stream session and user will also live migrate to the destination. Container B remains at the source edge cloud.

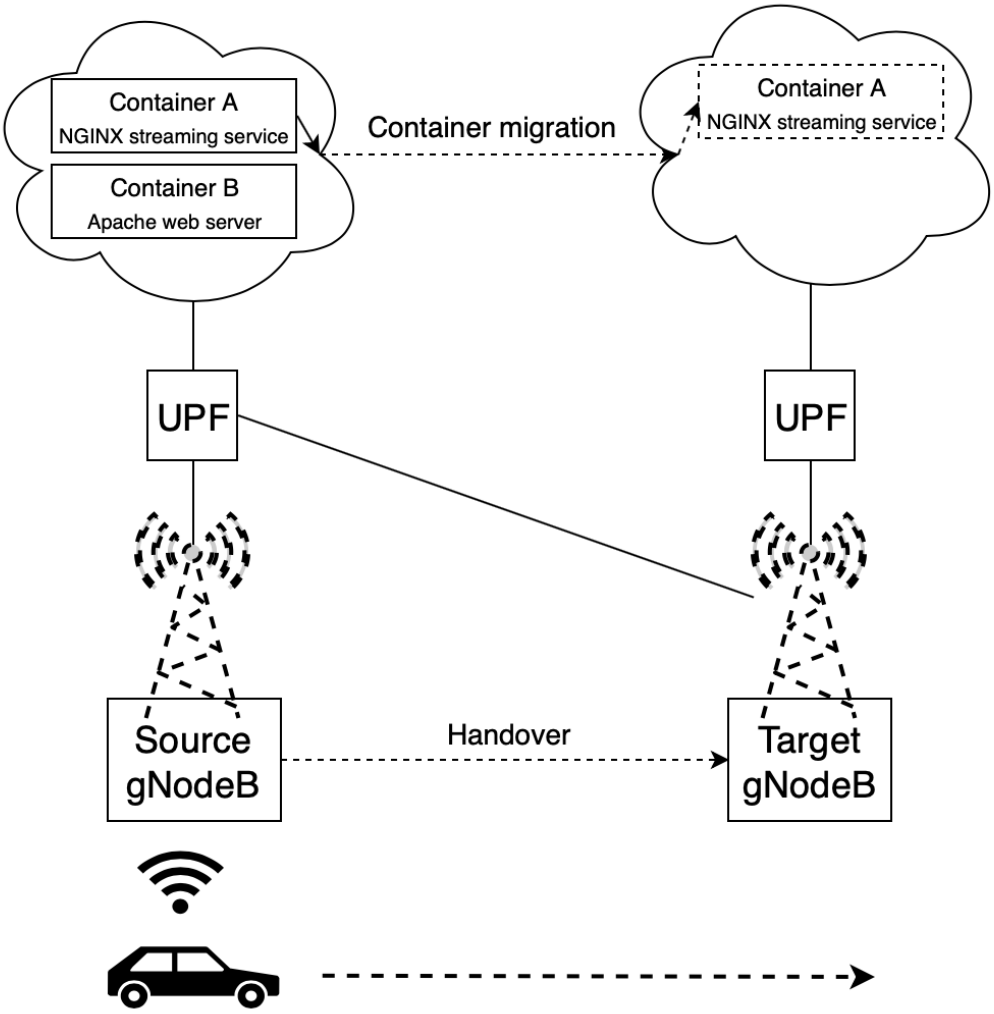

Figure 1.2: A simple example of a handover followed by a container migration.

The solution will yield higher performance, lower latency and optimal-routing for the 5GC and the users.

1.2 Problem statement

The problem to solve in this thesis is service/application server live migration in 5G. The idea behind live migration is that currently, in the 5GC network, UE handover is an existing feature but live migration of services in the edge cloud that the UE is using, is not. When the existing feature of handover is triggered, the solution to be implemented should be notified of it and perform live migration of services between edge clouds. The importance of performing live migration of services is to provide the UE with low latency, high availability and uninterrupted service, which means that the user experience is seamless and transparent to underlying changes. Consequently, it also means that the approach for the solution will be integrated in the 5GC network as a part of a NF, which means that there is a combination of both 5GC network development and edge cloud development. The first research question investigated in this thesis is the following:

- How can we design and implement live service migration as part of the edge cloud, triggered from the 5GC network?

The challenge here is to find a design that meets the requirements of optimization, good performance and suitable relocation method to be chosen. This leads to the second and third questions of this thesis:

- What is the most suitable and standard relocation method?
- How can the service downtime be as minimal as possible when performing live migration?

To realize the second and third questions, the main challenge is to know when and where to utilize the 5GC to listen for handover triggers. Additionally, there is a problem to be solved at the moment of handover when the UE does not have a traffic path to the service anymore, due to the change of NFs and mobile base station. This leads to answering the fourth question of this thesis:

- How can the UE be served by the service at the moment of handover and during live migration?

To the knowledge of the author, there is no existing solution for performing live migration of services in the edge clouds in combination with the 5GC network handover events.

1.3 Purpose

The purpose of this thesis is to investigate and analyze the absent parts of service mobility in the 5GC and edge cloud. This in turn leads to providing and implementing a solution for improving the current service mobility feature. In addition to providing a solution for a certain problem, the purpose is also to apply new ideas to the 5GC and edge cloud. In this case, utilizing the existing network functions in the 5GC to send the standardized notifications to edge clouds and then use lightweight containers to migrate services between edge clouds. This means that the main focus of the thesis is to perform migration of services between edge clouds to achieve optimal routing, there are a few benefits introduced with this. The benefits are:

- The network traffic from user to service will be using an optimized path.
- The latency between user and service will be kept low.
- The service will always be close to the user.

The three benefits also apply even after mobility events occur.

1.4 Goal

The goals of this thesis are detailed as follows:

- Research different service migration techniques and tools that can perform live migration while preserving TCP sockets.
- Investigate how the 5G Core Network can be utilized to assist in service migration.
- Investigate the required network traffic flow and how it can be integrated in the most suitable way for service migration.
- Integrate a service migration framework, assisted by the 5GC, to perform live migration between edge clouds.
- Implement and integrate a prototype of the service migration framework into a simplified 5GC prototype.
- Evaluate the framework by measuring the total migration time and downtime of the migrated service.

1.5 Ethics and sustainable development

The ethical aspect is that the solution does not waste resources in terms of the traffic path taken. However, it may disrupt the intended use of the service that is migrated, this is because with any type of service migration, there is a downtime of that service which affects the user experience. Additionally, the network resources such as, routers, switches and NF are not disrupted permanently with the solution, but they will be disrupted temporarily during a short period of time. Disruption in the sense of adding to the load of those resources during service migration. When disruption is added to a service, impacting the user experience, the following question must be carefully considered: What is the consequence for the user at the moment of service interruption due to migration?

Finally, the economical sustainability aspect is that the solution will result in lowered CAPEX and OPEX for operators, as it uses e.g. SDN [9] to manage the service mobility. The ecological sustainability is considered with the use of lightweight containers which introduces reduced complexity, less overhead, reduced resource usage and higher scalability, which inherently means less energy consumption. The solution also provides an optimized network traffic flow which also reduces overhead and energy consumption as fewer network hops can be taken. The social sustainability in the work is that it adds positive outcome to people working with networking. This is because it provides less involvement from people regarding hosted services in the cloud that have moved from one edge cloud to another, as it is automated and already set up. However, the negative aspect may be that there will be less work for engineers as the automation increases, replacing the engineer.

1.6 Methodology

The work in this thesis is based on design, implementation and experimental evaluation. The core part of this thesis is to develop and integrate a service mobility framework for 5GC and edge cloud. The development process will be performed in three phases:

- **Design:** This is the first phase and defines the blueprint of the entire service mobility orchestrator. The blueprint ensures that the designed service mobility framework can; support future services and features, be simple and be effective, while having good performance. This phase also includes designing and characterizing different modules that build up the entire orchestrator, while comparing and determining which design approaches to take and which requirements to set and achieve. The entire framework consists of 4 modules: event signaling module, handover negotiator module, service module and routing module. There is also an Application Program Interface (API) towards the handover negotiator module for different service module implementations to utilize. The event signaling module is part of both the 5GC and edge cloud while the other modules are all part of the edge cloud.

  The event signaling module is where the 5GC will be involved. An event will be triggered, e.g. due to a user plane path change, and propagated from the SMF via this module to the edge clouds involved or subscribed to the event. The triggered event will initiate the service migration process and provide relevant information for it to proceed.

  The handover negotiator module will receive events and act upon them. This is the core part of the orchestrator, since it will take decisions of migration, communicate with peer edge clouds, communicate with service modules and determine which routing rules to set. It also provides an API for service modules to register to.

  The service module hosts the actual service that is required. In addition to hosting the service, it registers itself to the handover negotiator module to expose its services. The module also provides a migration technique of its services to other edge clouds when requested for it.

  The routing module is here to ensure that traffic can flow to the different services if they are either hosted locally or have migrated to another edge cloud. It uses OpenFlow (OF) based rules to provide routing.

- **Implementation:** In this phase, the aforementioned modules are implemented using a microservice approach. The implemented microservice are all RESTful implementations with object-oriented programming. To provide a solution for service migration, multiple open source tools are researched, compared and analyzed to perform the action required while ensuring reliability, scalability and stability of the

tools. The tools that fulfil the mentioned requirements sufficiently, are chosen.

- **Evaluation:** This phase includes specifying the Key Performance Indicators to be used for measuring the service mobility framework. The measurements are done in both an emulated environment and a 5G prototype environment, this is to achieve measure the framework on an environment similar to a real deployed architecture.

1.7 Outline

The work in this thesis is outlined as follows:

- Chapter 2 presents the background and literature study of the area of service mobility with an overview of Linux containers, 5G cellular networks and edge clouds.

- Chapter 3 contains the design and requirements of the service mobility orchestrator to be implemented. It includes an architecture of the framework and each component involved in the call flow to perform live migration.

- Chapter 4 presents and explains the implementation of the service mobility orchestrator. It follows the design principles and requirements based on chapter 3.

- Chapter 5 performs an evaluation and analysis of two different test setups. One test setup with the 5GC prototype in Openstack which is a non-optimal environment and another setup without the 5GC in an optimal setup for performance.

- Chapter 6 concludes the work by mentioning the conclusions that can be drawn from the work. It also provides suggestions for future work.

2 Service Mobility

This chapter introduces different approaches to service migration whether it may be hot or cold migration. Additionally, an introduction to 5G and edge cloud is presented. The chapter concludes with related work that may or may not be built upon in this thesis.

2.1 Service migration

When it comes to service migration, there are multiple ways to achieve live service migration. The relevant approaches that are discovered are Linux container migration, creating a custom library or building upon a proprietary protocol.

2.1.1 Linux Containers

An efficient way to live migrate services, without requiring any modification to the underlying system, is to use Linux containers (LXC). The containers are a way to isolate processes or services from the underlying Operating System (OS), kernel and application layer. They are often compared to Virtual Machines (VM) but differ significantly. In Figure 2.1, a VM runs simultaneously on the same host machine, sharing the hardware resources using a hypervisor as a way to provide emulated hardware to the VMs, making it heavyweight and adding overhead. LXC, however, share the existing host OS kernel and isolates the processes running in the containers from the system itself, making it lightweight and portable [10] [11], without the need for the hypervisor layer, reducing overhead.

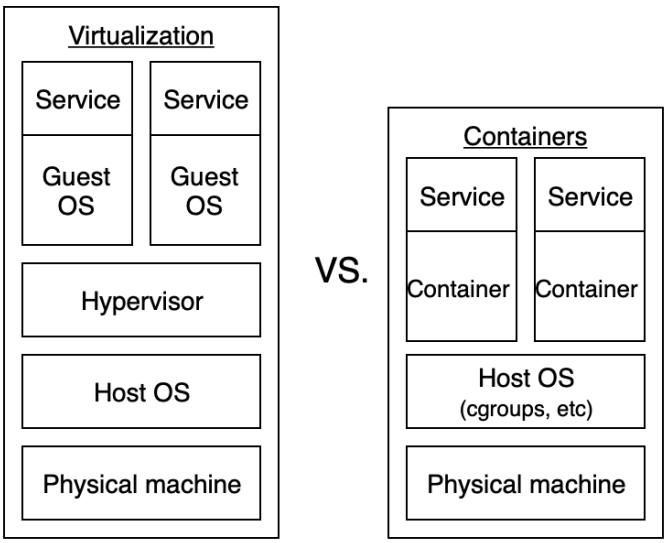

Figure 2.1: A comparison of virtualization and containers.

Linux Containers are possible due to the Control Groups (cgroups) and namespaces features in the Linux kernel, since they provide ways to isolate, prioritize and limit resources in the system, such as memory, disk and CPU [10] [11] [12]. In addition, the containers are meant to be ephemeral, meaning

that they can run for a certain amount of time and then be migrated, stopped or deleted.

To utilize LXC for migration, it exposes an API that can be used to manage the containers, i.e. create-, delete- and move containers. To efficiently make use of the LXC API, the same developers created LXD, which is a lightweight container hypervisor [14], without actually being a hypervisor or adding any additional layer. It is built on top of LXC to provide a RESTful API that utilizes the LXC API and at the same time making it remotely accessible, scalable and user friendly [11] [13].

As LXC is packaged with the Long Term Support (LTS) versions of Ubuntu [14], no further installation other than the OS of Ubuntu is required. This means that LXC can be used immediately after an Ubuntu LTS installation. LXC is also commonly used and extended by Docker [15] [16] and OpenVZ [11] [17]. So, for the reasons of LXC being packaged into some Linux distributions by default, lightweight, portable, offering low migration and downtime [11] and has less overhead than VMs, LXC and LXD are chosen as the service migration tools for performing live migration.

2.1.2 Live migration

To perform migration in the 5GC, where the uMTC use case is to be addressed, a certain type of migration has to be considered. There are different types of migration, e.g. cold- and hot migration, however, cold migration does not fulfil the requirement of uMTC.

In cold migration, the VM or container has to be shut off completely or suspended to then be moved from the source host machine to the destination host machine. This includes copying the memory and pages of the virtual environment to the destination and then booting or resuming it there [18].

Hot migration, also called live migration, is when the VM or container is still running and moves from the source host to the destination host, e.g. detached from the host OS kernel and attached to the new destination OS kernel. This means that during the live migration, the virtual environment will repeatedly be pre-copied to memory or disk and then transferred to the destination machine when there is a small difference in the state of the copied data [18]. On the destination machine, the virtual environment is loaded from the received data, and resumed there. Lastly, the source host machine shuts off and deletes the VM or container, as it is now on the destination machine and running. Using the live migration technique yields lower downtime compared to cold migration which, together with the uMTC requirements, is why it is chosen as the migration technique used in this thesis.

To perform live migration together with LXC and LXD, there is a tool called Checkpoint/Restore In Userspace (CRIU) which is integrated and used in LXC/LXD to handle the live migration process. The promising feature of CRIU is that it ensures freezing of running applications and checkpointing

them to files on disk or in-memory. The files are then used for restoring the applications, which are resumed from where they left off, e.g. the moment of freeze [19]. Furthermore, CRIU also handles established TCP connections and performs Checkpoint/Restore on the TCP sockets as well, so that they can be live migrated. Essentially, CRIU ensures that the running state of the applications are preserved, and the open network connections are maintained [20].

The LXD and CRIU live migration process performs pre-copy migration, which means that the memory state is copied from the source host to the destination host while, during the copying, the source host is still responsive and maintains all running applications. The pre-copy is due to the memory pages being updated on the source host, even after being copied to the destination host, so the pages are monitored for updates [11] [20]. To complete the pre-copy, one or more pre-dumps are performed by CRIU. This means that memory pages are extracted based on the running processes and they are extracted using parasite code that is injected to the process task. The parasite code ensures that CRIU can execute service routines inside the process tasks address space to extract the information it needs to dump. The parasite code remains there until everything has finished dumping and the process is ready to be restored at the destination host. As the processes are being restored, the parasite code is dropped, restoring the original state of the processes and CRIU detaches itself from them. As seen in Figure 2.2, the pre-copy and pre-dumps are done in the iterative dump phase, and when the dumping is finished e.g. the final dump, the container can be restored at the destination and deleted from the source.

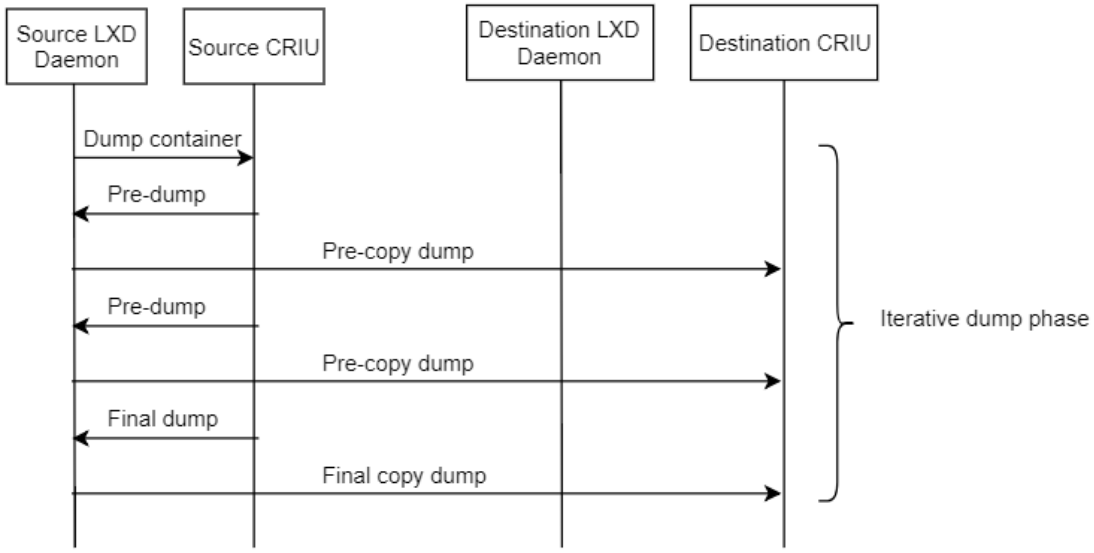

Figure 2.2: The iterative dumping process flow that CRIU has implemented.

Additionally, the IP address of the container on the source host has to be available on the destination host for the TCP migration to work. The way CRIU handles the aforementioned TCP connections and sockets is that it switches the sockets to a special mode, resulting in the socket being put into a

state that represents the end of a successfully finished operation, i.e. if a connect() request appears for that socket it will return “ESTABLISHED”. To have a complete restore of the TCP socket, the TCP Timestamps have to be addressed by CRIU, as the socket has moved to a different host which has a different timeclock. It is done so with a “TCP_TIMESTAMP” option in CRIU, that compensates for the difference in timestamps in the socket when moved to a different host. When the socket is restored at the destination host, CRIU lets TCP handle the restoration of the data sequence. Lastly, when the socket is in the phase between dump and restore, CRIU locks the connection so that no packets can enter the TCP stack in order to negate a transmission of “RST” by the kernel. This is done using netfilter, a packet filter framework, to drop all packets destined for that socket [21].

There is a limited number of alternatives to CRIU and one worth mentioning is Distributed MultiThreaded CheckPointing (DMTCP). It allows for computations to be checkpointed in user space and restarted but it does not mention preservation of TCP sockets, which indicates that DMTCP will let processes that uses TCP, establish new connections instead of migrating the current socket [22] [23]. Furthermore, DMTCP requires applications to be launched with DMTCP library which is then used to intercept certain calls from the application to be able to ensure checkpointing and restoring. As such, DMTCP does not support every application while CRIU does. CRIU also does not require any library to be used and it works with any arbitrary application. For these reasons, in addition to being built into LXC and LXD, CRIU is used for migrating the application and TCP sockets.

2.1.3 Custom library

As there are multiple solutions to solve the live migration problem, the custom library approach is taken into consideration. The general idea of this approach is to implement a set of APIs and enforce application or service developers to use that API. This means that developers will have to bundle their application with the proposed API which will in turn handle the live migration transparently. Consequently, it means that an additional layer is added, it may take time until a solution like this is widely adopted. However, it will result in having full control over the migration process. The approach is conceptualized in Figure 2.3. The figure shows that the application calls the source custom API for migration, and it in turn calls the, already deployed API on the destination host, that a new migration is incoming, for preparation. The application, e.g. context, session information, socket, data, etc. is sent to the destination host and the destination custom API handles the incoming transfer, until finally resuming the application there.

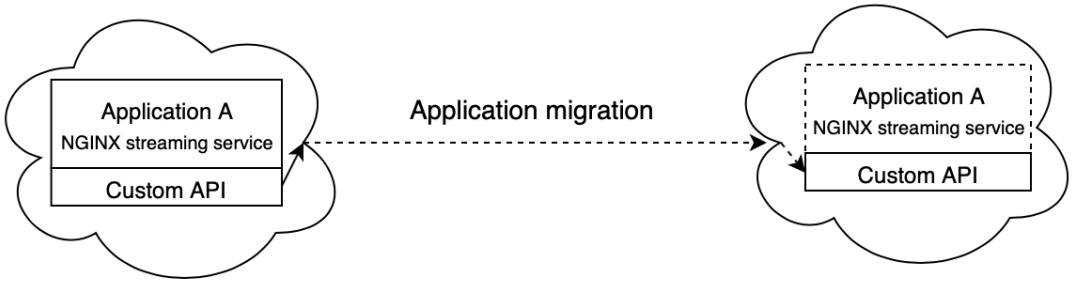

Figure 2.3: An example of application migration using the custom library approach.

To perform live migration, using this approach without LXC and LXD, every part of the migration framework has to be implemented from scratch. If this approach is used with LXC and LXD, then the live migration part is already in place requiring fewer parts of the migration framework to be implemented. As shown in LXC and LXD, this concept is essentially already implemented and pre-packaged in some Linux distributions, which is why the custom library approach is not used by this thesis.

2.1.4 Quick UDP Internet Connections

Another approach is to utilize Quick UDP Internet Connections (QUIC) with modifications to the protocol. QUIC is a transport protocol, running on top of UDP, that uses encryption and provides low latency and multiplexing, improving the performance of HTTPS. In addition, it runs in user-space, meaning that it can run almost anywhere [24].

The way QUIC can be used for live migration is based on its design and implementation which use Connection IDs. These IDs are meant for identifying the connections that are established, instead of using IP address and port number [24]. This means that users can migrate from source QUIC end-point to destination QUIC end-point, and also change IP address, without losing connectivity. The user will remain connected to the same Connection ID even if the users IP address changed. However, QUIC only supports client-side connection migration and would require modifications and implementation in the protocol to support server-side connection migration as well. Due to this reason, QUIC is not used in this thesis.

2.2 5G Cellular Networks

The 5G System, as seen in Figure 1.1, it consists of UEs, (R)AN and DN. The UEs are subscribers to Public Land Mobile Networks (PLMN) and CN and will have (R)ANs as their point of attachment. When a UE is powered on, an Initial Registration procedure is invoked, which involves a series of actions or controls such as network selection, authentication, authorization, policy control and lawful interception. After the Initial Registration, the UE may update its registration to the network, either periodically or upon mobility [6]. This is to remain reachable and updating its capabilities. As the registration is finalized, the UE will be served by an AMF.

(R)AN is the cellular technology that has the base station and antennas to provide cellular coverage ranging over a region or area. UEs connect to these access networks wirelessly and are served by the CN. The Next Generation RAN (NG-RAN) consist of Next Generation NodeBs (gNodeB or gNB) that are connected to the 5GC through the NG interface. The gNBs can be interconnected via the Xn interface [25], mentioned in subchapter 2.2.4.

The DN is the part where different operator-hosted clouds, operator services or the public internet can be deployed. For example, edge clouds provide hosting of applications and services to the edges of the network [26] so that use cases requiring low latency can be better supported in 5G.

The functionalities of the different NFs are mentioned below.

2.2.1 Control plane

In the 5GC network there is a clear separation between functions which leads to the control- and user planes. As Figure 1.1 show the two planes together, Figure 2.4 introduces the control plane.

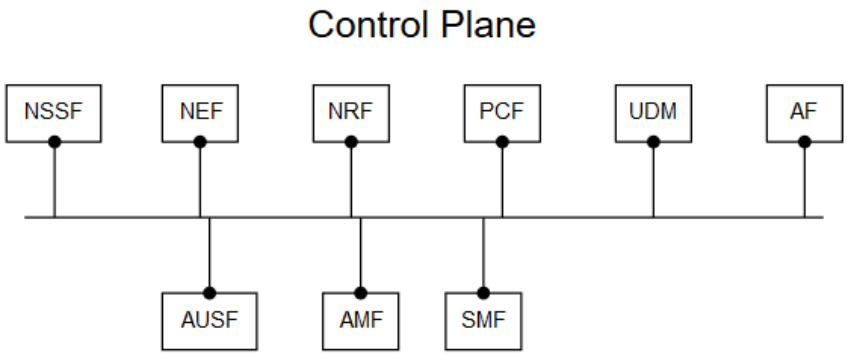

Figure 2.4: 5G control plane with the NFs and SBIs.

In the control plane there are essentially nine network functions, described below [6]:

- **Network Slice Selection Function (NSSF):** The NSSF supports the selection of Network Slice instances that should serve a UE.

- **Network Exposure Function (NEF):** The NEF is meant to provide exposure of capabilities and events. This means that for a Network Function (NF) that wants to, i.e. listen to events or expose events to other NFs, has to go via the NEF. This ensures secure exposure to third parties and Application Functions.

- **Network Repository Function (NRF):** The NRF provides service discovery meaning that it will receive NF Discovery Requests from NFs and propagate the discovered information to the requestee. With this, it also stores the NF profiles of the available NFs along with what services they support.

- **Policy Control Function (PCF):** A simple function which mainly has three functionalities:
    - Control network behavior.
    - Provides and enforces policy rules to Control Plane functions.
    - Retrieves information regarding subscriptions for policy decision.

- **Unified Data Management (UDM):** The UDM provides functionality that is related to subscriptions to the 5G network. This can be, e.g. SMS management, access authorization and subscription management, among many.

- **Application Function* (AF):** This function is a means to interact with the CN so that it can provide certain services. As such, the AF can impact the traffic routing, interact with NEF and interact with the PCF. If the AF is trusted by the operator, then it can directly interact with NFs without going via the NEF otherwise the NEF will be passed every time.

- **Authentication Server Function (AUSF):** The task of AUSF is to authenticate for access regarding 3GPP and untrusted non-3GPP.

- **Access and Mobility Management Function* (AMF):** The AMF has multiple functionalities such as, registration-, connection-, reachability and mobility management, among many. It also provides User Equipment (UE) mobility event notifications.

- **Session Management Function* (SMF):** For the SMF, session related functionalities are supported. These include, but not limited to, management and establishment of sessions, allocating IP addresses to UEs, data charging and configuration of traffic steering at User Plane Function (UPF). It can also send notifications to AFs, that are subscribed to user plane management events regarding, i.e. a user plane path change, which can then be used to trigger live migration of services between DNs.

The functionalities of the SMF, AMF and UPF do not need to be supported by a single SMF, AMF or UPF instance, they can be distributed to multiple instances. The functions marked with asterisk are functions that are most relevant for the purpose of this thesis.

2.2.2 User plane

As for the user plane, there is one main function there as seen in Figure 1.1, called UPF. The UPF is an important function which does, most notably, routing and forwarding, packet inspection and Quality of Service handling [6]. It also acts as the external PDU session point of interconnect to the DN which

means that it provides connectivity to and from, for example edge clouds if that is deployed there, so that services deployed in those edge clouds can be notified by the 5GC to initiate live migration of services.

2.2.3 Service-based architecture

As per the 5G standard, the functions in the 5GC control plane communicate using Service-Based Interfaces (SBI) per individual NF and a logical communication bus [6], as seen in Figure 2.4. That is the Service-Based Architecture (SBA), which ensures that e.g. the SMF can allow for other NFs to access its services. The interface between the functions can be of two models, request-response or subscribe-notify communication model, in which both use a common protocol, Hypertext Transfer Protocol (HTTP) and HTTP version 2 (HTTP/2), in an Application Programming Interface (API) based manner [27] [28]. The SBIs usage of HTTP/2 is based on IETF RFC 7540 [29] which provides better efficiency and reduced latency compared to HTTP. It also keeps the semantics of HTTP, so the messaging syntax remains the same.

For the two aforementioned communication models, the NF services support two operations, producer or consumer. When the NF service is a consumer, it will behave as a HTTP client and another NF service will act as a producer that is a HTTP server. In the case of request-response, the consumer will send a request to the server and receive a response. For the subscribe-notify model, the consumer will subscribe to the server for certain events and receive notifications from the server whenever an event has been triggered. For the request-response model, the consumer will send a request to the server and receive a response back, based on the type of request [27] [28]. As the NF service can be a consumer, it can also be the producer, which will then behave identically to the case of it being a consumer. The difference of being a consumer or producer is that the consumer will ask for information or resources, while the producer will provide or create resources [28] where resources can be e.g. events regarding handover.

In addition to there being two communication models, the syntax for messaging between the NFs are also specified. There are strict mappings of Uniform Resource Identifier (URI) structures to functions and request methods. The URI must follow the following structure: *“{apiRoot}/{apiName}/{apiVersion}/{apiSpecificResourceUriPart}“,* where the apiRoot contains the scheme, e.g. “http://” or “https://”, apiName defines the name of the API, apiVersion states the version of the API and the apiSpecificResourceUriPart specifies the resource to be called in the API specific to the NF itself. The first three mappings together define the base URI of the API, and the last mapping varies depending on the NF and actions. Furthermore, as the URI is standardized, the request and response messages are as well. In the case of subscribing to a NF service, the format is shown in Figure 2.5. The consumer sends a HTTP POST request to the NF service producer with the NF specific URI and if the subscription is valid, the producer replies with a HTTP response containing a HTTP status code of 201 Created. The data that is sent regarding the subscription is in the body of the

HTTP request in JSON format and processed by the producer [28] [30]. The contents of the request body are available in the standard for 5G in 3GPP [30] as it will not be presented here.

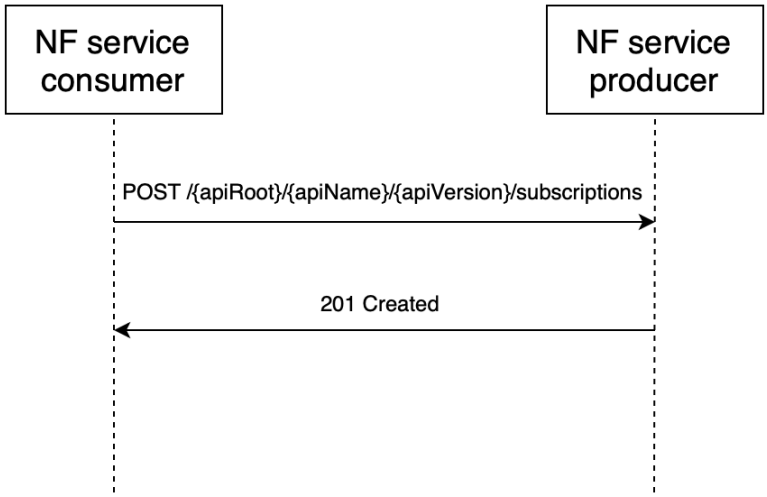

Figure 2.5: The standardized usage of HTTP request/response for NF consumers and producers. This is the case of subscription.

2.2.4 Signaling

Once the registration procedure is finalized, as mentioned in section 2.2, the UE is served by an appropriate AMF. The AMF will then provide UE related notifications concerning mobility and report them to NF consumers such as SMF, PCF or NEF. Depending on these NFs, if they want to receive the notifications, they can subscribe to the UE mobility event notification service at the AMF. As there are different interfaces between NFs, there is one reference point between the UE and AMF which handles the signaling of Registration- and Connection management. Once a SMF has been chosen and subscribed to the AMF, it is the AMFs responsibility to ensure that the SMF receives all related signals concerning that session, and that communication is through an interface between the AMF and SMF.

In case of a registered user wants to send and receive data to and from a DN, a Protocol Data Unit (PDU) session establishment procedure occurs. The UE sends a PDU Session Establishment Request to the AMF which in turn selects the SMF and then sends a request to the selected SMF and receives verification. The SMF then selects a UPF using a similar messaging model as for selecting the SMF but using the corresponding interface between SMF and UPF, see the 3GPP standard [7] for more details.

As previously mentioned, there is a Xn interface between gNBs. This interface is used to exchange signals between two gNBs while being interoperable with different manufacturers. The interface allows for many procedures over the control and user plane, such as mobility management e.g. handover procedures [31]. To support control plane signal messaging, there is an application layer signaling protocol called Xn Application Protocol (XnAP)

[32]. XnAP uses signals associated with UEs to perform basic mobility procedures such as handover. In the case where a UE changes physical location, the source gNB sends a handover request to the target gNB using XnAP so that the point of attachment of the UE can change.

There is the case of a UPF changing due to a handover in the network i.e. the UE has changed physical location. In this case, when the (R)AN and the UPF changes, there is a series of messages. The source (R)AN will detect that a handover is required and connect to the target (R)AN so that it can request a path switch. The full detailed flow is available in the 3GPP standard [7].

The two cases are mentioned here to point out that the main NFs that are relevant for this thesis are SMF, AMF and UPF as they are either relocated or trigger events which can be used to signal the edge clouds to perform live migration of services between clouds.

2.3 Edge cloud

The edge cloud is where most services or application servers are hosted, and they can be deployed by operators, cloud providers or third parties. The DN, where edge clouds can be deployed, are connected to the 5GC network via the UPF where there is a user plane interface between them [6], as seen in Figure 1.1. Edge clouds are efficient for computation and hosting at the edges of the (R)AN because they enable efficient use of the 5GC to provide reduced latency for UEs. Additionally, the presence of edge clouds leads to container and VM deployment, high availability of services, enables service mobility and ensures that the vast use cases in 5G can be supported [33].

The network edge cloud architecture consists of a Host level and System level. The Host level has three components and the System level has one, described below [33].

Host level:

- **Mobile Edge Host:** Promotes mobile edge applications and provides the virtualization infrastructure to offer cloud resources such as computation, storage and networking. Additionally, it provides the platform that is used to enable execution of mobile applications in the edge. In this case, the mobile applications are virtual instances executed on top of the virtualization infrastructure.

- **Mobile Edge Platform Manager:** Provides the lifecycle management of virtual instances such as instantiation and termination. It also communicates with the system level edge orchestrator.

- **Virtualization Infrastructure Manager:** Manages the virtualized cloud resources for the mobile applications including software images for fast instantiation of applications.

System level:

- **Mobile Edge Orchestrator:** This is the component responsible for managing the mobile applications and authenticate services in addition to provide relocation of applications if deemed necessary. This is possible because the orchestrator knows of the resources and capabilities of the entire mobile edge network and available applications.

The deployment of edge clouds in 5G are there to interact with the NFs in the 5GC network via the SBI. As the edge cloud is part of the DN, it can communicate with the UPF, however, in certain cases the UPF can be integrated in to the edge cloud system to provide control of the data plane. Additionally, the Mobile Edge Orchestrator acts as an AF which can communicate with NFs via the NEF or directly with the target NF depending on the trust level of the edge cloud [26]. This means that for mobility purposes, the edge cloud AF can subscribe to events of the SMF and AMF and be notified of handovers to perform service mobility of applications between edge cloud.

2.4 Related work

There is continuously work and research being done in the area of 5GC and less so in live migration. In this chapter, different related areas of interest to our subject will be presented, analyzed and discarded or potentially utilized.

2.4.1 5GC

In recent work, there was a proposed novel architecture for 5G that was based on mobile service chaining [34]. The authors presented the architecture with dividing it into a control- and user plane, similar to 3GPP, and also decomposing the user plane into smaller functions. However, the proposed architecture introduces new functions in the user- and control plane. The added control plane function, called Location Registry (LR), consists of a table or entries that maps UE IP address to UE location. That location is not a physical location but the id of the base station. The added user plane function, Internet Protocol Advertisement Point (IAP), acts similarly as a router. It exposes ranges of IP addresses to networks so that packets can enter the network via the IAP. The LR and IAP are henceforth called Optimal Routing. The Optimal Routing architecture is a mobility management scheme used for maintaining optimal routing when mobility occurs e.g. low latency for UE to server and UE to UE traffic and for preserving the IP addresses at mobility. Essentially, with the architecture, the Downlink (DL) traffic flow will be; Server → closest IAP → UPF → gNodeB → UE. And the Uplink (UL) traffic flow will be; UE → gNodeB → UPF → Server. The two flows are illustrated in Figure 2.6. Here, site B and its functions or edge cloud are not involved.

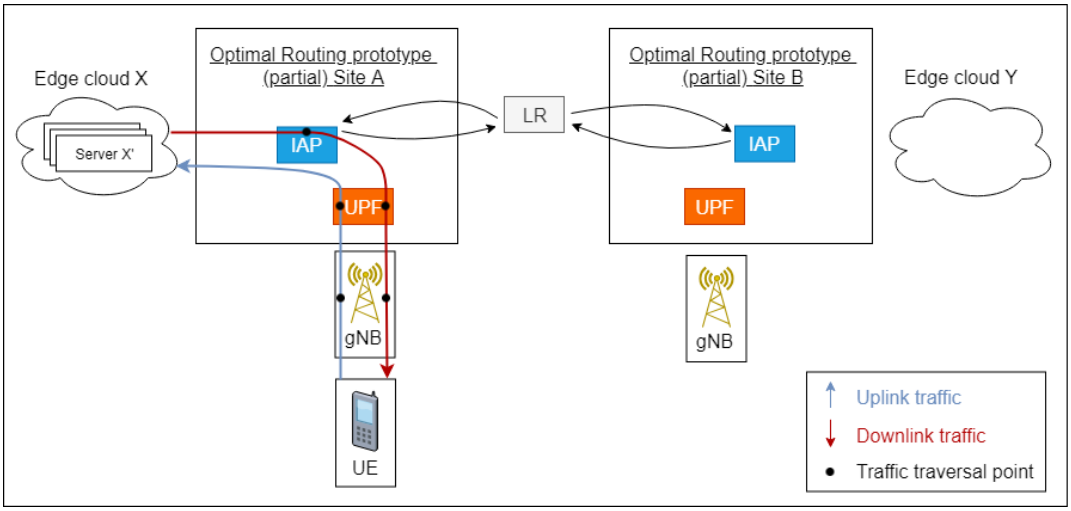

Figure 2.6: Optimal Routing before mobility occurs.

In the prototype, when mobility occurs e.g. SMF updates the LR and the LR updates the subscribed IAPs, and the UE changes physical location, the UPF will be relocated and UE IP unchanged. This is presented in Figure 2.7 and then the closest gNB and UPF will be in another site, site B.

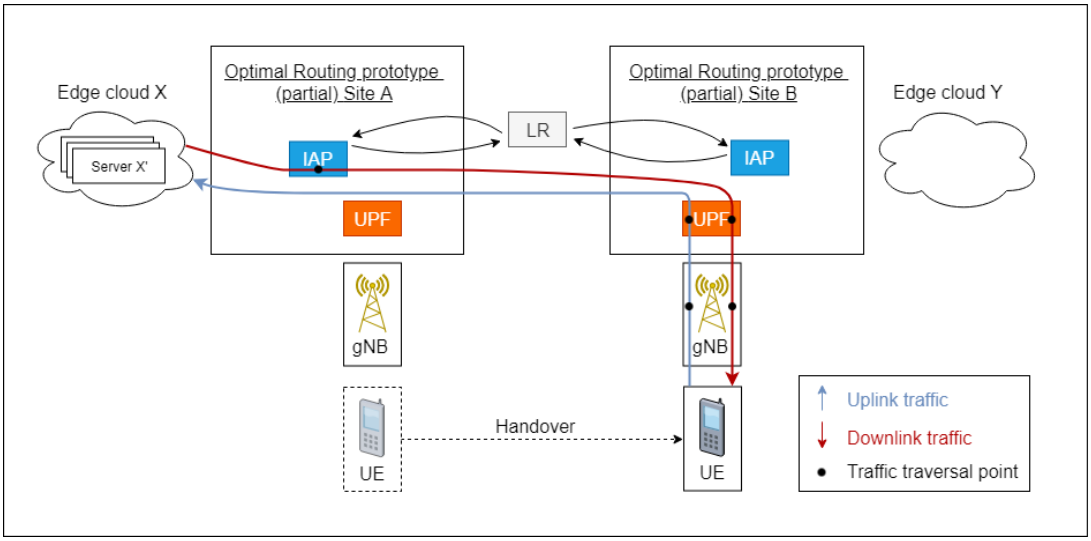

Figure 2.7: Optimal Routing after mobility occurs.

Finally, as this architecture is prototyped at Ericsson Research and accessible for this thesis, it will be utilized to integrate a server relocation solution and perform measurements on.

2.4.2 Service mobility and migration

In the area of service mobility and migration, there is some related work and some research has been done in the area. Firstly, there is a proposed solution for service mobility, targeting 4G. The work solves tromboned traffic when UEs change physical location by deploying services closer to the user [35]. This was done by implementing a Platform as a Service framework that migrates UE context from source cloud to target cloud. The context migration

includes moving TCP sessions as well, when handover signals are generated. The work utilized the IEEE 802.21 Media Independent Handover [36] and RFC 4067 Context Transfer Protocol [37] standards to perform the signaling and context migration. To move the TCP sessions, SockMi [38] was used. The work yielded a migration time of 12 ms, which is extremely fast but the drawbacks of using the aforementioned tools and standards are too great. The drawbacks of the work were that the Linux kernel had to be modified and compiled for every new kernel version, which is tedious, the RFC 4067 standard was experimental, and IEEE 802.21 was not widely adopted. For these reasons, this work will not be built upon and none of those tools will be used for this thesis.

Furthermore, research in service mobility for 5G is flourishing. Research in fast service migration for 5G has been worked on and are presented in a few papers. In the paper [39], the authors present a solution for doing container migration of services, targeting the requirements of 5G regarding low latency. The paper has the scope of migrating containers from one edge cloud to another. The containers are Linux containers and utilizes CRIU to do stateful migration of the containers to ensure service continuity. To perform the migration, two techniques were used; Temporary File System based Lightweight Container Migration and Disk-less based lightweight Container Migration. The paper shows that the service downtime time was around 1 second using the Disk-less approach. The drawbacks of the paper were that it does not specify whether a TCP connection was live migrated or if they are shut down and re-established. It also migrates unrealistically large containers of 350 MB and 590 MB, whereas realistic containers are of sizes around 30 MB, hence being lightweight. Lastly, it is not strictly an edge cloud solution targeting 5G but there is no involvement of 5G in the solution or the work.

Another approach is the Follow Me Edge-cloud concept [40] [41], which leverages edge clouds in 5G to always connect to the nearest one, ensuring low-latency communication to services in the cloud. This concept also acts when a UE changes location and performs service migration from one cloud to another. The concept describes that there is a decision making of whether a service should be migrated, which parts of the service should be migrated and where it should be migrated to. Unfortunately, this was a concept and a general approach, however, learnings will be taken from this as the approach has similarities to the one taken in this thesis. An implementation of the Follow Me Edge-cloud concept was the OpenFlow implementation [42], which migrates VMs between edge clouds. The implementation is heavily based on OpenFlow and does not migrate lightweight containers or services, which imposes high migration times so that implementation is not sufficient enough for commercial use.

Lastly, there was research done in regard to service replication in edge clouds [43]. The authors present a framework which was hosted in edge clouds acting as orchestrators to relocate stateless services between clouds using service replication. It utilized lightweight containers that are to be

replicated for achieving low-latency communication. The issue with this research is that it firstly does container replication and second, only supports stateless services. Container replication itself adds more overhead to the edge cloud but provides faster migration of services and low-latency communication. Since it also only supports stateless services, it means that any service requiring a TCP connection will be interrupted and shut down, since the container has to be stopped, moved and restarted.

3 Design

This chapter presents the design approach taken for this thesis and a proposal for a Service Migration framework.

3.1 Platform as a Service for Service Mobility

The system itself is designed and deployed as a Platform as a Service (PaaS). Figure 3.1 shows where the Service Mobility Framework will be deployed. The proposed Service Mobility Framework called Service Mobility Orchestrator (SMO) is part of the edge cloud and is comprised of different modules. Additionally, services that are deployed in edge clouds can utilize the SMO and its features via the Service module API.

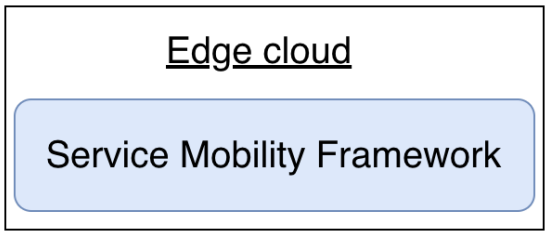

Figure 3.1: The deployment of the Service Mobility Framework in an edge cloud.

3.2 Design requirements

For the Service Mobility Framework to be commercially viable, it has to fulfil certain design requirements as specified below:

1. Utilize standardized communication protocols.
2. Make use of 5G standards and messaging protocol.
3. Minimize the total service migration time and downtime.

These requirements are set to make the solution interoperable with any service deployed in the edge cloud that requires service mobility functionality. Utilizing 5G standards and identical messaging protocols, makes it compatible with the 5GC as well.

The total service migration time and downtime are measured at the moment of a triggered handover signal from the 5GC until a successful service migration has completed.

3.3 Design Framework

The Service Mobility Framework which is deployed as an Orchestrator, SMO, is developed with four main modules, event signaling module, service module, handover negotiator module and routing module. The modules communicate using RESTful calls so that they can interact with each other and allow other services to utilize the SMO. Figure 3.2 shows the different modules in an edge

cloud and how the communication can occur. The different modules are described below.

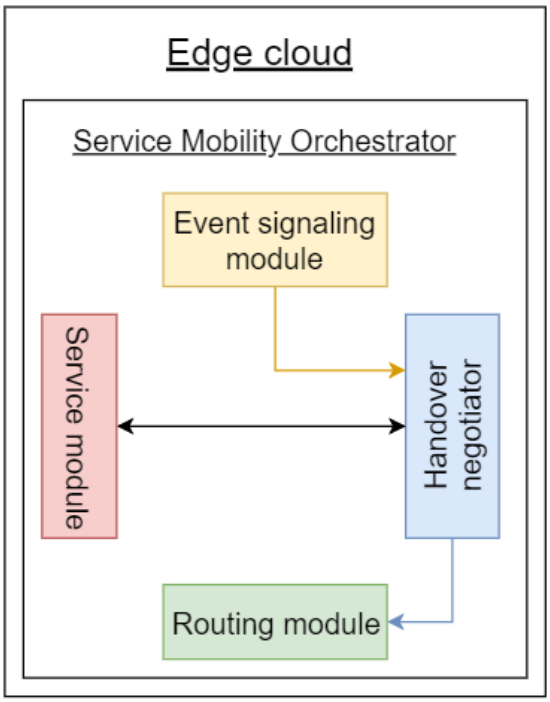

Figure 3.2: The general overview of the Service Mobility Framework, showcasing its four modules.

The SMO is deployed as an AF which, as mentioned in chapter 2.3, ensures that it can communicate with the 5GC or other NFs.

The only reason for using RESTful communication in the framework is that the 5G standard for NFs that use service-based interactions mainly use RESTful communication [27] [28]. This ensures that the required handover trigger from the 5GC can use RESTful calls to the SMO to initiate service migration.

The four modules are designed as microservices separated from each other, so that one module handles one task. The reason being that microservices allow for better scalability, low coupling and high cohesion. This means that any of the four modules can be distributed over multiple edge clouds or machines and still be effectively used to perform its task.

As mentioned, the messaging protocol used, are based on the 5G standards. This is also applied to the SMO. The reason for this is that there will not be any need new messaging model as the current standard can be re-used for this purpose. The current messaging model also provides more than enough messaging attributes to perform service migration.

3.3.1 Event signaling module

This module is responsible for receiving handover triggers from the 5GC and forwarding it to the handover negotiator to initiate a service migration. This is the entry point in which service migration can occur.

3.3.2 Handover negotiator module

This module is like a controller for the SMO, it handles the communication to every other module in the SMO as well as to a peer SMO handover negotiator. In addition, it is designed based on the relocation procedures as presented in the standard for Mobile Edge Computing [44]. The procedure that this module will follow is defined as follows:

- **Relocation Initiation phase:** This phase is responsible for handling the handover trigger and perform a preliminary decision for service migration.

- **Relocation Validation Phase:** Based on the service migration decision from the initiation phase, this phase will check and ensure that there are enough resources to accept an incoming service, based on certain service requirements.

- **Relocation Preparation Phase:** This phase will synchronize the source and destination involved for service migration. This means that the target should e.g. trombone traffic to the source until the completion phase.

- **Relocation Execution Phase:** This phase will find the service that is serving the UE, based on the relocation method, and then perform live migration to the target edge cloud.

- **Relocation Completion Phase:** As the aforementioned phases have completed, this phase is reached where old and temporary resources will be released.

Additionally, this module has a topology of its peers to know of the available migration targets. The module also provides a service mobility API where services can register themselves so that it knows of the available relocation methods and services. The API consists of two invocations; register and update.

3.3.3 Service module

The service module is a specific module that only handles one relocation method. It is designed in such a way that it is isolated from other types of services and relocation methods, as it only aware of its own services. The responsibilities of this module are:

- Register to handover negotiator.

- Keep track of its own services that it is hosting.
- Persist basic service information.
- Persist requirement data for services.
- Persist which UE a corresponding service is serving.
- Provide live migration feature.

Lastly, the service module can only be called from the handover negotiator and respond back to it.

3.3.4 Routing module

The routing module is a key part of the SMO as it provides accessibility to the different services during the relocation procedure. This module is only to be called from the handover negotiator and performs routing management. Its responsibilities are to:

- Generate OpenFlow rules.
- Install OpenFlow rules.
- Release old OpenFlow rules.

Those are the two general features of this module, but they also include the generation and installment of tromboning traffic to a service.

3.4 Framework Architecture

The proposed framework architecture is presented as shown in Figure 3.3. In the figure, the framework is using a certain service module, LXD, and the handover negotiator is seen by the 5GC as an Application Function. The LXD module is a specific module that is used for container migration and the service module can be any other live migration service module.

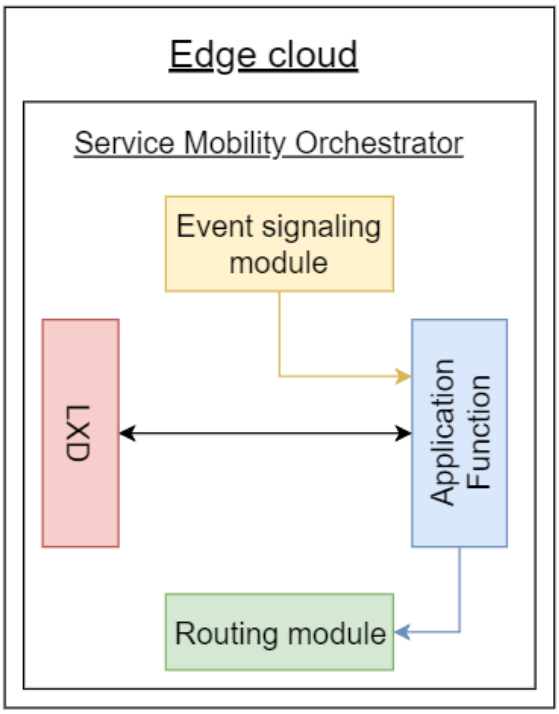

Figure 3.3: The Service Mobility Framework with the designed service module LXD and AF.

The choice of using LXD is because it is the only open-source tool that provides live container migration and is built in to Linux distributions. In addition, it provides the following features:

- Profiling of containers.
- Launch lightweight containers.
- Configure containers.
- Utilizes CRIU to perform live migration.
- Uses a database with container related information.
- Defines a standard way of container migration.

No other open-source tools are used as all the modules are implemented by following the 5G- and edge cloud standards. This is because there are no implementations of the modules yet.

To build up the entire framework, it requires part of the 5GC. Figure 3.4 shows the required component from the 5GC and how it may interact with the SMO. It shows that the SMF from the 5GC can be utilized to start a relocation procedure. The interactions are as follows:

1. SMF exposes an EventExposure API with messaging defined as per SMF Event Exposure standard [30].

2. Event signaling module interface towards handover negotiator.

3. Interface and API calls between handover negotiator and service modules.

4. Interface between handover negotiator and routing module.

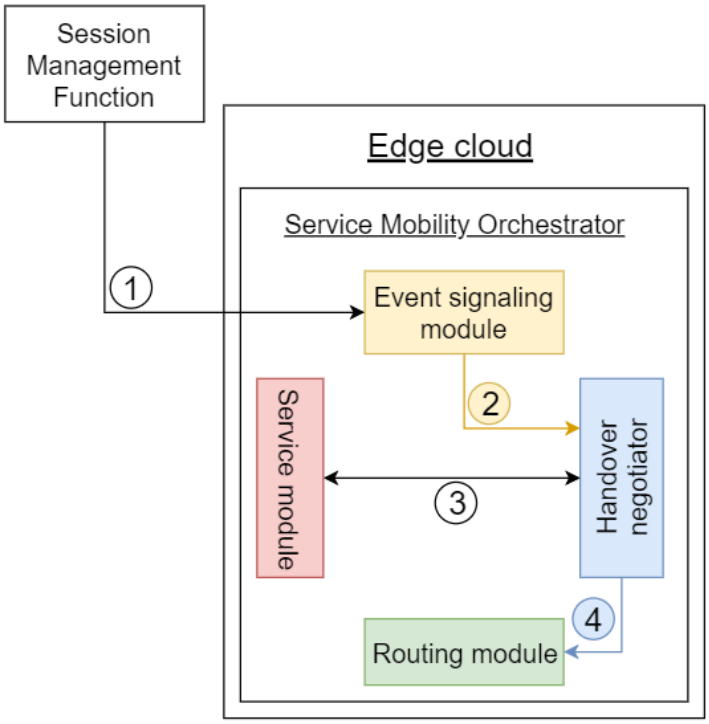

Figure 3.4: The Service Mobility Framework interactions including an SMF.

As previously mentioned, the interfaces and API calls are all RESTful between every module in the SMO and from the SMF.

3.4.1 Optimal Routing prototype

As the SMO framework is used with the assistance of the 5GC, the Optimal Routing prototype is used. The prototype, as previously mentioned, is developed at Ericsson Research, so the source code for the different NFs are available. This means that the emulated SMF that is in the Optimal Routing, but not shown in the figures, can be modified and extended with the event signaling module. The SMO can then listen to notification triggers, regarding user plane path change, from the Optimal Routing prototype and initiate live migration of services.

In Figure 2.6 Figure 2.7, the traffic flow always traverses to the selected UPF for the UEs session and then flows either to the edge cloud or elsewhere

based on routing rules in the UPF. However, the prototype sends all traffic to the edge cloud, but this can be changed to perform i.e. traffic breakout instead. For the purpose of this thesis, the UPF is not modified as the implementation and realization of the UPF is beyond the scope of this thesis.

3.4.2 Process

To initiate a service migration using the SMO, the proposed design provides only one entry point, the SMF to event signaling module. This means that the only case a live migration can occur is when the SMF decides to send a handover trigger to the SMO and that may be due to a UPF relocation. The handover trigger is caught by the event signaling module and propagated to the handover negotiator to initiate service migration. Once the handover negotiator receives the handover event, the relocation procedure starts. This means that the relocation phase takes over and determine the outcome of the service migration process.

The flow for the process, as seen in Figure 3.5, is described below and it explains how the different modules will interact and behave upon receiving different messages:

1. The SMF sends a notification message to the source edge cloud containing standardized attributes with relevant information required to distinguish e.g. which UE has changed path and which edge clouds are involved.

2. The source event signaling module listens to the notification from the SMF and propagates it to the handover negotiator.

3. Before the handover negotiator traverses the different relocation phases, it queries all registered service modules to get the service that is serving the particular UE that has changed path.

4. The source handover negotiator receives the initiation request for service migration with the corresponding data the SMF sent and starts the relocation procedure. It also checks whether there is a more suitable peer to select as the target edge cloud or not. Then, it communicates with the target peer, sending requests to it with the required information to receive a service, to be synchronized with the source edge cloud. It then makes a preliminary decision, based on if the target edge cloud has the registered service module as well, if service migration should occur or not.

5. During and before step 6, the handover negotiator also communicates with the routing module to install OpenFlow rules in the source and target edge clouds so that the service is accessible from the UEs perspective.

6. When the handover negotiator reaches the execution phase, it sends a live migration request to the service module in charge of the serving service.

7. At this point, the relocation execution phase has successfully finished, and the service has live migrated from source- to target edge cloud and new OpenFlow rules are installed to accommodate that. This will put both handover negotiators in the source and target in the relocation complete phase.

Steps 1-7 are the proposed design flow and it is done in combination with part of the 5GC. To illustrate the flow with Optimal Routing in place, Figure 3.6 is presented. The figure shows four stages using Optimal Routing. In the first stage, the UE is communicating with a server X′ connected to site A, traversing the closest gNB and UPF for uplink traffic and then also the IAP for downlink traffic. Then at stage 2, a handover has occurred, and the UE changed physical location so that it now communicates with server X′ in edge cloud X connected to site A via another gNB and UPF closer to the UE, site B. The SMF and SMO are not shown here for simplicity but in stage 3 the SMF has triggered the SMO to perform live migration of server X′ to the target edge cloud Y. Lastly, at stage 4 server X′ has migrated to edge cloud Y and the new traffic flow is installed to only traverse via site B, closest site, to server X′.

Additionally, as previously mentioned in section 3.3.2, tromboning rules are installed at a certain stage of the service migration process. In Figure 3.6 at stage 3, the communication between UE and service is tromboned which means that, since the UE has moved from site A to B, but the service is in site A, the traffic path flows through site B and its components, then towards site A and the service. The route handler in the SMO installs these tromboning rules and this is because, with tromboning rules, the UE to service communication will not be interrupted, maintaining the session. Consequently, the service downtime is not affected by the tromboned traffic, leading to a lower service downtime.

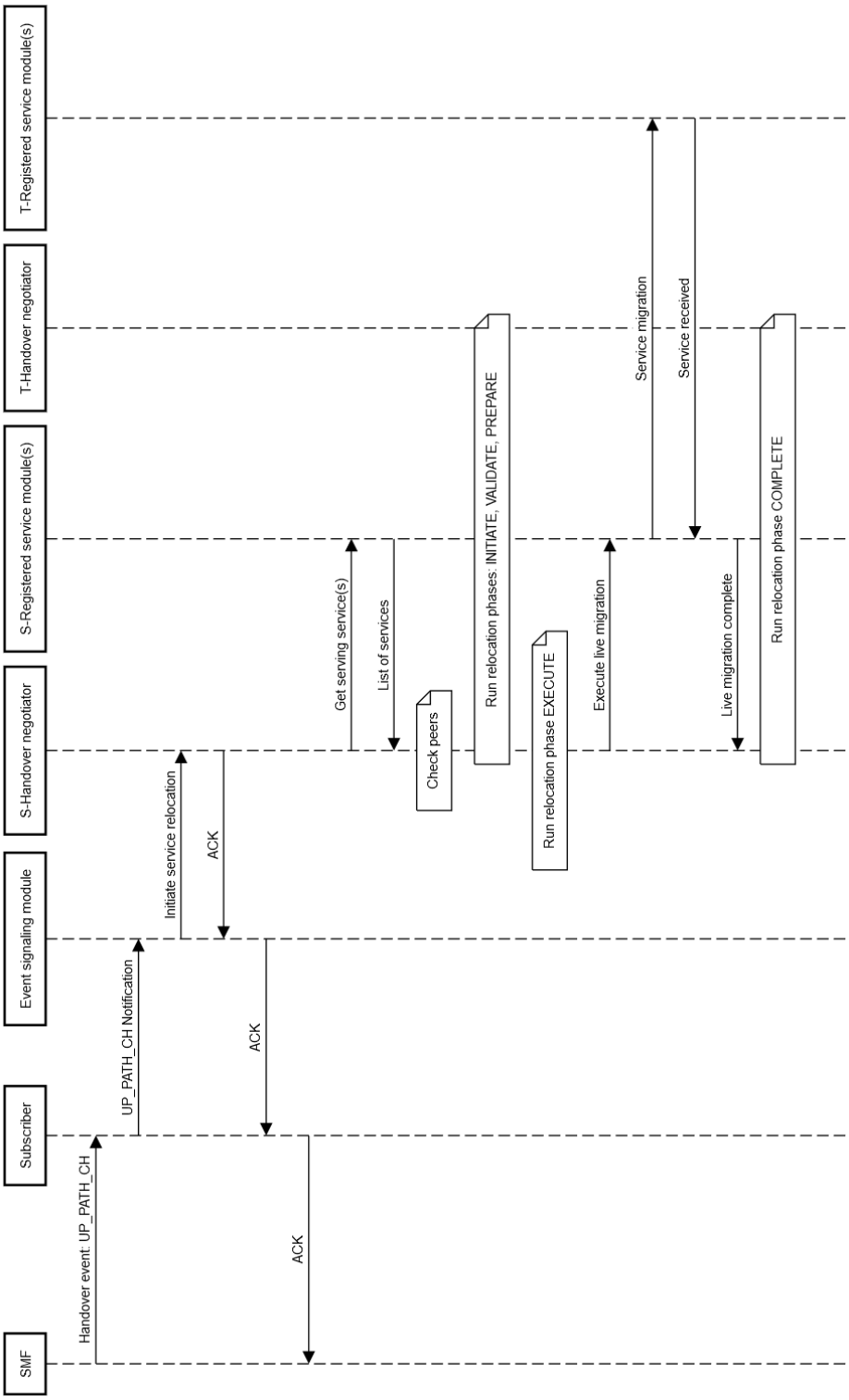

Figure 3.5: The relocation process as a sequence diagram. "S" refers to source and "T" refers target.

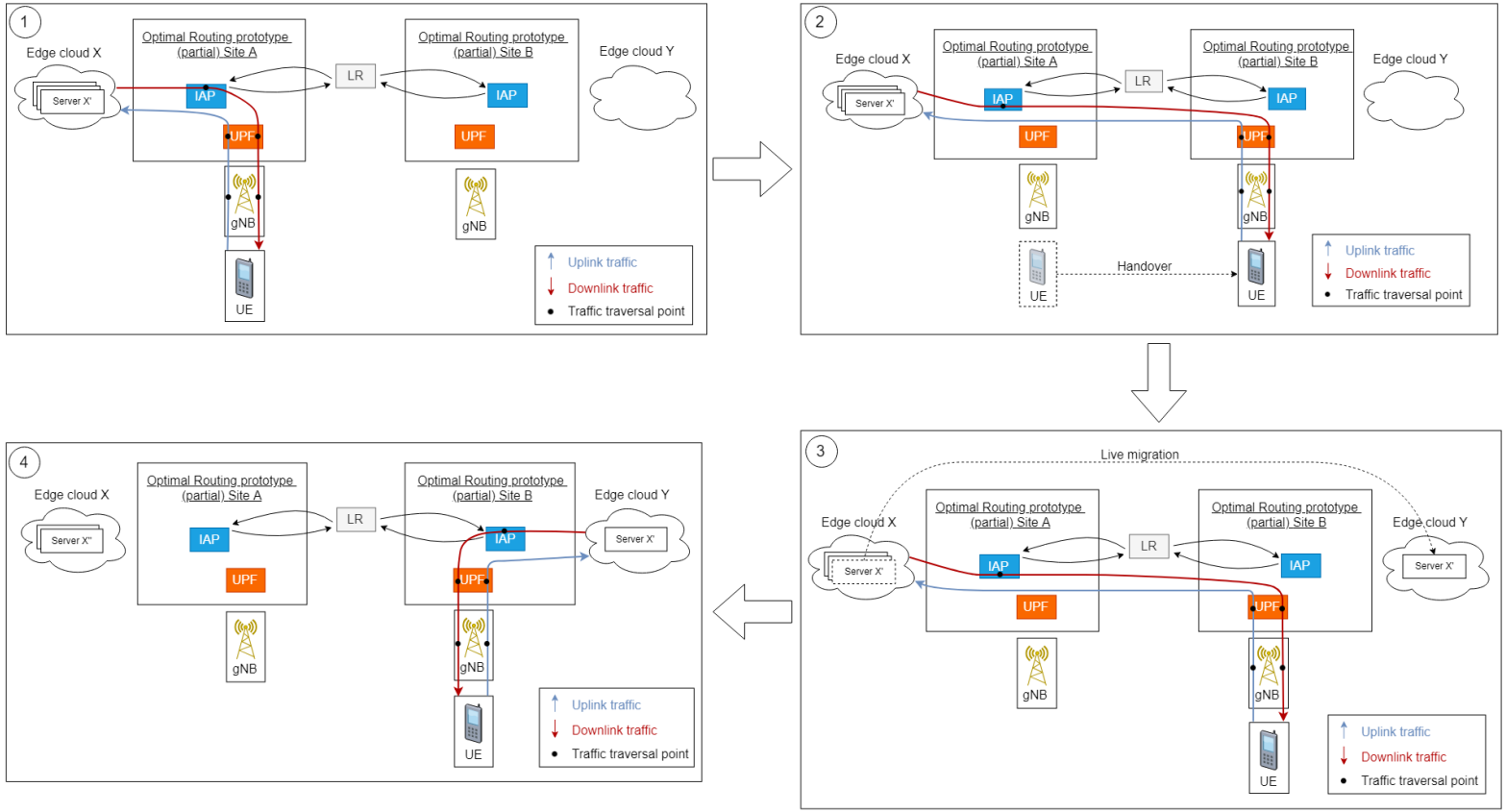

Figure 3.6: Optimal Routing and service mobility illustrated in four stages.

3.4.3 Messaging

The messaging protocol and the attributes that are used are described here. The chosen messaging and attributes from SMF to event signaling module, based on the SMF event exposure standard, the EventNotification message is chosen [30]. The EventNotification type consists of the following attributes that are used:

- **Event:** The event that is triggered, can be subscribed to.
- **SourceDnai:** The source DN access identifier, the source edge cloud.
- **TargetDnai:** The target DN access identifier, the target edge cloud.
- **SourceUeIpv4Addr:** The IPv4 address of the served UE.
- **TargetUeIpv4Addr:** The IPv4 address of the served UE, will be same as source since the IP address is kept.
- **SourceTraRouting:** The routing information for the SourceDnai.
- **TargetTraRouting:** The routing information for the TargetDnai.
- **DnaiChgType:** The DN access identifier change type, can be subscribed to.

See [30] for more attributes and their usages.

The attributes include two types that have sub-types, namely, the Event and DnaiChgType. The Event attribute can be of different event types but only **UP_PATH_CH** is considered here, see [30] for the remaining types. The DnaiChgType however, can be of two different types:

- **EARLY:** This is early notification of User Plane path reconfiguration. It means that if the subscriber is subscribed to EARLY notifications, the notification is sent before the new User Plane path is configured.
- **LATE:** This is late notification of User Plane path reconfiguration. It means that if the subscriber is subscribed to LATE notifications, the notification is sent after the new User Plane path is configured.

For the DnaiChgType, only the LATE notification will be utilized as the UPF relocation will occur first and then the service migration. This is because container migration is not executed at microsecond speeds, so the migration has to occur after UPF relocation, hence LATE notification. The SMF also publishes the aforementioned messages to functions that are subscribed to the corresponding DnaiChgTypes or events. Lastly, it has fixed URIs for the SMF event exposure API [30]:

- **/nsmf-event-exposure/v1/events:** URI used by consumers to get all available events from the producer.

- **/nsmf-event-exposure/v1/subscriptions:** URI used by consumers to subscribe to producers.

- **/nsmf-event-exposure/v1/notifications:** URI used by producers to invoke consumers and provide event messages.

The event signaling module has the simplest form of messaging which is also based on the SMF Event Exposure standard [30]. It has two features, subscribe and unsubscribe to the SMF and propagate events to the handover negotiator. The subscribe feature first asks the SMF for all the different events it exposes and subscribes to the ones that it requires. The unsubscribe feature invokes the SMF to not publish notifications to the event signaling module anymore. The propagation feature invokes the AF e.g. handover negotiator, at a fixed URI, "/application-function/v1/notifications" [30].

The handover negotiator, service module and routing module all use similar messaging attributes but with added custom attributes to fulfil registration, migration execution and routing management.

The handover negotiator to handover negotiator messaging is identical to that of the SMFs messaging. The difference is that there are three additional messages:

- **RelocationType:** This is the relocation method to be used.

- **RelocationPhase:** This is the relocation phase that is ongoing.

- **NotifUri:** This is a notification URI for the target to use for callbacks.

The handover negotiator also has the service API so that different service module implementations can utilize the SMO and interact with it, see Figure 3.7. The proposed API which is used to perform live migration of services is as follows:

- **Register service:** When a service module is implemented and deployed, it invokes this API to register itself with the AF. The service that is registered is then persisted and used to determine relocation methods and served services. The registration enforces a few messages, namely:

    - **RelocationType:** This is the relocation method to be used.

    - **ResourceRequirements:** This includes the resources that the service requires.

- **GetServiceNotifUri:** This is a URI callback to get the services that this service is hosting.
- **ServedServiceNotifUri:** This is a URI callback to get one particular service that is serving a specified UE.
- **ExecuteNotifUri:** This is a URI callback that is used to trigger a live migration of a specified service.
- **NotifAddress:** This is the address of where the service is hosted, as it may not be hosted locally.

- **Update service:** When a registered service wants to update any of its attributes, it invokes the API to update the persisted service resource.
- **De-register service:** When a registered service has terminated or wants to de-register itself, it invokes the API to do so and becomes unavailable for use.

Additionally, the handover negotiator provides the relocation procedure mentioned in section 3.3.2.

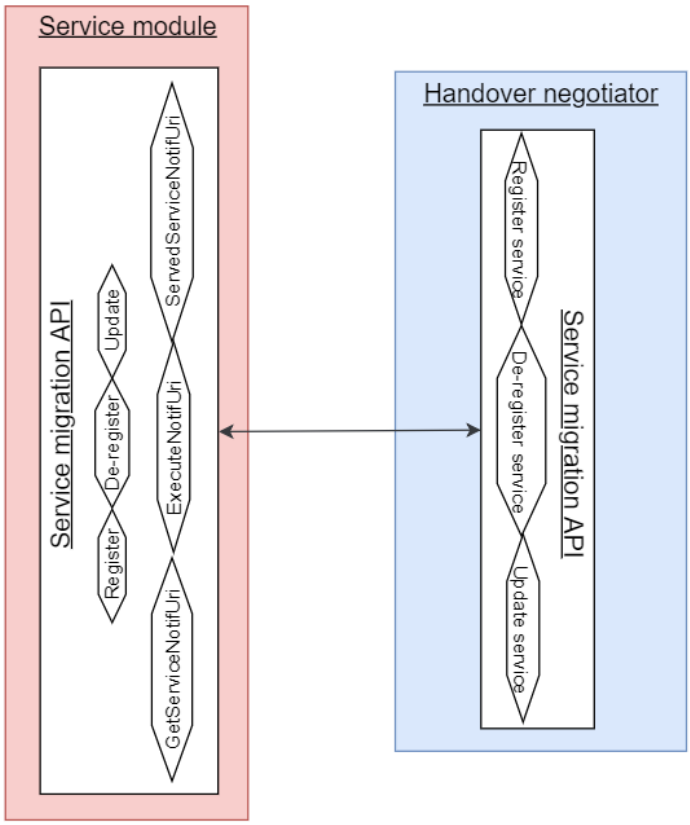

Figure 3.7: The service- and handover negotiator modules interaction via the service migration API.

Lastly, there is the service module messaging. As mentioned for the handover negotiator, the service module has the registration messaging to it. And enforced by the handover negotiator, the service module has three exposed API methods:

- **GetServiceNotifUri**:
- **ServedServiceNotifUri**:
- **ExecuteNotifUri**:

The API can be seen in Figure 3.7. These methods enable live migration of the service to the target edge cloud. The service migration API is available in the handover negotiator but used by the service module, hence the enforced calls.

4 Implementation

This chapter explains how the designed proposed framework is implemented and how it is assisted by 5GC NF.

4.1 Phases

Since the handover negotiator acts like a controller for the SMO, it is implemented using relocation phases, mentioned in section 3.3.2. The relocation phases for the SMO are similar to that of a state machine, see Figure 4.1. The figure shows that when a user plane path change notification trigger has reached the SMO, it will traverse the following relocation phases for the source edge cloud:

1. **Initial State:** The user plane path change notification has been captured and placed the source AF in the INITIATE relocation phase. This indicates a live migration initiation.

2. **INITIATE:** As the serving service for the UE has been found before the INITIATE phase, this phase will first verify that there is not a better peer to relocate to, based on the topology database. Then a suitable peer is chosen as the target for service relocation and verifies that it fulfils the requirements for receiving the service.

3. **VALIDATE:** The validation phase will prepare the requirements for the service that are needed to perform service migration and OF rule installation.

4. **PREPARE:** In the preparation phase, the source knows that the target has approved of the relocation. So, OF rules for incoming tromboned traffic is generated and installed here.

5. **EXECUTE:** The execution phase invokes the live migration by calling the callback method of the registered service that is hosting the serving service.

6. **COMPLETE:** If the execution phase was successful, this phase is reached. This will generate and install OF cleanup rules to remove the installed tromboning rules installed in the preparation phase. The serving service is also removed from the service database.

7. **FAILURE:** If there was a failure in any of the phases, this phase is reached. This will generate and install OF rules for incoming tromboned traffic to the service, as it means that it was not relocated but the UPF has still changed.

8. **Final State:** The final state means that there was a successful or failed live migration and the relocation phases have ended.

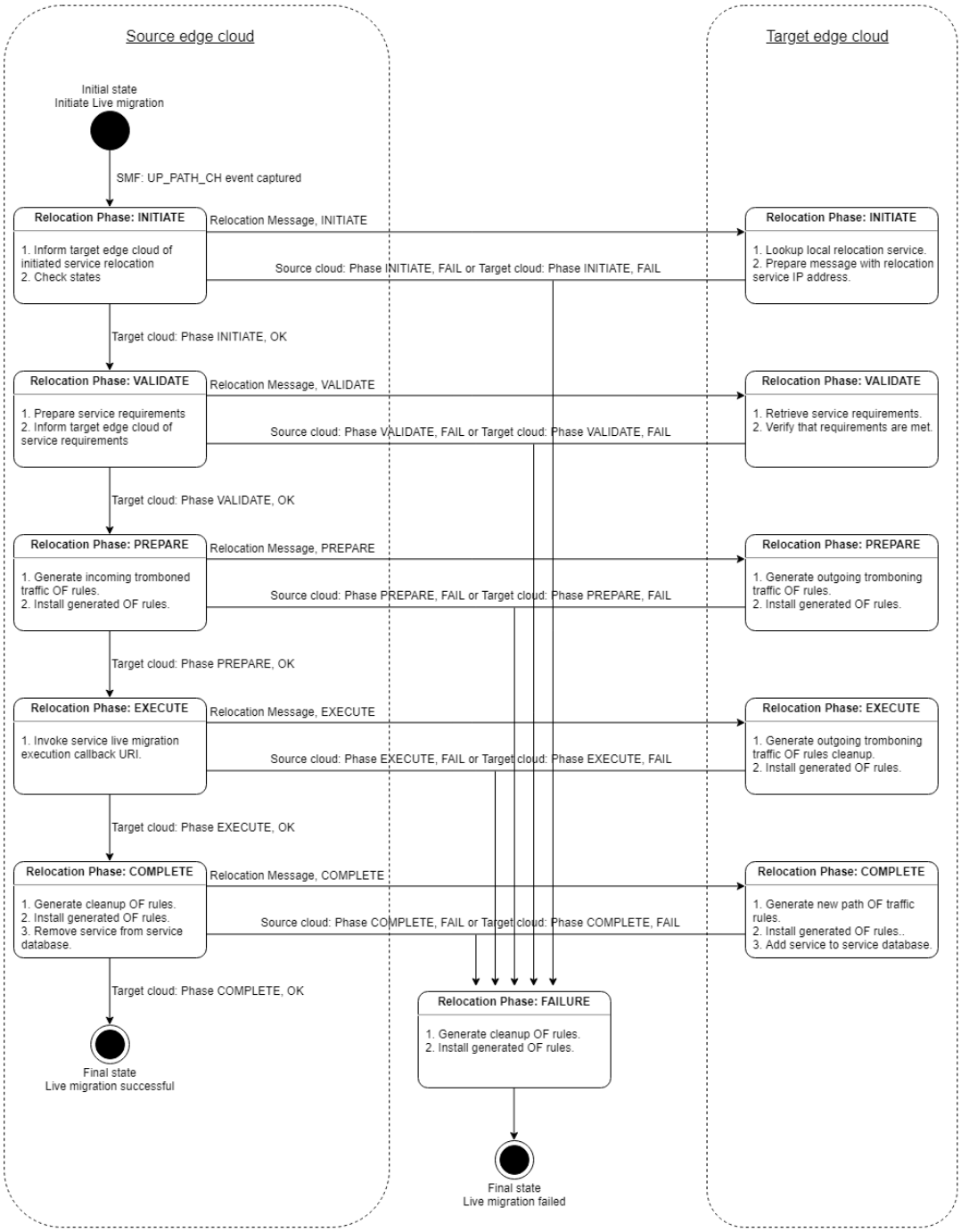

Figure 4.1: The implemented relocation phases illustrated. Every request, response, flow and action are shown.

And for the target edge cloud:

1. **INITIATE:** The source has invoked this phase for the target. This phase will verify that the service module exists in the service database e.g. it is registered to the AF. This phase will also extract the IP address of where the registered service module is hosted and append it to the relocation

message. This only occurs if the service module was found in the database.

2. **VALIDATE:** The validation phase will retrieve the service requirements from the source and verify that they are met so that the service can be relocated here.

3. **PREPARE:** In the preparation phase, OF rules to trombone traffic to the peer are generated and installed.

4. **EXECUTE:** When the execution phase is reached, the service has successfully migrated. So, this will generate and install OF cleanup rules to remove the installed tromboning rules installed in the preparation phase.

5. **COMPLETE:** If the execution phase was successful, this phase is reached. OF rules for the newly relocated service regarding its new traffic path are generated and installed here. The serving service is also added to the service database.

6. **FAILURE:** If there was a failure in any of the phases, this phase is reached. This will generate and install OF rules for outgoing tromboning traffic to the service, as it means that it was not relocated but the UPF has still changed.

For both the source and target edge clouds, every initiated relocation procedure will run in the above sequence. Every relocation phase will send and receive a relocation message containing relevant data for each phase. The messaging protocol is RESTful and uses the HTTP request-response communication model. Additionally, each message can either be an indication to proceed with the next phase or trigger a failure. If the message indicates a failure, the procedure jumps to the failure phase immediately.

4.2 Service Mobility Orchestrator modules

To ensure that the implementation is efficient, the different modules are implemented separately, and each module only contains its own necessary features. This means that each module can only perform one certain task and not be involved in other modules.

Based on the design goals and requirements set in chapter 3 for the SMO, the framework as seen in Figure 4.2 has been implemented.

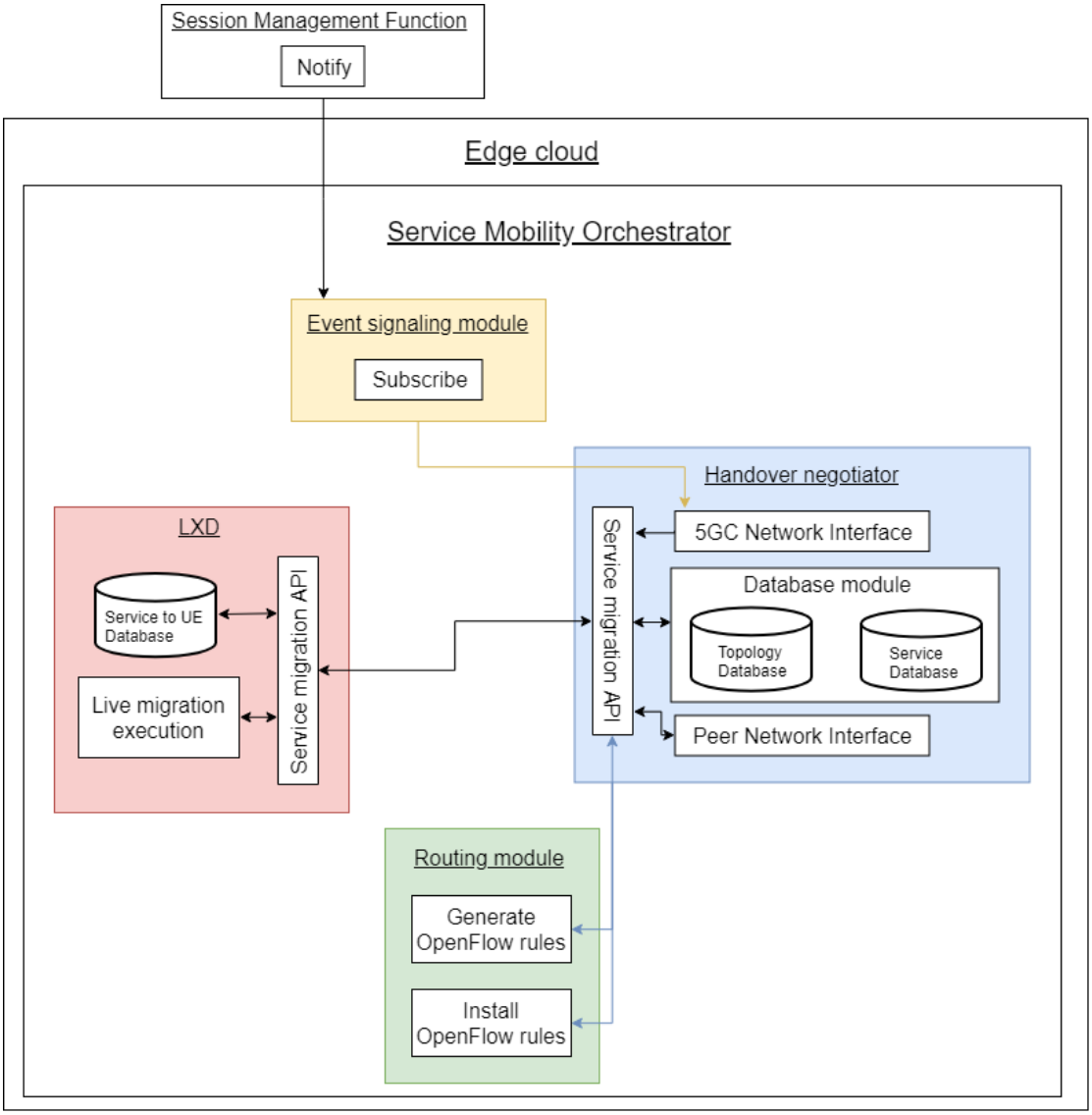

Figure 4.2: The full-fledged prototype implementation of the Service Mobility Framework.

The prototype is implemented entirely in Java using Spring RESTful services. To represent the flow of the implementation, it uses the phased flow presented in section 4.1. Using Java and Spring for this implementation does not mean that this is the only way to implement the framework. As such, any other language and RESTful service can be used to implement the same framework, so the choice of which language and service to use is less relevant in this thesis.

4.2.1 Event signaling

The event signaling module was designed as a single module, however, it is comprised of two blocks. The SMF Notify block and the event signaling module Subscribe block. These two blocks combined is the event signaling module, see Figure 4.3.

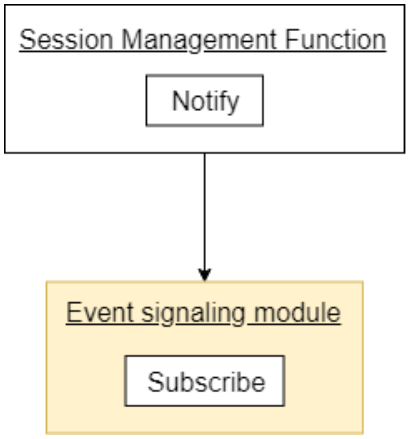

Figure 4.3: The event signaling modules two blocks, the Notify block is an extension to the SMF and the Subscribe block is part of the SMO.

The SMF Notify block is implemented to act as a producer, which means that it will only generate notifications and publish those notification to subscribers e.g. consumers. It is implemented as an extension to the SMF. The SMF sends a user plane path change trigger to its Notify block. The notify block will then:

1. Find every listening subscriber to the event.
2. Send notification with relevant information to the subscribers involved.
3. Reply to the SMF with successful or failure status.

The relevant information in step 2 is a message body in a HTTP request that consists of the required information to initiate a service migration later. The information includes every attribute presented in chapter 3.4.3 regarding notifications. With those attributes filled in and sent to the subscribers, they will know which kind of event it is e.g. UP_PATH_CH, which edge cloud is the source and target, which UE that moved, etc.

The subscribe block in the event signaling module is the consumer in the chain and has implemented two functions:

- Subscribe to SMF.
- Forward notification.

The first function will make the event signaling module behave as a consumer, so it subscribes to the SMF by specifying which events it is interested in. The producer will acknowledge the subscription and return a subscription id to the consumer. The second task, forwarding notifications, is simple, it only verifies the received notification from the SMF and then forwards it to the handover negotiator using the fixed URI for the AF.

4.2.2 Handover negotiator

The implementation for the handover negotiator acts as the controller part of the SMO. This module contains all the interaction APIs to every other module, except towards the event signaling module, see Figure 4.4.

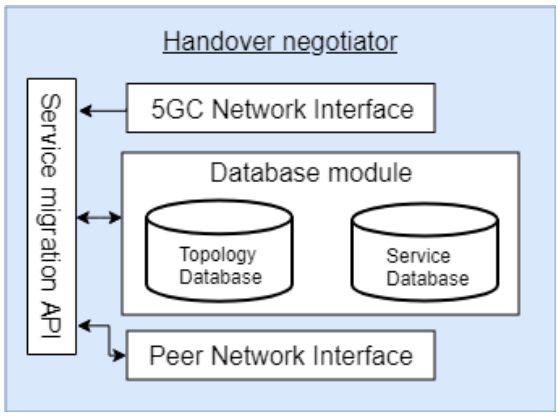

Figure 4.4: The handover negotiator module which acts as the controller for the SMO. It communicates with ever other module in the SMO.

The handover negotiator is divided into four sub modules:

- **5GC Network Interface:** The entry point to the handover negotiator is via this interface and it can only be interacted with from the event signaling module unidirectionally. The notifications from the event signaling module reach this interface and initiate a service migration. Consequently, the handover negotiator will start the relocation phases, in asynchronous mode, from here, described in section 4.1, by invoking the Service migration API.

- **Databases:** The database module consists of two databases, the topology database and service database, they store the neighbouring peers and every registered service module. The databases use a NoSQL database because there are no relational attributes involved in the persisted data.

- **Peer Network Interface:** This module is the outward-facing interface to other peers, specifically the target peer for relocating the service to. When the target peer has been chosen, it will request that peer and place it in the relocation phases for synchronization.

- **Service migration API:** The service migration API is the interface that communicates with every other module in the handover negotiator. It is here where the relocation phases interact the most. Its tasks are as follows:

    1. Retrieve the serving service by querying the registered services from the database module.

2. Find the suitable target peer to relocate the service to.

3. Start the source INITIATE relocation phase when the UP_PATH_CH event is captured.

4. Start the target peer INITIATE relocation phase.

Additionally, the API exposes calls for service modules to register themselves to it and update or de-register the services, as per the design in section 3.4.3.

4.2.3 Service

The service module that is implemented, LXD, is hosting the actual service. It is implemented to know of all its hosted services, which UE is served by which service and the requirements per hosted service, that is persisted in a database. Figure 4.5 shows that the Service migration API queries both the database and invokes the live migration of the service.

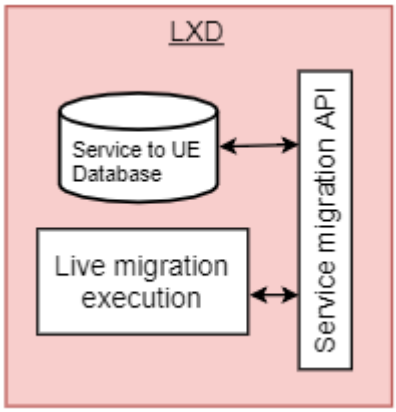

Figure 4.5: The LXD service module implementation. Exposes callback methods to the AF to perform live migration and service lookup.

Initially, the service module, on deployment, will register itself to the handover negotiator by providing which kind of relocation type it can perform, the requirements of the service, such as hardware resources. It also informs where the callback URIs are located so that the handover negotiator knows where to query to get all available services, get the serving service and execute live migration. This registration procedure is meant for the handover negotiator to know where the different service modules are located.

Additionally, the service module implements three callback URIs, enforced by the register part of the handover negotiator. So, it exposes three API calls in the service migration API:

- **GetServices:** This method will query the database of hosted services and return a list of them.

- **GetServingService:** When the handover negotiator requests each registered service for the serving service, this method is invoked. It takes a UEs source IP address as a parameter so that the database can be queried for that particular UE. If the services reply with a service, it means that it has found the hosted service that is serving that UE in the database. For the LXD module, it will reply with a container service that is serving the UE.

- **ExecuteMigration:** This is the core part of the service modules. When the handover negotiator has retrieved the serving service from the service modules, it will invoke this method when it is ready to relocate the service. When invoked, it expects the service as a data payload in the request message body, so that it knows which service to live migrate. It also expects the address of the target service module to live migrate to.

When the LXD module has been invoked to live migrate a container, it will begin to call LXCs API to dump and copy the container to the target. In Figure 4.6, the invocation of a live migration of a container A leads to internally calling the source and destination LXD daemon and source and destination CRIU. If the dumping and copying of the container are successful, the container will be stopped at the source and restored and started at the destination. Lastly, the source container will be deleted, and the live migration process is complete.

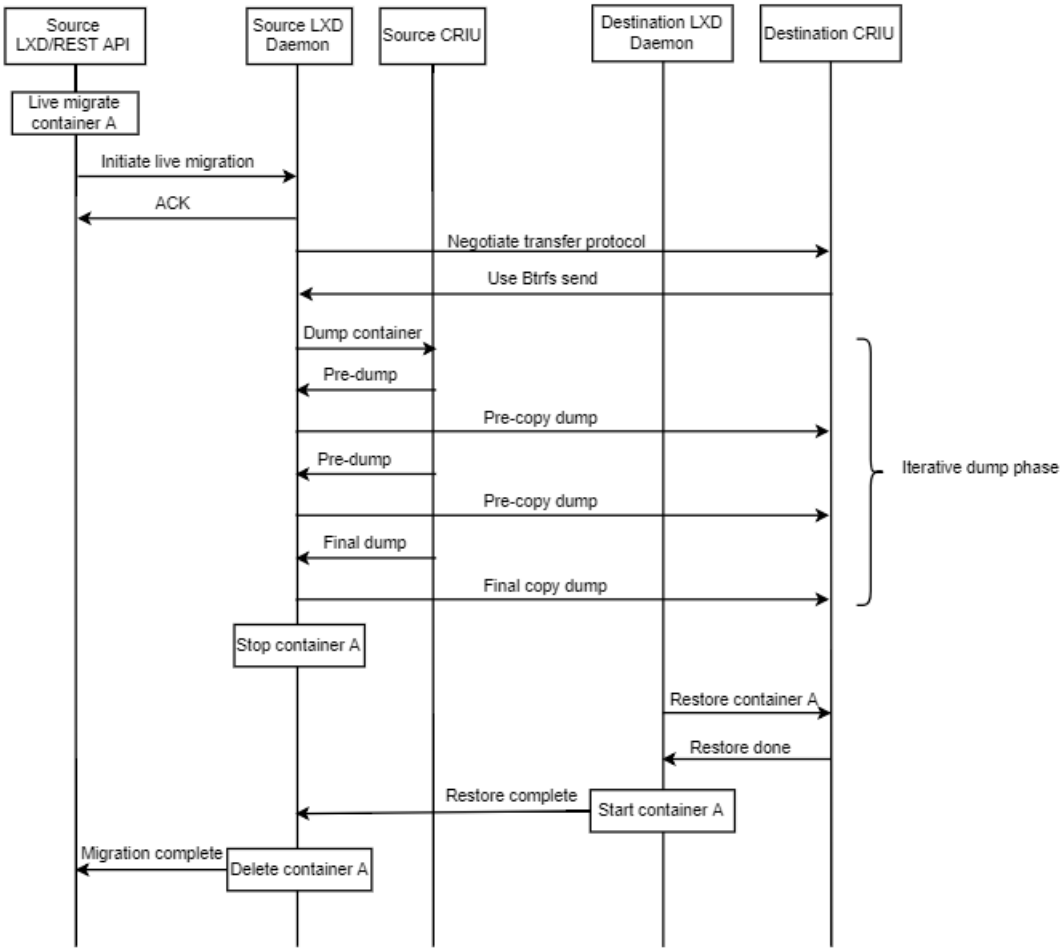

Figure 4.6: The flowchart of a successful live migration process using LXD, LXC and CRIU.

As LXD is used for live migration, it is important to know that each service will have one IP address for its lifetime, regardless if it is live migrated to another edge cloud or not. Consequently, this is critical for TCP connections established to the service. CRIU also ensures that the TCP sockets and connections are dumped and restored at the destination so that the communication can resume as soon as possible.

In the ExecuteMigration callback for the LXD module, the implementation is as follows:

1. **Find the target LXD module:** Before the LXD daemon is called to perform live migration of a container, it first requires a “remote” peer LXD module to be added to its internal remote list. With this knowledge, the handover negotiator provides the IP address of the target LXD module so that it can be found and used from the remote list in LXD.

2. **Copy the container profile to the target LXD:** Every container has a profile assigned to it which sets the requirements and configuration for it on startup. The exact same profile must exist in the target LXD, else

live migration will not work. The profile is copied from the source LXD to the target LXD by using the result from step 1.

3. **Execute live migration of the container to the target LXD:** Similar to copying the container profile to the target LXD, the live migration also uses the result from step 1 and moves the container there, following the steps in Figure 4.6.

4. **Clean up:** Delete the container from the local database.

Before step 3 of the execution callback, the routing module generates and installs OpenFlow rules to accommodate for the old traffic flow by tromboning the traffic. After step 3, the routing module generates and installs OpenFlow rules to accommodate for the new traffic flow by setting up a new traffic path.

4.2.4 Routing

The implementation for the routing module is exclusively based on OF rules. The module is implemented in a generic way. It uses a parent and child structure where the parent has the generic OF rules and methods required to implement. Children inherit the parent's generic rules and methods. This means that it is up to the child to implement the generation- and installation of rules. With this structure, different implementations of the routing module can be developed and integrated to the SMO.

The routing module has two generic blocks as shown in Figure 4.7.

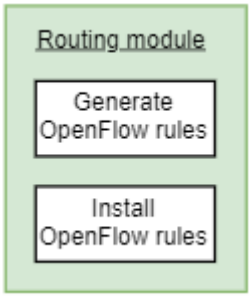

Figure 4.7: The routing module that handles every OpenFlow related tasks. Generates and installs flow rules.

The implementation of the generic block for OpenFlow rule generation are as follows:

- **GenerateOFRules:** This method generates the OF rules that are applied for the new traffic path after a live migration has successfully completed.

- **GenerateTromboningOFRules:** Before the live migration is invoked and as soon as the SMO has been informed of a user plane path change, the tromboning rules for the target edge cloud to the source edge cloud are generated.

- **GenerateCleanupOFRules:** After a successful live migration has completed, cleanup OF rules are generated to remove old and unused rules.

And the implementation for the OF rule installation block is:

- **InstallOFRules:** This method installs the generated OF rules whenever invoked.

4.2.5 Performance measurements

The SMO implementation above is a complete framework and to evaluate it, a specific performance measurement implementation is added to it. The implementation of the measurements is based on the last goal in section 1.4, i.e. measuring the downtime and total migration time of the migrated service. To fully measure the SMO, the total migration time is extended with additional measurement points namely, measuring the time spent in each relocation phase.

The performance measurements and the additional measurement points are an essential part of the framework as it will provide detailed information of potential bottlenecks and where it can be optimized further. To perform the measurements, a stopwatch is implemented in the SMO. The stopwatch can start- and stop taking time and calculate the elapsed time. For the total migration time, the stopwatch is started as soon as the SMO receives the user plane path change notification from the SMF and stopped when the live migration is successful and reached the relocation COMPLETE phase. Each relocation phase is also measured by starting the stopwatch as soon as a phase is started, and it is stopped when the phase has finished. Finally, the downtime is measured by capturing network traffic packets with a tool called tcpdump close to the UE. The data transferred for a live migration is also captured with tcpdump but on a dedicated migration link. For details of the measurement points see Table 4.1.

Table 4.1: The different measurement points to determine the performance of the SMO.

Involved part	Task	Description
Source edge cloud	Relocation phase INITIATE	The amount of time spent in this phase, informing the target edge cloud and checking states.
	Relocation phase VALIDATE	The time it took to prepare the service requirements and informing the target edge cloud.
	Relocation phase PREPARE	The time taken to generate and install OF

		rules for incoming tromboned traffic.
	Relocation phase EXECUTE	The time taken to invoke the service live migration execution callback URI. This runs the live migration synchronously.
	Relocation phase COMPLETE	The time taken to generate and install OF rules regarding cleanup and removing the service from the database.
Target edge cloud	Relocation phase INITIATE	The amount of time spent in this phase, finding the local relocation service and preparing the response with the relocation service IP address.
	Relocation phase VALIDATE	The time it took to retrieve service requirements and verifying them.
	Relocation phase PREPARE	The time taken to generate and install OF rules regarding outgoing tromboned traffic.
	Relocation phase EXECUTE	The time taken to generate and install OF rules regarding cleanup of tromboned traffic.
	Relocation phase COMPLETE	The time taken to generate and install OF rules regarding the new traffic path for the service and adding the service to the database.
Service	Downtime	The amount of time the client application at the UE did not receive any traffic from the service.
	Data transferred	The amount of data transferred during live migration from source- to target edge cloud.

SMO	Total migration time	The total elapsed time of the SMO from the point of receiving the user plane path change from the SMF until the relocation COMPLETE phase for a successful service migration.

All the measurement points, except for the downtime and total migration time, only measure the time spent in the phase and does not include the HTTP request-response times. Lastly, due to the SMO being based on source and target edge clouds, the measurements are also split per edge cloud instead of combining them into one measurement.

5 Evaluation

This chapter describes the experiments performed with the implemented SMO as well as its behavior and performance. In addition, the results of the evaluation are presented.

5.1 Video streaming service

The evaluation of the framework begins with creating a service appropriate for live migration. For this, a streaming service is chosen namely, a NGINX streaming service. The streaming service is hosted in an LXC and will be used for live migration. A UE with a video client will be able to access this service by establishing a new TCP connection towards it. In turn, the service will start streaming video content to the UE. Figure 5.1 illustrates this process.

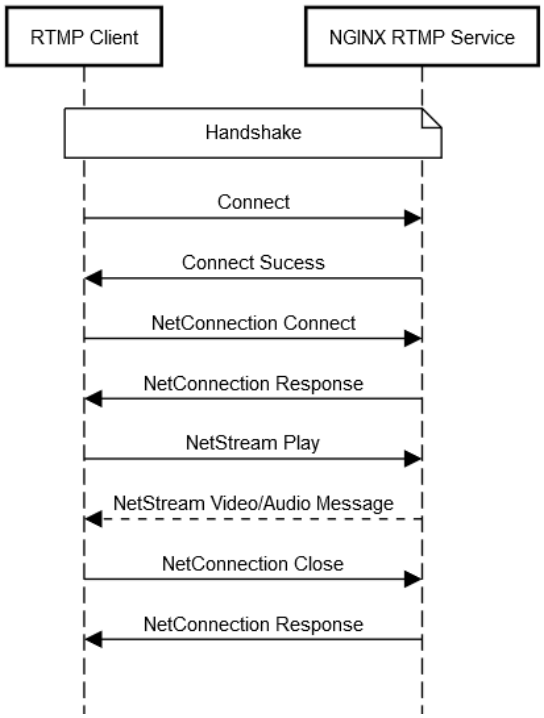

Figure 5.1: The RTMP on top of TCP process for establishing a new connection, streaming video/audio and closing the connection.

The NGINX streaming service uses plain Real-Time Messaging Protocol (RTMP) for the communication to stream the video and it works on top of TCP. RTMP combined with NGINX allows for video streaming by specifying a RTMP URI of the requested video, i.e. *"rtmp://<IP address or hostname>:PORT/media/video1.mp4"*. When the URI is entered, and connection started, the process in Figure 5.1 is executed.

As this service is hosted in an LXC, it will automatically be added to the service database to show that there is a service available for UEs. When a UE requests for a video from this service, the service module will track which UE the service is serving.

5.2 Test scenario

The test scenario that will be used is the one previously presented in Figure 3.5 and simplified in Figure 5.2. It represents a near real deployment of a handover use case. In the scenario, there are two sites with the 5GC setup, two gNBs, two edge clouds and one UE. A NGINX streaming service is deployed in one container on site A and the video to be streamed is located on the host machine on both sites. For the container to access the video file, its profile is configured to mount the video folder, from the host, inside the container. Initially, the UE is connected to the gNB connected to site A and establish a new connection towards the streaming service hosted in edge cloud X to stream the video. Finally, while the UE is streaming a video, mobility occurs, and a handover is performed from the gNB connected to site A to the gNB connected to site B. This also leads to the change of UPF. Consequentially, the handover triggers a user plane path change that the SMO is notified about and starts the live migration process using the relocation phases.

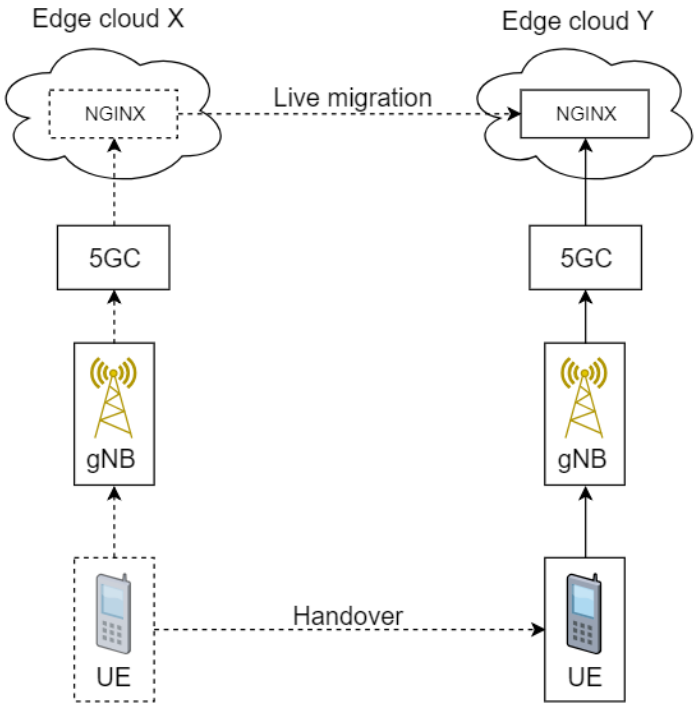

Figure 5.2: A simple test scenario combining 5GC and SMO.

Additionally, in Figure 5.2, the connection to the NGINX streaming service is a TCP connection under RTMP.

5.3 Openstack test setup

The test setup, as presented in Figure 5.3, tries to emulate the test scenario in a real-world deployment setup. The setup is deployed in Openstack, a cloud operating system, which means that every box in the figure is a VM. Each VM hosts either a NF, edge cloud or gNBs and UEs. Every VM runs Ubuntu 18.04 LTS.

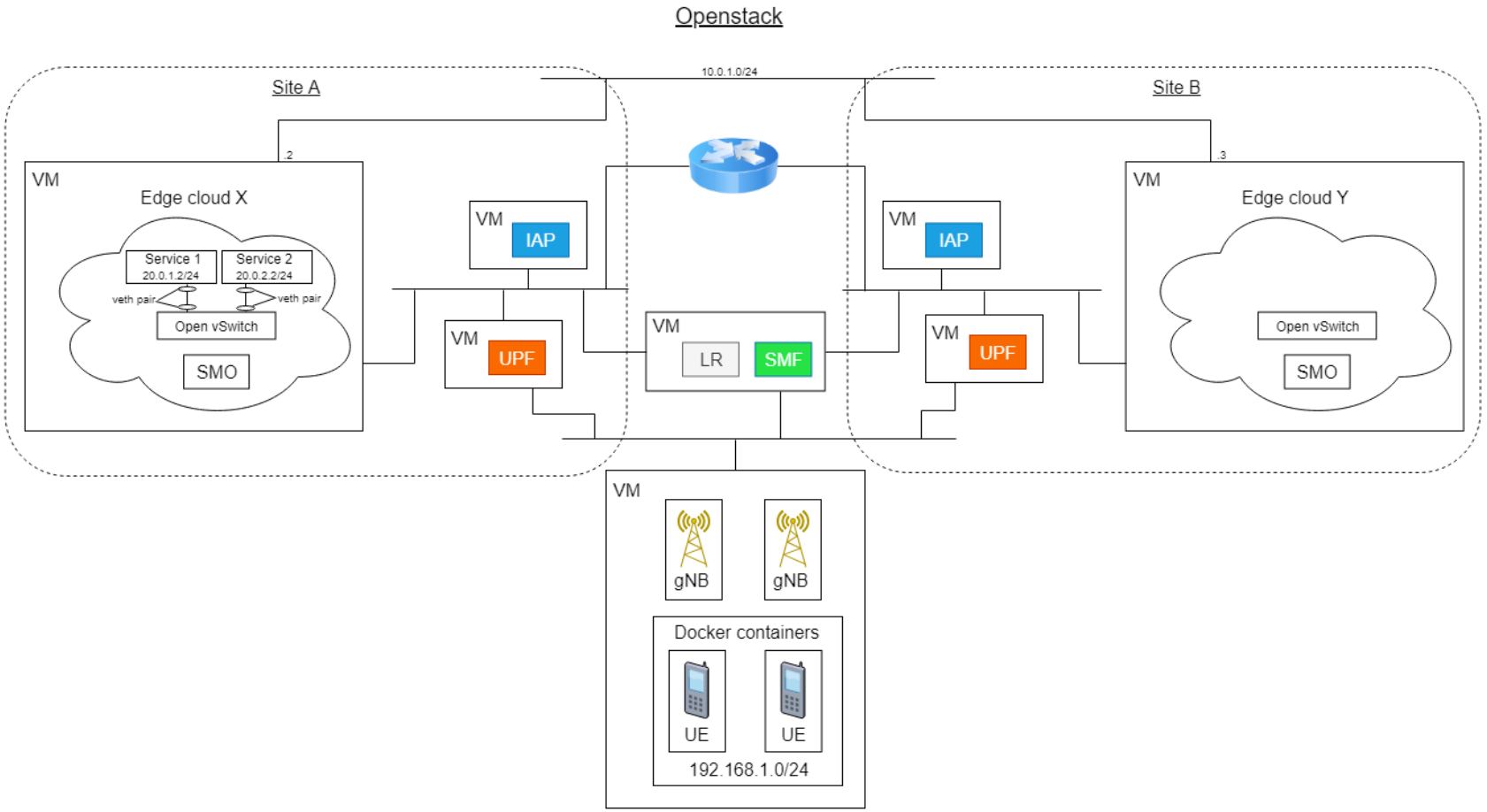

Figure 5.3: The emulated test deployment of OR and SMO.

Additionally, the VMs are set up as follows:

1. One VM hosting gNBs and UEs. The UEs are running as docker containers with a subnet unknown to the entire 5GC and edge clouds.

2. One VM hosting the LR and SMF, which sends user plane path change notification when a handover occurs.

3. Two VMs hosting two different UPFs, one for each site. Uplink traffic from the UE will be forwarded by the gNB in an encapsulated packet to a UPF, depending on which gNB the UE is attached to.

4. Two VMs hosting two different IAPs, one for each site. When downlink traffic is sent from the edge clouds, the traffic is always routed to the closest IAP. The IAP then encapsulates the traffic to the UPF where the user's session is established. Before that, if this information is not available in its local cache, the IAP contacts the LR.

5. Two VMs hosting two different edge clouds, X and Y. Each edge cloud has the SMO deployed there, one Open vSwitch (OVS) and potentially Linux containers. Both VMs have been dedicated 2 VCPUs and 8 GB RAM.

There are multiple different private networks in the setup but only the relevant ones are mentioned here and in the figure. In each edge cloud, there is a service network range, 20.0.0.0/16, it is used only for containers and different services. The 192.168.1.0/24 range is dedicated to the UEs and the 10.0.1.0/24 network range is used as the backhaul network to perform live migration on, e.g. the container dumps are transferred over this network. For each site, the IAP, UPF and edge cloud are all on the same network, e.g. site A is on one network and site B on another network.

The OVS in each edge cloud provides bridging for the Linux containers as well as OF rules. This means that, packets coming to an edge cloud with the destination of a service, it is captured in the OVS bridge, matching some OF rule and then sent to the service. So, it provides connectivity between UEs and services. Additionally, the containers utilize Virtual Ethernet Device (veth) pairs, one inside the container and one outside it. This veth pair is initially attached to the source OVS bridge and re-attached, after a successful live migration, to the OVS bridge on the target edge cloud.

The LXC that hosts the NGINX streaming service is running a clean Alpine Edge Linux OS with only necessary packages installed that are required to maintain NGINX and RTMP. This container is launched from an image file that is 12.02 MB in size and exists on both the source- and target edge cloud. The launched container, together with its specific profile, is roughly 23 MB in size. A container cannot be launched without a profile and the profile describes the configuration for a container. The container has a mounted

shared folder from the host to the container. This folder contains the video used for streaming, which is a 720p MPEG-4 video file with a size of 59 MB.

The chronological order of events of the test scenario is as follows:

1. A UE establishes a new RTMP connection towards one NGINX streaming service and starts streaming a video from it. The UE is set up to be connected to site A and connects to edge cloud X. The video client in the UE is also set to use a one second buffer.

2. After 30 seconds of streaming the video, the emulated SMF is manually invoked to trigger a handover, sending a user plane path change to the subscribed SMOs. This starts the relocation phases.

3. While the SMO is performing each phase, the final dump of the container is performed and transferred to the target edge cloud and the container is stopped on the source edge cloud. At this point, the container at the source edge cloud is still sending data to the UE. The container is then restored and resumed on the target edge cloud and the veth pair of the container is made active and added to the OVS bridge as well.

4. When the SMO has reached the COMPLETE phase and the live migration is successful, the container is deleted from the source edge cloud. OF rules are also installed to serve this newly added container on the target cloud.

5. At this point, the video streaming session continues, but now the newly migrated container is sending and receiving data to and from the UE.

5.3.1 Evaluation and analysis

The evaluation and analysis of the aforementioned test setup are based on the measurement points presented in section 4.2.5. The performance for each of the measurement points is reported in terms of the min, max, mean, standard deviation and 90th percentile. Additionally, the measured times are illustrated in a staple diagram for each edge cloud, showing the time spent in each relocation phase. The service downtime and total migration time are also presented in staple diagrams.

The figures in Appendix A represent 50 repeated runs of the test scenario and setup. For the source edge cloud, Figure A.1 shows each relocation phase and their respective measured times. It shows that the times spent in the corresponding relocation phases can vary quite a bit. The corresponding 50 runs for the target edge cloud are shown in Figure A.2. The figure shows how much time the target edge cloud spent in each relocation phase when invoked by the source edge cloud. The times for each phase do not vary too much except for the COMPLETE phase which has the most variation in time. In Figure A.3, the total migration time for the SMO is presented. It shows that for 50 test runs, the total time does vary but not that much compared to each run.

Lastly, Figure A.4 shows the service downtime of the migrated NGINX container. The results show the time it took for the UE to start receiving packets from the container again, after it stopped sending packets due to LXD.

To analyze the aforementioned figures further, a statistical approach is taken. The following analysis target the total migration time and service downtime to explain the contributing parts for the achieved values for the total migration time and service downtime. Table 5.1 and Table 5.2 show that each relocation phase, excluding the EXECUTE phase, has a relatively small impact on the total migration time. Additionally, the EXECUTE phase for the source edge cloud affects the SMO the most, with a mean time of 11377 ms. Further analysis into the large amount of time for the EXECUTE phase show that this phase is heavily dependent on the underlying relocation type. This means that the registered service to the SMO that takes care of the live migration, in this case LXD with CRIU, has some optimization issues.

Table 5.1: Statistical values for the source edge clouds different relocation phases.

Measurement point/Statistic	MIN (ms)	MAX (ms)	MEAN (ms)	STDEV (ms)	90th percentile (ms)
INITIATE	38,00	169,00	92,74	30,07	129,30
VALIDATE	2,00	21,00	6,92	3,99	12,10
PREPARE	18,00	89,00	40,54	17,40	62,40
EXECUTE	8674,00	15360,00	11377,24	1519,23	13325,50
COMPLETE	41,00	188,00	96,08	42,36	141,30

Table 5.2: Statistical values for the target edge clouds different relocation phases.

Measurement point/Statistic	MIN (ms)	MAX (ms)	MEAN (ms)	STDEV (ms)	90th percentile (ms)
INITIATE	1,00	28,00	6,70	5,40	13,40
VALIDATE	1,00	13,00	2,78	2,16	4,10
PREPARE	17,00	313,00	43,64	43,32	62,30
EXECUTE	16,00	57,00	29,36	11,38	46,30
COMPLETE	168,00	969,00	348,16	185,05	612,90

Since the known element that yields high total migration time is LXD, or the registered service, LXD is analyzed further. When testing, it is evident that the LXC move takes most of the time and this is due to CRIU, which is built into LXD. As mentioned in section 4.2.3, when a container is migrated LXD invokes CRIU which in turn performs iterative dumping of pages and transfers them to the target. The dumping phase of CRIU is highly dependent on the container size, the number of established TCP connections and in memory- and disk usage. Since the container has low disk usage, due to having a mounted folder of videos in it, and the container being roughly 23 MB in size, the potential bottleneck becomes the established TCP connection. In the tests, there is only one TCP connection towards the container, this

yields a higher migration time and that is because CRIU tries to dump pages iteratively and since there is a video streaming session, it is continuously dumping pages until it finally makes a final decision to create a final dump and freeze the container. The algorithm that CRIU uses for this may not have the best performance, which consequently will increase the migration time with TCP sockets.

As a final step to reduce the large time spent in the EXECUTE phase, the exact same test scenario was run again with the same test setup. But this time, the different aspect is that there is no RTMP streaming session or any TCP connection established at all. So, the container is running, but not serving any UE, hence it is a clean container that is not performing any task. This test showed that the live migration, even for an empty and clean container, yield similar results to that of having one UE streaming a video. This means that the issues are not with NGINX or TCP, rather it is CRIU and LXD that infer large times.

The relocation phases results, for both edge clouds, also show that the VALIDATE phase has the least impact on the total migration time while the PREPARE, EXECUTE and COMPLETE phases has the most impact. Since the VALIDATE phase is the only phase at the source edge cloud not performing any OF or service migration related tasks, its execution time will always be significantly lower than the other three phases. The PREPARE, EXECUTE and COMPLETE phases performs the heavier tasks of installing OF rules and executing live migration, so they naturally add to the time spent to finish.

The INITIATE phase was not mentioned because it does not perform any OF related tasks. The reason for this phase to have a larger time presence on the source edge cloud instead of the target edge cloud is that, at the source, the phase has a scalability problem. As it will query each neighboring edge cloud to find the most suitable one, it will add more time to the total migration time based on how many peers there are. In the testing scenario, there was only one peer, however, with more peers this measured time would increase. The target edge cloud, in this phase, performs a relatively cheap task, it simply does a database lookup of the registered service and replies, which is why it is greatly lower in time compared to the source edge clouds INITIATE phase.

In Table 5.3, the statistical results for the total migration time and service downtime are presented. Based on the previous analysis of CRIU increasing the time spent in the EXECUTE phase at the source edge cloud, it correlates with the total migration time. This is because the mean time for the EXECUTE phase is 11377 ms and the mean time for the total migration is 11767 ms. The delta between the two is 390 ms, which means that the total migration time is heavily reliant on the registered service to implement a good performance-oriented relocation method.

Table 5.3: Statistical values for the total migration time and service downtime.

Measurement point/Statistic	MIN (ms)	MAX (ms)	MEAN (ms)	STDEV (ms)	90th percentile (ms)
Total migration time	8966,00	15816,00	11767,66	1523,17	13627,00
Downtime	5619,96	9143,86	7199,23	992,75	8603,99

The service downtime is the most critical part of the entire chain in the SMO and, as presented in Table 5.3, the mean downtime was 7199 ms. To analyze why the downtime is as high as it is, further tests are performed.

One of the tests performed, to try and explain the high downtime, was to use the same setup and scenario but use ping instead, to avoid TCP sockets. This will show that if the downtime is similar to that of the streaming service, the issue is not with TCP sockets, rather CRIU and LXD instead. This test was performed 20 times and the UE pings, the now clean container, with an interval of 0.1 seconds. The results show that the mean total migration time was 14184 ms and the mean downtime was 8522 ms. This means, as previously mentioned, that the migration of TCP sockets is not inducing large downtimes to the migration process.

To potentially explain the high downtime, there are multiple key factors that may impact the downtime. They are:

- **Mounted shared folder of videos:** With the shared folder, on each migration, the folder is re-mounted in the container which means that the video is not accessible for the NGINX streaming service until this is done, increasing the service downtime.

- **Number of TCP sockets:** As previously mentioned, CRIU iteratively dumps pages and with an increased number of TCP sockets, these may become larger. This means that it will take more time for CRIU to finish, increasing the downtime.

- **OVS and OF:** The OVS and OF rules have a small part on the service downtime and that is when the target edge cloud installs the OF rules in the COMPLETE phase. Based on Table 5.2, the mean time of this is 348 ms, which mean that, on average, the OVS and OF adds little time to the service downtime in total.

- **In disk- and memory usage:** The in disk- and memory usage is also related to the CRIU dumping process. This means that whenever the disk or memory usage increases, CRIU dumping takes more time to finish.

- **CRIU restoration process:** Lastly, there is the actual CRIU restoration process occurring on the target edge cloud. This depends on

the number of pages, meaning the amount of data to use for restoring the container, every process and TCP sockets, which will increase the service downtime.

Furthermore, when running the tests, the video client VLC, does in some runs show no service interruption without any video tearing, while in other runs there is slight video tearing or a few seconds of service interruption. The case of no service interruption cannot be explained by the test results for the service downtime because, even though the video client has set a 1 second buffer, the service downtime is much larger than that. This leads to believe that either VLC or RTMP does some additional buffering or task that can mitigate large downtimes, as perceived by the UE. A simple proof that makes VLC an uncertainty is that when the video streaming is running in VLC and the NGINX streaming service is terminated in the container, it shows that VLC is still streaming the video for more than 4 seconds, even though the video buffering was set to 1 second.

There is also an issue when running live migration using LXD in quick succession, the total migration time increases greatly, with migration time greater than 30 seconds. The potential explanation for this is that, when a new RTMP session is established and then migrated, there is old data regarding this connection remaining in memory without being cleaned up. Then after a few migrations, the in-memory usage is larger than expected for one session making CRIU spend more time in the dumping and restoring phases. Additionally, this was mitigated by performing live migration and when this issue became noticeable, stop the streaming session, wait 30 seconds, start it again and live migrate.

The data transferred measurement point resulted in 69 MB for every test run, excluding the aforementioned test runs regarding the quick succession issue. This is the data transferred over the migration link that CRIU and LXD use to move the pages between edge clouds. The reason for it being the same amount of data every time is that, for one streaming session established, using the same video to stream, there is no difference in dumps that CRIU must perform and transfer, so the in-disk and memory usage and TCP sockets do not increase.

Lastly, the Openstack setup does not yield satisfying results regarding the total migration and service downtime. The reason for this was ultimately observed to be the environment in which LXD and CRIU are running. Since the edge cloud VMs in the setup uses 2 VCPUs, the performance of the CRIU dumping process is reduced significantly, compared to using 1 vCPU. Additionally, since the only networking option in Openstack is to use the built-in networking tools, consequently the live migration network link is not a direct link between the two edge clouds, leading to a slow transfer of the dumps that CRIU generates. These observations were proven, not only by the 50 tests, but also by using LXD and CRIU to create and restore stateful snapshots of the same NGINX container used in the tests. This showed that,

creating a stateful snapshot on a machine that has 2 vCPUS takes ~3 seconds, whereas, on a machine with 1 vCPU it takes ~0.7 seconds. The duration for restoring the snapshot is similar on both environments. This means that the first bottleneck in live migration using LXD and CRIU is the number of cores the host machine has. Secondly, as the network link used for live migration could not be modified, due to the limitations in the Openstack setup, the need for a new test setup is required, where the performance of the SMO, LXD and CRIU can be evaluated. The new test setup, with an optimal environment where edge clouds have 1 CPU core and a direct network link between them, is introduced below.

5.4 Performance test setup

The performance test setup, as presented in Figure 5.4, also tries to emulate the test scenario, presented in section 5.2, in a real-world deployment setup, but without the 5GC. The 5GC is excluded in this setup for two reasons, it is simplified, and it does not affect the live migration or SMO in any way. The main purpose of the 5GC was to utilize the handover trigger but this can easily be emulated instead. As such, the 5GC did not add any negative results to the total migration and service downtime.

The setup is deployed in a physical machine with two VMs running there, acting as different sites. Both the VMs and the physical machine runs Ubuntu 18.04 LTS.

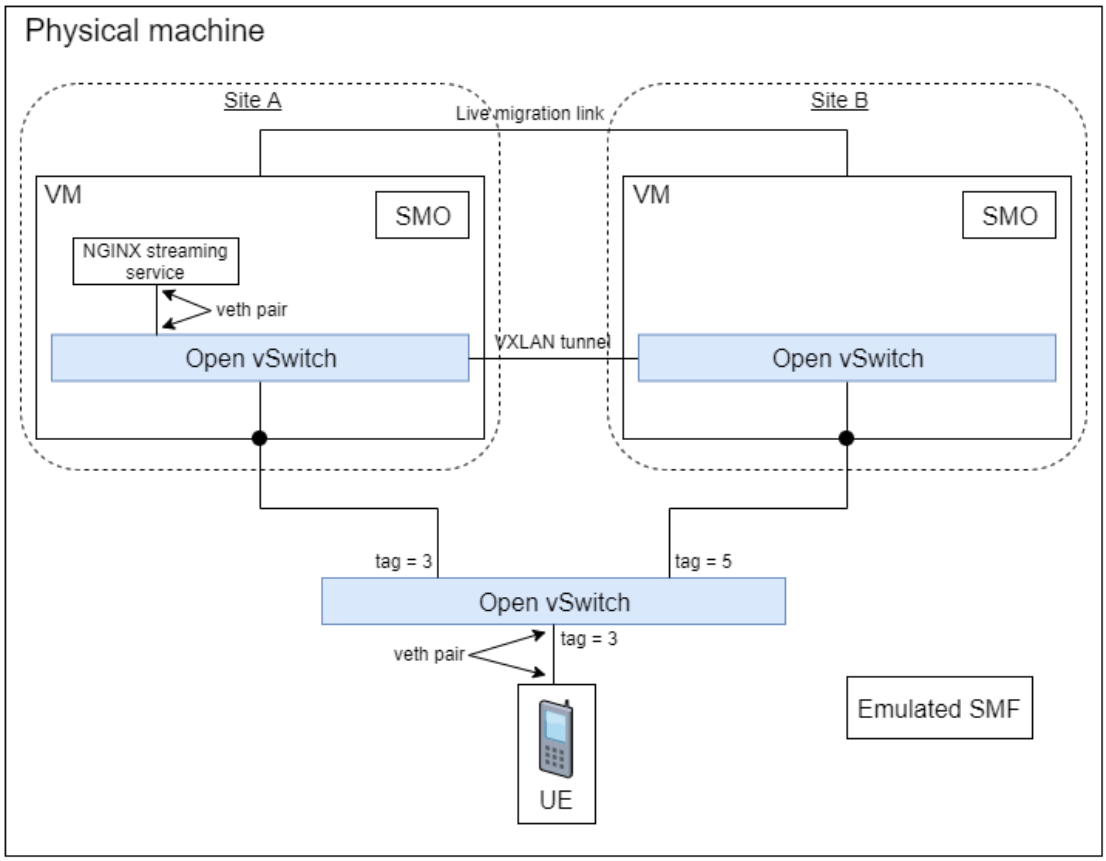

Figure 5.4: The emulated performance test setup on one physical machine with two VMs.

Additionally, the setup is as follows:

1. Two VMs hosting two different edge clouds. Each edge cloud has the SMO deployed there, one OVS and potentially Linux containers. Both VMs have been dedicated 1 CPU core and 6 GB RAM.

2. The physical machine hosts the two VMs, an OVS, a UE and an emulated SMF.

Similar to the test setup in section 5.3, there are multiple different private networks in the setup. They are not shown in Figure 5.4, but they are the same as for the previous test setup, the service network range, 20.0.0.0/16, the UE network range, 192.168.1.0/24 and the backhaul network range for live migration, 10.0.1.0/24. The backhaul network is an internal network that behaves as a direct link between the two VMs.

The OVS in the physical machine provides the ability to switch between the communicating VM. This means that there are two TAP interfaces in the host that are attached to the OVS bridge and each VM. Additionally, the two TAP interfaces are tagged to differentiate the VM connectivity so that the traffic flow can be switched from one VM to another. Site A is tagged to 3 and site B is tagged to 5.

The UE is a video client that has a veth pair, one end attached to the OVS bridge and the other to the client. The end attached to the OVS is also tagged to know which VM to send traffic to.

The OVS in each edge cloud and the LXC that hosts the NGINX streaming service is identical to the test setup in section 5.3.

The chronological order of events of the test scenario is as follows:

1. A UE establishes a new RTMP connection towards one NGINX streaming service and starts streaming a video from it. The UE is setup to be connected to site A by setting the OVS tag for the UE veth interface to 3. The video client is also set to use a one second buffer.

2. After 30 seconds of streaming the video, the emulated SMF is manually invoked to trigger a handover, sending a user plane path change to the subscribed SMOs. The trigger sets the OVS tag for the UE veth interface to 5, changing the communicating site. This starts the relocation phases.

3. While the SMO is performing each phase, the final dump of the container is performed and transferred to the target edge cloud and the container is stopped on the source edge cloud. The container is then restored and resumed on the target edge cloud and the veth pair of the container is made active and added to the OVS bridge as well.

4. When the SMO has reached the COMPLETE phase and the live migration is successful, the container is deleted from the source edge cloud. OF rules are also installed to serve this newly added container on the target cloud.

5. At this point, the user resumes the RTMP video streaming session from the place it left off, towards the newly migrated container, served by site B.

5.4.1 Evaluation and analysis

The evaluation and analysis of the aforementioned test setup are based on the measurement points presented in section 4.2.5. The performance for each of the measurement points is reported here in terms of the min, max, mean, standard deviation and 90th percentile. Additionally, the measured times are illustrated in a staple diagram for each edge cloud, showing the time spent in each relocation phase. The service downtime and total migration time are also presented in staple diagrams.

The following figures represent 50 repeated performance test runs of the test scenario and setup. For the source edge cloud, Figure 5.5 shows each relocation phase and their respective measured times. It shows that the time spent in the corresponding relocation phases do vary but not as heavily as for the Openstack setup.

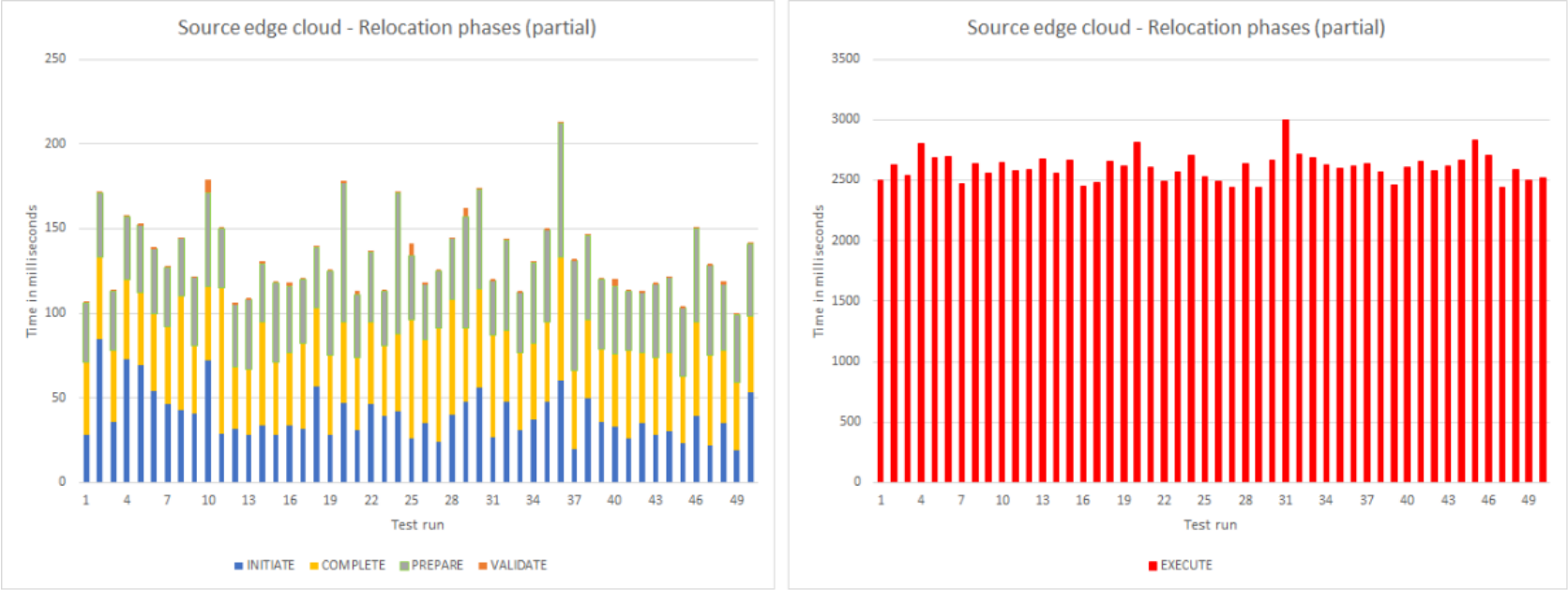

Figure 5.5: Test results of 50 runs for the source edge cloud. The left-hand side figure shows the INITIATE, COMPLETE, PREPARE and VALIDATE relocation phases time taken in milliseconds, while the right-hand side figure only shows the EXECUTE phase time taken in milliseconds.

The 50 test runs for the target edge cloud are shown in Figure 5.6. The figure shows how much time the target edge cloud spent in each relocation phase when invoked by the source edge cloud. The times for each phase have minimal variation and one outlier.

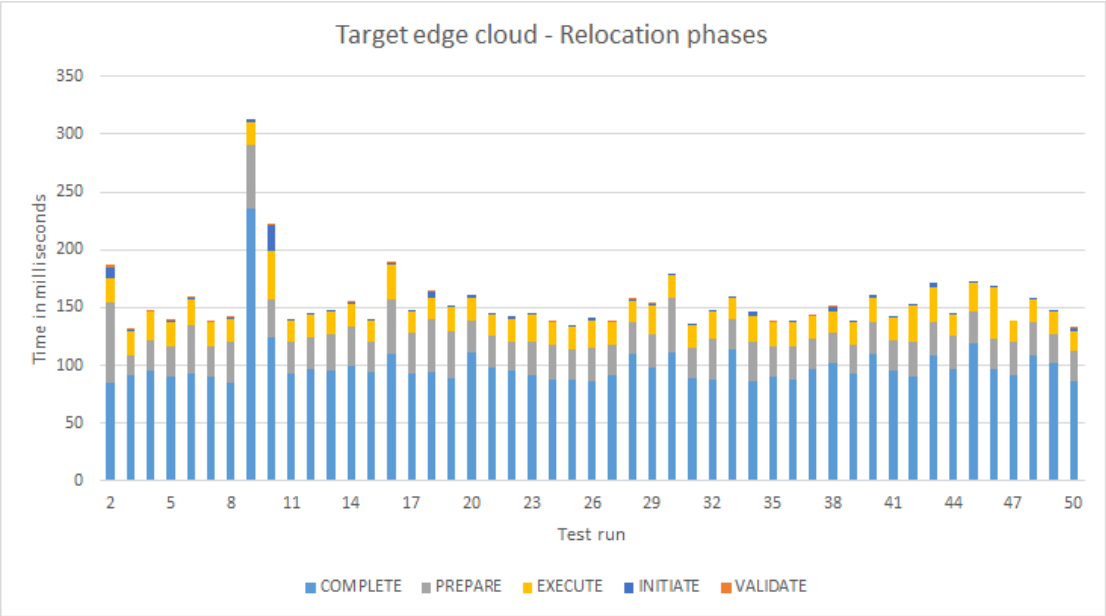

Figure 5.6: Test results of the corresponding 50 runs matching Figure 5.4. This shows every relocation phase the target edge cloud spent time in.

In Figure 5.7, the total migration time for the SMO is presented. It shows that for 50 test runs, the total migration time also has minimal variation, being relatively stable.

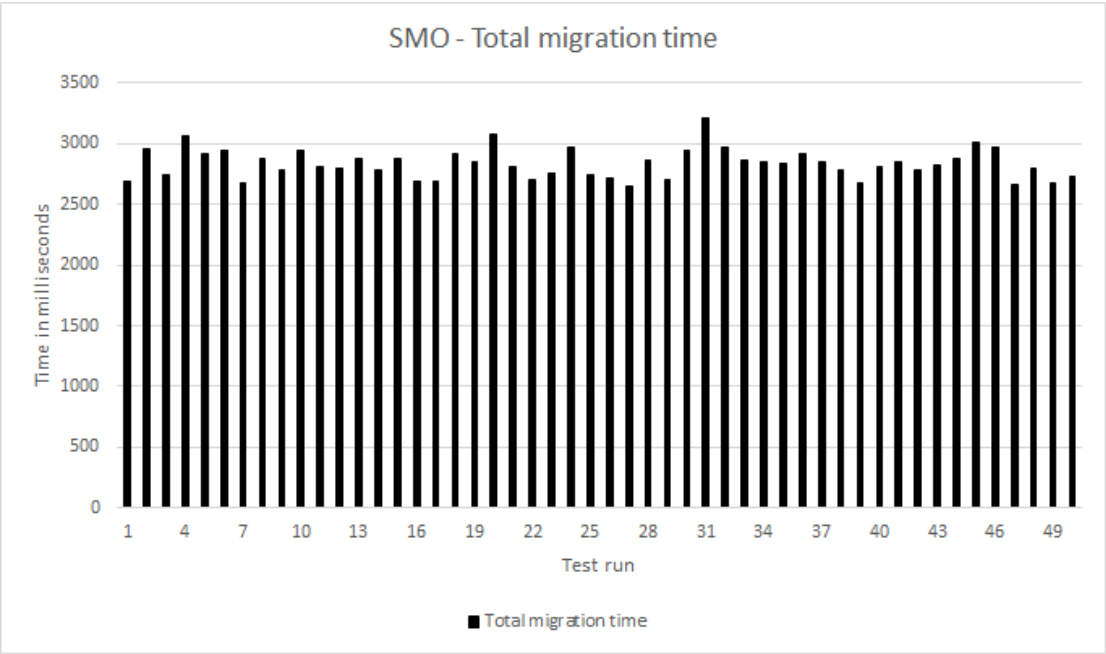

Figure 5.7: The test results for 50 runs. This shows the total migration time of the SMO.

Lastly, Figure 5.8 shows the service downtime of the migrated NGINX container. The results show the time it took for the UE to start receiving packets from the container again, after it stopped sending packets due to LXD.

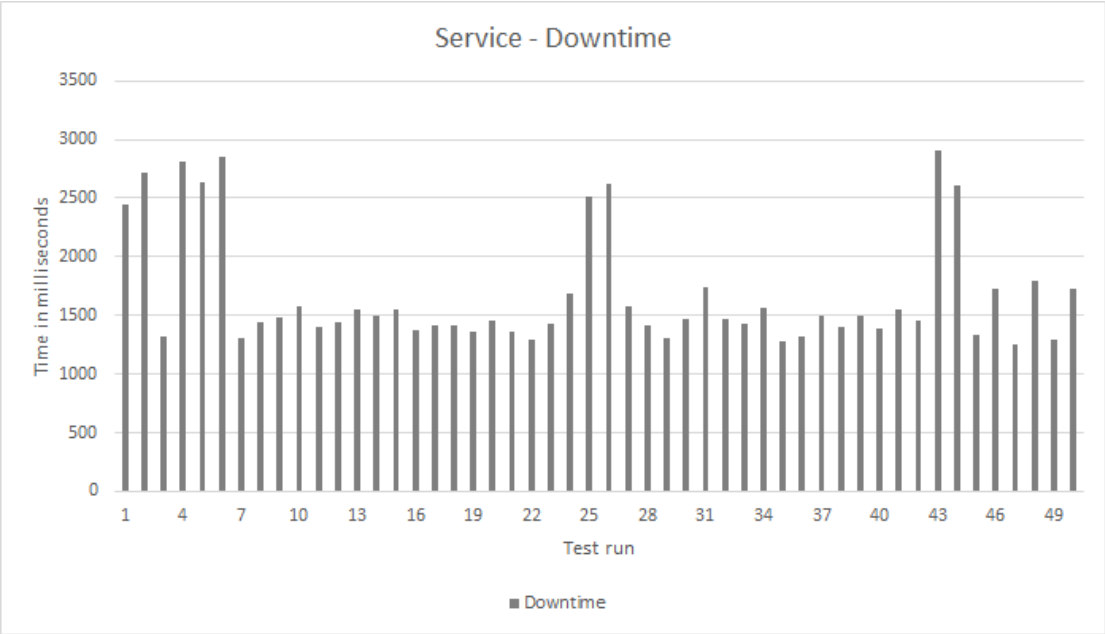

Figure 5.8: The test results for 50 runs. This shows the service downtime, e.g. the moment the UE stopped receiving packets until it started receiving packets again.

The analysis, using a statistical approach, is presented below. The analysis targets the total migration time and service downtime to explain the contributing parts for the achieved values for the total migration time and service downtime. Table 5.4 and Table 5.5 show that each relocation phase, excluding the EXECUTE phase, has a relatively small impact on the total migration time. Additionally, the EXECUTE phase for the source edge cloud affects the SMO the most, with a mean time of 2607 ms. Additional analysis in the amount of time for the EXECUTE phase show that this phase is similar to the previous test setup where it is heavily dependent on the underlying relocation type. This verifies the suspicions that LXD and CRIU might have some optimization issues. In addition, the SMO only utilized one relocation method, LXD, while other relocation methods may suffer from the same problem.

Table 5.4: Statistical values for the source edge clouds different relocation phases.

Measurement point/Statistic	**MIN (ms)**	**MAX (ms)**	**MEAN (ms)**	**STDEV (ms)**	**90th percentile (ms)**
INITIATE	19,00	85,00	39,66	14,60	57,30
VALIDATE	1,00	8,00	1,48	1,45	2,00
PREPARE	32,00	83,00	44,18	12,61	59,60
EXECUTE	2438,00	3001,00	2607,90	111,09	2707,70
COMPLETE	36,00	86,00	49,10	10,10	67,00

The VALIDATE phase has the least impact on the total migration time while the PREPARE, EXECUTE and COMPLETE phases has the most impact. This is as expected from the Openstack test setup results, due to OF related tasks.

Table 5.5: Statistical values for the target edge clouds different relocation phases.

Measurement point/Statistic	MIN (ms)	MAX (ms)	MEAN (ms)	STDEV (ms)	90th percentile (ms)
INITIATE	< 1	22,00	1,92	3,19	3,00
VALIDATE	< 1	3,00	< 1	< 1	1,00
PREPARE	18,00	69,00	31,30	8,82	42,40
EXECUTE	18,00	44,00	21,74	5,34	25,50
COMPLETE	85,00	236,00	99,32	21,85	111,00

In Table 5.6, the statistical results for the total migration time and service downtime are presented. The EXECUTE phase and total migration time always correlate because of CRIU being the most time-consuming part of the call flow, also seen by the mean time for the EXECUTE phase being 2607 ms and 2835 ms for the total migration time. The VALIDATE phase takes less than 1 ms to complete, which shows that the non-OF related phase is not contributing much to the total migration time.

Table 5.6: Statistical values for the total migration time and service downtime.

Measurement point/Statistic	MIN (ms)	MAX (ms)	MEAN (ms)	STDEV (ms)	90th percentile (ms)
Total migration time	2649,00	3217,00	2835,26	122,31	2966,50
Downtime	1255,79	2909,97	1679,18	493,16	2622,81

The service downtime, as mentioned in the previous test setup, is the most critical part of the entire chain in the SMO and, as presented in Table 5.6, the mean downtime was 1679 ms. Additionally, since the COMPLETE phase in the target edge cloud installs the OF rules for the new traffic path, creating the new traffic flow, it takes 99 ms on average to complete. This means that this phase can at most add 99 ms to the service downtime, on average.

All the results presented in this section verify that the environment in which LXD and CRIU perform live migration, has the most impact on the total migration and service downtime. The mean total migration time for the Openstack test setup was 11767 ms compared to the 2835 ms of the performance setup, similarly, the service downtime was 7199 ms compared to 1679 ms. With these values, combined with every test performed, it is evident that CRIU has issues with its iterative dumping strategy and transferring the dumps. When run on a 1 core machine it is by default forced to use the only available core and performs well. However, when there are 2 available cores, there seems to be inconsistencies regarding how CRIU utilizes them and how

the cores schedule the tasks for CRIU. This is ultimately determined to be a bug in CRIU.

The lowest service downtime that was achieved was 1255 ms which is comparable to the downtime achieved in the related work [39]. It also means that container migration is viable for TCP connections where the client application uses a 1255 ms buffer or more to compensate for the downtime.

Lastly, the test setup for the performance measurements is not bound to only a physical machine with two VMs setup. The setup can also be distributed across different physical machines and at a minimum with a 100 Mbit network link between them. This would yield similar results as presented above.

5.5 LXD/CRIU limitations

Apart from the limitations with LXD and CRIU mentioned in section 5.3.1, this section mentions the issues related to the non-blocking problems with the technology in LXD and CRIU.

The test performed in this chapter showed a few limitations that LXD and CRIU infer. Three of the most critical limitations are:

- **No Ipv6 TCP live migration support:** As UEs and services can have Ipv6, Ipv4 or both types of addresses, the SMO and relocation method should support these addressing schemes. The aforementioned test setup and scenario performed Ipv4 testing only solemnly because CRIU does not fully support Ipv6 TCP socket migration yet. This means that, while the SMO may handle both Ipv6 and Ipv4 migration, the service module LXD does not.

- **Unstable live migration:** Both LXD and CRIU are updated daily, fixing bugs and improving performance. This means that the tools are not perfect yet. This has also been observed in the performed tests, where runs either fail to live migrate, fail to resume the TCP session or induce inconsistent service downtimes. Additionally, a special case was observed where the container does not handle receiving high throughput of data well, in live migration. However, it does handle sending high throughput of data, in live migration.

- **Environment:** A crucial limitation is the environment in which LXD and CRIU runs and two observations are seen regarding that. The first observation is that if LXD with CRIU runs in a machine that has more than 1 CPU core, the performance of live migration becomes significantly lower. The second observation is the migration network link that is used for the live migration. If the network link it not a direct link between the two edge clouds, the dump transfer takes more time to complete.

6 Conclusions

This thesis presented parts of the 5GC network and Service Mobility, where the latter refers to performing migration of services between edge clouds. With this insight, seamless migration of a service from one edge cloud to another, with the assistance of the 5GC, can be achieved. To this end, a SMO was designed and implemented to be part of the edge clouds, utilizing standards of 5G to perform i.e. messaging and communicate with NFs. The SMO provides edge clouds with the ability to bring services close to the UE, reduce the latency and utilize a more preferred traffic path between UE and service even after mobility events. The framework architecture and design of the SMO has been discussed and an implementation of it have been tested and evaluated. The framework uses a set of standardized communication protocols and messaging models, based on 5G, combined with a RESTful structure. This allows for interoperability with the NFs in the 5GC as well as expansion and adoptability of the SMO by third parties.

The results achieved from the performed tests and evaluation, in an optimal setup, indicate that using the SMO yields low total migration time depending on the relocation method. It also shows that, using LXD as a service module for the relocation method, results in low service downtime, 1255,79 ms, with low or no service interruption of the migrated service, from the perspective of user experience. With the help of the results, the optimal setup consists of having a simplified and lightweight container that only contains the required packages and tools to run the service as well as running the SMO on a machine with 1 CPU core with a direct network link to the target edge cloud for migration. These factors ensured that the service downtime could be as minimal as possible when performing live migration of the container. In a non-optimal setup, there is high total migration and service downtime due to the underlying CRIU in LXD, that performs the dumping, transfer and restoration of the service. It is not optimized enough to provide ultra-reliability, meeting the uMTC use case of 5G. This does not imply that LXD and CRIU should not be used in combination with the 5GC or without it, it means that it is better to use this method for services not heavily reliant on being live or sensitive to delay. For example, if the video client uses a buffer of 1 second, which was used in our tests, the UE would not perceive any service interruption.

Additionally, the SMO can handle multiple use cases such as, more than one UE simultaneously due to the asynchronous support as well as falling back to a failure phase if something fails during any relocation phase, ensuring that the traffic flow is intact. It can also interact with a NF from the 5GC, the SMF. The results also show that the SMOs different relocation phases impose minor time delays, as the OF rule installation is the only time-consuming task. The implementation of the SMOs route handler, with OF rules, enabled the UE to always be served by the service at the moment of handover and during live migration, handling and managing the tromboned traffic.

The most suitable relocation method to use for the stated problem in section 1.2, is LXD. This is because it achieves IP mobility as well as preserving TCP sockets that are established. This means that the container that is migrated keeps its address when moved to another edge cloud and that the session towards the container is kept intact, maintaining a key aspect of service mobility. LXD is also built in to Linux already, making it a standard tool for many Linux distributions. The drawbacks of it, as mentioned in section 5.5, should be taken in to consideration when deciding to use LXD. If Ipv4 only UEs and services are sufficient for the deployer, then LXD is a good match.

Lastly, the results e.g. the SMO, LXD, CRIU and test runs, help us answer the questions in the problem statement by showing the performance and behavior of the framework and tools. They also provide insight in how optimization in said tools can yield either lower or higher migration or service downtime.

6.1 Future work

With the evaluation of the proposed SMO framework, the performance test setup uses one UE and one service. This in turn means that there was only one established TCP connection towards the RTMP streaming service. So, further testing of the SMO is required to achieve a broad view of the behavior and performance of the SMO. The additional testing may consist of trying different hosted services in containers, use a different service module and create a more optimized framework to handle many simultaneous users.

6.1.1 Alternative services

For alternative services, apart from NGINX streaming service, the following services are recommended to test:

- **File hosting service:** A future service to try is file hosting, this is because, in the streaming service, the container is sending the data payload while in file hosting, the user is usually uploading files e.g. the UE is sending the data payload to the container. The file hosting service also provides download cases, testing both scenarios. However, as mentioned in section 5.5, CRIU and LXD have issues with receiving high throughput.

- **Web server service:** The web service is mentioned because, in many cases, there is a web server for most services. A web server also utilizes multiple TCP connections to the server to divide the work load of different tasks of the server to different connections. For CRIU, this will mean that there is more than one TCP socket to migrate which should incur higher service downtime.

6.1.2 Different service modules

In this thesis, only LXD was used as a service module. Other service modules should be tested in place of LXD to provide a comparable alternative to container migration and perhaps improve the usability cases.

An alternative service module would be UE context migration, one example of this was presented in [35] using MIH and CXTP. This solution could be modified and integrated to utilize the SMO to perform service migration.

6.1.3 Design improvements

The SMO framework was designed with a goal in mind, to perform service migration, which led to not including additional components that makes up a complete system. The additional components, as seen in Figure 6.1, that are also improvements to the SMO are:

- **Local Break Out (LBO):** With LBO as an improvement, a decision can be made close to or at the UPF to send traffic either towards the closest edge cloud, to the internet or to another UPF. This means that, for example when a UE requires tromboning of traffic, it can be done early and not be left to the edge cloud to handle.

- **Network Address Translation 64 (NAT64):** In future cases, the UE may only use an Ipv6 address while services are Ipv4 only. To then utilize the SMO for live migration of TCP sockets, the UE needs to be able to connect to the service. This can be done using NAT64, which translates Ipv6 only addresses to Ipv4 addresses.

- **Domain Name System (DNS):** For most operators, a DNS is common to use for different services. This simplifies connections to the services and mitigates the use of IP addresses. Similarly, this could be used for service mobility by connecting to services with domain names. It also provides services to be deployed with the same IP addresses so that i.e. load balancing can be performed. Additionally, the infrastructure of the edge cloud will be transparent to the UE by hiding the IP addresses.

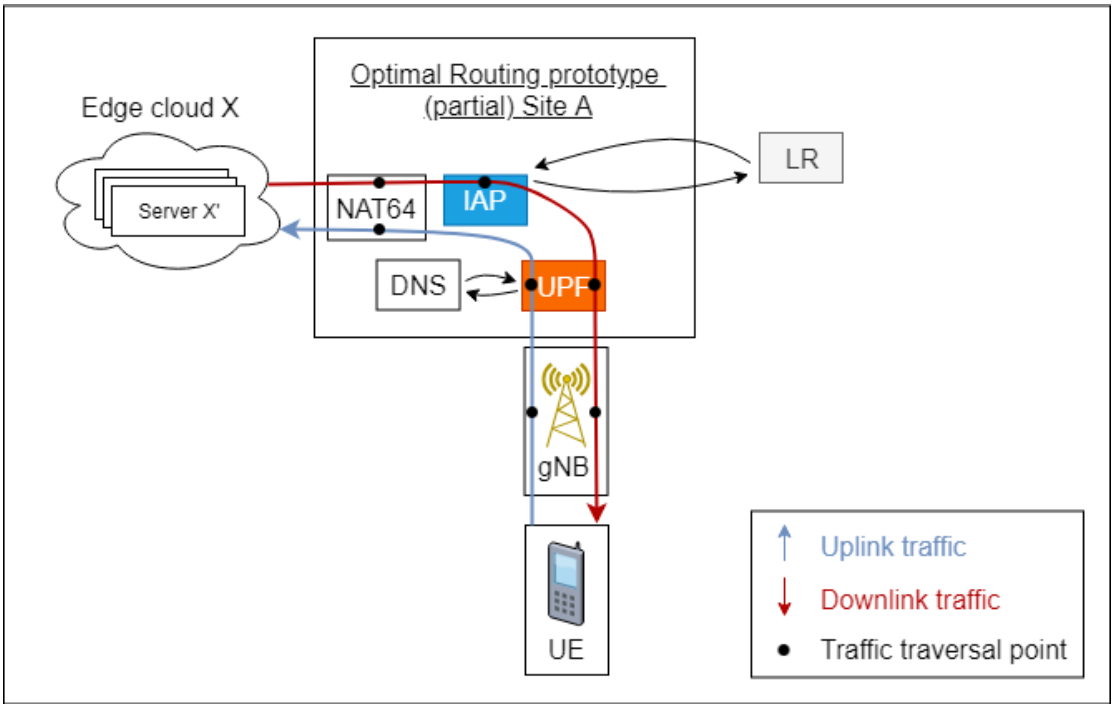

Figure 6.1: The potential placement of NAT64 and DNS in the Optimal Routing setup.

References

[1] Ericsson, "Ericsson Mobility Report," Fredrik Jejdling, Stockholm, 2018.

[2] Ericsson, "To be first in 5G, first get to the core," Ericsson AB, Stockholm, Sweden, 2018.

[3] 3rd Generation Partnership Project (3GPP), 2019. [Online]. Available: http://www.3gpp.org/release-15.

[4] A. Benjebbour, K. Kitao and H. Atarashi, "IMT-2020 Radio Interface Standardization Trends in ITU-R," *NTT DOCOMO Technical Journal,* 2018.

[5] S. E. El Ayoubi, M. Boldi, Ö. Bulakci, P. Spapis, M. Schellmann, P. Marsch, M. Säily, J. F. Monserrat, T. Rosowski, G. Zimmermann, I. Da Silva, M. Tesanovic, M. Shariat and A. M. Ibrahim, "5G RAN Architecture and Functional Design," 5G PPP METIS-II, 2016.

[6] 3GPP, "5G; System Architecture for the 5G System (3GPP TS 23.501 version 15.2.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[7] 3GPP, "5G; Procedures for the 5G System (3GPP TS 23.502 version 15.2.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[8] 3GPP, "5G; Policy and Charging Control Framework for the 5G System; Stage 2 (3GPP TS 23.503 version 15.2.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[9] P. Sapkale and U. Kolekar, "Mobility Management for 5G Mobile Networks," *International Journal of Computer Applications,* vol. 182, pp. 1-4, 2018.

[10] P. Vasconcelos, G. Freitas and T. Marques, "KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems," *International Journal of Multimedia and Image Processing,* vol. 6, 2016.

[11] P. S. V. Indukuri, "Performance comparison of Linux containers(LXC) and OpenVZ during live migration : An experiment," Blekinge Institute of Technology, Karlskrona, Sweden, 2016.

[12] P. Menage, "The Linux Kernel Archives," [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt. [Accessed 24 April 2019].

[13] "Linux Containers: What's LXD?," 2019. [Online]. Available: https://linuxcontainers.org/lxd/introduction/. [Accessed 24 April 2019].

[14] "Linux Containers: What's LXC?," 2019. [Online]. Available: https://linuxcontainers.org/lxc/introduction/. [Accessed 24 April 2019].

[15] "Docker: Enterprise Container Platform for High Velocity Innovation," 2019. [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 24 April 2019].

[16] D. Bernstein, "Containers and Cloud: From LXC to Docker to Kubernetes," *IEEE Cloud Computing,* vol. 1, no. 3, pp. 81-84, 2014.

[17] "OpenVz: Open source container-based virtualization for Linux," [Online]. Available: https://openvz.org/. [Accessed 24 April 2019].

[18] S. Outadi and J. Trchalikova, "Performance comparison of KVM and XEN for telecommunication services," Blekinge Institute of Technology, Karlskrona, Sweden, 2013.

[19] "CRIU," 2019. [Online]. Available: https://criu.org/Main_Page. [Accessed 24 April 2019].

[20] R. Stoyanov and M. J. Kollingbaum, "Efficient Live Migration of Linux Containers," 2018. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-02465-9_13. [Accessed 24 April 2019].

[21] "CRIU: TCP connection," 11 March 2019. [Online]. Available: https://criu.org/TCP_connection. [Accessed 24 April 2019].

[22] "DMTCP: Distributed MultiThreaded CheckPointing," 15 November 2017. [Online]. Available: http://dmtcp.sourceforge.net/index.html. [Accessed 24 April 2019].

[23] K. Arya, R. Garg, A. Y. Polyakov and G. Cooperman, "Design and Implementation for Checkpointing of Distributed Resources Using Process-Level Virtualization," in *IEEE International Conference on Cluster Computing (CLUSTER)*, Taipei, Taiwan, 2016.

[24] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang and Z. Shi, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*, New York, NY, USA, 2017.

[25] 3GPP, "5G; NG-RAN; Architecture description (3GPP TS 38.401 version 15.2.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[26] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin, K.-W. Wen, K. Kim, R. Arora, A. Odgers, L. M. Contreras and S. Scarpina, "MEC in 5G networks," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[27] 3GPP, "5G; 5G System; Technical Realization of Service Based Architecture; Stage 3 (3GPP TS 29.500 version 15.0.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipoli, France, 2018.

[28] 3GPP, "5G; 5G System; Principles and Guidelines for Services Definition; Stage 3 (3GPP TS 29.501 version 15.0.1 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[29] M. Belshe, R. Peon and M. Thomson, "IETF Tools: RFC 7540," May 2015. [Online]. Available: https://tools.ietf.org/pdf/rfc7540.pdf. [Accessed 24 April 2019].

[30] 3GPP, "5G; 5G System; Session Management Event Exposure Service; Stage 3 (3GPP TS 29.508 version 15.2.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2019.

[31] 3GPP, "5G; NG-RAN; Xn general aspects and principles (3GPP TS 38.420 version 15.0.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[32] 3GPP, "5G;NG-RAN; Xn Application Protocol (XnAP) (3GPP TS 38.423 version 15.1.0 Release 15)," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2018.

[33] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys & Tutorials,* vol. 19, no. 3, pp. 1657-1681, 2017.

[34] D. Roeland and Z. Fu, "Novel Core Network Architecture for 5G Based on Mobile Service Chaining," in *Mobile Networks and Management, MONAMI*, 2017.

[35] H. Assasa, "Service Mobility in Mobile Networks," Royal Institute of Technology, KTH, Stockholm, Sweden, 2014.

[36] "IEEE 802.21," [Online]. Available: http://www.ieee802.org/21/. [Accessed 24 April 2019].

[37] J. L. Ed, M. Nakhjiri, C. Perkins and R. Koodli, "IETF Tools: Context Transfer Protocol (CXTP)," July 2005. [Online]. Available: https://tools.ietf.org/html/rfc4067. [Accessed 24 April 2019].

[38] M. Bernaschi, F. Casadei and P. Tassotti, "SockMi: a solution for migrating TCP/IP connections," in *15th EUROMICRO International Conference on Parallel, Distributed and Network-Based Processing (PDP'07)*, Naples, Italy , 2007.

[39] R. A. Addad, D. Dutra, T. Taleb, M. Bagaa and H. Flinck, "Towards a Fast Service Migration in 5G," in *IEEE Conference on Standards for Communications and Networking (CSCN)*, 2018.

[40] A. Aissioui, A. Ksentini and . T. Taleb, "On Enabling 5G Automotive Systems Using Follow Me Edge-Cloud Concept," *IEEE Transactions on Vehicular Technology,* vol. 67, no. 6, pp. 5302-5316, 2018.

[41] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks," *IEEE Network,* vol. 27, no. 5, pp. 12-19, 2013.

[42] T. Taleb, P. Hasselmeyer and F. G. Mir, "Follow-Me Cloud: An OpenFlow-Based Implementation," in *IEEE International Conference on Green Computing and Communications and IEEE Internet of*

Things and IEEE Cyber, Physical and Social Computing, Beijing, China, 2013.

[43] I. Farris, T. Taleb, A. Iera and H. Flinck, "Providing ultra-short latency to user-centric 5G applications at the mobile network edge," *Transactions on Emerging Telecommunications Technologies,* 2017.

[44] 3GPP, "Mobile Edge Computing (MEC); End to End Mobility Aspects," 3rd Generation Partnership Project (3GPP), Sophia Antipolis, France, 2017.

Appendix A – Openstack test setup results

This appendix presents the results generated from 50 test runs in Openstack, with the combined 5GC and SMO test setup.

Figure A.1 shows the results for the different relocation phases in the source edge cloud.

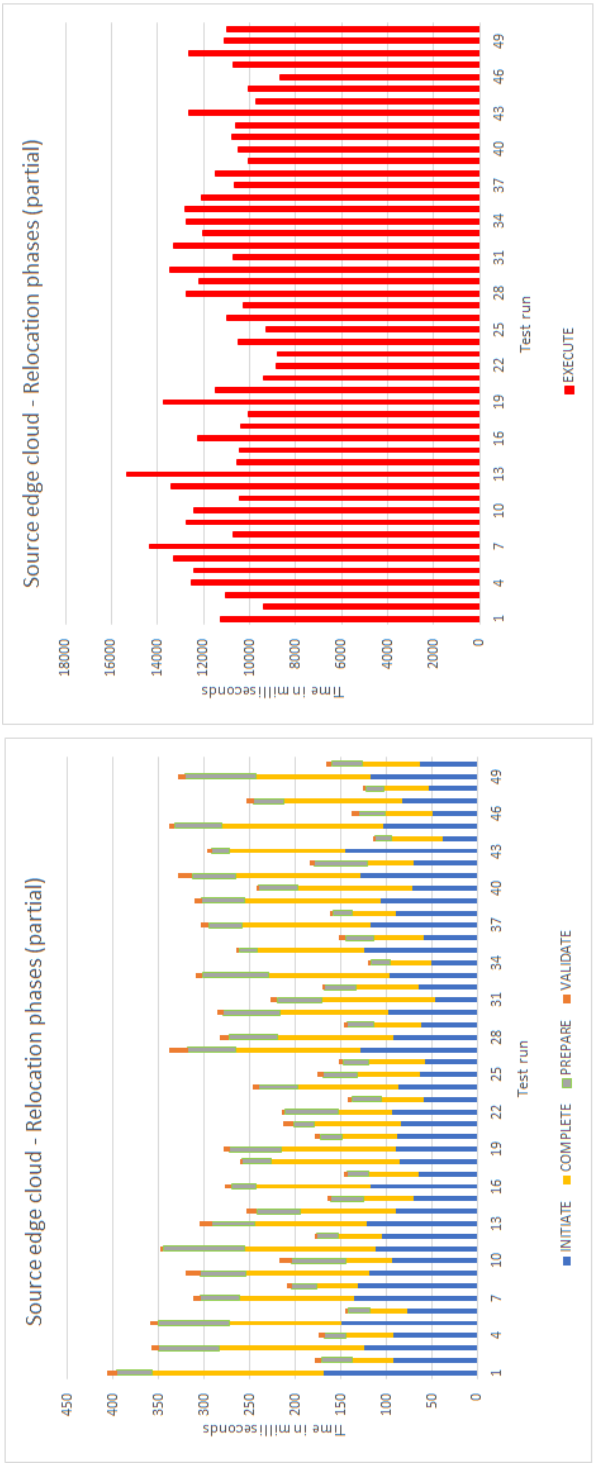

Figure A.1: Test results of 50 runs for the source edge cloud. The left-hand side figure shows the INITIATE, COMPLETE, PREPARE and VALIDATE relocation phases time taken in milliseconds, while the right-hand side figure only shows the EXECUTE phase time taken in milliseconds.

Figure A.2 shows the results for the different relocation phases in the target edge cloud.

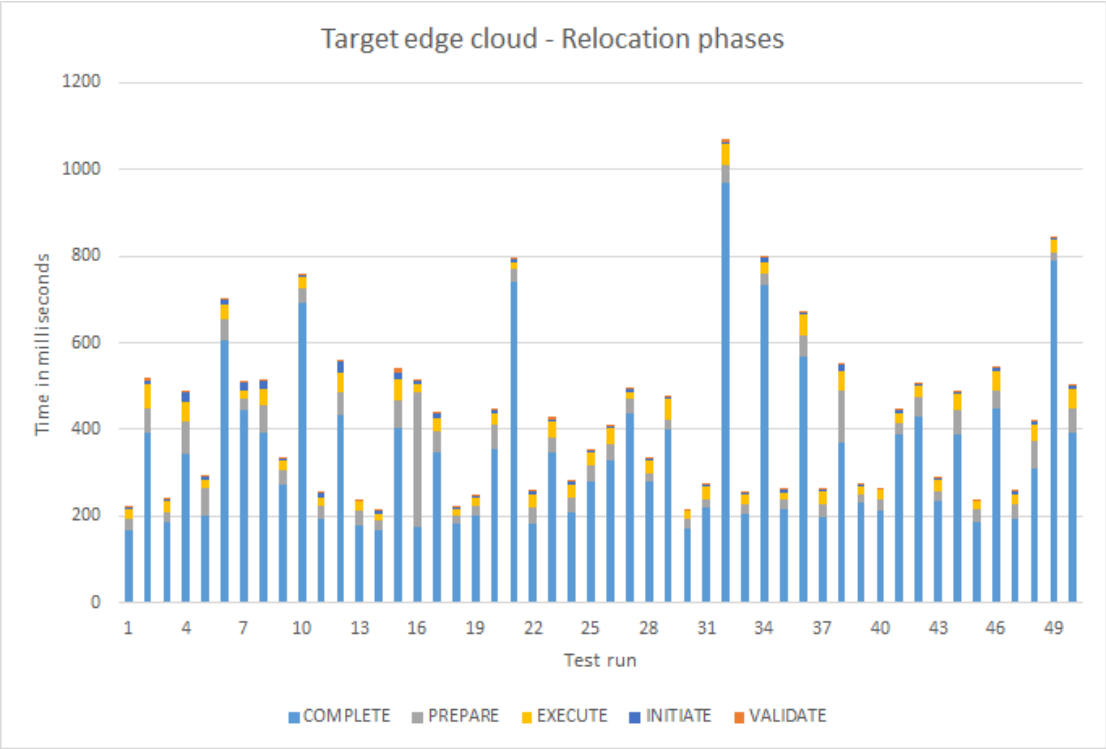

Figure A.2: Test results of the corresponding 50 runs matching Figure 5.4. This shows every relocation phase the target edge cloud spent time in.

Figure A.3 shows the results for the total migration time.

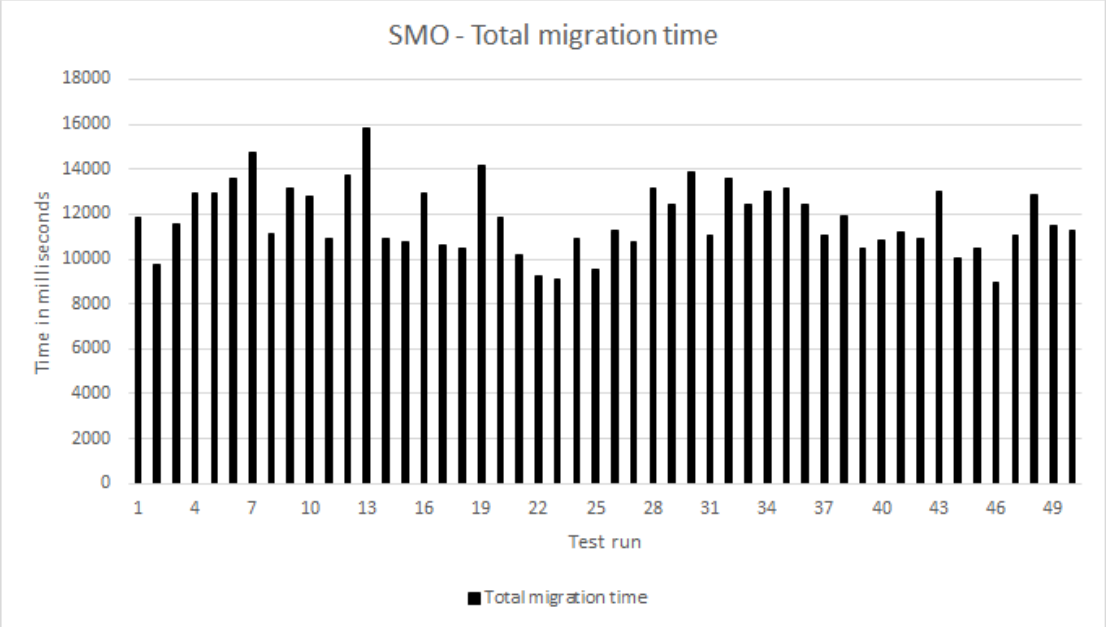

Figure A.3: The test results for 50 runs. This shows the total migration time of the SMO.

Figure A.4 shows the results for the service downtime.

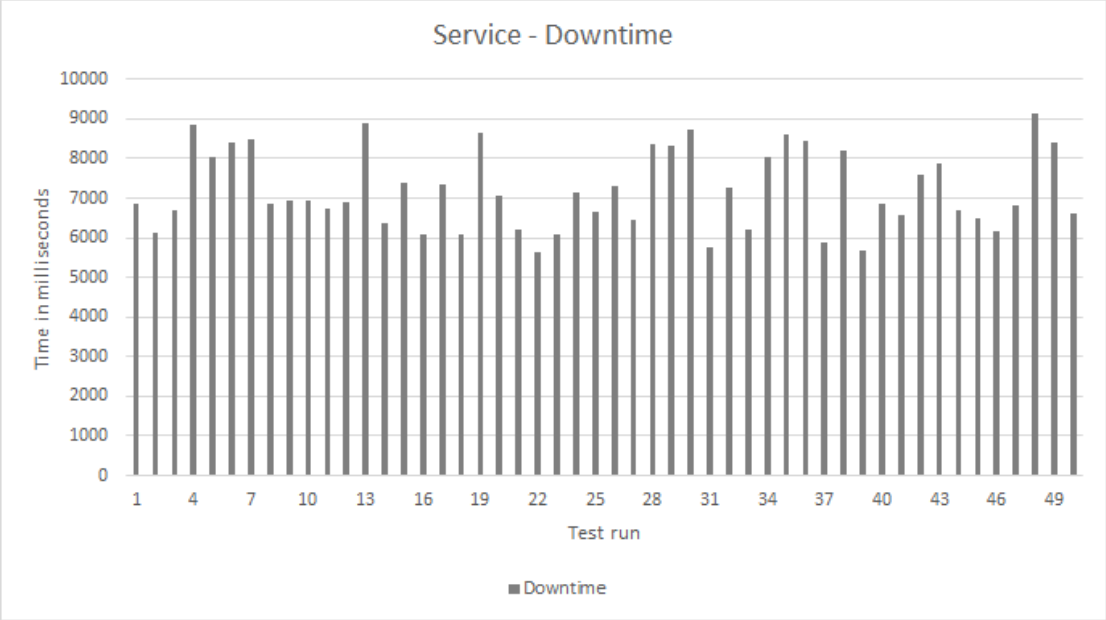

Figure A.4: The test results for 50 runs. This shows the service downtime, e.g. the moment the UE stopped receiving packets until it started receiving packets again

TRITA-EECS-EX-2019:271