SDN Benefits in a Legacy World

VASILEIOS CHATZIS VOVAS
SDN Benefits in a Legacy World

Vasileios Chatzis Vovas

September 4, 2016
Abstract

This dissertation aims to explore how one could leverage Software Defined Network (SDN) and Network Function Virtualization (NFV) principles in order to realize Service Function Chaining (SFC) in a network. SDN is a new networking paradigm, which makes a network programmable through the use of a software entity called SDN controller. NFV is intended to enable deployment of virtualized network functions, therefore replacing existing hardware solutions. SFC provides the ability to route user traffic to one or more network functions in an orderly manner. SFC will potentially enable many use cases such as data providers being able to dynamically steer user traffic through a set of network functions such as firewall and loadbalancer.

This study is based on a set of goals. These goals evolve around the implementation of a prototype that will enable a SDN controller to steer user traffic through a series of virtualized network functions (VNFs). An important part of the prototype setup is a Network Management Software (NMS) named BECS, which is developed by Packetfront Software AB. BECS is acting as an orchestrator on the network and has complete awareness of all the network devices present on the network it manages. One of the main requirements of the prototype is to enable BECS to communicate with a SDN controller. Once that has been achieved, BECS could provide the necessary information that the controller needs in order to create and install a set of forwarding rules in the SDN enabled switches of the network. All those steps are necessary in order to achieve SFC. In this prototype, SFC is realized by demonstrating the user specific traffic steering through a set of VNFs in a specific order, based on control messages originated from BECS.

Until now, network architecture has been limited to the capabilities of the actual hardware equipment. SDN and NFV help us to overcome this limitation. Information needs to be available anywhere and at any time, in a reliable and secure way. To ensure that, we propose a new scheme of network architecture through our prototype solution. This solution intends to give the ability to network managers to re-shape their networks based on their needs by the use of SFC.

Keywords: SDN, NFV, MNS, traffic steering.
Sammanfattning

Denna avhandling syftar till att undersöka hur man kan utnyttja principer för Software Defined Network (SDN) och Network Function Virtualization (NFV) för att förverkliga Service Function Chaining (SFC) i ett nätverk. SDN är en ny typ av nätverksparadigm som gör ett nätverk programmerbart genom användning av en programvaruexternhet som kallas SDN controller. NFV syftar till att möjliggöra utbyggnaden av virtualiserade nätverksfunktioner och på så sätt ersätta befintliga hårdvarulösningar. SFC bidrar till en förmåga att dirigera trafiken till en eller flera nätverksfunktioner på ett ordnat sätt. SFC kommer potentiellt att möjliggöra många användningsområden, t.ex. uppgiftslämnare som dynamiskt kommer kunna styra användartrafik genom en uppsättning av nätverksfunktioner såsom firewall och loadbalancer.

Studien är baserad på en uppsättning av mål. Dessa mål kretsar kring genomförandet av en prototyp som gör det möjligt för en SDN-styrenhet att styra användartrafik genom en serie av virtualiserade nätverksfunktioner (VNFs). En viktig del av prototypinstallationen är en Network Management Software (NMS) som heter BECS, vilken är utvecklad av Packetfront Software AB. BECS agerar som en Orchestrator på nätet och har fullständig kännedom om alla nätverksenheter som finns i nätverket som den förvaltar. Ett av de viktigaste kraven för prototypen är att göra det möjligt för BECS att kommunicera med en SDN controller. När detta uppnåts kunde BECS lämna nödvändiga uppgifter som styrenheten behöver för att kunna skapa och installera en uppsättning vidarebefordrade regler i SDN-aktiverade switchar på nätet. Alla dessa åtgärder är nödvändiga för att uppnå SFC. I denna prototyp realiseras SFC genom att påvisa den användarspecifika trafficstyrningen genom en uppsättning VNFs i en viss ordning, vilket baseras på styrmeddelanden som härstammar från BECS.


Nyckelord: SDN, NFV, MNS, traffic steering.
Acknowledgments

I would like to thank my mother Matina and my father Georgios for their tremendous support during the whole duration of my studies. Further i would like to thank my girlfriend Amalia who helped me on the grammatical and syntax check of this thesis. Also my professor Peter Sjödin who guided and supported me through the whole process of the thesis. Many thanks to ACREO research institute and specifically, Anders Gavler and Pontus Sköldström for their guidance during this thesis. Of course all these would not be possible without Packetfront AB and their initial idea to extend BECS into being able to control an SDN enabled network. Of course i give thanks to Dave Van t’Hof with whom we undertook this thesis project together.
## Contents

Abstract i
Sammanfattning ii
Acknowledgements iii
Table of Contents iv
List of Figures vi
Acronyms vii

1 Introduction 1
  1.1 Background ......................................................... 1
  1.2 Problem Definition ............................................... 2
  1.3 Goals ............................................................. 2
  1.4 Delimitations ...................................................... 2
  1.5 Structure of the Thesis .......................................... 3

2 Background 5
  2.1 Service Function Chaining (SFC) ................................... 5
     2.1.1 Standardization and current solutions ...................... 5
     2.1.2 Additional requirements and pre-requisites .............. 6
  2.2 SDN .............................................................. 8
     2.2.1 Current Network Design and Limitations ................... 8
     2.2.2 OpenFlow Protocol ............................................ 9
     2.2.3 SDN Controllers ............................................. 10
     2.2.4 SDN enabled Switches ...................................... 12
  2.3 Network Function Virtualization (NFV) ......................... 16
     2.3.1 Virtualized Networking Functions (VNFs) in SDN .......... 16
  2.4 BECS ............................................................ 18

3 Methodology and Approach 19
  3.1 Selecting a methodology scheme .................................. 19
  3.2 Experimental design .............................................. 20
     3.2.1 Limitations and Resource allocation ....................... 20
     3.2.2 Virtual Environment as solution .......................... 21
  3.3 Assessing reliability and validity of the data collected .... 22
  3.4 Evaluation framework ............................................ 23
## 4 Design

4.1 Experimental Design ......................................................... 25
4.2 ODL Modules ........................................................................ 26
   4.2.1 Native Modules ............................................................... 26
   4.2.2 Flow Translator Module Design ........................................ 27
4.3 Demonstration Setup .............................................................. 28
4.4 Allocation of VNFs ................................................................. 29
   4.4.1 Back-end Design .............................................................. 29
   4.4.2 Front-end Design ............................................................ 29
4.5 Demonstration Flow Design .................................................... 30

## 5 Implementation

5.1 Network Infrastructure .......................................................... 31
   5.1.1 Network Entities ............................................................. 31
   5.1.2 Topology and Flexibility .................................................. 32
5.2 VNF Creation ........................................................................ 32
5.3 Topology Discovery & Traffic Steering ....................................... 33
   5.3.1 The FlowController Module ............................................. 34
   5.3.2 The TopologyManager Module .......................................... 36
5.4 Demonstration Flow Setup ...................................................... 38

## 6 Results and Evaluation

6.1 Analysis of Demo Setup .......................................................... 39
   6.1.1 Components of Demo Setup ............................................. 39
   6.1.2 Demo Flow .................................................................... 40
6.2 Result Analysis ...................................................................... 41
   6.2.1 Goal Assessment ............................................................. 41
   6.2.2 Issues ........................................................................... 42

## 7 Conclusions and Future Work

7.1 Conclusions .......................................................................... 44
7.2 Potential Environmental and Social Impact ................................. 44
7.3 Future Work ......................................................................... 46
# List of Figures

2.1 SDN Framework Overview .................................................. 10  
2.2 Opendaylight Overview .................................................... 12  
2.3 OpenFlow Flow Description ............................................. 15  
2.4 NFV Benefits ............................................................... 17  

3.1 Engineering Methodology .................................................. 20  
3.2 Data Collection Process .................................................. 23  
3.3 Issues with too specific evaluation approaches ....................... 24  

5.1 ODL Modules .............................................................. 34  
5.2 Basic Prototype Network Topology .................................... 37  

6.1 Scenario 3 Topology ....................................................... 42
Abreviations

API  Application Program Interface.
B-RAS  Broadband Remote Access Server.
BGP  Border Gateway Protocol.
CAPEX  Capital Expenditure.
CC  Command and Control.
CLI  Command Line Interface.
Gb  Gigabyte.
GRE  Generic Routing Encapsulation.
HTTP  Hypertext Transfer Protocol.
IP  Internet Protocol.
JSON  JavaScript Object Notation.
NAT  Network Address Translation.
NFV  Network Function Virtualization.
ODL  Opendaylight.
ONF  Open Networking Foundation.
OPEX  Operational Expenditure.
OS  Operating System.
OVS  Open vSwitch.
OVSDB  Open vSwitch Database Management Protocol.
REST  Representational state transfer.
SDN  Software Defined Networking.
SFC  Service Function Chaining.
Abreviations


TCAM  Ternary Content-Addressable Memory.

TCP   Transport Layer Protocol.

TLS   Transport Layer Security.

VM    Virtual Machine.

VNF   Virtualized Network Function.

VPN   Virtual Private Network.
1. Introduction

1.1 Background

Network operators use service functions such as firewalls and load-balancers when they are handling user traffic. The deployment of such services currently requires strict and careful network configuration by highly skilled network engineers. Moreover, the physical installation of these services is required in different points of a network in a specific order. Overall these limitations result into an increase of operational expenses (OPEX) for the network providers.

Network Function Virtualization (NFV) [1] and Software Defined Networking (SDN) [2] are coming to revolutionize the way of deploying and managing the services mentioned before. NFV essentially replaces the physical network services hardware with a virtualized instance with the same functionality running in a remote computer in the network. SDN then enables the smart orchestration of user traffic through those Virtualized Network Functions (VNFs) by the use of an entity called SDN controller. This entity oversees the whole network and is able to configure the forwarding devices in the network as required for different traffic flows.

This steering of traffic through selected service functions based on their position in the network is summarized in the term Service Function Chain (SFC) [3]. A service chain consists of a set of services which are to be accessed in a predefined order sequentially. Of course the main issue is the transition period between the networks that do not use any SDN or NFV functionality (for these networks we are going to use the term legacy networks for the rest of the thesis).

Since it is hard to replace every network in the world with SDN enabled hardware and VNFs, the use of a third party network management software such as BECS [4] which is created and developed from PacketFront Software, can make the transition smoother. BECS has the ability to control the legacy part of the network currently, so if an extension was built for it to be able to communicate with an SDN controller then we wouldn’t need to replace the whole network directly. Instead slowly integrate SFC via SDN and NFV with the help of BECS orchestrating both the SDN and the non-SDN part of the network.
Chapter 1. Introduction

1.2 Problem Definition

Integration of SFC functionality in legacy networks seems to be a challenge since there is not real standardization currently, plus current solutions such as OSPF and BGP poses limitations regarding the complexity and their physical location in the network. A potential solution could be combining a Network Management System (NMS), such as BECS, with an SDN controlled SFC domain. Main focus of this thesis is to find a solution on how we could deploy SFC in a legacy network in an efficient way without disturbing the already existing functionality of the network. The solution could include receiving a SFC request from BECS and trigger appropriate traffic steering through the requested SFC nodes from the customer to the endpoint of the service (e.g. Internet).

1.3 Goals

1. Demonstrate a SFC domain triggered and orchestrated by BECS but controlled by OpenDaylight SDN controller with mock VNFs to simulate Network Services that the user traffic should go through.

2. Translate the incoming BECS commands into forwarding entries in the OpenFlow switches by implementing new functionality in the OpenDaylight, which will result forwarding traffic from users to the internet, via customer specific Service Function Chains.

3. Ability to perform SFC in the network without strict VNF ordering in the customer specific chain (cherry picking).

4. Address scalability issues such as increasing number of users and VNFs in the network and the effect it could have on the functionality of our implemented module in ODL which is responsible of translating the BECS commands into actual OpenFlow forwarding entries.

5. Develop a demonstration framework and work-flow in order to be able to present and visualize the SFC realization to the public in a simple but detailed manner.

1.4 Delimitations

This project was sponsored by Packetfront AB which had the intention to examine how BECS as a network management software could expand its current functionality into the SDN network by collaborating with a controller entity. Therefore, external limitation has been for this project, the use of BECS as a third party network management software which will control the legacy part of the network. For the SDN enabled part, BECS would communicate with OpenDaylight, which was the SDN controller of choice by Packetfront based on the extensive documentation and modularized structure.

Also as we will describe in the next chapter in more detail, OpenDaylight enables the developers of adding new functionality tailored exactly to their needs by the implementation of new bundles. This, together with the network awareness the BECS possesses, enabled us to create a SFC solution in a legacy network.

Furthermore, since main focus in this project was to realize SFC in a legacy network using SDN principles, the implementation of real VNFs was out of scope due to time limitations. Instead a packet bouncer was deployed in different virtual machines, which simply bounces packets into the network.
and for every packet it gets an internal counter would be increased by one. This way we know if the specific VNF has been accessed in the correct order in a SFC.

Additional limitation was the inability to use real OpenFlow enabled hardware such as switches due to resource limitations. To counter that, OpenVirtualSwitch (OVS) was chosen in order to demonstrate the functionality of an OpenFlow switch in our network.

Last but not least, since we wanted to investigate the ability of our solution to support a large number of users and VNFs, a virtualized network had to be created due to resources limitations. Mininet was the solution in this issue which enables the deployment of many virtual machines which can be interconnected in a network and can be used as users or VNFs based on our needs.

1.5 Structure of the Thesis

Chapter 2 is divided into three sections. The first section provides the necessary background for understanding the concept of SFC and the limitations encountered, with the current state of the art networks and the need of SDN for a more definite and flexible solution. The second section provides background information about SDN and its principles. Also we describe the different entities that are present in a SDN network such as SDN controller, SDN enabled switch and how OpenFlow protocol can be used to bind everything together. Later, VNFs and their use is analyzed along with their purpose in the network and in SFC. In the final subsection of this chapter detailed information is given regarding BECS and its role in our solution as a third party network management software.

Chapter 3 illustrates the methodology followed throughout the project. The research process and the research paradigm are covered in more extend in section 3.1 and 3.2 respectively. While on the subsection 3.3 the Data Collection process is discussed, together with the different variables during that process such as the sampling size and target population. In section 3.4 a full description of the experimental design is given including the testing environment used to conduct our experiments but also which hardware and software was used through the experimentation stage. In the next section (3.5) the reliability and validity is assessed based on the data extracted from our experiments are by using a plethora of criteria for both attributes. This chapter concludes by the description of the tools, hardware and software that were used in order to evaluate the validity and the reliability of the data gathered from the aforementioned experiments and the overall evaluation framework.

In Chapter 4 the design process of our work is presented. In the first subsection the depth about the experimental design is discussed. Next, the new modules that we implemented in ODL were analyzed, in order to create the functionality needed to realize SFC in our network. Then the demonstration setup of our prototype solution is explained. Also, we discuss the placement of our VNFs in the network. Finally, the demonstration flow followed during the live demo of our prototype to Packetfront, is specified.

In Chapter 5 the implantation process of our work is presented. In the first subsection we discuss in depth about the software and hardware used alongside with the configuration done in our tools during the experimental phase. In the second section of this chapter, we get more in the mechanics of our experiment, the different techniques used and the thinking process behind the decisions that where made, in order to provide traffic steering through a customer specific SFC. Chapter 6 is dedicated to the results yielded from the experiments illustrated in the previous chapter. The first section includes
an analysis of the demo setup followed by a section in which our results are assessed and if the goals set in the introduction chapter were achieved or not.

Finally we summarize all the work done during this master thesis project in Chapter 7. Also the limitations that existed through the whole process of the study are listed and categorized by their attribute. That enables us to easily differentiate between the physical constrains, such as time, experimental equipment and other kind of limitations. Also in this chapter the ethics and sustainability of our prototype solution is discussed. Moreover, how the solution we are suggesting affects economic, environmental and ethical aspects of the society.
2. Background

This chapter provides background information about Service Function Chaining, SDN and NFV. Also the principles that these concepts use are going to be explored, in order to tackle the new challenges that modern network operators face. Additionally, this chapter describes the different entities that the SDN architecture consist of, their role and their position in the network topology as well.

Finally this chapter also includes related work from other researchers that are currently or previously have been working on the topic of SDN and its features. Using this information, the state of the art situation on SDN can be realized before we move into presenting our own design and implementation in the next chapter.

2.1 Service Function Chaining (SFC)

2.1.1 Standardization and current solutions

Service chaining has been a big question to the network community for many years now. Basically SFC is the ability to steer traffic from one service function to another based on the needs of the network administrator. This dynamic way of traffic forwarding calls for an approach that successfully can route the traffic to different places of the network on the fly. Since more and more network functions are being replaced by virtualized versions running on a server in the network, SFC is starting to become a main requirement for many operators and big IT companies.

Most of the current solutions regarding traffic forwarding between different network functions evolve around the use of VLAN tags or Policy based Routing (PBR). Even though these approaches could seem functional at first, they are not dynamic and scalable regarding the placement of the network services in the network topology. In theory, someone can perform SFC in a small network where the network functions are positioned in a rack as physical devices. But concepts like PBR and VLAN tagging will collapse if we replace those network functions with VNFs spread out in the network. Simple approaches that use a mapping between the destination IP address and which switch port, to be used in order to forward the traffic, are very limited. Therefore, they will possibly fail in our study case since IP addresses can be overwritten by layer 3 devices (e.g. router) that the traffic might encounter in the network. The PBR approach uses a combination of the source and destination IP of a packet, in order to determine which forwarding route it should follow. PBR has the same limitation since if a router interjects the traffic the scheme falls apart. A solution here would be to create tunnels between those network entities to keep the IP header intact though.

As described in [5], VLAN tagging can be used to bind mac and IP information into a VLAN tag that then could be used throughout the local network to steer the traffic between different functions in the network. The number of VLAN tags of course is of course finite and in a big network this
approach might not work, especially, if we consider the large number of available VNFs that the user can pick and choose from. Inevitably, the use of encapsulation or another form of protecting the initial packet, has to be considered in order to by-pass the limitations with the approaches mentioned above. In [6] we can find an approach that introduces a new header which will be inserted into encapsulated packets in order to make them find their way through the service chain.

The Network Service Header (NSH) includes information for the length of the header and also the IP protocol that will be used through the traffic forwarding. Then the header includes a Service Path ID which identifies the path that the packet should follow. Essentially, this is a one to one mapping to the service chain the user has requested for their traffic. Another piece of information on the NS header is the Service Index. This index specifies the location of a specific service in the overall chain and has to be decremented by each service function after the packets has successfully pass through. Based on this action, the next node can keep track if the packet accessed all the necessary nodes on its way. While NSH seems to be a very robust solution, introducing a new header always comes with some disadvantages. This additional header needs to be parsed from the network devices. Also after the parsing the device has to take a routing decision based on the acquired information. Essentially, introduction of a new header requires updates on the software of the network devices. This can take time and money and most importantly the standardization procedure can prove to be quite long.

At this point it also becomes clear that approaches as OSPF and BGP which use solutions like the ones mentioned before (e.g. PBR), suffer from a restriction regarding the physical topology location of the network functions in the network as illustrated in [7]. Based on this input, we started looking for alternative methods and techniques in order to realize the SFC in a legacy network and that led us to SDN. According to SDN, the decoupling of control plane enables customized and highly dynamic flows to be installed in SDN enabled hardware equipment such as network switches. NFV benefits greatly from that concept, since the controller can generate a set of flows that will ultimately allow customer traffic to access a set of VNFs in a specific order, both outgoing and incoming from the internet.

2.1.2 Additional requirements and pre-requisites

That Service Chaining concept can revolutionize the current state of networks since the ability to route traffic to specific NFVs enables the Network managers to configure networks with minimal effort and very small error rate. This new advancement can decrease the amount of time and effort needed to create or update a network configuration for a specific host. Ultimately, this can free up time from the network engineers that are currently responsible to take care of these matters. There is a set of main requirements which have to be met in order to successfully set up and configure a NFV chain. First and foremost is the support of multi-versioning and multi-tenancy in the network functions [8]. That essentially means, that one or more virtualized network functions should be able to be deployed in a single physical platform, with the opportunity to be accessed by a number of users and tenants. The advantage of this concept is that the allocation of physical resources is being kept to a minimum and at the same time one can offer the same network function service to many users regardless of their location.

The next step would be to make sure that this kind of networking solution is actually resilient not only in a security manner but also in availability. The idea of virtualizing a network function would not be as attractive if the availability was limited based on location, work load or bandwidth. So these are issues that have to be taken into consideration before even deploying this kind of solutions in a real network. Also the way of configuring and testing the changes in these NFVs should be easily
accessible and fast. Ensuring that way, minimum time will be spend by whomever is configuring these
network function instances. This will result into more efficient and flexible NFV solutions.

Of course the above mentioned requirements are the basic prerequisites for deploying even just a
single NFV. The real challenge comes later on when the NFVs are successfully configured and deployed
in a network. The next obstacle is to find a way to route traffic from a specific client through specific
VNFs and to ensure that the VNFs are accessed in the correct order both for outgoing and incoming
traffic. Additionally, the latter requirement is closely connected with the load generated on a SDN
enabled equipment which is responsible for relaying and routing the network traffic through a specific
chain of VNFs. Therefore, SDN, in collaboration with the VNFs can produce a cloud based network
infrastructure. Using this infrastructure, any customer can specify the network functions that he or
she would like to handle their traffic before accessing the rest of the internet. Ultimately, this can
help virtualize entire network infrastructures by providing remote VNFs and SDN enabled switches
in the between. In the previously mentioned scenario, SDN can lead specifically identified user traffic
through the VNFs by utilizing the controller entity.

The SDN controller is able to generate flow entries and install them in specific SDN enabled hard-
ware. Extending this paradigm, the controller is a highly customizable and programmable piece of
software, it can get policy input from a third party network management entity as described in [9].
After receiving the necessary policies the controller can generate and install to the appropriate switches
a set of flows that will enable user traffic to access a chain of VNFs. The policies provided to the
controller [9] can be, for example, IP address of a user specified associated with the order and the type
of the VNFs that the user traffic should go through.

The number and the type of flows that will be generated from the controller to be used from the
switches can be a key aspect regarding if an SDN solution will be viable in a real high load network.
That is why we should consider the limitations that the SDN switches have and try to find an optimal
way to create as few flows as possible. At the same time we should be able to provide NFV service
chaining capabilities to all users. Let’s assume a very simple scenario where we have a network consist-
ing out of a certain number of VNFs and a small number of users. The main variable in this scenario
is the number of VNFs and clients in the network. So if we don’t use an optimal way of creating the
appropriate flows for the service chaining scenario to work, the more users the network supports the
bigger the load on the switches will be since they are going to carry a lot more forwarding rules than
actually needed. The same issue might rise if one increases not only the number of users in a network
but also the number of the VNFs that the users have the opportunity to access.

Moreover, the position of the VNFs in the network in relation with the SDN enabled hardware is
very important, since careful selection of the location of VNFs can lead to an optimized work flow of
the network and also reveal new methodologies of selecting the right combination of flows. Ensuring
that way the traffic is going through the correct NFV service chain. Apparently, both network topology
specification and optimal flow entry creation logic are necessary in order to produce a final solution
that covers all the different prerequisites we have discussed in this sub-section.
Chapter 2. Background

2.2 SDN

2.2.1 Current Network Design and Limitations

It is safe to say that SDN has become more and more popular the last couple of years, by promising new dynamic techniques to control and manage networks. But the vision has become very wide, so it is quite difficult to give a specific description for SDN. According to [10], ”SDN is a new approach to designing, building, and managing networks that separates the network’s control (brains) and forwarding (muscle) planes to better optimize each other”. And while this was the initial idea that researchers had in mind when developing SDN, it clearly has become a larger concept containing lots of different implementation designs that all use the same basic SDN principles in order to solve different networking issues.

Moreover, SDN in its nature is a very dynamic concept which aims on decoupling the forwarding plane from the control plane, something that can result into less costly hardware equipment. By moving all the control plane processing away from the actual hardware we get the opportunity to program the control plane much easier and in a highly flexible way. The dynamic nature of SDN is one of the main reasons why it has become such a popular concept. Before the arrival of SDN, vendors were required to put a lot of time and effort to develop the perfect control plane functionality on their hardware. To succeed in that goal, vendors had to design expensive chips that would be able to provide and support a very functional control plane. The main responsibility of the control plane chip is to make a decision about what will happen to a packet based on a set of rules that the designer decided upon. At the same time, the time needed to complete this operation should be kept to a minimum, something that adds even more load on the silicon based chip.

Although a vendor can include a great amount of functionality into a hardware equipment, the nature of hardware by itself is limiting. Once the product has been shipped one can of course update existing functionality but not make deeper changes. Moreover, the number of users that every hardware device supports can also be a limiting factor. Just imagine having a 48 port switch and you get client number 4096 (if we consider the use of VLANs). That means you don’t have enough space to support this new customer. Problems like this, can be solved by purchasing a new hardware of the same type in order to extend customer support. And while this seems easy, the cost of this solution can be from relatively low to very high.

That is another point where hardware shows its limiting nature. This can become a serious issue when demand is constantly increasing and vendors have to keep up with the new features that are requested in a very short period of time. On the other hand, one of the core SDN concepts is to allow others than the vendors themselves to develop the control plane functionality by providing for example an OpenFlow interface to the forwarding hardware (e.g. an ASIC). Hardware manufacturers tried to overcome this limitation by releasing products with massive functionality and features, making them this way “future proof” for at least a couple of years. Eventually, the hardware became bigger in size and more expensive in market price. The functionality of the hardware though was very advanced and able to satisfy the needs of the different kinds of customers, regardless if they are internet providers or if their core business is focused on data centers.

Carriers focus on reliability and speed, while data centers want a more concrete and redundant infrastructure to ensure the integrity and availability of data they have in store. So up until now the use of big expensive and highly customized boxes did not seem to be a problem. Companies bought these big hardware boxes with a lot of functionality in a relatively expensive price. Then, they had to
allocate a significant amount of space to set up and store this equipment since that kind of hardware requires a very delicate handling when it comes to cooling systems or humidity and so on. Evidently, entire buildings were allocated to host this equipment, and massive amounts of energy is yearly spent in order to power the hardware plus to keep it in the right temperature. It is more than clear that the needs of the industry, when it comes to networking equipment, is only going to increase. Therefore, the current solution appears to be less and less practical.

SDN is proposing new solutions to overcome these problems. By moving the control plane to a remote server one can make sure that the functionality of the hardware can be dynamic and flexible. On the same time the hardware doesn’t need to do the calculations about what to do with the traffic by itself so the cost of manufacturing the same type of hardware will eventually fall. Another approach that SDN is promoting, is the use of Virtualized Network Functions which can replace the actual hardware boxes such as firewalls and load balancers with a software that will provide the same functionality. This software will be deployed and run in a remote server, where traffic is going to be sent through. This migration can provide more functionality in the networks with, at the same time, lower cost.

2.2.2 OpenFlow Protocol

When we began our research on different kinds of communication protocols used between different SDN entities, we came across protocols mostly like OpenFlow and OVSDB. Of course other protocols have been used in SDN for communication, for example BGP, NetConf and SNMP, but mostly for organizing and managing the network and less to actually program the SDN switches with flows generated from the controller entity.

OpenFlow is an open interface used to control and configure the forwarding tables in network devices such as switches, routers and access points. The process that OpenFlow follows to establish this functionality on the network devices is based on having a set of OpenFlow enabled switches where the controller entity will push the flows into. The connection between the controller and the OpenFlow switches can either be fully encrypted by using a TLS tunnel between the two entities or by a simple unencrypted TCP connection, depending on the level of security required on the current network topology.

On the other hand, OVSDB is more of a complementary protocol to OpenFlow than a standalone solution as explained at [11]. While OpenFlow is used to create and program flow entries into the switch, OVSDB is used to create, modify and even delete bridge ports and interfaces for the OpenFlow switches. OVSDB is widely supported by different network hardware vendors such Arista and Dell and it is highly integrated in most of the OpenFlow enabled switches. In non SDN enabled switches, the forwarding of packets is done on the same device as the forwarding decision of where each specific packet should go. So both data plane and control plane exist in the same device. This results into adding heavier load on the hardware by expecting it to carry out complex calculations over a very limited period of time and at the same time forward the packet to the correct output.

The main difference that separates old hardware by SDN capable devices is that OpenFlow can transport the high-level routing decisions made from the controller entity to the switch, hence enabling better load balancing of the resources in a network device. Furthermore, that transition of the place where the routing decision is made can provide new abilities to the network switches and thus change the way we have been thinking of their abilities up until now. New areas such as virtual machine
(VM) mobility, mission-critical networks, and next generation IP-based mobile networks could benefit greatly by the separation of control and data plane. This separation will make the packet forwarding faster without the need to upgrade the existing hardware.

### 2.2.3 SDN Controllers

Until now the processing of the traffic in a network was done in the hardware. And while that solution is still viable, it comes with a set of problems and issues as well. Hardware is becoming bigger in size and also more expensive. This is connected to the fact that vendors are pushing more and more features and functionality in order to make their products competitive against products of the same type from other companies. Moreover, network hardware vendors want to cover the needs of their customers for the next one or two years until the release of their next updated hardware product.

A SDN controller is more of a concept than an actual piece of equipment. Its concept is based on enabling us to overcome hardware limitations by adding new features on the control plane on demand. To visualize what an SDN controller is and its position on the network, as shown in Figure 2.1, one has to understand first the current architecture regarding the control and data plane in the actual hardware. The data plane is responsible for receiving the traffic from a specific input and forwarding it to a specific output. The current state of the art hardware does that extremely fast and reliable.

![Figure 2.1: SDN Framework Overview](image)

On the other hand the control plane is responsible for deciding what should happen to the traffic when it enters the hardware. In other cases, there are changes that should be done on specific attributes of a packet, and in some cases the traffic can even be discontinued because there is a policy that forbids it to go through to the rest of the network. The control plane is of great importance since it provides all the functionalities and features that a piece of hardware has and ultimately it defines the role of the hardware in a network by providing a specific network function through it. Since there is an issue though with the limited and in-elastic nature of the control plane as it is right now, SDN is proposing that we move the decision making in an external entity. The role of that entity is to get questions from actual hardware and then provide answers that contain the necessary action that the hardware should take if it encounters the same kind of traffic in the future. That way the controller is generating and installing rules, or flows as the SDN term is, to the actual hardware equipment so the latter entity can be able to handle the decision making in the future.
Chapter 2. Background

This approach is based on the premise that the hardware is not able to handle anything but simple forwarding during the bootstrap phase of setting up a network. Then when a packet arrives, the hardware will redirect it to the controller entity through a designated channel of communication. The controller then responds and pushes a set of flows that are installed to the actual hardware. After this point, the hardware will not contact the controller regarding the same kind of traffic, since now it has the necessary forwarding rules to handle this traffic. That way we can have a transparent learning phase which is based on a single packet and a single response from the controller. As the time goes by and the amount of traffic increases, the hardware sends different kinds of packets to the controller and gets the necessary replies, thus becomes a powerful piece of equipment by using the installed flows that the controller entity provided.

There are different kinds of SDN controllers, main difference between them can either be the protocols they use to relay information to the network of SDN switches, or the architecture and programming language that each specific controller has been built in. The first wave of SDN controllers that were developed was based on covering the needs of SDN in a more general concept. But after a while engineers began to develop controllers that were focused on solving more specific problems. For example some controllers provide a more scalable platform so that each developer can create their own extensions on top of the existing code of the controller. That of course agrees with one of the main goals of SDN [12] which is the ability to program the control plane of network dynamically, since everybody can customize their controller based on their specific needs.

NOX was one of the first SDN controllers that acquired worldwide acceptance. It was developed at Nicira Networks alongside the OpenFlow protocol which we will describe more detailed later in this chapter. That particular controller is open-source and provides a C++ API so that programmers can expand and configure it to their specific needs. While NOX is highly customizable, but problems exists regarding network topology awareness [13]. POX is another controller maintained by the same organization as NOX and provides a GUI to manage the controller plus a python API which can result into faster and more user-friendly configuration of the controller. While this was the first generation of controllers, OpenDaylight (ODL) made its first appearance in 2013 and introduced a more standalone way of deploying a SDN controller since it runs within a Java virtual machine, something that makes it possible to run independent of platform.

This high availability, coupled with an easy way of clustering the network, gave some edge to ODL against the other controllers. The OpenDaylight project is open-source and it is widely supported by the industry and companies like Cisco, HP and others. Other controllers have been developed as competitors of ODL like ONOS from On.Lab and have been supported by many companies like Ericsson, Microsoft and others but still it is not an open source project. OpenDaylight [14] offers a wide variety of APIs both on northbound (from the network to the end user) and southbound (from the user to the actual network). Moreover, ODL supports the most common protocols of communication when it comes to SDN as OpenFlow, Open Virtual Switch Database (OVSDB) etc. An overview of ODL is shown in Figure 2.2 [15] where different kinds of APIs and Protocols have been included.
Chapter 2. Background

2.2.4 SDN enabled Switches

Hardware Requirements

Hardware itself appears to play a huge role in SDN principle based network solutions. Even though the whole point of SDN is to move away from being dependent on the functionalities and capabilities of the hardware, SDN sets a list of requirements that the equipment has to fulfill in order to be SDN capable. In the simplest scenario a hardware device can be identified as SDN enabled if it can support a version of the OpenFlow protocol since most of the SDN controllers support OpenFlow among other protocols for their communication between the controller and the SDN enabled equipment. In more advanced scenarios, the hardware will run a customized operating system \[16\] (i.e. a Linux kernel based OS) or even like the one described in \[17\].

Now that we have established the requirements that a network equipment has to fulfill in order to be able to participate in a SDN network solution the question is what is going to be the capacity and the limits of such devices. First and foremost, for SDN to be successful in its goals, the whole communication between controller and switch has to be transparent. That means that the amount of traffic used and time needed for this communication should be relatively small. The controller needs to make a forwarding decision based on a specific packet that it received from a hardware equipment.
like a switch. But if the number of devices that the controller is responsible for increases significantly, then we should consider scenarios as the ones being described in [18] and [19].

Even if we assume that the controller can run at maximum speed and make a routing decision in a minimum amount of time, the switch still has to be fast enough in sending the necessary packets to the controller and receiving the routing decision. As described in [20], another aspect that we need to be aware of regarding the capacity of an SDN switch is the low level signal processing. This can prove to be quite tricky when we have to deal with large bandwidth such as 300 Gb/s.

SDN is trying to tackle this issue by performing more educated decisions regarding which packets will be sent from the actual switch to the controller entity. That way the amount of packet traffic between the two can be kept to a bare minimum. Certain policies and routing decisions can be sent from the controller in a bootstrapping phase to the switches and give them all the necessary information they might need in the immediate future. This initial installation of flows makes the switch able to survive by itself, without contacting the controller unless in specific situations. Eventually we arrive to the conclusion that since SDN is highly programmable and customizable, it is up to the developers of the controller to come up with an optimized solution that will enable the controller to handle the routing decision making in an efficient manner [21] and ultimately minimize the traffic between the SDN enabled network devices and the controller.

Number of Flows as a limitation

At the same time there are more variables that we have to take into consideration when discussing the limitations of SDN enabled switches. One very important issue can be the maximum number of flows that can be installed in a switch and how that can affect its functionality. In the following subsection we are explaining this issue in more detail since it is a main part of our study. OpenFlow enables switches to install flows which later can be used to make forwarding decisions about the traffic based on attributes such as, source or IP address protocol for example. In essence, OpenFlow provides the ability to dissect a network packet and filter on its different elements to make a forwarding decision. Of course this gives us great power over the control plane with a variety of options.

These flows are created from the controller and they are pushed into specific switches. Of course, the controller needs to keep track of these flows in order to have full awareness of its surrounding network. This can ultimately create bottlenecks if we apply this concept in real-life high-performance networks. This proves that the controller can end up being a serious bottleneck to the overall functionality of a SDN network. On the other hand, OpenFlow switches could be a potential bottleneck as well. As described in [22], after experiments that were performed by researchers, there have been results that show that the ratio of data and control plane traffic is four times lower than its aggregate forwarding rate. That can directly be translated as a sign that the amount of traffic exchanged between a switch and the controller can be significantly larger than we expected.

This phenomenon can be explained by the fact that the controller queries the switches periodically for information that later will be used to provide statistic information. Moreover, complete visibility of all the flows in the network can result into hundreds of thousands of flow tables in each switch, something that simple SDN switches are not capable of maintaining. On the other hand, when a switch needs to install a flow it has to contact the controller for it to provide the necessary flows. This action, while good for flow visibility, can generate excess traffic between the two entities. Another possible threat could be not only the number of flows that we install in a switch but also what kind
of flows we use. To elaborate a bit more we start by describing what an Openflow flow can actually look like.

As shown in 2.3, an OpenFlow flow consists of twelve fields which are used to encapsulate all the information that a switch might need, in order to receive a packet from a specific port and then take an action about it. That action will depend on which fields are matched after comparison with the attributes of the network packet that arrives in the switch. This field comparison is not as simple as it sounds thought. As presented in [23], there are limitations based on how many of the fields should the comparison be made of. The reason behind that, as illustrated in [24], is that flow tables are built out of ternary content-addressable memory (TCAMs) that supports wild-card entries and parallel look-ups.

TCAM is a specialized type of high speed memory which focuses on searching the packet’s entire contents in only one clock cycle. That can be extremely useful when it comes to SDN and OpenFlow in specific, since we have large tables of flows that then switch has to go through every time a packet is accessing it. Therefore the use of TCAM is greatly advisable in order to keep up with high data throughput. Unfortunately, the use of TCAMs while necessary, can be limiting at the same time. TCAMs consume high amounts of power and they add extra burden on the asic chip as well. Switches can therefore support a very limited number of such flows which doesn’t even begin to cover the actual needs of modern high bandwidth networks.
OpenFlow-enabled switches from different vendors seem to be able to handle a number of flows that ranging from 750 flows (NEC PF58820) to 4000 (MLX) [25]. Those numbers are surprisingly low, and would make scenarios of SDN integrated networks in real life become non-viable due to the low number of flow entries per switch. At the same time though, these switches can handle up to 80000 flow entries if the fields that are going to be matched belong to the data-link layer of the IP stack. This can be explained by the fact that Layer 2 contains limited information such as MAC addresses, ethernet type and VLAN ID. So if the switch is interested into matching only these fields to conclude to an action, then that can result into making SDN and Openflow a viable solution.

In the case where the maximum number of flows is reached, the switch might start refusing to install new ones. In the best scenario, the switch will try to process them in its software module. The latter will result into a slower decision making from the switch since the matching process will not be done by the high speed asic chip, anymore but from a much slower processor which is responsible for the software part of the switch. ONF acknowledging these scalability issues, suggested the use of multiple flow tables. The advantage here is that each table can match different fields and thus TCAMs could be replaced with much lower cost RAM devices as described in [26]. This suggestion demonstrates that there are ways to move past the hardware limitations by using intelligent network designs and proper algorithms in the controller for the generation and distribution of flows.
Based on that premise, we would like to provide a solution that will be relatively "light" when it comes to the number of flows that a switch should handle. In our scenario, we would like to ultimately built a prototype solution that will be able to route traffic from specific users through a chain of VNFs by only using simple flows. This way we can ensure that each switch will be able to handle the necessary flows for this concept to work. That will of course imply that we can find a clever way to minimize the necessary information that a switch has to check in order to make a forwarding decision about the packet. This can become more complicated when we consider adding a large number of VNFs in the network that can be accessed in different sequences by the hosts. Evidently we needed to research more about what a VNFs really is, its origins and its overall functionality which is covered in the next subsection.

### 2.3 Network Function Virtualization (NFV)

Network Functions have been around since the idea of connecting two computers together in order to exchange data between them came up. The necessity of network functions rose from the nature of the networks themselves. The moment we start thinking about different scenarios that can take place in a network, we will eventually stumble into problems. For example, how can we filter traffic before entering a host, or how can one split traffic between ten computers when all the traffic comes from a single input.

These are just a few of the reasons why network functions exist either in the form of a simple router or more advanced networking equipment such as firewalls and load balancers. On the other hand while this equipment successfully fulfills its purpose, there are issues that are introduced as technology expands and networking becomes more and more essential to everyday life. As described in the beginning of this chapter, the network function equipment is becoming more expensive lately, since vendors keep adding more functionality to their products so they can serve even the more demanding customers.

Not only the hardware becomes more expensive, but also increases in size. Companies allocate great amount of space just to house their networking equipment and a lot of money is also spent in power supply and cooling systems in order to let the hardware function optimally. Most of the networking equipment is powered on continuously for long periods of time without being turned off. That forces the owners of such equipment to spent a lot of money just to maintain it.

The issues mentioned above led the networking community to start thinking of ways to suppress the size of the network functions and provide cheaper solutions with the same or even more functionality. Eventually, the idea of virtualizing the functionality that the network functions provided through the hardware became more popular since one can save space, lower the cost and dynamically create multiple instance of a specific network function on demand.

#### 2.3.1 Virtualized Networking Functions (VNFs) in SDN

As it is delicately described in [27], while SDN was an idea that was created in an academic environment, the idea of VNFs have been developed by service provider companies such as AT&T and China Mobile, just to include a few. The obvious reason that this idea seemed viable to these companies was that virtualizing a network function can solve many problems that were blocking their growth in the networking market. VNFs and NFV in general are trying to solve many issues as described in great
detailed in [1].

At this point one can start seeing clearly some of the benefits that NFV brings to the table as shown in Figure 2.4. Reduction of CAPEX and OPEX, alongside minimizing the space and power consumption are just a few rewards that one can get by using VNFs. Of course the benefits of NFV do not stop there. Also one might say that NFV is a standalone solution and has nothing to gain from SDN. That assumption is partly correct since we can replace our current network hardware with a virtual image that provides the same functionality, without implementing any of the SDN principles.

Of course if we try to implement NFV in a SDN oriented network we can observe that the two disciplines complement each other greatly. Since the controller can make executive decisions about the flow table of a switch, the logical next step is to use that ability in order to redirect traffic through specific network functions. Building up from that concept, by virtualizing the network functions, we gain the ability of steering user traffic though a number of VNFs regardless of their physical location. After the traffic is successfully filtered through the VNFs, then the controller is going to redirect the traffic to the internet. The same procedure will take place (but following the reverse order of VNFs) when the traffic comes back from the internet.

This effect can be amplified by taking the virtualization of network functions one step further. Currently when we talk about network functions most people think of devices such as firewalls, load balancers, B-RAS access points etc. But what if the virtualization does not stop there. As an immediate consequence, the development of customized network functions can begin. Vendors and even individuals will have the opportunity to highly customize their networks by creating virtual images that have functionalities specifically designed toward their needs.

Eventually, this co-existence of SDN and NFV can create game-changing opportunities when it comes to networking as we know it. The ability to filter traffic through specific network functions that run in a remote server with just a single decision from the controller entity can greatly increase the
performance of any network, regardless of its size or location. But we are going to talk about these new flexible solutions in the next subsection of this chapter.

2.4 BECS

We have been saying that the policies concerning how many and which VNFs the traffic of a host shall be routed through, is decided by an external entity. In our case this entity is BECS which is a proprietary software produced and offered from PacketFront Software. This software is a network management tool which offers a wide variety of features, from network topology management to service enabling on actual network hardware devices. In our scenario we extend the functionality of BECS by enabling it to issue policies regarding the NFV chain each host will have to route its traffic to in a network. So essentially, BECS can influence and manage SDN networks by leveraging its already existing network topology awareness and the our newly added feature which creates a logical bridge between BECS and OpenDaylight.

As explained into more detail in [9], the connection between BECS and ODL is based on an inter bundle communication which takes place inside ODL. We achieve this by using Representational State Transfer (REST) calls that are initiated from BECS’s side and contain the necessary information needed in order for ODL to create and install the necessary flows to every switch in the network. OpenDaylight offers a variety of solutions when it comes to the communication from an external source to ODL. These northbound interfaces either use Yang Model Languages in order to translate JavaScript Object Notation (JSON) object send through Hypertext Transfer Protocol (HTTP) requests. Another choice is to create a REST api on the ODL side which will translate calls made externally and then take actions based on the information provided.

BECS has complete knowledge of the network topology plus the IP addresses of each host and VNF. Also it is aware of their location relative to the SDN enabled switch they are connected to. Hence it can make REST calls with providing the IP of a specific host along with the names or IP addresses of the VNFs that the host wishes its traffic to get routed through. ODL will get this REST call and dissect it to gather the different elements contained in it. ODL can later use them to create the appropriate flows and then push them to the OpenFlow switches.

This functionality carries multiple advantages when it comes to real-time SDN network management. First and foremost, the fact that when BECS provisions a service chain for a host, the changes take place immediately since ODL directly translates the call from BECS and uses it to produces the flows. That really positions BECS into a strong orchestration position based on the fact that it can influence the traffic of the network with a single call. In [9] one can explore the different ways that BECS can change the way the SDN network functions. ODL provides a very independent set of interfaces so any third party software can easily send or get information by the controller to make further deduction about the stage of the network.
3. Methodology and Approach

3.1 Selecting a methodology scheme

The main goal of this thesis is to invent a smart way to organize the network in such way that we can achieve transparent forwarding of user traffic through VNF chains, hence realizing SFC in a legacy network with BECS as an external network management tool and orchestrator. The nature of our goals led us to evaluate our results regarding if our prototype solution is able to realize SFC in a legacy network using NFV and SDN principles. After comparison between different concepts of methodology, such as qualitative and quantitative methodology, we decided to follow an engineering methodology as illustrated in Figure 3.1. This decision was based on the fact that our approach was to develop a prototype that would realize SFC in a network, therefore easier to treat it as an engineering problem.

During the first stage of this thesis work, we aimed to find a precise definition of the actual problem we were trying to solve. Then we isolated the aspects it consists of. Due to lack of standardization regarding SFC, we had to improvise and come up with a customized solution that would fit our situation and fulfill all our goals as described in the introductory chapter of this thesis. Finally we concluded in a problem definition by specifying a set of requirements that have to be met in order to conclude if we were really able to facilitate service chaining in a legacy network.

The next step in our process, as illustrated in Figure 3.1, was to begin a process of collecting ideas about how we could achieve a solution to our problem. Careful evaluation had to take place after a significant number of ideas been presented. Finally we could conclude on which solution will be more suitable to be used when developing our prototype. This evaluation process holds special significance since once the development process starts it is not particularly easy to steer or pivot into a new direction. Moreover, building a prototype requires a moderate amount of time and in specific cases resources to be allocated. That is why critical evaluation of each solution has to take place before the beginning of the implementation process.
The evaluation of each idea included an initial feasibility check, since we had a pre-specified time of development and a defined set of resources in our disposal. Next we evaluated how robust the solution would be against a non static number of users in the network and also an increasing number of VNFs. Following we performed a compatibility check of our proposed solution regarding the ability to use it with OpenDaylight SDN controller and BECS as a network management software and overall orchestrator.

When a solution was found that pass the whole evaluation process the development of the prototype started. When the development phase came to an end, we would test the prototype in order to safely conclude if it meets all requirements set in the initial scope or not. In case the solution was not covering all the requirements, we should go back in the brainstorming phase in order to develop new ideas and solutions to our problem, based on input gathered after the testing phase of the last prototype.

3.2 Experimental design

3.2.1 Limitations and Resource allocation

We went into more detail in the previous section about the way we decided to produce and gather our test data. Still we need to define what kind of test environment we are going to use and by which
components it will consist of. SDN by itself is based on the existence of SDN enabled devices in the network that are able to communicate with an entity called SDN controller, which we described more detailed in the second chapter of this thesis. To satisfy this requirement, first we had to decide which protocol the controller and the switches in the network are going to use in order to communicate with each other. Most of the documentation we found pointed to the OpenFlow protocol and since most of the SDN controllers support it we decided to use OpenFlow.

Next we had to decide what kind of SDN controller we are going to use. Between proprietary and open source controllers we chose the latter since we felt the need to construct a test environment that everybody can replicate just by downloading the open source projects from their respected official sites or repositories. With over five well matured open-source SDN controller projects, the decision was not obvious. As we pointed out in Chapter 2, one of the main differences between the existing controller projects was the language that they were developed in and secondly how they were constructed. The latter issue was of main importance since the layout of the controller dictates the way we were going to develop our customized functionality. After all we concluded into a very well-structured controller such as Opendaylight since its architecture was more discrete and easy to understand than other controllers such as Nox or Pox.

Opendaylight is based on the development and use of entities called bundles. These bundles are installed into a main framework and spawned upon demand the controller starts. That structured way of operations coupled with the very detailed and extensive documentation of the project, prompted us into selecting it as our SDN controller for the length of our thesis study implementation. Next choice we had to make was what kind of SDN enabled devices we wished to include in our network. We also had to come up with a solution regarding which kind or brand of SDN switch we were going to use, but we understood pretty early that we might need a dynamic number of switches in order to perform different kinds of tests. Especially when we would increase the number of hosts or VNFs in the network, using real hardware switches, would not be a viable choice.

3.2.2 Virtual Environment as solution

Ultimately we chose to use a set of virtual SDN enabled switches. A couple of choices came up when we searched about it on the internet with the more well documented and supported to be OpenVirtualSwitch (OVS) and Lincx [28]. OVS seems to be a more mature project with a very large support community and a lot of tutorials on how to set it up and troubleshoot problems. At this point, we had concluded the basic element of our network. The next step was to figure out what kind of implementation we are going to follow for the edges of the network such as VNFs and hosts. Soon we stumbled upon a generic solution that included everything we needed in a software call Mininet [29] which provided the ability to create virtual networks with a dynamic number of hosts and virtual SDN switches in it. The tool was relatively easy to use and automate, since it provided a wide series of tutorials plus a very comprehensive API written in Python. One could use it to dynamically create a number of very customized network topologies regarding the number of hosts and switches in it.

Eventually we were able to create dynamic topologies with any number of hosts we needed by connecting them into specific SDN enabled OVS switches. The only thing that was left in order for us to start performing experiments and tests with our virtual network was the implementation of different VNFs. Then we could test if we could route traffic from specific hosts through those VNFs. That particular issue appeared to be extremely complicated since the development of an VNF was a very large research field by itself. Based on the fact that this project was a master thesis with
limited time of execution and development, we decided to simplify the VNF creation issue by creating a packet bouncer written in C. This simple software had a basic functionality of receiving packets from a specific network interface and then bouncing them back into the network through the same interface.

Thus, instead of developing a number of different VNFs to satisfy our requirements we created and deployed a software in a number of virtual hosts provided by Mininet. We distinguished the different VNFs by name and IP address so we could differentiate between them during testing. We also added an internal counter in each bouncer that would increase based on the amount of packets it would receive from a specified interface. That proved to be very helpful when debugging if the bouncer actually receives traffic for hosts or not. Afterwards, the packet counter would be reported to a MySQL database together with the IP of the VNF. Then we could display this information in a simple website that queries the database and prints the counter of each VNF together with its name. Therefore, we could keep track of the accessed VNFs in our network.

3.3 Assessing reliability and validity of the data collected

Since SDN is a relatively new concept and research topic, a lot of different researchers have been diving into it. Although this might be received as something encouraging, we still have to take into consideration the fact that not everybody is successfully following the two basic quality criteria of any form of research, reliability and validity. The validity factor refers to how truthful a piece of information is. In order to actually evaluate if a set of data is valid one has to assess if the data actually measure the subject under study. A brief example could be using a ruler to measure the length of a specific object. Regarding the reliability aspect, as described in [30], one needs to consider the possibility that the equipment that is used for measuring purposes during the experiment phase is somehow rigged or flawed. That might mean it could actually be providing false and not reliable data.

That is a common issue when the data sample is too small or restricted and thus we don’t have the opportunity to take different kinds of samples and cross-check between them through a triangulation method. This method includes a set of different checks or tests used to ensure the reliability of the instruments that are performing the measurements during our experiments. In our research we guaranty the validity and reliability of our results by using very discrete methods during the data collection phase. For example, when we need to ensure that the service chaining procedure succeeds, we need to follow and perform a series of checks to ensure the validity and reliability of the data. Initially we need to be able to acquire quantifiable data from our tests. In particular, when it comes to assessing the functionality of the network, a large number of tests have to take place in order to safely assume that our network behave the way it was intended to. More importantly, the behavior of the network should be consistent regardless the fluctuation of the variables in the scenario, which in our case are the number of hosts and VNFs.

Our main objective is to route traffic through specific VNFs based on policies provided through a third party network management software to the SDN controller. Ultimately those policies would be translated and installed as flows into the SDN switches. In order to conclude what is the optimal and feasible way of implementing this scenario, we had to gather different kind of data. Mostly the date would concern the amount of flows installed in the SDN enabled switches and the stability / robustness of the network. To be able to produce results about all these requirements we use a waterfall approach as illustrated in Figure 3.2 below. The first step revolves around the development of the logic that controller will use in order to generate the flows for the switches. Then we move one step further into our process by evaluating the scalability of the network through extensively testing the
same algorithm while increasing the number of hosts trying to access a number of VNFs.

If the algorithm fails to sustain these enlarged versions of the network we need to make adjustments in the initial logic that the controller is using to create flows. Basic requirement would be for the logic to be able to support multiple hosts and VNFs without disruption of the regular traffic while the host’s traffic gets filtered through the correct VNFs. Then we could move to the next step of the data collection process which concerns about the stability and robustness of the network.

![Figure 3.2: Data Collection Process](image)

In order to evaluate if the solution is robust and stable, regardless the state of the network, we need to gather qualitative data about the network’s state. If the measurements are acceptable then we need to run multiple tests of the same nature on different networks in order to ensure the reliability of the instruments that are used to get the measurements. If the data collected seem to be persistent then we can move on into the next step, which is the enumeration of the number of flows used to setup this routing decisions successfully. In any other case we need to make adjustments in the tools used to acquire the measurements until we get consistent data results.

### 3.4 Evaluation framework

After the data have been successfully acquired, we need to design an evaluation framework in order to ultimately conclude if the prototype solution actually meets all the specified goals of this thesis study.

There have been a series of SDN specific evaluation framework approaches as for example [31]. But most of them focus on a very particular point of interest regarding SDN much like in Figure 3.3. In some cases people try to do a performance evaluation against increasing traffic load and in others people try to check for bottlenecks in the control plane. Our approach includes a dynamically changing network with increasing number of hosts that have to be able to route their specific traffic
through a chain of services which is created and modified on demand. That led us into developing a more generic evaluation framework.

Figure 3.3: Issues with too specific evaluation approaches

Since we need a highly customizable evaluation process, we need to define our own evaluation model and framework. Inside this framework we include a set of methods that aim to test the limits of our prototype against high traffic load, multiple users, different number of VNFs and robustness of the network.

In order to successfully evaluate these requirements we automated a series of tests performed while the prototype is executing in order to conclude if the solution fulfills all the goals in introduction chapter. Initially we try to increase the number of hosts in the network and check if the control plane gets any kind of bottleneck through the process. Afterwards, we continued by adding more VNFs in the network so the hosts can choose over an abundance of service chains. Based on this scenario we try to evaluate if there are any conflicts between different users since there is also the multi-tenancy issue that we are trying to tackle with this research study.

Moreover, we are interested to test what happens if the network undergoes some kind of disruption such as added latency over a main communication link and how that would affect our prototype in a real time scenario. Based on this evaluation framework of highly customized and automated tests, we try to assess the robustness, stability and scalability of our prototype solution.
4. Design

In this chapter we are going to explain the design of our proposed solution that will help realizing SFC in a legacy network. At first we will go through the different entities in our setup and give a basic description about their functionalities. Later we will move deeper into implementation design obstacles and how we overcame them. Lastly, an overview of the demonstration of our prototype will be discussed including the reasoning behind the setup and the way we decided to present our solution.

4.1 Experimental Design

Starting from the top we have BECS which includes different element managers (EMs) that are responsible for the translation of the user input into configuration commands for the network devices. Through BECS’s user interface a network manager can decide to create a connection that will route traffic between two network elements and then push the decision to the EM. At this point BECS will first identify the vendor and model of the targeted hardware and then use the necessary EM to translate the high level decision into a configuration file. That functionality of BECS takes care of the legacy network, but in order for BECS to be able to control the SDN part of the network we need to create a new EM. This new EM must specifically be able to translate the high level traffic steering decision of the network manager into something that ODL will be able to understand. Once ODL acquires this information it will use it to create forwarding flows on the OpenFlow switches.

ODL is a very modularized SDN controller which offers a variety of available interfaces that makes it able to be accessed externally through protocols such as REST, SOAP or even JSON encoded string of information. More information about the design of how we managed to access ODL from the outside through BECS will be provided in the following section of this chapter. At the time we were developing this prototype, ODL had a big selection of built-in modules but not all of them have been fully functional. One of the modules that was malfunctioning was the topology awareness module which could be used to get information inside the controller about the actual topology of switches in the network and their state. This fact in combination with our need of tracking the position and state of end user nodes in the network as well as SDN enabled switches, led us to the decision to create our own topology awareness module.

After the controller has collected the necessary information about the forwarding decisions from BECS the next step is to translate them internally into OpenFlow rules which can be communicated to the SDN switches, which ODL is managing. For this purpose we created a new module in ODL which does exactly what we described above. The module gets as an input information from BECS along with information about the topology of the SDN network. Then it uses this combined information into configuring the OpenFlow switches in the network. If all the steps above are done correctly, we establish a transparent connection between BECS and the SDN part of the network. Then the third
party network management software can use this connection to provide specific users service chaining functionality, by steering their traffic through specific chains of VNFs. As described in previous chapters, the scope of this thesis is more concentrated into providing the necessary infrastructure that can realize SFC in a legacy network. Therefore, we chose to create basic VNFs which just bounce the traffic they receive back into the network. Every time the would bounce a packet an internal counter would increment by one. Therefore, we can be sure that the user specific traffic accessed the specific VNF.

4.2 ODL Modules

4.2.1 Native Modules

Opendaylight prove to be a very easy platform to use and develop on as well. It was also easy to get access to online tutorials and a very active community of users. This SDN controller gave us the necessary tools in order to develop our own modules that would solve our project specific issues and problems. We knew that one of our biggest obstacles would be how we could communicate commands from BECS to ODL and how ODL would translate those commands into something meaningful for the OpenFlow switches in the network. That issue concealed another problem as we realized later on. The underline issue was that if ODL was supposed to configure the SDN part of the network in order to execute BECS’s commands, it would need to know all the switches and end nodes in the SDN network. This need for complete topology awareness proved to be a core element during our designing process.

As described in the previous section of this chapter, ODL was by nature very modularized and that led us to the decision of implementing our own modules in the ODL platform. We also made use of the already developed modules in the controller in order to have a more harmonized communication with the rest of the controller’s platform. Based on our requirements, we needed to be able to acquire data externally from BECS. That suggested the use or creation of an API through which ODL and BECS could communicate. Initially, we though about having a dedicated port open in ODL which BECS could use to connect and transmit the necessary information to ODL. That was not something that could take a lot of time to develop but the drawback was that we had to implement a standalone server in ODL. That server would have to run alongside the other modules and APIs present in ODL in a harmonized way.

To avoid this, we looked for alternatives and more specifically how we could leverage already existing functionality in ODL in order to fulfill our requirements. OpenDaylight comes with many built-in modules that their sole purpose is to send and receive information from outside the controller. So the question was really which kind of API we should use. REST, SOAP and JSON encoded strings were good candidates based on their strong documentation and advanced libraries. But after some research we realized that REST is faster and more simple than SOAP and at the same time provides more capabilities than JSON. Therefore we selected REST as our interface protocol.

ODL had a module called SouthBoundAPI which offered a REST API. This module resulted to be a very good candidate. Before concluding to this module we tried to use other modules of similar nature that mostly were utilizing Yang data models. Even though those modules were fast, the amount of time needed in order to get familiar with Yang Models, their way of working and how we
could configure them towards our needs was significantly more than using SouthBoundAPI and REST.

4.2.2 Flow Translator Module Design

Next requirement was the need of translating the received information from BECS into OpenFlow commands that later would be installed in the SDN enabled switches in the network. Here ODL had very few options available among its built-in modules. A couple of translator modules were available but nothing that could cover our very customized needs. Therefore we concluded into creating our own flow translating module which we called FlowController.

This module would be responsible of collecting data from the SouthboundAPI. Then it should create rules according to BECS's commands and finally install them into the OpenFlow switches. Though we soon realized that we need to have complete knowledge of the network topology in the FlowController module. The topology information was a required input in order to be able to push the correct flows in the correct switches depending their position in the network. Moreover the relative connections between the switches, the positions of the VNFs and the user nodes in the network proved to be a very important factor in our solution. Based on this information the requirement of complete network topology awareness was created.

This new requirement of network topology awareness, made us look through different options about how we can realize the network topology inside ODL. Again, standalone solutions were quickly ruled out since it was hard to implement external bundles and harmonize them with the other ODL modules. That is why we looked through the built-in modules in ODL hoping we can find something that could fulfill this requirement. At first we found a module which was called TopologyAwarenessManager and its purpose was to continuously gather information about the status of the network and its topology.

We tried this module in many different scenarios only to realize that it was not as reliable and stable as we needed it to be for our prototype. This was of course expected, since ODL is an open-source project. At that point we decided to create our own module in order to fulfill the topology awareness requirement, which we named TopologyManager.

In our module we wanted to be able to first acquire information about all the existing OpenFlow switches in the SDN network. Then we were interested into knowing the interconnections between those switches so we can have the ability to construct traffic paths between known switches if needed. The final step towards complete topology awareness was the ability to know which end nodes were connected to which switch and all the available information regarding their identity, meaning their IP and MAC addresses. Moreover we would like to know if an end node is a user node, a gateway to the internet or a VNF.

When all these components would be summarized we could have a transparent work flow between BECS and the steering of the user’s traffic. All it would take is our two new modules and then we could fulfill all our requirements. During the next chapter we will go deeper into details about the actual implementation of those aforementioned new modules we would introduce on the ODL platform.
4.3 Demonstration Setup

Since one of our main goals in this project was to realize SFC in a legacy network, an important requirement was to find a way to visualize the actual service chaining done in real time. It is particularly difficult to really demonstrate how traffic steering is done since the packets are changing nodes based on information that BECS gives into the network. Therefore, just showing a trace of the packets that are transmitted over the network is clearly not enough for an observer to understand the different SFC changes that are done in the SDN network.

To overcome this issue, we start thinking of visualization techniques that could enable us watching the path of a packet from source to destination and through which service chains it could go through. One suggestion was to have packet sniffers in crucial parts of the network as checkpoints and use the packet traces acquired from them. In a later step we could analyze the data with a rendering algorithm that could isolate a single packet from the moment it has been originated from an end user node. Then we would try to find the fingerprint of this packet through out the network in order to conclude if this packet, which was a part of a user’s specific traffic, went through all the necessary VNFs in the network before reaching the gateway node.

This scenario proved to not be the best candidate since the time that we needed to spend in order to create this packet tracking algorithm was more than we could afford based on our implementation timetable. Instead we came up with a more efficient idea which was to make each VNF a checkpoint. We could manage this by adding an internal counter in each VNF which would be increasing by one for every packet that comes through the VNF. That was easy to implement since our VNFs were a basic packet bouncer software. Of course we would have to extend its functionality in order to give us a clear view of the trail that a user specific traffic follows, but more information about the design of the VNFs we will provide in the following section of this chapter.

Ultimately, by using those internal counters per VNF we were able to pin point if a series packets originated from a specific user were passing through the correct VNFs and in the correct order according to BECS guidelines. Of course during the development phase this was able to be done by checking debugging messages from each specific VNF. But for a demonstration this wouldn’t be a good solution especially if we have a network with five or more VNFs, then all this different command prompts would only confuse the audience.

In order to overcome this problem we expanded our VNF design by adding an ability to report the internal counter in a database in real time. This was a big step towards making SFC more visible. At that point the counters of each VNF would be stored in a database which we could visualize through a simple interface in a web browser. This way every member of the audience could just type the IP address of our website and there they could see live which VNFs had active counters, which meant that those VNFs are part of an active chain at that particular moment.
4.4 Allocation of VNFs

4.4.1 Back-end Design

The design of the VNFs played a big role in our SFC realization. Since the VNFs were the key network services that we were routing traffic through, their design and their capabilities could change the way that the whole network was behaving. In theory a VNF could be any piece of software that runs in a workstation which is part of the network. This software could receive packets and after some kind of packet parsing it will re-send them back to the network or not send them back at all (e.g. firewall functionality of dropping packets).

Initially we thought about using already made virtual images of network functions such as virtualized firewalls and load balancers to participate as VNFs in our network. After some searching we found out that those images had very specific dependencies as to which SDN controller we needed to use. They actually dictated the overall infrastructure and architecture of the network they were in. This would not really work with our concept since our prototype needed to be highly customizable and flexible regarding changing the architecture of the network based on the main requirements that we had from the beginning of the thesis from PacketFront.

Therefore we removed all the unnecessary functions of a VNF and as a result we were left with a VNF that was just able to receive and re-transmit packets. This seemed really close to the concept of a packet bouncer which was exactly what we decided on implementing. For us it was extra advantageous to create our own simple VNF since we could decide what could happen within the VNF with the received packet before sending it out to the network again. This could give us more freedom when designing the rest of our network solution since the VNF could provide help, as a debugging checkpoint for the traffic in the network. Our goal was to realize SFC in a legacy network by steering user specific traffic through a set of VNFs in a specific order, specified from BECS and translated into OpenFlow rules from ODL. Therefore, in a simplistic scenario if user A is supposed to route all its traffic through VNF B and C, then we would be sure that SFC was successful if these VNFs received traffic originated from this user in the correct order.

4.4.2 Front-end Design

What we needed at this point was a method of debugging if traffic was sent through the required VNFs or not. Since we had complete control over the code that the bouncer was built upon we just had to add a function counting incoming packets to the VNF from the rest of the network. This was extra important to isolate the VNFs from the rest of the network and allow only traffic originated from user and gateway nodes to access them, in order to be sure that SFC was working correctly. For that reason we connected the VNFs directly into their designated OpenFlow switch which we gave the name Command and Control CC switch. More details about the different types of switches in the network is provided in the Implementation chapter.

The CC switch acted as a firewall for the VNFs, letting only valid traffic through, based on the OpenFlow rules provided by our FlowController ODL module. Now the only thing that was left was the ability to visualize the counters of each specific VNF in order to have an overview of what was happening in the network in real time. At first we tried different approaches as to acquiring the counter from each VNF by adding different kind of interfaces such as REST and JSON APIs on each bouncer. That way we could contact externally and query the bouncer for its current counter. Soon we realized
that this approach would produce extra workload and complexity on our solution so a more simplistic approach had to be found.

Then, we came up with the idea of having a database running in a server that all the bouncers could access. The bouncer could add or update an entry in a table in the database by opening a direct connection with it. Then it would send information regarding a specific VNF and its current counter. This was a very effective solution since it was simple to setup and really flexible regarding visualization of the VNF counters. That flexibility originated from the fact that the contents of the database could be visualized by many different user interfaces.

Finally, we created a user interface hosted on a web server that as a single purpose had to visualize the content of our database. The flexibility provided, from the easy to develop and visualize web user interface, discouraged us to consider other interface approaches. The main point of this decision was that we already had a server allocated for the database. Therefore, it was very easy to just add a webserver functionality on the same server, eliminating any need for developing standalone solutions as a user interface. A standalone solution would be more appropriate in case the interface could directly contact the bouncer and acquire the counter from it and then visualize the bouncer’s response.

4.5 Demonstration Flow Design

Our final challenge was to design an appealing and easy to understand demonstration flow in order to present the realization of SFC to PacketFront during a demo session. This was an overly complex concept, with many different actions happening in the background. Therefore, we needed to find a balance between distancing the observers from the low level aspects but still giving them the opportunity of realizing the traffic steering actions needed from BECS. BECS comes with a comprehensive user interface that is very visual and easy to use. This GUI was used as a first step in our demo flow. We would show how we can specify which VNFs should a user’s traffic go through and how somebody can input that through BECS’s interface. The next stop in the process is the translation of those commands from the FlowController module into OpenFlow rules based on input from the TopologyManager module. This step we decided to not visualize during the demonstration since it would be extra hard to show how the internal ODL modules work in a comprehensive way for the observers. Instead we skipped into the step where all the rules have been translated and pushed into the appropriate OpenFlow switches.

From that point on, the only thing left to do was to open a command prompt from one of the user nodes workstations and send a series of ping messages to a target IP on the internet. Then the audience would directly see on our web interface the counters of the VNFs specified during the first step from BECS as a SFC for this user. The counters would then start to increase, while we initiate ping packets from the user’s node. Ultimately, we wanted to show how flexible our approach is concerning changing the service chain for a specific user and even adding new users in the network. This was part of our demonstration as we could re-configure the VNF chain of a user through BECS. Different counters would then be activated in the web interface. That gave a real-time response feeling to the audience. Also it simplified the concept of SFC and how it could be realized it in an legacy network by the use of SDN and NFV principles.
5. Implementation

In this chapter we are going to discuss the experimental setup of our prototype solution into more detail. Our solution is focused on creating a dynamic topology where every host routes its respective traffic through a service chain of VNFs. This chain will consist of specific VNFs which are based on the flows issued by the controller entity and installed to SDN enabled network devices.

Initially we are going to cover the different hardware and software used during the experimental setup and the reason why we chose them specifically. Since SDN is a very wide subject, there is an overabundance of choices when it comes to creating a network that will use SDN principles in order to optimize its performance. Below we are going to argue about our decisions and the reasons behind them.

5.1 Network Infrastructure

5.1.1 Network Entities

As we pointed out in the previous chapter when we were trying to identify what kind of prototype solution we were going to develop, we concluded that we were in need of creating a network that included a dynamic number of hosts and VNFs at the same time. Moreover the network should be dynamic regarding its topology and SDN enabled. This last requirement prompted us to search what kind of SDN enabled hardware was available out in the market. As we expected, many vendors like Cisco, Juniper, Extreme and others had already released a series of hardware that supported SDN.

Due to physical limitations we were unable to find such equipment in short period of time. Hence we decided to look into alternatives. As many have been pointing out on the web [32], a very easy and viable way to setup a SDN enabled network is the use of virtualized switches. These switches are capable to communicate with a SDN controller through many protocols, OpenFlow included. We went through a couple of the available choices such as Lincx and OVS and finally we concluded to use OVS as our SDN switch. We based our decision on the fact that OVS provides a good amount of documentation and a support community that gives feedback to many bugs and difficulties one can come across while configuring and using OVS.

Moreover, our basic setup included the need of VNFs. Again, there were vendors supplying a set of basic VNFs such as firewall, NAT, load balancer etc. but we would like to have VNFs in our network that were highly customized to our needs and also open source.

We finally decided to develop a simple VNF software which would receive packets from a specific interface and bounce them back into the network from the same networking interface. The reason why we concluded to use that kind of VNFs, was that our main objective in this thesis project was to
illustrate that we can route traffic from different hosts through different kinds of VNFs in a specific order. With that said, it wouldn’t really matter if we use actual VNFs or a simple software that imitates the basic functionality of a network service.

5.1.2 Topology and Flexibility

Finally, we were in need of a dynamically created number of hosts to participate in our network and create traffic that would be routed through a different set of VNFs for each host. This last requirement really pushed us into considering our experimental setup since the amount of virtual machines needed to run our experiments would increase radically when we were about to reach the end stage of the testing phase. Ultimately this made us understand that using different VMs for each host, VNF and SDN switch would not be viable.

Mininet proved to be the optimal solution in order to run virtual networks consisting of a large and dynamic number of hosts. Although that solved the issue of the network setup, we still had to figure out where the VNFs would run and also which SDN switches we could create inside Mininet’s network. Finally, we decided to run our VNFs in different hosts in the network. Mininet fully supported OVS, so our complete setup was able to be developed purely by using Mininet.

Mininet appears to be an all-round solution when it comes to actualizing networks. It is written in Python programming language and it provides an API alongside a fully functional CLI. This way the users can create their own automated scripts that produce a virtual network, when executed by Mininet.

The API provides access to all the calls that Mininet uses. Hence we were able to automate the creation of our Mininet topology by taking advantage of this capability. The CLI also provides calls from the core functionality of Mininet but most importantly it contains a set of commands that can be used after the initial topology script has been run. These commands can perform a variety of actions including making every host in the Mininet’s network ping each other or to run stress tests on the network by creating huge amount of traffic, adding latency etc.

5.2 VNF Creation

Our approach to the VNF creation was simple. We developed a software in C which utilizes the pcaplib library. This library offers some extra network functionality by being able to intercept packets before being processed by the kernel of the operating system. When the packet is captured it is parsed and copied into an IP packet structure and stored into memory.

At this point, we have all the packet information and we are able to recreate the captured packet and then send it back into the network. Also when a packet is received an internal integer counter is increased. In that way we can keep track of when the VNF is accessed. Once the counter is changed, we need to visualize that change. For this reason we used a remote web server with a MySQL database to store the IP addresses and the counters of every VNF. In that way we were able to keep track of each VNF’s counters in real time.

For that reason we built an extension to our original VNF software where we used the native MySQL library for C. This library gave us all the necessary functions in order to connect to a remote
database and make SQL calls to it. Essentially, we inform the database of the IP address of the current VNF that its counter has increased. We also provide a web interface in the form of a website where we visualize the current state of the packet counters per VNF.

We distinguish the VNFs by IP address of course but we have another feature where the third party orchestrator software gives a mapping of the IP of a specific VNF and the real life name of the VNF as well (e.g. firewall, load balancer, B-RAS etc.). This information is transferred to the website and then we are able to present the name of the VNF together with the counter information for all the network services in our network.

Furthermore, we extended our web interface to be able to make real-time calculations by setting starting points for the counters of the VNFs. This way we are able to debug any kind of changes that may take place from the side of the SDN controller. For example, if there is a missing flow in our Command and Control switch we can actually see it by generating traffic from all our hosts and then check through our web interface if the VNF counter is increasing according to the policies that BECS provided or not.

Ultimately a set of visualization effects have been added on the visual web interface in order for us to best identify which VNF is currently in use. All these appliances have been added in order to better debug and check the reliability and validity of the data collected. Since a large number of test cases had to be done, this visualization approach significantly decreased the time needed for data collection and evaluation.

5.3 Topology Discovery & Traffic Steering

One of the big implementation decisions we had to take during the development of this prototype was how to add complete topology awareness in our SDN controller. ODL provided a set of built in modules that were using LLDP packets in order to get an overview of every node and switch in the network. These modules were not completely functional by the time we were doing our implementation so instead we decided to create our own modules in ODL to handle the topology awareness issue.

For this prototype we needed to create two new modules in ODL. The first module we called FlowController 5.1 and gave us the ability of receiving commands from BECS in the form of REST calls. Using REST to interface between different network entities, is common practice nowadays since it is cross-platform compatible, very well documented and fast. Additionally, this first module was able to parse the incoming information and then combine it with the output from our second module called TopologyManager into OpenFlow rules.
5.3.1 The FlowController Module

The FlowController module is the core of our prototype and uses many already existing modules from ODL in order to perform all its tasks. Initially it uses the SouthBoundAPI module from ODL which provides a socket in the ODL controller that externally can be accessed by different methods such as JSON encoded strings and REST or SOAP calls. Our preferred way of communication between BECS and this module was using REST calls. Those calls included very specific information from BECS about the user identity that BECS wanted to provide a traffic steering decision together with the identity of the VNFs that the traffic should access and in which order. All this information was provided into a tuple of strings which was decided from BECS and later encapsulated by an element manager of the BECCS platform into a REST call to the socket of ODL’s SouthBoundAPI. The question arises of how BECS should know the necessary information in order to make this call to ODL. Actually BECS needs to have awareness of two things before being able to make a traffic steering decision in the SDN network. Initially BECS needs to know the MAC and IP address of all the nodes in the network that BECS is managing. That will make BECS able to query our FlowController module in ODL which will relay the question to the TopologyManager module that has complete awareness of all the entities in the network. This last module keeps a database of all the active end nodes which in our scenario can be considered clients or users in the network.

Essentially every customer has a unique ID which is mapped to their IP and MAC address and is used as a reference when BECS wants to perform a SFC decision about a specific user’s traffic. We are going to give more information about the design of the Topology manager in the following subsection of this chapter. At this point an obstacle was identified regarding how BECS will be informed in real-time about the existence and the IP address of a new user. The solution here came by BECS’s own architecture and Packetfront’s way of working. BECS is a commercial software, which is used by various companies in order to manage their networks. Packetfront is also providing network management and configuration support to their customers alongside with BECS. Therefore every new node that joins the network has to be registered to BECS directly. That usually happens through an official request done through telephone call to Packetfront in order to ensure security and stability of
the customer’s network.

That means that every key device, such as a customer’s router, is registered directly to BECS. Therefore we don’t need to add extra functionality to ODL in order to trigger information sharing about end node’s IP address with BECS. In our prototype, BECS already knows the IP addresses of all the end nodes but not their relative position to the ODL controller or any of the OpenFlow switches. This is covered by the TopologyManager module we developed. The network service functions in our prototype are VNFs which exist in a specific part of the network, allocated for VNFs only. Using this approach, BECS doesn’t need to keep track of the IPs for these VNFs. The idea behind this design was that we wanted to make a solution that will realize SFC in a legacy network through the creation of a prototype. This prototype, should be scalable in the future by being able to support increasing number of VNFs that can be located in different part of the network. BECS should only provide a chain of names of VNFs that it wants a customer traffic to go through. For instance, we have customer "A" with IP of 10.0.0.10 and we want all traffic generated from this customer to be routed through a firewall and a load balancer, in this specific order. Then the only thing that BECS would need to send to ODL is the following tuple: (ip=10.0.0.10, vnf1=firewall, vnf2=loadbalancer).

This input is enough for our Flow controller module, which will do a look-up on the TopologyManager’s module database and find which VNF has been tagged as firewall and its position in the network relative to its closest OpenFlow switch. Then the FlowController module will make calculations based on the input information and give us as an output, a set of OpenFlow rules. These rules can make one or more SDN switches in the network able to route the user’s traffic correctly through the VNFs specified from BECS. The rendering of the information into rules is done by using the user’s IP address in combination with its MAC address. The reason why the MAC address is used is that the source IP address will be overridden after accessing a layer 3 element such as a router or in our case one of the VNFs. Therefore, another rule has to be made to repair the overridden IP based on the packet’s original MAC address. This happens after a look-up on the TopologyManager’s database in order to determine if the IP and the MAC of a certain packet is not in sync. In this case our FlowController module will install a correction OpenFlow rule on the SDN switch that will change the IP back to the original one. This way we ensure that the packet has been intact throughout the SFC procedure.

Finally the FlowController module will create rules that actually forward the traffic through specific VNFs in the correct order. For this we need a set of rules that check the source IP of the Packet after coming from a VNF and before its corrected. This information is used as a reference to which part of the service chain the packet was during its last hop. Then another rule will forward the packet to the next VNF and continue to do so, until the packet has passed through the whole SFC. Then the packet will be sent to the default gateway which for us is the gateway to the rest of the internet and the final destination of the outgoing traffic. Of course, we are not only interested for our traffic to go through the VNF chain on its way to the internet but also to go through the same chain, but in reverse on its way back from the internet. This results into a new set of flows created by our FlowController module that ensures that the traffic will successfully reach the end user after passing through all the VNFs specified by BECS in the reverse order.
5.3.2 The TopologyManager Module

Topology layout and Awareness

We have already mentioned part of the functionality provided from the TopologyManager module but here we will explain in more detail the implementation and the reasoning behind this module. This module was designed to provide real time information about the position of new end nodes, such as VNFs and users, of the network. In addition to that, a very important requirement was that this module should be able to map the connections between the SDN enabled switches in the network. That way we have complete awareness of our topology at any moment, something which can be crucial in our goal of realizing SFC in a legacy network.

Initially we used a built in module of ODL which sends discovery packets to all the OpenFlow switches that ODL is connected to. This discovery packets are sent automatically from ODL and can be leveraged in order to know if and when, an OpenFlow switch becomes available in the network and which is its MAC address. Our TopologyManager module stores this information in a database for future use. Also ODL has a module that gives us the opportunity to gather information about the interconnections between different switches in the network but this module was not stable by the time our implementation was ongoing so we decided to add this functionality on our TopologyManager module. In order to figure out the connections between the switches in the network we leverage the fact that OVS switches, when deployed, do not have any OpenFlow rules installed to them. So the default action when they receive a packet from and interface is to broadcast it. That way all their neighbor nodes will receive a packet originated from their neighbor at some point. When that happens we add the connection information in our database in a separate table that has as a reference the previously discussed table of the OpenFlow switches that includes the MAC address of the switch with a list of their known interfaces. In real life, this is a reference to a switch port but since we are using virtualized switches, ports are translated internally in the switch’s software as interfaces.

At this point we are able to know all the available switches in the SDN network plus the available connections between them. Next we are interested into having a database of our end nodes that are connected to an OVS switch. That can be easily acquired as soon as we have populated our switch database. After this step is done we will leverage the fact that in an SDN environment, when a switch receives a packet from an end node it will relay that to the controller. That is done in order to ask for advice on what to do with that packet in the future in order to avoid re-sending it to the controller and creating extra traffic. At that step we capture the packet which includes the end node’s MAC and/or IP address, depending what kind of packet it is. Then we can make an addition to our database of this information as a new end node entry. The information would include the MAC and IP address plus a reference to which switch this end node was connected to for future use.

The last step in order to have a complete picture of our network is to differentiate the end nodes between users and VNFs. Since we are using virtual machines for both VNFs and users in the network, its really hard to differentiate between the two in our end node database. For this reason we made our suggested network topology more strict regarding the actual placement of the VNFs. We decided to have dedicated VNF switches in our network to overcome this issue. This means that if a switch is designated as VNF switch, all the end nodes connected to it when identified, are going to be tagged as VNFs. This decision influences us to evolve our network setup and divide our OpenFlow switches into three categories which are gateway, user and Command and Control (CC) switches. When a switch is identified by our TopologyManager module it will be tagged as either a gateway switch or a user switch depending its position relevant to our CC switch.
The Command and Control switch

The core of our SDN network when it comes to the network devices in it, is the CC switch. This switch category is responsible for the relay of information between user end nodes and VNFs. Of course one can have multiple CC switches in a network but since this project was oriented towards a proof of concept we decided to have only one CC switch in order to simplify the network architecture and reduce the amount or required resources. As illustrated in figure 5.2 the CC switch, which is referenced as "S2" in the figure, has direct connection with the VNFs in our network. That was a solution to the problem that we faced when trying to distinct between VNFs and user nodes on our TopologyManager module. So based on this setup, every node that is directly connected to the CC node is tagged as a VNF in our end node database.

![Basic Prototype Network Topology](image)

The same logic applies to the rest of the switch types. Gateway switches are defined as switches that are connected to end nodes that represent the gateways to external network (e.g. internet) and those nodes are the end destination of outgoing user traffic. Also all the replies from the internet to those users traffic are originated and treated as incoming traffic from those gateway nodes. For the user nodes, we followed the same concept as for the gateway nodes. If a switch is tagged as user switch in our TopologyManager module, all the nodes connected to it are considered and tagged as user nodes. These nodes are the initiators of traffic from inside our SDN network and therefore these are the user specific traffic we wish to ultimately steer through a service chain.

The last issue that still existed was the categorization of the switches of the network between CC, user and gateway switches. There were many suggestions regarding how to solve this core issue. One
of them was to make the architecture of the network very static and therefore kind of pin point that a CC node always should be between the gateway and user switches. This approach would simplify our problem by design but we had a main requirement from PacketFront that we should be able to specify the type of a specific switch in the network on demand. Also if we consider the scalability factor of this approach when we would need to increase the number of users and VNFs in the network, one CC switch will not be enough and more should be added in the network. This scenario can be expanded into adding more user and gateway switches and suddenly the static network architecture approach collapses.

Our solution to this problem was to leverage BECS’s orchestrator position on the network by making BECS able to directly categorize the available switches in the network as CC, user or gateway. The only thing we needed to add in our design is the possibility of BECS sending REST command to our TopologyManager module mapping specific switches to a distinct type. For example: (S1=user, S2=CC, S3=gateway). Utilizing the SouthBoundAPI module from ODL we were able to get this command from BECS and tag the available switches in the switch database. More regarding how we managed to send this command from BECS can be found in [9]. Everything from that point on can be handled automatically, since we have information about all our switches and all the end nodes in the network and their roles as well.

It’s worth mentioning that an SDN switch is considered a network element that BECS is informed about its physical existence from the clients of Packetfront through an offline communication. Therefore there was no need to be able to send information from our TopologyManager to BECS about the available SDN switches and their IDs. BECS knows this information by concept and the only thing needed is for BECS to map the switch identities to specific types, something that also provides extra flexibility on the network. BECS only has to send a new role-assignment command and the network can shift according to BECS’s commands and the wishes of the network administrator.

### 5.4 Demonstration Flow Setup

The flow of our demo was implemented very close to what was described in the previous chapter. Since our implementation provided us with a visual interface, we could show the differently accessed VNFs at real-time. BECS’s user interface could be used to issue the SFC commands to our ODL module.

Part of the demonstration was the addition of new user nodes during the presentation in order to show that our prototype solution is really flexible and scalable regarding increasing the number of users in the network. The only thing that we had to do was to send a new command from the BECS’s user interface, that would inform ODL and our FlowController module about which user’s traffic should be steered through which VNF chain.
6. Results and Evaluation

As explained in the Implementation and Design chapters, we created a network topology where we have a set of interconnected SDN enabled virtual switches that have specifically designated roles. Based on these roles, the switches receive different OpenFlow flows from the controller in order to function correctly in the network and route the traffic of specific hosts through a set of VNFs successfully.

In this chapter we will go through the demonstration setup of our prototype and what results were gathered after performing the demo. Next we are going to assess if the goals set in the introduction chapter of this thesis have been reached or not. Finally, we will describe any remaining issues that our prototype solution was not able to cover. Suggestions regarding how those remaining issues could be solved will be described in the next and final chapter called Conclusions and Future work.

6.1 Analysis of Demo Setup

6.1.1 Components of Demo Setup

In order to explain our solution regarding the realization of SFC in a legacy network, we decided to perform a demonstration of our prototype to Packetfront AB. This demo should include a complete run of our prototype, with the aim of visualization and better understanding of how we can perform service chaining, using SDN and NFV principles in a legacy network which is controlled by BECS.

In the previous chapters we described in detail the different particles of our prototype and the core modules which consist of our own developed OpenDaylight modules. These modules gather information from the virtual network we created using Mininet. The information is then stored into a local database in order to later be combined with the administrative SFC information given by BECS. The SFC policy provides mapping information about which chain of VNFs should a specific user’s traffic go through. The final result consists of a set of commands send from ODL controller to the OpenFlow enabled switches. Those commands will configure them so traffic originated from that specific user will go through all the VNFs, already specified from BECS, and in the right order both on the way to and from the internet.

The network we used during the demo setup is represented in 5.2. Main purpose of this demo was to show that SFC could be realized in a legacy network, which meant that the amount of VNFs and users in our network was not as important as the proof that SFC could be achieved. So in order to skip long configuration time and make a short but detailed demo, we decided to limit the number of client in the network to two and the number of VNFs to three. The demo would also include a presentation of all the actions taken in the background by the different entities of our setup. For this setup we also used one Command and Control switch which was connected to a user gateway switch, a set of three
VNFs and finally an internet gateway switch which was acting as an endpoint to the rest of the internet.

It became clear to us that due to the complexity of actions been performed inside the ODL controller we need a set of indicators that would help the observers of the demo to understand what is happening in any given point of the presentation. This resulted into implementing a web interface which would be used as a way of representing the state of the different VNFs in the network and how many packets those VNFs have received in any given point.

The visualization of the information would help the viewers to understand how an action given by BECS could result into user traffic steering in an actual network. For this reason we set up a webserver to host our website accompanied with a database server to store and update the current amount of packets received by our VNFs. Last addition to our demo setup was the use of BECS’s user interface which would be used to create and send the administrative commands to ODL and later affect the virtual Mininet network as well.

### 6.1.2 Demo Flow

Our demo started by giving a brief description on the work flow of actions that will happen from the moment BECS issues an administrative command until the user’s traffic is steered successfully. Then we setup the BECS environment, ODL and our virtual network by running our custom made Mininet script which instantiates a virtual network with the topology described in 5.2. Last we enabled the web server and the database server in order to load up our visualization interface. There the information of which specific VNF is enabled and when traffic passes through was going to be displayed.

To kick-start the demo, we open the BECS user interface and check the list of pre-configured VNFs and their IP addresses. Furthermore we have available a list of all the users participating in the network. In the meantime we issue the pingall() command to our Mininet client which results into all the workstations in the virtual Mininet network to start pinging each other. This step is very important in order for the ODL controller to be updated regarding the topology situation in the network, since the OpenFlow switches will relay the ping packets to ODL due to lack of knowledge about what to do with them.

At that point we create a call from BECS to ODL by choosing our newly created ODL element manager. As an input, we provide the IP of the user that we want to pass its traffic through a specific set of VNFs. Then we provide the IP addresses of those VNFs in a specific order. This information is send through the ODL element manager from BECS to our ODL FlowController module. After translation of this information and after the TopologyManager module queries the network for the current topology, the BECS commands are translated into a set of OpenFlow commands that are installed to the Command and Control switch in our virtual network. After this step everything is setup and we are ready to perform the service function chaining action in the virtual network.

Next we open up a console window in the specific workstation, which represents the user that we configured the SFC path through Mininet previously. From there we initiate a ping message to a server outside the virtual network, which in this case for ease of use we choose the same webserver as the one we are using to visualize the status of the VNFs in our network. The ping packet will start from the user’s workstation and then reach the user gateway switch where the user is connected to. Then based on the OpenFlow rules that ODL has installed in this switch, the switch will forward the packet to the Command and Control switch. At this point the SFC process will start.
The Command and Control switch has the necessary rules installed into it in order to perform service chaining of this packet. First it will check the first VNF that was provided in BECS’s administrative command and will forward the packet there. The VNF will receive the packet and the internal counter will increase by one. Then the packet will be bounced back to the Command and Control switch. The moment the internal counter changes, a call to the database server will be made updating the state and the counter of that specific VNF. The result would be that the viewers will be able to see a change in the visualization website concerning this specific VNF and the updated number of packets that the VNF has received.

The Command and Control switch will then send the packet to the next specified VNF, if any has been specified, until the whole service chain has been satisfied. Then the packet would be transferred to the internet gateway switch which will release the packet on the internet. After the webserver replies to the ping request from the user a response will be issued. This reply will access the internet gateway switch and then the Command and Control switch. After that, the reverse SFC action will be initiated in order for the response to be steered through a VNF chain in reverse order than the one specified from BECS. This happens since this packet has been issued from outside the network and as destination it has a user who is present in the virtual network.

Throughout the whole process the packet is bounced back and forth between different VNFs and the viewers of the demo can transparently follow the process by checking the VNF visualization web interface to keep up with which VNFs are accessed and in which order. As a last part of the demonstration we issue another command from BECS in order to change the VNF chain for this specific user so this change can be mirrored in the visualization web interface. For example, we could reduce the amount of VNFs in the chain from three to one in order to make the change easier to spot.

In that case only one VNF will keep being updated in the user interface. Then we conclude the demo by issuing a last command which will include SFC information for the other user in the network using all three available VNFs in the network. The purpose of this action is to show the ability of a VFN to be accessed at the same time by two different users and the traffic to still follow different paths afterwards. For this to happen we leverage the intelligence that ODL has added to the OpenFlow Command and Control switch by installing the necessary rules based on BECS’s commands. It’s also necessary to mention that the VNF that is used in both chains will increase its counter by two packets for every ping request outgoing from the two users. That will result in a better and more easy way for the viewers to grasp the concept of different users using the same VNF simultaneously.

6.2 Result Analysis

6.2.1 Goal Assessment

The results of our demonstration were positive, since we were able to fulfill most of the requirements that we set for our prototype solution in the beginning of this thesis work.

In more detail, we created a prototype solution which makes us able to demonstrate a SFC domain triggered and orchestrated by BECS and controlled by ODL using VNFs, from which user traffic is steered through. Also we successfully added new functionality to BECS and ODL so they can communicate with each other and result into BECS orchestrating the SDN enabled network through ODL.
Moreover, SFC has been achieved without any restrictions regarding the order of the VNFs used in the service chain. Ultimately, we created a demonstration framework and a developed work-flow which was used to present and visualize the SFC realization to the public.

Last but not least, we were able to steer user specific traffic in the legacy network correctly according to the commands that BECS has issued. That is how we were able to show that traffic originated from a specific user was accessing the right VNFs and in the right order. Therefore fulfilling one of the most important requirements we have set initially.

### 6.2.2 Issues

However, not all goals have been achieved by this demo framework to their full extend. Scalability of users and VNFs still has to be handled in a better way. Our current topology might be able to handle an increasing number of users and VNFs but if we start thinking about topologies with million users and thousand VNFs we understand that a more scalable topology has to be developed. That could be done using multiple Command and Control switches that can relay traffic and ultimately the work load between each other. Based on our result evaluation about scalability, we proposed the following solution. This new topology aims towards increased scalability.

As illustrated in Figure 6.1, we have a user switch in the bottom left corner where all the hosts are directly connected. This region of the network is tagged as CPE farm and all the traffic from it is terminated in a tunnel gateway. This can be another host that works as an endpoint to the tunnel we create towards the control center where the CC switch resides. The tunnel can be anything from a Generic Routing Encapsulation (GRE) tunnel, VXLAN tunnel to a traditional Virtual Private Network (VPN) tunnel.

![Figure 6.1: Scenario 3 Topology](image-url)
Chapter 6. Results and Evaluation

The purpose of the tunnel insertion here is that in a real network the user switches and the CC switch are not going to be directly connected in all cases. Instead there are going to be many layer three devices, such as routers, between them. Since we want to introduce a more scalable version of the network topology, we intend to keep the number of flows in the Command and Control switch as low as possible with this solution. Therefore we introduce a form of tagging to each specific user traffic.

The tagging procedure can be either as simple as setting something unique in the ethernet header in order to describe which service chain the traffic has to go through. Alternatively, it could be as complicated as the introduction of a new header that will encapsulate the original packet as described in [33]. We wanted to keep our approach as simple as possible and that is why we selected VLAN tagging as our way to mark the packets in a unique way based on the VNF chain they will have to get routed through.

Assuming we have "n" number of VNFs then the maximum number of service chains that can be created by those VNFs is equal to "n!". That means that we can assign a specific VLAN tag number to represent a specific VNF chain and then tag the packets that enter the user switch accordingly. That way we ensure that the service chaining process will succeed and at the same time we dramatically reduce the number of flows based on the following formula.

\[ c + (n + 1) \sum_{k=0}^{n} \frac{n!}{(n-k)!} \]

In the formula "c" is representing the number of hosts in the network and "n" is the number of VNFs. This equation can give us a very accurate projection concerning the number of flows that the Command and Control switch needs to have in order to fully support all possible service chaining combinations. Ultimately, the number of flows will be considerably low but this solution still has to be developed and then tested in a real network in order to conclude its feasibility.
7. Conclusions and Future Work

7.1 Conclusions

In this chapter we finalize our work by presenting our conclusions gathered from this thesis project. The demonstration was a success, fulfilling nearly all the goals presented in the initial scope of this thesis. Throughout the process we encountered problems with the current state of the art software regarding the ODL controller. ODL was and still is an ongoing open source project that is far away from being complete.

That resulted into some malfunctioning modules by the time we were developing our prototype and therefore we were prompted to create our own topology manager module in order to monitor the topology situation in the network. Of course that resulted into making a very customized topology awareness module that helped us a lot in the development of the other ODL module we needed for our prototype.

One final remark we would like to make is regarding the pending standardization of SFC in general. Since this standardization is not complete yet many different ideas will emerge and ultimately will result into a very well defined concept. Of course that can take time and the best we can do meanwhile is to try developing customized prototypes that work as solutions to very specific problems. But the question still remains regarding what will happen to these solutions if and when SFC is actually standardized and available. Maybe all previous solutions would be obsolete and have to be re-designed based on the new created standardization. That can also require time and effort so it would be interesting to see what decisions will be made about this situation in the future.

7.2 Potential Environmental and Social Impact

As discussed in previous chapters, our prototype solution is aiming on the realization of SFC in a legacy network. While this can be a very interesting subject regarding its technical perspective, there are more aspects to consider. NFV and SDN principles can have obvious environmental and social impact. Therefore, since our prototype is based on those principles, its impact on both environment and ethics has to be estimated.

During the COP21 conference in Paris, the issue of how humans impact the environment in a global level was in focus. The carbon footprint of planet earth has been estimated and presented during the conference [34]. One of the main conclusions was that since 1950 the carbon oxide emissions have been increased dramatically. That of course had an impact in the environment by raising the average temperature all around the world by one degree Celsius. Based on projections and statistical analysis,
it has been estimated that if the global carbon footprint keeps increasing at the same pace, by 2050 the average global temperature will increase by one more degree. That of course can have a critical impact in the environment overall. As a result, it has been decided from the COP21 conference on a United Nations level that all countries over the world should contribute into reducing the global carbon footprint by fifty percent until 2050.

There are many ways to conform with this decision and computer science should be able to contribute on that goal. More specifically, new technological principles such as SDN and NFV are in process of minimizing the dependency that a network infrastructure has on hardware equipment. NFV aims in the virtualization of network functions such as firewall, BRAS and many other equipment that are currently used in the networking world. That concept by default, can benefit the environment by reducing the size of big data centers dramatically. Vendors will no longer need to allocate special buildings to house their networking hardware. Instead they will be able to route their traffic through VNF farms located in the cloud. Therefore, since the amount of hardware device will reduce, effectively the energy consumption for powering on and cooling the hardware devices will be minimized also.

This approach show high prospects of sustainability since the reduction in actual hardware will enable vendors to re-invest their profits into more environmental friendly infrastructures. Moreover, the economical impact that the virtualization of network functions will bring has still be to evaluated. Whenever a big change regarding technology happens, it affects the overall market and the amount of available positions in each technological field. People will need to adapt in those new needs of the market in order to survive this new technological shift. Eventually, that can re-distribute the available market share between different companies and ultimately result into new available positions in new exiting fields for many individuals.

Our prototype uses the NFV principles mentioned above in combination with SDN in order to enable seamless service function chaining. If we succeed into providing a sustainable solution that can support a large amount of users, then it can alter the way we think about transferring and accessing information. At the same time, since we are talking about SFC and the handling of sensitive user information, one should not forget to address the social and ethical impact this solution. Whenever a technological advancement is made, its impact in society is not completely clear. Instead, the main focus seems to be on the new features and their application in various aspects of everyday life. But its not always clear or easy to foresee the pitfalls and possible ethical violations that might occur with the use of this new technological discovery. The least anybody could do in these situations is to address the ethical and social impact of the end product during the development phase.

That is exactly how we proceeded in this thesis work. Initially our main interest was to enable seamless traffic steering of user data between various VNFs before and after accessing the rest of the internet. When we achieved this main goal, our next step was to address the sustainability of the solution. In order to provide a sustainable solution, we realized we needed to make it scalable regarding the number of users that this prototype could support. Also we start thinking that since the user data will be steered through many VNFs and switches, the integrity and confidentiality of the data should be a main requirement. That led to a suggested second version of our prototype which utilizes security tunnels while transferring the information between different switches in the network or different VNFs. That way we can prevent any ethical pitfalls concerning user data tampering and lack of confidentiality. We include our vision of our prototype’s future version in the following subsection of this chapter.
Chapter 7. Conclusions and Future Work

7.3 Future Work

Our prototype works well with a limited number of users and VNFs. But if we want to integrate this solution in a real time network, we would have to address the scalability factor in a more sufficient way. Therefore we came up with the alternative topology scenario as presented in 6.1.

And while this solution appears to been very reliable, another issue still remained. The VLAN tags that we add to the packet so that the CC switch can be able to know which VNFs to forward the packet to, is going to be lost after entering a layer three devices such as a router. That happens because the router will erase and recreate the ethernet header of the packet which contains the MAC source and destination addresses. Most importantly, the VLAN tag will be lost since it is contained in the ethernet header as well.

That is the reason why we introduce some form of tunneling in our prototype as shown in 6.1. The tunneling offers security for traffic that goes through it. But most importantly, it provides a path where the packets can transport through the network without their ethernet header stripped or changed. This happens because of the way tunneling works. It provides an encapsulation to the original packet by adding an extra header and trailer to the original packet, which then is considered as payload.

This encapsulation takes place at the tunnel gateway directly after the user switch. At the user switch, the VLAN tagging takes place based on the VLAN tag of the VNF chain that the host has to route its traffic through. Then the packet is safely transported to the tunnel gateway and through that to the opposite side of the tunnel. Then, the packet is de-capsulated and returned to its original form with its VLAN tag still intact. The CC switch is then receiving the packet and checks the VLAN tag that the packet has on its ethernet header.

The next step is to send the packet to the first VNF of the service chain that VLAN tag is corresponding to. During this transport, we also have to consider the integrity of the packet since a layer three device can exist on the way to the service. That is why we have tunnels going directly to each NFV we support in the network. That way we ensure the integrity of the packet and moreover we introduce new flexibility since we don’t need the VNF to be directly connected to the CC switch anymore. Instead, the VNFs can be in remote server relatively to our SDN network. Our tunneling technique takes care of the connectivity issue and thus we are able to route traffic to distant VNF farms.

It is very important for the same VLAN tagging process to take place on each packet on their way back from the internet as well. That happens in the gateway switch and is based on the VNF chain that the packet needs to go through. The packet enters a tunnel right after the tagging process in the gateway switch and goes directly onto the command and control switch. From there the exact same process takes place as before. The difference now is that since the destination address is a host of the network, that means the packets come back from the internet. Therefore, the service chain has to be accessed in the reverse order. Finally the packet is transferred to the correct user switch where it is forwarded to the host it was intended to.

This scenario seem to provide a very flexible, robust and scalable network environment regarding the number of hosts and VNFs in the network. But still development and testing of this new solution has to be conducted, something that was outside of the time limitations we had when developing our initial prototype. This resulted into leaving this new topology proposal as future work.
Bibliography


