FACULTY OF SCIENCE AND TECHNOLOGY

BACHELOR THESIS

Study programme / specialisation:
Computer Science – Bachelor's Degree Programme

The spring semester, 2022

Open

Author: Sylwia Wasilewska

.....

(signature author)

Course coordinator:

Supervisor(s): Gianfranco Nencioni

Thesis title: Pre-study on Network Function Virtualization at Communication Technology Lab: Emulator

Credits (ECTS): 20

Keywords:
NFV
Emulation
Virtualization
Network Services

Pages: 38

+ appendix: 5

Stavanger, May 30, 2022

University of Stavanger

SYLWIA WASILEWSKA

DEPARTMENT OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Pre-study on Network Function Virtualization at Communication Technology Lab: Emulator

Bachelor's Thesis - Computer Science - May 2022

I, **Sylwia Wasilewska**, declare that this thesis titled, “Pre-study on Network Function Virtualization at Communication Technology Lab: Emulator” and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a bachelor’s degree at the University of Stavanger.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my work.
- I have acknowledged all main sources of help.

“Problems are not stop signs, they are guidelines.”

– Robert Schuller

Abstract

Network Function Virtualization (NFV) is a network architecture framework that virtualizes functions that were traditionally tied to hardware and allows for higher flexibility of deployment of network services. It is growing in popularity as more research is being conducted in the field, and because of this it is beneficial to investigate its possible uses in academia. The objectives of this thesis are to research NFV as a whole to investigate its different aspects, as well as research different NFV emulators. Later, the aim is to design tests that could be run in a selected emulator to showcase NFV academically to potential students.

Out of the three thoroughly investigated emulators, the MeDICINE emulator was selected based on several criteria, most notably the ease of installation. However, due to some major technical issues and time constraints, the emulator could not be run, meaning no tests could be executed. Six tests were designed with two different topologies to showcase as many aspects and functionalities of NFV as possible. These tests open up possibilities for future work in the area, where potential students could use these tests to investigate NFV further through running them and checking parameters, but also adapting the topologies to other emulators and comparing the process of running the tests in several different emulators.

Acknowledgements

I would like to thank my supervisor, Gianfranco Nencioni, for his enthusiasm and help with writing this thesis. Without his pointers and tips, my work would have been of much lower quality.

I would also like to thank the faculty, especially Erlend Tøssebro and Sheryl Josdal, for their continued support through the difficult situation and all issues that arose during this semester.

Finally, I would like to thank my family and friends for always having my back as I worked on my studies and writing this thesis. Without their support and advice, I would not have been able to work as efficiently and stay as positive during the tougher moments.

Contents

List of Figures

Chapter 1

Introduction

As technology advances, a need for more flexible solutions for network services arises. Networks grow fast, requiring methods of expanding that are both more cost-efficient and time-effective to implement. Because of this, the telecommunication industry creates and implements new concepts like Software Defined Networking (SDN) and Network Function Virtualization (NFV), which moves networking functionalities to software-based solutions.

SDN allows for dynamic control of networks by separating the control plane and the forwarding plane. This makes network functions possible to configure and manage through a centralized control point. Thanks to this, networks become scalable, dynamic, and agile, which in turn allows them to respond to business requirements that are changing more rapidly than ever, especially in modern data centers that use virtualized infrastructure, and provision the network as a service [3].

NFV, on the other hand, uses Virtual Machines (VMs) to replace network appliance hardware [4]. NFV uses a Virtual Machine Monitor (VMM) for running network software and virtualizing processes such as load balancing and routing. Moving these processes to the digital world allows for the development of more dynamic, fully automated virtual networks that are faster to grow and evolve with new functionalities, also requiring less trained maintenance staff and floor space [5]. While SDN and NFV are complementary, they are growing increasingly co-dependent [3]. This means that it is highly likely that both concepts will become integrated in the future.

There are many benefits to virtualizing network functionalities through the

use of NFV. It virtualizes the network services as a whole, but specifically certain aspects of it. The network operator will have advantages like reduced space required for hardware like routers and switches, a reduction in network power consumption, a drop in maintenance cost, as well as easier upgrades to the network and hardware having a longer lifecycle [5]. Using NFV makes building and configuring networks much more time-effective and cost-efficient, as the time used to manually connect and configure hardware is lost. However, there are also risks associated with using NFV. Most of these are security risks, like malware being difficult to isolate and contain, as well as network traffic becoming less transparent [4]. In addition, physical security controls become ineffective, and the network becomes more open to new types of attacks as opposed to physical equipment that is kept in a secure data center.

1.1 NFV History

In 2012, the European Telecommunications Standards Institute (ETSI) established the NFV Industry Specification Group (ETSI ISG NFV). Since then, the group has worked on setting standards through over a hundred publications, taking the project through several 2-year-long phases called Releases that had different focus areas ([6], [7]). In 2013-2014, the ISG NFV focused on building a culture and understanding of important concepts in network virtualization. The original vision was outlined in a publication from October 2012, where the publishers set ”pre-standardization” documents [8]. The work of the group built on this paper, and the first important milestone came a year later, when the first five ETSI Group Specifications were published in October 2013 [1]. These include NFV use cases, terminology, virtualization requirements, and an architectural framework. The following year, ISG NFV published a further 11 documents that focused in different technical areas of NFV, including management and orchestration, infrastructure, and the architecture of Virtualized Network Functions (VNFs) [6].

In NFV Release 2, spanning from 2015 to 2016, the work of ISG NFV was defined by ”selecting and prioritizing a set of key capabilities” for NFV deployment and ensuring ”interoperability of NFV solutions” when used at a larger scale [6]. These capabilities include lifecycle management, software image management, Network Service (NS) information modeling, hardware-independent acceleration, and more. Furthermore, testing specifications for Management and Or-

chestration (MANO) and Application Programming Interface (API) Conformance were also created in this Release. Release 3 focused mostly on preparing NFV for global deployment and operations. In 2017-2019, 22 new features categorized into three main areas were outlined, of which ten were completed, some were closed, two were partially completed, and others were carried over to Release 4. The three areas of features were advances in virtualization, support for new network technologies, and new operational aspects [6].

NFV Release 4 was officially launched in the summer of 2019, where ISG NFV worked on both new features and unfinished features that were carried over from the previous release. These include NFV-MANO automation, policy models, security and licensing management, and more. It appears that ETSI ISG NFV is still working on this Release as no new information is available, but documents are continuously being updated and created [6].

There are other open-source projects that are also striving to develop NFV standards, such as Metro Ethernet Forum (MEF), Open Network Automation Platform, Open Platform for NFV, and more. Even ETSI ISG NFV is open to both members and non-members, which allows people all over the world to contribute to the research and creation of new functionalities [7].

1.2 Objectives

Since NFV and NFV emulators are still very new on the market, the original aim of this thesis was to select two emulators and showcase how NFV works through designing tests and comparing how these tests are executed on both emulators. However, due to time constraints and technical issues, the objective first changed to only showcase NFV through one emulator; because the used computers did not correctly run the selected emulator, the objective changed again. This thesis will outline tests that can be used by potential students whose computers correctly run the emulator. These tests will showcase different aspects and functionalities of NFV, allowing for potential future students to learn more about the concept.

The objectives are then as follows:

- Studying the state-of-the-art NFV emulation frameworks;
- Evaluation and analysis of the best suitable emulator;

- Designing tests that focus on showcasing how NFV works with future work in mind;
- Presenting NFV academically to potential students.

1.3 Approach and Contributions

The approach to meet the objectives of the thesis is to research NFV as a whole, followed by researching NFV emulators, both commercial and open-source, with more focus on the open-source emulators. The best suitable emulator is then chosen judging by the availability, the specifications, and the ease of installation, among other criteria. Tests are designed to showcase several functionalities and aspects of NFV, with focus on prospects of future work that may be done by future students.

The main contributions of this thesis are:

- State of the art of the NFV emulators with description and systematic comparison;
- Design of tests or exercises that showcase NFV characteristics to potential students and open up the possibility for future works;

1.4 Outline

The structure of this thesis is as follows:

- Chapter 2 describes background information on NFV architecture, service composition, and differences between simulation, emulation, and experimentation. These are aspects that are part of any NFV emulator.
- Any related works and detailed descriptions of emulators are covered in Chapter 3, as well as a short comparison, which will allow for the selection of the most suitable emulator.
- Chapter 4 focuses on the installation and running of the selected emulator, as well as an evaluation of the emulator.

- The design of the tests and the academic aspect of these tests, as well as the academic possibilities for future work, will be covered in Chapter 5.
- Finally, the conclusions of this thesis will be in Chapter 6.

Chapter 2

Background

This chapter focuses on background information about NFV like its architecture and service composition, and the key differences between experimentation, simulation, and emulation.

2.1 NVF Architecture

NFV is comprised of three major components: the Virtualized Network Functions (VNF), the Network Functions Virtualization Infrastructure (NFVI), and the Management and Orchestration (MANO), as shown in Figure 2.1. This section will discuss these parts in more detail.

2.1.1 The Virtualized Network Functions (VNFs)

The Virtualized Network Functions (VNFs) is the collection of software applications that deliver the network functions like file sharing, IP configuration, and directory services [9]. VNFs are run in virtual machines on top of the networking hardware infrastructure, and they include routers, switches, firewalls, and more [10]. What makes VNFs flexible, is that one VNF does not have to be wholly deployed on a single Virtual Machine (VM), but it can be spread over multiple VMs where different VMs host separate components [11].

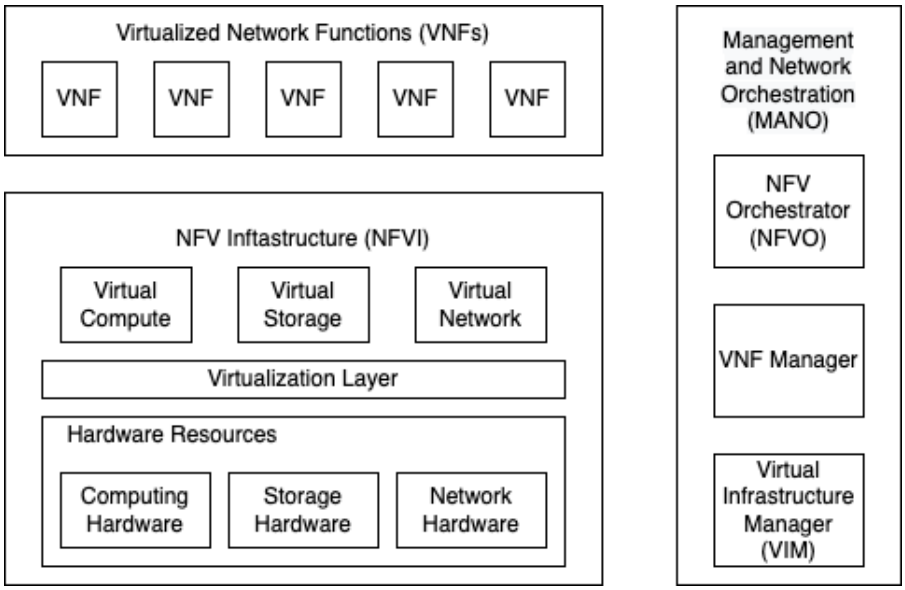

Figure 2.1: A diagram showing the NFV Architecture [1]

2.1.2 Network Functions Virtualization Infrastructure (NFVI)

Network Functions Virtualization Infrastructure (NFVI) provides the virtualization layer and physical infrastructure components - the storage, computation, and networking on a platform to support software needed to run network applications [10]. Some examples include the hypervisor, which creates and runs virtual machines (VMs) and allocates Central Processing Unit (CPU) memory storage to new and existing VMs, as well as Docker containers that do not use hypervisors for virtualization.

There are two types of hypervisors: Type 1, also called ”bare metal”, runs on host hardware directly to manage the VMs, while Type 2, also called ”hosted”, is a hypervisor that runs as a software layer on conventional operating systems (OS) [12]. Another example of an infrastructure component is the Kernel-Based Virtual Machine (KVM), which turns Linux into a hypervisor by getting some OS level components from Linux [13].

The NFVI is comprised of all of the components that build up the environment where VNFs are deployed, which includes hardware, software and network. The virtualization layer and the hardware resources are viewed as one, which allows the NFVI to correctly provide the desired virtualized resources; in addition, the

network that connects these locations is also considered a part of the NFVI [11].

2.1.3 The Management Automation and Network Orchestration (MANO)

The Management Automation and Network Orchestration (MANO) component provides the framework for managing the NFV infrastructure and provisioning new VNFs [10]. It consists of three main functional areas: the NFV Orchestrator, the VNF Manager, and the Virtual Infrastructural Manager (VIM) [9].

The NFV Orchestrator is the component that is responsible for handling the VNF onboarding and management of both lifecycle and global resources. It is the main component of the MANO. It also validates and authorizes resource requests coming from the NFVI [9]. This component does not function through direct communication with VNFs, but rather through the VIM and the VNF Manager [10]. The NFV Orchestrator plays a big role in the on-boarding, terminating, scaling, and instantiating network services and network service lifecycle, as well as interacting with support systems through which users can execute service operations [14].

The VNF manager component is responsible for the management of the VNF lifecycle, which encompasses the updating, querying, and termination of VNFs [10]. Depending on the architecture, the VNF manager can serve multiple VNFs at once, or just a single VNF. This component is also responsible for event reporting, configuration of Element/Network Management Systems, and provides the NFVI with a coordination and adaptation role [9].

Finally, the VIM manages and controls resources of the NFVI, including the compute, network, and storage resources [10]. It provides the functionalities that allow the control and management of the interactions between VNFs and computing. This means that the VIM also allocates virtualization enablers and manages the allocation of infrastructure resources. In addition, this component also performs planning and analysis of issues like performance issues or infrastructure faults, as well as monitoring and optimization [9].

2.2 Service Composition

Using the architecture mentioned previously, NFV is used to provide network services. What NFV does is virtualizing each network function instead of using dedicated hardware for deployment of a network service. This is done through running different VNFs in a specific order. An example of this would be sending a packet from one virtual machine to another, through a firewall and a packet inspector [15]. If the order in which the VNF instances are run is incorrect, the packet will not be sent correctly. This is called the service chain, as shown in Figure 2.2.

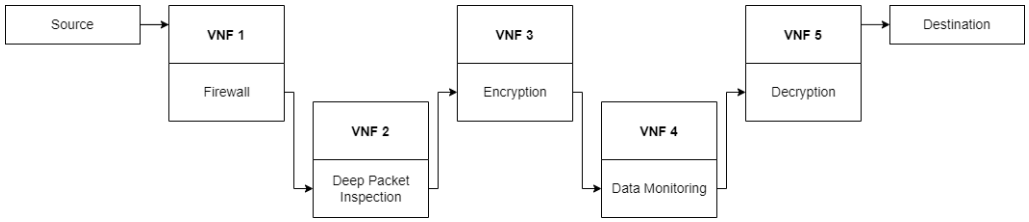

Figure 2.2: NFV Service Chain Diagram [2]

As stated in the introduction, using NFV makes providing network services both faster and cheaper; to accomplish better service, there are several differences in the way of providing network services between current practice and NFV. Some of these differences include dynamic scaling, separating software from hardware, and flexibility of network function deployment [11].

Dynamic scaling indicates that it is possible to scale the VNF performance more in accordance of, for example, the network traffic through decoupling of network functions into separate software components. Separating software from hardware is self-explanatory; decoupling of these allows for independent evolution of both and separate maintenance for software and hardware, as well as separate timelines of development. Finally, flexibility of network function deployment implies that separating software and hardware aids in reassignment of the infrastructure resources, which allows for the performance of different functions at differing times. This allows for faster deployment of new network services over the same physical platform through flexible setup of connections in the network [11].

When deploying a network service on VNFs, these VNFs do not all have to be

in the same location. Each service also does not have to be on only one VNF. It is possible to combine several VNFs from several online locations into one part of the service chain composition [15]. Because of this, NFV is incredibly flexible in providing network services, as not every component needs to be present in the same physical location.

2.3 Simulation, Emulation, and Experimentation

There are several ways of testing systems and software. This section will discuss the differences between the three methods, simulation, emulation, and experimentation.

An experimentation is done when a device is running locally, but in a lab setting. A good example of this is running a local network on routers and switches located in a lab, which creates an environment just like reality but on a smaller scale. This is useful for learning how reality works in a controlled environment.

An emulation is very similar to an experimentation in the way that it functions like reality, however everything that runs locally in the experimentation now runs virtually, on a computer [16]. Instead of physical equipment, a virtual version runs on the user's computer with all the functionalities the physical equipment would have. Emulations can therefore be used as an option to create a controlled environment with all the real functionalities away from a lab.

Compared to the emulation, a simulation is a more abstract way of testing. It is a model that represents a specific network or platform, which does not behave exactly like reality but rather focuses on one specific aspect of what they are representing. The degree of accuracy of the simulation depends on how comprehensive the simulator being used is; some simulators simulate an entire system, while other simulators encompass only specific parts [17]. For example, a simulator may simulate an OS entirely, while others simulate only a specific software for that OS.

The difference between simulators and emulators is quite subtle. Simulators allow the user to set up a similar environment to an original device's system, but they do not attempt to simulate it entirely as the real device. This means that simulators may not be able to run all the features that are associated with that device. Because of this, many programs run slightly differently, or have to be changed to be able to run on a simulator. However, an emulator duplicates every aspect of

the original device's behavior, including the hardware. This allows for the exact same features to run on the emulator, with no modifications. Because simulators do not attempt to simulate the device's hardware, they tend to run faster than emulators, but they are often less accurate. Emulators are more accurate, but this accuracy is in exchange for much slower run time, sometimes reaching an order of magnitude slower than a program would take on the original device. [18]

Chapter 3

State of the Art of the NFV Emulation

This chapter will cover any work related to the topic of the thesis, as well as cover different NFV emulators. In the end, there will be a comparison of the NFV emulators' different functionalities, architecture, any limitations, and what technologies they use.

3.1 Related Work

Although there are many papers and articles that cover NFV emulators, there are not many that compare them. This means that the original objective of this thesis was unique and new, however due to time constraints and other issues the objectives changed. This section will discuss comparisons of different aspects or components of NFV emulators.

There are several papers that compare certain aspects of NFV emulators, like "Comparing virtualization solutions for NFV deployment: A network management perspective" from 2016 that compares three NFV emulators in terms of virtualization solutions. Here, the authors selected three NFV emulators or OSs with emulating possibilities based on three aspects: code availability, being under development, and being in accordance with ETSI's NFV Virtualization Requirements document. They compared the virtualization solutions in terms of objective performance metrics like boot time, response time, and memory consumption among others. The authors also mention that NFV is an innovative

and fairly new concept that is used for creating new solutions and continuously investigated in academia. They came to the conclusion that of the three tested NFV emulators, two performed very well in terms of the selected performance metrics [19].

Another paper published in 2022 compares MANO frameworks, specifically the open-source platforms Open Source MANO (OSM), Open Network Automatic Platform (ONAP), Cloudify, Openstack Tacker, and OpenBaton and their functional characteristics. The authors selected nine aspects to compare the different MANO frameworks, including architecture, MANO Block, Standard Development Organization (SDO), NFVO functions, and more. The paper aimed to provide an overview of five open-source NFV-MANO frameworks that are widely used, identify the challenges and further guide the research and development in the field of Orchestration, and to characterize the frameworks through layered taxonomy based on their functions. The paper highlighted some research challenges like scalability, resource management, security and privacy, and interoperability, that provide research opportunities within this field. The authors concluded that different MANO frameworks can be distinguished from one another through different properties like licenses and governance policy, and that there are several possibilities of implementation through the architecture and language support of each MANO framework [20].

3.2 Emulator Evaluations in Literature

Many papers and articles that discuss NFV emulators discuss one specific emulator in detail, rather than comparing multiple emulators. These papers often use case studies or experimental evaluations of the emulators to describe their functionalities and evaluate their overall effectiveness.

3.2.1 NIEP

A paper from 2018 discusses the open-source emulator NFV Infrastructure Emulation Platform (NIEP), including its architecture and module interactions. NIEP is developed by a research team consisting of 11 researchers from Brazil and Belgium. Part of its architecture is actually based on pre-existing tools that are integrated into a full emulator. These tools are Click-on-OSv as a VNF, KVM hy-

pervisor as VM management, and Mininet for network emulation. These are tied together by an orchestration module [21].

Click-on-OSv is an OS that is OSv-based and built specifically for NFV experimentation. It is a complete virtual machine that uses the Representational State Transfer (REST) interface for monitoring underlying operations like lifecycle management and different metrics. The KVM hypervisor is a virtual VM manager that supports executing numerous VMs that run different types of OSs by implementing full virtualization. The NIEP-Orchestrator uses the Virsh tool for management of the VMs created through KVM through Command Line Interface (CLI) system calls. Mininet is a network emulator that uses lightweight process-level virtualization, which allows designing large-scale network environments through the emulation of guest VMs as isolated processes. This reserves memory, network, CPU, and inherits the functions and programs of the host. All network topology definitions can be done using a JavaScript Object Notation (JSON) file, which is then received in the NIEP-Orchestrator that performs the instantiation process [22].

The NIEP-Orchestrator consists of four elements that each provide methods to control the NIEP environment through specific action executions. These elements are the VNF Repository, the Virtualized Elements Manager (VEM), the Topology Manager, and the Interpreter. These are responsible for tasks like storing virtualized network functions, creating the Mininet network topology and initializing it, validating the definition of the JSON topology, controlling VNF execution, handling user requests, and more [22].

In the aforementioned paper, a case study is set up to conduct an experimental evaluation of NIEP. The scenario of the case study consists of two hypothetical locations, the Customer Premises and the Internet Service Provider, with a Mininet host connected to a VNF in each. In the Customer Premises, the VNF had limited resources, while the Internet Service Provider had considerably more resources in comparison, as well as running a firewall and connecting to an SDN controller and a host that acts as the application server. Four configurations with increasing numbers of VNFs were used to investigate how this would impact NIEP's performance. After performing 30 executions of each configuration, the team writing the article reached a confidence interval of 99% on all of the experiments. It was found that NIEP provides possibilities for easily changing the defined topology, as many of the configuration parameters had fine-grained controls. This way, it

is possible to use the emulator for testing many diverse NFV scenarios through changing network paths, changing link properties, or adding, re-configuring, or removing hosts [22].

The results of the experimental evaluation from the aforementioned paper showed that NIEP's VNFs all initialize in parallel, which allows a fairly quick boot time even with multiple VNFs and firewall configurations. The emulator's performance was also tested under heavy load, where a bottleneck was simulated by testing the link between the Customer Premises and the Internet Service Provider and analyzing the throughput. It was found that increasing the amount of traffic to the server had no effect on the throughput in all experimental setups, which meant the results were positive. The conclusions of the article state that NIEP supports the emulation of "heterogeneous infrastructures [...] composed of devices with different characteristics," based on their results and the architecture of the emulator [22]. The authors also found that expanding the setup by adding more VNFs had no effect on metrics like boot time and throughput, where the results showed a linear increase in both depending on the number of VNFs. In addition, it was found that NIEP's VMs are isolated from the kernel, offering better security when testing third-party VNFs and allowing for emulation of more precise security scenarios [21].

3.2.2 ClickOS

ClickOS is a "high-performance, virtualized software middlebox platform" that is created specifically for NFV emulation [23]. The creators of this middlebox platform realized that many middleboxes were hardware-based, which made them difficult to manage with functionalities that were almost impossible to change, as well as costly. Because of this, as well as the rise of NFV emulation, ClickOS was created to fill the void of software-based middlebox platforms [23]. It was one of the emulators described in the aforementioned paper comparing emulators in terms of virtualization solutions [19].

"ClickOS and the Art of Network Function Virtualization" from 2014 discusses the problem statement mentioned above, as well as the design and architecture of ClickOS. The paper also includes an evaluation of the platform as well as describes some middlebox implementations. ClickOS is Xen-based and is optimized for running hundreds of middleboxes that allow for processing speeds of millions

of packets per second. The authors conducted experiments that showed the processing speed of a low-end server is around 30 Gb/s using ClickOS [24].

The design of ClickOS relies on hypervisor virtualization, specifically para-virtualization, which allows to achieve flexibility and isolation between the hardware and the middlebox software. Para-virtualization is preferred due to full virtualization possibly increasing delay and ”hurting” the throughput. Because the creators decided to use para-virtualization, they based ClickOS on Xen, which supports para-virtualized VMs and therefore allows to build a platform with low delay ahd high throughput. Xen also comes with its own OS called MiniOS, which supports building efficient virtualized middleboxes. Because of this, all ClickOS VMs are based in MiniOS. ClickOS is programmed in C, which is a flexible language despite its high cost of development and debugging [24].

”Enabling Fast, Dynamic Network Processing with ClickOS” is a paper written in 2013 that focuses on the emulator in terms of SDN. Here, ClickOS is presented as being a ”tiny, Xen-based virtual machine that can instantiate middlebox processing in milliseconds while achieving high performance” [25]. The architecture is described, where Xen is used to create a system comprised of several ClickOS VMs. The control plane of ClickOS that handles all operations is separated into three parts: the C-based CLI that provides an interface for users, the MiniOS control thread that is created when a guest domain of ClickOS boots, and a new element of Click that is called ClickOSControl that allows for communication between the other elements of the control plane and the Xen store. This paper also evaluates various aspects of the emulator, where all tests were performed on x86 commodity servers connected by direct cabling, one server acting as a packet generator and sink, while the other runs Xen 4.1.2 and the ClickOS VMs. The aspects tested were middlebox instantiation and networking performance, where it was found that the start-up time was in the range of several seconds, but with all optimizations the time dropped as low as 7-21 milliseconds, depending on how many VMs were running. In addition, the line rate for packets was optimized to fill up the 10 Gb pipe [25].

ClickOS does not seem to be a full emulator, but rather focused on creating a software solution for packet forwarding and middleboxes. This makes it more of a tool used for NFV emulation, rather than being an emulator in itself.

3.2.3 MeDICINE

The Multi Datacenter servIce ChaIN Emulator (MeDICINE), also known as Vim-Emu, is an NFV emulation platform that has been created together with Open Source MANO (OSM). These are both created in projects related to ETSI, called the SONATA Project and the 5GTANGO project [26]. MeDICINE and OSM are both open source and continuously improved. MeDICINE uses Containernet as the core of its platform, and it offers OpenStack-like APIs that will allow to integrate with OSM or other MANO solutions. Other than that, it is highly customizable and offers plugin interfaces like topology generators, container resource imitation models, and others for most of its components, meaning the emulator is highly flexible [27]. The emulator also focuses on a multi point-of-presence (PoP) approach, which means that the topologies in this emulator define available PoPs that start compute resources at emulation time instead of describing single hosts that are connected to the emulated network.

This emulator is, as mentioned, heavily reliant on several APIs. Most notably, it uses an extended version of Mininet's Python-based topology API that gives methods to define and create PoPs. Because it is a Python-based API, it gives the benefit of allowing the usage of scripts or algorithms to generate topologies within the emulator [27]. Another API is used to initiate and stop compute resources. MeDICINE also uses the concept of "flexible cloud API endpoints" that are interfaces that link to one or multiple PoPs that manage compute resources. There is also an API created for the support of developers, which focuses on the service chain, that simplifies the SDN protocols to calling one method to allow for chaining emulated containers [27].

MeDICINE builds on the Containernet tool, allowing the use of Docker containers in the emulation. Containernet also allows for the addition and removal of containers while the network emulation is running, which Mininet, the base of Containernet, does not support [28]. Because of this, Containernet can be used as a cloud infrastructure, making MeDICINE even more flexible [26].

Prior to the creation of the emulator, there was a lack of prototyping tools that would allow testing of multi-PoP scenarios of complete network services. According to the creators of the emulator and authors of the 2016 article "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments", the main goal of the emulator was to create a "novel prototyping platform

for network services" that would allow for testing of these scenarios [27]. In the article, the authors outlined several examples for testing NFV solutions, including single and several VNFs, MANO systems, and more, that existing tools may not have supported. The MeDICINE platform provides a new support tool that could grant functionalities that existing support tools for network development lacked. Because of this, the creators believe that the emulator is a crucial step in the development of a fully integrated support toolchain for NFV [27].

3.2.4 ComNetsEmu

ComNetsEmu is an NFV-SDN emulator created specifically for educational purposes. It is a tool used in the textbook "Computing in Communication Netowrks: From Theory to Practice" written by F. Fitzek and F. Granelli and published in 2020 [29]. The emulator's design focuses on it being possible to run any examples on something as simple as a single laptop, meaning it could be a powerful tool for students to learn about NFV and SDN emulation. Because of this, the performance of the emulation is outside the main focus of the project [29].

ComNetsEmu builds on the Mininet project, extending it by concepts that are also in the Containernet project. This is done to allow for NFV/SDN emulation of network applications that are more versatile than it is possible to create with Mininet on its own [30]. ComNetsEmu uses complete host isolation and sibling containers to allow for the emulation of network systems with computing. To create sibling containers, the emulator uses Docker hosts that are deployed within Docker hosts, the so-called Docker-in-Docker. This allows for lightweight emulation of nested virtualization, as Docker-in-Docker mimics a physical host that deploys Docker containers. This is all done in Mininet topologies [30].

For simplicity, the emulator was developed first with Python 3.6, and Python 3.8 when that became available, with examples and applications written in high-level script language. Because of this, the performance of these programs is not optimized. However, it is possible to contact the creators of the emulator for highly optimized implementations. The emulator's public repository contains a collection of examples with sample code, as well as documentation that allows for easy reproduction. The examples can also be extended easily thanks to the provided documentation [31].

This emulator is presented in an article published in 2021 titled "An Open

Source Testbed for Virtualized Communication Networks". Here, the creators of the emulator and authors of the textbook discuss the architecture of the emulator as well as describe uses of the emulator for both researchers and educators. Here, the authors go into detail about the sibling containers and Docker-in-Docker topologies and emphasize how ComNetsEmu is designed for allowing setup on more resource-limited hardware, such as student laptops. The authors also describe that the emulator provides a collection of built-in examples that show the usage of its main features, as well as APIs, to aid students in learning about NFV [30].

3.2.5 Commercial Emulators

In addition to the emulators that are described above, there is a number of other emulators, both commercial and open-source, available on the market. Several of these were not taken into consideration for this thesis due to cost, like NE-ONE, VMware, Equinix, and others. Many of these commercial emulators consisted of several products that together make up the complete emulator, making it impractical from an academic point of view; when investigating emulators that could be used by future students to study NFV, it is best to select an emulator that does not require the purchase and installation of several products due to both the accumulation of cost and the usage of PC memory.

An example of this is Juniper's Contrail product line [32]. Here, the full emulator is split into Contrail Networking [33], Contrail Service Orchestration [34], and Paragon Insights [35], not including a separate Firewall product. The company promises ultra-fast processing speeds, flexible topologies, an intuitive console, and automated analytics, security, and networking. The emulator features an elastic VPN, API services, visualizations of the network traffic flow, and a number of other features [32]. The emulator does not seem to have been mentioned in any articles or case studies, which makes it difficult to judge without a deeper investigation.

Another example of commercial emulators is the aforementioned NE-ONE emulator range. These products were created by Calnex, with features such as the RESTful API, a scenario builder, and more [36]. According to the website, the purpose of NE-ONE is to provide an accurate test network that is both controllable and repeatable, for developing and accelerating application readiness in

network emulation. The emulator range has been utilized in several case studies for different companies [36]. There are several models of NE-ONE, with each model having additional features and increased bandwidth. However, this emulator comes with its own hardware in two different versions: the Enterprise version and the Professional version, with availability as a virtual appliance as well [37]. The necessity of separate hardware for the emulator makes it impractical and impossible for students to acquire on their own for academic use.

3.3 Emulator Comparison

Four emulators were described in detail in the last section, these being NIEP, MeDICINE, ClickOS, and ComNetsEmu. All four of these were considered for this thesis. This section will compare their specifications and features in as much detail as possible without testing each emulator.

As previously mentioned, after closer inspection it seems that ClickOS is not a full emulator in itself, but rather a tool that is used for NFV emulation. This means that it could be used as a component in a full emulator. Because of this, ClickOS will not be taken into consideration during the comparison of the emulators. Instead, the focus will be on the three full NFV emulators.

There are several factors that influenced the choice of the emulator for this thesis, including the features, accessibility, ease of installation, and more. The most deciding factor ended up being the ease of installation, seeing as two of the emulators had some issues when installation was attempted.

First, the emulators were all developed using Python. However, in comparison to ComNetsEmu and MeDICINE, NIEP was developed using the now deprecated Python 2.7. ComNetsEmu was initially developed using Python 3.6, but as newer versions of Python rolled out, the development moved to using Python 3.8. MeDICINE also uses Python 3, but there is no information about which version specifically. As Python libraries evolve, some functionalities can change, meaning that it is important to keep programs up to date.

All three emulators utilise elements of Mininet in the network emulation, with both ComNetsEmu and MeDICINE expanding into functionalities of Containernet in addition to Mininet; NIEP is limited to using just Mininet functionalities. Mininet uses lightweight, process-level virtualization that allows designing network environments at a large scale, while Containernet is a fork of the Mininet

emulation that allows for more flexibility through the use of Docker containers as hosts, as opposed to isolated processes like in classic Mininet. Thanks to the added features, Containernet is used in research in cloud and fog computing, NFV, and more, meaning that an emulator utilising this will be more flexible and can create more realistic emulations compared to an emulator only utilising Mininet [28].

The topology definition is different for each emulator. The MeDICINE platform uses a topology API that contrasts with classical Mininet topologies by describing network hosts as available PoPs [27]. ComNetsEmu, on the other hand, uses the classical Mininet topologies, but employs Docker containers as hosts [30]. Finally, NIEP also uses Mininet topologies, but these are initialized through a JSON file to simplify the process of deploying the infrastructure [22].

The emulators also differ in the implementation of orchestration. NIEP uses an orchestration module that includes the VNF repository, the topology manager, the interpreter, and the VEM. It also receives the JSON file with the topology definition to execute the instantiation process [22]. Several MANO systems can be used coupled with MeDICINE, but the most obvious choice is OSM, as it was developed as part of the same project. This makes the emulator more flexible, and yet more complex as different MANO systems can have different functionalities [27]. ComNetsEmu, however, provides a special Manager class called APPContainer to orchestrate the internal Docker containers [30].

The installation of all three emulators can be done through the Ubuntu terminal. However, ComNetsEmu is only possible to install through Ubuntu version 20.04 or later [31]. MeDICINE is recommended to install using Ubuntu 18.04 or later [38], while NIEP does not mention a limit to which version of Ubuntu must be used [21]. All three emulators also require the installation of extra software for it to be possible to install and use them. ComNetsEmu requires the installation of Vagrant and Virtualbox before installing the emulator [31]. NIEP requires the installation of hypervisors (Qemu and Libvirt) as well as Python 2.7 [21], and MeDICINE requires the installation of Containernet and Ansible, a bare-metal hypervisor [38]. All three emulators also require the cloning of the Github repositories to the local machine prior to installation.

Examples of use of the emulators can be very useful for getting accustomed to the functionalities available. All three emulators in this comparison provide examples; NIEP provides a list of examples for different topologies, including VMs,

VNFs, and more. These were updated in February 2022 [21]. MeDICINE also has a long list of examples, including full stack emulations of different complexities, different topologies, and more, dating back to March 2019 [38]. ComNetsEmu also provides examples, though the creators stress that these examples are not made with optimized performance in mind. Here, the examples are basic and can easily be reproduced and extended, and they encompass deploying Docker-in-Docker, deploying a simple network service, and more. The latest update to these happened in January 2022 [31].

A summary of the above comparison can be found in Table 3.1.

	NIEP	MeDICINE	ComNetsEmu
Language	Python 2.7	Python 3	Python 3.6 and 3.8
Based on	Mininet	Containernet	Containernet
Topology	Mininet initialized through JSON	Topology API	Mininet using Docker hosts
Orchestration	Built-in orchestration module	Compatible with several MANO systems but OSM recommended	APPContainer Manager class
Installation	Ubuntu, requires installation of hypervisors	Ubuntu (18.04 or later), requires installation of Containernet and Ansible	Ubuntu (20.04 or later), requires installation of Vagrant and Virtualbox
Examples	13+ different examples (Feb 2022)	8+ different examples (Mar 2019)	7 basic examples (Jan 2022)

Table 3.1: Comparison of the three emulators considered for this thesis.

Based on this comparison, one emulator was selected. The biggest factor that influenced the selection of the emulator was the ease of installation, where the installation of all three emulators was attempted. Only one of the emulators was possible to install, and that was MeDICINE. Because of this, MeDICINE will be used for the remainder of this thesis. The issues that happened with the other two emulators were either the inability to install components due to them being deprecated, or the inability to configure the components correctly. In addition to

these issues, MeDICINE also seems like it is a very flexible emulator with many good features that allows for the creation of an accurate and realistic network emulation. However, upon attempts of running some basic examples, MeDICINE malfunctioned, which is why it was ultimately not used to run any created tests.

Chapter 4

Investigating MeDICINE

This chapter will be focusing on the installation and running of MeDICINE, as well as evaluating the emulator more in-depth than in the previous chapter. In addition, technical problems that arose during the writing of this thesis will be discussed briefly.

4.1 MeDICINE Installation and Use

There are several steps that go into the installation of MeDICINE through Ubuntu. First, the installer for OSM needs to be downloaded by entering the command `wget https://osm-download.etsi.org/ftp/osm-11.0-eleven/install_osm.sh` which places the `install_osm.sh` file in the current repository [39]. This method installs MeDICINE in combination with OSM, which is the simplest method. The next step is to run the command `./install_osm.sh --vimemu` which guarantees that the emulator is installed together with OSM. These instructions are taken from the Github repository for Vim-Emu (Vim-Emu and MeDICINE are interchangable names for the emulator) [38].

The next step is to install the Ansible hypervisor. To do this, enter the command `sudo apt-get install ansible git aptitude` into the terminal and follow further instructions. When this is complete, it is necessary to clone the Containernet Github repository into the local repository on the machine that will be running the emulation. Once the repository is cloned, the right directory needs to be accessed to install Containernet. Once the installation process completes, the user must go up one directory and run another command before proceeding to the

installation of the emulator itself. The code block below illustrates the necessary commands for the installation of Containernet.

```
git clone https://github.com/containernet/containernet.git
cd ~/containernet/ansible
sudo ansible-playbook -i "localhost," -c local install.yml
cd ..
sudo make develop
```

Next is the installation of MeDICINE itself. This is done by cloning the Vim-Emu Github repository outside of the Containernet repository, entering the right directory, and running the installation command. Once this process is completed, move up one directory and run the setup command. The code block below illustrates the necessary commands for the installation of the emulator.

```
git clone https://osm.etsi.org/gerrit/osm/vim-emu.git
cd ~/vim-emu/ansible
sudo ansible-playbook -i "localhost," -c local install.yml
cd ..
sudo python3 setup.py develop
```

Once this is complete, if Docker is not installed on the user's machine already, return to the home directory and execute the command `sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin` to do this. This completes the installation of MeDICINE and other necessary software required to run the emulator.

To use the emulator, it is necessary to run a Docker daemon in one terminal and using at least two others for the emulator and its different components. To run a Docker daemon, enter the command `sudo dockerd` into the terminal. This will ensure that running any Docker hosts will be possible. Then, in all other terminals that will be used, enter the `vim-emu` repository to build and run a Docker container. The commands needed are shown in the code block below.

```
# build the container:
docker build -t vim-emu-img .
# run the (interactive) container:
docker run --name vim-emu -it --rm --privileged --pid='host'
    -v /var/run/docker.sock:/var/run/docker.sock vim-emu-img
    /bin/bash
```

An example of the use of the emulator without OSM, is done through running a previously created Python3 topology file in one terminal, while creating a few VNFs in a second terminal. Then, through the first terminal, it is possible to check the connectivity of the VNFs created in the second terminal. The more features, including the OSM, are included in the running of the emulator, the more advanced it becomes.

4.2 Evaluation of MeDICINE

MeDICINE is an emulator that was created ”in the framework of the SONATA project” [38] and is most often used together with OSM. There are instructions on how to install every component of the emulator, but as there are multiple pages with slightly differing instructions, it can be confusing to find the correct ones. Most notably, some instructions come from an already deprecated page [26] that later redirects to a page that is dedicated to OSM rather than MeDICINE [39]. The instructions above, as already mentioned, come from the official Github repository for the emulator [38]. Here, there are also instructions for the setup of the emulator as well as instructions on how to run a simple example. In the page dedicated to OSM, it is possible to find the minimum and recommended requirements for the machine on which it is to be installed [39]. These minimum requirements are 6 GB RAM, 40 GB of disk space, 2 CPUs, and internet access. It is not recommended to use Ubuntu for Windows, but rather a VM running Ubuntu, preferably version 20.04 or later.

There are, as mentioned in the previous chapter, more than eight different topology files under the `/examples` folder. These can be used in different scenarios, for example for a topology with a single data center versus a full-stack emulation with multiple MANO frameworks. When running the emulator, it is possible

to run the `--help, -h` positional argument in combination with the main commands to learn more about their functionalities. These main commands are:

```
vim-emu compute
vim-emu datacenter
vim-emu network
```

`vim-emu compute` is responsible for any compute instances and containers. It is run together with the `start, status, list, stop, xterm` positional arguments. The `start` argument initiates a compute instance, while `status` gives the status of an already created compute instance. `list` gives a list of all created compute instances, while `stop` terminates a compute instance. Finally, `xterm` is coupled with a list of VNF names and opens terminals for the VNFs listed. The `compute` command has a list of optional arguments, of which the most important ones are `--datacenter, -d` that indicates the name of the data center to which the command will be applied and `--name, -n` that gives the name of the compute instance, for example ”vnf2”. In addition, there are other positional arguments like `--image, -i` that indicates the name of the container image that should be used, or `--net` that gives the network properties of the compute instance, and more.

The `vim-emu datacenter` command, on the other hand, is responsible for giving information about the data centers that are running. These are initialized through the given topology file, and the positional arguments for this command are `list` and `status`. The `list` positional argument is used to list all running data centers, while the `status` positional argument is used in combination with the optional argument `--datacenter, -d` to give information about a specific data center.

Finally, the `vim-emu network` command is used to create networks within the emulation. The positional arguments here are `add` and `remove` and are quite self-explanatory. There are several optional arguments, the most important ones being `--datacenter, -d` that indicates which datacenter the network should be initiated for, `--source, -s` that indicates the name of the VNF that should be the source of the chain, and `--destination, -dst` that indicates the name of the VNF that will be the destination of the chain.

Using these three commands, it is possible to build a complete network service chain. The setup of the network service depends very much on which topology file

is used when it comes to the amount of data centers and MANO frameworks, but the user controls how many VNFs are created and which of these have terminals. Through the terminals in the VNFs, it is possible to configure each VNF to a different role, for example a router, a firewall, or a client computer.

4.3 Technical Problems

This short section will discuss the technical issues that did not allow the MeDICINE emulator to run, forcing a change to the objectives of this thesis. Several attempts were made to install the emulator, on two different computers and several VMs.

The first attempt, the installation was done through Ubuntu for Windows, which has its own issues when it comes to more complex programs and commands. Because of this, it was possible to install what seemed to be the emulator and run it, but nothing more than the initial setup file was possible to run. As it turns out, what was installed through Ubuntu for Windows was not the complete emulator, which became apparent when attempting installation via a VM running Ubuntu. However, the VM on the initial computer would crash before completing the installation. This is when attempts were made using a second computer. Again, installation was attempted on a VM running Ubuntu, ensuring the VM had all the necessary hardware requirements. However, here there was also an issue during installation, where the process downloaded files into the VM and later threw an error because the files were present. This means that there might be an issue with the installation process in itself.

Due to time constraints and an uncertain situation that arose during the writing process, it was not possible to investigate this issue further, which is what forced the change to the objectives of the thesis.

Chapter 5

Design of Emulator Tests

This chapter will focus on the design of tests for the MeDICINE emulator. As mentioned multiple times previously, due to technical issues and time constraints, the tests designed in this section were not possible to run in MeDICINE as the emulator malfunctioned upon trying to run some basic examples. Due to this, the tests designed will encompass many aspects and functionalities of NFV, with the intention of being run in MeDICINE. The chapter will also mention any potential for future works using the designed tests.

5.1 Design Process

The first step to running MeDICINE is to use a topology file that is used as a base for the emulation. Since there are already eight pre-existing topologies, it was seen as beneficial to utilize two of these in the design of the tests. The first, simpler topology utilizes a single data center, and the code for this can be found in Appendix A.1. The second topology is more complex and utilizes two data centers, as well as the Tango Lightweight Life Cycle Manager (LLCM), and can be found in Appendix A.2.

The method used to design these tests is to start very basic, and build up complexity with each test. Some of the architectures designed are inspired by exercises from Cisco Packet Tracer, used in the Communication Technology courses at the University of Stavanger.

5.1.1 Tests With Single Data Center

Keeping in mind that potential students may not be familiar with concepts of NFV and network emulation in general, it is crucial to start with less complex tests to not make the learning curve too steep from the beginning. Because of this, the first test will be using the simpler topology and will utilize only two VNFs. A diagram of the architecture of this test can be seen in Figure 5.1. In this test, both of the VNFs are intended to be client computers that are connected to each other via the data center. These client computers can communicate with each other using for example the ping command.

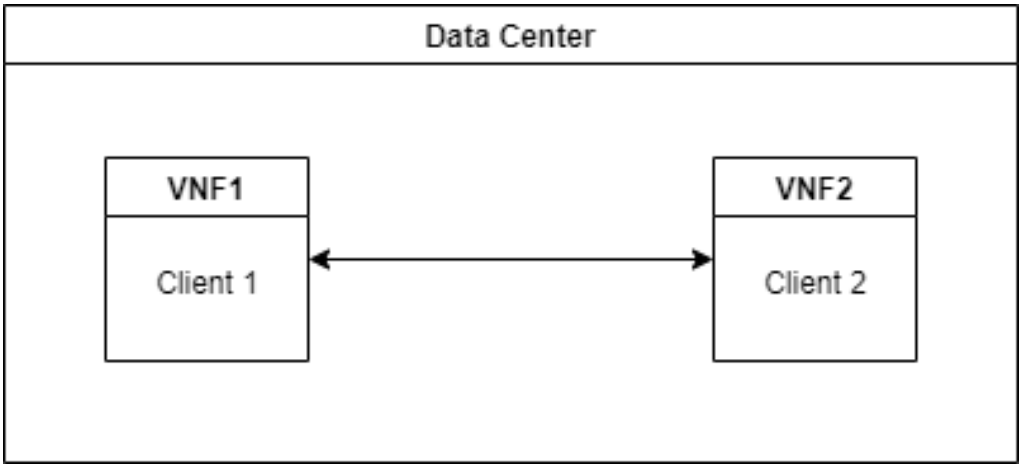

Figure 5.1: Diagram of the first test architecture.

Staying with the single data center topology, it is possible to build a more complex architecture for the second test. The first test utilized only two VNFs as client computers; this test will contain five client computers that are connected by two switches. That makes seven VNFs in total. A diagram of the architecture can be seen in Figure 5.2. Here, the key is to configure each VNF to each component of the architecture to create two separate networks for each group of client computers and allow for communication between all of them.

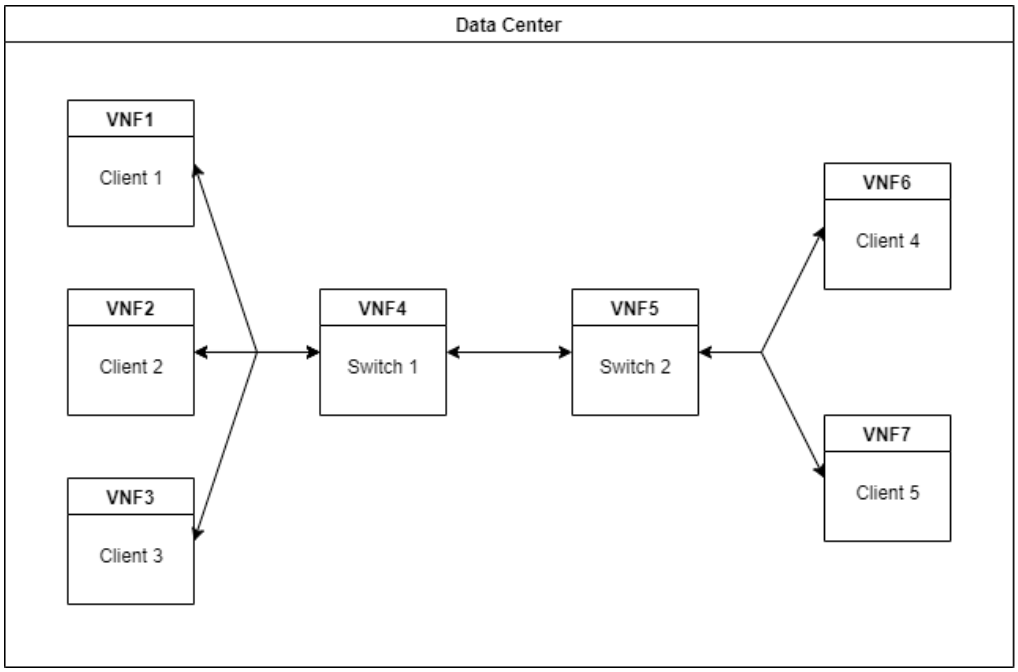

Figure 5.2: Diagram of the second test architecture.

To increase the complexity even further, it is possible to increase the number of switches and client computers, still within the single data center topology. Here, the principle will be similar to that of the previous test, where the key is to configure a network for each switch and ensure communication between all client computers; however, it involves a far greater amount of VNFs, with three switches and nine client computers, which makes twelve VNFs in total. For this test, the aim is to set up three networks and establish communication between them, so that all client computers can ping each other. This architecture can be seen in Figure 5.3.

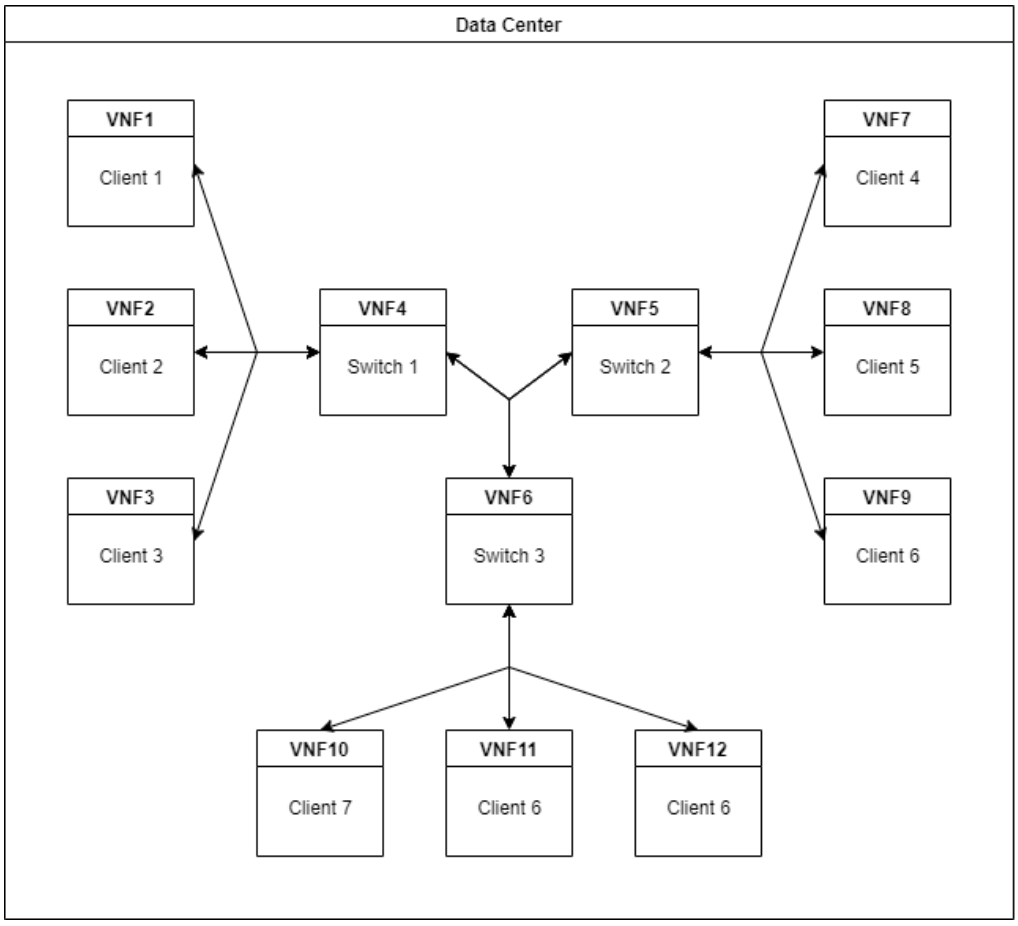

Figure 5.3: Diagram of the third test architecture.

5.1.2 Tests With Two Data Centers

The previous test was more complex than the first two, but it was still built in a topology with a single data center. To increase the complexity and make the next test closer to a real network service, the next tests will utilize the topology file with two data centers. This will mimic two separate Local Area Networks (LANs) that need to communicate with each other.

The first of these tests will reduce the amount of VNFs to compensate for the introduction of the complexity of the second data center. Here, there will be one router and two client computers in each data center, which adds up to a total of six VNFs. This architecture can be seen in Figure 5.4. In this test, it is crucial to

not only ensure communication between the client computers in the same data center, but most importantly to ensure the communication between the two data centers, and subsequently the communication between client computers across data centers.

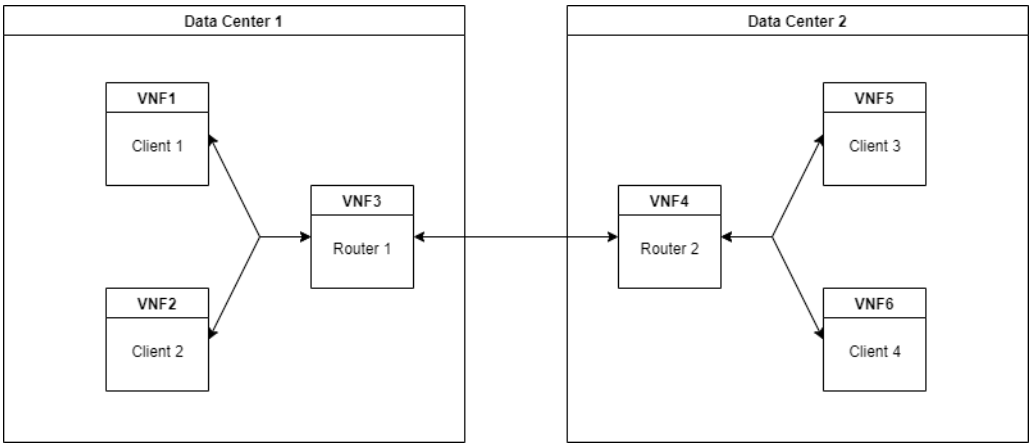

Figure 5.4: Diagram of the fourth test architecture.

To again increase the complexity, the next test will increase both the amount of VNFs, and what type of component they represent. In the first data center, there will be one router, two switches, and four client computers. However, in the second data center, there will be one router, one firewall, one switch, and three client computers. The diagram of this architecture is illustrated in Figure 5.5. Thirteen VNFs are used in this test, with seven VNFs in the first data center and six VNFs in the second data center. Before communication between data centers can be established, it is crucial to ensure communication within each data center, and the correct configuration of the firewall in the second data center.

The sixth and final test is the most complex of the designed tests. It uses the topology with two data centers, with one router, one firewall, two switches, and five client computers in each data center, which adds up to a total of 18 VNFs, nine in each data center. That means that the architecture is symmetrical. This can be seen in the diagram in figure 5.6. Again, it is important to ensure communication within a data center before attempting to establish a connection between the two data centers. This includes the communication between client computers that are connected to the same switch as well as to a different switch, and the correct configuration of the firewall in each data center.

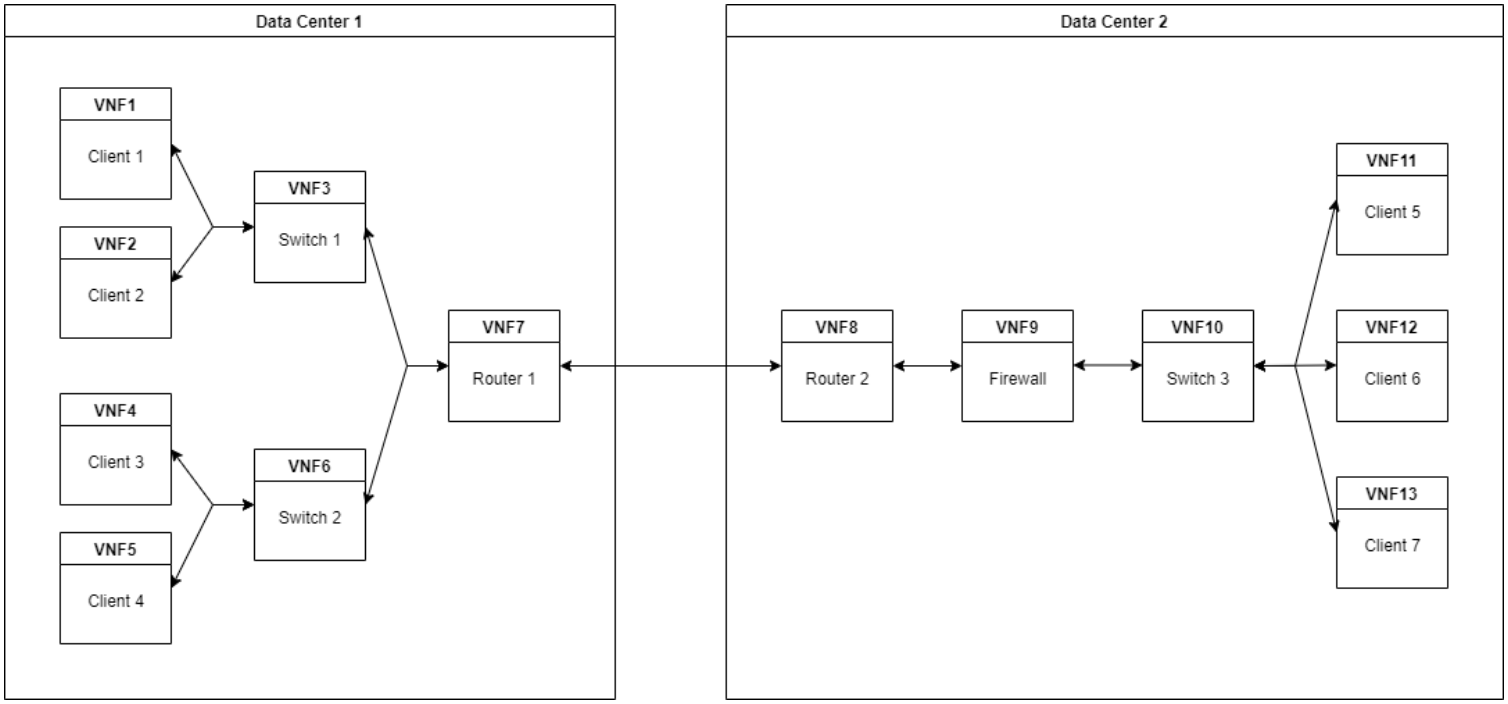

Figure 5.5: Diagram of the fifth test architecture.

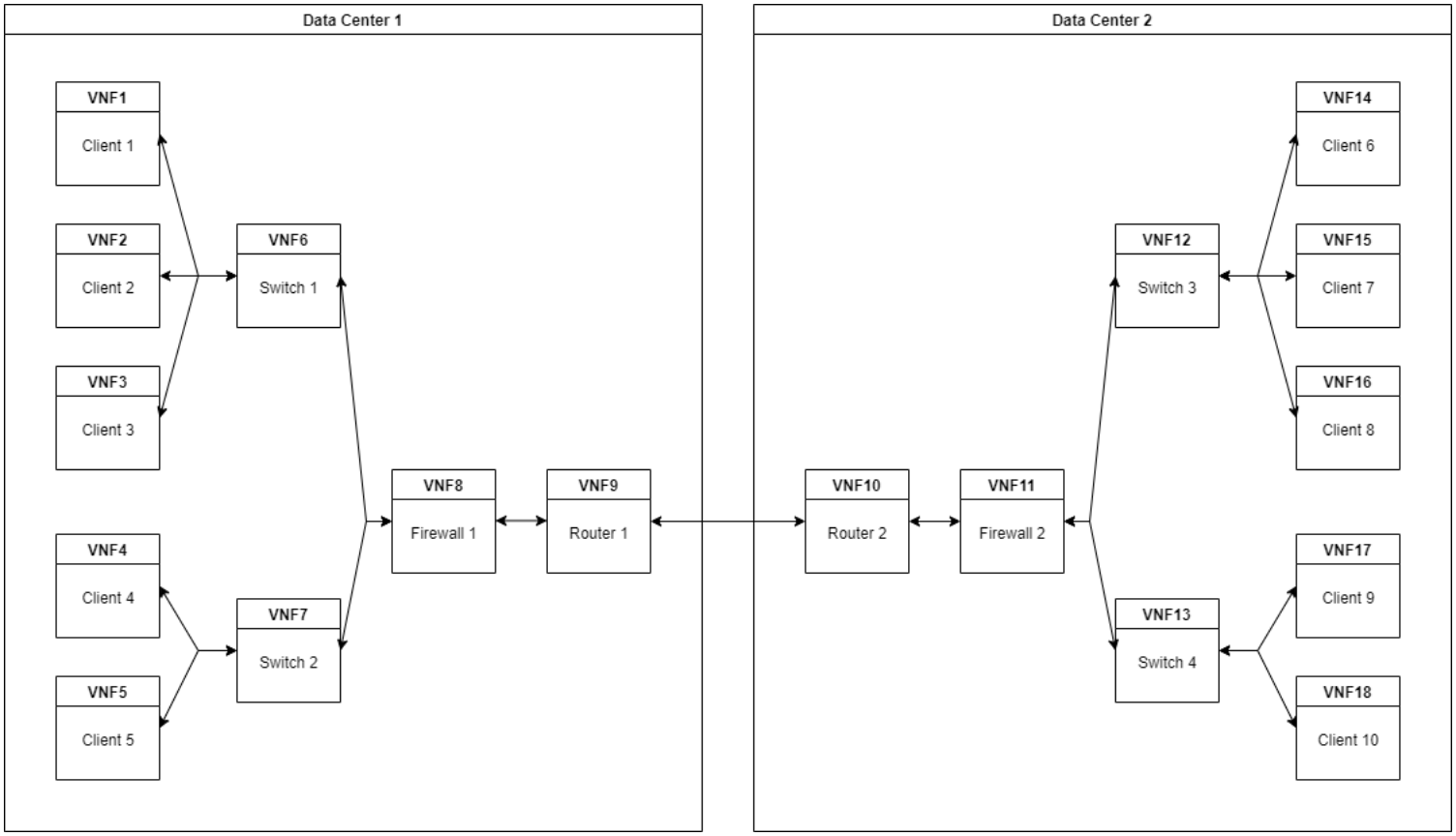

Figure 5.6: Diagram of the sixth test architecture.

These tests, all growing in complexity, illustrate different architectures of network service chains. The number of VNFs increases with each test, stepping down only with the introduction of a second data center. The tests, done in progression from 1-6 should be possible to complete while learning how to use the emulator itself, meaning the learning curve is not too steep. The tests are slightly similar to exercises from the aforementioned Communication Technology courses, with possibilities for IP configuration, sending packets, and more.

5.2 Potential for Future Works

The tests designed in the previous section are meant to be an introduction to NFV and network emulation for potential students. Because these were not possible to test at the time of writing this thesis due to technical issues, it opens up the possibility for future works.

First, the tests could be built in the MeDICINE emulator, with all of the different infrastructure like routers or firewalls. Here, different functionalities could be tested through pinging or attempting to send packets with data, as well as configuration of IP addresses in tests where switches are involved. Parameters like the time it takes to send a packet in the different architectures could be investigated.

Each test could be replicated in not only the MeDICINE platform, but also in other emulators like ComNetsEmu or NIEP. This would require creating topology files that are compatible with these other emulators, but that have the same functions as the original topology files. Then, the tests could be run and compared. The comparison could be for example the differences in building the architectures in each emulator and how many steps it took, or how smoothly the emulator runs the test and how long the different functionalities take.

Chapter 6

Conclusions

NFV is still a novel concept that is gaining popularity, both in academia and in the telecommunication industry. Because of this, it is important to investigate and create a path for future students to learn more about the topic. The objectives of this thesis were just that.

Because of unforeseen issues and time constraints, the objectives changed several times during the writing process. Starting off as the design and running of tests in two different emulators and the comparison of these, the objectives later changed to only using one emulator to run designed tests; however, when the selected emulator did not function correctly, a last-minute change to the objectives had to be made yet again to design tests that may be used by future students that will be able to correctly install and run the emulator. In addition, the unchanged objective was to research the state-of-the-art of NFV as a whole, as well as compare several NFV emulators with the intention of selecting one or two for running of the designed tests.

After thorough into the architecture and service composition of NFV, several open-source NFV emulators were investigated in depth to determine which emulator would be the most suitable for this thesis. After attempting to install three different emulators, it appeared that only MeDICINE was possible to install and run the setup. Because of this, the MeDICINE platform was selected. However, upon closer inspection, it became apparent that the emulator was not running correctly and it was impossible to run any kinds of tests.

Since it was not possible to run any tests, a more thorough investigation into the functions used in the emulator was done, as this was possible to run in the

incomplete, malfunctioning emulator. After this, the tests were designed based on two pre-existing topology files. Six tests were designed in total, with different difficulty levels. These tests were designed with a learning curve for both NFV concepts and the emulator in mind.

These six tests that were designed for use in MeDICINE open up possibilities for future works. Either to run the tests and check metrics and parameters like execution time, or adapting the topology files to be able to run in a different emulator to be able to compare how two different emulators may run the same network service architecture. The tests can be used for IP configuration, sending and tracing packets, and more, meaning they are versatile and give varying prospects for future students to use them for their own learning and research.

Appendix A

Topology Files Used For Tests

A.1 Topology With Single Data Center

Note: This code was written by Manuel Peuster and is located on the Vim-Emu (MeDICINE) Github repository at `/examples/default_single_dc_topology.py` [38].

```
# Copyright (c) 2015 SONATA-NFV and Paderborn University
# ALL RIGHTS RESERVED.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Neither the name of the SONATA-NFV, Paderborn University
# nor the names of its contributors may be used to endorse or promote
# products derived from this software without specific prior written
```

```
# permission.
#
# This work has been performed in the framework of the SONATA project,
# funded by the European Commission under Grant number 671517 through
# the Horizon 2020 and 5G-PPP programmes. The authors would like to
# acknowledge the contributions of their colleagues of the SONATA
# partner consortium (www.sonata-nfv.eu).
import logging
from mininet.log import setLogLevel
from emuvim.dcemulator.net import DCNetwork
from emuvim.api.rest.rest_api_endpoint import RestApiEndpoint
from emuvim.api.openstack.openstack_api_endpoint import OpenstackApiEndpoint

logging.basicConfig(level=logging.INFO)
setLogLevel('info')  # set Mininet loglevel
logging.getLogger('werkzeug').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.base').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.compute').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.keystone').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.nova').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.neutron').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.heat').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.heat.parser').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.glance').setLevel(logging.DEBUG)
logging.getLogger('api.openstack.helper').setLevel(logging.DEBUG)


def create_topology():
    net = DCNetwork(monitor=False, enable_learning=True)

    dc1 = net.addDatacenter("dc1")
    # add OpenStack-like APIs to the emulated DC
    api1 = OpenstackApiEndpoint("0.0.0.0", 6001)
    api1.connect_datacenter(dc1)
    api1.start()
```

```
    api1.connect_dc_network(net)
    # add the command line interface endpoint to the emulated DC (REST API)
    rapi1 = RestApiEndpoint("0.0.0.0", 5001)
    rapi1.connectDCNetwork(net)
    rapi1.connectDatacenter(dc1)
    rapi1.start()

    net.start()
    net.CLI()
    # when the user types exit in the CLI, we stop the emulator
    net.stop()


def main():
    create_topology()


if __name__ == '__main__':
    main()
```

A.2 Topology With Two Data Centers

Note: This code was written by Manuel Peuster and is located on the Vim-Emu (MeDICINE) Github repository at `/examples/tango_default_cli_topology_2_pop.py` [38].

```
# Copyright (c) 2018 SONATA-NFV, 5GTANGO and Paderborn University
# ALL RIGHTS RESERVED.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
```

```
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
# Neither the name of the SONATA-NFV, 5GTANGO, Paderborn University
# nor the names of its contributors may be used to endorse or promote
# products derived from this software without specific prior written
# permission.
#
# This work has also been performed in the framework of the 5GTANGO project,
# funded by the European Commission under Grant number 761493 through
# the Horizon 2020 and 5G-PPP programmes. The authors would like to
# acknowledge the contributions of their colleagues of the 5GTANGO
# partner consortium (www.5gtango.eu).
import logging
from mininet.log import setLogLevel
from emuvim.dcemulator.net import DCNetwork
from emuvim.api.rest.rest_api_endpoint import RestApiEndpoint
from emuvim.api.tango import TangoLLCMEndpoint

logging.basicConfig(level=logging.DEBUG)
setLogLevel('info')  # set Mininet loglevel
logging.getLogger('werkzeug').setLevel(logging.DEBUG)
logging.getLogger('5gtango.llcm').setLevel(logging.DEBUG)


def create_topology():
    net = DCNetwork(monitor=False, enable_learning=True)
    # create two data centers
    dc1 = net.addDatacenter("dc1")
    dc2 = net.addDatacenter("dc2")
    # interconnect data centers
    net.addLink(dc1, dc2, delay="20ms")
```

```
    # add the command line interface endpoint to the emulated DC (REST API)
    rapi1 = RestApiEndpoint("0.0.0.0", 5001)
    rapi1.connectDCNetwork(net)
    rapi1.connectDatacenter(dc1)
    rapi1.connectDatacenter(dc2)
    rapi1.start()
    # add the 5GTANGO lightweight life cycle manager (LLCM) to the topology
    llcm1 = TangoLLCMEndpoint("0.0.0.0", 5000, deploy_sap=False)
    llcm1.connectDatacenter(dc1)
    llcm1.connectDatacenter(dc2)
    # run the dummy gatekeeper (in another thread, don't block)
    llcm1.start()
    # start the emulation and enter interactive CLI
    net.start()
    net.CLI()
    # when the user types exit in the CLI, we stop the emulator
    net.stop()


def main():
    create_topology()


if __name__ == '__main__':
    main()
```

Bibliography

[1] ETSI Industry Specification Group, "Network functions virtualisation (NFV) - network operator perspectives on industry progress." ETSI ISG NFV, 2013, p. 16. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[2] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016, conference Name: IEEE Transactions on Network and Service Management.

[3] A. Leonhardt. (2020) SDN vs NFV: Understanding their differences, similarities and benefits. [Online]. Available: https://blog.equinix.com/blog/2020/03/10/sdn-vs-nfv-understanding-their-differences-similarities-and-benefits/

[4] VMWare. (2022) What is network functions virtualization (NFV)? | VMware glossary. [Online]. Available: http://www.vmware.com/topics/glossary/content/network-functions-virtualization-nfv.html

[5] Ciena Corporation. (2016) What is network function virtualization (NFV). [Online]. Available: https://www.blueplanet.com/resources/What-is-NFV-prx.html

[6] ETSI. (2022) ETSI - NFV. [Online]. Available: https://www.etsi.org/committee/nfv

[7] ——. (2022) ETSI - standards for NFV - network functions virtualisation | NFV solutions. [Online]. Available: https://www.etsi.org/technologies/nfv

[8] ETSI Industry Specification Group, "Network functions virtualisation - an introduction, benefits, enablers, challenges & call for action," 2012, p. 16. [Online]. Available: https://docbox.etsi.org/ISG/NFV/Open/Publications_pdf/White%20Papers/NFV_White_Paper1

[9] A. Leonhardt. (2019) Defining the elements of NFV architectures. [Online]. Available: https://blog.equinix.com/blog/2019/10/17/networking-for-nerds-defining-the-elements-of-nfv-architectures/

[10] F. Khan. (2015) A cheat sheet for understanding "NFV architecture". Section: NFV. [Online]. Available: https://telcocloudbridge.com/blog/a-cheat-sheet-for-understanding-nfv-architecture/

[11] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016, conference Name: IEEE Communications Surveys Tutorials.

[12] VMWare. (2022) What is a hypervisor? | VMware glossary. [Online]. Available: https://www.vmware.com/topics/glossary/content/hypervisor.html

[13] RedHat. (2022) Kernel virtual machine. [Online]. Available: https://www.linux-kvm.org/page/Main_Page

[14] B. Tola, Y. Jiang, and B. E. Helvik, "Model-driven availability assessment of the NFV-MANO with software rejuvenation," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2460–2477, 2021, conference Name: IEEE Transactions on Network and Service Management.

[15] S. Bian, X. Huang, Z. Shao, X. Gao, and Y. Yang, "Service chain composition with failures in NFV systems: A game-theoretic perspective," *arXiv:2008.00208 [cs]*, p. 14, 2020. [Online]. Available: http://arxiv.org/abs/2008.00208

[16] D. Antonelli and W. Johnson. (2020) Emulators can turn your PC into a mac, let you play games from any era, and more — here's what you should know about the potential benefits and risks of using one. [Online]. Available: https://www.businessinsider.com/what-is-an-emulator

[17] B. McGuigan. (2022) What is a computer simulator? [Online]. Available: http://www.easytechjunkie.com/what-is-a-computer-simulator.htm

[18] N. Singh, A. Mura, S. Raghani, and B. Joshi. (2013) What are the differences between simulation and emulation? [Online]. Available: https://www.quora.com/What-are-the-differences-between-simulation-and-emulation

[19] L. Bondan, C. R. P. dos Santos, and L. Z. Granville, "Comparing virtualization solutions for NFV deployment: A network management perspective," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, 2016, pp. 669–674.

[20] K. Kaur, V. Mangat, and K. Kumar, "Towards an open-source NFV management and orchestration framework," in *2022 14th International Conference on COMmunication Systems NETworkS (COMSNETS)*, 2022, pp. 251–255, ISSN: 2155-2509.

[21] Garcia, Vinícius Fülber. (2021) NIEP: NFV infrastructure emulation platform. Original-date: 2017-10-10T17:45:43Z. [Online]. Available: https://github.com/ViniGarcia/NIEP

[22] T. N. Tavares, L. da Cruz Marcuzzo, V. Fulber Garcia, and G. Venâncio de Souza, "NIEP: NFV infrastructure emulation platform," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, 2018, pp. 173–180, ISSN: 2332-5658.

[23] NEC. (2021) ClickOS - systems and machine learning. [Online]. Available: http://cnp.neclab.eu/projects/clickos/

[24] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, and F. Huici, "ClickOS and the art of network function virtualization." USENIX, 2014, p. 16.

[25] J. Martins, M. Ahmed, C. Raiciu, and F. Huici, "Enabling fast, dynamic network processing with clickOS," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13*. ACM Press, 2013, p. 67. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2491185.2491195

[26] M. Peuster. (2016) Vim-emu: A nfv multi-pop emulation platform. [Online]. Available: https://osm.etsi.org/wikipub/index.php/VIM_emulator

[27] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 148–153.

[28] M. Peuster. (2022) Containernet. [Online]. Available: https://containernet.github.io/

[29] F. Fitzek and F. Granelli. (2021) ComNetsEmu SDN/NFV emulator - granelli's laboratory. [Online]. Available: https://www.granelli-lab.org/researches/relevant-projects/comnetsemu-sdn-nfv-emulator

[30] Z. Xiang, S. Pandi, J. Cabrera, F. Granelli, P. Seeling, and F. H. P. Fitzek, "An open source testbed for virtualized communication networks," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 77–83, 2021, conference Name: IEEE Communications Magazine.

[31] Z. Xiang. (2022) ComNetsEmu. [Online]. Available: https://git.comnets.net/public-repo/comnetsemu

[32] Juniper Networks. (2022) Contrail | juniper networks. [Online]. Available: https://www.juniper.net/us/en/products/sdn-and-orchestration/contrail.html

[33] ——. (2022) Cloud-native contrail networking (CN2) | juniper networks. [Online]. Available: https://www.juniper.net/us/en/products/sdn-and-orchestration/contrail/cloud-native-contrail-networking.html

[34] ——. (2022) Contrail service orchestration | juniper networks. [Online]. Available: https://www.juniper.net/us/en/products/sdn-and-orchestration/contrail/contrail-service-orchestration.html

[35] ——. (2022) Paragon insights | juniper networks. [Online]. Available: https://www.juniper.net/us/en/products/network-automation/paragon-insights.html

[36] Calnex. (2022) NE-ONE enterprise network emulator range. [Online]. Available: https://itrinegy.com/ne-one-enterprise-range/

[37] ——. (2022) NE-ONE professional network emulator range. [Online]. Available: https://itrinegy.com/ne-one-professional-range/

[38] M. Peuster. (2021) Vim-emu: A nfv multi-pop emulation platform. [Online]. Available: https://github.com/containernet/vim-emu

[39] ETSI OSM. (2020) Open source MANO 6.0 documentation. [Online]. Available: https://osm.etsi.org/docs/user-guide/latest/index.html

4036 Stavanger
Tel: +47 51 83 10 00
E-mail: post@uis.no
www.uis.no

Cover Photo: Hein Meling