Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Sondre Sørensen Kielland

Secure communication with WireGuard

VPN-as-a-Service in beyond 5G

Master's thesis in Communication Technology
Supervisor: Katina Kralevska
Co-supervisor: Danilo Gligoroski, Ali Esmaeily
February 2022

NTNU
Norwegian University of Science and Technology

Sondre Sørensen Kielland

Secure communication with WireGuard

VPN-as-a-Service in beyond 5G

Master's thesis in Communication Technology
Supervisor: Katina Kralevska
Co-supervisor: Danilo Gligoroski, Ali Esmaeily
February 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Title:	Secure Communication with WireGuard – VPN-as-a-Service in Beyond 5G
Student:	Sondre Sørensen Kielland

Problem description:

In a 5G environment, a number of verticals can be connected together in shared physical infrastructure on different locations. Network slicing is used in 5G networks as a way to separate verticals. A network slice consists of one or more network services. To ensure security of the exchanged traffic between network services or network functions inside network services VPN can be a solution. The thesis will study this through the following research questions.

How can WireGuard be used in a 5G virtualized environment to guarantee slice isolation? How does WireGuard deployed in a virtualized environment satisfy performance requirements in terms of security and latency? How can OSM support WireGuard as a VPN-as-a-Service?

Date approved:	2021-10-13
Supervisor:	Assoc. Prof. Katina Kralevska, IIK
Cosupervisor:	Prof. Danilo Gligoroski and Ali Esmaeily, IIK

Abstract

The introduction of network slicing in 5G networks has allowed verticals to deploy their services alongside other applications. Compared to the earlier cellular generations, the goal is for 5G to deliver various services simultaneously. Two key enabling technologies are Software-Defined Networking (SDN) and Network Function Virtualization (NFV). SDN and NFV bring flexibility for deploying network functions and chaining them together.

Security is one of the main challenges for shared infrastructure like networks and computational resources of an NFV infrastructure. Virtual Private Network (VPN) tunneling is one countermeasure a tenant manager, controlling the network service, can use to separate network slices, services, and functions when using shared environments in a service function chaining context.

In this thesis, we have studied how WireGuard can provide an encrypted VPN tunnel as a service between network functions. We used Open Source MANO (OSM) to deploy and orchestrate the network functions into network services and slices. We have created multiple scenarios simulating a real-life cellular network during our development process. WireGuard was then used as a VPN-as-a-Service between the different network functions to secure and isolate the interfaces. In our work, we have implemented a Proof-of-Concept where the WireGuard tunnel is configured automatically using the inbuilt Juju controller of OSM to minimize manual steps.

After deploying our network services, we have then performed measurements to observe the performance of WireGuard. The performance measurements show between 0.8 Gbps and 2.5 Gbps throughput and under 1 ms delay between network functions using WireGuard. These measurements are within several key performance indicators of 5G, making WireGuard suited to be used to provide security in slice isolation in 5G and beyond networks.

Sammendrag

Innføring av skivedeling i 5G nettverk åpner for at vertikale virksomheter kan utvikle og benytte sine tjenester ved siden av hverandre i telenett. Sammenliknet med tidligere generasjons mobilnettverk, søker 5G å kunne tilby flere tjenester simultant. Programvaredefinerte Nettverk (SDN) og Virtualiserte Nettverksfunksjoner (NFV) er, ved å innføre nødvendig fleksibilitet til å deployere nettverksfunksjoner og koble de sammen i ønsket rekkefølge, to hovedteknologier for å kunne lage kjeder av tjenestefunksjoner dynamisk med felles infrastruktur.

Sikkerhet er en av utfordringene med å ha en delt nettverks- og prosesseringsinfrastruktur. Tunnelering gjennom Virtuelle Private Nettverk (VPN) er et sikkerhetstiltak som kan iverksettes av den skiveansvarlige for å skille nettverksskiver, tjenester og funksjoner fra hverandre.

Vi har studert hvordan WireGuard kan brukes som VPN tilbyder for å skape en kryptert tunnel som en tjeneste mellom nettverksfunksjoner. Open Source MANO (OSM) er brukt som programvare for å kombinere og orkestrere nettverksfunksjoner til nettverkstjenester og nettverkskiver. For å simulere reelle mobilnettverk har vi designet og testet flere scenarioer. WireGuard har blitt lagt til som VPN som en tjeneste i de ulike situasjonene for å isolere og sikre trafikken mellom nettverksfunksjoner. Gjennom utvkling i oppgaven har vi kommet opp med et konsept for å implementere en WireGuard som en tjeneste der vi bruker minimalt med manuelle steg. For å redusere manuelle steg i oppsett og konfigurasjon av WireGuard har vi brukt den innebygde Juju kontrolleren i OSM.

Etter å ha deployert nettverkstjenestene vi har utviklet til en delt infrastruktur har vi gjort målinger for å studere hvordan WireGuard påvirker ytelsen. Vi har målt en gjennomstrømning på mellom 0.8 Gbps og 2.5 Gbps og forsinkelse på under 1 ms ved bruk av WireGuard-tunneler mellom nettverksfunksjoner. Målingene vi har gjort ligger innenfor flere av nøkkeltallsindikatorene for 5G nettverk. Det gjør WireGuard passende for å brukes til å tilføre sikkerhet for isolering av nettverksskiver i utviklingen av 5G nettverk og kommende teknologier.

Preface

This project has been done as a master thesis in Communication Technology as part of the Digital Infrastructure and Cyber Security (MSTCNNS) program at the Norwegian University of Science and Technology (NTNU). The project has been carried out by Sondre Kielland and supervisors Assoc. Prof. Katina Kralevska, Prof. Danilo Gligoroski, and Ali Esmaeily have carried out the project. I want to thank my supervisors for all their excellent guidance and insight during the work with the thesis. Their valuable inputs have improved the practical work and written report. I will also thank my family, friends and colleagues that have shown interest and support during this work. All referenced code and descriptors throughout the thesis are made available on Github.[1]

[1] https://github.com/sondrki/TTM4905

Contents

List of Figures

List of Tables

Listings

List of Acronyms

5GC 5G Core.

5G NR 5G New Radio.

5GS 5G System.

5QI 5G Quality of Service (QoS) Identifier.

AAA Authentication, Authorization and Accounting.

AKA Authentication and Key Agreement.

AMF Access and Mobility Management Function.

API Application Programming Interface.

AS Access Stratum.

AUSF Authentication Server Function.

CI/CD Continuous Integration/Continuous Deployment.

CIA Confidentiality, Integrity and Availability.

CNF Container Network Function.

CUPS Control and User Plane Separation.

DDoS Distributed Denial of Service (DoS).

DHCP Dynamic Host Configuration Protocol.

DoS Denial of Service.

E-UTRAN Evolved-Universal Terrestrial Radio Access Network (RAN).

E2E End-to-End.

eMBB enhanced Mobile BroadBand.

eNB eNodeB.

EPC Evolved Packet Core.

EPS Evolved Packet System.

ETSI European Telecommunications Standards Institute.

gNB gNodeB.

GTP GPRS Tunnelling Protocol.

GUI Graphical User Interface.

HSS Home Subscriber Server.

IaaS Infrastructure-as-a-Service.

IMS IP Multimedia System.

IMSI International Mobile Subscriber Identity.

IoT Internet of Things.

K8s Kubernetes.

KMS Key Management System.

KNF Kubernetes-based Network Function.

KPI Key Performance Indicator.

LCM Life-Cycle Management.

LTE Long Term Evolution.

MANO Management and Orchestration.

MCC Mobile Country Code.

MEC Multi-Access Edge Computing.

MME Mobility Management Entity.

mMTC Massive Machine-type Communication.

MNC Mobile Network Code.

MNO Mobile Network Operator.

MSIN Mobile Subscription Identification Number.

MSISDN Mobile Station International Subscriber Directory Number.

MTU Maximum Transmission Unit.

NAS Non-Access Stratum.

NEF Network Exposure Function.

NF Network Function.

NFV Network Function Virtualisation.

NFVI Network Function Virtualisation (NFV) Infrastructure.

NRF Network Repository Function.

NS Network Service.

NSA 5G Non-Standalone.

NSD Network Service (NS) Descriptor.

NSI Network Slice Instance.

NSSAAF Network Slice Specific Authentication and Authorization Function.

NSSF Network Slice Selection Function.

NST Network Slice Template.

OAI Open Air Interface.

ONAP Open Network Automation Platform.

OS Operating System.

OSM Open Source Management and Orchestration (MANO).

OTT Over the Top.

PCF Policy Control Function.

PDB Packet Delay Budget.

PDN Packet Data Network.

PGW Packet Data Network (PDN) Gateway.

PLMN Public Land Mobile Network.

PNF Physical Network Function.

PoC Proof of Concept.

QoS Quality of Service.

RAM Random Access Memory.

RAN Radio Access Network.

RAT Radio Access Technology.

RBAC Role-based Access Control.

SA 5G Standalone.

SAPD Service Access Point Descriptor.

SBA Service Based Architecture.

SBI Service Based Interface.

SCP Service Communication Proxy.

SDN Software-Defined Networking.

SDR Software-Defined Radio.

SEPP Security Edge Protection Proxy.

SF Service Function.

SFC Service Function (SF) Chaining.

SGW Serving Gateway.

SMF Session Management Function.

SOL Solution.

SPGW-C Service Packet Gateway-Control plane.

SPGW-U Service Packet Gateway-User plane.

SRT Service Response Time.

SSH Secure Shell.

TA Tracking Area.

TAC Tracking Area Code.

UDM Unified Data Management.

UE User Equipment.

UICC Universal Integrated Circuit Card.

UPF User Plane Function.

URLLC Ultra Reliable Low Latency Communication.

V2X Vehicle-to-Everything.

VCA VNF Configuration and Abstraction.

vCPU virtual Central Processing Unit.

VDU Virtual Deployment Unit.

VIM Virtual Infrastructure Manager.

VLAN virtual Local Area Network.

VM Virtual Machine.

VNF Virtual Network Function.

VNFD Virtual Network Function (VNF) Descriptor.

VNFM VNF Manager.

VPN Virtual Private Network.

VPNaaS VPN-as-a-Service.

YANG Yet Another Next Generation.

Chapter 1
Introduction

1.1 Background and Motivation

The enrollment of 5G cellular networks has already been happening for some time. Yet there are many of the foreseen features like Vehicle-to-Everything (V2X), 5G Standalone (SA), and tactile Internet that are under development [3GPa, 3GPb]. Like the rolling releases of the previous generations, we can expect that the functionality of 5G networks will continue to increase. Introduction to 5G New Radio (5G NR) has created media and regular users' attention. Likewise, communication between vehicles, V2X, is likely to get attention from regular consumers, media, and legislative authorities. A more hidden development for end-users is the work on the 5G Core (5GC). The 5GC is an important part of the 5G System (5GS) in order to fulfill its Key Performance Indicators (KPIs) [GP]. Another ongoing work on the core network is the evolution from 5G Non-Standalone (NSA) toward a SA core network. The biggest difference between the two is that SA is independent of support from the previous generation Evolved Packet Core (EPC). The separation from the EPC will likely be an important improvement to introduce new features to 5G networks.

The introduction of the 5GS using commercial products and new technology intends to help verticals deploy applications in an agile way. To have multiple verticals running on a shared infrastructure, End-to-End (E2E) services and network slice isolation are key factors. NFV, Software-Defined Networking (SDN) and Multi-Access Edge Computing (MEC) are technologies introduced to enable flexible 5G networks in providing services with diverse QoS requirements [BFG+17].

With NFV, the network operator should be capable to deploy and manage applications in a flexible way. To do this a NFV MANO is used to connect to a hypervisor running on a NFV Infrastructure (NFVI) and Virtual Infrastructure Manager (VIM). In that way, the network operator is able to deploy Virtual Machines (VMs) to the VIM. In the 5G network context the terminology for VMs with their functionalities are VNFs. The NFV MANO is in other words responsible for handling the setup and

management of different applications combined in VNFs, NSs and Network Slice Instances (NSIs) running on one or multiple VIMs [GOLH+20].

Another important aspect with the MANO and VIM relation is the network setup. The MANO should be able to set up the network between VNFs. The networking can be either internal within a VNF or between VNFs on the same or different VIMs. As the setup of applications should be rapid, there is a need to make the networking part flexible. SDN introduces this flexibility for creating and configuring virtual networks and forwarding their traffic flow [KGFG19, FKGG19].

Compared to the earlier cellular networks, 5G is more service-oriented in the way of enabling deployments of Over the Top (OTT) services in the cellular network. The three main use cases in 5G are categorized as Ultra Reliable Low Latency Communication (URLLC), enhanced Mobile BroadBand (eMBB), and Massive Machine-type Communication (mMTC) [ETSb, ETSd]. MEC brings computational power closer to the end-users supporting functionality for the different use case categories. By utilizing distributed computational power, User Equipments (UEs) can get a quicker service response. Simultaneously, moving more data to the edges can help offload bandwidth further into the core network. Deploying functionality closer to the user may require additional network infrastructure or computational resources not owned by the Mobile Network Operator (MNO). Computation in a cloud or using hypervisors that are rented or shared and transport data through shared networks may therefore be a reality for verticals to deliver E2E services. Using shared environments introduces new security challenges. One of the challenges is how a vertical makes the application data secured between Network Functions (NFs).

To summarize, the combination of NFV, SDN, MEC, and other new technologies enable verticals to deploy services in 5G networks. With a service-oriented future, the services should be E2E and handled by a MANO. Besides, different levels of physical hardware control introduce security challenges, such as securing application data transferred over networks.

1.2 Problem Description

The introduction of new functionality and design in 5G networks have presented the possibility for verticals to deploy their services alongside or OTT of traditional cellular network applications. Separating NSs employing a shared environment can be done using several countermeasures. For example, deploying a Virtual Private Networks (VPNs) between NFs can be one solution to secure data transportation. However, adding additional overhead is potentially problematic for near-real-time applications with the necessity of low latency. The same is valid for applications that require high throughput.

This thesis will look into how a NFV MANO can deploy real-life or close to real-life applications in a NS and secure the network traffic between VNFs of the NS. Furthermore, the VPN tunneling in a NS should contribute in isolating network slices apart. The VPN setup should secure the confidentiality of network data without adding significant overhead related to URLLC or eMBB applications in order to fulfill their QoS requirements.

1.3 Research Scope

Open Source MANO (OSM) and Open Network Automation Platform (ONAP) are two of many NFV MANO systems which are built to support flexibility of deployment in a 5G context [YYSCP20]. Both ONAP and OSM are getting attraction from large cellular network providers and are in many ways comparable in functionality, performance, and resource usage. In this thesis we will use OSM as our NFV MANO.

In OSM, network slices are defined by combining NSs together into templates. The primary focus of the thesis will be on building and testing the relevant NS and the relationship between VNFs. However, we will also demonstrate creating network slices to examine how this affects the performance of VPNs in a NS.

OSM supports different types of VIMs. We have limited ourselves to using MicroStack because of available resources. MicroStack has some limitations that will affect monitoring of our VNFs and NSs. Monitoring the performance when producing test traffic will therefore require some manual steps. Except for manual performance monitoring, all other Life-Cycle Management (LCM) should be within the OSM framework.

The NSs and NSI should include an application simulating actual cellular network functionality. We will use Open Air Interface (OAI) to introduce such capability. Further, we will focus on the inbuilt functionality of OSM by, as long as possible, avoiding the introduction of additional third-party applications that are not created in a NS deployment. An exception is introducing components to create realistic test data into the network. However, the thesis will not consider additional external applications such as centralized key management systems. We have made this choice because of time concerns and decreased additional complexity to an already complex environment. Also, we will not look at the scalability of NFs feature in OSM in this thesis because of the increased complexity in the OAI configuration setup it will entail.

1.4 Contribution

Comparisons between VPN software packages are documented both directly from VPN providers and in scientific work [TT19, Hag20, Don17]. Creating new VMs and setting up VPN tunnels between them can be classified as mainstream work for IT personnel. In a 5G context, additional complexity is, however, added with MANO software and dynamic deployments of NSs. The MANO helps the verticals to deploy and control a potentially vast amount of NFs across sites. One of the goals for using a MANO is to make the verticals deploy their services faster. Several examples already exist on how OSM, the MANO solution in this thesis, can deploy VNFs into NSs and NSIs. However, how a VPN can be deployed in this context to improve the security of slice isolation has not been well investigated.

This thesis studies how WireGuard as the VPN application can be deployed in a 5G environment. OSM is used to orchestrate VNFs with WireGuard into NSs and NSIs, and to establish VPN tunnels between them.

The thesis also contributes to the OSM community by extending the work done by Haga in [Hag20] and Dreibholz in [Dre20] in making a Proof of Concept (PoC) of a WireGuard-OSM lifecycle in the Evolved Packet System (EPS).

We make our code and descriptors publicly available to the research community.[1]

Finally, the main contributions of the thesis are summarized in the academic paper "Providing Network Slice Isolation with WireGuard in Beyond 5G" to be submitted to 4th International Workshop on Cyber-Security in Software-defined and Virtualized Infrastructures (SecSoft), co-hosted at 8th IEEE International Conference on Network Softwarization (NetSoft2022). A draft of the paper is given in Appendix G.

1.5 Hypothesis Statement

We think OSM should be able to support our operations by deploying both WireGuard and authentic functionality for VNFs. We also work with the hypothesis that adding WireGuard tunnels in our network will not reduce performance considerably regarding the use cases related to eMBB and URLLC applications.

1.6 Research Questions

1. How can WireGuard be used in a 5G virtualized environment to guarantee slice isolation?

[1] https://github.com/sondrki/TTM4905

2. How does WireGuard deployed in a virtualized environment satisfy performance requirements in terms of security, throughput, and latency?

3. How can OSM support WireGuard as a VPN-as-a-service?

The numbering is for reference purposes only and does not prioritize the research questions.

1.7 Tools and Resources

The following subsections introduce different software packages we have used in developing and testing use-cases to answer our research questions.

1.7.1 OSM

OSM is a MANO software solution from the European Telecommunications Standards Institute (ETSI) for orchestrating E2E NSs. The aim of the software is to simplify operational-ready NS deployments and its LCM. This is done by helping developers creating operations that can be run in the whole lifecycle of NSs and its NFs. In the OSM community Day-0, Day-1 and Day-2 operations are used as terms for required actions regarding LCM. Day-0 describes the initial configuration, for instance setting user names and passwords. Further, Day-1 operations concerns actions such as installation and configuration of applications. Lastly, Day-2 operations refer to maintenance and other runtime operations [OSMk]. Instantiation of NSs is performed via a number of files describing the different VNFs and NS and eventually a NSI. The latest version of OSM is version 11 which was released in December 2021 [OSMl]. In the thesis OSM will be used to manage and orchestrate different NS and NSIs including 4G and WireGuard services.

1.7.2 WireGuard

WireGuard is a lightweight VPN software intending to be an easy configurable, fast alternative to other VPN software solutions [Don]. In the thesis, WireGuard will be used to set up tunnels between VNFs to secure data transport and provide isolation of NSIs. WireGuard is described further in section 3.1.6.

1.7.3 MicroStack

MicroStack is an Infrastructure-as-a-Service (IaaS) platform being a container-based downscaled implementation of OpenStack. In the thesis, we use MicroStack as VIM. In MicroStack our NSs are deployed as VMs for the VNFs, with virtual networks connecting them. To be able to do this, OSM will connect to MicroStack. MicroStack

itself installs on top of one of several Operating Systems (OSs). In our case, we will use Ubuntu 18.04 as the base OS for MicroStack.

1.7.4 Cloud-init

Cloud-init is a tool for doing basic configuration and managing of a VM hosting VNF during instantiation [Ltda]. Possible tasks include setting credentials and installing software using the VNFs' package manager. Cloud-init is included in OSM to provide day-0 operations.

1.7.5 Juju

To provide Day-1 and Day-2 operations in the thesis, we will use Juju. When describing Day-1 and Day-2 operations together, we will often use Day1-2 operation as a term. Juju is an integrated VNF Manager (VNFM) in OSM providing LCM of VNFs [Ltdb]. By using packaging scripts and metadata, Juju can perform user-specified actions to install, set up, and maintain applications in a VNF or Kubernetes-based Network Function (KNF). Operations in an NF using Juju can be either native, running directly in an NF, or by using a centralized proxy controller, a VNF Configuration and Abstraction (VCA). The latter requires connectivity between the controller and the NF while the first one can perform standalone. Further, Juju units can be related to each other. In this thesis, we will use proxy charms carried out by the inbuilt VCA in OSM. How Juju works in the context of OSM is further described in Section 3.1.5.

1.7.6 iPerf3

iPerf3 is a cross-platform tool to perform network performance tests [DEM+]. iPerf3 can be used to measure throughput and latency in multiple ways based on input parameters. We will use iPerf3 in the thesis to test how introducing WireGuard affects network performance.

1.7.7 OpenAirInterface

OAI is an Open Source project which aims to deliver 3GPP compliant cellular networks [All]. OAI provides code for 4G and 5G components, both for the core network and the RAN. In the thesis, we use OAI to provide a realistic cellular network service for testing.

1.8 Thesis Structure

The thesis is organized into seven chapters. Chapter 2 describes how we have worked to answer the research questions. The chapter also describes the lab environment and

the reasoning for the different tools. Chapter 3 gives an overview of relevant work and background knowledge of important aspects in the thesis. The results from the work are split into Chapter 4 for the development and implementation and Chapter 5 includes the results including performance measurements. The following Chapter 6 discusses our findings. Finally, Chapter 7 concludes the thesis.

Chapter 2
Methodology

The following chapter covers how we have carried out our work. We have split the chapter into three sections. The first section describes the methodology we have followed for implementation and testing. Next, we explain the tools we have chosen to use. Finally, we describe how we have set up our lab environment.

2.1 Work Process

Prototyping has been an essential method of working with the project. OSM and the supporting features like Juju and cloud-init consist of a vast amount of possibilities. Therefore, we have used practical implementations with incrementally high complexity to check that everything is working and learn the tools. With the different options, a combination of prototyping and understanding already given examples has been a way to increase the complexity of the NSs gradually. Therefore, prototyping has been the primary procedure to deploy applications for helping us answer the research questions.

The methodology we have used to study the research question is by using single-case mechanism experiments [Wie14]. We have chosen this overall methodology to build our knowledge simultaneously as we structure our observations. The research method of single-case mechanism experiments is described as observations when intervening with a phenomenon based on the architecture of the case. Single-case experiments can be done both as isolated tests in the lab, simulation of real-world scenarios, and studying real-world behavior in the field. Thus OAI being able to connect and function with real-world UEs is not intended to provide production-ready components. Therefore, this thesis will use a simulated real-world scenario for our single-case experiments. The validation of our data should include code that should be repeatable and reusable. With our lab setup, different factors can affect the deployment of NSs. We will therefore try to deploy the NSs with various resources and use the samples to reduce uncertainty in regards to measurements.

To get a better overview, we have broken down the research questions from Section 1.6 into practical tasks. Table 2.1 lists the tasks and their related research question(s). The numbering of research questions are in the order they appear in Section 1.6. Functionality, ease of deployment, and measurement to gather empirical data are parts that the tasks should cover.

Table 2.1: Practical tasks formed to solve our research questions.

Task description	Corresponding research question
Setup OSM and VIM in a lab environment.	3
Create a simple NS to ensure functionality of lab environment.	3
Deploy NSs for OAI and WireGuard with support of previous work.	3
Include WireGuard in the OAI NSs.	1 and 2
Define the lifecycle of WireGuard as a service	1
Produce traffic for testing the URLLC and eMBB KPIs of 5G. Measure both with and without WireGuard inline.	2

2.1.1 Testing

To test how WireGuard affects latency and throughput in a realistic environment, we will use a combination of data traffic going over the different protocols employed in a cellular network and using arbitrary data directly in the VPN tunnel. We believe high throughput cellular measurements are done most straightforward over the user plane described in chapter 3. On the other hand, we will observe the latency of telecommunication-specific data and throughput from arbitrary data traffic for the control plane. To produce realistic data and traffic flow as in an actual cellular network, we will implement a full EPS. To create traffic into the user plane, we will include a simulated UE. Resources available in the VNFs may affect the performance of WireGuard. We will therefore do a series of tests with various amounts of resources.

After deploying an NS, we will run multiple tests to observe the performance of WireGuard in our architecture. The tests will give measurable results on how the different deployments of WireGuard-as-a-service perform. For reference, we will create an EPS NS without WireGuard. This NS will be in equal to a functioning NS with WireGuard between EPS components, except the VPN tunnels. The following items describe the overall measurement tasks we plan for the NSs and NSIs we deploy.

- Observe latency on response time of packets on eNodeB (eNB) or Mobility Management Entity (MME) when a UE connects.
- Observe maximum throughput and latency in the user plane with a UE using iPerf3 to a reachable service when connected to the cellular network.
- Measure maximum throughput and latency directly between two components in the EPS using iPerf3.

2.2 Tool Decisions

To implement and perform the tests described in Section 2.1 several tools and methods can be used. The following subsections will reason why we have chosen the tools used in this thesis.

2.2.1 VNF Architecture

We have approached the tool selecting to choose tools that make the lab and tests as realistic as possible. OSM can deploy processing units in different ways. A popular way of deploying processing power is by using Kubernetes (K8s) pods as Container Network Functions (CNFs)/KNFs [OSMe]. With the installation method we have used for OSM, a K8s cluster is installed ready to use on the same VM as OSM. However, we will not deploy any CNF on it as part of our NSs. In this way, we deploy the OSM on a separate server.

A way of splitting applications and connecting them with internal networks is by using multiple Virtual Deployment Units (VDUs) or CNFs in a single VNF. A multi-VDU deployment is the way OAI is deployed in the work of Dreibholz [Dre20]. To be able to split the EPS components to support a multi-site deployment, we have chosen to use an approach with one VDU per VNF. With this approach, a simulation of a realistic network is also possible. For example, can an NS supporting a URLLC application in the edge have a VPN-as-a-Service (VPNaaS) solution back to the core.

2.2.2 OSM Version

The second choice we have made is the version of OSM we will use. At the start of the thesis, version 10 was the newest version of OSM. Since version 9 the new descriptor language, ETSI NFV-Solution (SOL)006 is used for the VNF Descriptors (VNFDs), NS Descriptors (NSDs) and Network Slice Templates (NSTs) [OSMa]. NFV-SOL006, is not backward compatible with SOL005 which causes need for translation to use projects created pre-OSM version 9. To be able to be up to date with the OSM community, we choose to use OSM version 10. We make the selection intending to make our work repeatable and usable in the state-of-the-art version of the software.

The impact of choosing version 10 is that we need to rewrite descriptors and charms from related work to reuse functionality for the practical part of the thesis.

2.2.3 WireGuard

We have chosen WireGuard as the VPN software to use in this thesis. IPsec and OpenVPN are similar software solutions that, in addition to WireGuard, are well-known VPN solutions that use state-of-the-art cryptography. However, as WireGuard is used in Haga's master thesis [Hag20], we will benefit from the previous work done in the thesis. The performance measurements carried out in the work of Haga and Donenfeld show that WireGuard performs well compared to other VPN solutions when it comes to throughput and latency [Hag20, Don17]. The performance measurements from these studies build up under our hypothesis that WireGuard is suitable to use in a 5G and beyond context.

2.2.4 Juju Charms

To deploy Day1-2 operations, Juju will be used [OSMe]. There are two ways of using Juju charms in OSM. Native charms runs the operations in a VNF independent of other VNFs and controllers. On the other hand, Proxy charms use a central controller to manage the actions. The VCA controller is normally installed as an integrated part of OSM. The controller most commonly uses Secure Shell (SSH) to configure and run commands in VNFs. There are some differences in the coding part of the different types among which libraries to use in the OSM integration. Therefore, it is preferable to use only one type in the thesis. Proxy charms enable some additional functionality as relations between nodes for scaling, management, and cross-service dependencies [Ltdb]. The previous work of both Dreibholz [Dre20], and Haga [Hag20], uses proxy charms. To reuse previous work and explore the additional proxy-functionalities, we will use proxy charms as our method for deploying charms.

2.3 Lab Setup

We have had the intention of having a lab environment to be as realistic as possible based on the suggestion of Esmaeily et al. [EKG20, EK21]. However, the lab environment has some differences from the suggested test-bed because of available resources.

The main building components for the overall lab environment is OSM as the NFV MANO, and MicroStack as the VIM. We have had access to two VIMs during the work with the thesis. One VIM having the NFVI directly on the physical hardware, while the other VIM being on top of an already cloud-based/virtual infrastructure. The resources for the VIMs have been 56virtual Central Processing Units (vCPUs),

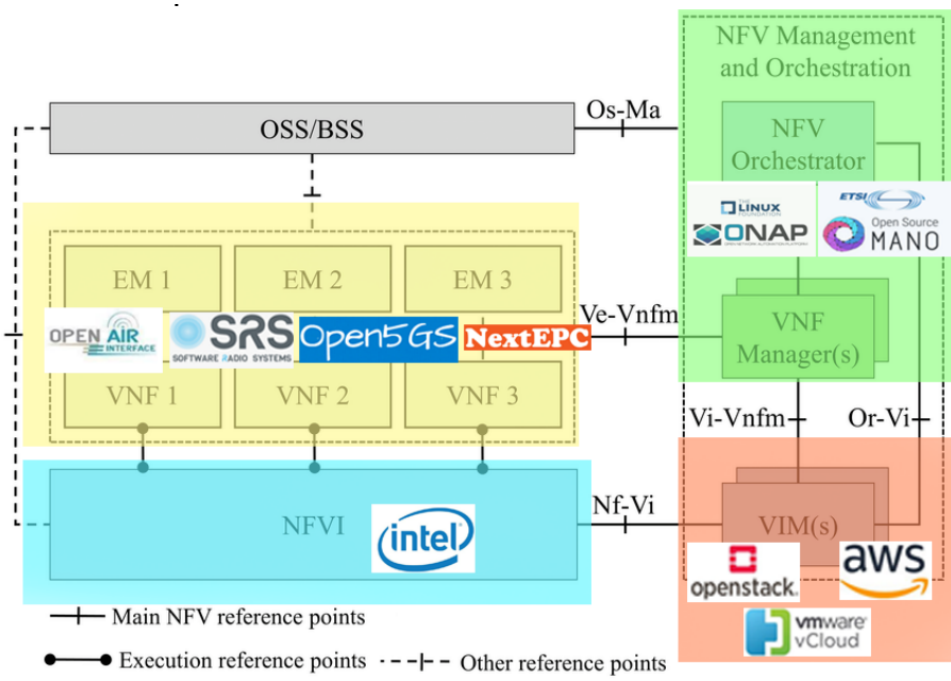

Figure 2.1: Software to ETSI NFV MANO framework slicing [EKG20].

126GB Random Access Memory (RAM) and 915GB storage for the first and 9vCPUs, 32GB RAM and 150GB storage for the other one. The relations between the different components and related products are shown in Figure 2.1.

With limited hardware to run on at the start, with only having the second-mentioned VIM available, we were limited to what NSs we could build. We struggled with the OAI EPC, in particular the Home Subscriber Server (HSS) application because of the limited resources. With limited resources, the success rate for building the HSS database was low. The overall deploy time was also high, causing a timeout in OSM when running Day-1 operations.

After a while, we got access to a separate infrastructure where we installed a fresh Ubuntu 18.04 instance. To enable the new server as a VIM, we installed MicroStack and connected OSM to it. We also established a connection to the other VIM to prepare for multi-site deployments. From OSM we have had access to both MicroStack instances and have therefore been able to deploy NSs on both VIMs.

To connect the MicroStack instance as a VIM in OSM we ran the commands in Listing 2.1. The same commands were used for both VIM connections, only changing the name and IP address.

```
osm vim-create --name a2ntnu_microstack --user admin --
    ↪ password <password> --auth_url https://<ip address
    ↪ >:5000/v3/ --tenant admin --account_type openstack
osm vim-update a2ntnu_microstack --config '{use_floating_ip:
    ↪ True}'
osm vim-update a2ntnu_microstack --config '{insecure: True}'
```

Listing 2.1: Instantiation of VIM in OSM.

For connecting OSM to multiple VIMs, we needed to adjust the network settings on one of the MicroStack instances. The management network on the VIM should be reachable from the VCA. As we have had a co-located OSM and Juju VCA controller, we should be able to reach both the VIM Application Programming Interface (API) and the VNFs management network from the OSM VM. MicroStack comes with a predefined subnet, 10.20.20.0/24, which works fine for a single-VIM setup. However, to enable multi-VIM usage, we have changed the subnet of the second MicroStack instance to avoid network conflicts. To achieve this we change the IP address of the interface on the second VIM with the command, *ifconfig br-ex 10.21.21.2/24*, to set the new IP address. We also needed to change the config file of Horizon, the web Graphical User Interface (GUI) module of MicroStack, by replacing the old IP address with the new in the file */var/snap/microstack/common/etc/horizon/local_settings.d/_05_snap_tweaks.py*.

Lastly, we connected the VIMs with a WireGuard tunnel. Connecting the VIMs was done to make VNFs connect across VIMs. Therefore, VNF application data goes through a double, nested WireGuard tunnel when testing WireGuard between VNFs in the multi-site setup. The bandwidth between the two MicroStack instances is limited by the ISP given to 200 Mbps.

Chapter 3
Background and Related Work

This chapter consists of three sections. In the first section, we explain topics related to essential aspects of this thesis. The second section introduces several relevant papers and works to this thesis. Finally, the third section describes how we have gained knowledge of the OSM framework through attending their OSM Hackfests.

3.1 Background Theory

3.1.1 4G Networks

The transition from 4G to 5G networks consists of introducing new KPIs and network components to achieve them. Still, 4G components will exist in a 5G context, as a NSA solution, at least until the MNOs migrate over to full 5G SA networks.

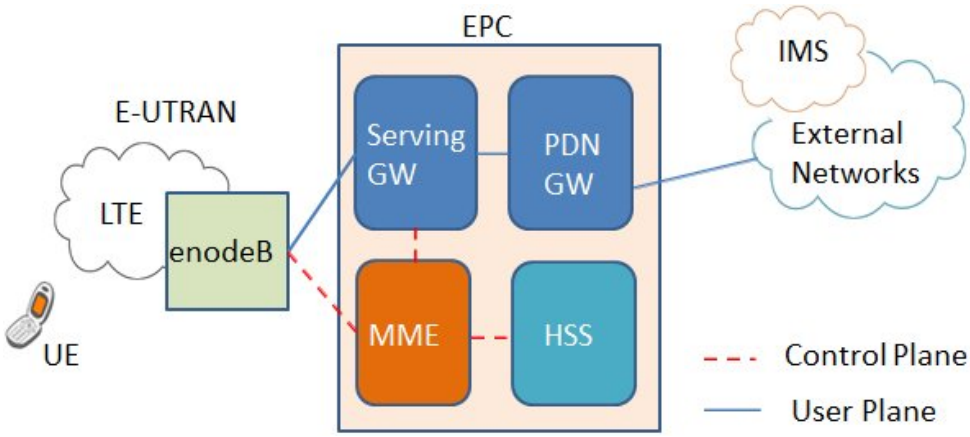

Figure 3.1: EPS architecture [BVA+18].

Another name for 4G networks is EPS. The EPS consists of the EPC and the Evolved-Universal Terrestrial RAN (E-UTRAN) [Sau14]. The E-UTRAN is more commonly known under the name Long Term Evolution (LTE). The different components in the EPS and their relations are shown in Figure 3.1. The HSS provides the database where subscriber information is stored [BVA+18]. Further, the HSS connects to one or more MMEs. To locate an UE a network is divided in multiple Tracking Area Codes (TACs) representing geographical areas. The MME is the component responsible for handling mobility features in the EPS. This includes keeping control of a UEs Tracking Area (TA) and supporting handover between eNBs. In the authentication of UEs, the MME is involved to allow and verify access to the network. The MME communicates over the S1-C interface towards the eNBs and through the S6a interface to the HSS.

eNBs are autonomous units responsible for the wireless part of the network. Therefore, the eNB performs the direct communication for providing network service to UEs. In addition to the air interface, Uu, to the UEs and the S1-C interface going to the MME, the eNB connects to the Serving Gateway (SGW) through the S1-U interface. The function of the S1-U interface compared to the S1-C interface is the user data, such as IP traffic, which is sent directly to the SGW. On the other hand, control data that is essential for the cellular network, like location updates, goes over the S1-C interface.

The role of both the SGW and the PDN Gateway (PGW) is to handle user data by routing traffic. The difference is that the SGW is the connection point for the eNB to route data to while the PGW routes traffic to other (external) networks. Other networks could for instance be other Radio Access Technologys (RATs), MNOs or IP networks such as IP Multimedia System (IMS) services and the Internet. There is also a difference between the SGW and the PGW in the additional control and tunnel setup, done by the SGW.

The subscriber information for a UE consists of information used by the network to authenticate the UE to offer its services. A UE is assigned a global unique identifier, International Mobile Subscriber Identity (IMSI). The IMSI is constructed using the Public Land Mobile Network (PLMN) and an operator unique id, the Mobile Subscription Identification Number (MSIN). The MNO also assigns a phone number to the subscriber, the Mobile Station International Subscriber Directory Number (MSISDN). The network, on the other hand, also uses identifiers to be recognized internally and by UEs. The primary network identifier is the PLMN. The PLMN consists of the Mobile Country Code (MCC) representing the country of network and the Mobile Network Code (MNC), a designated code for the MNO normally given by the telecommunication authorities in a country. When connecting to a network the UE does two mutual authentications, one for the Access Stratum (AS) between the

UE and the eNB and one for the Non-Access Stratum (NAS) terminated at the MME. The Universal Integrated Circuit Card (UICC) in the UE and the HSS store keys used in the authentication process, known as Authentication and Key Agreement (AKA). During the AKA, the UE sends its IMSI to the eNB. The eNB then forwards the IMSI to the MME which asks the HSS for derived keys from the operator key and the shared secret key, K, stored in the HSS and UICC. The derived keys are then used to challenge the UE to check if it has the same keys as the HSS.

3.1.2 Towards and Beyond 5G Networks

Control and User Plane Separation (CUPS) was introduced in 3GPPs release 14 [PS]. Separating user data from the control plane allows processing user data in other places than the core. CUPS continues in beyond 5G to improve flexibility and support new use case scenarios. This is shown in Figure 3.2 where the two new 5GC components, Session Management Function (SMF) and User Plane Function (UPF) handle the control and user data previously sent to the gateway and MME components in the EPC.

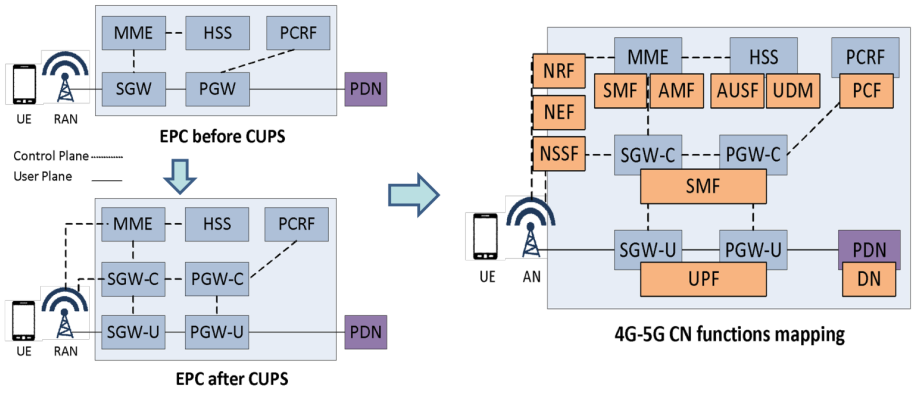

Figure 3.2: 4G and 5G core network components comparison [Ope].

The architecture of 5G networks is flexible, and it depends on the desired functionality. An example of a non-roaming reference architecture of the 5GS is shown in Figure 3.3. The figure places the components in three levels. The user plane components at the bottom of the figure, and the control plane NFs in the middle and top section [ETS20]. The control plane is built up as a Service Based Architecture (SBA) with Service Based Interface (SBI) to access the NFs. The name for the SBI is used to connect to the various user plane NFs. Between the control and user plane are the names of the different interfaces. These are similar to the interfaces in the EPS. The use of SBI in

the control plane differs from the point-to-point-based interfaces of the user plane. In both Figure 3.2 and Figure 3.3 three important components are placed centrally, the UPF, the SMF and the Access and Mobility Management Function (AMF).

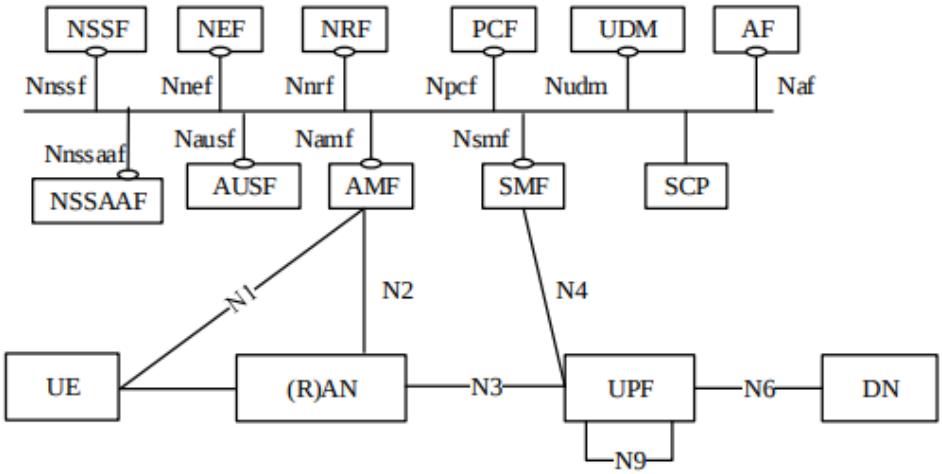

Figure 3.3: 5GS reference architecture [ETS20].

The role of the SMF is to handle multiple UE related actions and manage its sessions, such as IP address allocation and control policy functionality. This is the control plane functionality done by the SGW and PGW in the EPS. The user plane part of the SGW and PGW is handled by the UPF, with routing and forwarding as its main tasks. The AMF together with the SMF terminates the NAS signaling between the UE and the core. The AMF is also responsible for mobility management and it acts as a security anchor for the UE by keeping intermediate keys, communicating with the Unified Data Management (UDM), and being responsible for access authorization and authentication. The SMF, AMF and UPF are the base components in the 5GC with the rest being added dependent on the service. To have a functional network, additional NFs are needed. The functionality for the other NFs in Figures 3.2 and 3.3 are briefly described below.

- The RAN is responsible for the wireless part of the network as in earlier RATs [EKM22]. The gNodeB (gNB) holds the same functionality as the eNB in LTE networks.
- The UDM together with the Authentication Server Function (AUSF) handle functionality similar to the HSS in the EPC. While the AUSF manages the incoming authentication request and policies, the UDM stores the subscriber information. The AUSF controls the communication to the UDM and AMF.

- Managing network slices in 5G has introduced multiple new components. The Network Slice Selection Function (NSSF) holds control over the network slices and routes UEs to desired network slices and AMFs. To allocate the UE to a network slice, the NSSF checks if the UE is allowed to connect to the it.
- Service Communication Proxys (SCPs) can operate on different levels to route and forward messages to NFs or other SCPs. Some examples of usage include load balancing, monitoring, and communication security.
- The Network Slice Specific Authentication and Authorization Function (NSSAAF) handles the communication to external Authentication, Authorization and Accounting (AAA) servers for network slices.
- The Network Repository Function (NRF) provides functionality to discover services and maintains profiles for NFs and SCPs as well as their status and health.
- The Network Exposure Function (NEF) communicates between internal and external networks by translating and securing messages at the border of the networks. The NEF also keeps track of information about NF capabilities which it can make available for other internal or external parties.
- The Policy Control Function (PCF) handles various rules in the network. For instance mobility management to provide QoS and access control to NFs.

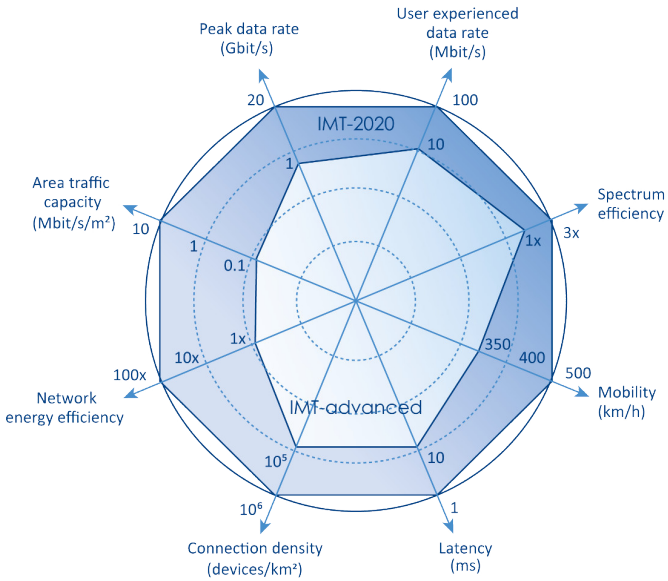

Figure 3.4: KPI comparison between 4G and 5G networks [ETSd].

A comparison of the KPIs of 4G and 5G networks is visualized in Figure 3.4. 20 Gbps peak data rate, lower latency, and an increase of 10 to 100 times for user data rates are examples of the increased demands of 5G networks shown in the figure.

In a service-oriented network, different services require different performance. The use-case scenarios in 5G networks can be categorized in three main categories: eMBB, URLLC, and mMTC. The KPIs cover all these scenarios. An eMBB slice can, for instance, be suitable for high data rates, but not necessarily low latency. To differentiate between the different requirements, ETSI has specified a list of 5G QoS Identifier (5QI) [ETS20]. The 5QI specifies the priority and requirements of a network slice.

3.1.3 Isolation Theory

A network slice can be understood as an E2E logical network on top of a shared infrastructure [BAMH20], i.e. NSIs can be created on-demand, mutually isolated from other NSIs. While NSIs are isolated from each other, NFs in an NSI can be shared across NSIs and to be cross-domain. One NSI can therefore use functionality of other NSIs. The focus on sharing application services enables the creation of service-oriented networks.

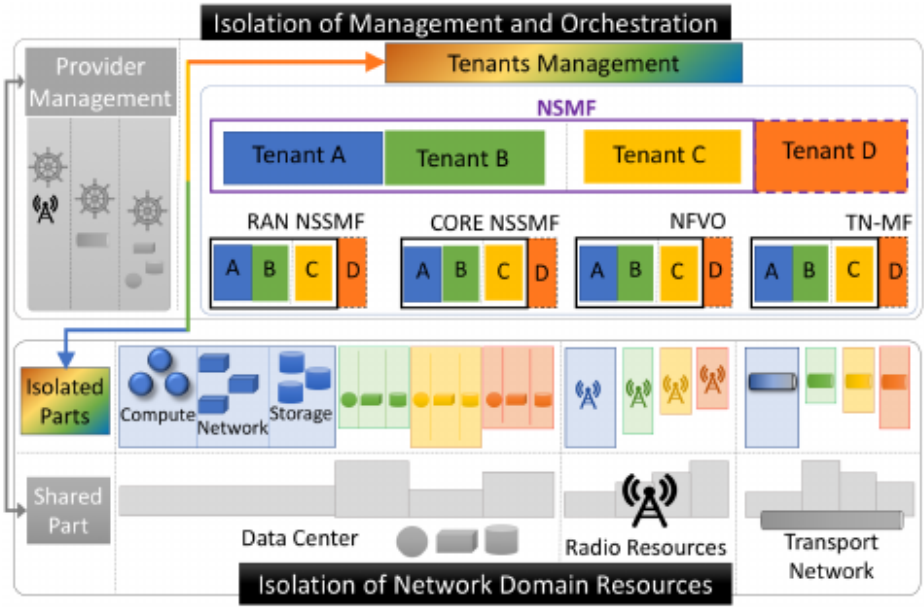

Figure 3.5: MANO of isolated network slices using a shared infrastructure [GOLH+20].

Figure 3.5 illustrates the separation of NSIs by showing a shared part of data centers, radio resources and transport networks [GOLH+20]. The different tenants are then

given resources from the shared part of the different domains. The resources are isolated from the other tenants, meaning that one tenant only will use the assigned logical resources. As application usage can vary, assigning physical resources to given network slices may be ineffective in isolating physical resources. An alternative to physical isolation is logical isolation. For logical isolation, a physical resource distributes over multiple network slices. While the tenant managers control their logical resources, the network slices access the physical resource using access schemes. Prioritization of tenants to ensure QoS can be done in different ways while using logical isolation. For example using virtual Local Area Network (VLAN) tagging.

Using shared infrastructure opens for security issues through the different security dimensions, Confidentiality, Integrity and Availability (CIA). Gonzales et al. [GOLH+20] look at security in an isolation context using three dimensions. The three dimensions are performance, security, and dependability. For the performance dimension, NSIs should have enough resources available to fulfill its KPIs. Furthermore, enough resources should be available regardless of the activity of other NSIs. A network slice delivering an URLLC application should, as an example, not be affected by other NSIs network traffic. The security dimension of Gonzales et al. includes the CIA triad. An attack in either of the CIA directions in one NSI should not affect others. For example, having malicious activity breaching the confidentiality in a NF in one NSI should not enable the adversary to attack other NSIs running on the same shared infrastructure. The same principle is also valid for the dependability dimension. Hardware or software failure in one NSI should not cause errors across other NSIs.

Figure 3.5 also shows the two levels of management for realizing network slicing, provider and tenants management. Provider management describes supervising the set up of network slices on the physical infrastructure. To create network slices, allocating the resources needed for an E2E composition of the slice must be done [GK19]. For instance, it is the provider manager responsible for setting up the connection between the NFV MANO and VIM. On the other hand, tenant management describes the administration done within a NSI. Tenant management includes setting up the applications and configuration of VNF into NSs. OSM supports differentiating in management level by its Role-based Access Control (RBAC) [OSMi]. A manager can control one or more tenants as shown in Figure 3.5. One manager controls tenants A, B, and C in the figure, while tenant D gets controlled by a separate tenant manager. Figure 3.5 shows the separation of managers with the dotted line around “Tenant D.”

3.1.4 OSM Descriptor Language

OSM uses Yet Another Next Generation (YANG) data models to describe how objects are created. For objects like NSD files with YAML syntax, compliant with ETSI NFV-SOL are used [ETSc]. In addition to the SOL descriptors used directly in VNFDs, NSD and NSTs, other SOL descriptions regulate other building blocks like the descriptors folder structure. The different SOL descriptors and how they are related in the NFV architecture are shown in Figure 3.6. The version for VNFDs, NSDs, and NSTs used in OSM version 9, 10 and 11 is SOL006 [ETSa]. Older SOL versions are not directly transferable to SOL006.

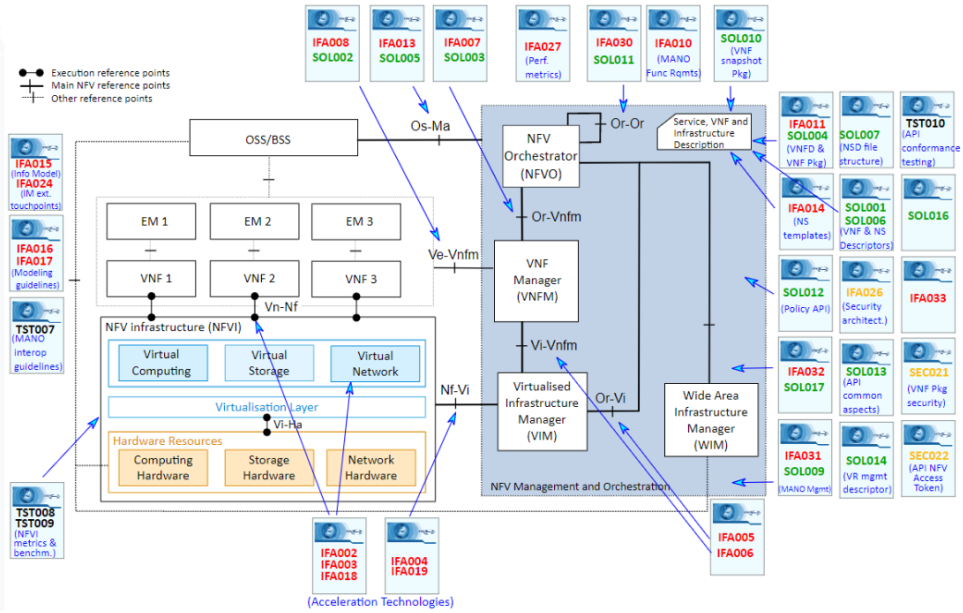

Figure 3.6: SOL specifications supporting the different parts in the NFV Architecture Framework [ETSc].

While ETSI NFV-SOL006 is a description of how to build the data models, OSM is a software using it. A description of OSMs information model is therefore necessary to know how and what OSM impelement NFV-SOL006. References [OSMc, OSMf, OSMd] show the keywords of the information model that can be used for respectively NSD, VNFD and NST.

3.1.5 OSM Onboarding

OSM onboarding describes the LCM of NFs. The operations phase in Figure 3.7 includes the OSM terminology of Day-0, Day-1 and Day-2 operations. Day-0 actions and the release phase matches. Further is the deploy phase matching OSM Day-1 actions, and Day-2 matches the operational phase in the figure. Monitoring in OSM is done during the whole life cycle for NSs [OSMe]. Contrarily is the manager and developer work done before the deployment part of the development phases of Figure 3.7. Within OSM the descriptor, validation, and packaging phases are accomplished by having standardized ways of describing and packing objects as described in Section 3.1.4. OSM validates these when modified to avoid errors. However, OSM do not support the emulating and testing phase of Day1-2 operations. Other tools OSM must be introduced separately for the Continuous Integration/Continuous Deployment (CI/CD) for these phases.

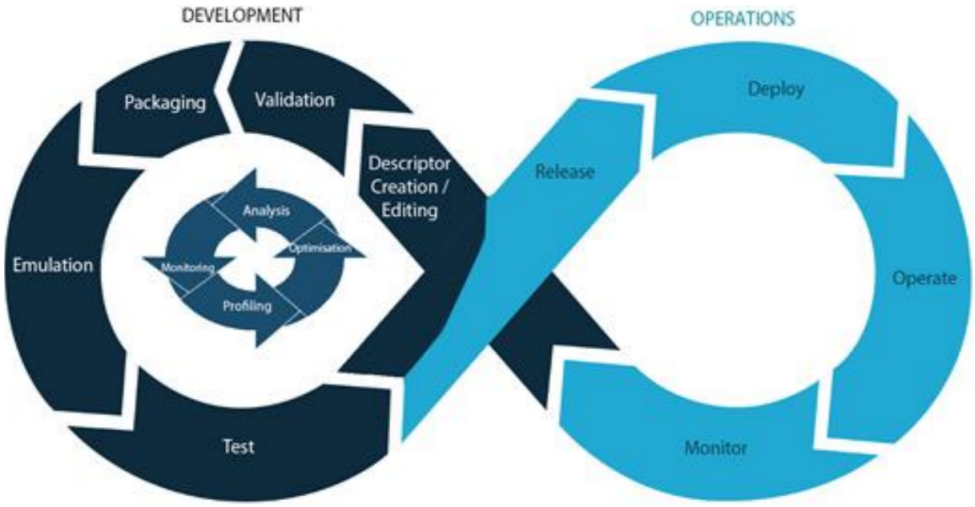

Figure 3.7: Steps of lifecycle management of services in a 5G and CI/CD context [Groa].

There are two ways of deploying Day-1 operations in OSM, either using Helm charts or Juju charms. Helm charts, being a packet manager for Kubernetes, can be used when building KNFs [OSMe]. Juju on the other hand can be used for both KNF, directly in VNF and on NS level [OSMc, OSMf]. Juju charms have two modes of operation. Either native inside a VNF or by using a proxy charm where a central controller, VCA, connects to the VNF through their management interface to run the instructed commands. By default proxy charms in OSM use SSH for the VCA to VNF connection [OSMe]. In both native and proxy charms, Juju uses Python scripts to

perform intended actions for both Day-1 and Day-2 operations. To reference actions in the VNFD or NSD, the developer must place a folder including configuration and the charms script alongside the descriptor files. The Juju config files use YAML syntax to describe metadata and the actions implemented in the Python code. To connect the Python code in proxy charms with OSM a specified python library, *charms.osm.sshproxy*, is used [OSMb].

The OSM End User Advisory group has made a white paper to show how OSM can be used in real-world applications [Grob]. Even with 5G being focused on cloud-native infrastructure, Physical Network Functions (PNFs) will likely still be in use for several scenarios. OSM supports integrating PNFs alongside VNFs and CNFs to manage and orchestrate hybrid E2E networks. In the paper, CI/CD is discussed as a future feature that is currently under evaluation and development. A proposed architecture of the CI/CD pipeline for NFV development is also discussed. The main parts of this architecture are a separated VIM as NFV test infrastructure before it goes into production. Streamlining descriptor development, code review and automated build and test suite are some of the potential advantages of CI/CD. The steps in a CI/CD cycle overlap with the steps for LCM in Figure 3.7 [Inc]. Development of a CI/CD architecture for NFV in OSM will therefore likely improve LCM of NS.

3.1.6 WireGuard

WireGuard is a VPN software that intends to be a compact and faster alternative to other popular VPN software like IPSec and OpenVPN [Don]. The original version described in paper [Don17] totals under 4000 lines except for cryptography libraries. The limited code lines and configuration options intend to make WireGuard fast and avoid user-caused configuration errors. WireGuard has been included in the Linux kernel since version 5.6 [Tor]. In other words, the underlying libraries for installing and using WireGuard get preloaded in OS images with kernel version 5.6 and higher. After initial setup, the WireGuard protocol opens up for full mobility and flexibility, which means that both the server and client can move to another IP address. The other part will then update the peer's location by verifying the incoming public key.

Unlike IPSec, WireGuard users cannot choose their cryptography configuration. Instead of the user choosing cryptography parameters, WireGuard has selected state-of-the-art cryptography prior. The cryptographic protocol is pre-defined using ChaCha20 for symmetric encryption, with Poly 1305 providing the authentication in the protocol. The peers are assigned one private and one public key from key generating. The elliptic curve, Curve25519, is used for the key creation. While the private key stays at the owner, the public key gets transferred to the other peer(s) to make them able to verify data encrypted with the private key.

The VPN tunnel uses the handshake procedure shown in Figure 3.8. The connection setup depends on the workload of the responder receiving the initial message. Either the responder answers with a *response* message to enable the session or with a *cookie reply* message. In the case of receiving a *cookie reply* message, the initiator must send a new handshake initialization. During the handshake procedure, the participants verify that the traffic received is from a previously received public key added in WireGuard's key routing table. If the public key matches during the handshake process, symmetric session keys are generated. WireGuard then uses the symmetric keys to encrypt the transport data. While connected, the peers will regularly refresh the symmetric keys to provide perfect forward secrecy.

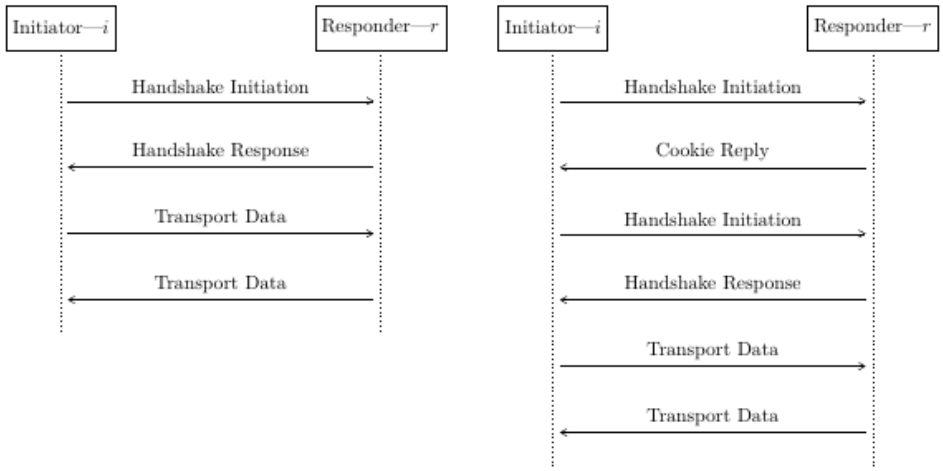

Figure 3.8: Handshake procedure in WireGuard [Don17].

3.2 Related Work

Studying different aspects of the 5G sphere is a focus area for several recent studies. This thesis itself builds on previous work, in particular, the code given in the work of Haga and Dreibholz [Hag20, Dre20]. To get an overview of other work relevant to the topics of 5G MANO in general, OSM, isolation and security in 5G are therefore helpful.

3.2.1 Towards 5G Network Slice Isolation with WireGuard and Open Source MANO

The master thesis of Simen Haga [Hag20] compares WireGuard to other VPN solutions like IPSec and OpenVPN. Further, Haga et al. [Hag20, HEKG20] study

how to use OSM to deploy WireGuard in a NS and give a demo of this. The demo uses two VNFs which are accessible through a management network. In addition, a data network is created for the WireGuard tunneling between the VNFs. The network diagram for the NS is shown in Figure 3.9. Providing peer connectivity is done using Day-2 operations. To retrieve the necessary keys for the peer setup, the manager must manually collect the public keys from the VNFs.

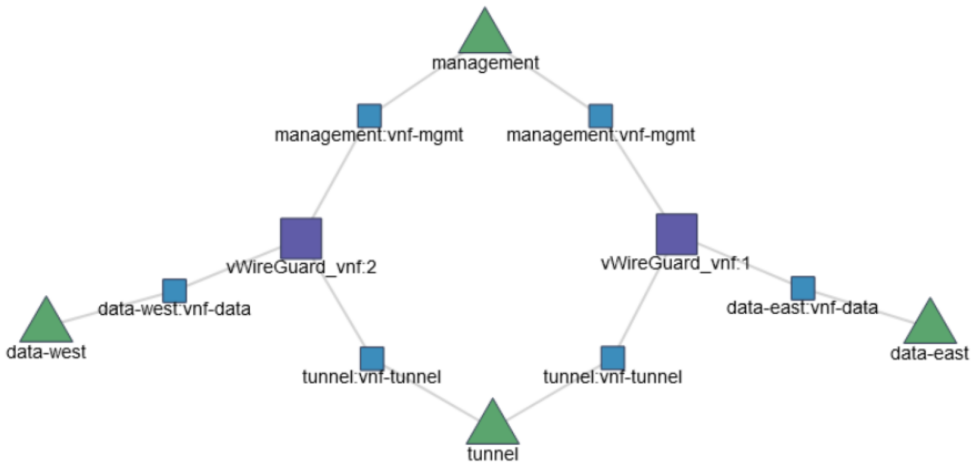

Figure 3.9: Network connection in the NS of Haga et al. [Hag20].

The master thesis has some key findings that are relevant to this thesis. The first is that OSM can manage and orchestrate WireGuard to secure communication between VNFs. Further, tests in the thesis show that WireGuard gives better throughput and lower latency than OpenVPN.

3.2.2 5G VINNI

In the 5G VINNI project, Dreibholz has deployed the EPC in OSM using OAI. The work is described in paper [Dre20]. The supporting code for the paper is available on GitHub [Ser]. As shown in Figure 4.10 different EPC components and the interfaces between them are implemented as VDUs in a single VNF. One of the goals for the study is to have a base core network ready for further future expansion. One of the extensions suggested is to include a MEC setup.

The project also looks at the radio part of LTE using OAI for the implementation of eNB and an actual modem for UE. An Ettus USRP B210 Software-Defined Radio (SDR) was used for the eNB. The UE was connected over the air using a Huawei E392 USB modem on a laptop. The paper presents throughput measurements for the

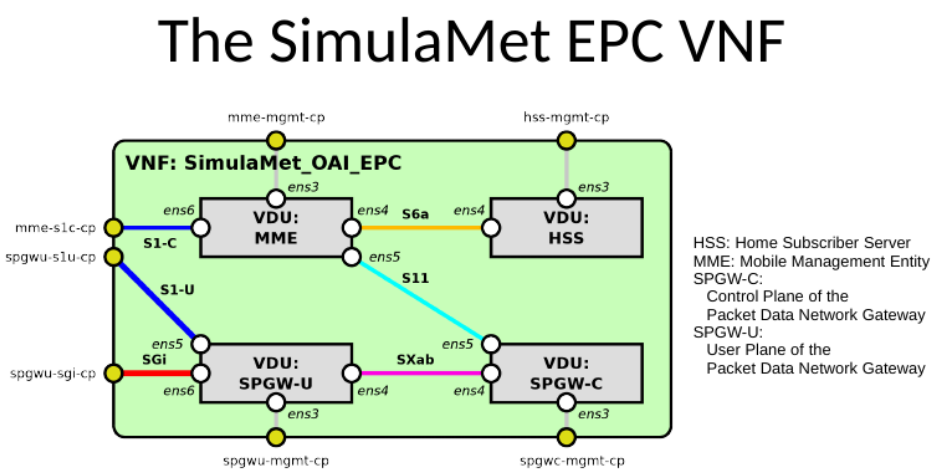

Figure 3.10: Architecture of SimulaMet EPC VNF [Dre20].

UE connectivity to the EPC. A download speed of around 11 Mbps, and an upload speed of around 4 Mbps were measured using SCTP in one direction. The paper's throughput measurements for TCP were 0.3 Mbps and 4.5 Mbps for download and upload speed. The low download throughput for TCP is an unexpected result. The paper addresses the result but does not conclude the low throughput. However, it points out that there might be a software bug with OAI or the setup. As written in the conclusion section, deployment of OAI can be an error-prone task. When OAI first is set up, using OSM with charms helps to deploy the NS in the same way every time. Preparation of a working configuration is done in templates of NSDs and VNFDs. The templating can therefore reduce errors compared to installing OAI manually.

3.2.3 Service Function Chaining in 5G and Beyond Networks: Challenges and Open Research Issues

SDN and NFV technologies open up for a flexible way of chaining services together. SF Chaining (SFC) is described as the way of connecting and ordering SFs together. Using the mentioned technologies, SFC is introduced in mobile networks. For instance, SFC can be used for granular policy control or QoS optimization. SDN enables forwarding of data traffic in a dynamical and flexible way. As described in section 3.1.4 the NFV orchestration enables creation of the virtualized SFs, VNFs, and connectivity between them used in SFC.

Several challenges for SFC are identified in the paper of Hantouti et al. [HBT20]. Different challenges regarding MANO, composition of SF into chains, QoS and security are studied. For the security challenges, authentication and classification of data traffic along with trust internally and between SF components are important aspects for the SFC operations. A key concern when it comes to security in an SFC context is networks managed by different operators. The paper suggests encrypted tunnels and encapsulation as a countermeasure. Operators should use encrypted data tunnels to provide packets' integrity and prevent bypassing of policies.

3.2.4 5G Multi-access Edge Computing: Security, Dependability, and Performance

The paper [NGO21] by Nencioni et al. looks at challenges regarding security, dependability, and performance using MEC in 5G networks. A key finding is that there have been multiple studies of security in a 5G MEC environment, however, with the dependability as a less studied facet. To be reliable, URLLC applications have dependability requirements with the use of MEC. Therefore, the paper points out the few studies of dependability as contradictory. The paper further addresses software and hardware errors as challenges regarding dependability.

MEC security challenges are categorized into three levels in the paper, general challenges, system-level, and host level. Further, Nencioni et al. compare the composition of an NFV-based NS with SFC used in association with SDN. Figure 3.11 illustrates the system and host level and shows the connectivity between different components. The components of the MANO part of the figure overlap with those represented in Figure 2.1. On the other hand are the application and infrastructure parts of Figure 3.11 expressed with more specific MEC parts.

At the host level, the paper addresses challenges regarding VNF management, VNF applications, VIM, data-plane and NFVI. Some of the described challenges are physical security and malicious activity running alongside in a shared environment. Further, the paper discusses challenges using virtualized hardware. For instance, is resource usage both in computation and network an example of a virtualized environment challenge.

For the topic of shared virtualized networks, the paper addresses challenges related to the overhead caused by cryptographic protocols. Overhead in, for instance, VPNs may introduce both potential performance and security issues. The latter originated by verticals not using security features to become lightweight and improve other performance.

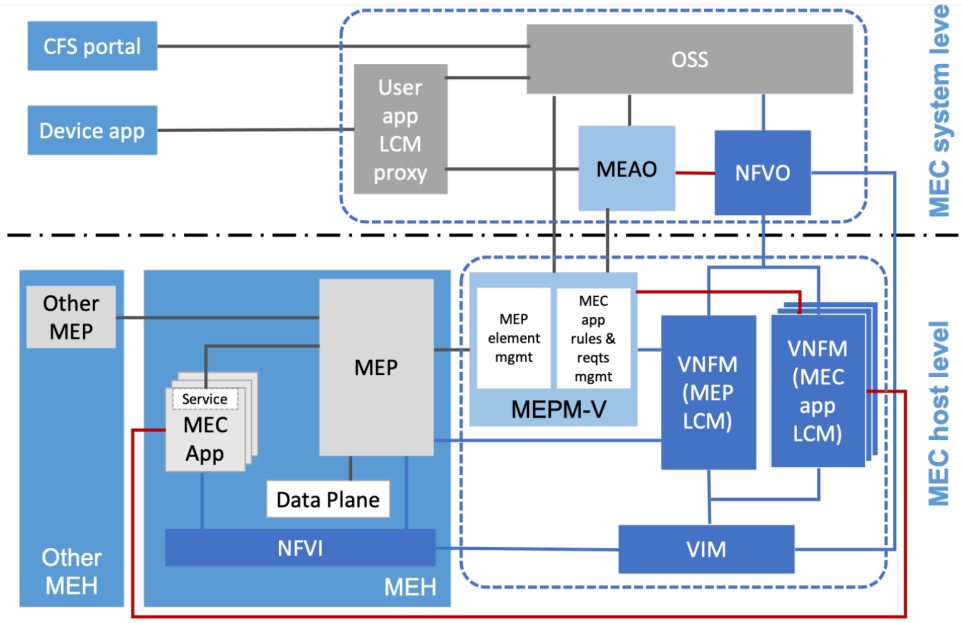

Figure 3.11: MEC-in-NFV architecture presented in two levels with deployed applications and their management and orchestration [NGO21].

Geographic areas related to MEC security is another topic discussed in the paper. One potential security advantage of having smaller geographical areas are the resistance of Distributed DoS (DDoS) attacks. By using MEC with narrow coverage, it is possible to limit the attack geographically.

For the MEC system-level, challenges regarding LCM and MANO are addressed. As a reference to components used in this thesis, OSM introduces LCM by using cloud-init and Juju. In the paper, Nencioni et al. suggest a combination of centralized and decentralized defense mechanisms to encounter system-level challenges. Intercommunication between MEC and LCM systems is one example where a combination of centralized and decentralized countermeasures is applicable. Other examples include hardening of the MANO, MEC platform and VMs.

One of the general challenges mentioned is trust between stakeholders. An example where this is a challenge is between MNOs. Further, virtualization technologies in NFV, SDN and network security are technologies that bring their challenges to the table. Finally, to finish the general challenges, a specific risk raised for network security is the data transit between the edge and the core.

3.2.5 5G Core Network Security Issues and Attack Classification from Network Protocol Perspective

The paper [Kim20] by Hwankuk explores new security threats introduced with 5G networks and classifies several potential attacks. Using a common infrastructure with logically separated resources between network slices introduces the risk of attacks from one slice to another if not properly isolated. The paper suggests that an attacker could eavesdrop or tamper with data in other slices without proper encryption. Possible vulnerabilities for the different protocols used are a major part of the paper. New protocols required for inter-operations may introduce new security issues. Further, the use of a SBA and APIs generally inherits vulnerabilities known from other IP networks. For GPRS Tunnelling Protocol (GTP) traffic, man-in-the-middle and DoS are mentioned as possible attacks in the control and user plane. Using protocols designed for closed internal environments introduces other challenges. Issues such as non-encryption, unauthenticated packets with unknown origin, and errors requiring manual fixes are examples of challenges for these protocols.

Further, the paper addresses the security of different parts introduced by 5G networks. Some of the topics mentioned are MEC, 5G RAN, Internet of Things (IoT) traffic and software-based infrastructure through SDN and NFV technology. In addition, the paper discusses several improvements as an improvement of the 5G standard compared to earlier cellular technologies. These include improved IMSI capture prevention and Security Edge Protection Proxy (SEPP) functionality between MNOs and other domains. In total, the paper summarizes five security issues related to the development of 5G networks. The five issues are listed below.

- DDoS caused by vulnerable IoT devices.
- Coverage of cells and RAN failure.
- Monitoring and protection of decentralized devices.
- Proper isolation of shared physical infrastructure.
- Reliability and security of third-party applications and their API connections to the internal core network.

3.2.6 Virtualized Cellular Networks with Native Cloud Functions

In the master thesis [Gon21] by Gonzales, OSM is used to show how it can deploy 5G NFs using different techniques. The thesis focuses on creating container images and deploying CNFs by compiling development done in the Open-VERSO project. The complexity with this approach is moved to the image creation itself, making Day-1 actions more basic.

Two open source projects, free5GC and open5GS, that implement the 5GC are used to achieve the PoC. After preparing K8s images, OSM is used to orchestrate the

deployment. The NSs created in the thesis consist of a single VNF with a single KNF inside. To configure the KNFs, Helm charts are used as Day-1 operations method. After the deployment of the NSs, a simulated gNB and UE are added in the lab to test E2E connectivity.

3.2.7 A Secure Link-Layer Connectivity Platform for Multi-Site NFV Services

Comparable to this thesis, Vidal et al. [VNL+21] addresses secure communication between NFVIs using OSM. The paper introduces a PoC of a platform, L2S, to provide secure link-layer connectivity between NSs at multiple sites. In the PoC, a VNF with switch functionality at each site to provides the cross-NFVI connection. *Open vSwitch* with VXLAN is the main part in the L2S VNF solution, performing the switching. To secure the link between VNFs, Vidal et al. uses IPsec as VNFs solution. Figure 3.12 shows the lab setup used in the paper. Multiple NSs can route their traffic through the switch. For instance, is site B in the figure serving two NSs. The L2S are creating secure tunnels between the NFVIs to abstract the lower layer networking for the other VNFs. To add configuration parameters like VLAN identifiers and cryptographic keys OSM must be supplied together with the instantiation for automatic setup of the tunnel.

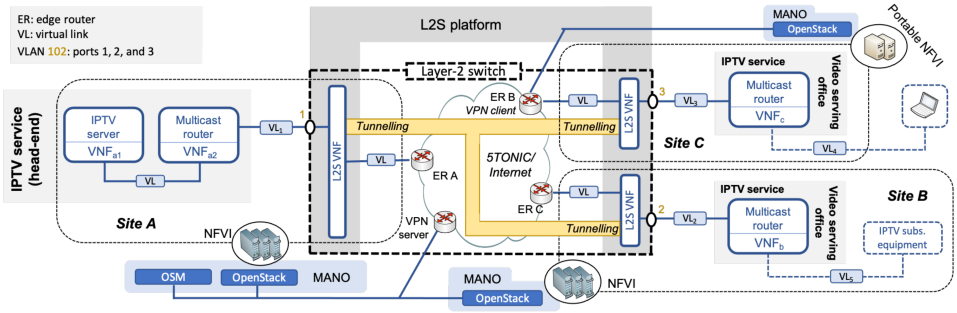

Figure 3.12: Multi-site L2S connectivity [VNL+21].

In addition to validating the functionality, performance measurements through the L2S are done in the paper. The measurements are done both with and without encryption and from one to four vCPUs. When introducing the L2S without encryption, the throughput drops from around 15 Gbps to 1.13 Gbps. The bare *Open vSwitch* throughput is measured similar also when adding more vCPUs. When adding IPsec, the throughput reduces. Depending on the number of vCPU and Maximum Transmission Unit (MTU) size, the lowest measured throughput with

IPsec was around 550 Mbps, and the highest around 1.05 Gbps, converging towards the unsecured throughput.

3.3 OSM Hackfests

The OSM community regularly has conferences to show examples of usage, extend knowledge for the users, and encourage the use of OSM. During the work with the thesis, there were two Hackfest conferences and one OSM ecosystem day that we attended. All of these were online. The Hackfests were focused on practical skills, while the ecosystem day presented new features and examples of usage. We have used the Hackfests to gain hands-on experience in the preliminary work of the thesis.

The first Hackfest we attended focused on building Day-1 and Day-2 operations for specific components, like routing with VyOS and DNS resolution with PowerDNS [vyo, BV., OSMh]. The second Hackfest focused on building OAI using their 5GC implementation. The job for the participants of the Hackfest was to experiment with OAI in different teams. First, without using OSM, then write the necessary description files and extend the deployment with Day1-2 actions. The OAI 5GC was prepared ready to build using a Juju charmed KNFs. The Juju charm came preconfigured with relationships between the different components was prepared in the Juju charm. To build a working NS, the different groups at the Hackfest had to prepare multiple K8s pods within a single VNF to use the Juju charm.

Chapter 4
Implementations

This chapter will describe how we have carried out our practical work. In particular, is the chapter addressing how we have created the descriptors and Juju charms to build use cases used in answering the research questions. The chapter also includes verification of the functionality at different development stages.

4.1 Intended End-state

To get a realistic architecture to deploy WireGuard as VPNaaS in it, we want to extend the NS described in chapter 2 [Ser]. For testing purposes, we need to have a UE connecting to the EPS. The UE will trigger the use of realistic protocols and interfaces in both the control and user planes.

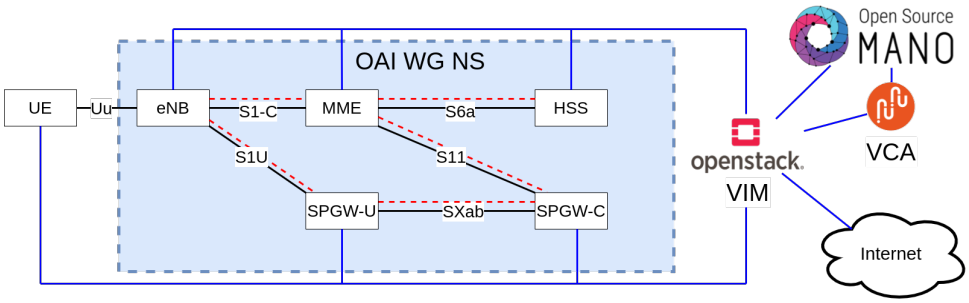

Figure 4.1: Target network architecture.

Figure 4.1 shows the overall architecture for the NS we want to build. The black lines between the VNFs are the direct network between the components. The red dotted lines illustrate the WireGuard tunnel we want to deploy on top of the black-lined interfaces. The blue lines show the management interfaces needed to communicate out of the NS. The management interfaces are used for the OSM and the Juju VCA

to access the VNFs for deploying Day-0 to Day-2 operations. The management interfaces are also the route to external networks to access Internet resources. Since all virtual networks are SDN-based on the MicroStack VIM, all external traffic to or from VNFs goes through the virtual router in MicroStack.

4.2 Adapting Previous Work

As described in section 3.2 deploying NSs for both WireGuard and OAI has been done separately by others earlier. However, both the work of Haga in [Hag20] and Dreibholz in [Ser] require some changes to function in our lab environment. The most important change is the descriptor language. Rewriting the work of the two NSs is important to extend the functionality and eventually answer our research questions.

4.3 WireGuard NS

To establish a WireGuard NS, we use the outcome of Haga's master thesis. Since the NS of that thesis was built to support OSM version 8 it utilizes the NFV descriptor SOL005. We must therefore update both the VNFD and the NSD to support OSM version 10 SOL006 descriptor language.[1]

The architecture with a NS consisting of two similar VNFs with one VDU is kept in our modification. The networking includes two networks - one management network to reach the VNFs from outside the NS and one data network to demonstrate the WireGuard functionality. The rewrite will reuse the network architecture.

We changed the folder layout and the supporting scripts for Juju charms to comply with the current structure recommended for proxy charms. For the charm functionality itself, we keep it as is. Retaining the charm includes performing the key generating and initial configuration as Day-1 operations. Peer connectivity is then manually configured as Day-2 operations using the *addpeer* action. The input to the action is the peer's public key and the network information behind the peer. Listing 4.1 shows the code snippet where the charm function creates the WireGuard base configuration.

```
def on_generateconfig_action(self, event):
    ...
    proxy = self.get_ssh_proxy()
    gateway_ip = self.model.config["ssh-hostname"]
    cmd = ['echo -e "[Interface]\nAddress = {}\nListenPort =
        ↪ 51820\nPrivatekey = $(sudo cat /etc/wireguard/
        ↪ privatekey)" | sudo tee /etc/wireguard/wg0.conf'.format
        ↪ (gateway_ip)]
```

[1] https://github.com/sondrki/TTM4905/tree/main/basic_wg_ns

```
    result , err = proxy.run(cmd)
    ...
```

Listing 4.1: Content of the on_generateconfig_action function in charm code to create inital WireGuard configuration.

Table 4.1 represents basic information for the different VNFs in the NS. The amount of vCPU and RAM is virtually allocated at the VIM.

Table 4.1: VNF information of WireGuard NS.

VNF name	OS	number of vCPU	amount of RAM (GB)	storage (GB)
vWireGuard_vnf1	ubuntu18.04	1	1.0	10
vWireGuard_vnf2	ubuntu18.04	1	1.0	10

4.4 OAI EPC

The first step we took in order to rewrite the NS from Dreibholz [Ser], was to build the base structure of the NS.[2] The NS was structured by preparing the transition from one VNF with multiple VDUs to multiple VNFs. After the transition, the new NS architecture we created consists of four VNFs running different components of the EPC. The four components each running in its own VNF are HSS, MME, Service Packet Gateway-User plane (SPGW-U) and Service Packet Gateway-Control plane (SPGW-C).

The implementation of Dreibholz in [Dre20] specifies most of its parameters using a large configuration file during instantiation. However, to avoid possible sources of errors, we included the variables inside the VNFDs directly.

After deploying the base NS, we noticed that we had to adjust the IP address configuration to use static IP addresses for the internal interfaces of the NS. When creating the NS by initializing it with additional parameters, we achieved setting the intended network configuration. Listing 4.2 exposes the necessary parameters in the additional file to set the IP addresses. The listing also includes the actual IP addresses we have used in the NS. To assign these static IP addresses, we also had to specify the network configuration in the NSD. As an example, Listing 4.3 displays the composition for the S6a interface in the NSD.

[2] https://github.com/sondrki/TTM4905/tree/main/oai_epc_ns

```
vld:
  - name: mgmtnet
    vim-network-name: test
    vnfd-connection-point-ref:
    - member-vnf-index-ref: "1"
      vnfd-connection-point-ref: hss-ens4
      ip-address: "172.16.6.129"
    - member-vnf-index-ref: "2"
      vnfd-connection-point-ref: mme-ens4
      ip-address: "172.16.6.2"
    - member-vnf-index-ref: "2"
      vnfd-connection-point-ref: mme-ens5
      ip-address: "172.16.1.102"
    - member-vnf-index-ref: "3"
      vnfd-connection-point-ref: spgwc-ens4
      ip-address: "172.55.55.101"
    - member-vnf-index-ref: "3"
      vnfd-connection-point-ref: spgwc-ens5
      ip-address: "172.16.1.104"
    - member-vnf-index-ref: "4"
      vnfd-connection-point-ref: spgwu-ens4
      ip-address: "172.55.55.102"
```

Listing 4.2: Additional VLD configurations for OAI EPC.

```
nsd:
  ...
  df:
  - id: <name of deployment flavour (df)>
    virtual-link-profile:
    - id: s6a
      virtual-link-desc-id: s6a
      virtual-link-protocol-data:
        l3-protocol-data:
          ip-version: ipv4
          cidr: 172.16.6.0/24
          dhcp-enabled: true
```

Listing 4.3: S6a network configuration in NSD.

The keys *name* and *vim-network-name* in Listing 4.2 specify the mapping between the name used for the management network in the NSD and the actual name for the management network at the VIM. The configuration after *vnfd-connection-point-ref* in the same listing is used to set static IP addresses. The identifier, *member-vnf-index-*

ref specifies the internal VNF numbering in the NSD while *vnfd-connection-point-ref* is the interface name used in the NSD.

After creating the intended base NS, we started the transition from the old to the new recommended structure for the Juju charms to perform the Day1-2 operations [OSMj]. Due to some struggles with different libraries from the old structure, we decided to use the template for the Python code we had successfully created during initial testing with OSM. We, therefore, copied the required Python functions from the original OAI EPC NS, to the working template [Ser]. The VNFs for MME, SPGW-U and SPGW-C were coming up as expected after the transformation. For the HSS, we needed to make additional adjustments to install it without errors.

OAI and in particular the HSS have multiple dependencies which must finish in the correct order to function appropriately. While reproducing the implementation of the OAI EPC we experienced that we had to make adjustments to the charm code. Therefore, we added steps in the code for error handling to wait for dependencies to be completed and catch exceptions. With the modifications, we experienced a higher success rate of deploying OAI properly.

Cassandra version 2.1, as the database used in Dreibholz's version of the HSS VNF, is an old version of the Cassandra database software [Foub]. When installing Cassandra from the package manager, we ended up with the newer version 3. However, the more recent version is incompatible with the provided setup script. Therefore, to avoid recreating the database setup scripts, we installed version 2 by adding the repository and the packet's signature in the cloud-init file.

We have also experienced that having enough resources available for the Cassandra setup is essential. After a service restart, we have observed that it normally takes several minutes before Cassandra is ready. When we tried to initialize the HSS VNF with 2 vCPUs and 6 GB RAM the HSS charm failed consequently. To encounter the failing HSS, we made two changes. First, we moved the installation of Cassandra from inside a charm action to using cloud-init. Then Cassandra is installed when the Juju actions configure the HSS. Secondly, we waited 180 seconds to restart Cassandra in the charm code. With reduced resources, Cassandra would, however, still fail. After also changing the available resources for the VNF to 3 vCPUs and 6 GB RAM the HSS installation has mostly succeeded.

With the changes to the HSS, we got a functioning NS with completed installation for all four VNFs. At the Juju VCA a successful installation is represented by the configuration completion messages in Figure 4.2. The figure also shows the components Juju creates for the VNFs with machines, applications, and the units performing the operations.

```
user@osm:~$ juju status
Model                           Controller  Cloud/Region       Version  SLA          Timestamp
3801bcc5-3c87-47c8-af64-b6c3698d0f94  osm         lxd-cloud/default  2.8.6    unsupported  17:20:42Z

App                    Version  Status  Scale  Charm       Store  Rev  OS      Notes
app-vnf-0f6baad87c-z0           active      1  spgwccharm  local    0  ubuntu
app-vnf-09db7c5b89-z0           active      1  hsscharm    local    0  ubuntu
app-vnf-38b6719157-z0           active      1  mmecharm    local    0  ubuntu
app-vnf-eb9a50d08a-z0           active      1  spgwucharm  local    0  ubuntu

Unit                      Workload  Agent  Machine  Public address  Ports  Message
app-vnf-0f6baad87c-z0/0*  active    idle   2        10.192.251.69          restart_spgwc completed!
app-vnf-09db7c5b89-z0/0*  active    idle   1        10.192.251.156         restart_hss completed!
app-vnf-38b6719157-z0/0*  active    idle   0        10.192.251.63          restart_mme completed!
app-vnf-eb9a50d08a-z0/0*  active    idle   3        10.192.251.230         restart_spgwu completed!

Machine  State    DNS             Inst id        Series  AZ  Message
0        started  10.192.251.63   juju-15c7d7-0  bionic      Running
1        started  10.192.251.156  juju-15c7d7-1  bionic      Running
2        started  10.192.251.69   juju-15c7d7-2  bionic      Running
3        started  10.192.251.230  juju-15c7d7-3  bionic      Running
```

Figure 4.2: Juju status for working OAI NS.

After deploying the NS, we tested the connectivity on the S6a interface by sending ICMP packets from the HSS to the MME. Figure 4.3 displays the successful connectivity by receiving answer packets. At the same time as we performed the connectivity tests, we listened for ICMP packets at the NFVI. Figure 4.4 shows that we can observe the ICMP packets going in the virtual network when monitoring the network traffic at the server running the MicroStack VIM. This can be seen by looking at the allocated IP addresses and the use of the ICMP protocol. Table 4.2 shows the resources used in the final version of the OAI NS.

```
ubuntu@hss:~/openair-hss$ ping 172.16.6.2
PING 172.16.6.2 (172.16.6.2) 56(84) bytes of data.
64 bytes from 172.16.6.2: icmp_seq=1 ttl=64 time=0.950 ms
64 bytes from 172.16.6.2: icmp_seq=2 ttl=64 time=0.736 ms
64 bytes from 172.16.6.2: icmp_seq=3 ttl=64 time=0.948 ms
64 bytes from 172.16.6.2: icmp_seq=4 ttl=64 time=0.898 ms
^C
--- 172.16.6.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 0.736/0.883/0.950/0.087 ms
```

Figure 4.3: Ping from HSS to MME over the S6a interface.

```
user@microstack:~$ sudo tcpdump -i any icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
20:54:20.688806 IP 172.16.6.129 > 172.16.6.2: ICMP echo request, id 17124, seq 1, length 64
20:54:20.689478 IP 172.16.6.129 > 172.16.6.2: ICMP echo request, id 17124, seq 1, length 64
20:54:20.690315 IP 172.16.6.2 > 172.16.6.129: ICMP echo reply, id 17124, seq 1, length 64
20:54:20.690818 IP 172.16.6.2 > 172.16.6.129: ICMP echo reply, id 17124, seq 1, length 64
20:54:21.690816 IP 172.16.6.129 > 172.16.6.2: ICMP echo request, id 17124, seq 2, length 64
20:54:21.691206 IP 172.16.6.129 > 172.16.6.2: ICMP echo request, id 17124, seq 2, length 64
20:54:21.691896 IP 172.16.6.2 > 172.16.6.129: ICMP echo reply, id 17124, seq 2, length 64
20:54:21.691921 IP 172.16.6.2 > 172.16.6.129: ICMP echo reply, id 17124, seq 2, length 64
20:54:22.692444 IP 172.16.6.129 > 172.16.6.2: ICMP echo request, id 17124, seq 3, length 64
20:54:22.692471 IP 172.16.6.129 > 172.16.6.2: ICMP echo request, id 17124, seq 3, length 64
20:54:22.692880 IP 172.16.6.2 > 172.16.6.129: ICMP echo reply, id 17124, seq 3, length 64
20:54:22.692895 IP 172.16.6.2 > 172.16.6.129: ICMP echo reply, id 17124, seq 3, length 64
```

Figure 4.4: Listening for ICMP messages on the NFVI.

Table 4.2: VNF information of OAI NS.

VNF name	OS	number of vCPU	amount of RAM (GB)	storage (GB)
HSS	ubuntu18.04	3	6.0	20
MME	ubuntu18.04	2	4.0	20
SPGWU	ubuntu18.04	1	3.0	20
SPGWC	ubuntu18.04	2	3.0	20

4.5 Juju Relations

OSM in combination with its included features has several possibilities of extending the WireGuard implementation in section 4.3. In this section, we will go through our further development of the VPNaaS solution.

When using Juju proxy charms, OSM creates a single Juju machine with an associated Juju application for each VNF. By default, the applications do not have any connectivity between them. Internally in the application, the VCA and the VDU establish a relationship named *proxypeer* in the OSM charms. The provided libraries from *charms.osm*[3] offer the necessary support for running Day1-2 actions over the connection. In addition to the mandatory *proxypeer* relation, the charm developer can connect other Juju applications using the Juju framework. We wanted to see if we could use Juju relations to pass the necessary information to set up the WireGuard peering. Ideally, the peer setup gets configured without manual steps.

Figure 4.5 illustrates the relationship between the different parts of the charm structure. OSM communicates with one or more VIMs to deploy VNFs. Co-located

[3]https://github.com/charmed-osm/charms.osm

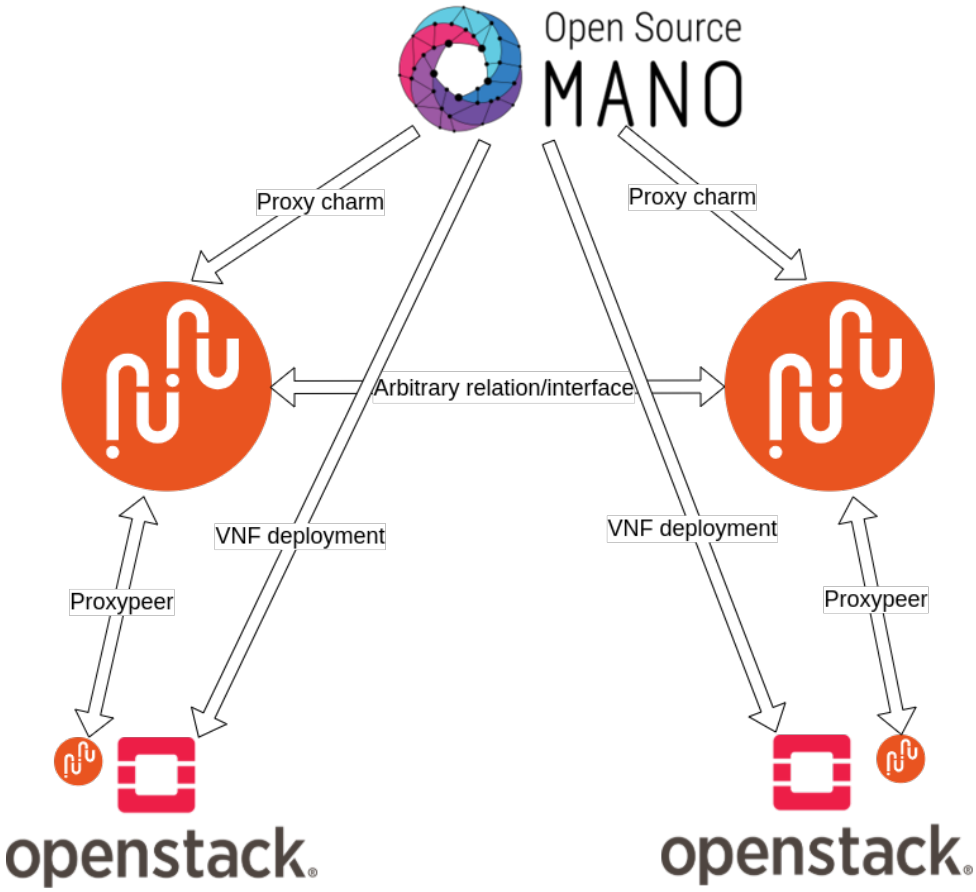

Figure 4.5: Overview of juju relation setup using OSM.

with OSM is the VCA. The VCA creates Juju machines, applications, and units represented with the Juju icon centrally in the figure. The Juju machines and their units perform the Day1-2 actions specified in the OSM descriptors using the *proxypeer* relation. To transfer additional user-defined information between Juju units, we specify an arbitrary Juju relation. The relation is created by appending the charm configuration and NSD. When the relation is created or changed, it triggers an action similar to Day1-2 operations. The VNFs can then use this action to transfer information to the VCA. The opposite VNF can then retrieve the values. The peers can transfer and receive values for both Juju units at the VCA.

The peer connections in the file *metadata.yaml* located in the charm folder specify the name of the connections of the interface. The name of the interface key must be

the same on both sides of the relation. Listings 4.4 and 4.5 show the setup for the VCA and VNF connectivity and the HSS and MME Juju application relationships. For the S6a interface we have chosen *interfaces6a* as the name to the *interface* key. Juju has two types of relationships, peers or provides/requires pairs [Ltdc]. The provides/requires relation expects the provider to make information available.

On the other hand, peer relation causes a mutual response in the application. We started using provide/require pairs to pass information in the development process. This relationship type has been successful for us, and we have continued to use the provides/requires pairs. It is likely that also peer connectivity would work for sharing WireGuard connection information.

```
name: hsscharm
...
peers:
  proxypeer:
    interface: proxypeer
requires:
  interfaces6a:
    interface: interfaces6a
```

Listing 4.4: Addition to Juju configuration file metadata.yaml for the HSS VNF to provide relation between the HSS and MME VNFs.

```
name: mmecharm
...
peers:
  proxypeer:
    interface: proxypeer
provides:
  interfaces6a:
    interface: interfaces6a
```

Listing 4.5: Addition to Juju configuration file metadata.yaml for the MME VNF to provide relation between the HSS and MME VNFs.

When adding the same interface name in *metadata.yaml* on both sides of the peer, the relationship can be established manually using the command *juju add-relationship <application 1> <application 2>*. To make OSM carry out the setup of relationship beyond *proxypeer* we need some additions to the NSD. Listing 4.6 shows the necessary configuration in the NSD to set up the relationship between the HSS and the MME. In the relation list the entity ID specifies the VNF. The endpoint is the common name between the two VNFs specified in the *interface* key in *metadata.yaml*.

```
# EPC_nsdwg.yaml

 nsd:
  nsd:
  - description: NS based on SimulaMet
    ↪ OpenAirInterface Evolved Packet Core NS
    df:
    - id: default-df
    ...
    ns-configuration:
      relation:
      - name: relation
        entities:
        - id: '1'
          endpoint: interfaces6a
        - id: '2'
          endpoint: interfaces6a
```

Listing 4.6: NSD relation configuration to provide a relation between the HSS and MME VNFs.

To send variables from one side of the relation to the other, we wait for the relationship to change. Initializing the relationship triggers both a *created* and *change* action. If the peers generate no new information, the on_<relation_name>_changed-action will not trigger after the initial phase. The Juju relationship establishes before the Day1-2 actions finish in our VNFs. We, therefore, need a variable we can regularly update to keep triggering the action until the dependencies of the WireGuard configuration finish.

For the Day1-2 actions, variables are inputted through the VNFD. We have not found a way to transfer variables from OSM to the relation actions. To transfer the variables between VNFs, we have stored the variables in files where the relation actions can pick them up. The variables can then be put on either of the two application sides at the VCA. In our code we transfer the variables to the application associated with the opposite VNF. The *wg-ready* variable is set to indicate when the peer has finished the WireGuard configuration and is ready to stop the keep-alive of the *relation-change* action. The code line *event.relation.data[self.model.unit][''wg-ready''] = ''True''* sets the data retrieved on VNF 1 on the VNF 2 application. VNF 2 can then read the data with the code line *event.relation.data[self.unit].get(''wg-ready'')*. Figures 4.6 and 4.7 show the bi-direction variable transfer as observed from the Juju units. The counter updates regularly to keep the relation

alive while waiting for the dependencies to finish. *wg-peer*, *wg-listenport*, *wg-subnet* and *wg-pubkey* provide the necessary information to add a WireGuard peer. The last user-defined variable in the figure is the *wg-peered* used to tell the peer that it has finished configuring the peer at its side. The other data variables in the figures are created by Juju.

```
- endpoint: interfaces6a
  related-endpoint: interfaces6a
  application-data: {}
  related-units:
    app-vnf-75c5acac01-z0/0:
      in-scope: true
      data:
        counter: "9629"
        egress-subnets: 10.192.251.248/32
        ingress-address: 10.192.251.248
        private-address: 10.192.251.248
        relation-joined: failed1
        wg-gwip: 192.168.8.129
        wg-listenport: "51820"
        wg-peered: "True"
        wg-pubkey: d986t+b2XimlLbQZz7guExK/ZqvIIltWiGCv8LGvRz4=
        wg-ready: "True"
        wg-subnet: 172.16.6.0/24
```

Figure 4.6: Variables from HSS peer at Juju unit for MME.

```
- endpoint: interfaces6a
  related-endpoint: interfaces6a
  application-data: {}
  related-units:
    app-vnf-16b32c0b5b-z0/0:
      in-scope: true
      data:
        counter: "4466"
        egress-subnets: 10.192.251.246/32
        ingress-address: 10.192.251.246
        private-address: 10.192.251.246
        relation-joined: failed1
        wg-gwip: 192.168.8.2
        wg-listenport: "51820"
        wg-peered: "True"
        wg-pubkey: J0J7CTfChxQf2CzbXqTBgxh4clBpryx3vbaPDFTQzwc=
        wg-ready: "True"
        wg-subnet: 172.16.6.0/24
```

Figure 4.7: Variables from MME peer at Juju unit for HSS.

To summarize will the user variables shown in Figures 4.6 and 4.7 establish the WireGuard peer. With the Juju relationship implementation, we have automatically established the information exchanged between VNFs.

4.6 Combining Elements

The next step after having functional NSs for WireGuard and OAI is to combine them and start to explore different aspects of OSM to extend the functionality. We

also want to test the setup's performance with and without WireGuard. In the first test, we want to add a WireGuard tunnel between two of the EPC components in OAI before extending to the rest of the interfaces. To route the application data into the WireGuard tunnel, we must change the network configuration. To make Cassandra, the HSS database available, we create a local dummy interface that is always reachable. Adding the new interface causes a change in the Juju charm of the HSS. To avoid changing the application configuration of the HSS and MME, we reuse the original IP address. Therefore, we must change the IP addresses for the tunnel external endpoints to avoid conflicts. Changes of the variables using the external IP addresses of the HSS and MME are done in the VNFD. Updates of the network specification in the NSD are also required. Using a subnet equal to the dummy IP address of the Cassandra database in the tunnel makes the S6a traffic go through the WireGuard tunnel.

In the Python code to translate actions in actions.yaml to Python functions we add *self.framework.observe(self.on.test_action, self.on_test_action)* in the *___init___* function of the proxy charm class. In the example *test* is the name of the action specified in the actions.yaml file in the charm folder with the first parameter *self.on.test_action* referring to the action. It is worth mentioning that the appended *_action* is not part of the name in the actions.yaml file. The second parameter *self.on_test_action* is the name of the Python function [OSMk].

The steps to add the VPN tunnel are summarized in the list below.

- Append actions for Wireguard in Jujus actions.yaml file. This is copied from the WireGuard NS.
- Copy the WireGuard functions from the WireGuard NS - generatekeys, config, wireguardup, addpeer, deletepeer and wgrestart.
- Add a *self.framework.observe* in the ___init___ function of the *HSSProxyCharm* class for all the WireGuard functions.
- Add Day-1 (initial) and Day-2 config-primitives in the VNFD

When deploying the NS we get the same VM instances in MicroStack as for the OAI NS. The virtual hardware configuration of the VNFs is equal to the information in Table 4.2. The MicroStack representation of the virtual network topology is shown in Figure 4.8 while Figure 4.9 shows how the VMs of the EPC and their corresponding information is represented in MicroStack. In Figure 4.8 the various networks are represented as colored posts and the VNFs as squares with a computer icon. In addition is, the virtual router providing connectivity to external networks, such as the Internet, represented as a square with arrows. The router and VNFs are connected to the network named *test*. For external access, the VNFs are then assigned floating IP addresses on the *external* network via their connection to the *test* network. The topology is the same for the NSs with and without WireGuard implemented but with

different IP addresses. In Figure 4.9, the name of the VMs is created by the name used in the instantiation with OSM, and the name and number the VNFs have in the NSD and VNFDs. Further information given in the figure is the OS, IP addresses for the different interfaces of the VNFs, resource flavor, and the state of operation for the VMs. A dropdown menu with several operations like assigning network interfaces manually and creating snapshots is on the right-hand side of Figure 4.9.

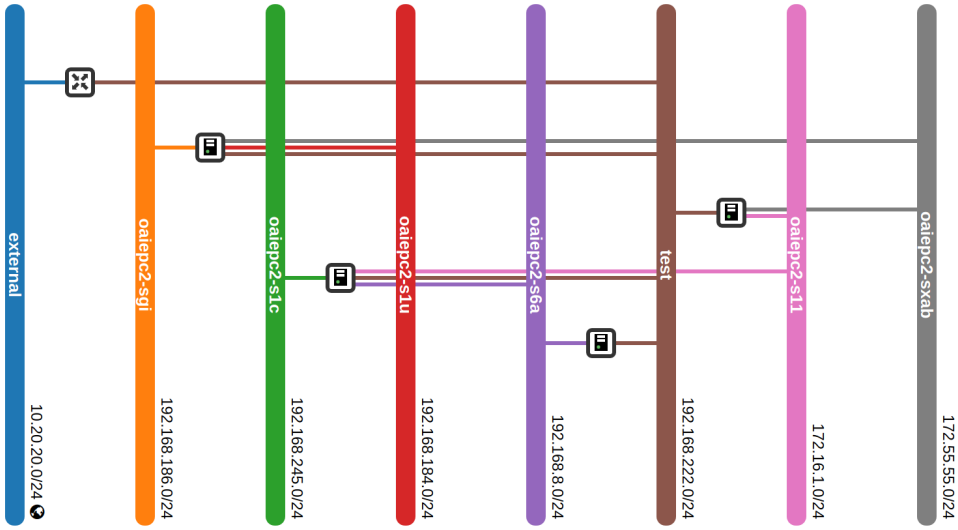

Figure 4.8: Network setup of the EPC NS in MicroStack.

4.7 Adding eNB and UE to the NS

To create a realistic environment for testing in a cellular network context we should include authentic traffic. With OAI-based implementation of eNB and UE we can create traffic using protocols used in actual cellular networks. To keep everything within the MicroStack environment we use a virtual OAI UE.

The eNB and UE are then attached to the EPC NS described in section 4.4. We add the eNB as a VNF and keep the UE as a VM unmanaged by OSM. When connecting the two components to the EPC, the setup corresponds with the intended architecture in Figure 4.1. Figure 4.10 shows the finished NS as represented in the OSM web GUI.[4]

[4]https://github.com/sondrki/TTM4905/tree/main/oai_eps_ns

☐	oaiepc2-4-spgwu-0	ubuntu18.04	**oaiepc2-sxab** 192.168.16.102 **oaiepc2-s1u** 192.168.9.159 **oaiepc2-sgi** 10.254.1.203 **test** 192.168.222.148, 10.21.21.163	spgwu-flv	-	Active	nova	None	Running	2 days, 18 hours	Create Snapshot
☐	oaiepc2-3-spgwc-0	ubuntu18.04	**oaiepc2-s11** 192.168.10.4 **test** 192.168.222.103, 10.21.21.170 **oaiepc2-sxab** 192.168.16.101	spgwc-flv	-	Active	nova	None	Running	2 days, 18 hours	Create Snapshot
☐	oaiepc2-2-mme-0	ubuntu18.04	**test** 192.168.222.32, 10.21.21.203 **oaiepc2-s6a** 192.168.8.2 **oaiepc2-s11** 192.168.10.2 **oaiepc2-s1c** 192.168.7.102	m1.medium	-	Active	nova	None	Running	2 days, 18 hours	Create Snapshot
☐	oaiepc2-1-HSS-0	ubuntu18.04	**test** 192.168.222.122, 10.21.21.171 **oaiepc2-s6a** 192.168.8.129	m1.large	-	Active	nova	None	Running	2 days, 18 hours	Create Snapshot

Figure 4.9: VNF instances of OAI EPC components in MicroStack with WireGuard tunnels implemented.

The reasoning behind adding only the eNB in the NS is that we will focus on the network connectivity in the core network. The user data from the UE will go through the S1-U interface via the eNB. In our implementation we will therefore add one eNB VNF to the NS to establish a WireGuard tunnel programmatically between the eNB and SPGW-U.

The eNB VNF is prepared using the SPGW-C charm and descriptors as a template. To fit the eNB application, we change the installation and configuration procedure in the charm. For the installation of the UE and eNB applications, we use equal commands. Listing 4.7 gives the used commands. The first command in the listing downloads the OAI project code while the following prepares and installs the application. The installation includes libraries used to simulate the air interface, Uu.

```
# commands use to install eNB and UE
git clone https://gitlab.eurecom.fr/oai/
    ↪ openairinterface5g.git
cd openairinterface5g/
source oaienv
cd cmake_targets/
```

```
./build_oai -I --phy_simulators
./build_oai -w SIMU --UE --eNB
```

Listing 4.7: eNB installation commands.

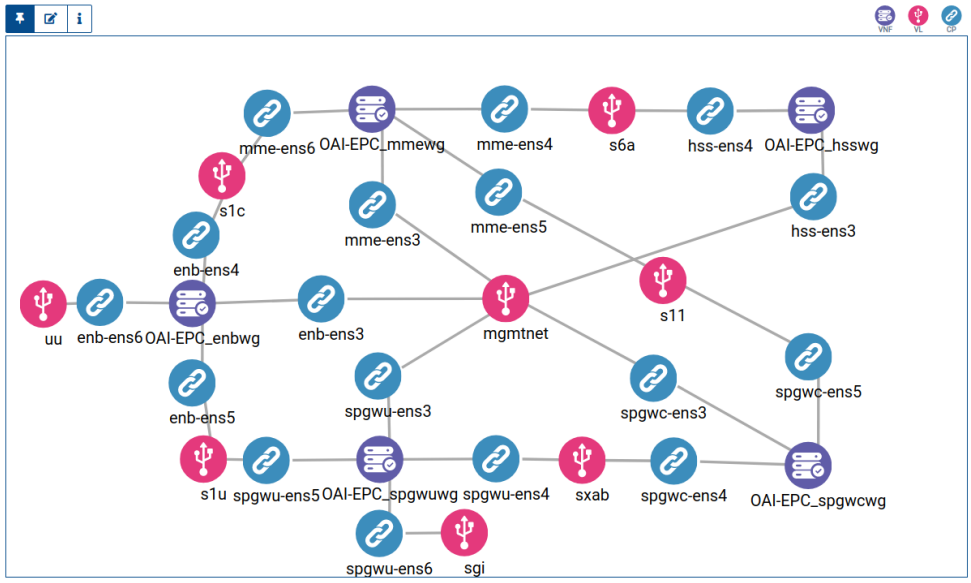

Figure 4.10: Architecture of the network shown in OSM web GUI.

```
cqlsh> SELECT imsi,key,mmehost,msisdn,opc FROM vhss.users_imsi;

 imsi            | key                              | mmehost     | msisdn          | opc
-----------------+----------------------------------+-------------+-----------------+----------------------------------
 208951234500808 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000808 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500932 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000932 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234501006 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880001006 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500732 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000732 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500862 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000862 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500541 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000541 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500426 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000426 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500872 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000872 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500940 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000940 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500677 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000677 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500929 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000929 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500581 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000581 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500019 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000019 | 9245cd6283cc53ce24ac1186a60dee6b
 208951234500013 | 449C4B91AEACD0ACE182CF3A5A72BFA1 | mme.ntnu.no | 20895880000013 | 9245cd6283cc53ce24ac1186a60dee6b
```

Figure 4.11: Subscribers in the HSS Cassandra database.

After installation we configure the UE and eNB accordingly to connect to the EPC. In the HSS installation the script named *data_provisioning_users* populates the database with default subscribers. Figure 4.11 shows the information for the first subscribers in the Cassandra database. To get appropriate authentication parameters,

we pick one of the available subscribers in the database. The MCC and MNC we have used are based on the default PLMN of OAI. To configure the UICC we need to specify the PLMN, MSIN, the keys K and OPc and MSISDN. This information is set in the file *ue_eurecom_test_sfr.conf* at the eNB. The complete file in our implementation is presented in appendix C. To replace the default configuration with the new, we run the command *conf2uedata*.

For the eNB we configure the PLMN, TAC and IP address for the MME. We also specify which network interfaces allocated for S1-U and S1-C. The TAC must be one of the allowed TACs in the file *mme.conf* at the MME. Unlike the UE we do all configuration in the Juju charm.

When included in the NS the eNB connects successfully to the EPC. Starting the eNB is performed in the Juju action responsible for the configuration. An excerpt of the log for the MME application is shown in Figure 4.12. The log shows a successful connection of the eNB seen as the connected eNB line in the statistics changes.

After deploying the new NS with the eNB we manually connect the UE. To connect the UE to the virtual network created for the Uu interface we use the *attach interface* action located under the *action* column in MicroStacks web GUI. After verifying the configuration we start the UE simulation with the command *RFSIMULATOR=<mme ip> ./lte-uesoftmodem -C 2685000000 -r 50 –rfsim*. We check that it connects to the EPC by reading the MME log. In Figure 4.13, the UE has successfully connected. In addition to the eNB connected as shown in Figure 4.12, Figure 4.13 presents one new UE that has been attached and connected since the last statistic update at the MME. Table 4.3 includes the resources for the NS described in this section and the UE VM.

Table 4.3: VNF information of OAI NS with RAN.

VNF name	OS	number of vCPU	amount of RAM (GB)	storage (GB)
HSS	ubuntu18.04	4	8.0	20
MME	ubuntu18.04	2	4.0	20
SPGWU	ubuntu18.04	1	3.0	20
SPGWC	ubuntu18.04	3	4.0	30
eNB	ubuntu18.04	4	8.0	20
UE	ubuntu18.04	2	4.0	20

After creating the NS without WireGuard tunnels between the VNFs we add the necessary configuration.[5] We use the approach described in Section 4.5 to create

[5]https://github.com/sondrki/TTM4905/tree/main/oai_eps_wg_nst

```
======================================== STATISTICS ========================================

               |   Current Status| Added since last display|  Removed since last display |
Connected eNBs |          0      |              0             |             0              |
Attached UEs   |          0      |              0             |             0              |
Connected UEs  |          0      |              0             |             0              |
Default Bearers|          0      |              0             |             0              |
S1-U Bearers   |          0      |              0             |             0              |

======================================== STATISTICS ========================================

Client association changed: 0
----------------------
SCTP Status:
assoc id .....: 2
state ........: 4
instrms ......: 2
outstrms .....: 2
fragmentation : 1312
pending data .: 0
unack data ...: 0
rwnd .........: 106496
peer info     :
    state ....: 2
    cwnd .....: 4380
    srtt .....: 0
    rto ......: 3000
    mtu ......: 1360
----------------------
New connection
----------------------
Local addresses:
    - [192.168.247.102]
----------------------
----------------------
Peer addresses:
    - [192.168.247.101]
----------------------
SCTP RETURNING!!
Create eNB context for assoc_id: 2
[2][49] Msg of length 59 received from port 36412, on stream 0, PPID 18
SCTP RETURNING!!
S1-Setup-Request macroENB_ID.size 3 (should be 20)
New s1 setup request incoming from macro eNB id: 00e00ESC[0m
g eNB to the list of served eNBs
Adding eNB id 3584 to the list of served eNBs
[49][2] Sending buffer 0x7ff2f800a330 of 27 bytes on stream 0 with ppid 18
Successfully sent 27 bytes on stream 0
SCTP RETURNING!!
======================================== STATISTICS ========================================

               |   Current Status| Added since last display|  Removed since last display |
Connected eNBs |          1      |              1             |             0              |
Attached UEs   |          0      |              0             |             0              |
Connected UEs  |          0      |              0             |             0              |
Default Bearers|          0      |              0             |             0              |
S1-U Bearers   |          0      |              0             |             0              |
```

Figure 4.12: Logs on MME showing successful connection of eNB.

automatic setup of the VPN tunnels. This procedure confirms full secured service automation provisioning by integrating WireGuard capability in OSM. Relations

```
========================================== STATISTICS ==========================================

                |   Current Status|  Added since last display|   Removed since last display |
Connected eNBs  |          1      |              0           |              0               |
Attached UEs    |          1      |              1           |              0               |
Connected UEs   |          1      |              1           |              0               |
Default Bearers|          0      |              0           |              0               |
S1-U Bearers    |          0      |              0           |              0               |

========================================== STATISTICS ==========================================
```

Figure 4.13: UE and eNB connected to the EPC.

can be seen in OSM after an NS is created. Figure 4.14 shows how the relations in the EPS NS with WireGuard tunnels is presented in OSM. Both the *proxypeer* and arbirary relations are listed in the figure for the different Juju units that are created for the NS.

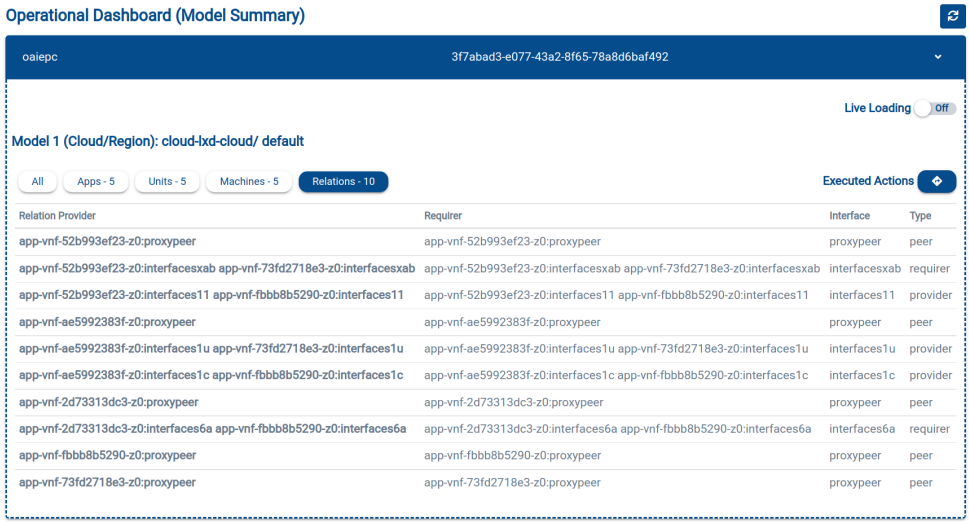

Figure 4.14: Relations in OAI EPC with WireGuard tunnel between components as seen from the OSM web GUI.

The resources for the OAI EPS NS is the same both with and without WireGuard tunnels implemented. Therefore, Table 4.3 also refers to the resources allocated for the VNFs and UE for the NS described in this section with WireGuard tunnels implemented.

4.8 Double Resources

To check how the designated resources affect our results, we create a NS with the double amount of vCPU and RAM of the OAI EPS NS. Except for the change of resources, the implementation of the NS is equal to the NS at end-state in Section 4.7.[6] The measurements on this NS will help verify the other results and show eventual major changes in performance when the resources change. The resources for the VNFs are given in Table 4.4. As earlier, the UE is not included in the NSD. We have therefore kept the resources for the UE.

Table 4.4: VNF information of OAI NS with RAN, doubled resources and WireGuard connection between components.

VNF name	OS	number of vCPU	amount of RAM (GB)	storage (GB)
HSS	ubuntu18.04	8	16.0	20
MME	ubuntu18.04	4	8.0	20
SPGWU	ubuntu18.04	2	6.0	20
SPGWC	ubuntu18.04	6	8.0	30
eNB	ubuntu18.04	8	16.0	20
UE	ubuntu18.04	2	4.0	20

4.9 Network Slice Template

In the last development step of the thesis we add the EPS NS into a NST.[7] The NSTs consist of a single NS, the OAI EPS NS with WireGuard described in section 4.7. The resources are kept equal to Table 4.3.

The *slice-service-type* key in the NST is only descriptive indicating the use case type of the network slice [OSMe]. However, the *quality-of-service* key is related to the 5QI describing characteristics of the traffic flow [ETS20]. However, to actually implement priority and quality assurance for the NSI, we also need to specify other parameters like the Packet Delay Budget (PDB) to build the QoS profile[OSMg]. To observe how this affects our NS we make two NSTs. The first NST as a URLLC slice type with ID 3. For the second NST, we use eMBB as the slice type with *quality-of-service* ID 6. In addition is the QoS parameters in Listing 4.8 used in the NST.

```
# eMBB NST:
    quality-of-service:
        id: 6
```

[6] https://github.com/sondrki/TTM4905/tree/main/oai_eps_wg_double
[7] https://github.com/sondrki/TTM4905/tree/main/oai_eps_wg_nst

```
        resource-type: GBR
        priority-level: 2
        packet-delay-budget: 100 #ms
        default-max-data-burst: 1354

# URLLC NST:
    quality-of-service:
        id: 3
        resource-type: delay-critical-GBR
        priority-level: 1
        packet-delay-budget: 1 #ms
```

Listing 4.8: QoS parameters for the NSIs.

To create the NSTs we first configure connection points in the NSD that should be accessible at NST level. The Service Access Point Descriptor (SAPD) connection points are used to provide network(s) from the NS available as management or shared network(s) between NSs in the NST. Listing 4.9 includes the necessary configuration to enable the management network, *mgmtnet*, in the NS as an external management network for the NST. In the listing the SAPD is exposed to allow for data transfer externally via the NSI to the NS and further to the VNFs [ETSa]. The *is-shared-nss* key specifies if the subnet is shareable between NSIs. In our case it is set to *false* because we only use one NS in our NST.

To deploy the NST we change NS with NSI and NST in the instantiation, resulting in the following command, *osm nsi-create --nsi_name oaiepc_nst --nst_name oai_epcwg_embb_nst --vim_account a2ntnu_microstack*. The configuration file parameter used for the NS implementation are not used as the parameters are inserted directly into the NST.

```
# NS descriptor
    sapd:
    - id: mgmtcp
      virtual-link-desc: mgmtnet

# NST descriptor
    netslice-subnet:
    -   id: OAI-EPC_nsdwg
        is-shared-nss: false
        description: network slice template for OAI
            ↪ EPC with WireGuard
        nsd-ref: OAI-EPC_nsdwg
```

```
...
    netslice-vld:
    -   id: mgmtcp
        mgmt-network: true
        name: mgmtcp
        nss-connection-point-ref:
        -   nsd-connection-point-ref: mgmtcp
            nss-ref: OAI-EPC_nsdwg
        type: ELAN
```

Listing 4.9: Connection points shared between NS and NST.

4.10 Multi-site Deployment

The OSM guide [OSMe] gives an example on how to deploy a NS on different VIMs. The guide is written for an older version of OSM, but OSM version 10 can use the same method. The interfaces between VNFs need to be accessible from both VIMs. In our case we need to change the internal network interfaces between VNFs to be management interfaces. For testing purposes we split out the HSS to a separate VIM from the other VNFs. To have a working S6a interface we configure the S6a network in the NSD to use management network instead of creating an NS internal network. In the instantiating of the NS we specify where the different VNFs will be created.[8] List 4.10 shows this with the HSS, *member-vnf-index: "1"*, being created at vim *microstack* while the other VNFs are created on the VIM named *a2ntnu_microstack*.

```
wim_account: False
vnf:
  - member-vnf-index: "1"
    vim_account: microstack
  - member-vnf-index: "2"
    vim_account: a2ntnu_microstack
  - member-vnf-index: "3"
    vim_account: a2ntnu_microstack
  - member-vnf-index: "4"
    vim_account: a2ntnu_microstack
  - member-vnf-index: "5"
    vim_account: a2ntnu_microstack
    ...
```

Listing 4.10: Additional instantiating parameters for the multi-site deployment.

[8] https://github.com/sondrki/TTM4905/tree/main/oai_eps_multisite

The command for deployment a multi-site NS is similar to the creation of single site NSs. Only one VIM account is specified, being the primary one. We ran the command, *osm ns−create −−ns_name oaiepc_multivim2 −−nsd_name OAI−EPC_nsdwg_multi −−vim_account a2ntnu_microstack −−config_file paramswg_multivim.yaml* to deploy the multi-site NS. The VNFs' resource information is given in Table 4.5.

To set up a WireGuard peer configuration, we use the endpoint IP address of the peer. In the multi-site deployment, this is the management interface. For OpenStack and MicroStack, the management interfaces use Dynamic Host Configuration Protocol (DHCP) to assign IP addresses dynamically. To get the automatic WireGuard establishment, we need to get the dynamically allocated addresses transferred to the peer. The provided charm code obtains the management IP address and uses it in the setup if a *gateway IP* is not specified.

Table 4.5: VNF information of multi-site OAI NS with RAN and WireGuard connection between components.

VNF name (VIM)	OS	number of vCPU	amount of RAM (GB)	storage (GB)
HSS (microstack)	ubuntu18.04	4	8.0	20
MME (a2ntnu_microstack)	ubuntu18.04	2	4.0	20
SPGWU (a2ntnu_microstack)	ubuntu18.04	1	3.0	20
SPGWC (a2ntnu_microstack)	ubuntu18.04	3	4.0	30
eNB (a2ntnu_microstack)	ubuntu18.04	4	8.0	20

Chapter 5
Results and Observations

The following chapter will present our results when testing the NSs and NSIs we have built. The results commented on in this section are the main findings regarding how introducing WireGuard affects the performance and networking between VNFs. The complete outputs from the performance tests are available in appendix F. Since we utilize the results for observation and not benchmark purposes, we used the results as-is from the measurements tools.

5.1 Network Traffic

In this section we will present the results regarding the network flow with and without a WireGuard tunnel between components.

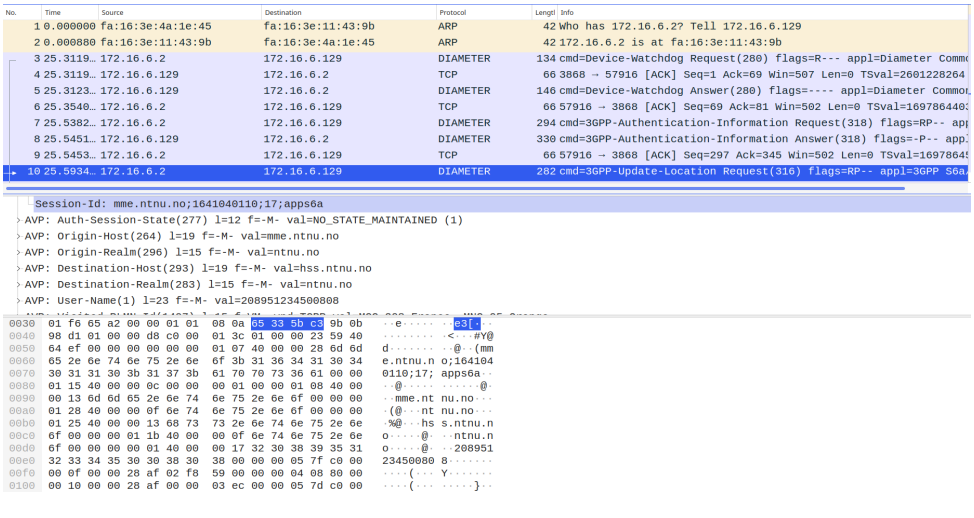

Figure 5.1: Traffic on S6a interface without VPN.

In Figure 5.1 we have done a tcpdump at the MicroStack server. Since the VIM creates the virtual networks in the NSs, it is also able to listen for traffic going in the same networks. In the figure, we see a screenshot from Wireshark. The upper third in the figure shows seven packets of different protocols going over the S6a interface. The middle and the bottom section show the information in one of the packets using the DIAMETER protocol. The IMSI, hostnames and realms for the MME and HSS are information presented in the packet as shown in the bottom section of the figure. While doing the tcpdump we used the NS with the EPS without WireGuard, described in Section 4.7.

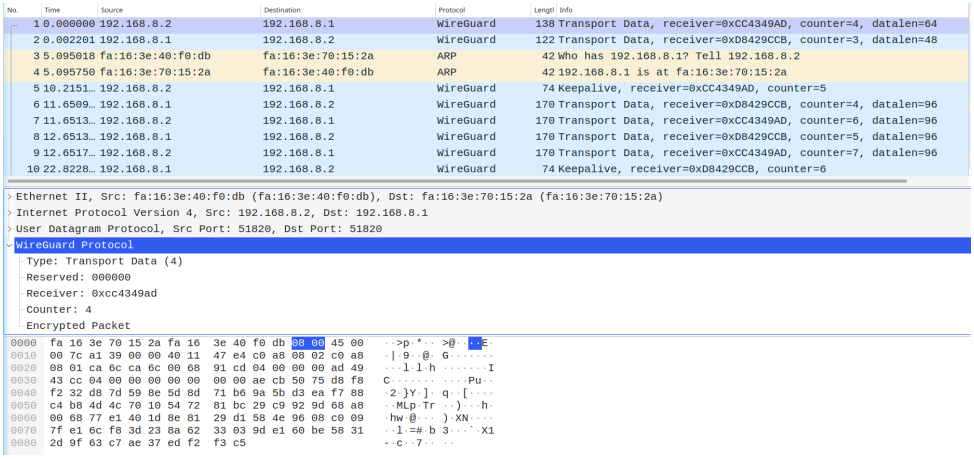

Figure 5.2: Traffic on S6a interface with WireGuard deployed.

In Figure 5.2 we repeat listening with tcpdump on the S6a interface. However, we deploy a new NS to introduce WireGuard tunneling in the EPS. The new NS is the same as described in Section 4.7 with WireGuard connectivity. Different from Figure 5.1 we no longer see the DIAMETER protocol with its clear text data. All data except link layer discovery ARP messages are packed and transferred inside the WireGuard tunnel, as shown in the upper figure's section.

5.2 Performance Monitoring

Gnocchi, being the component in OpenStack handling the telemetry, is not included in MicroStack [Foua]. The inbuilt monitoring functionality at VIM level in OSM can therefore not be used together with MicroStack out of the box. Juju is able to monitor on VNF and VDU level through the VCA. However, monitoring measurements with the VCA only happens every five minutes. We have also had issues setting up this

monitoring. Therefore, we have manually monitored the system load in our tests when encountering unexpected results.

5.2.1 Throughput

When testing the throughput, we did a combination of executing iPerf3 with and without specified MTU. In our deployments, we have seen that WireGuard sets the MTU size to 1362 bytes for the tunnel interfaces. iPerf3 defaults to an MTU size of 1500 bytes. However, iPerf3 can, for several occasions, automatically discover and use the MTU size of an interface. Automatic discovery works in our VNFs. The MTU size has therefore been the same for both approaches and has not affected the results.

iPerf3 can be used to measure high-performance environments. For connectivity between different VNFs, we have measured stable throughput at around 20 Gbps with some exceptions at around 14 Gbps. For instance, it is the measurement for the S6a interface in the NS with double resources for the VNFs averaging to 14.6 Gbps. Based on our measurements, the internal networking of the MicroStack instance is around 20 Gbps, with peak performance up to 30 Gbps. The average throughput has been consistent for the unencrypted connection comparing the measurements with default MTU size of 1500 bytes and when reducing to 1362 bytes. Other applications out of our control that are using the VIMs resources may be one reason why the bandwidth variates. Internal iPerf3 mechanisms could be another reason. However, the raw throughput is still high enough to affect the WireGuard throughput significantly.

The difference in throughput for the S6a, S1-U, and S1-C interfaces when using WireGuard is notable. The S1-U and S1-C measurements are similar. However, the throughput measurements for the S6a interface are notably lower. All tests have consistently shown the same, leading us toward resource allocation and other loads in the VNF as a possible reason. The HSS is the component with the most assigned resources, but from the implementation, we know that the Cassandra database can be resource-demanding. When checking the performance in the MME and HSS VNFs after the throughput test, we find the resource consumption being higher than for the other components, but still under 1 % load over the last 5 minutes of test time for iPerf3.

Table 5.1 summarizes the throughput measurements for the different deployments with WireGuard. The *OAI EPS NS* with and without WireGuard is described in Section 4.9. Further, is the NS named *Double resources* described in Section 4.8. The NST for the URLLC and eMBB network slices are described in Section 4.9. We have tested with workload on the NS separately and simultaneously for the same interface with the suffix respectively being *separate* and *simultaneous* in the table.

Table 5.1: Throughput measurements over WireGuard for different NSs and VNFs.

NS name/ description	**S6a** (bps)	**S1-C** (bps)	**S1-U** (bps)	**UE connectivity** (bps)
OAI EPS NS without WireGuard	21.7G			1.61M
Double resources without WireGuard	(14.6, 19.2)G		18.7G	
OAI EPS NS with WireGuard	918M	1.02G	(1.43, 1.44)G	1.65M
Double resources	1.07G	1.46G	2.19G	1.70M
URLLC slice separate	811M	1.12G	1.43G	
eMBB slice separate	(767, 1050)M	1.05G	1.48G	
URLLC slice simultaneous	(798, 1050)M	973M	1.41G	
eMBB slice simultaneous	(823, 1060)M	993M	1.47G	

When doubling the resources, the measured throughput for the WireGuard tunnels is consequently higher. For example, over the S6a interface, the throughput ranges between 800 and 900 Mbps to 1.1 Gbps observed over 10 minutes for the NS with similar and double resources as in Table 4.1. For the S1-U interface, we observe the same. After doubling the resources, the observed throughput over the S1-U interface changes from between 1.4 Gbps and 1.5 Gbps to 2.2 Gbps.

The UE application provides significantly lower results in the case of throughput. The NSs we have used in testing the UE connectivity are the EPS NSs with and without WireGuard, described in Section 4.7 and the double resources NS described in Section 4.8. The throughput is stable between 1.6 Mbps and 1.7 Mbps for all three test scenarios with E2E connectivity to the external interface on the SPGW-U. 1.7 Mbps is lower than the measurements of Dreibholz in [Dre20], but still representative compared to the observations we have made for the other interfaces. The Uu interface is, therefore, the bottleneck of the lab environment.

When comparing the throughput between the NSIs, we have not seen any major differences. For the S1-U, S6a, and S1-C interfaces of the first deployment of the eMBB slice while testing the URLLC network slice simultaneously, we measured throughput in the WireGuard tunnel as 1.47 Gbps, 0.82 Gbps, and 0.99 Gbps. For the URLLC network slice over the same interfaces were observed to be 1.41 Gbps, 0.80 Gbps, and 0.97 Gbps.

The difference between putting load from iPerf3 on one NSI at a time versus both simultaneously gives some slight differences. For example, we measured an average of 0.77 Gbps on the S6a interface for the eMBB slice running the measurements alone, slightly lower than the 0.82 Gbps when co-running with the URLLC slice. For the URLLC network slice, we observed an average of 0.81 Gbps for the separate measurement and 0.80 Gbps simultaneous with the eMBB slice. However, we ran the measurements two times. The second time with a new deployment of the NSTs. In the second test, we measured higher throughput with around 1.05 Gbps for both slices over the S6a interface with WireGuard. The difference between running one NSI alone and simultaneous with another is minor. The minor differences may also be dependent on the NSI. However, we have not measured with throughput totaling close to the internal networking performance of the MicroStack server, which may give other results.

5.2.2 Latency

The average latency approximately doubles when comparing the EPS NS with and without WireGuard. For the S6a and S1-C with and without WireGuard in the NS described in Section 4.7, the factor is 2.12 for both interfaces, $0.805\,ms/0.379\,ms = 2.12$ and $0.881\,ms/0.416\,ms = 2.12$. In the NS with double resources the observed factors are $0.960\,ms/0.041\,ms = 23.41$ on the S6a interface and $0.946\,ms/0.393\,ms = 2.41$ on the S1-C interface. Another way of viewing the data is that the average extra delay is around 600 ms, $((0.805\,ms - 0.379\,ms) + (0.881\,ms - 0.416\,ms) + (0.960\,ms - 0.041\,ms) + (0.946\,ms - 0.393\,ms))/4 = 0.591\,ms$. However, the average latency on both interfaces with WireGuard is higher with more resources. We expected lower latency when doubling the resources as seen for the measurement without WireGuard. We have observed differences from one deployment to another for other measurements. Differences between deployments could also be the case here. We nevertheless observe that all measurements are lower than 1 ms. Latency measurements for the different deployments are gathered in Table 5.2.

When measuring the latency in the NSIs, the latency drops compared to the NSs initialized without the NST superstructure. For example, the observed latency on the S1-C interface goes from 0.881 ms to 0.819 ms for the eMBB slice and 0.836 ms for the URLLC network slice. For the S1-U interface, we observe the same going from 0.784 ms to 0.667 ms and 0.709 ms in the NSIs. Finally, on the S6a interface, the latency is similar with 0.805 ms compared to 0.857 ms and 0.778 ms.

Table 5.2: Latency measurements over WireGuard for different NSs and VNFs.

NS name/ description	**S6a** (ms)	**S1-C** (ms)	**S1-U** (ms)
OAI EPS NS without WireGuard	0.379	0.416	0.631
Double resources without WireGuard	0.041	0.393	0.351
OAI EPS NS with WireGuard	0.805	0.881	0.784
Double resources	0.960	0.946	0.889
URLLC slice separate	0.843	0.837	0.733
eMBB slice separate	0.874	0.746	0.737
URLLC slice simultaneous	0.778	0.836	0.709
eMBB slice simultaneous	0.857	0.819	0.667

5.2.3 Service Response Time

To check if introducing WireGuard affects the service response time in the EPS, we connect the UE to the EPS. We have seen from the other observations that the Uu interface is our bottleneck. Therefore, we observe the forwarded data on the MME to find out if the response time for the HSS changes. Wireshark has built-in functionality of measuring Service Response Time (SRT) of the Diameter protocol. In Figures 5.3 and 5.4, ten successful registrations of the UE to the network are done. Figure 5.5 shows one additional registration where the connection on the Uu interface timed out, with a total of eleven connections to the HSS.

Index	Procedure	Calls	Min SRT (s)	Max SRT (s)	Avg SRT (s)	Sum SRT (s)
1	3GPP-Authentication-Information	10	0.005091	0.009857	0.006156	0.061556
2	3GPP-Update-Location	10	0.003518	0.005951	0.004519	0.045192

Figure 5.3: Diameter SRT statistics on MME without WireGuard tunneling.

Index	Procedure	Calls	Min SRT (s)	Max SRT (s)	Avg SRT (s)	Sum SRT (s)
2	3GPP-Authentication-Information	10	0.003545	0.006168	0.005377	0.053769
3	3GPP-Update-Location	10	0.004153	0.006068	0.004904	0.049039
1	Capabilities-Exchange	1	0.029413	0.029413	0.029413	0.029413
4	Device-Watchdog	1	0.000925	0.000925	0.000925	0.000925

Figure 5.4: Diameter SRT statistics on MME with WireGuard tunneling.

Index	Procedure	Calls	Min SRT (s)	Max SRT (s)	Avg SRT (s)	Sum SRT (s)
2	3GPP-Authentication-Information	11	0.004686	0.006832	0.005607	0.061680
3	3GPP-Update-Location	11	0.004011	0.007922	0.004954	0.054491
1	Device-Watchdog	5	0.000510	0.004034	0.001741	0.008706

Figure 5.5: Diameter SRT statistics on MME without WireGuard tunneling - double resource NS.

The SRT observed on the MME is totaling the network latency to and from the HSS and the response time of the HSS application. The average SRT for both NSs with WireGuard connectivity have a lower average value than the NS without VPN tunnel. The SRT values for the different NSs are shown in Figure 5.4 for the OAI EPS NS, Figure 5.5 for the NS with double resources, and Figure 5.3 for the NS without WireGuard. The latter is averaging on an SRT of 6.156 ms. The SRT is also higher for the NS with double resources which is 5.607 ms compared to 5.377 ms for the EPS NS with resources as presented in Table 4.3. This indicates that the SRT for the HSS varies too much on a small number of connections. However, testing with more attachments is ineffective in our lab environment because of manual set up and tear down of the UE. With the attachments we have done, we have not seen any negative effect on the SRT with and without WireGuard.

5.3 Measurements of the Multi-site NS

As mentioned in chapter 2, the two VIMs connect with a WireGuard tunnel over a bandwidth of 200 Mbps. We have therefore reduced the MTU size for our throughput test with iPerf3 to $1500\,B - (1500\,B - 1346\,B) * 2 = 1224\,bytes$.

Both the throughput and latency are affected by the additional WireGuard tunnel. In general, the latency with the multi-site setup is significantly higher than with our other measurements. We also observe a higher average time difference for our latency test. The difference is $19.769\,ms - 18.355\,ms = 1.414\,ms$ with a factor of $19.769\,ms/18.355\,ms = 1.01$. However, the deviation and the maximum response time are higher for WireGuard measurements than single-site observations. A more dedicated connection might therefore give more stable results. For the throughput tests, we observe a reduction of $179\,Mbps - 156\,Mbps = 23\,Mbps$ on average compared to the S6a throughput on the single-site EPS NS with WireGuard. The variance is $161\,Mbps - 145\,Mbps = 16\,Mbps$ for the nested WireGuard measurement with 60 seconds intervals averaging on 156 Mbps. The variance is significantly higher than the variance of $180\,Mbps - 175\,Mbps = 5\,Mbps$ for the single-site WireGuard tunnel. Therefore, we recommend a more controlled lab environment for more stable

measurements, both throughput, and latency-wise. However, because of available VIMs' resources, we have concentrated on how we can use OSM to deploy a multi-site NS with WireGuard and not focused on optimizing the performance tests in this thesis.

A difference with the multi-site throughput measurement compared to the single-site tests is the location we run the iPerf3 server. Because we were not able to connect with the iPerf3 server using the floating IP address of a VNF we ran the iPerf3 server on it. Running the iPerf3 measurement against the microstack VM and not a VNF causes lack of routing and forwarding from the VIM to the VNF on one of the sides of the connected VNFs. However, based on the other scenarios that we have done measurements, we believe that the possible missing latency and reduced throughput is negligible.

Chapter 6
Discussion

In the following sections, we will give our thoughts on the setup, development process, and results given in Chapter 5. During the development process in the thesis, we have gained knowledge on the process of working with OSM and WireGuard. Firstly, we will share our thoughts on the development of WireGuard using OSM. Then, we will discuss how lifecycle of VNFs with WireGuard as a VPNaaS can be done. Lastly, we will discuss the results of our performance measurements.

6.1 Charms in OSM

To answer the research question of how OSM can support WireGuard as a VPNaaS, we have made practical tasks with OSM and Juju charms. Ease of charm development and modification is subjective, and it depends on the developer's skills. To make the development process more accessible, the OSM community have an extensive repository of examples. However, several deprecated VNF packages and examples may give a higher threshold for reusing charms and doing charm development in OSM. For instance, the transition from SOL005 to SOL006 in this thesis was an error-prone task. OSM provides a tool for translating older descriptors that we have not used in this thesis. Using the transition tool could probably ease some of the work. However, updated examples would still be a good addition.

As stated in section 4.7, moving network traffic to go inside the VPN tunnel requires changes to connection interfaces and charms. To ease the creation of new NF the developer should be aware of the necessary changes when introducing a VPN. On the one hand, making the interface available outside of the tunnel reveals the traffic as seen in Figure 5.1. On the other hand, not making the interface available may cause issues for applications. For instance, waiting for the WireGuard configuration to finish may cause the application to fail if it needs the WireGuard interface to be available. Therefore, before deployment to a production environment, we recommend proper testing when introducing VPNaaS.

One way to combine the advantages of both keeping the configuration as is and adding VPN functionality to provide confidentiality or network traversal is to have a WireGuard VDU. The VDU could either be a separate VNF or a VDU connected through an internal network. The WireGuard VDU could then act as a proxy to other NFs. However, including a VDU brings further complexity in other ways by adding routes and connectivity through the WireGuard proxy. Probably, the application VDU(s) can be as it is if the WireGuard VDU performs most of the routing. Appending a VDU will also require additional resources fixed to the new VDU. A separate VDU for WireGuard differs from our approach where WireGuard uses the resources in the application VDU. As we have focused on adding WireGuard to the existing architecture of the NFs, testing a dedicated WireGuard VDU has not been in the scope of our development.

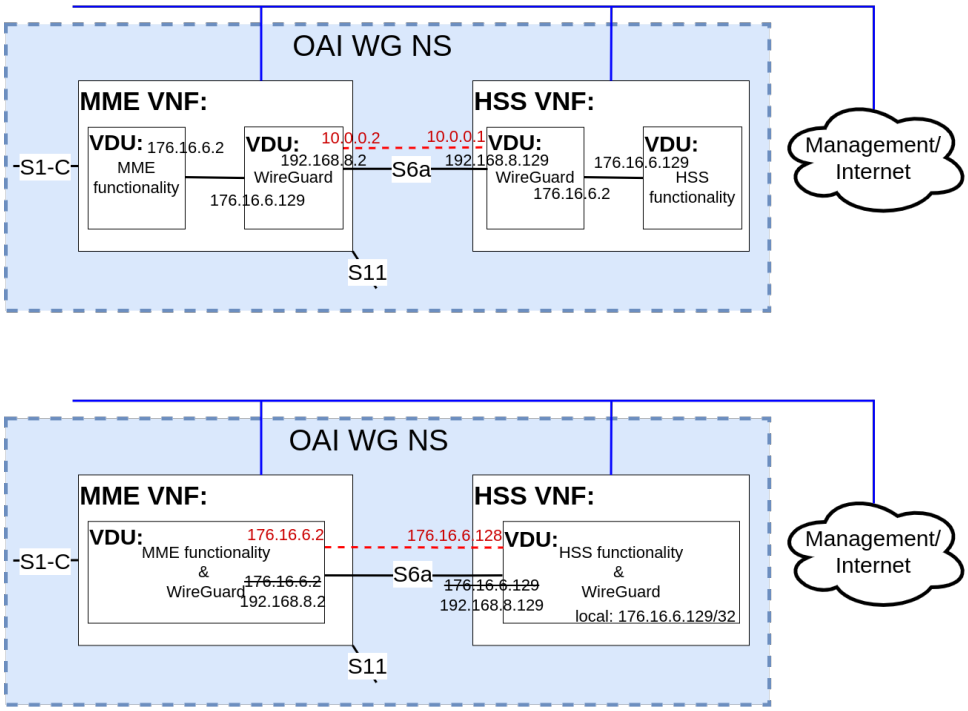

Figure 6.1: Comparison of WireGuard implemented in a single VDU and a multi-VDU VNF with a dedicated WireGuard VDU.

Figure 6.1 shows how a multi-VDU setup might look like. The encrypted tunnel and belonging configuration are displayed in red. Further, the unencrypted connection interfaces are in black, and the VNF management network is in blue. The figure focuses on two of the VNFs we have used in our NS to demonstrate the idea, the

MME and the HSS. We have then extended the same architecture to the other interfaces. A possible solution for a dedicated WireGuard VDU can be masquerading the IP address of the VNF application. This is illustrated in the upper NS in Figure 6.1. The WireGuard VDUs in the figures imitate the application IP addresses for the opposite VNF. In that way the MME and HSS application get answer from the expected IP address. At the same time, the network traffic gets routed into the VPN tunnel. In comparison is the single VDU solution we have used in this thesis shown in the bottom *OAI WG NS* of Figure 6.1. For the solution used in this thesis, we have changed the internal VDU to enable the WireGuard tunnel. The complexity of routing is therefore added directly in the charm actions.

We believe that both techniques described above to provide WireGuard as a VPNaaS will work. However, the multi-VDU setup must be tested by anyone who will follow this approach. The developer must also change the VNFDs and NSDs in both approaches. However, we believe the changes will arguably be larger in the multi-VDU setup. Since both approaches require additional configuration, we see the single-VDU as the simplest solution to implement. However, depending on the charm and potential changes to it, there might be applications where the multi-VDU solution suits better than VNFs or CNFs with a single VDU.

6.2 Lifecycle for a VPNaaS

Using Juju relations with OSM supplies a way to transfer information across VNFs. We have shown that setting up WireGuard peers with few to no manual steps is possible with Juju relations in OSM. To be used as a service, it should be easy for a developer to add WireGuard to VNFs. With our work, we have some requirements to add WireGuard to a VNF. The first requirement is the use of proxy charms. For native charms, the VCA dealing with Juju relations is not presented. A generalized solution that may work for both proxy and native charms is using a separate Key Management System (KMS). For native charms, the distribution of other information like the gateway IP address must then use a similar distributed system or manual input. The second requirement is the relation between two VNFs that must be in the form of a provides/requires pair. We have also experienced some instability regarding the first-time connection of the WireGuard tunnel. A restart of the interface may therefore be needed.

WireGuard supports peer mobility. However, the initial connection uses the endpoint where the peer is accessible. We use pre-defined IP addresses to set up the peer connection in our implementation. The VMs in MicroStack do not know their associated floating IP addresses at OS level. For a multi-site deployment with floating IP addresses, the VNF should have a way to transfer its external addresses to its peer. The OSM, VIM, and VCA triplet can retrieve the management address

of a VNF to perform its actions. The Juju VCA obtains the floating IP address of the management interface using *self.model.config["ssh-hostname"]* in the VCA *proxypeer* connection where *self* is inherited from the *SSHProxyCharm* library. We can then use the management interfaces as connection points between VNFs. The provided code already does this. However, the code in a more complex network setup adjustments is needed to add additional interfaces with WireGuard tunnel.

We have faced several challenges in our experiments regarding exception handling when actions from other tasks have not yet been finished. Day1-2 Juju actions run sequentially. An action can then be dependent on a prior. However, we have not found a way to stack cloud-init, Juju relations, and regular Juju actions in the same way. Because the Juju actions depend on the initial cloud-init setup of username, password, and SSH access cloud-init always starts its Day-0 tasks first. Juju Day1-2 actions can then begin after the basic cloud-init tasks finish. However, when running package updates with cloud-init, we have seen that the package manager is not ready when we need it in the charms. Day-1 actions depending on the packet manager should therefore use it cautiously or add additional functionality to wait for cloud-init.

Further, we have not found a way to synchronize Day1-2 with relation-charm actions. If the relation-charm depends on tasks finished from the Day-1 operations, we need additional logic in the relation-charm function. For instance, is the relation-changed function in our implementation depending on WireGuard files created during Day-1 actions. For WireGuard as a VPNaaS, moving the functionality in the Day-1 initial configuration primitives to the relation function will reduce the exchange back and forth between VNFs. The sending back and forth using a counter is introduced to trigger the *relation-change* action regularly. While waiting for dependencies to finish, the relationship is typically not changed when needed. The WireGuard peering will then never start. To keep the *relation-change* action running when the dependencies are complete, we need a way to keep the relation exchanging data to keep it up. Our way to solve this has been to use a counter on both sides that regularly updates to keep the connection active. Even though this works, we would prefer a more stable, inbuilt functionality to synchronize the tasks and start the *relation-change* action at our signal.

A way to bypass the dependencies of Day-1 charms could be to input parameters into the relation function. However, we have not found a way to input these similarly to the other charms. Instead, we have stored the parameter input from Day-1 actions in files. Then, we can read the files to pick up the parameters in the relation-change function and transmit the parameters to the other peer. From our perspective, deeper integration of Juju relations in OSM could help with both synchronization and parameter inputs.

With the tests we have performed with our code, we suggest deploying WireGuard as a VPNaaS with the steps described in the list below. Appendix E shows the full additional configuration needed for the different stages.

1. Add installation of WireGuard package in cloud-init.
2. Add actions in the charm file, actions.yaml.
3. Add relations in metadata.yaml.
4. Include the Python code in appendix D. Be sure to change the name of the relation both in the *___init___* function where relation in and in the Python function itself. *Relation* in *interface_relation_changed* should be the name of the relation.
5. Append actions to the VNFD. If the VPN tunnel should be created as a Day-1 operation the actions should be in *initial-config-primitive* in the *day1-2* section. For actions intended to be run as Day-2 actions actions should be in the *config-primitive* section of the *day1-2* section.
6. Day-2 actions such as further configuration and maintenance can be done using Day-2 operations in OSM. Relevant Day-2 avtions can be to restart WireGuard, add or delete peers.

The few steps for adding WireGuard in the VNFD to establish connection between two VNFs demonstrate the use of WireGuard as a VPNaaS. However, there are possibilities to make the code and process smoother. Introducing a KMS and further testing of the provided code we have developed are potential ways to make a more robust solution. The PoC demonstrated in this thesis confirms that OSM consists of tools that enable an automated way of making the WireGuard tunnel ready in complex and realistic VNFs.

6.3 Key Management

Key distribution is often a topic that requires extra attention when setting up a VPN. Manually transferring keys between services requires no extra logic but can be a tedious job when setting up several VPN tunnels. The EPS NSs used in this thesis is one example of a network with multiple connections. Manually peering all the interfaces in the NS will be time-consuming. While waiting for a tenant manager to set up the tunnels, the cellular network does not provide any service to a UE because of the dependencies between components. The complexity and amount of workload can lead to the tenant manager skipping VPN tunneling on parts or the whole NS. A KMS, on the other hand, will provide a central repository of keys. OSM

does not provide a KMS function. The manager must therefore provide functionality outside the OSM framework. This thesis has solved the key management with a non-standard third option. Using Juju relations, we can transfer the public keys between the peers of the WireGuard connection. In this way, we can create new keys for every instantiation of a VNF. Using the Juju relation approach requires the use of proxy charms. For native charms, both manual transfer and using a KMS are possible alternatives.

The SSH connection between the VNFs and VCA should provide confidentiality of the keys and metadata transfer of the Juju peer. However, the provider manager controlling the VCA could get access to this information. The provider manager will, in that case, already have credentials to the VNFs via the access to the Juju units. Therefore, the provider manager can retrieve this information and the private key. An adversary without access to the VCA may obtain the public key and external connection points in other ways. For instance, if using an insecure connection protocol during the information transfer of Juju relations. However, due to the key structure and protocol of WireGuard described in Section 3.1.6, the attacker does not have the necessary information from the exchange over the Juju relation to compromise the WireGuard tunnel.

WireGuard provides forward secrecy for data going in the tunnel, meaning that previous session data is secure even if an attacker gets the private key. Still, an attacker may find a peer's connection history by retrieving the private key. We only have one peer connecting on each interface for the lab used in the thesis making the connected peers obvious without knowing the keys. For more connected peers, finding the public key would be trivial for the attacker as handshakes occur with a few minutes interval based on the information in the paper [Don17]. Hiding the identity of the peer is a general limitation of WireGuard. Therefore, providing the public key over the Juju relation will not introduce a significantly higher risk than manual transfer. Other sensitive data is already shared in the peered relationship between the VCA and a VNF. Hence, we do not consider exchanging setup data for WireGuard over Juju to decrease security.

6.4 Performance

With OSM creating virtual networks on the VIM, multiple network slices can use the same internal IP address range without being able to communicate across slices. In our lab environment, we have had a lot of reused IP addresses that have not conflicted with each other. For example, is the IP addresses similar on the S6a, S1-C, S1-U, and SXab for all NSs and NSIs deployment with WireGuard we have deployed. For the management interface, the different VNFs can communicate across NSIs. Users with access to the VIM also can listen on all of the virtual networks.

With SFC on one or multiple VIMs, WireGuard can therefore be used to provide confidentiality of network traffic between VNFs. The hiding of information in the transition adding WireGuard from Figure 5.1 to 5.2 demonstrates this capability. We have shown that implementing WireGuard tunneling with OSM in a NS or NSI running other primary applications is possible, and it provides confidentiality for data running in the networks.

We have observed throughput and latency changing more when redeploying a NS than the differences measured between the methods/scenarios. The measurements done in Chapter 5 should therefore be seen as observations in a smaller number of scenarios and not definite results for introducing WireGuard with OSM. The total throughput in all measurements with WireGuard has been significantly lower than the internal NFVI bandwidth. We have therefore not seen the inconsistent differences we have measured for the raw NFVI throughput when doing our WireGuard tunnel measurements. However, the WireGuard result depends on the resources available for the VNFs. We have observed that changing resources impacts the WireGuard throughput from our measurements. A developer should therefore be aware of resources and the load of the VNFs before deploying a production-ready NS.

The KPIs for latency in 5G specify a control plane latency of 10-20 ms and E2E latency of lower than 1 ms for the data plane for URLLC use cases. The SRT includes both the data transportation over the network and the work done at the NF. In general, we have observed that we can use WireGuard within both URLLC and eMBB related KPIs. The diagram in Figure 6.2 compares the different latency measurements we have done to the 1 ms KPI and each other.

The control plane latency depends on the number of chained services and eventually communication between VIMs. With a fast internal network, most of the overhead from WireGuard comes from computing the encryption and packing the original packets into the WireGuard protocol. We had therefore expected the latency of the data over WireGuard to be lower with double vCPU resources. The performance, however, varies much in the NS with double resources, and we cannot conclude that the amount of resources does not affect latency. Hence, adding WireGuard will add a cost but is usable for multiple use cases. As we observe that the latency on the S1-C interface for the two NSIs to be lower than the pure NSs deployments, monitoring the workload in the VNFs and QoS indicators may be usable for tuning the latency.

For the KPIs regarding throughput, 100 Mbps data rate for the users is satisfying within the measured results. Usually, multiple users should be expected for a gNB, UPF or SPGW-U. With 2.2 Gbps throughput as our highest measure, the network can serve a total of $2200\,Mbps/100\,Mbps = 22$ devices with 100 Mbps per user. 5G network expects more connected devices but smaller cells for high bandwidth usage.

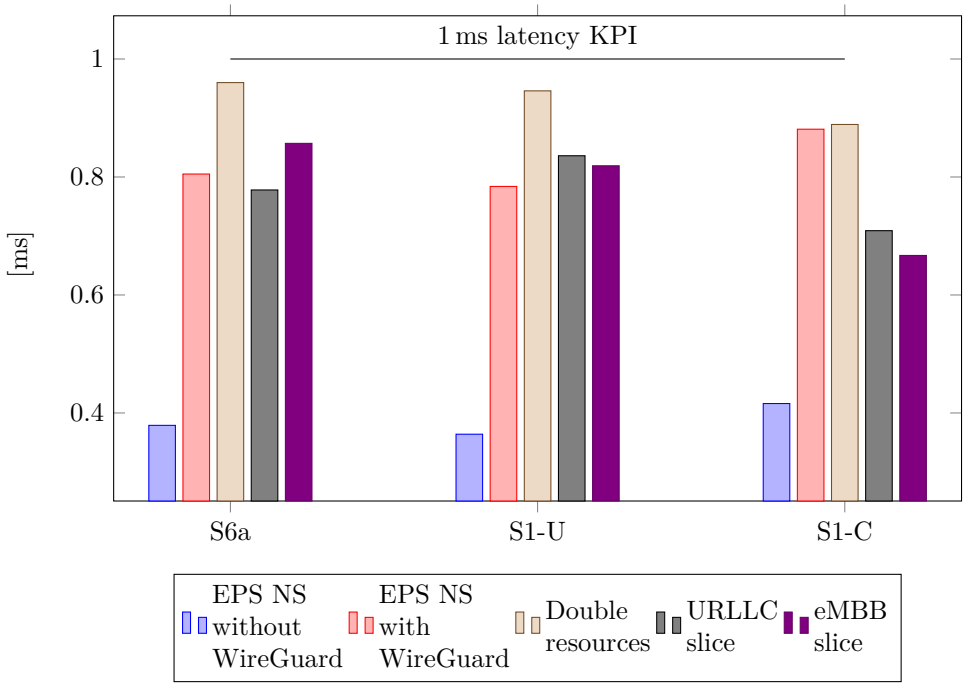

Figure 6.2: Latency comparison for different interfaces with WireGuard.

Also, not all devices will likely use the maximum data rate simultaneously. The UE implementation in our lab is, for instance, only able to use approximately 1.7 Mbps in the uplink. Therefore, using WireGuard connectivity can likely serve more than 22 devices of data plane traffic between NFs. The diagram in Figure 6.3 comparing the throughput for the different instantiations shows that the measurements are higher than the 100 Mbps Downlink user data rate KPI.

The performance roadmap for WireGuard lists several actions that could increase the performance, including bandwidth [Don17]. Based on our observation WireGuard still needs to make additional efforts to support performance close to the peak performance KPI. However, we have shown that WireGuard can support applications dependent on data rate in the Gbps scale. The comparison in Figure 6.3 also shows how the UE has been the bottleneck of our lab in regards to throughput.

We have seen some tendencies that introducing NSIs may affect latency and throughput of NSs. The differences between the two NSIs we have tested are, however, low and not consistent. The marginal differences make it challenging to conclude how much different NSIs are affected, if any.

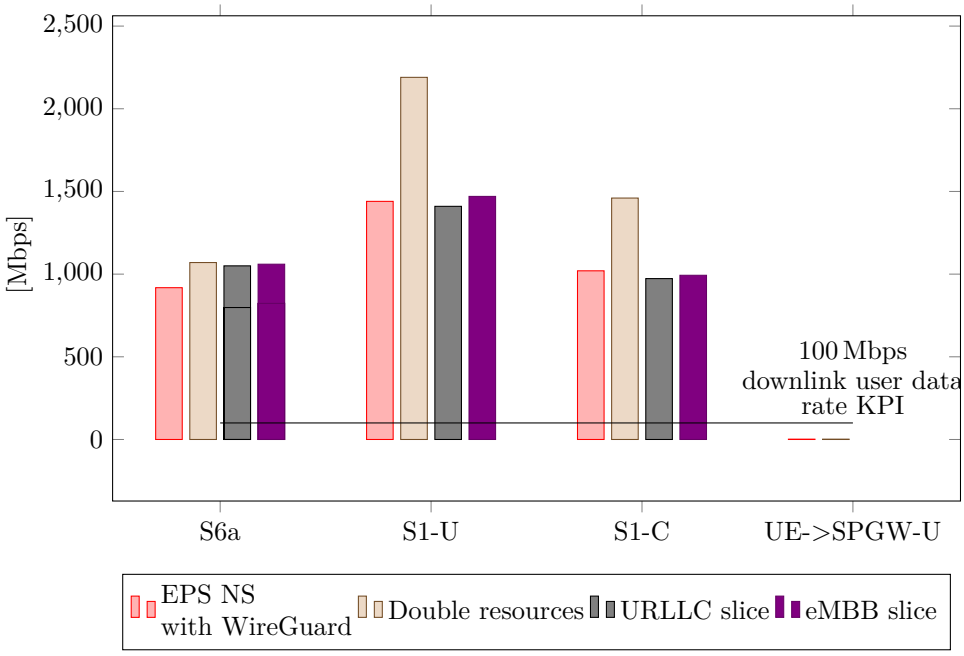

Figure 6.3: Throughput comparison for different interfaces with WireGuard.

Throughput measurements show some differences between the two NSIs. A comparison between the different throughput measurements is shown in the diagram of Figure 6.4 to illustrate the minor differences. When we ran the iPerf3 measurements over WireGuard, the throughput totaled around 3 Gbps. 3 Gbps is significantly lower than the internal networking, measured to be stable at around 20 Gbps. Therefore, the queue should not be overwhelming, even though there might be a need to prioritize at times. Furthermore, the average latencies are lower than the specified QoS parameters. Therefore, the observations for both the latency and the throughput measurements are as expected.

For future work, producing a higher total load and using other values of the values defining QoS and 5QI could help with clarifying how the NSIs get prioritized. As we have observed for the NS deployments, the performance varies between instantiations. For instance, for the S6a interface in Figure 6.4 the NSIs we have deployed and tested a second time went from a throughput of around 800 Mbps to 1.1 Gbps.

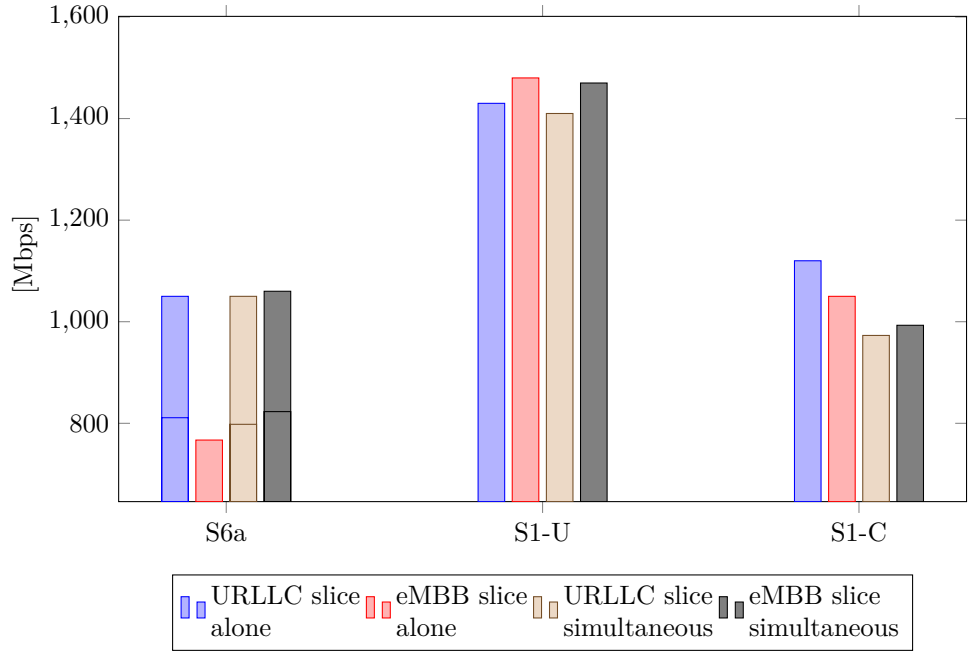

Figure 6.4: Throughput comparison with WireGuard for NSIs measured separately and simultaneously.

Chapter 7
Conclusion and Future Work

7.1 Conclusion

In this thesis, we have studied WireGuard as a VPNaaS deployed with OSM in beyond 5G context. Designing and effectuating several test cases have helped us answering our research questions. This section concludes our findings.

The use of WireGuard to guarantee slice isolation requires knowledge of the application in the NFs. The tenant manager or developer should design the outgoing data flow of NFs to route traffic over the WireGuard tunnel and not directly between peers. To achieve this, reconfiguration of the inter-working in a VNF or making architecture decisions for the data flows is necessary. The architecture should, in both cases, be able to chain services through the VPN tunnel and avoid application failure while waiting for establishing of the VPN tunnel. With proper configuration, we have observed that WireGuard can be used to guarantee slice isolation by tunneling and securing network connections.

In this thesis, we have measured throughput and latency in different scenarios. We have verified the applicability of WireGuard in a shared virtualized environment. The performance of WireGuard in our lab depended on resources and configuration of the NSs. The total SRT between NFs depends on the network latency between the NFs, the overhead added by WireGuard, and the SRT of the NFs applications. To support high-performance NSs, the developer should be aware of the effect of tuning resources and the additional overhead WireGuard introduces in the SFC. We have not measured data rates close to the peak data rate specified of the KPIs. For network slices where this is crucial, for instance, eMBB slices, the current version of WireGuard with our setup is not suitable. However, for other 5G KPIs for throughput and latency based on our operations, our observation has shown that WireGuard can be suitable for numerous applications. This includes both URLLC and eMBB slices.

The use of OSM to perform the deployment and orchestration of WireGuard in

network slices has been the significant construction task of this thesis. We have used the construction of descriptors and charms to answer the research question on how OSM supports WireGuard as a VPNaaS. Charms in OSM provide flexibility in running Day1-2 operations of the developers' choices. Implementing WireGuard in addition to other applications is, in our opinion, straightforward for developers with basic knowledge of proxy charms and experience with coding in Python. With relations and built-in functionality in OSM, it is possible to deploy VNFs with WireGuard rapidly in an automated way. From our observations and development process, we reckon OSM to be able to support WireGuard as a VPNaaS.

To summarize, we have found out that WireGuard is usable for verticals in a shared environment to secure information between NFs. OSM supports deploying WireGuard as a VPNaaS with various inbuilt methods. We have shown that Juju relations are feasible for automating a VPNaaS setup. To build network slices fulfilling verticals terms regarding throughput, latency, and security, the developer should design the WireGuard tunnel with appropriate resources and in a way that routes the outgoing data flow of NFs inside the VPN tunnel.

7.2 Future Work

In this thesis, we have provided a PoC of how WireGuard can be used as VPNaaS with OSM in 5G and beyond 5G networks. However, we could have followed several other paths to provide and expand the test environment even more realistically. Therefore, there are other scenarios and benchmarking tests that are possible to do for exploring the applicability of WireGuard in an NFV environment further. Below are several suggestions of topics for future work extending our study.

- Deploy WireGuard connectivity between the UE and multiple NSIs.
- Add or replace the core network part of this thesis with 5GS components.
- Appending a distributed KMS for WireGuard deployed by OSM using a third-party application.
- Use the scalability feature of VNFs in OSM to deploy more WireGuard peers and see how multiple connections affect performance.
- Study further differences of 5QI in slices deployed from OSM. For instance with other QoS parameters in OSM and with higher load.

References

[3GPa] 3GPP. Release 16. https://www.3gpp.org/release-16. Accessed: 22.12.2021.

[3GPb] 3GPP. Release 17. https://www.3gpp.org/release-17. Accessed: 22.12.2021.

[All] OpenAirInterface Software Alliance. Openairinterface. https://openairinterface.org/. Accessed: 04.01.2022.

[BAMH20] Alcardo Alex Barakabitze, Arslan Ahmad, Rashid Mijumbi, and Andrew Hines. 5g network slicing using sdn and nfv: A survey of taxonomy, architectures and future challenges. *Computer Networks*, 167:106984, 2020.

[BFG+17] Bego Blanco, Jose Oscar Fajardo, Ioannis Giannoulakis, Emmanouil Kafetzakis, Shuping Peng, Jordi Pérez-Romero, Irena Trajkovska, Pouria Sayyad Khodashenas, Leonardo Goratti, Michele Paolino, and Evangelos Sfakianakis. Technology pillars in the architecture of future 5g mobile networks: Nfv, mec and sdn. *Computer Standards & Interfaces*, 54, 01 2017.

[BV.] PowerDNS BV. Powerdns welcome! powerdns.com. Accessed: 28.12.2021.

[BVA+18] Annasamy Bagubali, Tanmay Verma, Anurag Anand, V Prithiviraj, and Partha Sharathi Mallick. Performance analysis of handover schemes in heterogeneous networks. *Journal of Circuits, Systems and Computers*, 27(11):1850177, 2018.

[DEM+] Jon Dugan, Seth Elliott, Bruce A. Mah, Jeff Poskanzer, and Kaustubh Prabhu. iperf - the ultimate speed test tool for tcp, udp and sctp. https://iperf.fr/. Accessed: 03.01.2022.

[Don] Jason A. Donenfeld. Wireguard. https://www.wireguard.com/. Accessed: 12.04.2021.

[Don17] Jason A Donenfeld. Wireguard: next generation kernel network tunnel. In *NDSS*, pages 1–12, 2017.

[Dre20] Thomas Dreibholz. Flexible 4g/5g testbed setup for mobile edge computing using openairinterface and open source mano. In *Workshops of the international conference on advanced information networking and applications*, pages 1143–1153. Springer, 2020.

[EK21] Ali Esmaeily and Katina Kralevska. Small-scale 5g testbeds for network slicing deployment: A systematic review. *Wireless Communications and Mobile Computing*, 2021, 2021.

[EKG20] Ali Esmaeily, Katina Kralevska, and Danilo Gligoroski. A cloud-based sdn/nfv testbed for end-to-end network slicing in 4g/5g. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 29–35. IEEE, 2020.

[EKM22] Ali Esmaeily, Katina Kralevska, and Toktam Mahmoodi. Slicing scheduling for supporting critical traffic in beyond 5g. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, pages 637–643, 2022.

[ETSa] ETSI. Etsi gs nfv-sol 006 v2.7.1. https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/006/02.07.01_60/gs_nfv-sol006v020701p.pdf. Accessed: 03.01.2022.

[ETSb] ETSI. Multi-access edge computing (mec). https://www.etsi.org/technologies/multi-access-edge-computing. Accessed: 22.12.2021.

[ETSc] ETSI. Network functions virtualisation (nfv). https://www.etsi.org/technologies/nfv/nfv. Accessed: 03.01.2022.

[ETSd] ETSI. Why do we need 5g? https://www.etsi.org/technologies/mobile/5g. Accessed: 22.12.2021.

[ETS20] ETSI. System architecture for the 5g system (5gs). Technical Report TS 123 501 V16.6.0, ETSI, October 2020.

[FKGG19] Mathias Kjolleberg Forland, Katina Kralevska, Michele Garau, and Danilo Gligoroski. Preventing ddos with sdn in 5g. In *2019 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7, 2019.

[Foua] Open Infrastructure Foundation. Gnocchi. https://wiki.openstack.org/wiki/Gnocchi. Accessed: 21.12.2021.

[Foub] The Apache Software Foundation. Downloading cassandra. https://cassandra.apache.org/_/download.html. Accessed: 20.11.2021.

[GK19] Danilo Gligoroski and Katina Kralevska. Expanded combinatorial designs as tool to model network slicing in 5g. *IEEE Access*, 7:54879–54887, 2019.

[GOLH+20] Andres J Gonzalez, Jose Ordonez-Lucena, Bjarne E Helvik, Gianfranco Nencioni, Min Xie, Diego R Lopez, and Pål Grønsund. The isolation concept in the 5g network slicing. In *2020 European Conference on Networks and Communications (EuCNC)*, pages 12–16. IEEE, 2020.

[Gon21] Iria Miguez Gonzalez. Virtualized cellular networks with native cloud functions. Master's thesis, Telecommunications Engineering School (Universida de Vigo), 2021.

[GP] 5G-PPP. Key performance indicators. https://5g-ppp.eu/kpis/. Accessed: 12.04.2021.

[Groa] 5G PPP Architecture Working Group. View on 5g architecture. https://5g-ppp.eu/wp-content/uploads/2020/02/5G-PPP-5G-Architecture-White-Paper_final.pdf. Accessed: 07.01.2022.

[Grob] OSM End User Advisory Group. Osm in action. https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_in_Action.pdf. Accessed: 12.01.2022.

[Hag20] Simen Haga. Virtualized cellular networks with native cloud functions. Master's thesis, Norwegian University of Science and Technology (NTNU), 2020.

[HBT20] Hajar Hantouti, Nabil Benamar, and Tarik Taleb. Service function chaining in 5g amp; beyond networks: Challenges and open research issues. *IEEE Network*, 34(4):320–327, 2020.

[HEKG20] Simen Haga, Ali Esmaeily, Katina Kralevska, and Danilo Gligoroski. 5g network slice isolation with wireguard and open source mano: a vpnaas proof-of-concept. In *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 181–187. IEEE, 2020.

[Inc] GitHub Inc. Ci/cd explained. https://resources.github.com/ci-cd/. Accessed: 12.01.2022.

[KGFG19] Katina Kralevska, Michele Garau, Mathias Førland, and Danilo Gligoroski. Towards 5g intrusion detection scenarios with omnet++. In Meyo Zongo, Antonio Virdis, Vladimir Vesely, Zeynep Vatandas, Asanga Udugama, Koojana Kuladinithi, Michael Kirsche, and Anna Förster, editors, *Proceedings of 6th International OMNeT++ Community Summit 2019*, volume 66 of *EPiC Series in Computing*, pages 44–51. EasyChair, 2019.

[Kim20] Hwankuk Kim. 5g core network security issues and attack classification from network protocol perspective. *J. Internet Serv. Inf. Secur.*, 10(2):1–15, 2020.

[Ltda] Canonical Ltd. cloud-init documentation. https://cloudinit.readthedocs.io/en/latest/index.html. Accessed: 03.11.2021.

[Ltdb] Canonical Ltd. Juju. juju.is. Accessed: 01.12.2021.

[Ltdc] Canonical Ltd. Relations. https://juju.is/docs/sdk/relations. Accessed: 28.01.2022.

[NGO21] Gianfranco Nencioni, Rosario Giuseppe Garroppo, and Ruxandra F. Olimid. 5g multi-access edge computing: Security, dependability, and performance. *CoRR*, abs/2107.13374, 2021.

[Ope] OpenAirInterface. 5g core network. https://openairinterface.org/oai-5g-core-network-project/. Accessed: 18.04.2021.

[OSMa] ETSI OSM. Annex 3: Osm information model. https://osm.etsi.org/docs/user-guide/11-osm-im.html. Accessed: 26.12.2021.

[OSMb] ETSI OSM. Day 1: Vnf services initialization. https://osm.etsi.org/docs/vnf-onboarding-guidelines/03-day1.html. Accessed: 03.01.2022.

[OSMc] ETSI OSM. etsi-nfv-nsd. http://osm-download.etsi.org/repository/osm/debian/ReleaseTEN/docs/osm-im/osm_im_trees/etsi-nfv-nsd.html. Accessed: 22.11.2021.

[OSMd] ETSI OSM. etsi-nfv-nst. http://osm-download.etsi.org/repository/osm/debian/ReleaseELEVEN/docs/osm-im/osm_im_trees/nst.html. Accessed: 22.11.2021.

[OSMe] ETSI OSM. etsi-nfv-vnfd. https://osm.etsi.org/docs/user-guide/05-osm-usage.html#advanced-instantiation-using-instantiation-parameters. Accessed: 22.11.2021.

[OSMf] ETSI OSM. etsi-nfv-vnfd. http://osm-download.etsi.org/repository/osm/debian/ReleaseTEN/docs/osm-im/osm_im_trees/etsi-nfv-vnfd.html. Accessed: 22.11.2021.

[OSMg] ETSI OSM. nst.yang. https://osm.etsi.org/gitweb/?p=osm/IM.git;a=blob;f=models/yang/nst.yang. Accessed: 10.02.2022.

[OSMh] ETSI OSM. Osm hackfests. osm.etsi.org/wikipub/index.html/OSM_Hackfests. Accessed: 28.12.2021.

[OSMi] ETSI OSM. Osm platform configuration. https://osm.etsi.org/docs/user-guide/06-osm-platform-configuration.html. Accessed: 09.01.2021.

[OSMj] ETSI OSM. Starting with juju bundles. https://osm.etsi.org/docs/vnf-onboarding-guidelines/05-quickstarts.html#starting-with-juju-bundles. Accessed: 30.12.2021.

[OSMk] ETSI OSM. Vnf onboarding guidelines. https://osm.etsi.org/docs/vnf-onboarding-guidelines/. Accessed: 12.04.2021.

[OSMl] ETSI OSM. What is osm? https://osm.etsi.org/. Accessed: 03.01.2022.

[PS] Frank Yong Yang Peter Schmitt, Bruno Landais. Control and user plane separation of epc nodes (cups). https://www.3gpp.org/cups. Accessed: 04.01.2022.

[Sau14] Martin Sauter. *From GSM to LTE-Advanced*. John Wiley & Sons Incorporated, 2 edition, 2014.

[Ser] Amazon Web Services. 5gvinni oai ns. https://github.com/simula/5gvinni-oai-ns. Accessed: 03.04.2021.

[Tor] Linus Torvalds. Linux kernel source tree. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=bd2463ac7d7ec51d432f23bf0e893fb371a908cd. Accessed: 16.01.2022.

[TT19] Dinesh Taneja and SS Tyagi. Factors impacting the performance of data transferred via vpn. *International Journal of Innovative Technology and Exploring Engineering*, 8:2962–2966, 2019.

[VNL+21] Ivan Vidal, Borja Nogales, Diego Lopez, Juan Rodríguez, Francisco Valera, and Arturo Azcorra. A secure link-layer connectivity platform for multi-site nfv services. *Electronics*, 10(15), 2021.

[vyo] vyos.io. Vyos.io. vyos.io. Accessed: 28.12.2021.

[Wie14] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[YYSCP20] Girma M. Yilma, Zarrar F. Yousaf, Vincenzo Sciancalepore, and Xavier Costa-Perez. Benchmarking open source nfv mano systems: Osm and onap. *Computer Communications*, 161:86–98, 2020.

Appendix A
Preparation of MicroStack

This appendix shows the command line history from our installation of MicroStack. After the initial installation, we add two OS images for Ubuntu 18.04 and Ubuntu 20.04. Lastly, we show the commands we have used to adjust the VIMs to cooperate and route VNF traffic to get Internet connectivity.

```
Common for both instances:
    7  sudo snap install microstack --devmode --edge
    8  sudo microstack init --auto --control
    9  sudo snap get microstack config.credentials.keystone-
       ↪ password
   10  microstack launch cirros -n test
   12  ssh -i /home/sondrki/snap/microstack/common/.ssh/
       ↪ id_microstack cirros@10.20.20.253
   13  sudo iptables -L
   14  wget https://cloud-images.ubuntu.com/bionic/current/bionic
       ↪ -server-cloudimg-amd64.img
   15  mkdir images
   16  mv bionic-server-cloudimg-amd64.img images/
   17  microstack.openstack image create --file images/bionic-
       ↪ server-cloudimg-amd64.img --public --container-format=
       ↪ bare --disk-format=qcow2 ubuntu18.04
   18  sudo sysctl net.ipv4.ip_forward=1
   19  wget https://cloud-images.ubuntu.com/focal/current/focal-
       ↪ server-cloudimg-amd64.img
   20  mv focal-server-cloudimg-amd64.img images/
   21  microstack.openstack image create --file images/focal-
       ↪ server-cloudimg-amd64.img --public --container-format=
       ↪ bare --disk-format=qcow2 ubuntu20.04
   22  sudo snap get microstack config.credentials.keystone-
       ↪ password
```

```
Extra command for VIM "microstack":
   sudo iptables -t nat -A POSTROUTING -s 10.20.20.1/24 ! -d
       ↪ 10.20.20.1/24 -j MASQUERADE # do not run when working
       ↪ with multi-site connections

Extra commands for VIM "a2ntnu_microstack":
   sudo ifconfig br-ex 10.21.21.2/24
   sudo nano /var/snap/microstack/common/etc/horizon/
       ↪ local_settings.d/_05_snap_tweaks.py # change from
       ↪ 10.20.20.1 to the new IP address 10.21.21.2
   sudo systemctl restart snap.microstack.*
   sudo iptables -t nat -A POSTROUTING -s 10.21.21.2/24 ! -d
       ↪ 10.21.21.2/24 -j MASQUERADE # do not run when working
       ↪ with multi-site connections
```

Listing A.1: Command line history from installation and setup of Microstack VIMs.

Appendix B
Preparation of OSM VM

Following are the commands we have used to install OSM and add VIMs.

```
wget https://osm-download.etsi.org/ftp/osm-10.0-ten/install_osm.
    ↪ sh
chmod +x install_osm.sh
./install_osm.sh -c k8s -t releaseten 2>&1 | tee osm_install_log.
    ↪ txt

osm vim-create --name microstack_a2 --user admin --password <
    ↪ password> --auth_url https://<ip address>:5000/v3/ --tenant
    ↪  admin --account_type openstack
osm vim-update a2ntnu_microstack --config '{use_floating_ip: True
    ↪ }'
osm vim-update a2ntnu_microstack --config '{insecure: True}'
```

Listing B.1: Command line history from OSM intallation.

Appendix C
Creation of UE VM

The appendix includes the command line history when installing the OAI UE and the configuration file to adjust the UE to our implementation.

C.1 Command Line History

```
sudo apt update && sudo apt upgrade && sudo apt-get install
    ↪ subversion git
git clone https://gitlab.eurecom.fr/oai/openairinterface5g.git
cd openairinterface5g/
source oaienv
cd cmake_targets/
./build_oai -I --phy_simulators
./build_oai -w SIMU --UE --eNB
cd lte_build_oai/build/
sudo nano ../../../openair3/NAS/TOOLS/ue_eurecom_test_sfr.conf
/home/ubuntu/openairinterface5g/targets/bin/conf2uedata -c
    ↪ ../../../openair3/NAS/TOOLS/ue_eurecom_test_sfr.conf -o .
sudo RFSIMULATOR=192.168.249.1 ./lte-uesoftmodem -C 2685000000 -r
    ↪  50 --rfsim
```

Listing C.1: Command line history for UE VM.

C.2 ue_eurecom_test_sfr.conf

```
# List of known PLMNS
PLMN: {
    PLMN0: {
            FULLNAME="Test network";
            SHORTNAME="OAI4G";
            MNC="01";
            MCC="001";
    PLMN1: {
```

```
            FULLNAME="NTNU testnet";
            SHORTNAME="testnet";
            MNC="95";
            MCC="208";
    };
};

UE0:
{
    USER: {
        IMEI="356113022094149";
        MANUFACTURER="EURECOM";
        MODEL="LTE Android PC";
        PIN="0000";
    };
    SIM: {
        MSIN="1234500808";
        USIM_API_K="449C4B91AEACD0ACE182CF3A5A72BFA1";
        OPC="9245cd6283cc53ce24ac1186a60dee6b";
        MSISDN="880000001";
    };

    # Home PLMN Selector with Access Technology
    HPLMN= "20895";

    # User controlled PLMN Selector with Access Technology
    UCPLMN_LIST = ();

    # Operator PLMN List
    OPLMN_LIST = ("00101", "20895");

    # Operator controlled PLMN Selector with Access Technology
    OCPLMN_LIST = ("22210", "21401", "21406", "26202", "26204");

    # Forbidden plmns
    FPLMN_LIST = ();

    # List of Equivalent HPLMNs
    #TODO: UE does not connect if set, to be fixed in the UE
    # EHPLMN_LIST= ("20811", "20813");
    EHPLMN_LIST= ();
};
```

Listing C.2: Content of modified ue_eurecom_test_sfr.conf.

Appendix D
WireGuard Charm

This appendix displays the addition needed for the Python code in Juju proxy charms to implement WireGuard as a VPNaaS. The code must be combined with the descriptor files of Appendix E to work.

```
import logging
from time import sleep
from random import randint

logger = logging.getLogger(__name__)


class HSSProxyCharm(SSHProxyCharm):

    def __init__(self, framework, key):
        # WireGuard application
        self.framework.observe(self.on.generatekeys_action, self.
            ↪ on_generatekeys_action)
        self.framework.observe(self.on.generatewgconfig_action,
            ↪ self.on_generateconfig_action)
        self.framework.observe(self.on.wgup_action, self.
            ↪ on_wireguardup_action)
        self.framework.observe(self.on.wgaddpeer_action, self.
            ↪ on_addpeer_action)
        self.framework.observe(self.on.wgdelpeer_action, self.
            ↪ on_delpeer_action)
        self.framework.observe(self.on.wgrestart_action, self.
            ↪ on_wgrestart_action)
        self.framework.observe(self.on.interface_relation_changed
            ↪ , self._on_interface_relation_changed)
```

```
    # WireGuard requirer config
    def on_generatekeys_action(self, event):
        err = ''
        result = ''
        wgifname = ''
        try:
            wgifname = event.params['wg-interface']
        except:
            wgifname = "wg0"
        try:
            proxy = self.get_ssh_proxy()
            cmd = ['sudo test -f /etc/wireguard/publickey{} && echo "$FILE exists." '.format(wgifname)]
            result, err = proxy.run(cmd)
        except:
            pass
        if len(result) > 5:
            event.set_results({'outout': result})
        else:
            try:
                proxy = self.get_ssh_proxy()
                cmd = ['wg genkey | sudo tee /etc/wireguard/privatekey{} | wg pubkey | sudo tee /etc/wireguard/publickey{}'.format(wgifname, wgifname)]
                result, err = proxy.run(cmd)
                event.set_results({'outout': result})
            except:
                event.fail('command failed:' + err)

    def on_generateconfig_action(self, event):
        err = ''
        try:
            proxy = self.get_ssh_proxy()
            gateway_ip = ''
            try:
                gateway_ip = event.params['gateway-ip']
                if not gateway_ip:
                    gateway_ip = self.model.config["ssh-hostname"]
            except:
                    gateway_ip = self.model.config["ssh-hostname"]
            wgifname = ''
            listenport = ''
```

```
            try:
                wgifname = event.params['wg-interface']
            except:
                wgifname = "wg0"
            try:
                listenport = event.params['listenport']
            except:
                listenport = "51820"
            subnet_of_tunnel = event.params['tunnel-subnet']
            endpoint = event.params['endpoint']
            cmd = ['echo -e "[Interface]\nAddress = {}\
                ↪ nListenPort = {}\nPrivatekey = $(sudo cat /etc/
                ↪ wireguard/privatekey{}) " | sudo tee -a /etc/
                ↪ wireguard/{}.conf'.format(endpoint, listenport,
                ↪  wgifname, wgifname)]
            result, err = proxy.run(cmd)
            cmd = ['echo -e "{}" | sudo tee /etc/wireguard/
                ↪ gateway_ip{}'.format(gateway_ip, wgifname)]
            result, err = proxy.run(cmd)
            cmd = ['echo -e "{}" | sudo tee /etc/wireguard/subnet
                ↪ {}'.format(subnet_of_tunnel, wgifname)]
            result, err = proxy.run(cmd)
            cmd = ['echo -e "{}" | sudo tee /etc/wireguard/
                ↪ listenport{}'.format(listenport, wgifname)]
            result, err = proxy.run(cmd)

            event.set_results({'outout': result})
        except:
            event.fail('command failed:' + err)

    def on_wireguardup_action(self, event):
        err = ''
        wgifname = ''
        try:
            wgifname = event.params['wg-interface']
        except:
            wgifname = "wg0"
        try:
            proxy = self.get_ssh_proxy()
            cmd = ['sudo wg-quick up {}'.format(wgifname)]
            result, err = proxy.run(cmd)
            event.set_results({'outout': result})
        except:
            event.fail('command failed:' + err)
```

```
    def on_addpeer_action(self, event):
        err = ''
        try:
            proxy = self.get_ssh_proxy()
            if self.model.unit.is_leader():
                peer_public_key = event.params['peer-publickey']
                gateway_ip = self.model.config["ssh-hostname"]
                subnet_behind_tunnel = event.params['subnet-
                    ↪ behind-tunnel']
                wgifname = ''
                listenport = ''
                try:
                    wgifname = event.params['wg-interface']
                except:
                    wgifname = "wg0"
                try:
                    listenport = event.params['listenport']
                except:
                    listenport = "51820"
                public_endpoint = event.params['public-endpoint']
                cmd = ['sudo wg set {} peer {} allowed-ips {},{}
                    ↪ endpoint {}:{} persistent-keepalive 25'.
                    ↪ format(wgifname, peer_public_key,
                    ↪ gateway_ip,subnet_behind_tunnel,
                    ↪ public_endpoint, listenport)]
                result, err = proxy.run(cmd)
                cmd = ['sudo ip -4 route add {} dev {}'.format(
                    ↪ gateway_ip, wgifname)]
                result, err = proxy.run(cmd)
                cmd = ['sudo ip -4 route add {} dev {}'.format(
                    ↪ subnet_behind_tunnel, wgifname)]
                result, err = proxy.run(cmd)
                cmd = ['sudo wg-quick save {}'.format(wgifname)]
                result, err = proxy.run(cmd)
                event.set_results({'outout': result})
        except:
            event.fail('command failed:' + err)
        else:
            event.fail("Unit is not leader")

    def on_delpeer_action(self, event):
        err = ''
        try:
            proxy = self.get_ssh_proxy()
            peer_public_key = event.params['peer-publickey']
```

```
            subnet_behind_tunnel = event.params['subnet-behind-
                ↪ tunnel']
            wgifname = ''
            try:
                wgifname = event.params['wg-interface']
            except:
                wgifname = "wg0"
            cmd = ['sudo wg set {} peer {} remove'.format(
                ↪ wgifname, peer_public_key)]
            result, err = proxy.run(cmd)
            cmd = ['sudo ip -4 route del {} dev {}'.format(
                ↪ subnet_behind_tunnel, wgifname)]
            result, err = proxy.run(cmd)
            cmd = ['sudo wg-quick save {}'.format(wgifname)]
            result, err = proxy.run(cmd)
            event.set_results({'outout': result})
        except:
            event.fail('command failed:' + err)

    def on_wgrestart_action(self, event):
        err = ''
        try:
            proxy = self.get_ssh_proxy()
            wgifname = ''
            try:
                wgifname = event.params['wg-interface']
            except:
                wgifname = "wg0"
            try:
                cmd = ['sudo systemctl restart wg-quick@{}.
                    ↪ service'.format(wgifname)]
                result, err = proxy.run(cmd)
            except:
                cmd = ['sudo wg-quick down {}'.format(wgifname)]
                result, err = proxy.run(cmd)
                cmd = ['sudo wg-quick start {}'.format(wgifname)]
                result, err = proxy.run(cmd)
                event.set_results({'restarted wg0': result})
        except:
            event.fail('command failed:' + err)


    def _on_interface_relation_changed(self, event): # change to
        ↪ correct juju interface name
        # INPUT correct wireguard interface name:
```

```
        wgifname = "wg0"
        self.wgrelation(event, wgifname)

    def wgrelation(self, event, wgifname):
        #logger.debug("RELATION DATA: {}".format(dict(event.
            ↪ relation.data[event.unit])))
        #parameter = event.relation.data[event.unit].get("
            ↪ parameter")
        #if parameter:
        #    self.model.unit.status = ActiveStatus("Parameter
            ↪ received: {}".format(parameter))
        proxy = False
        proxypass = False
        try:
            proxy = self.get_ssh_proxy()
            proxypass = True
        except:
            sleep(5)
            c = randint(1,10000)
            event.relation.data[self.model.unit]["counter"] = str
                ↪ (c)
            proxypass = False
        if proxypass:
            err = ''
            result = ''
            readyint = 0
            sshready = event.relation.data[event.unit].get("wg-
                ↪ ready")
            sshready_unit = event.relation.data[self.unit].get("
                ↪ wg-ready")
            if not sshready:
                sshready = "False"
            if not sshready_unit:
                sshready_unit = "False"
            peered = event.relation.data[self.model.unit].get("wg
                ↪ -peered")
            if not peered:
                peered = "False"
            if "True" not in sshready or "True" not in peered or
                ↪ "True" not in sshready_unit:
                c = randint(1,10000)
                event.relation.data[self.model.unit]["counter"] =
                    ↪  str(c)
                result = ''
```

```
leader_pubkey = event.relation.data[event.unit].
    ↪ get("wg-pubkey")
leader_listenport = event.relation.data[event.
    ↪ unit].get("wg-listenport")
leader_gwip = event.relation.data[event.unit].get
    ↪ ("wg-gwip")
leader_subnet = event.relation.data[event.unit].
    ↪ get("wg-subnet")
if not peered:
    peered = "False"
if leader_pubkey and leader_gwip and
    ↪ leader_subnet and leader_listenport:
    if "True" not in peered:
        proxy = self.get_ssh_proxy()
        cmd = ['echo -e "[Peer]\nPublicKey = {}\
            ↪ nAllowedIPs = {}\nEndpoint = {}:{}"
            ↪ | sudo tee -a /etc/wireguard/{}.
            ↪ conf'.format(leader_pubkey,
            ↪ leader_subnet, leader_gwip, str(
            ↪ leader_listenport), wgifname)]
        result, err = proxy.run(cmd)
        event.relation.data[self.model.unit]["wg-
            ↪ peered"] = "True"
        try:
            cmd = ['sudo wg-quick down {}'.format
                ↪ (wgifname)]
            result, err = proxy.run(cmd)
            cmd = ['sudo systemctl start wg-
                ↪ quick@{}.service'.format(
                ↪ wgifname)]
            result, err = proxy.run(cmd)
        except:
            pass
        #event.relation.data[self.unit].update({"
            ↪ wg-peered": "True"})

# send pubkey and other variables back to leader
result = ''
try:
    cmd = ['sudo test -f /etc/wireguard/publickey
        ↪ {} && echo "$FILE present"'.format(
        ↪ wgifname)]
    result, err = proxy.run(cmd)
except:
```

```
            event.relation.data[self.unit].update({"
                ↪ relation-joined": "failed1"})
        if len(result) > 5:
            #try:
            cmd = ['sudo cat /etc/wireguard/publickey{}'.
                ↪ format(wgifname)]
            result, err = proxy.run(cmd)
            readyint += 1
            event.relation.data[self.model.unit]["wg-
                ↪ pubkey"] = result
            #event.relation.data[self.unit].update({"
                ↪ wgpeer-pubkey": result})
            #except:
            #    event.relation.data[self.unit].update({"
                ↪ relation-joined": "failed2"})
        else:
            try:
                cmd = ['wg genkey | sudo tee /etc/
                    ↪ wireguard/privatekey{} | wg pubkey
                    ↪ | sudo tee /etc/wireguard/publickey
                    ↪ {}'.format(wgifname, wgifname)]
                result, err = proxy.run(cmd)
                cmd = ['sudo cat /etc/wireguard/publickey
                    ↪ {}'.format(wgifname)]
                result, err = proxy.run(cmd)
                readyint += 1
                event.relation.data[self.model.unit]["wg-
                    ↪ pubkey"] =      result
            except:
                pass
            #event.relation.data[self.unit].update({"
                ↪ wgpeer-pubkey": result})
            #event.relation.data[self.unit].update({"
                ↪ ready": "True"})
        result = ''
        try:
            cmd = ['sudo test -f /etc/wireguard/
                ↪ gateway_ip{} && echo "$FILE present"'.
                ↪ format(wgifname)]
            result, err = proxy.run(cmd)
        except:
            pass
        if len(result) > 5:
            cmd = ['sudo cat /etc/wireguard/gateway_ip{}'
                ↪ .format(wgifname)]
```

```
        result, err = proxy.run(cmd)
        event.relation.data[self.model.unit]["wg-gwip
            ↪ "] =      result
        #event.relation.data[self.unit].update({"
            ↪ wgpeer-gwip": result})
        readyint += 1
    result = ''
    try:
        cmd = ['sudo test -f /etc/wireguard/subnet{} 
            ↪ && echo "$FILE present"'.format(
            ↪ wgifname)]
        result, err = proxy.run(cmd)
    except:
        pass
    if len(result) > 5:
        cmd = ['sudo cat /etc/wireguard/subnet{}'.
            ↪ format(wgifname)]
        result, err = proxy.run(cmd)
        event.relation.data[self.model.unit]["wg-
            ↪ subnet"] =      result
        #event.relation.data[self.unit].update({"
            ↪ wgpeer-subnet": result})
        readyint += 1
        #if readyint >= 3:
        #    event.relation.data[self.unit].update({"
            ↪ ready": "True"})
    else:
        pass
    result = ''
    try:
        cmd = ['sudo test -f /etc/wireguard/
            ↪ listenport{} && echo "$FILE present"'.
            ↪ format(wgifname)]
        result, err = proxy.run(cmd)
    except:
        pass
    if len(result) > 5:
        cmd = ['sudo cat /etc/wireguard/listenport{}'
            ↪ .format(wgifname)]
        result, err = proxy.run(cmd)
        event.relation.data[self.model.unit]["wg-
            ↪ listenport"] =      result
        #event.relation.data[self.unit].update({"
            ↪ wgpeer-listenport": result})
        readyint += 1
```

```
            if readyint >= 4 and "True" in peered:
                event.relation.data[self.model.unit]["wg-
                    ↪ ready"] = "True"
                cmd = ['sudo wg-quick down {}'.format(
                    ↪ wgifname)]
                result, err = proxy.run(cmd)
                cmd = ['sudo wg-quick up {}'.format(wgifname)
                    ↪ ]
                result, err = proxy.run(cmd)
```

Listing D.1: Python code to share WireGuard data between peers.

Appendix E

VPNaaS Additions to Descriptor Files

This appendix lists the additions needed for different files to support WireGuard as a VPNaaS. The Python charm code to support the additions are given in Appendix D

```
# cloud-init.yaml
...
packages:
  - wireguard
```

Listing E.1: Additional configuration in cloud-init for implementing WireGuard as a VPN-as-a-Service.

```
# actions.yaml
...
generatekeys:
  description: "generates wireguard keys"
generatewgconfig:
  description: "generates wireguard config"
  params:
        tunnel-subnet:
            description: "subnets the other side
              ↪ should allow for in Allowed IPs"
            type: string
            default: ""
  required:
        - tunnel-subnet
wgup:
  description: "bring up wireguard"
wgaddpeer:
  description: "add peer to wireguard config"
  params:
        peer-publickey:
```

```
            description: "publickey of a peer"
            type: string
            default: ""
        subnet-behind-tunnel:
            description: "subnet to allow from other
              ↪ side of the wg tunnel"
            type: string
            default: ""
        wg-interface:
            description: "interface name"
            type: string
            default: "wg0"
        listenport:
            description: "port to listen on"
            type: string
            default: "51820"
  required:
        - peer-publickey
        - subnet-behind-tunnel
wgdelpeer:
  description: "delete peer from wireguard config"
wgrestart:
  description: "restarts the wireguard service"
  params:
        wg-interface:
            description: "interface to restart"
            type: string
            default: "wg0"
```

Listing E.2: Additional configuration in action.yaml charm file for implementing WireGuard as a VPN-as-a-Service.

```
# metadata.yaml - provider side
...
provides:
  relationprovider:
    interface: relation # must be equal on both
      ↪ require and provider side

# metadata.yaml - require side
...
requires:
```

```
  relationrequire:
    interface: relation # must be equal on both
       ↪ require and provider side
```

Listing E.3: Additional configuration in metadata.yaml charm file for implementing WireGuard as a VPN-as-a-Service.

```
# vnfd.yaml
...
    lcm-operations-configuration:
      operate-vnf-op-config:
        day1-2:
        - id: vnfd\_id
          juju:
            charm: name # name of charm given in
               ↪ metadata.yaml and normally also the
               ↪  folder name
          initial-config-primitive:
            ...
          - seq: '2'
            name: generatekeys
            execution-environment-ref: vnfd\_id
          - seq: '3'
            name: generatewgconfig
            execution-environment-ref: vnfd\_id
            parameter:
            - name: tunnel-subnet
              value: '192.168.248.0/24' # WireGuard
                 ↪ subnet the peer should allow data
                 ↪  from
            - name: gateway-ip
              value: '10.21.21.57' # external
                 ↪ connection point
            - name: endpoint
              value: '192.168.248.157/24' # internal
                 ↪ IP address to use for WireGuard
            - name: wg-interface
              value: 'wg0' # interface name. Not
                 ↪ mandatory. Used in case of
                 ↪ multiple WireGuard tunnels.
            - name: listenport
```

```
        value: '51820' # Port external
          ↪ connection point is listening on.
          ↪  Not mandatory. Used in case of
          ↪ multiple WireGuard tunnels.
    - seq: '4'
      name: wgup # Brings up the WireGuard
        ↪ interface
      parameter:
      - name: wg-interface
        value: 'wg0'
      execution-environment-ref: vnfd\_id
    config-primitive: # Manual steps. Day-2
      ↪ operations
    - name: wgaddpeer
      parameter:
      - name: peer-publickey
        data-type: STRING
        default-value: ''
      - name: subnet-behind-tunnel
        data-type: STRING
        default-value: '192.168.0.0/16'
      - name: public_endpoint
        data-type: STRING
        default-value: '192.168.0.1/24'
    - name: wgdelpeer
    - name: wgrestart
```

Listing E.4: Additional configuration in VNFd for implementing WireGuard as a VPN-as-a-Service.

Appendix F
Performance Measurements

The following appendix shows the full log output for the performance tests we have done. The distilled results are presented in Chapter 5. We have split the appendix into sections based on the different use cases.

F.1 Single Network Services with and Without WireGuard Connectivity

The following measurements are done using the NSs described in Section 4.7. A separate deployment is done for the OAI EPS NS with WireGuard implemented and the plain EPS NS without WireGuard tunneling.

```
eNB: iperf3 -B 192.168.248.157 -c 192.168.248.159 -M 1362 -t 600
    ↪ -i 60
SPGW-U: sudo iperf3 -s -B 192.168.248.159

ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -M
    ↪ 1362 -t 600 -i 60
Connecting to host 192.168.248.159, port 5201
local 192.168.248.157 port 49461 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  10.0 GBytes  1.44 Gbits/sec  305   1.51 MBytes
 60.00-120.00 sec  9.98 GBytes  1.43 Gbits/sec  436   1.54 MBytes
120.00-180.00 sec  9.98 GBytes  1.43 Gbits/sec  674   1.54 MBytes
180.00-240.00 sec  9.95 GBytes  1.42 Gbits/sec  731   1.52 MBytes
240.00-300.00 sec  10.1 GBytes  1.45 Gbits/sec  362   1.54 MBytes
300.00-360.00 sec  9.97 GBytes  1.43 Gbits/sec  600   1.10 MBytes
360.00-420.00 sec  9.98 GBytes  1.43 Gbits/sec  364   1.54 MBytes
420.00-480.00 sec  9.88 GBytes  1.42 Gbits/sec  117   1.54 MBytes
480.00-540.00 sec  10.0 GBytes  1.44 Gbits/sec  201   1.54 MBytes
540.00-600.00 sec  9.96 GBytes  1.43 Gbits/sec  576   1.54 MBytes
```

```
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  99.9 GBytes  1.43 Gbits/sec  4366     sender
  0.00-600.00 sec  99.9 GBytes  1.43 Gbits/sec          receiver
```

Listing F.1: Performance on S1-U over WireGuard tunnel for the EPS NS with WireGuard.

```
eNB: iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.248.159

ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600
    ↪ -i 60
Connecting to host 192.168.248.159, port 5201
local 192.168.248.157 port 53007 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  10.1 GBytes  1.45 Gbits/sec  207   1.51 MBytes
 60.00-120.00 sec  10.1 GBytes  1.45 Gbits/sec  539   1.51 MBytes
120.00-180.00 sec  10.0 GBytes  1.43 Gbits/sec   81   2.30 MBytes
180.00-240.00 sec  10.1 GBytes  1.45 Gbits/sec  614   1.51 MBytes
240.00-300.00 sec  9.83 GBytes  1.41 Gbits/sec  419   1.51 MBytes
300.00-360.00 sec  9.93 GBytes  1.42 Gbits/sec   88   1.51 MBytes
360.00-420.00 sec  10.0 GBytes  1.43 Gbits/sec  123   1.51 MBytes
420.00-480.00 sec  10.1 GBytes  1.45 Gbits/sec  793   1.56 MBytes
480.00-540.00 sec  10.1 GBytes  1.44 Gbits/sec  465   1.56 MBytes
540.00-600.00 sec  10.1 GBytes  1.44 Gbits/sec  472   1.44 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   100 GBytes  1.44 Gbits/sec  3801     sender
  0.00-600.00 sec   100 GBytes  1.44 Gbits/sec          receiver
```

Listing F.2: Performance on S1-U over WireGuard in the EPS NS with default MTU of iPerf3 (1500 bytes).

```
eNB: iperf3 -B 192.168.247.101 -c 192.168.247.12 -t 600 -i 60
MME: sudo iperf3 -s -B 192.168.247.12

ubuntu@enb:~$ iperf3 -B 192.168.247.101 -c 192.168.247.12 -t 600
    ↪ -i 60
Connecting to host 192.168.247.12, port 5201
local 192.168.247.101 port 60471 connected to 192.168.247.12 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  9.29 GBytes  1.33 Gbits/sec  390   1.04 MBytes
 60.00-120.00 sec  7.12 GBytes  1.02 Gbits/sec   84   1.11 MBytes
```

```
120.00-180.00  sec  8.04 GBytes  1.15 Gbits/sec   10   1.94 MBytes
180.00-240.00  sec  7.95 GBytes  1.14 Gbits/sec   57   1.33 MBytes
240.00-300.02  sec  7.94 GBytes  1.14 Gbits/sec   12   1.14 MBytes
300.02-360.00  sec  8.39 GBytes  1.20 Gbits/sec   11    893 KBytes
360.00-420.00  sec  7.93 GBytes  1.13 Gbits/sec  126   1.07 MBytes
420.00-480.00  sec  4.65 GBytes   665 Mbits/sec   12    952 KBytes
480.00-540.00  sec  4.29 GBytes   614 Mbits/sec   10   1.18 MBytes
540.00-600.00  sec  5.39 GBytes   772 Mbits/sec   12   1.13 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec  71.0 GBytes  1.02 Gbits/sec  724    sender
  0.00-600.00  sec  71.0 GBytes  1.02 Gbits/sec         receiver
```

Listing F.3: S1-C throughput with WireGuard in the OAI EPS NS.

```
UE: iperf3 -B 12.1.1.2 -c 192.168.222.15 -M 1362 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.222.15

ubuntu@ue:~$ iperf3 -B 12.1.1.2 -c 192.168.222.15 -M 1362 -t 600
    ↪ -i 60
Connecting to host 192.168.222.15, port 5201
local 12.1.1.2 port 52253 connected to 192.168.222.15 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00   sec  10.8 MBytes  1.52 Mbits/sec  168   17.1 KBytes
 60.00-120.00  sec  11.1 MBytes  1.56 Mbits/sec  165   15.8 KBytes
120.00-180.00  sec  11.1 MBytes  1.55 Mbits/sec  198   23.7 KBytes
180.00-240.00  sec  11.9 MBytes  1.67 Mbits/sec  223   11.9 KBytes
240.00-300.00  sec  12.2 MBytes  1.70 Mbits/sec  178   15.8 KBytes
300.00-360.00  sec  11.7 MBytes  1.64 Mbits/sec  183   34.3 KBytes
360.00-420.00  sec  12.5 MBytes  1.75 Mbits/sec  203   29.0 KBytes
420.00-480.00  sec  12.2 MBytes  1.70 Mbits/sec  180   19.8 KBytes
480.00-540.00  sec  12.0 MBytes  1.68 Mbits/sec  306   18.5 KBytes
540.00-600.00  sec  12.6 MBytes  1.76 Mbits/sec  144   47.5 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec   118 MBytes  1.65 Mbits/sec  1948    sender
  0.00-600.00  sec   118 MBytes  1.64 Mbits/sec         receiver
```

Listing F.4: Performance between UE and SPGW-U over Uu (10MHz) and WireGuard in the EPS NS with WireGuard.

```
MME: iperf3 -B 172.16.6.28 -c 172.16.6.128 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 172.16.6.128
ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
```

```
local 172.16.6.28 port 46581 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  5.92 GBytes   848 Mbits/sec   14    934 KBytes
 60.00-120.00 sec  6.34 GBytes   908 Mbits/sec   64   1.03 MBytes
120.00-180.00 sec  6.32 GBytes   904 Mbits/sec   21   1.28 MBytes
180.00-240.00 sec  6.59 GBytes   943 Mbits/sec   45    998 KBytes
240.00-300.00 sec  6.62 GBytes   947 Mbits/sec  148   1.24 MBytes
300.00-360.00 sec  6.87 GBytes   983 Mbits/sec   11   1.35 MBytes
360.00-420.00 sec  6.60 GBytes   944 Mbits/sec   12    889 KBytes
420.00-480.00 sec  5.94 GBytes   850 Mbits/sec   11   1.02 MBytes
480.00-540.00 sec  6.55 GBytes   938 Mbits/sec   11    957 KBytes
540.00-600.00 sec  6.41 GBytes   918 Mbits/sec   10   1.10 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  64.1 GBytes   918 Mbits/sec  347    sender
  0.00-600.00 sec  64.1 GBytes   918 Mbits/sec         receiver
```

Listing F.5: Throughput on S6a with WireGuard in the OAI EPS NS.

```
MME: iperf3 -B 192.168.8.2 -c 192.168.8.129 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 192.168.8.129

ubuntu@mme:~$ iperf3 -B 192.168.8.2 -c 192.168.8.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 192.168.8.129, port 5201
local 192.168.8.2 port 46009 connected to 192.168.8.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   157 GBytes  22.5 Gbits/sec    0   3.15 MBytes
 60.00-120.00 sec   155 GBytes  22.3 Gbits/sec    0   3.15 MBytes
120.00-180.00 sec   148 GBytes  21.1 Gbits/sec    0   3.15 MBytes
180.00-240.00 sec   162 GBytes  23.2 Gbits/sec    1   3.15 MBytes
240.00-300.00 sec   145 GBytes  20.8 Gbits/sec    0   3.15 MBytes
300.00-360.00 sec   150 GBytes  21.5 Gbits/sec    1   3.15 MBytes
360.00-420.00 sec   147 GBytes  21.1 Gbits/sec    1   3.15 MBytes
420.00-480.00 sec   151 GBytes  21.6 Gbits/sec    0   3.15 MBytes
480.00-540.00 sec   143 GBytes  20.5 Gbits/sec    1   3.15 MBytes
540.00-600.00 sec   156 GBytes  22.3 Gbits/sec    0   3.15 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  21.7 Gbits/sec    4    sender
  0.00-600.00 sec  0.00 (null)s  21.7 Gbits/sec         receiver
```

Listing F.6: Throughput on S6a in the OAI EPS NS without WireGuard.

```
OAI EPS NS with WireGuard:
eNB: ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
```

```
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 202341ms
rtt min/avg/max/mdev = 0.517/0.881/2.111/0.161 ms

OAI EPS NS without WireGuard:
eNB: ping -q -i 0.2 -c 1000 -I ens4 192.168.7.102
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.7.102
PING 192.168.7.102 (192.168.7.102) from 192.168.7.101 ens4:
    ↪ 56(84) bytes of data.

--- 192.168.7.102 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203770ms
rtt min/avg/max/mdev = 0.243/0.416/1.991/0.085 ms
```

Listing F.7: Latency on S1-C run on two EPS NS with and without WireGuard.

```
OAI EPS NS with WireGuard:
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203387ms
rtt min/avg/max/mdev = 0.440/0.784/1.873/0.156 ms

OAI EPS NS without WireGuard:
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.7.102
PING 192.168.7.102 (192.168.7.102) from 192.168.7.101 ens4:
    ↪ 56(84) bytes of data.

--- 192.168.7.102 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203796ms
rtt min/avg/max/mdev = 0.209/0.364/0.631/0.064 ms
```

Listing F.8: Latency on S1-U with and without WireGuard in the EPS NS.

```
OAI EPS NS with WireGuard:
MME: ping -q -i 0.2 -c 1000 -I wg0 172.16.6.128

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.128
PING 172.16.6.128 (172.16.6.128) from 172.16.6.28 wg0: 56(84)
    ↪ bytes of data.

--- 172.16.6.128 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203508ms
rtt min/avg/max/mdev = 0.523/0.805/2.748/0.127 ms

OAI EPS NS without WireGuard:
MME: ping -q -i 0.2 -c 1000 -I ens4 192.168.8.129

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.8.129
PING 192.168.8.129 (192.168.8.129) from 192.168.8.2 ens4: 56(84)
    ↪ bytes of data.

--- 192.168.8.129 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203833ms
rtt min/avg/max/mdev = 0.221/0.379/2.109/0.090 ms
```

Listing F.9: Latency on S6a with and without WireGuard in the EPS NS.

```
UE: iperf3 -B 12.1.1.2 -c 192.168.222.88 -M 1362 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.222.88

ubuntu@ue:~$ iperf3 -B 12.1.1.2 -c 192.168.222.88 -M 1362 -t 600
    ↪ -i 60
Connecting to host 192.168.222.88, port 5201
local 12.1.1.2 port 57223 connected to 192.168.222.88 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  11.7 MBytes  1.64 Mbits/sec  199   18.5 KBytes
 60.00-120.00 sec  12.1 MBytes  1.68 Mbits/sec  219   19.8 KBytes
120.00-180.00 sec  12.1 MBytes  1.68 Mbits/sec  200   19.8 KBytes
180.00-240.00 sec  11.4 MBytes  1.60 Mbits/sec  211   11.9 KBytes
240.00-300.00 sec  11.9 MBytes  1.66 Mbits/sec  188   15.8 KBytes
300.00-360.00 sec  11.8 MBytes  1.65 Mbits/sec  183   21.1 KBytes
360.00-420.00 sec  12.0 MBytes  1.68 Mbits/sec  198   21.1 KBytes
420.00-480.00 sec  11.4 MBytes  1.60 Mbits/sec  231   18.5 KBytes
480.00-540.00 sec  9.27 MBytes  1.30 Mbits/sec  216   13.2 KBytes
540.00-600.00 sec  11.9 MBytes  1.67 Mbits/sec  174   33.0 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
```

```
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   116 MBytes  1.61 Mbits/sec  2019     sender
  0.00-600.00 sec   115 MBytes  1.61 Mbits/sec           receiver
```

Listing F.10: UE throughput to SPGW-U in the EPS NS without WireGuard.

F.2 Two Different Network Slice Instances

The following section shows the results for the measurements done using the NSTs described in Section 4.9.

F.2.1 Running Tests on Only One NSI at a Time

eMBB descripted slice:

```
MME: iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 172.16.6.129

ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 44409 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  5.20 GBytes   745 Mbits/sec   13   1.08 MBytes
 60.00-120.00 sec  5.46 GBytes   782 Mbits/sec   14   1.19 MBytes
120.00-180.00 sec  5.33 GBytes   762 Mbits/sec   12    834 KBytes
180.00-240.00 sec  5.42 GBytes   777 Mbits/sec   10   1.19 MBytes
240.00-300.00 sec  5.44 GBytes   779 Mbits/sec   14   1.18 MBytes
300.00-360.00 sec  5.40 GBytes   773 Mbits/sec   12    832 KBytes
360.00-420.00 sec  5.35 GBytes   766 Mbits/sec   13    974 KBytes
420.00-480.00 sec  5.30 GBytes   758 Mbits/sec   10   1.12 MBytes
480.00-540.00 sec  5.25 GBytes   752 Mbits/sec   11   1.06 MBytes
540.00-600.00 sec  5.45 GBytes   780 Mbits/sec   12    858 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  53.6 GBytes   767 Mbits/sec  121     sender
  0.00-600.00 sec  53.6 GBytes   767 Mbits/sec           receiver
```

Listing F.11: S6a throughput with WireGuard in the eMBB descripted slice.

```
ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 34199 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
```

```
  0.00-60.00   sec  7.37 GBytes  1.06 Gbits/sec   12   1.02 MBytes
 60.00-120.00  sec  7.50 GBytes  1.07 Gbits/sec   10   1.12 MBytes
120.00-180.00  sec  7.58 GBytes  1.08 Gbits/sec   11   1.03 MBytes
180.00-240.00  sec  7.64 GBytes  1.09 Gbits/sec   10   1.20 MBytes
240.00-300.00  sec  7.20 GBytes  1.03 Gbits/sec   11   1.23 MBytes
300.00-360.00  sec  7.41 GBytes  1.06 Gbits/sec   11   1.33 MBytes
360.00-420.00  sec  7.40 GBytes  1.06 Gbits/sec   11   1.01 MBytes
420.00-480.00  sec  7.28 GBytes  1.04 Gbits/sec   10   1.13 MBytes
480.00-540.00  sec  6.74 GBytes   964 Mbits/sec   11   1.14 MBytes
540.00-600.00  sec  6.98 GBytes   999 Mbits/sec   11   1.04 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec  73.1 GBytes  1.05 Gbits/sec  108     sender
  0.00-600.00  sec  73.1 GBytes  1.05 Gbits/sec          receiver
```

Listing F.12: S6a throughput with WireGuard in the eMBB slice - run 2.

```
ubuntu@enb:~$ iperf3 -B 192.168.247.111 -c 192.168.247.12 -t 600
    ↪ -i 60
Connecting to host 192.168.247.12, port 5201
local 192.168.247.111 port 46667 connected to 192.168.247.12 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00   sec  7.57 GBytes  1.08 Gbits/sec  869   1.09 MBytes
 60.00-120.00  sec  7.73 GBytes  1.11 Gbits/sec   10   1.05 MBytes
120.00-180.00  sec  7.50 GBytes  1.07 Gbits/sec   10   1.29 MBytes
180.00-240.00  sec  7.25 GBytes  1.04 Gbits/sec   17   1.10 MBytes
240.00-300.00  sec  6.96 GBytes   997 Mbits/sec   10   1.25 MBytes
300.00-360.00  sec  6.90 GBytes   987 Mbits/sec   11   1.11 MBytes
360.00-420.00  sec  7.56 GBytes  1.08 Gbits/sec   10   1.12 MBytes
420.00-480.00  sec  7.04 GBytes  1.01 Gbits/sec  116   1.27 MBytes
480.00-540.00  sec  7.29 GBytes  1.04 Gbits/sec   11   1.16 MBytes
540.00-600.00  sec  7.36 GBytes  1.05 Gbits/sec   10   1.21 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec  73.1 GBytes  1.05 Gbits/sec  1074    sender
  0.00-600.00  sec  73.1 GBytes  1.05 Gbits/sec          receiver
```

Listing F.13: S1-C throughput with WireGuard in the eMBB slice.

```
Unencrypted interface:
MME: ping -q -i 0.2 -c 1000 -I ens4 192.168.8.129

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.8.129
PING 192.168.8.129 (192.168.8.129) from 192.168.8.2 ens4: 56(84)
    ↪ bytes of data.
```

```
--- 192.168.8.129 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203767ms
rtt min/avg/max/mdev = 0.218/0.411/1.559/0.111 ms

Inside WireGuard tunnel:
MME: ping -q -i 0.2 -c 1000 -I wg0 172.16.6.129

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.129
PING 172.16.6.129 (172.16.6.129) from 172.16.6.28 wg0: 56(84)
    ↪ bytes of data.

--- 172.16.6.129 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 202988ms
rtt min/avg/max/mdev = 0.503/0.874/1.498/0.123 ms
```

Listing F.14: S6a latency in the eMBB slice.

```
eNB: ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203594ms
rtt min/avg/max/mdev = 0.402/0.746/1.932/0.153 ms
```

Listing F.15: Latency on S1-C in the eMBB slice.

```
eNB: ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 202846ms
rtt min/avg/max/mdev = 0.517/0.866/4.935/0.192 ms
```

Listing F.16: S1-C latency in WireGuard tunnel in the eMBB slice - run 2.

```
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159
PING 192.168.248.159 (192.168.248.159) from 192.168.248.157 wg0:
    ↪ 56(84) bytes of data.

--- 192.168.248.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203664ms
rtt min/avg/max/mdev = 0.413/0.737/1.930/0.142 ms
```

Listing F.17: S1-U latency in WireGuard tunnel in the eMBB slice.

```
eNB: iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.248.159

ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600
    ↪ -i 60
Connecting to host 192.168.248.159, port 5201
local 192.168.248.157 port 34533 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  10.7 GBytes  1.53 Gbits/sec  203   1.58 MBytes
 60.00-120.00 sec  10.3 GBytes  1.47 Gbits/sec  208   1.55 MBytes
120.00-180.00 sec  10.3 GBytes  1.48 Gbits/sec  396   1.53 MBytes
180.00-240.00 sec  10.2 GBytes  1.46 Gbits/sec  459   1.40 MBytes
240.00-300.00 sec  10.6 GBytes  1.51 Gbits/sec  227   1.51 MBytes
300.00-360.00 sec  10.5 GBytes  1.50 Gbits/sec  433   1.56 MBytes
360.00-420.00 sec  10.3 GBytes  1.48 Gbits/sec  221   1.42 MBytes
420.00-480.00 sec  10.3 GBytes  1.48 Gbits/sec  555   1.42 MBytes
480.00-540.00 sec  10.2 GBytes  1.47 Gbits/sec  266   1.38 MBytes
540.00-600.00 sec  10.2 GBytes  1.46 Gbits/sec  217   1.56 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   104 GBytes  1.48 Gbits/sec  3185    sender
  0.00-600.00 sec   104 GBytes  1.48 Gbits/sec          receiver
```

Listing F.18: S1-U throughput with WireGuard tunnel in the eMBB slice.

```
Measured between management interfaces of SPGW-U and eNB in eMBB
    ↪ descripted slice:
SPGW-U: iperf3 -B 192.168.9.159 -c 192.168.9.57  -t 600 -i 60
eNB: sudo iperf3 -s -B 192.168.9.57

ubuntu@spgw-u:~$ iperf3 -B 192.168.9.159 -c 192.168.9.57  -t 600
    ↪ -i 60
Connecting to host 192.168.9.57, port 5201
```

```
local 192.168.9.159 port 43735 connected to 192.168.9.57 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   116 GBytes  16.6 Gbits/sec    0   3.01 MBytes
 60.00-120.00 sec   113 GBytes  16.1 Gbits/sec    0   3.01 MBytes
120.00-180.00 sec   115 GBytes  16.4 Gbits/sec    0   3.01 MBytes
180.00-240.00 sec   115 GBytes  16.4 Gbits/sec    0   3.01 MBytes
240.00-300.00 sec   117 GBytes  16.8 Gbits/sec    0   3.01 MBytes
300.00-360.00 sec   115 GBytes  16.5 Gbits/sec    0   3.01 MBytes
360.00-420.00 sec   115 GBytes  16.4 Gbits/sec    0   3.01 MBytes
420.00-480.00 sec   116 GBytes  16.6 Gbits/sec    0   3.01 MBytes
480.00-540.00 sec   115 GBytes  16.5 Gbits/sec    0   3.01 MBytes
540.00-600.00 sec   117 GBytes  16.7 Gbits/sec    0   3.01 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  16.5 Gbits/sec    0    sender
  0.00-600.00 sec  0.00 (null)s  16.5 Gbits/sec         receiver


eNB workload when running throughput measurement:
Tasks: 109 total, 1 running, 57 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.1 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 8167500 total, 1200084 free, 133388 used, 6834028 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 7728040 avail Mem

 PID USER PR NI VIRT  RES  SHR  S %CPU %MEM TIME+   COMMAND
 1   root 20 0  77976 9160 6684 S 0.0  0.1  0:11.38 systemd

SPGW-U workload when running throughput measurement:
top - 08:00:15 up 5 days, 1:23, 1 user, load average: 0.14, 0.36,
    ↪  0.24
Tasks: 86 total, 1 running, 45 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.3 sy, 0.0 ni, 99.0 id, 0.7 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 3073036 total, 191844 free, 276592 used, 2604600 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 2603872 avail Mem

 PID  USER PR NI VIRT    RES    SHR   S %CPU %MEM TIME+   COMMAND
 1838 root 20 0  20.000t 186264 11028 S 0.3  6.1  1:02.89 spgwu

Reverse:
eNB: iperf3 -B 192.168.222.59 -c 192.168.222.37 -t 600 -i 60
```

```
SPGW-U: sudo iperf3 -s -B 192.168.222.37

ubuntu@enb:~$ iperf3 -B 192.168.222.59 -c 192.168.222.37 -t 600 -
    ↪ i 60
Connecting to host 192.168.222.37, port 5201
local 192.168.222.59 port 53303 connected to 192.168.222.37 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec    120 GBytes  17.2 Gbits/sec    0   3.02 MBytes
 60.00-120.00 sec    114 GBytes  16.3 Gbits/sec    0   3.02 MBytes
120.00-180.00 sec    131 GBytes  18.8 Gbits/sec    0   3.02 MBytes
180.00-240.00 sec    135 GBytes  19.3 Gbits/sec    0   3.02 MBytes
240.00-300.00 sec    132 GBytes  18.8 Gbits/sec    0   3.02 MBytes
300.00-360.00 sec    128 GBytes  18.4 Gbits/sec    0   3.02 MBytes
360.00-420.00 sec    132 GBytes  18.9 Gbits/sec    0   3.02 MBytes
420.00-480.00 sec    119 GBytes  17.0 Gbits/sec    0   3.02 MBytes
480.00-540.00 sec    121 GBytes  17.3 Gbits/sec    0   3.02 MBytes
540.00-600.00 sec    117 GBytes  16.8 Gbits/sec    0   3.02 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  17.9 Gbits/sec    0    sender
  0.00-600.00 sec  0.00 (null)s  17.9 Gbits/sec         receiver
```

Listing F.19: S1-U throughput outside tunnel in the eMBB slice.

```
eNB: sudo iperf3 -s -B 192.168.222.59
SPGW-U: iperf3 -B 192.168.222.37 -c 192.168.222.59 -M 1362 -t 600
    ↪  -i 60

ubuntu@spgw-u:~$ iperf3 -B 192.168.222.37 -c 192.168.222.59 -M
    ↪ 1362 -t 600 -i 60
Connecting to host 192.168.222.59, port 5201
local 192.168.222.37 port 52241 connected to 192.168.222.59 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec    101 GBytes  14.5 Gbits/sec    0   3.02 MBytes
 60.00-120.00 sec   98.8 GBytes  14.1 Gbits/sec    0   3.02 MBytes
120.00-180.00 sec   98.8 GBytes  14.1 Gbits/sec    0   3.02 MBytes
180.00-240.00 sec    100 GBytes  14.4 Gbits/sec    0   3.02 MBytes
240.00-300.00 sec    101 GBytes  14.4 Gbits/sec    0   3.02 MBytes
300.00-360.00 sec    101 GBytes  14.5 Gbits/sec    0   3.02 MBytes
360.00-420.00 sec    101 GBytes  14.4 Gbits/sec    0   3.02 MBytes
420.00-480.00 sec   98.8 GBytes  14.2 Gbits/sec    0   3.02 MBytes
480.00-540.00 sec   98.4 GBytes  14.1 Gbits/sec    0   3.02 MBytes
540.00-600.00 sec    101 GBytes  14.4 Gbits/sec    0   3.02 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
```

```
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  1000 GBytes  14.3 Gbits/sec    0     sender
  0.00-600.00 sec  1000 GBytes  14.3 Gbits/sec          receiver


Sample of resource usage in VNFs:
SPGW-U:
top - 08:12:05 up 5 days, 1:35, 1 user, load average: 0.21, 0.36,
    ↪  0.30
Tasks: 85 total, 1 running, 45 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.3 sy, 0.0 ni, 98.3 id, 1.3 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 3073036 total, 190020 free, 278152 used, 2604864 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 2602312 avail Mem

 PID  USER PR NI VIRT    RES    SHR   S %CPU %MEM TIME+   COMMAND
 1838 root 20 0  20.000t 187720 11028 S 0.3  6.1  1:03.34 spgwu
eNB:
top - 08:13:06 up 5 days, 1:34, 1 user, load average: 0.08, 0.36,
    ↪  0.38
Tasks: 109 total, 1 running, 57 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.1 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 8167500 total, 1199608 free, 133848 used, 6834044 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 7727580 avail Mem

 PID  USER   PR NI VIRT  RES  SHR  S %CPU %MEM TIME+   COMMAND
 8688 ubuntu 20 0  44552 4016 3348 R 0.3  0.0  0:00.08 top
```

Listing F.20: S1-U throughput between management interfaces in the eMBB slice.

```
HSS: 08:37:29 up 5 days,  2:01,  1 user,  load average: 0.32,
    ↪ 0.07, 0.02
MME: 08:36:53 up 5 days,  2:00,  1 user,  load average: 0.08,
    ↪ 0.03, 0.01
SPGW-U:  08:30:46 up 5 days,  1:53,  1 user,  load average: 0.01,
    ↪  0.29, 0.39
SPGW-C: 08:37:54 up 5 days,  2:01,  1 user,  load average: 0.00,
    ↪ 0.00, 0.00
eNB: 08:30:18 up 5 days,  1:51,  1 user,  load average: 0.01,
    ↪ 0.31, 0.43
```

Listing F.21: eMBB slice load average sample.

URLLC descripted slice:

```
MME: iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 172.16.6.129

ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 38855 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  5.44 GBytes   779 Mbits/sec   12   1.13 MBytes
 60.00-120.00 sec  5.47 GBytes   783 Mbits/sec   12   1.04 MBytes
120.00-180.00 sec  5.78 GBytes   828 Mbits/sec   11    970 KBytes
180.00-240.00 sec  5.53 GBytes   792 Mbits/sec   12    922 KBytes
240.00-300.00 sec  5.68 GBytes   813 Mbits/sec   11   1.03 MBytes
300.00-360.00 sec  5.96 GBytes   853 Mbits/sec   10   1.07 MBytes
360.00-420.00 sec  5.87 GBytes   841 Mbits/sec   12    844 KBytes
420.00-480.00 sec  5.52 GBytes   790 Mbits/sec   11   1.07 MBytes
480.00-540.00 sec  5.93 GBytes   849 Mbits/sec   11   1.15 MBytes
540.00-600.00 sec  5.47 GBytes   783 Mbits/sec   12    920 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  56.7 GBytes   811 Mbits/sec  114     sender
  0.00-600.00 sec  56.7 GBytes   811 Mbits/sec          receiver
```

Listing F.22: S6a throughput with WireGuard in the URLLC slice.

```
Command on MME: iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
HSS: sudo iperf3 -s -B 172.16.6.129

ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 53531 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  7.69 GBytes  1.10 Gbits/sec   12   1.25 MBytes
 60.00-120.00 sec  7.21 GBytes  1.03 Gbits/sec   11   1.25 MBytes
120.00-180.00 sec  7.39 GBytes  1.06 Gbits/sec   11   1.23 MBytes
180.00-240.00 sec  6.98 GBytes   999 Mbits/sec   12   1.09 MBytes
240.00-300.00 sec  7.25 GBytes  1.04 Gbits/sec   10   1.37 MBytes
300.00-360.00 sec  7.24 GBytes  1.04 Gbits/sec   21   1.01 MBytes
360.00-420.00 sec  7.52 GBytes  1.08 Gbits/sec   11   1.02 MBytes
420.00-480.00 sec  7.53 GBytes  1.08 Gbits/sec   11   1.09 MBytes
480.00-540.00 sec  7.43 GBytes  1.06 Gbits/sec   10   1.40 MBytes
540.00-600.00 sec  7.32 GBytes  1.05 Gbits/sec   12   1.00 MBytes
```

```
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  73.6 GBytes  1.05 Gbits/sec  121     sender
  0.00-600.00 sec  73.6 GBytes  1.05 Gbits/sec          receiver
```

Listing F.23: S6a throughput with WireGuard in the URLLC slice.

```
MME:
ping -q -i 0.2 -c 1000 -I wg0 172.16.6.129

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.129
PING 172.16.6.129 (172.16.6.129) from 172.16.6.28 wg0: 56(84)
    ↪ bytes of data.

--- 172.16.6.129 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203268ms
rtt min/avg/max/mdev = 0.499/0.843/2.930/0.137 ms
```

Listing F.24: S6a latency in the URLLC slice with WireGuard.

```
eNB: ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159
PING 192.168.248.159 (192.168.248.159) from 192.168.248.157 wg0:
    ↪ 56(84) bytes of data.

--- 192.168.248.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203644ms
rtt min/avg/max/mdev = 0.394/0.733/1.938/0.121 ms
```

Listing F.25: S1-U latency in the URLLC slice with WireGuard.

```
eNB: ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203001ms
rtt min/avg/max/mdev = 0.492/0.837/5.323/0.193 ms
```

Listing F.26: S1-C latency in WireGuard tunnel in the URLLC slice.

```
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.7.102
PING 192.168.7.102 (192.168.7.102) from 192.168.7.101 ens4:
    ↪ 56(84) bytes of data.

--- 192.168.7.102 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203792ms
rtt min/avg/max/mdev = 0.215/0.364/0.690/0.071 ms
```

Listing F.27: S1-C latency outside VPN tunnel in the URLLC slice.

```
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I ens5 192.168.9.159
PING 192.168.9.159 (192.168.9.159) from 192.168.9.57 ens5: 56(84)
    ↪  bytes of data.

--- 192.168.9.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203789ms
rtt min/avg/max/mdev = 0.196/0.367/1.364/0.088 ms
```

Listing F.28: S1-U latency outside tunnel in the URLLC slice.

```
eNB: iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.248.159
ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600
    ↪  -i 60
Connecting to host 192.168.248.159, port 5201
local 192.168.248.157 port 49755 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  9.69 GBytes  1.39 Gbits/sec  287   1.24 MBytes
 60.00-120.00 sec  9.99 GBytes  1.43 Gbits/sec  321   1.15 MBytes
120.00-180.00 sec  9.82 GBytes  1.41 Gbits/sec  263   1.34 MBytes
180.00-240.00 sec  9.89 GBytes  1.42 Gbits/sec    6   1.33 MBytes
240.00-300.00 sec  10.2 GBytes  1.47 Gbits/sec  225   1.33 MBytes
300.00-360.00 sec  10.0 GBytes  1.43 Gbits/sec  191   1.60 MBytes
360.00-420.00 sec  9.89 GBytes  1.42 Gbits/sec  321   1.60 MBytes
420.00-480.00 sec  10.2 GBytes  1.45 Gbits/sec  265   1.60 MBytes
480.00-540.00 sec  10.1 GBytes  1.45 Gbits/sec    2   1.53 MBytes
540.00-600.00 sec  10.2 GBytes  1.46 Gbits/sec  130   1.53 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   100 GBytes  1.43 Gbits/sec  2011    sender
  0.00-600.00 sec   100 GBytes  1.43 Gbits/sec          receiver
```

Listing F.29: S1-U throughput in WireGuard tunnel in the URLLC slice.

```
ubuntu@enb:~$ iperf3 -B 192.168.247.111 -c 192.168.247.12 -t 600
    ↪ -i 60
Connecting to host 192.168.247.12, port 5201
local 192.168.247.111 port 33125 connected to 192.168.247.12 port
    ↪  5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  7.91 GBytes  1.13 Gbits/sec   12   1.40 MBytes
 60.00-120.00 sec  7.97 GBytes  1.14 Gbits/sec   11   1.13 MBytes
120.00-180.00 sec  7.46 GBytes  1.07 Gbits/sec   18   1.06 MBytes
180.00-240.00 sec  7.81 GBytes  1.12 Gbits/sec   34   1.02 MBytes
240.00-300.00 sec  7.65 GBytes  1.10 Gbits/sec   10   1.36 MBytes
300.00-360.00 sec  7.64 GBytes  1.09 Gbits/sec   11   1.31 MBytes
360.00-420.00 sec  8.01 GBytes  1.15 Gbits/sec   28   1.38 MBytes
420.00-480.00 sec  8.05 GBytes  1.15 Gbits/sec   11   1.01 MBytes
480.00-540.00 sec  7.89 GBytes  1.13 Gbits/sec   11   1.22 MBytes
540.00-600.00 sec  7.74 GBytes  1.11 Gbits/sec   10   1.28 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  78.1 GBytes  1.12 Gbits/sec  156     sender
  0.00-600.00 sec  78.1 GBytes  1.12 Gbits/sec          receiver
```

Listing F.30: S1-C throughput in WireGuard tunnel in the URLLC slice.

F.2.2 Running Tests Simultaneously on NSIs to See Any Difference in Performance

The tests are done doing the same measurement in both NSIs at the same time.

```
Command for MME in eMBB slice: ping -q -i 0.2 -c 1000 -I wg0
    ↪ 172.16.6.129

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.129
PING 172.16.6.129 (172.16.6.129) from 172.16.6.28 wg0: 56(84)
    ↪ bytes of data.

--- 172.16.6.129 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203075ms
rtt min/avg/max/mdev = 0.474/0.857/2.099/0.150 ms

Command for MME in URLLC slice: ping -q -i 0.2 -c 1000 -I wg0
    ↪ 172.16.6.129
ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.129
PING 172.16.6.129 (172.16.6.129) from 172.16.6.28 wg0: 56(84)
    ↪ bytes of data.
```

```
--- 172.16.6.129 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203539ms
rtt min/avg/max/mdev = 0.388/0.778/1.871/0.135 ms
```

Listing F.31: S6a latency with WireGuard.

```
eNB in eMBB slice: ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159
PING 192.168.248.159 (192.168.248.159) from 192.168.248.157 wg0:
    ↪ 56(84) bytes of data.

--- 192.168.248.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203709ms
rtt min/avg/max/mdev = 0.376/0.667/1.733/0.148 ms

eNB in URLLC slice: ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159
PING 192.168.248.159 (192.168.248.159) from 192.168.248.157 wg0:
    ↪ 56(84) bytes of data.

--- 192.168.248.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203688ms
rtt min/avg/max/mdev = 0.417/0.709/4.310/0.178 ms
```

Listing F.32: S1-U latency with WireGuard in two NSIs measured simultaneously.

```
eNB eMBB:
ping -q -i 0.2 -c 1000 -I wg2 192.168.247.102
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.102
PING 192.168.247.102 (192.168.247.102) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.102 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203196ms
rtt min/avg/max/mdev = 0.531/0.819/2.410/0.138 ms

URLLC:
ping -q -i 0.2 -c 1000 -I wg2 192.168.247.102
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.102
```

```
PING 192.168.247.102 (192.168.247.102) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.102 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203304ms
rtt min/avg/max/mdev = 0.512/0.836/5.858/0.196 ms
```

Listing F.33: S1-C latency with WireGuard in two NSIs measured simultaneously.

```
Command for eNB in eMBB slice: iperf3 -B 192.168.248.157 -c
    ↪ 192.168.248.159 -t 600 -i 60
Command for SPGW-U in eMBB slice: sudo iperf3 -s -B
    ↪ 192.168.248.159

ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600
    ↪ -i 60
Connecting to host 192.168.248.159, port 5201
local 192.168.248.157 port 54567 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  10.2 GBytes  1.46 Gbits/sec  580   1.53 MBytes
 60.00-120.00 sec  10.4 GBytes  1.49 Gbits/sec  348   1.53 MBytes
120.00-180.00 sec  10.3 GBytes  1.47 Gbits/sec  264   1.53 MBytes
180.00-240.00 sec  10.3 GBytes  1.48 Gbits/sec  206   1.53 MBytes
240.00-300.00 sec  10.2 GBytes  1.46 Gbits/sec  212   1.53 MBytes
300.00-360.00 sec  10.2 GBytes  1.47 Gbits/sec   14   1.53 MBytes
360.00-420.00 sec  9.99 GBytes  1.43 Gbits/sec  163   1.53 MBytes
420.00-480.00 sec  10.3 GBytes  1.48 Gbits/sec  565   1.53 MBytes
480.00-540.00 sec  10.3 GBytes  1.47 Gbits/sec  350   1.13 MBytes
540.00-600.00 sec  10.3 GBytes  1.48 Gbits/sec    9   1.62 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   103 GBytes  1.47 Gbits/sec  2711    sender
  0.00-600.00 sec   103 GBytes  1.47 Gbits/sec          receiver

Sample of workload on eNB when running the throughput measurement
    ↪ :
Tasks: 115 total, 1 running, 59 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.7 us, 0.3 sy, 0.0 ni, 98.8 id, 0.2 wa, 0.0 hi, 0.0 si,
    ↪ 0.1 st
KiB Mem: 8167500 total, 302264 free, 1046496 used, 6818740 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 6785260 avail Mem
 PID   USER PR NI VIRT    RES    SHR   S %CPU %MEM TIME+
    ↪ COMMAND
```

```
 24996 root 20 0  1477892 940512 29400 S 6.0  11.5 259:36.65 lte-
    ↪ softmodem

Command in eNB in URLLC slice: iperf3 -B 192.168.248.157 -c
    ↪ 192.168.248.159 -t 600 -i 60
Command in SPGW-U in URLLC slice: sudo iperf3 -s -B
    ↪ 192.168.248.159

ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -t 600
    ↪  -i 60
Connecting to host 192.168.248.159, port 5201
local 192.168.248.157 port 45247 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  9.67 GBytes  1.38 Gbits/sec   65   1.54 MBytes
 60.00-120.00 sec  9.58 GBytes  1.37 Gbits/sec  122   1.54 MBytes
120.00-180.00 sec  9.91 GBytes  1.42 Gbits/sec  173   1.54 MBytes
180.00-240.00 sec  9.78 GBytes  1.40 Gbits/sec  171   1.54 MBytes
240.00-300.00 sec  9.92 GBytes  1.42 Gbits/sec  580   1.54 MBytes
300.00-360.00 sec  9.79 GBytes  1.40 Gbits/sec  169   1.43 MBytes
360.00-420.00 sec  9.81 GBytes  1.40 Gbits/sec  323   1.54 MBytes
420.00-480.00 sec  9.92 GBytes  1.42 Gbits/sec  408   1.33 MBytes
480.00-540.00 sec  9.91 GBytes  1.42 Gbits/sec  397   1.53 MBytes
540.00-600.00 sec  9.89 GBytes  1.42 Gbits/sec  182   1.53 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  98.2 GBytes  1.41 Gbits/sec  2590    sender
  0.00-600.00 sec  98.2 GBytes  1.41 Gbits/sec          receiver

Sample of workload on eNB when running the throughput measurement
    ↪ :
Tasks: 115 total, 1 running, 59 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.9 us, 0.5 sy, 0.0 ni, 98.4 id, 0.2 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 8167500 total, 269228 free, 1043232 used, 6855040 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 6788528 avail Mem
 PID   USER PR NI VIRT    RES    SHR   S %CPU %MEM TIME+
    ↪ COMMAND
 25717 root 20 0  1543508 940432 29400 S 5.6  11.5 179:16.43 lte-
    ↪ softmodem
```

Listing F.34: S1-U throughput in WireGuard tunnel measured simultaneously for NSIs.

```
eNB in eMBB slice: iperf3 -B 192.168.9.57 -c 192.168.9.159 -t 600
    ↪ -i 60
SPGW-U in eMBB slice: sudo iperf3 -s -B 192.168.9.159

ubuntu@enb:~$ iperf3 -B 192.168.9.57 -c 192.168.9.159 -t 600 -i
    ↪ 60
Connecting to host 192.168.9.159, port 5201
local 192.168.9.57 port 57997 connected to 192.168.9.159 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   127 GBytes  18.2 Gbits/sec    0   3.12 MBytes
 60.00-120.00 sec   123 GBytes  17.6 Gbits/sec    0   3.12 MBytes
120.00-180.00 sec   125 GBytes  17.8 Gbits/sec    0   3.12 MBytes
180.00-240.00 sec   127 GBytes  18.2 Gbits/sec    0   3.12 MBytes
240.00-300.00 sec   126 GBytes  18.1 Gbits/sec    0   3.12 MBytes
300.00-360.00 sec   127 GBytes  18.1 Gbits/sec    0   3.12 MBytes
360.00-420.00 sec   126 GBytes  18.1 Gbits/sec    0   3.12 MBytes
420.00-480.00 sec   128 GBytes  18.3 Gbits/sec    0   3.12 MBytes
480.00-540.00 sec   129 GBytes  18.4 Gbits/sec    0   3.12 MBytes
540.00-600.00 sec   123 GBytes  17.7 Gbits/sec    0   3.12 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  18.1 Gbits/sec    0    sender
  0.00-600.00 sec  0.00 (null)s  18.1 Gbits/sec         receiver


eNB in eMBB slice: iperf3 -B 192.168.9.57 -c 192.168.9.159 -t 600
    ↪ -i 60
SPGW-U in eMBB slice: sudo iperf3 -s -B 192.168.9.159

ubuntu@enb:~$ iperf3 -B 192.168.9.57 -c 192.168.9.159 -t 600 -i
    ↪ 60
Connecting to host 192.168.9.159, port 5201
local 192.168.9.57 port 49537 connected to 192.168.9.159 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   147 GBytes  21.0 Gbits/sec    0   3.02 MBytes
 60.00-120.00 sec   139 GBytes  19.9 Gbits/sec    0   3.02 MBytes
120.00-180.00 sec   145 GBytes  20.7 Gbits/sec    0   3.02 MBytes
180.00-240.00 sec   151 GBytes  21.7 Gbits/sec    0   3.02 MBytes
240.00-300.00 sec   155 GBytes  22.2 Gbits/sec    0   3.02 MBytes
300.00-360.00 sec   157 GBytes  22.4 Gbits/sec    0   3.02 MBytes
360.00-420.00 sec   150 GBytes  21.5 Gbits/sec    0   3.02 MBytes
420.00-480.00 sec   156 GBytes  22.4 Gbits/sec    0   3.02 MBytes
```

```
480.00-540.00 sec   152 GBytes  21.7 Gbits/sec    0   3.02 MBytes
540.00-600.00 sec   148 GBytes  21.2 Gbits/sec    0   3.02 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  21.5 Gbits/sec    0    sender
  0.00-600.00 sec  0.00 (null)s  21.5 Gbits/sec         receiver
```

Listing F.35: S1-U throughput outside tunnel measured simultaneously on NSIs.

```
MME in eMBB slice: iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362
    ↪ -t 600 -i 60
HSS in eMBB slice: sudo iperf3 -s -B 172.16.6.129

ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 41141 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  5.76 GBytes   824 Mbits/sec   12    998 KBytes
 60.00-120.00 sec  5.82 GBytes   833 Mbits/sec   10    916 KBytes
120.00-180.00 sec  5.78 GBytes   827 Mbits/sec   10   1.06 MBytes
180.00-240.00 sec  5.85 GBytes   837 Mbits/sec   10   1.12 MBytes
240.00-300.00 sec  5.73 GBytes   820 Mbits/sec   11    910 KBytes
300.00-360.00 sec  5.66 GBytes   810 Mbits/sec    9   1.23 MBytes
360.00-420.00 sec  5.86 GBytes   838 Mbits/sec   11    910 KBytes
420.00-480.00 sec  5.85 GBytes   837 Mbits/sec   10   1.03 MBytes
480.00-540.00 sec  5.68 GBytes   813 Mbits/sec   10   1.14 MBytes
540.00-600.00 sec  5.50 GBytes   787 Mbits/sec   12    991 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  57.5 GBytes   823 Mbits/sec  105    sender
  0.00-600.00 sec  57.5 GBytes   823 Mbits/sec         receiver


MME in URLLC slice: iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362
    ↪  -t 600 -i 60
HSS in URLLC slice: sudo iperf3 -s -B 172.16.6.129

ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 50831 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  5.57 GBytes   798 Mbits/sec   11   1.10 MBytes
 60.00-120.00 sec  5.74 GBytes   821 Mbits/sec   11    778 KBytes
120.00-180.00 sec  5.56 GBytes   796 Mbits/sec   10    968 KBytes
```

```
180.00-240.00  sec  5.58 GBytes   799 Mbits/sec   10    961 KBytes
240.00-300.00  sec  5.44 GBytes   779 Mbits/sec   11    870 KBytes
300.00-360.00  sec  5.53 GBytes   791 Mbits/sec   10   1.09 MBytes
360.00-420.00  sec  5.71 GBytes   817 Mbits/sec   10   1.09 MBytes
420.00-480.00  sec  5.59 GBytes   801 Mbits/sec   11    885 KBytes
480.00-540.00  sec  5.57 GBytes   797 Mbits/sec   10    919 KBytes
540.00-600.00  sec  5.47 GBytes   783 Mbits/sec   10   1.18 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec  55.8 GBytes   798 Mbits/sec  104    sender
  0.00-600.00  sec  55.8 GBytes   798 Mbits/sec         receiver
```

Listing F.36: S6a throughput with WireGuard tunnel in two NSIs measured simultaneously.

```
eMBB slice:
ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 40759 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00   sec  7.61 GBytes  1.09 Gbits/sec   12   1.32 MBytes
 60.00-120.00  sec  7.53 GBytes  1.08 Gbits/sec   11   1.02 MBytes
120.00-180.00  sec  7.74 GBytes  1.11 Gbits/sec   11    963 KBytes
180.00-240.00  sec  7.63 GBytes  1.09 Gbits/sec   10   1.09 MBytes
240.00-300.00  sec  7.76 GBytes  1.11 Gbits/sec   10   1.30 MBytes
300.00-360.00  sec  7.64 GBytes  1.09 Gbits/sec   10   1.41 MBytes
360.00-420.00  sec  7.48 GBytes  1.07 Gbits/sec   12    993 KBytes
420.00-480.00  sec  7.27 GBytes  1.04 Gbits/sec   10   1.14 MBytes
480.00-540.00  sec  6.97 GBytes   998 Mbits/sec   11   1.07 MBytes
540.00-600.00  sec  6.65 GBytes   952 Mbits/sec   11   1.08 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec  74.3 GBytes  1.06 Gbits/sec  108    sender
  0.00-600.00  sec  74.3 GBytes  1.06 Gbits/sec         receiver


URLLC slice:
ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.129 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.129, port 5201
local 172.16.6.28 port 50013 connected to 172.16.6.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00   sec  7.58 GBytes  1.09 Gbits/sec   12   1.02 MBytes
 60.00-120.00  sec  7.50 GBytes  1.07 Gbits/sec   10   1.10 MBytes
120.00-180.00  sec  7.48 GBytes  1.07 Gbits/sec   10   1.19 MBytes
```

```
180.00-240.00  sec  7.58 GBytes  1.09 Gbits/sec   10   1.25 MBytes
240.00-300.00  sec  7.67 GBytes  1.10 Gbits/sec   10   1.31 MBytes
300.00-360.00  sec  7.50 GBytes  1.07 Gbits/sec   10   1.29 MBytes
360.00-420.00  sec  7.36 GBytes  1.05 Gbits/sec   10   1.32 MBytes
420.00-480.00  sec  7.10 GBytes  1.02 Gbits/sec   11   1.05 MBytes
480.00-540.00  sec  7.12 GBytes  1.02 Gbits/sec   10   1.25 MBytes
540.00-600.00  sec  6.64 GBytes   950 Mbits/sec   11   1006 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec  73.5 GBytes  1.05 Gbits/sec  104     sender
  0.00-600.00  sec  73.5 GBytes  1.05 Gbits/sec          receiver
```

Listing F.37: S6a throughput with WireGuard tunnel - run 2.

```
Throughput between management interfaces between SPGW-U and eNB
    ↪ in eMBB slice:
iperf3 -B 192.168.222.37 -c 192.168.222.59 -M 1362 -t 600 -i 60
eNB: sudo iperf3 -s -B 192.168.222.59

ubuntu@spgw-u:~$ iperf3 -B 192.168.222.37 -c 192.168.222.59 -M
    ↪ 1362 -t 600 -i 60
Connecting to host 192.168.222.59, port 5201
local 192.168.222.37 port 43597 connected to 192.168.222.59 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00   sec   101 GBytes  14.5 Gbits/sec    0   4.01 MBytes
 60.00-120.00  sec  98.7 GBytes  14.1 Gbits/sec    0   4.01 MBytes
120.00-180.00  sec   100 GBytes  14.3 Gbits/sec    0   4.01 MBytes
180.00-240.00  sec  98.4 GBytes  14.1 Gbits/sec    0   4.01 MBytes
240.00-300.00  sec  98.2 GBytes  14.1 Gbits/sec    0   4.01 MBytes
300.00-360.00  sec   100 GBytes  14.3 Gbits/sec    0   4.01 MBytes
360.00-420.00  sec   100 GBytes  14.3 Gbits/sec    0   4.01 MBytes
420.00-480.00  sec  99.9 GBytes  14.3 Gbits/sec    0   4.01 MBytes
480.00-540.00  sec   102 GBytes  14.6 Gbits/sec    0   4.01 MBytes
540.00-600.00  sec  98.1 GBytes  14.0 Gbits/sec    0   4.01 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00  sec   997 GBytes  14.3 Gbits/sec    0     sender
  0.00-600.00  sec   997 GBytes  14.3 Gbits/sec          receiver


Throughput between management interfaces between SPGW-U and eNB
    ↪ in URLLC slice:
SPGW-U: iperf3 -B 192.168.222.207 -c 192.168.222.165 -M 1362 -t
    ↪ 600 -i 60
eNB: sudo iperf3 -s -B 192.168.222.165
```

```
ubuntu@spgw-u:~$ iperf3 -B 192.168.222.207 -c 192.168.222.165 -M
    ↪ 1362 -t 600 -i 60
Connecting to host 192.168.222.165, port 5201
local 192.168.222.207 port 45307 connected to 192.168.222.165
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   143 GBytes  20.5 Gbits/sec    0   3.10 MBytes
 60.00-120.00 sec   150 GBytes  21.5 Gbits/sec    0   3.10 MBytes
120.00-180.00 sec   152 GBytes  21.8 Gbits/sec    0   3.10 MBytes
180.00-240.00 sec   164 GBytes  23.6 Gbits/sec    0   3.10 MBytes
240.00-300.00 sec   159 GBytes  22.7 Gbits/sec    0   3.10 MBytes
300.00-360.00 sec   162 GBytes  23.1 Gbits/sec    0   3.10 MBytes
360.00-420.00 sec   160 GBytes  22.9 Gbits/sec    0   3.10 MBytes
420.00-480.00 sec   152 GBytes  21.7 Gbits/sec    0   3.10 MBytes
480.00-540.00 sec   167 GBytes  23.8 Gbits/sec    0   3.10 MBytes
540.00-600.00 sec   158 GBytes  22.6 Gbits/sec    0   3.10 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  22.4 Gbits/sec    0    sender
  0.00-600.00 sec  0.00 (null)s  22.4 Gbits/sec         receiver
```

Listing F.38: S1-U throughput between management interfaces in two NSIs measured simultaneously.

```
eMBB slice:
ubuntu@enb:~$ iperf3 -B 192.168.247.111 -c 192.168.247.12 -t 600
    ↪ -i 60
Connecting to host 192.168.247.12, port 5201
local 192.168.247.111 port 44757 connected to 192.168.247.12 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  6.46 GBytes   925 Mbits/sec   12   1.17 MBytes
 60.00-120.00 sec  6.66 GBytes   954 Mbits/sec   12    908 KBytes
120.00-180.00 sec  6.84 GBytes   979 Mbits/sec   14   1.20 MBytes
180.00-240.00 sec  7.49 GBytes  1.07 Gbits/sec   13   1.37 MBytes
240.00-300.00 sec  7.19 GBytes  1.03 Gbits/sec   11   1.19 MBytes
300.00-360.00 sec  7.09 GBytes  1.01 Gbits/sec   11   1.07 MBytes
360.00-420.00 sec  7.29 GBytes  1.04 Gbits/sec   89   1.32 MBytes
420.00-480.00 sec  6.60 GBytes   944 Mbits/sec   11   1.31 MBytes
480.00-540.00 sec  6.57 GBytes   941 Mbits/sec   12   1.20 MBytes
540.00-600.00 sec  7.15 GBytes  1.02 Gbits/sec   10   1.29 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  69.3 GBytes   993 Mbits/sec  195    sender
  0.00-600.00 sec  69.3 GBytes   993 Mbits/sec         receiver
```

```
URLLC slice:
ubuntu@enb:~$ iperf3 -B 192.168.247.111 -c 192.168.247.12 -t 600
    ↪ -i 60
Connecting to host 192.168.247.12, port 5201
local 192.168.247.111 port 37809 connected to 192.168.247.12 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  6.19 GBytes   886 Mbits/sec   13   1.21 MBytes
 60.00-120.00 sec  6.44 GBytes   922 Mbits/sec   12    890 KBytes
120.00-180.00 sec  6.71 GBytes   961 Mbits/sec   11   1.16 MBytes
180.00-240.00 sec  7.47 GBytes  1.07 Gbits/sec   10   1.20 MBytes
240.00-300.00 sec  7.11 GBytes  1.02 Gbits/sec   11   1.03 MBytes
300.00-360.00 sec  6.87 GBytes   983 Mbits/sec   11   1.13 MBytes
360.00-420.00 sec  6.88 GBytes   984 Mbits/sec   34   1.25 MBytes
420.00-480.00 sec  6.59 GBytes   943 Mbits/sec   11   1.27 MBytes
480.00-540.00 sec  6.81 GBytes   975 Mbits/sec   11   1.05 MBytes
540.00-600.00 sec  6.93 GBytes   992 Mbits/sec   10   1.36 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  68.0 GBytes   973 Mbits/sec  134     sender
  0.00-600.00 sec  68.0 GBytes   973 Mbits/sec          receiver
```

Listing F.39: S1-C throughput with WireGuard in two NSIs measured simultaneously.

```
Command run in SPGW-U in eMBB slice:ping -q -i 0.2 -c 1000 -I
    ↪ ens3 10.21.21.2

ubuntu@spgw-u:~$ ping -q -i 0.2 -c 1000 -I ens3 10.21.21.2
PING 10.21.21.2 (10.21.21.2) from 192.168.222.37 ens3: 56(84)
    ↪ bytes of data.

--- 10.21.21.2 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203753ms
rtt min/avg/max/mdev = 0.107/0.225/2.765/0.130 ms

Command run in SPGW-U in URLLC slice: ping -q -i 0.2 -c 1000 -I
    ↪ ens3 10.21.21.2

ubuntu@spgw-u:~$ ping -q -i 0.2 -c 1000 -I ens3 10.21.21.2
PING 10.21.21.2 (10.21.21.2) from 192.168.222.207 ens3: 56(84)
    ↪ bytes of data.
```

```
--- 10.21.21.2 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203786ms
rtt min/avg/max/mdev = 0.110/0.216/3.344/0.118 ms
```

Listing F.40: Measurement of latency between SPGW-U and Microstack VIM router when measuring for two NSIs simultaneously.

```
Command run in SPGW-U toward eNB in eMBB slice: ping -q -i 0.2 -c
    ↪  1000 -I ens3 192.168.222.59

ubuntu@spgw-u:~$ ping -q -i 0.2 -c 1000 -I ens3 192.168.222.59
PING 192.168.222.59 (192.168.222.59) from 192.168.222.37 ens3:
    ↪ 56(84) bytes of data.

--- 192.168.222.59 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203779ms
rtt min/avg/max/mdev = 0.217/0.372/2.516/0.107 ms

Command run in SPGW-U toward eNB in URLLC slice: ping -q -i 0.2 -
    ↪ c 1000 -I ens3 192.168.222.165

ubuntu@spgw-u:~$ ping -q -i 0.2 -c 1000 -I ens3 192.168.222.165
PING 192.168.222.165 (192.168.222.165) from 192.168.222.207 ens3:
    ↪  56(84) bytes of data.

--- 192.168.222.165 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203789ms
rtt min/avg/max/mdev = 0.193/0.386/2.456/0.103 ms
```

Listing F.41: S1-U management interface latency.

F.3 Double Resources of vCPU and RAM

The measurements done in the following section uses the NS described in Section 4.8.

```
eNB: iperf3 -B 192.168.248.157 -c 192.168.248.159 -M 1362 -t 600
    ↪ -i 60
SPGW-U: sudo iperf3 -s -B 192.168.248.159

ubuntu@enb:~$ iperf3 -B 192.168.248.157 -c 192.168.248.159 -M
    ↪ 1362 -t 600 -i 60
Connecting to host 192.168.248.159, port 5201
```

```
local 192.168.248.157 port 56551 connected to 192.168.248.159
    ↪ port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  14.9 GBytes  2.13 Gbits/sec  470   1.44 MBytes
 60.00-120.00 sec  15.0 GBytes  2.14 Gbits/sec  751   1.94 MBytes
120.00-180.00 sec  15.6 GBytes  2.23 Gbits/sec  1166  1.08 MBytes
180.00-240.00 sec  15.5 GBytes  2.23 Gbits/sec  219   1.52 MBytes
240.00-300.00 sec  14.8 GBytes  2.12 Gbits/sec  1110  1.52 MBytes
300.00-360.00 sec  15.1 GBytes  2.16 Gbits/sec  892   968 KBytes
360.00-420.00 sec  15.2 GBytes  2.18 Gbits/sec  1127  1.48 MBytes
420.00-480.00 sec  15.2 GBytes  2.18 Gbits/sec  573   1.63 MBytes
480.00-540.00 sec  16.0 GBytes  2.29 Gbits/sec  973   1.58 MBytes
540.00-600.00 sec  15.8 GBytes  2.26 Gbits/sec  867   1.34 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   153 GBytes  2.19 Gbits/sec  8148   sender
  0.00-600.00 sec   153 GBytes  2.19 Gbits/sec         receiver
```

Listing F.42: S1-U throughput in WireGuard tunnel in EPS NS with double resources.

```
eNB: iperf3 -B 192.168.9.57 -c 192.168.9.159 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.9.159

ubuntu@enb:~$ iperf3 -B 192.168.9.57 -c 192.168.9.159 -t 600 -i
    ↪ 60
Connecting to host 192.168.9.159, port 5201
local 192.168.9.57 port 45627 connected to 192.168.9.159 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   116 GBytes  16.6 Gbits/sec    0   3.08 MBytes
 60.00-120.00 sec  97.4 GBytes  13.9 Gbits/sec    0   3.08 MBytes
120.00-180.00 sec   104 GBytes  14.9 Gbits/sec    0   3.08 MBytes
180.00-240.00 sec   137 GBytes  19.7 Gbits/sec    0   3.08 MBytes
240.00-300.00 sec   142 GBytes  20.3 Gbits/sec    0   3.08 MBytes
300.00-360.00 sec   117 GBytes  16.7 Gbits/sec    0   3.08 MBytes
360.00-420.00 sec   157 GBytes  22.5 Gbits/sec    0   3.08 MBytes
420.00-480.00 sec   147 GBytes  21.1 Gbits/sec    0   3.08 MBytes
480.00-540.00 sec   145 GBytes  20.7 Gbits/sec    0   3.08 MBytes
540.00-600.00 sec   146 GBytes  20.9 Gbits/sec    0   3.08 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  18.7 Gbits/sec    0   sender
  0.00-600.00 sec  0.00 (null)s  18.7 Gbits/sec        receiver
```

Listing F.43: S1-U throughput outside tunnel in EPS NS with double resources.

```
eNB: iperf3 -B 192.168.247.101 -c 192.168.247.12 -t 600 -i 60
MME: sudo iperf3 -s -B 192.168.247.12

ubuntu@enb:/usr/local/etc/oai$ iperf3 -B 192.168.247.101 -c
    ↪ 192.168.247.12 -t 600 -i 60
Connecting to host 192.168.247.12, port 5201
local 192.168.247.101 port 54293 connected to 192.168.247.12 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  10.9 GBytes  1.56 Gbits/sec  2954  1.43 MBytes
 60.00-120.00 sec  9.05 GBytes  1.30 Gbits/sec  707   1.22 MBytes
120.00-180.00 sec  8.88 GBytes  1.27 Gbits/sec  738   1.20 MBytes
180.00-240.00 sec  10.6 GBytes  1.52 Gbits/sec  894   1.15 MBytes
240.00-300.00 sec  11.0 GBytes  1.58 Gbits/sec  1796  1.22 MBytes
300.00-360.00 sec  11.0 GBytes  1.58 Gbits/sec  2636  1.13 MBytes
360.00-420.00 sec  9.89 GBytes  1.42 Gbits/sec  994   1.10 MBytes
420.00-480.00 sec  9.72 GBytes  1.39 Gbits/sec  999   1.47 MBytes
480.00-540.00 sec  10.4 GBytes  1.48 Gbits/sec  2229   568 KBytes
540.00-600.00 sec  10.5 GBytes  1.50 Gbits/sec  2148  1.35 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   102 GBytes  1.46 Gbits/sec  16095     sender
  0.00-600.00 sec   102 GBytes  1.46 Gbits/sec          receiver
```

Listing F.44: S1-C throughput with WireGuard in EPS NS with double resources.

```
MME: iperf3 -B 172.16.6.28 -c 172.16.6.128 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 172.16.6.128

ubuntu@mme:~$ iperf3 -B 172.16.6.28 -c 172.16.6.128 -M 1362 -t
    ↪ 600 -i 60
Connecting to host 172.16.6.128, port 5201
local 172.16.6.28 port 43557 connected to 172.16.6.128 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  7.84 GBytes  1.12 Gbits/sec   18   1.30 MBytes
 60.00-120.00 sec  8.01 GBytes  1.15 Gbits/sec  288   1.30 MBytes
120.00-180.00 sec  7.86 GBytes  1.13 Gbits/sec  265    945 KBytes
180.00-240.00 sec  7.17 GBytes  1.03 Gbits/sec  547   1.31 MBytes
240.00-300.00 sec  7.14 GBytes  1.02 Gbits/sec   52   1.27 MBytes
300.00-360.00 sec  7.45 GBytes  1.07 Gbits/sec  201   1.18 MBytes
360.00-420.00 sec  6.98 GBytes  1000 Mbits/sec   11   1.21 MBytes
420.00-480.00 sec  7.51 GBytes  1.07 Gbits/sec   11   1.18 MBytes
480.00-540.00 sec  7.33 GBytes  1.05 Gbits/sec   11   1.34 MBytes
540.00-600.00 sec  7.56 GBytes  1.08 Gbits/sec   11   1.16 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
```

```
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  74.9 GBytes  1.07 Gbits/sec  1415     sender
  0.00-600.00 sec  74.8 GBytes  1.07 Gbits/sec           receiver
```

Listing F.45: S6a throughput in WireGuard tunnel in EPS NS with double resources.

```
MME: iperf3 -B 192.168.8.2 -c 192.168.8.129 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 192.168.8.129

ubuntu@mme:~$ iperf3 -B 192.168.8.2 -c 192.168.8.1 -M 1362 -t 600
    ↪ -i 60
Connecting to host 192.168.8.1, port 5201
local 192.168.8.2 port 49709 connected to 192.168.8.1 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   114 GBytes  16.3 Gbits/sec    0   3.14 MBytes
 60.00-120.00 sec   102 GBytes  14.6 Gbits/sec    0   3.14 MBytes
120.00-180.00 sec   101 GBytes  14.5 Gbits/sec    0   3.14 MBytes
180.00-240.00 sec   102 GBytes  14.6 Gbits/sec    0   3.14 MBytes
240.00-300.00 sec  99.7 GBytes  14.3 Gbits/sec    0   3.14 MBytes
300.00-360.00 sec  99.2 GBytes  14.2 Gbits/sec    0   3.14 MBytes
360.00-420.00 sec  99.4 GBytes  14.2 Gbits/sec    0   3.14 MBytes
420.00-480.00 sec   101 GBytes  14.4 Gbits/sec    0   3.14 MBytes
480.00-540.00 sec   100 GBytes  14.4 Gbits/sec    0   3.14 MBytes
540.00-600.00 sec   100 GBytes  14.4 Gbits/sec    0   3.14 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  1019 GBytes  14.6 Gbits/sec    0     sender
  0.00-600.00 sec  1019 GBytes  14.6 Gbits/sec           receiver
```

Listing F.46: S6a throughput outside tunnel in EPS NS with double resources.

```
MME: iperf3 -B 192.168.8.2 -c 192.168.8.129 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 192.168.8.129

For run two, we deleted the previous deployment and started fresh
    ↪ with a new instantiation.

ubuntu@mme:~$ iperf3 -B 192.168.8.2 -c 192.168.8.129 -t 600 -i 60
Connecting to host 192.168.8.129, port 5201
local 192.168.8.2 port 54147 connected to 192.168.8.129 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec   104 GBytes  14.9 Gbits/sec    0   3.15 MBytes
 60.00-120.00 sec   122 GBytes  17.4 Gbits/sec    0   3.15 MBytes
120.00-180.00 sec   142 GBytes  20.3 Gbits/sec    0   3.15 MBytes
180.00-240.00 sec   147 GBytes  21.1 Gbits/sec    0   3.15 MBytes
```

```
240.00-300.00 sec   147 GBytes  21.1 Gbits/sec    0   3.15 MBytes
300.00-360.00 sec   134 GBytes  19.2 Gbits/sec    0   3.15 MBytes
360.00-420.00 sec   128 GBytes  18.3 Gbits/sec    0   3.15 MBytes
420.00-480.00 sec   133 GBytes  19.0 Gbits/sec    0   3.15 MBytes
480.00-540.00 sec   142 GBytes  20.3 Gbits/sec    0   3.15 MBytes
540.00-600.00 sec   142 GBytes  20.3 Gbits/sec    0   3.15 MBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  0.00 (null)s  19.2 Gbits/sec    0    sender
  0.00-600.00 sec  0.00 (null)s  19.2 Gbits/sec         receiver


Sample of workload on the MME VNF when running the throughput
    ↪ measurement:
top - 07:44:35 up 12:24, 1 user, load average: 0.36, 0.44, 0.27
Tasks: 112 total, 1 running, 60 sleeping, 0 stopped, 0 zombie
top - 07:44:44 up 12:24, 1 user, load average: 0.30, 0.43, 0.27
Tasks: 112 total, 1 running, 60 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.1 sy, 0.0 ni, 99.8 id, 0.0 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 8167492 total, 5774884 free, 180288 used, 2212320 buff/
    ↪ cache
KiB Swap: 0 total, 0 free, 0 used. 7680980 avail Mem

 PID  USER PR NI VIRT    RES   SHR   S %CPU %MEM TIME+   COMMAND
 7919 root 20 0  2477252 65744 11952 S 0.3  0.8  1:15.35 mme

Sample of workload on the HSS VNF when running the throughput
    ↪ measurement:
top - 07:45:24 up 12:24, 1 user, load average: 0.32, 0.58, 0.38
Tasks: 144 total, 1 running, 78 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.0 si,
    ↪  0.0 st
KiB Mem: 16424316 total, 9318308 free, 4634508 used, 2471500 buff
    ↪ /cache
KiB Swap: 0 total, 0 free, 0 used. 11424236 avail Mem

 PID   USER      PR NI VIRT    RES    SHR   S %CPU %MEM TIME+
    ↪ COMMAND
 13893 cassand+ 20 0  6214336 4.283g 52772 S 0.7  27.3 4:41.91
    ↪ java
```

Listing F.47: Run two of S6a throughput measurement outside VPN tunnel in EPS NS with double resources.

```
ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg0 192.168.248.159
PING 192.168.248.159 (192.168.248.159) from 192.168.248.157 wg0:
    ↪ 56(84) bytes of data.

--- 192.168.248.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 202620ms
rtt min/avg/max/mdev = 0.460/0.889/1.968/0.131 ms
```

Listing F.48: S1-U latency in WireGuard tunnel in EPS NS with double resources.

```
eNB: ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 201791ms
rtt min/avg/max/mdev = 0.550/0.963/1.506/0.157 ms
```

Listing F.49: S1-C latency in WireGuard tunnel in EPS NS with double resources.

```
eNB: ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I wg2 192.168.247.12
PING 192.168.247.12 (192.168.247.12) from 192.168.247.111 wg2:
    ↪ 56(84) bytes of data.

--- 192.168.247.12 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 202014ms
rtt min/avg/max/mdev = 0.553/0.946/1.956/0.150 ms
```

Listing F.50: Run 2 of latency measurements on the S1-C interface in WireGuard tunnel in EPS NS with double resources.

```
eNB: ping -q -i 0.2 -c 1000 -I ens4 192.168.7.102

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.7.102
PING 192.168.7.102 (192.168.7.102) from 192.168.7.101 ens4:
    ↪ 56(84) bytes of data.

--- 192.168.7.102 ping statistics ---
```

```
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203789ms
rtt min/avg/max/mdev = 0.232/0.393/2.221/0.091 ms
```

Listing F.51: S1-C latency outside tunnel in EPS NS with double resources.

```
eNB: ping -q -i 0.2 -c 1000 192.168.9.159

ubuntu@enb:~$ ping -q -i 0.2 -c 1000 192.168.9.159
PING 192.168.9.159 (192.168.9.159) 56(84) bytes of data.

--- 192.168.9.159 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203793ms
rtt min/avg/max/mdev = 0.194/0.351/0.652/0.074 ms
```

Listing F.52: S1-U latency outside tunnel in EPS NS with double resources.

```
MME: ping -q -i 0.2 -c 1000 -I wg0 172.16.6.128

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.128
PING 172.16.6.128 (172.16.6.128) from 172.16.6.28 wg0: 56(84)
    ↪ bytes of data.

--- 172.16.6.128 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 201767ms
rtt min/avg/max/mdev = 0.510/0.960/1.940/0.148 ms
```

Listing F.53: S6a latency with WireGuard tunnel in EPS NS with double resources.

```
MME: ping -q -i 0.2 -c 1000 -I ens4 192.168.8.129

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I ens4 192.168.8.2
PING 192.168.8.2 (192.168.8.2) from 192.168.8.2 ens4: 56(84)
    ↪ bytes of data.

--- 192.168.8.2 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 203793ms
rtt min/avg/max/mdev = 0.024/0.041/0.075/0.010 ms
```

Listing F.54: S6a latency outside tunnel in EPS NS with double resources.

```
UE: iperf3 -B 12.1.1.2 -c 192.168.222.137 -M 1362 -t 600 -i 60
SPGW-U: sudo iperf3 -s -B 192.168.222.137
```

```
ubuntu@ue:~$ iperf3 -B 12.1.1.2 -c 192.168.222.137 -M 1362 -t 600
    ↪ -i 60
Connecting to host 192.168.222.137, port 5201
local 12.1.1.2 port 39103 connected to 192.168.222.137 port 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  13.2 MBytes  1.85 Mbits/sec  173   25.0 KBytes
 60.00-120.00 sec  12.1 MBytes  1.69 Mbits/sec  206   21.1 KBytes
120.00-180.00 sec  12.2 MBytes  1.71 Mbits/sec  184   27.7 KBytes
180.00-240.00 sec  12.0 MBytes  1.68 Mbits/sec  203   25.0 KBytes
240.00-300.00 sec  12.0 MBytes  1.68 Mbits/sec  130    108 KBytes
300.00-360.00 sec  13.0 MBytes  1.82 Mbits/sec  153   15.8 KBytes
360.00-420.00 sec  11.6 MBytes  1.62 Mbits/sec  148   22.4 KBytes
420.00-480.00 sec  10.8 MBytes  1.51 Mbits/sec  119   42.2 KBytes
480.00-540.00 sec  12.8 MBytes  1.79 Mbits/sec  121   46.1 KBytes
540.00-600.00 sec  11.4 MBytes  1.60 Mbits/sec  169   29.0 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec   121 MBytes  1.70 Mbits/sec  1606    sender
  0.00-600.00 sec   121 MBytes  1.69 Mbits/sec          receiver
```

Listing F.55: S6a throughput with WireGuard in EPS NS with double resources.

```
HSS: 08:39:52 up 13:19,  1 user,  load average: 0.01, 0.02, 0.00
MME: 08:39:31 up 13:19,  1 user,  load average: 0.08, 0.02, 0.01
SPGW-U: 08:39:05 up 13:17,  1 user,  load average: 0.00, 0.00,
    ↪ 0.00
SPGW-C:  08:38:39 up 13:18,  1 user,  load average: 0.00, 0.00,
    ↪ 0.00
ENB:  08:40:19 up 13:16,  1 user,  load average: 0.05, 0.06, 0.01
```

Listing F.56: Sample of load average in VNFs after initial setup.

F.4 Multi-site Deployment

The following measurements are run using the NS and procedure described in Section 4.10.

```
MME: iperf3 -B 172.16.6.2 -c 172.16.6.128 -M 1362 -t 600 -i 60
HSS: sudo iperf3 -s -B 172.16.6.128

ubuntu@mme:~$ iperf3 -B 172.16.6.2 -c 172.16.6.128 -M 1224 -t 600
    ↪ -i 60
Connecting to host 172.16.6.128, port 5201
local 172.16.6.2 port 43481 connected to 172.16.6.128 port 5201
```

```
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  1.02 GBytes   146 Mbits/sec  510    328 KBytes
 60.00-120.00 sec  1.01 GBytes   145 Mbits/sec  115    496 KBytes
120.00-180.00 sec  1.12 GBytes   160 Mbits/sec   61    665 KBytes
180.00-240.00 sec  1.13 GBytes   161 Mbits/sec   67    634 KBytes
240.00-300.00 sec  1.11 GBytes   159 Mbits/sec  688    627 KBytes
300.00-360.00 sec  1.11 GBytes   160 Mbits/sec  133    489 KBytes
360.00-420.00 sec  1.11 GBytes   159 Mbits/sec   73    645 KBytes
420.00-480.00 sec  1.11 GBytes   158 Mbits/sec  251    621 KBytes
480.00-540.00 sec  1.12 GBytes   161 Mbits/sec   69    628 KBytes
540.00-600.00 sec  1.08 GBytes   155 Mbits/sec   87    615 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  10.9 GBytes   156 Mbits/sec  2054    sender
  0.00-600.00 sec  10.9 GBytes   156 Mbits/sec          receiver
```

Listing F.57: S6a throughput between VIMs - with WireGuard (MME and HSS management interfaces as endpoints).

```
Command run in the MME VNF in VIM 1: iperf3 -c 10.20.20.1 -t 600
    ↪ -i 60
Command run in VM running MicroStack for VIM 2: sudo iperf3 -s -B
    ↪  10.20.20.1

ubuntu@mme:~$ iperf3 -c 10.20.20.1 -t 600 -i 60
Connecting to host 10.20.20.1, port 5201
local 192.168.222.136 port 41184 connected to 10.20.20.1 port
    ↪ 5201
Interval           Transfer     Bandwidth       Retr  Cwnd
  0.00-60.00  sec  1.25 GBytes   179 Mbits/sec  199    557 KBytes
 60.00-120.00 sec  1.25 GBytes   180 Mbits/sec   89    596 KBytes
120.00-180.00 sec  1.25 GBytes   179 Mbits/sec   85    632 KBytes
180.00-240.00 sec  1.22 GBytes   175 Mbits/sec   87    480 KBytes
240.00-300.00 sec  1.25 GBytes   179 Mbits/sec   52    620 KBytes
300.00-360.00 sec  1.23 GBytes   176 Mbits/sec   93    553 KBytes
360.00-420.00 sec  1.25 GBytes   180 Mbits/sec   73    513 KBytes
420.00-480.00 sec  1.24 GBytes   178 Mbits/sec   58    561 KBytes
480.00-540.00 sec  1.26 GBytes   180 Mbits/sec   69    604 KBytes
540.00-600.00 sec  1.25 GBytes   180 Mbits/sec   99    472 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Interval           Transfer     Bandwidth       Retr
  0.00-600.00 sec  12.5 GBytes   179 Mbits/sec  904    sender
  0.00-600.00 sec  12.5 GBytes   179 Mbits/sec         receiver
```

Listing F.58: S6a throughput between VIMs - without WireGuard.

```
Command in MME VNF at VIM 1 towards management interface of HSS
    ↪ VNF at VIM 2: ping -q -i 0.2 -c 1000 10.20.20.118

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 10.20.20.118
PING 10.20.20.118 (10.20.20.118) 56(84) bytes of data.

--- 10.20.20.118 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 200589ms
rtt min/avg/max/mdev = 14.962/18.355/389.972/17.291 ms, pipe 2
```

Listing F.59: S6a latency between VIMs - without WireGuard.

```
Command in MME VNF at VIM 1 towards WireGuard interface of HSS
    ↪ VNF at VIM 2: ping -q -i 0.2 -c 1000 -I wg0 172.16.6.128

ubuntu@mme:~$ ping -q -i 0.2 -c 1000 -I wg0 172.16.6.128
PING 172.16.6.128 (172.16.6.128) from 172.16.6.2 wg0: 56(84)
    ↪ bytes of data.

--- 172.16.6.128 ping statistics ---
1000 packets transmitted, 1000 received, 0% packet loss, time
    ↪ 200589ms
rtt min/avg/max/mdev = 15.293/19.769/439.485/18.749 ms, pipe 3
```

Listing F.60: S6a latency between VIMs - with WireGuard.

Appendix G
Research Paper

Providing network slice isolation with WireGuard in beyond 5G

Sondre Kielland, Ali Esmaeily, Katina Kralevska, and Danilo Gligoroski
Dep. of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU)
Email: {sondrki, ali.esmaeily, katinak, danilo.gligoroski}@ntnu.no

***Abstract*—The introduction of network slicing in 5G networks has authorized verticals to deploy their services alongside other applications over a shared infrastructure. Nevertheless, security is still one of the main challenges for shared infrastructure. In this paper, we study how WireGuard can provide an encrypted VPN tunnel as a service between network functions in 5G and beyond setting. The Open Source Management and Orchestration entity deploys and orchestrates the network functions into network services and slices. We create multiple scenarios emulating a real-life cellular network exposing VPN-as-a-Service between the different network functions to secure and isolate network slices. The performance measurements demonstrate from 0.8 Gbps to 2.5 Gbps throughput and under 1ms delay between network functions using WireGuard. These measurements are aligned with 5G key performance indicators, making WireGuard suited to provide security in slice isolation in 5G and beyond networks.**

***Index Terms*—OSM, WireGuard, VPN, NFV, 5G, Network slice**

I. Introduction

The enrollment of 5G Non-Standalone (NSA) cellular networks is already in operation by Mobile Network Operators (MNOs). In developing 5G networks, several planned functionalities will enable verticals to establish their services with diverse Quality of Service (QoS) requirements on shared physical infrastructure. Commercial products and new technologies in the 5G System (5GS) open up for MNOs and verticals deploy their applications in an agile way. Providing End-to-End (E2E) services over isolated network slices are key factors to empower multiple services on a shared infrastructure. To develop agile 5G networks for supporting applications with different QoS requirements, Network Function Virtualisation (NFV), Software-Defined Networking (SDN) and Multi-Access Edge Computing (MEC) are essential technologies [1]–[3].

An NFV Management and Orchestration (MANO) entity connected to one or several Virtual Infrastructure Managers (VIMs) is used to control and monitor the deployment of Network Services (NSs) by employing necessary infrastructure resources. For an agile network deployment, the NFV MANO also administrates connections between Virtual Network Functions (VNFs), including creation of virtual networks with help of SDN. Therefore, instead of manually creating and connecting the NSs together, the NFV MANO help verticals deploy and control Network Function (NF) programmatically. With its programmatic and reusable functionality a large amount of NFs and NSs can be rapidly deployed on a single or multiple VIMs.

Using multiple VIMs can be used with benefit for all three main use cases in 5G, Ultra Reliable Low Latency Communication (URLLC), enhanced Mobile BroadBand (eMBB), and Massive Machine-type Communication (mMTC) [4], [5]. For instance can use of MEC reduce latency and decrease the volume of network traffic going back to a core network.

Using hypervisors or cloud infrastructure that are rented or shared is necessary to utilize resources efficiently for financial and load distribution purposes. However, introducing shared infrastructure also raises further security challenges. Securing application data transferred over shared networks is one example of a security challenge for shared environments. A countermeasure that can be initiated against such security concerns is operating Virtual Private Network (VPN) between NFs. However, establishing VPN tunnels introduce additional overhead. For applications dependent on low latency or high throughput, the additional overhead may be problematic if it affects the performance of the application.

Additional security characteristic can also be introduced for NFs in NSs with the NFV MANO which is able to deploy real-life applications in a programmatic way. By deploying VPN tunneling feature between NFs and connecting them, the confidentiality of application data can be achieved. Besides, such secure tunneling contributes to isolate Network Slice Instances (NSIs) and the provided NSs via the NSIs. Nevertheless, this approach is only feasible if the VPN does not introduce significant overhead violating QoS requirements.

How a VPN can be deployed between VNF in an automatic mode to enhance security isolation between slices and how the overhead may affect the performance isolation among slices in a shared environment are not clearly specified. In this paper, we use WireGuard to demonstrate an approach to utilize a VPN application in a 5G environment with real-life functionality. Open Source MANO (OSM) is employed to orchestrate NSs and NSIs, and establish VPN tunnels between the VNFs. Our approach exhibits how WireGuard deployed in a virtualized environment can satisfy security, throughput, and latency performance requirements.

Our contribution: In this paper, we present an integrated WireGuard-OSM architecture that ensures security and performance isolation between network slices. This integrated architecture enables us to 1) Provide secured communication between the involving VNFs of NSs and NSIs. Such capability, in turn, presents security isolation among slices. 2) Besides, our architecture also provides performance isolation

between slices. The performance analysis tests confirm that our integrated architecture fulfills the required KPI values in terms of high throughput for the eMBB slices and low latency for the URLLC slices according to their corresponding QoS requirements. 3) Moreover, by employing open-source solutions, our architecture also grants multi-site deployment, showcasing more realistic service development scenarios.

II. Related Work

As introduced in [6], the isolation concept between network slices can be studied from security, performance, and dependability aspects. In addition, the Confidentiality, Integrity and Availability (CIA) triad is a widely used way of looking at different security aspects. A shared infrastructure introduces security challenges in all dimensions of the CIA triad. The key for shared infrastructures is that an attack on or from another party sharing the infrastructure should not affect others. This definition of CIA is harmonic with the isolation concept in network slicing. Other parties should also be unaffected when it comes to performance and dependability, extending the availability dimension. Workload, amount of resources, and hardware or software failure of other NS should not reduce the performance of an NF in a separate NS or NSI.

While 5G intends to fix some security issues present in the previous generations of cellular networks, it also introduces several new security threats. Some of them raised by providing services via network slices. Paper [7] explores and classifies different security challenges of 5G networks. Proper isolation of logical resources is essential to avoid introducing several new risks. Eavesdropping and tampering with data, for instance, are two vectors an attacker could use to interfere with security if application data is not properly encrypted.

An example of where NFV and SDN open up for new functionality is in Service Function (SF) Chaining (SFC). Hantouti et al. suggest that operators should deploy encrypted tunnels as a way to establish trust between SFs to provide packet integrity and prevent bypassing of policies [8].

Further, a specific use case to utilize shared infrastructure is MEC. In [9], Nencioni et al. studies security, performance, and in particular, dependability challenges when using MEC.

The work in [10] proposes a novel mutual authentication and key establishment protocol utilizing proxy re-encryption. The protocol grants specific authentication between components of a network slice to enable secure connection protected key establishment among component pairs for slice security isolation. Paper [11] offers a secure keying scheme by adopting a multi-party computation strategy, which is appropriate for network slicing architecture in the case that third-party applications access the slices. This mechanism ensures the satisfaction of use cases or devices in which the data is collected.

Both Haga et al. in [12] and Vidal et al. in [13] focus on how a VPN can be deployed using OSM. [12] demonstrates how WireGuard can be added in VNFs and compares WireGuard and OpenVPN performances. This practical work is carried out using two VNFs in a single NS with manual configuration of peer connectivity in WireGuard. For the peer setup, keys and other necessary information are obtained manually.

Vidal et al. in [13] uses IPsec as VPN solution to provide link-layer connectivity for multi-site deployments. In this work, OSM is employed to deploy multiple NSs connected through one VNF at each NFV Infrastructure (NFVI). These VNFs handles the link layer abstraction for the other VNFs. IPsec is used to secure the connection between the link layer providing VNFs. Keys and connection parameters are supplied by the operator when instantiating the NSI.

As evident from the mentioned papers, none of them provides a secured service automation provisioning utilizing complex and real-life NFs. This motivates us to integrate WireGuard tunneling with OSM, which grants secure communication between the involving NFs to establish automated and realistic network services. As a result, this system architecture guarantees security and performance isolation between NSIs.

III. System Architecture

Day-0, Day-1, and Day-2 operations are terminologies used in the OSM community referring to the stages of Life-Cycle Management (LCM) of NFs. The steps of the *operations* phase in Figure 1 are used to handle LCM of NFs via the NF onboarding process and they closely link to Day-0 to Day-2 operations. The construction of charms and descriptors beforehand are illustrated in the *development* part of the figure.

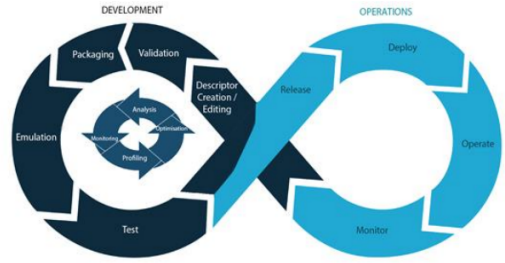

Fig. 1. Steps for service lifecycle [14].

OSM has three inbuilt supporting applications for LCM [15]. Cloud-init is responsible for the initial Day-0 operations like setting username and password. For Day-1 operations, Helm charts or Juju charms can be used, while Day-2 operations are also possible with Juju. Another difference between Helm and Juju is that Helm is used solely for Kubernetes-based Network Functions (KNFs), while Juju is also usable at NS level and for VNFs that are not Kubernetes (K8s) based [16], [17]. We have used cloud-init and Juju charms for OSM onboarding in our implementations.

Further, Juju has two operation modes. Native charms run operations directly inside a VNF. On the other hand, proxy charms use a centrally placed controller, a VNF Configuration and Abstraction (VCA), to manage the Day-1 and Day-2 actions. The VCA connects to the VNFs through their management interface and instructs the VNFs. The VCA-VNF

connection uses the Secure Shell (SSH) protocol by default. In the paper, we have used proxy charms with a VCA installed co-located and integrated with OSM. Both the VCA and OSM should therefore be able to access the VNFs management interface to execute their actions.

To build user-defined actions, Juju uses Python scripts. The connection to the OSM instance is made through the description files of the VNFs, NSs, Juju config files describing metadata, and the available Day-1 and Day-2 actions. For the OSM integration of proxy charms, the *charms.osm.sshproxy* library is provided by OSM to take care of, among other tasks, the basic Juju proxy peer setup.

In addition to running actions in VNFs, Juju can be used to create relations between Juju units for management, scaling, and for handling dependencies across VNFs. In this paper, we will use Juju relations to transfer WireGuard peer information between VNFs.

Figure 2 illustrates how we have used proxy charms and relations in Juju to create a bridge for transferring information between VNFs. The figure shows the architecture for the multi-site demonstration. However, for the main performance measurements, we have used a single-VIM, moving also the Home Subscriber Server (HSS) into *VIM 1*. The architecture for the single-VIM setup is as illustrated at the rightmost half of the figure showing *VIM 1*.

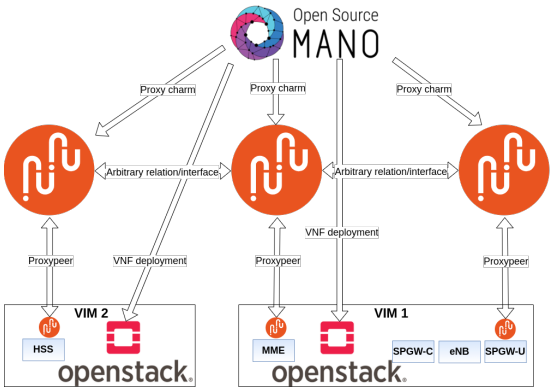

Fig. 2. Interactions between elements in our Juju proxy implementation.

Distributing keys is a task that often requires manual steps when establishing a VPN tunnel. Manual setup can be time-consuming for dynamic environments or environments with many interfaces to be secured. If the tenant manager needs to do configuration, the NS is only usable after initializing the VPN tunnels. However, if using the approach by Vidal et al. and input the necessary information, including keys, the application can start sending data immediately after Day-1 actions have finished. A similar approach is using a Key Management System (KMS). However, OSM does not provide such functionality. To use the KMS approach additional functionality outside of the OSM framework must be added.

To perform key management, we have used a non-standard approach using Juju relations with the requirement of using proxy charms for our VNFs. By using Juju relations, we create new individual keys for every new deployment for the different interfaces and make the application of the NS usable directly after the Day-1 tasks finish. Furthermore, with our approach, the private keys are only stored inside the VNFs. The public key and other necessary information for the peer setup get automatically transferred to the peer.

IV. Implementation

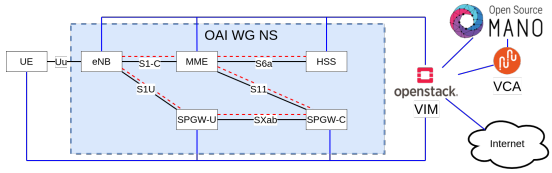

Fig. 3. Architecture of our implementation.

To enable WireGuard in a realistic environment we have created a NS with Evolved Packet System (EPS) components from Open Air Interfaces (OAIs) [18]. We have then added WireGuard connectivity on the different interfaces. Figure 3 shows the architecture we have employed. OSM is used to communicate to a MicroStack VIM. The VIM is hosting different VNFs, creating virtual networks and performing routing of outgoing traffic from the VNFs represented with the blue lines. A WireGuard tunnel is created automatically between the VNFs on the interfaces in the NS, represented by the red dotted lines. In addition to the primary VIM, we have utilized a second VIM. The second VIM have been used to explore the EPS NS deployment in multiple sites.

A. Development

The steps we have followed to prepare our deployments are; **composing a virtualized EPS**, set up a mechanism for **automatic WireGuard peering**, structuring NSs into **Network Slice Template (NST) packages**, and lastly test the WireGuard connectivity in a **multi-site deployment**. The descriptors and charm code we have used are made available on GitHub.[1] In the following paragraphs, we further describe the development steps to create the descriptors and scripts.

a) Composing a Virtualized EPS: In [19], Dreibholz implements an Evolved Packet Core (EPC) with HSS, Mobility Management Entity (MME), and a combined Serving Gateway (SGW) and Packet Data Network (PDN) Gateway (PGW) separated in two components, Service Packet Gateway-User plane (SPGW-U) and Service Packet Gateway-Control plane (SPGW-C), for user- and control-plane tasks respectively. To extend this NS with real-life traffic we add a virtualized eNodeB (eNB). Further, we create an User Equipment (UE) in a Virtual Machine (VM) kept outside the NS. The UE is

[1] https://github.com/sondrki/TTM4905/

still able to connect to the eNB after instantiating the NS with manual network setup in MicroStack. To establish the air interface, Uu, we have compiled and used OAIs simulation option. When connecting the UE to the eNB we have verified that the different EPS components functions as expected and provides service for the UE. The functionality includes that the UE can connect to outer network through the SPGW-U via the eNB. At this fist step of implementation, we have not included WireGuard between the components.

We have chosen to build our NS spreading the EPS components into separate VNFs. With this approach, we are able to split out VNFs to other VIMs. By being able to extend to multi-site environments, we can emulate a scenario where other components, for instance, MEC is deployed closer to the end-users. When adding WireGuard, the VNFs distributed to remote sites should be able to communicate back to the core securely.

b) Automatic WireGuard Peering: Manually setup can be time-consuming for several interfaces in which we would like to VPN tunnels. We use Juju relations for automatic peering with no extra information given to the other side of the peer at the time of instantiating the NS. The first step in the automatic peering is establishing relationships between VNFs on both sides. Then the paired VNFs retrieve information like *public key*, *endpoint*, and *listening port* to communicate this information for retrieval on the other side.

To establish WireGuard connectivity on all interfaces given in Figure 3, we have changed the IP address configuration in the components. Changing the interface addresses is necessary to route application data over the VPN tunnel and at the same time to ensure that applications inside the VNF have been installed and started correctly even when waiting for the tunnel establishment. Besides, to verify the NS with WireGuard, we connect the UE and observe that it connects and gets PDN service.

Further, to observe how resources affect the WireGuard performance, we have prepared a copy of the EPS NS with WireGuard connectivity with double resources.

c) NST Packaging: After having a working NS with WireGuard connectivity between the interfaces, we include it in two NSTs to observe if and how the performance is affected. The two NSTs are prepared with different values for quality indicators corresponding to different 5G QoS Identifiers (5QIs) [20]. The QoS parameters we have chosen are usable for eMBB and URLLC applications respectively. Further, the NST is prepared with only the management interface of the VNFs being external connection points in the NSTs.

d) Multi-site Deployment: To verify that the automatic peer setup also works in a multi-site condition, we have separated the HSS VNF to a second VIM. When using OpenStack/MicroStack, the external, floating IP address is by default not known inside a VM. However, the VCA can retrieve the management IP address to perform its actions. To find the floating IP addresses of the VNFs, we use the same function as Juju employs for its *proxypeer* connection between a Juju unit at the VCA and the Virtual Deployment Unit (VDU) in the VNF. After the endpoint IP address is found, the MME and HSS connect automatically with WireGuard connectivity, likewise in this scenario. A prerequisite for multi-site WireGuard connectivity is to use a port opened in the firewalls.

B. Proof of Concept for VPN-as-a-Service

With the automatic peering, we present a few-step procedure to add WireGuard as a VPN-as-a-Service (VPNaaS). The steps pursued in our proof of concept are summarized in the following. The additions required for the different files, referenced to use WireGuard as a VPNaaS, are available on GitHub.[2]

1) Append installation of WireGuard in cloud-init.
2) Add name and parameters for Day-1 and Day-2 actions in the actions.yaml file.
3) Add relations between VNFs in the metadata.yaml file.
4) Include the Python code to append the charm script. The name of the relationship must correspond between the name used in metadata.yaml and the listener in the *__init__* function of the Python script.
5) Add the actions from actions.yaml into *Day-1, Day-2 operations* in the VNF Descriptor (VNFD). To create the WireGuard tunnel as a Day-1 operation, the relevant actions should be included in the *initial-config-primitive* section in the VNFDs. Day-2 actions are placed in the *config-primitive* section.
6) While the default implementation sets up the VPN, Day-2 actions can be used for further configuration and maintenance, for instance, if a new connection should be added towards an NF.

V. Performance Evaluation

To observe how introducing WireGuard in a cellular network in a 5G context, we have done performance tests in multiple ways. We have done tests in both the control and user plane, with and without WireGuard with both arbitrary data and using the UE to generate realistic traffic in the network. Since one UE is not producing a large number of packets in the control plane, throughput measurements for the UE are experienced to easiest create large amount packets using realistic protocols in the user plane.

While only producing arbitrary data for high network load in the control plane, we have measured the latency and Service Response Time (SRT) in the control plane combining multiple EPS components. In general, have we performed the following tasks to test the performance in our NSs and NSIs.

- Observe SRT on the MME when a UE connects.
- Observe throughput and latency in the user plane with a UE.
- Measure throughput and latency between components in the EPS.

[2]https://github.com/sondrki/TTM4905/tree/main/vpnaas

A. Lab Environment

The primary VIM have been a server running MicroStack on top with resources of 56 virtual Central Processing Units (vCPUs), 126 GB Random Access Memory (RAM) and 915 GB storage. The second VIM used for multi-site deployment is also running MicroStack but have less resources with the total of 9 vCPUs, 32 GB RAM and 150 GB storage. For the EPS NS a total of 14 vCPU, 27 GB RAM and 110 GB storage have been utilized. According to the limiting ISP, the bandwidth between the two NFVIs is specified to be 200 Mbps. For VNFs to communicate across the VIMs a WireGuard tunnel has been established between the NFVIs. Our measurement shows a throughput between the MicroStack instances of approximately 180 Mbps. When adding WireGuard on the S6a interface for the multi-site deployment, a nested WireGuard tunnel is therefore used. In addition is the internal throughput of the NFVI where the primary VIM runs, measured to around 20 Gbps.

The resources we have used for the VNFs are assembled in Table I. The resources are similar for all use cases we have tested, except for the NS where we double the RAM and vCPU for all VMs but the UE.

TABLE I
VNF INFORMATION OF THE OAI EPS NS.

VNF name	Operating System	number of virtual CPUs	amount of RAM (GB)	storage (GB)
HSS	ubuntu18.04	4	8.0	20
MME	ubuntu18.04	2	4.0	20
SPGWU	ubuntu18.04	1	3.0	20
SPGWC	ubuntu18.04	3	4.0	30
eNB	ubuntu18.04	4	8.0	20
UE	ubuntu18.04	2	4.0	20

B. Observations

Before adding the VPN tunnels, we were able to capture connection information like the International Mobile Subscriber Identity (IMSI), network realms, and hostnames at the VIM. However, after we introduce WireGuard, the only information observable at the VIM is the use of the WireGuard protocol and link-layer discovery messages.

For the control plane data flow we have observed how the SRT is for the HSS application responding to a connecting UE. When monitoring the HSS application SRT including networking from the MME, we observed that the NS with WireGuard was the one with the lowest average SRT. With ten successful connections for the UE the SRT measured average latency drops from 6.156 ms for the EPS without WireGuard to 5.377 ms when WireGuard is added. When doubling the resources on the EPS NS with WireGuard a SRT of 5.607 ms is measured. Based on the other measurements, it is likely that the HSS application itself is the delaying part. With a reduced number of connections, we have not observed a negative effect on the SRT when using WireGuard.

A comparison of the latency measurements done for the different instances and interfaces is shown in Figure 4. The red line in the figure indicates 1 ms, representing one of the E2E Key Performance Indicator (KPI) supporting URLLC applications in 5G. All single-site instances in the figure are lower than the 1 ms line. However, adding WireGuard introduces a visible overhead when comparing the NS without WireGuard to the other instances in the figure. On the other hand, we observe that the average latencies for the S1-C interface in the NSIs are lower than for the other measurements. The differences between instances and interfaces tell us that the latency may be dependent on multiple factors like 5QI parameters and workload of components in an NS.

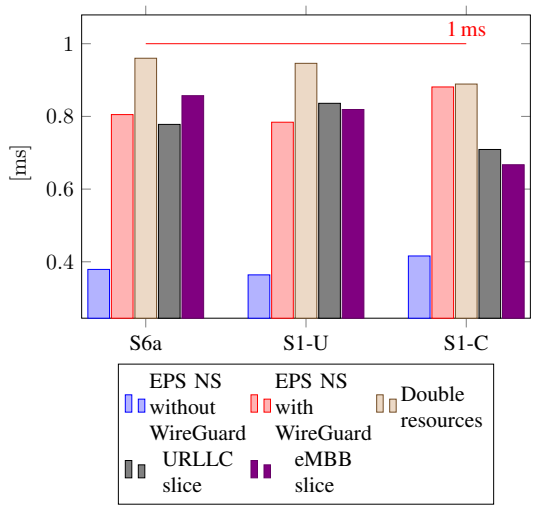

Fig. 4. Latency comparison for different interfaces with WireGuard.

Figure 5 compares the throughput between components with WireGuard on the different interfaces across instances. The red line represents the 100 Mbps Downlink user data rate KPI. From the throughput comparison, we highlight three main results. The first one is that the throughput changes according to the available resources. When comparing the NS with double resources to the others, we notice that the throughput is higher with the double resources NS. A tenant manager can therefore adjust the WireGuard performance by tuning the resources. The second aspect we will highlight is that the throughput of the UE measuring towards the SPGW-U is significantly lower than the other measurements. The throughput over the Uu and S1-U is measured to around 1.7 Mbps, while the average throughput for the S1-U alone averages over 1 Gbps. We also measure the same for the NS without WireGuard, making the Uu the bottleneck of our EPS. The last highlight is the maximum throughput we have measured when averaging over 10 minutes. For the NS with double resources, we observed throughput of 2.2 Gbps. For the other instances, a range from 770 Mbps to 1.48 Gbps is measured.

The multi-site deployment is measured only over the S6a

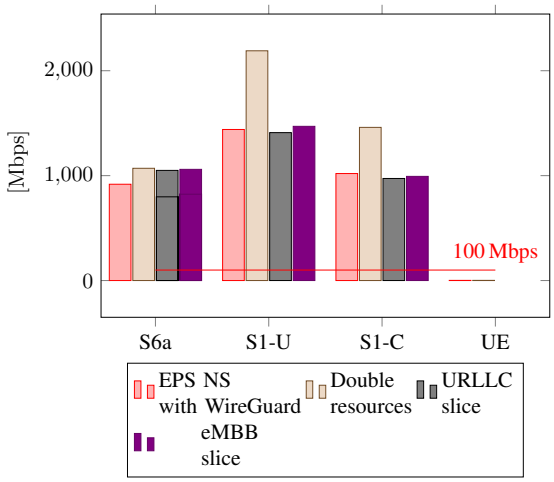

Fig. 5. Throughput comparison for different interfaces with WireGuard.

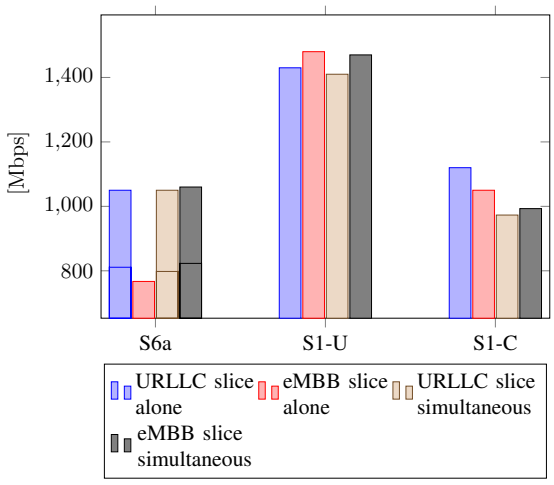

Fig. 6. Throughput comparison with WireGuard for NSIs - measured separately and simultaneously.

interface, which is the one that differs from the other NSs and NSIs. The throughput is significantly lower and the latency higher compared to the other instances as expected as the performance is lower also without WireGuard between the VNFs. However, we observe that WireGuard adds an overhead also in this scenario. For the multi-site NS an average latency over 1000 ICMP packets went from 18.355 ms to 19.769 ms when using WireGuard. For the average throughput, we observe a reduction from 179 Mbps to 156 Mbps, which is expected based on the given 200 Mbps bandwidth.

Figure 6 compares the throughput in the two NSIs. In the figure, we also observe if there are differences when running alone and with workload simultaneously. Like the other deployments without an NST, we observe that the performance can differ for the same NS or NSI when deploying a second time. For instance, is the throughput of the S6a interface changing from around 800 Mbps to 1.1 Gbps when we build the NSIs as second time. In general, the differences between the NSIs are only minor and inconsistent. All our observations fulfill the requirements set in the QoS parameters. Therefore, the results are as we expected.

When testing with putting a workload on the same logical interface for the two NSIs simultaneously, we observe a total throughput of approximately 3 Gbps. As 3 Gbps is significantly lower than the internal networking of around 20 Gbps, we may observe larger differences between the NSI based on their QoS parameters when we close in on the internal networking limit.

VI. Conclusions

By using Juju relations and providing a proof of concept for adding WireGuard as a VPNaaS we have shown that WireGuard can be implemented with automatic peer setup after instantiating. The performance measurements we have done demonstrate that WireGuard is suitable for applications with requirements corresponding to several of the 5G KPI values. However, we have observed that the performance of WireGuard is dependent on available resources. Therefore, a tenant manager should be aware of how tuning can affect the overall performance. We have observed that WireGuard can be used as a VPNaaS in the context of 5G networks and beyond to provide secure communication to support isolation.

We have identified several directions a new project can derive from our project described in this paper. Replacing the arbitrary Juju relations with a KMS, using 5G Core (5GC) instead of EPC components, and adding multiple UEs are some of the suggestions that can be valuable to study further.

References

[1] B. Blanco, J. O. Fajardo, I. Giannoulakis, E. Kafetzakis, S. Peng, J. Pérez-Romero, I. Trajkovska, P. Sayyad Khodashenas, L. Goratti, M. Paolino, and E. Sfakianakis, "Technology pillars in the architecture of future 5g mobile networks: Nfv, mec and sdn," *Computer Standards and Interfaces*, vol. 54, 01 2017.

[2] K. Kralevska, M. Garau, M. Førland, and D. Gligoroski, "Towards 5g intrusion detection scenarios with omnet++," in *Proceedings of 6th International OMNeT++ Community Summit 2019*, ser. EPiC Series in Computing, M. Zongo, A. Virdis, V. Vesely, Z. Vatandas, A. Udugama, K. Kuladinithi, M. Kirsche, and A. Förster, Eds., vol. 66. EasyChair, 2019, pp. 44–51. [Online]. Available: https://easychair.org/publications/paper/sNcK

[3] M. K. Forland, K. Kralevska, M. Garau, and D. Gligoroski, "Preventing ddos with sdn in 5g," in *2019 IEEE Globecom Workshops (GC Wkshps)*, 2019, pp. 1–7.

[4] ETSI, "Multi-access edge computing (mec)," https://www.etsi.org/technologies/multi-access-edge-computing.

[5] ——, "Why do we need 5g?" https://www.etsi.org/technologies/mobile/5g.

[6] A. J. Gonzalez, J. Ordonez-Lucena, B. E. Helvik, G. Nencioni, M. Xie, D. R. Lopez, and P. Grønsund, "The isolation concept in the 5g network slicing," IEEE, pp. 12–16, 2020.

[7] H. Kim, "5g core network security issues and attack classification from network protocol perspective." *J. Internet Serv. Inf. Secur.*, vol. 10, no. 2, pp. 1–15, 2020.

[8] H. Hantouti, N. Benamar, and T. Taleb, "Service function chaining in 5g amp; beyond networks: Challenges and open research issues," *IEEE Network*, vol. 34, no. 4, pp. 320–327, 2020.
[9] G. Nencioni, R. G. Garroppo, and R. F. Olimid, "5g multi-access edge computing: Security, dependability, and performance," *CoRR*, vol. abs/2107.13374, 2021. [Online]. Available: https://arxiv.org/abs/2107.13374
[10] V. Sathi, M. Srinivasan, P. Kaliyammal Thiruvasagam, and S. Chebiyyam, "A novel protocol for securing network slice component association and slice isolation in 5g networks," pp. 249–253, 10 2018.
[11] P. Porambage, Y. Miche, A. Kalliola, M. Liyanage, and M. Ylianttila, "Secure keying scheme for network slicing in 5g architecture," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2019, pp. 1–6.
[12] S. Haga, A. Esmaeily, K. Kralevska, and D. Gligoroski, "5g network slice isolation with wireguard and open source mano: A vpnaas proof-of-concept," IEEE, pp. 181–187, 2020.
[13] I. Vidal, B. Nogales, D. Lopez, J. Rodríguez, F. Valera, and A. Azcorra, "A secure link-layer connectivity platform for multi-site nfv services," *Electronics*, vol. 10, no. 15, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/15/1868
[14] 5G-PPP Architecture Working Group, "View on 5g architecture," https://tinyurl.com/2p9dxph4, accessed: 07.01.2022.
[15] ETSI OSM, "etsi-nfv-vnfd," https://osm.etsi.org/docs/user-guide/05-osm-usage.html.
[16] ——, "etsi-nfv-nsd," https://tinyurl.com/26dt45xv.
[17] ——, "etsi-nfv-vnfd," https://tinyurl.com/2p9yp7cr.
[18] O. S. Alliance, "Openairinterface," https://openairinterface.org/, accessed: 04.01.2022. [Online]. Available: https://openairinterface.org/
[19] T. Dreibholz, "Flexible 4g/5g testbed setup for mobile edge computing using openairinterface and open source mano," Caserta, Campania/Italy, 2020.
[20] ETSI, "System architecture for the 5g system (5gs)," ETSI, Tech. Rep. TS 123 501 V16.6.0, October 2020. [Online]. Available: https://tinyurl.com/2p8392jt

Sondre Sørensen Kielland

Secure communication with WireGuard - VPN-as-a-Service in beyond 5G

NTNU
Norwegian University of
Science and Technology