Rodolfo Vieira Valentim

An Architecture for End-to-End Network Slicing in Multiple Data Centers using Tableless Source Routing

Brazil

2020

Rodolfo Vieira Valentim

An Architecture for End-to-End Network Slicing in Multiple Data Centers using Tableless Source Routing

Dissertação de Mestrado apresentado ao Departamento de Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Mestre em Informática.

Universidade Federal do Espírito Santo (Ufes)

Centro Tecnológico (CT)

Programa de Pós-Graduação em Informática (PPGI)

Supervisor: Prof. Dr. Rodolfo da Silva Villaça

Co-supervisor: Prof[a]. Dr[a]. Cristina Klippel Dominicini

Brazil

2020

Ficha catalográfica disponibilizada pelo Sistema Integrado de Bibliotecas - SIBI/UFES e elaborada pelo autor

---

Valentim, Rodolfo Vieira, 1992-

V155a An Architecture for End-to-End Network Slicing in Multiple Data Centers using Tableless Source Routing / Rodolfo Vieira Valentim. - 2020.

81 f. : il.

Orientador: Rodolfo da Silva Villaça.

Coorientadora: Cristina Klippel Dominicini.

Tese (Mestrado em Informática) - Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Computação em nuvem. 2. Centros de processamento de dados. I. Villaça, Rodolfo da Silva. II. Dominicini, Cristina Klippel. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

AN ARCHITECTURE FOR SERVICE FUNCTION CHAINING IN MULTIPLE DATA CENTERS USING TABLELESS SOURCE ROUTING

Rodolfo Vieira Valentim

Dissertação submetida ao Programa de Pós-Graduação em Informática da Universidade Federal do Espírito Santo como requisito parcial para a obtenção do grau de Mestre em Informática.

Aprovada em 30 de julho de 2020:

Prof. Dr. Rodolfo da Silva Villaca
Orientador – DTI/CT/UFES

Profª. Drª. Cristina Klippel Dominicini
Coorientadora – IFES Campus Serra

Prof. Dr. Magnos Martinello
Membro Interno – DI/CT/UFES

Prof. Dr. Rafael Pasquini
Membro Externo – FACOM/UFU

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
Vitória-ES, 30 de julho de 2020.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

PROTOCOLO DE ASSINATURA

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
RODOLFO DA SILVA VILLACA - SIAPE 2650719
Departamento de Tecnologia Industrial - DTI/CT
Em 30/07/2020 às 11:35

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/43428?tipoArquivo=O

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

PROTOCOLO DE ASSINATURA

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
MAGNOS MARTINELLO - SIAPE 1669875
Departamento de Informática - DI/CT
Em 31/07/2020 às 09:52

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/43722?tipoArquivo=O

Acknowledgements

Agradeço a Deus por me dar sabedoria e força para passar por essa etapa da minha vida.

Agradeço aos meus pais por todo amor, carinho e dedicação durante todos esses anos.

Agradeço a minha irmã por ser amiga, carinhosa e por sempre acreditar em mim.

Agradeço a Natasha por ser uma companheira que não mede esforços para me ajudar e por garantir minha sanidade mental nesse período conturbado.

Agradeço a todos meus amigos do NERDS pelas conversas, pelas partidas acaloradas de Bomberman e pela pronta ajuda sempre que necessária.

Agradeço aos meus orientadores pela imensa paciência que tiveram comigo e pela essencial ajuda que recebi durante esses anos de estudo.

Finalmente, agradeço a todos meus amigos que de um jeito ou de outro estiveram ao meu lado durante essa caminhada.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Abstract

Telecommunication Service Providers (TSP) need to offer a wide range of services to their customers. The network functions virtualization (NFV), software-defined networks (SDN), and network slicing paradigms facilitate the deployment of these services by managing virtualized resources in a software-driven way. These paradigms have gained traction with the growth of new services that have different business demands, because their composition enhance programmability and innovation on top of COTS equipment and open-source tools. However, there are cases when these services are composed by functions hosted in geographically distant data centers (or clouds) in different domains. Telecommunication networks are almost ubiquitous, but multiple TSPs provide access to these networks, and each TSP has a footprint focused on a specific region. This market fragmentation makes it challenging to deploy cost-effective network services spanning multiple clouds and few works in the literature focus on the mechanisms for implementing the stitching of network slices from multiples service providers or infrastructures.

In this context, this work presents a solution for implementing an end-to-end network slicing solution using tableless source routing. We propose the use of tableless source routing to specify a set of physical links, sub-networks, and network functions using a single piece of information without stacking protocol headers. At the same time, it improves scalability in the network core by reducing the number of forwarding states. Our proposal of the architecture has two main concerns. First, it needs to enable tableless source routing in data centers in a flexible, easy, and programmatic way, keeping compatibility with legacy applications hosted in these data centers. The second task and our major contribution is to perform the stitching of slicing segments in each data center in a way that abstracts from operators the huge amount of work required to perform these tasks.

As a proof-of-concept, we implemented prototypes of the proposed solution with cutting edge cloud technologies, such as OpenFlow, OpenStack, and Open vSwitch. The results of functional and performance tests showed that the proposed solution enables end-to-end network slicing in a low-cost, efficient and flexible manner. Moreover, our proposal provides mechanisms to the NFV orchestrator that allow fine-grained traffic engineering decisions to optimize the selection of network paths and connections between domains.

Keywords: NFV, VNF, SFC, *OpenStack*, Network Slicing, Cloud Computing.

Resumo

Os Provedores de Serviços de Telecomunicações (TSP) precisam oferecer uma ampla gama de serviços aos seus clientes. A Virtualização de Funções de Rede (do inglês Network Function Virtualization - NFV), Redes Definidas por Software (do inglês Software Defined Networks - SDN) e paradigmas de fatiamento de redes (do inglês Network Slicing) facilitam a implantação desses serviços com o gerenciamento de recursos virtualizados orientada por software. Esses paradigmas ganharam tração com o crescimento de novos serviços que possuem requisitos de negócios diferentes, uma vez que sua composição aprimora a programabilidade e a inovação em cima de equipamentos de prateleira e ferramentas de código aberto. No entanto, há casos em que esses serviços são compostos por funções hospedadas em data centers (ou nuvens) distantes geograficamente em domínios diferentes. As redes de telecomunicação são quase onipresentes, mas vários TSPs fornecem acesso a essas redes, e cada TSP tem uma pegada focada em uma região específica. Essa fragmentação do mercado torna um desafio implantar serviços de rede econômicos abrangendo várias nuvens e poucos trabalhos na literatura enfocam os mecanismos para implementar a costura de fatias de rede de vários provedores de serviços ou infraestruturas.

Nesse contexto, este trabalho apresenta uma solução para a implementação de uma solução de fatiamento de rede fim a fim usando roteamento fonte sem tabelas. Propomos o uso de roteamento na origem sem tabelas de encaminhamento para especificar um conjunto de enlaces físicos, sub-redes e funções de rede usando uma única informação sem empilhar cabeçalhos de protocolo. Ao mesmo tempo em que melhora a escalabilidade no núcleo da rede, reduzindo o número de estados de encaminhamento. Nossa proposta de arquitetura tem duas preocupações principais. Primeiro, ele precisa habilitar o roteamento de fontes sem tabelas em data centers de maneira flexível, fácil e programática, mantendo a compatibilidade com aplicações legadas hospedadas nesses data centers. A segunda tarefa e nossa maior contribuição é realizar a costura de segmentos de fatiamento de rede em cada data center de uma forma que abstraia dos operadores a enorme quantidade de trabalho necessária para realizar essas tarefas.

Como prova de conceito, implementamos protótipos da solução proposta com tecnologias de nuvem modernas, como OpenFlow, OpenStack e Open vSwitch. Os resultados dos testes funcionais e de desempenho mostraram que a solução proposta permite o fatiamento da rede fim-a-fim de forma econômica, eficiente e flexível. Além disso, nossa proposta fornece mecanismos para o orquestrador NFV que permitem decisões de engenharia de tráfego refinadas para otimizar a seleção de caminhos de rede e conexões entre domínios.

Keywords: NFV, VNF, SFC, *OpenStack*, Fatiamento de Rede, Computação em Nuvem.

List of Figures

List of Tables

List of abbreviations and acronyms

IoT	Internet das coisas, do inglês *Internet of Things*
CAPEX	Gastos de capital, do inglês *Capital Expenditure*
OPEX	Gastos operacionais, do inglês *Operational Expenditure*
NFV	Virtualização de Funções de Rede, do inglês *Network Functions Virtualization*
NAT	Tradução de Endereço de Rede, do inglês *Network Address Translation*
SDN	Rede Definidas por Software, do inglês *Software Defined Networks*
TSP	Provedor de Serviços de Rede, do inglês *Telecommunication Service Provider*
VIM	Gerenciador de Infraestrutura Virtualizada prevista no *ETSI Framework*, do inglês *Virtualized Infrastructure Manager*
SFC	Encadeamento de Funções de Serviço, do inglês *Service Function Chaining*
NFV MANO	Gerenciamento e Orquestração de Funções de Rede Virtualizadas, do inglês *NFV Management and Orquestration*
VNFI	Instância de Função de Rede Virtual, do inglês *Virtual Network Function Instance*

Contents

1 Introduction

The advent of the Network Functions Virtualization (NFV) paradigm has enabled telecommunications service providers to migrate part of the network functions from specialized equipment to Virtualized Network Functions (VNF), instantiated on commercial-of-the-shelf (COTS) servers located in public or private clouds (MIJUMBI et al., 2016). Concomitantly, the popularization of Software-defined Networks (SDN) (KREUTZ et al., 2015) allowed greater programmability of network elements compared to traditional approaches. SDN implies a separation between the control plane and the data plane. In addition, network slicing (NGMN, 2016) emerges as a general concept for partitioning network resources for specific users and a way to guarantee performance for end-to-end services. These technologies have gained traction with the increasing growth of new applications that have different business demands, because their composition enhance programmability and innovation on top of COTS equipment and open-source tools.

Figure 1 illustrates this composition with an example of the deployment of two services for a company that has two branch unities. There are three layers used to abstract the provisioning of resources (NGMN, 2016). The Service Provider (SP) sells the *Service Layer*, which is implemented using VNFs from the *Network Slice Instance Layer*. These VNFs are positioned considering the availability of resources and other metrics, such as latency and available bandwidth. Depending on the positioning, the providers need to create a slicing of network resources in the *Resource Layer* to implement the service. SDN enables network slicing by providing a programmable layer for the service-oriented control of network resources. Besides, it centralizes the control via a SDN Controller which, in fact, manages the slices and the network flows.

Scenarios intra-data center need to consider the cloud infrastructure. Cloud infrastructures are pools of computational, storage and network resources and an important asset for Service Providers that can use it to deploy modern services by concatenating virtualized network functions (VNFs) using NFV paradigm. Moreover, cloud computing is used to host several network applications simultaneously which implies in multi-tenancy and in the sharing of physical resources. These characteristics present as a challenge for Service Providers to provision applications with conflicting requirements that share the same infrastructure. Network slicing is fundamental to assure quality of service and the correct performance of the network services.

The deliver of cutting edge applications might require some network flows to transit from edge data centers to other edge or to core data centers. Service Providers are able to use proprietary or third-party computational resources distributed in different

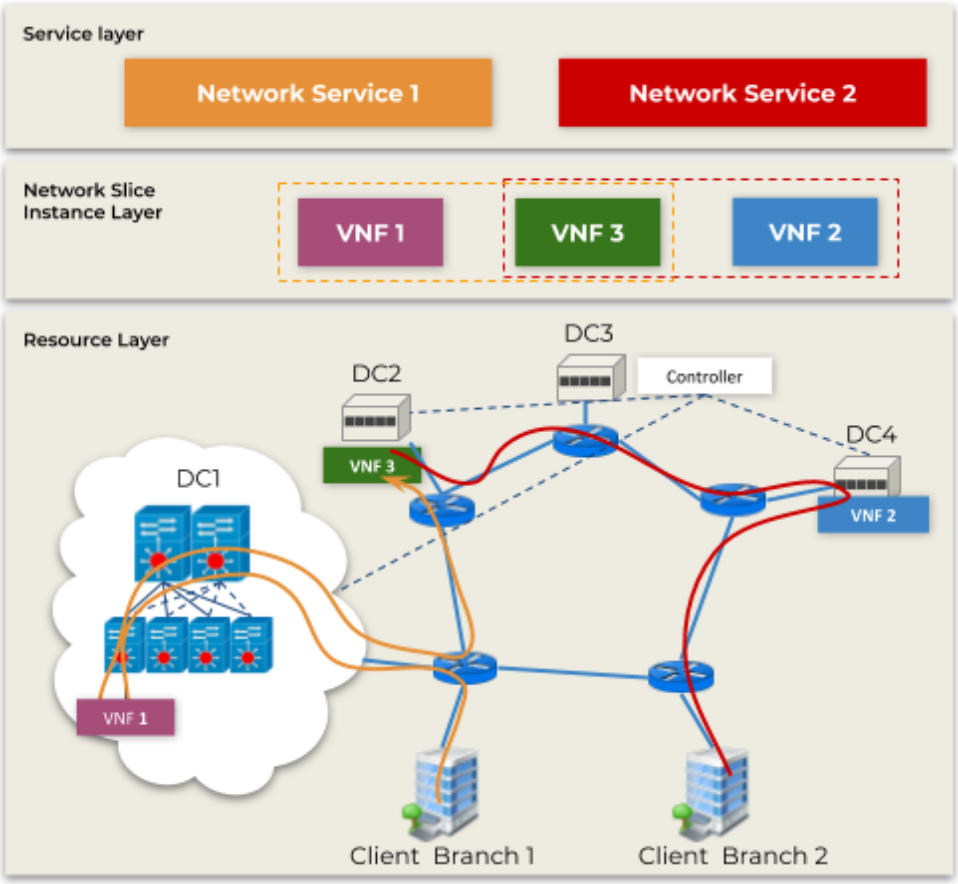


Figure 1 – Modern applications deployed in large and complex geo-distributed networks.

infrastructure and/or geographical domains to perform a dynamic composition of network functions. For example, composing low latency applications using edge data centers at different locations, closer to the users. In these inter-data centers scenarios, the delivery of end-to-end services to users requires additional complexity.

Although several works have tackled this task from the point-of-view of an optimization problem by choosing best routes to steer the flows, less attention has being paid to the methods of how to transfer flows between data centers. Also, the proposal of methods to stitch several network slicing segments deployed in different data centers need to consider the complexity of metro, core, and intercontinental networks that connect different cloud infrastructures.

Many technologies are capable of performing the slicing of network resources in intra-data center scenarios. However, most of these solutions use traditional table-based routing mechanisms, which present problems related to poor scalability and high degree of complexity in the management of network element states (table entries) due to the high transience of flows (JIN et al., 2016). A consequence is the long convergence time required to reconfigure network states. Also, traffic engineering may be restricted to a subset of all available paths in the network, preventing the network orchestrator from selecting

optimized paths to avoid congestion or failures (TSO; JOUET; PEZAROS, 2016).

An alternative to mitigate table-based routing problems is the mechanism of Source Routing (SR), which has gained prominence for mitigating scalability problems related to the management of forwarding tables on networking devices (JYOTHI et al., 2015) and the control plane overhead when compared to traditional forwarding mechanisms (MARTINELLO et al., 2014). Strict Source Routing (SSR) is a variant of Source Routing where the complete path between source and destination of a flow is specified and no deviation of this path is allowed.

SSR facilitates network slicing, because it can specify the slicing of resources at Service Layer, Network Slice Instance Layer and Resource Layer with the same encapsulation. In other words, SSR is capable of specifying a set of physical links, sub-networks and network functions using a single piece of information without stacking header protocols. Inside a data center, the Service Function Chaining (SFC) and the provisioning of physical resources problems can be represented as a network slicing problem and solved using SSR approach.

In inter-data center scenarios, a way to deal with the complexity of their interconnection technologies is to create an overlay network that connects different infrastructures. The point of connection between two infrastructures are gateways that manage these interconnections, abstracting its complexity and speeding up the process of service delivery. These gateways use SSR allied to a L3VPN to build an overlay network that conveniently speed up the setup of inter-data center connections.

Given the presented context, we propose an architecture for end-to-end network slicing in multiples data centers using tableless strict source routing. The proposal of the architecture has two main concerns, first it needs to enable tableless source routing in data centers in a flexible, easy and programmatic way, keeping compatibility with legacy applications hosted in these data centers. The second task and our major contribution is to perform the stitching of slicing segments in each data centers in a way that abstracts from operators huge amount of work required to perform this task.

The next section will further discuss the challenges that evolve around build the proposed solution for an end-to-end architecture for networking slicing using tableless strict source routing.

1.1 Research Problem

This section describes the challenges to build an end-to-end architecture to network slicing in multiple data centers using tableless strict source routing.

1.1.1 Challenges in intra-data center scenarios

A network slice consists of interconnected Virtualized Network Functions (VNFs), which host one or more applications (DIMOLITSAS et al., 2020). This simple definition of network slicing allows to use the concepts of Service Function Chaining to model a solution for network slicing inside data centers. We propose the use of strict source routing arguing that this method is capable of fulfill the requirements of resource slicing.

However, we need to understand that the context of solutions for SFC in cloud infrastructures is dominated by a single solution. The main open-source tools that implement SFC, such as `OpenDayLight` (FOUNDATION, 2020c), `ONOS` (FOUNDATION, 2020a), `OPNFV` (FOUNDATION, 2020b), and `OpenStack Networking-SFC` (FOUNDATION, 2020), use the architecture proposed by RFC 7665 (PIGNATARO; CARLOS, 2015) and NSH (Network Service Headers) protocol (QUINN; GUICHARD, 2014) for SFC encapsulation.

The NSH protocol, shown in Figure 14, specifies a header that includes two chain identifiers: the first indicates the selected chain and the second indicates the current position in the chain. NSH relies on the underlying transport methods to deliver the packet between elements. Most data center administrators uses tunneling solutions to provide connectivity to the SFC elements and use the NSH as encapsulation to select the next hop in the chain.

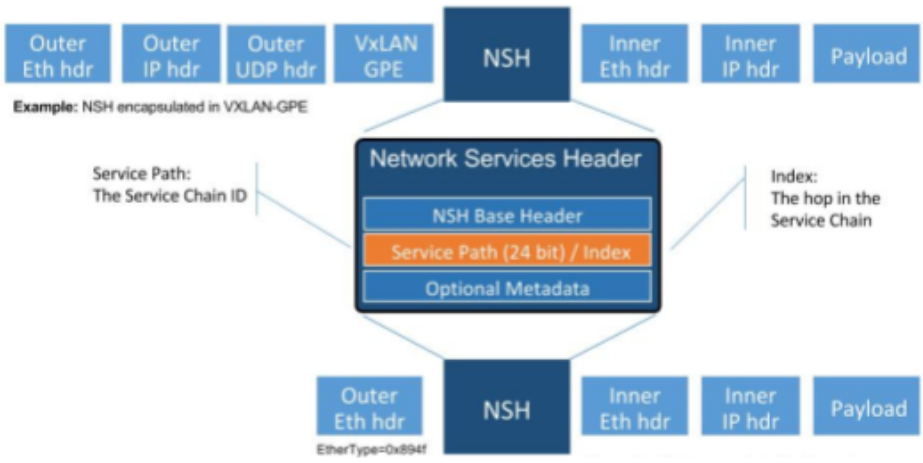


Figure 2 – NSH header.

NSH is the standard approach to implement SFC in data centers and it uses table-based forwarding mechanisms. Forwarding tables can be managed by human interaction or by a control plane. Human-based management is more suitable for small data center networks with little dynamism which lacks robustness. Control plane management requires overhead in the configuration and, in most cases, involve distributed protocols that present relevant convergence time. The use of a control plane is indicated for big data centers

when human management is not feasible.

Data center networks increased in size with the popularization of cloud computing over the years. Capable of hosting thousands of virtual machines and hundreds of tenants, each tenant requires traffic isolation from other clients. In order to implement network resources slicing the cloud administrator, often, uses a tunneling method such as VxLAN. However, these techniques do not scale very well without the adoption of a well-designed control plane. The control plane is complex to set up and incur high management overhead. Also, there is an overhead required to manage flow entries in a data center with hundreds of nodes and thousands of SFCs. Furthermore, when we consider the dynamic characteristics of the NFV traffic, the sequence of Service Functions (SF) in a chain may change during its life cycle requiring live migrations of SFCs. Also, the traffic engineering of the data center may need to migrate paths and SFs to guarantee optimal resource usage (DOMINICINI et al., 2020).

In fact, cloud administrators need to use more than one protocol and control plane towards building a complete network slicing solution. It requires SFC for virtual network functions, IP/VxLAN protocols for network isolation and ICMP or other link protocols to guarantee the link usage. The complexity to manage all these different technologies simultaneously postpone the adoption of NFV as an industry standard.

The advantages of adopting Source Routing are not limited to the reduction of management overhead. We claim that SSR changes the SFC paradigm and creates the possibility to enable a network slicing solution that unifies the traffic steering, traffic engineering and network slicing. In other words, we propose a solution that covers all three layers of abstraction as shown in Figure 3.

However, the adoption of tableless Strict Source Routing as transport mechanism requires some changes in the control plane of DCNs. Currently, there are not any commercial solutions available. For research purposes the options are to customize software switches and use them in the data plane to add support for Source Routing, or to implement hardware switches using prototyping boards like NetFPGAs. Both solutions are time-consuming and hard to maintain. There is a clear need of solutions for DCNs to enable Source Routing mechanisms that are easy to deploy and maintain compatibility with legacy technologies.

1.1.2 Challenges in inter-data center scenarios

Telecommunications networks are almost ubiquitous, however, market fragmentation leads to each Telecommunications Service Provider (TSP) has a footprint focused on a specific region. This fragmentation makes it difficult to deploy cost-effective infrastructure services, such as virtual connectivity or compute resources, spanning multiple countries as

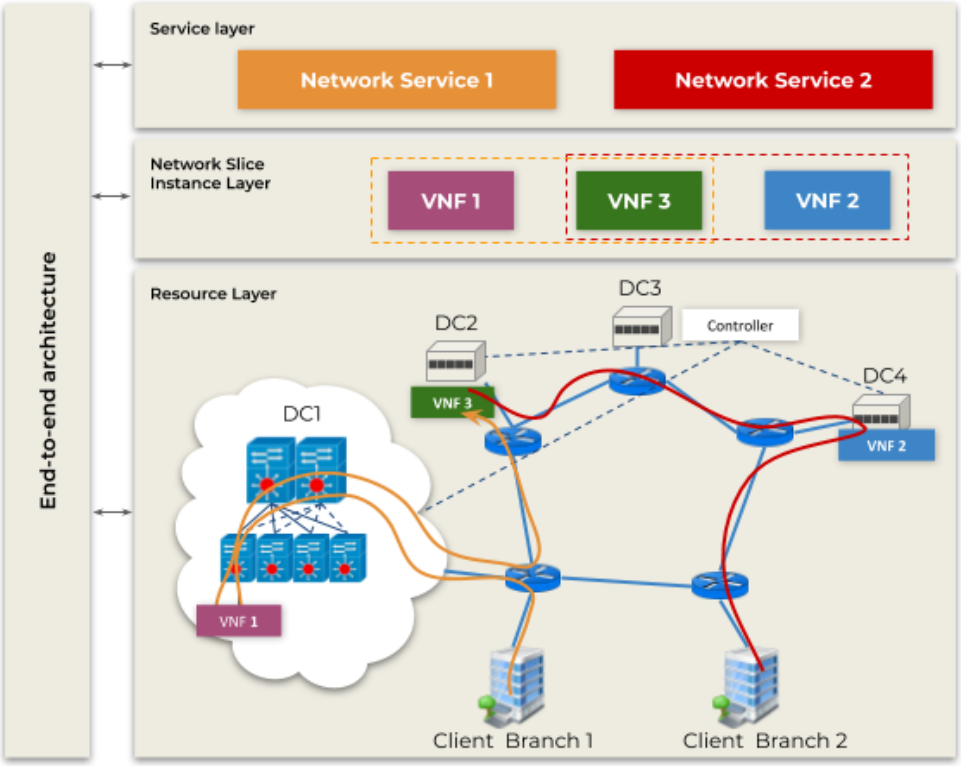


Figure 3 – SSR as a multiple layer solution.

no single operator has a big enough footprint (Hermiyanty, Wandira Ayu Bertin, 2017). Also, combined with geo-distributed data centers, service providers could place virtual network functions closer to customers, reducing latency and improving reliability (PEI et al., 2019).

As it was already discussed, we can model these sequences of virtual network functions both as a network slicing and as a SFC. Moreover, the most important aspect to analyse when we have VNFs distributed in data centers located in different places is the mechanisms that perform the stitching of every segment of SFC (or slices) towards building an end-to-end service.

The NFV architecture provides the means to deploy network services spawning over multiple geographically distributed domains, which can be advantageous to provide services with better quality in a broader area. For instance, a company with multiple branch offices covering entire countries can hire only one service provider instead of having to make several contracts depending on the availability of services in the region. On the other hand, inter-operator collaboration tools are very limited, which makes service deployments and provisioning very time-consuming. Even internally, a single-operator can also experience some troubles when dealing with technology fragmentation between data centers. Different data centers in the same administrative domain can use different equipment from different vendors with different technologies.

In general, to implement an NFV service two stages are required: (i) placement,

i.e. where to place a virtual function in order to optimize resource usage and to attend requirements; (ii) SFC, i.e. to decide how to steer the traffic flows across an ordered set of VNFs that compose the service. In inter-data center scenarios, both stages require additional considerations, since there are more variables such as: closeness to the customer, optimization of a path between source and destination, WAN access optimization, and others (DOMINICINI et al., 2020).

Some researches have already addressed the problem as an optimization problem. While most researches focus on how to optimize the placement and routing, the studies about the design and implementation of a mechanism to reduce the time and effort needed to implement such scenarios are scarce. As the mechanisms of placement do not change considerably in inter-data center scenarios, our focus will be onto the SFC stage.

The ETSI (European Telecommunications Standards Institute) proposed an architecture for the NFV control plane managing multiple data centers (MAHMOODI et al., 2017). Although this architecture contemplates several aspects of this problem, our approach considers the use of SSR combined with L3VPNs and we assign to the transport network the responsibility to deliver flows to gateway that forward these flows to other cloud infrastructures. Our architectures differs from ETSI, since we rely on a single solution for SFC inside and between data centers.

Figure 4 illustrates the view of the Service Providers regarding the data center interconnections (on the left) and how we intend to abstract these connections by the use of a virtual private network (on the right). Traditionally, the service providers need to manage the state of forwarders in the path between infrastructures in order to maintain the connectivity. Since this task is complex and time consuming, we propose the use of gateways. The gateways handle the interconnections between data centers and translate the encapsulation methods.

Figure 4 – Abstraction of inter-data center connections.

1.2 Objectives

In order to tackle these research problems, this work has the following general objectives:

- To develop an architecture that enables end-to-end network slicing between service functions located in different data centers in an efficient and flexible way;
- To develop an architecture that enables data centers to support tableless Strict Source Routing mechanisms as transport encapsulation and as network slicing method;
- To implement both control and data planes in order to enable network slicing deployment into data center network infrastructures.

The scope of this work is composed of services over single and multiple data centers, that can be in one or multiple Infrastructure Domains. The only restriction is that between two Infrastructure Domains must exist connectivity provided by the Internet, data center network itself or any third-party WAN solution.

These objectives, when achieved, should result in a fully functional prototype using industry-standard technology that will be used as a proof-of-concept for the testing scenarios.

1.3 Proposal and contributions

This work investigates the possibility to use tableless strict source routing architecture in data center networks in order to facilitate end-to-end network slicing in a context of multiple data centers.

Given this scenario, we propose an architecture for end-to-end network slicing that covers both control plane and data plane, adding new functionalities to industry standard infrastructures and enabling strict source routing as a transport method. The hypothesis is that we can use a single mechanism to perform full stack network slicing by changing the forwarding mechanism. This will not just make a data center network more manageable, but also allow fine-grained (flow level) traffic engineering, improve scalability in the network core and optimize WAN access. The major contributions of this work are:

- We developed an architecture of services to enable data centers to host end-to-end NFV services;
- We implemented a prototype using OpenStack Cloud Framework to prove the applicability in cutting edge cloud technologies;

- We developed SDN controllers to enable multiple source routing protocols in the data center network;
- We developed an control plane to collect topological information, deploy flow entries, and manage services' life cycle;
- We developed a set of Ansible scripts that automate the task to deploy SDN Controllers, modify software switch configurations and establish inter-data center connections;
- We tested our solution using RNS-based, list-based and MPLS-based source routing;
- We developed a gateway mechanism that virtualizes the WAN and integrates it to the source routing mechanism. As a result, we abstract the complexity to orchestrate the stitching of slice segments allowing traffic engineering can be used and on-demand data center connections can be activated using a simple Internet access;

In summary, this work is a first step to enable researchers and service providers to build network slicing solutions using Strict Source Routing networks.

This work is structured as follows: in Chapter 2, we present some background subjects and related works. Chapter 3 describes the architecture proposed. Chapter 4 proposes and implements network slicing and stitching using SSR-based forwarding. Then, Chapter 5 evaluates and discusses the proposal and the benefits of using it. Finally, we outline the conclusions and future works in Chapter 6.

2 Background and Related Work

This chapter will present the main concepts related to the research problem and discuss the related works.

2.1 Cloud Computing

According to (ZHANG; CHENG; BOUTABA, 2010), cloud computing is a model in which the general-purpose resources (compute, storage and others) can be allocated or freed on-demand via WAN access. in (MELL; GRANCE et al., 2011) a list of the desired characteristics of a cloud infrastructure can be found:

- **On demand services:** a user should be capable of allocating and release resources or services without the need for human interaction;
- **Broad network access:** the services should be available over the Internet or an intranet;
- **Resource repository:** The resources are allocated dynamically for multiple tenants using several physical and virtual resources;
- **Fast elasticity:** should be allowed to applications to increment the number of resources allocated, sometimes automatically and on-demand;
- **Measured services:** Cloud systems automatically control and optimize resource usage by leveraging a metering capability at some level of abstraction, appropriate to the type of service.

In the following sections some relevant aspects of this framework are detailed.

2.1.1 OpenStack

One of the major cloud computing framework is OpenStack, that deserves special interest since it is an open-source project with a very active community, which, usually, represents a very high level of bug reports and solutions. OpenStack software controls large pools of computing, storage, and networking resources throughout a data center managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure (FOUNDATION, 2017).

OpenStack supports Linux environments (Ubuntu, OpenSUSE e CentOS) and adopts a modular architecture where each module has its specific roles. Figure 5 shows how the components interact. The complexity involved in the process of communication is clear. There is an evident security concern and therefore all services authenticate through an identity service (Identity) and the individual services interact with each other through public APIs, except when privileged administrator commands are required.

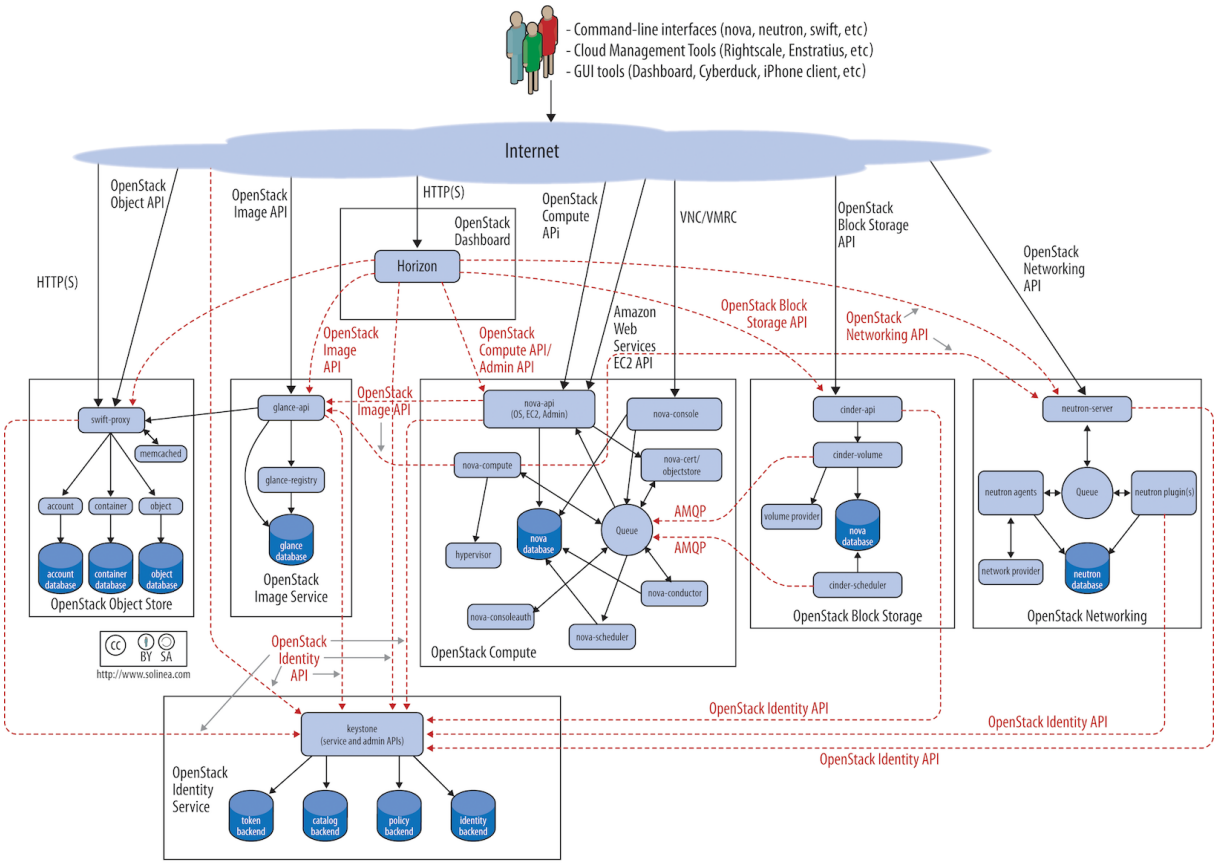


Figure 5 – OpenStack logical architecture. (FOUNDATION, 2017)

2.1.2 OpenStack Networking

OpenStack can enable access to a physical networks through Open vSwitch and Linux Bridges. Figure 6 shows how virtual switches scheme on a computing node looks like.

Virtual machines have virtual Network Interface Cards (vNICs) that act in the same way as physical interfaces. As they are simulations of physical interfaces, virtual interfaces are only capable of receiving Ethernet frames. A tap is a piece of software that tells Linux Bridge to forward Ethernet frames to them as they were received. A tap is a way that a hypervisor, like KVM and Xen, implement a vNIC. An Ethernet frame intended for a vNIC is received by the host's operating system.

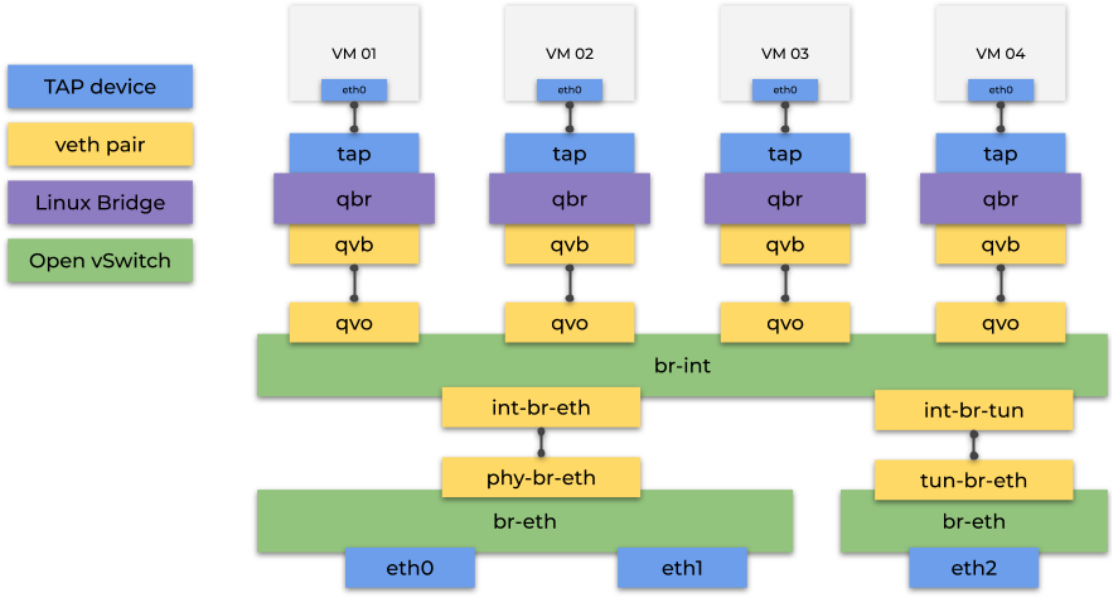


Figure 6 – Architecture used by OpenStack when configured with OpenvSwitch for L2 mechanism.

The `qvb-qvo` veth pair is a directly connected pair where everything that one receives the other also receives. A veth pair is the virtualization of a network cable. The Linux Bridge acts with a Layer 2 switch. It is possible to connect several interfaces to Linux Bridge and the bridge will use a table of MACs to switch the packets, as well as a physical switch.

Open vSwitch bridges can behave like a software switch too. However, there is a crucial difference with Linux Bridge and that justifies its use: the ability to redirect traffic based on rules that can either be installed locally or remotely using a network controller. In a compute node, two Open vSwitch bridges are installed and each one has its specific role in routing, which are explained as follows:

- **br-int**: integration bridge, where all virtual machines allocated on a node are connected. Responsible for isolating the virtual machines' vNICs and the host's physical interfaces;
- **br-eth**: responsible for giving access to the physical network.
- **br-tun**: responsible for giving access to the tunneled network. Used when the cloud administrator needs to create virtual networks for tenants.

2.2 Network Function Virtualization

NFV (Network Function Virtualization) is a new concept, used to represent the virtualization of network functions that used to be executed by an expensive single purpose device to COTS (Commercial off-the-shelf) equipment. The hypotheses are that the virtualization will incur lower CAPital EXpenditures (CAPEX) and OPerating EXpenditures (OPEX).

NFV initiative was born in the industry and, subsequently, a standardization group called Network Functions Virtualisation Industry Specification Group (NFV ISG) was created by ETSI. Later, this group released an architecture of services to guide developers and researches as shown in Figure 7. The architecture is composed of three major layers: (i) VNF Layer, (ii) NFVI Layer and (iii) MANO Layer

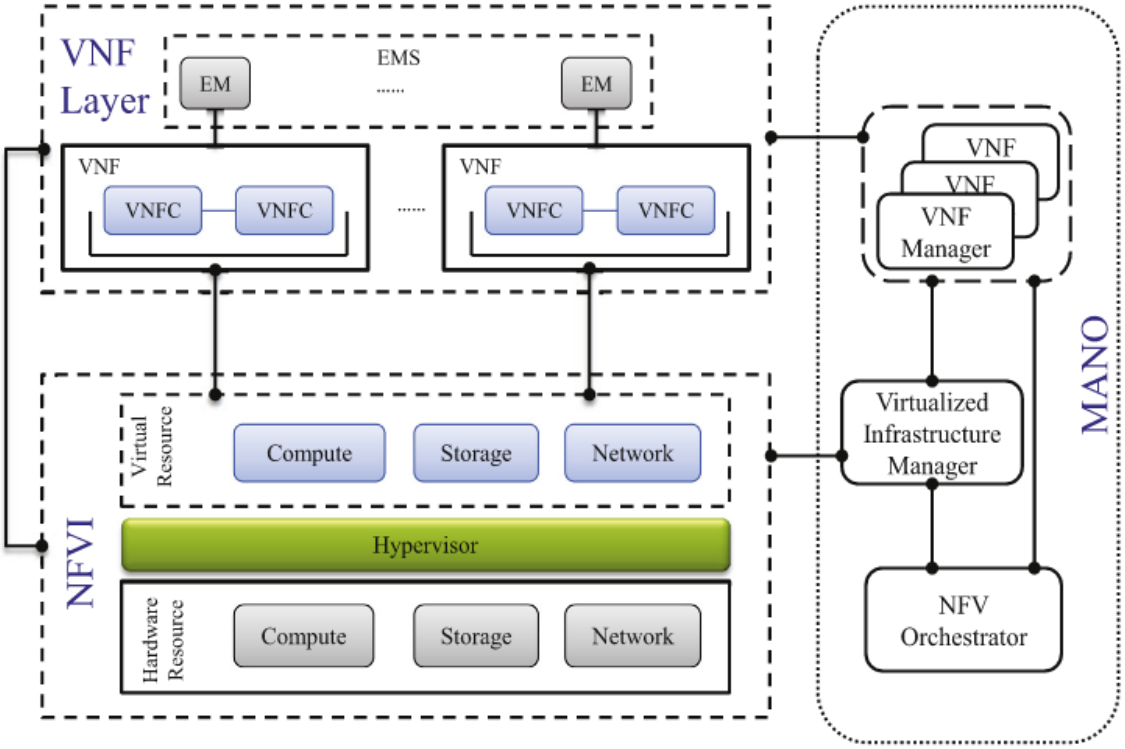


Figure 7 – Proposed NFV architecture by ETSI. (YI et al., 2018)

- **Virtualized Network Function Layer**: Represents the scope where the Virtualized Network Functions (VNF) are deployed. A VNF is composed of several VNF Components that are managed by an Element Manager. All Element Managers of a specify domain composes the Element Manager System (EMS);
- **Network Function Virtualization Layer**: Built on top of several general-purpose hardware. It provides a virtualization environment to deploy and execute VNF instances. The hypervisor acts as an interface between virtual and physical resources;

- **Management and Orchestration Layer**: Responsible for the management of the entire virtualized context within the framework of NFV. The context includes virtualization mechanism, hardware resource orchestration, life cycle management of VNF instances, interface management between modules, etc. All the responsibilities are categorized into three parts according to ETSI, namely, Virtualized Infrastructure Manager (VIM), NFV Orchestrator (NFVO) and VNF Manager (VNFM).

Particularly, the VIM (Virtualized Infrastructure Manager) deserves some attention. The VIM act as an interface between physical resources and Virtual Machines. Also, it provides an Application Programming Interface that allow uses to retrieve information regarding the virtualized infrastructure and the allocated resources. Lastly, some authors and developers have suggested using Cloud Frameworks as VIMs since it has the same functionalities. Although is a relatively new concept, it has several use cases described by (YI et al., 2018):

- **NFVI as a Service (NFVIaaS):** NFVIaaS indicates that NFVI can be offered and sold from one service provider to the other.
- **Content Delivery Network Virtualization:** cache node virtualization to gain preferable performance on some aspects such as network latency and throughput by instantiating CDN nodes closed to the public;
- **VNF Forwarding Graph (VNF FG):** VNF FG can be regarded as an analog of physical network forwarding graph that connects physical appliances via bidirectional cables, which connects VNFs via virtual links to describe the traffic among these VNFs. In particular, connecting multiple VNFs in sequence can constitute a service that is referred to as SFC in the context of NFV.

The management and orchestration of virtualized resources should be able to handle NFVI resources in NFVI Points of Presence (NFVI-PoPs). Management of non-virtualized resources is restricted to provisioning connectivity to PNFs, necessary when an NS instance includes a PNF (Physical Network Function) that needs to connect to a VNF, or when the NS instance is distributed across multiple NFVI-PoPs or N-PoPs.

In the case of virtualized resources distributed across multiple NFVI-PoPs, those services could either be exposed directly by the management and orchestration functions for each individual NFVI-PoP, or via a higher-level service abstraction presenting the virtualized resources across multiple NFVI-PoPs (MAHMOODI et al., 2017).

The NFV management and orchestration functions that coordinate virtualized resources in a single NFVI-PoP and/or across multiple NFVI-PoPs need to ensure exposure of services that support accessing these resources in an open, well known abstracted manner.

These services can be consumed by other authenticated and properly authorized NFV management and orchestration functions (e.g. functions that manage and orchestrate virtualized network functions) (MAHMOODI et al., 2017).

2.3 Service Function Chaining

The implementations of network services require, often, several network functions, such as Deep Packet Inspection (DPI), Firewall, Network Address Translator (NAT) and others. The definition of an ordered list of functions where network traffic must be steered through them in a Service Function Chaining (SFC).

2.3.1 Architecture

The RFC 7665 (PIGNATARO; CARLOS, 2015) standardized the concepts and the architecture of an SFC, listing the core components and considerations regarding security, resiliency, polices, loop detection and others. The architecture of an SFC is shown in Figure 8a. It is built on top of its core components which are classifiers, service function forwarders, service functions, and SFC proxies.

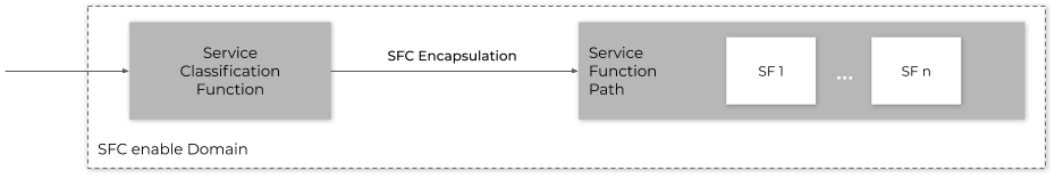
The SFs are inter-connected using SFC encapsulation that is transport-independent. As a result, there is a SFC-enable domain. In order to understand the process of steering network packets, it is important to detail each component present in the Figure 8a and Figure 8b.

2.3.2 SFC encapsulation

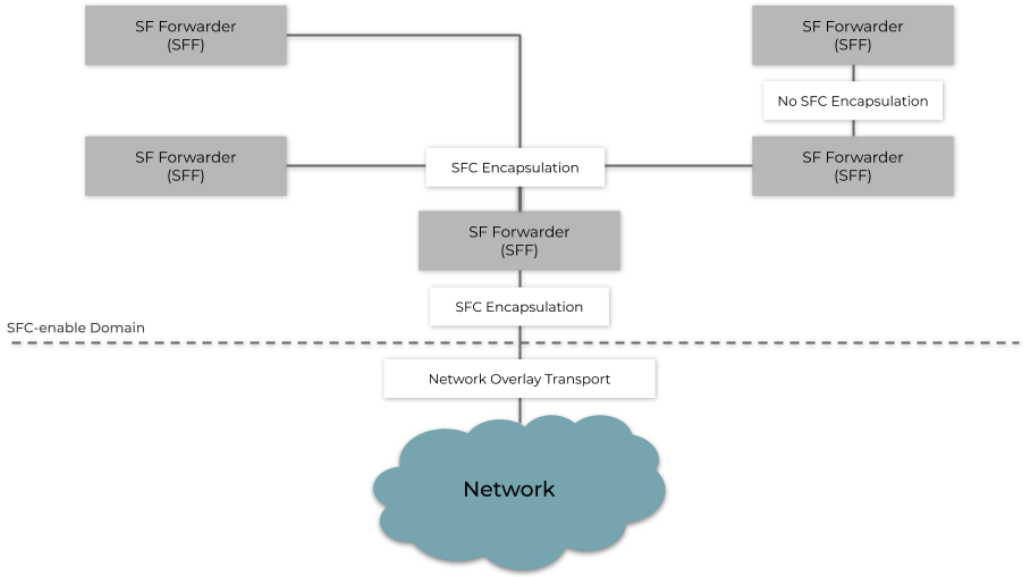
The encapsulation designed by the IETF team proposes to be transport-independent, which means that to transit between network equipment (switches and routers) it relies on the outer encapsulation, such as VLAN, VxLAN and any other slicing technique. This characteristic of being transport-independent enables SFC encapsulation to work with legacy equipment and to work with any transport protocol. Therefore, the SFC encapsulation enables the selection of a Service Function Path (SFP) and the sharing of metadata and context when needed.

2.3.3 Service Function (SF)

A SF is a resource that can be used inside a domain and it is available to multiples SFCs. SFC aware service functions receive traffic with the SFC encapsulation, while SFC unaware SFs receive traffic without encapsulation (Legacy SFs).



(a)



(b)

Figure 8 – (a) Service Function Chain architecture; (b) SFC architecture components after initial classification

2.3.4 Service Function Forwarder (SFF)

SFFs are responsible for forwarding packets (or frames) received from the network after classification to the SFs associated with this SFF. SFF uses the service function path to identify to each SF it should forward the packet. The responsibilities of a SFF are:

- **SFP forwarding:** when the packet arrives from the network to the SFF, the packet is delivered to the appropriated SF. Eventually, the packet returns from the SF to the same SFF. If the next hop in the SFC is a SF connected to the current SFF then the packet is forwarded to this SF. If the next hop is external to the SFF, the forwarder encapsulates with the appropriated transport layer header and pushes the packet back to the network;
- **Maintaining flow state:** the SFF handles the flow state which it is related to the position in the SFC. The SFF need to update the flow state according to the SFP.
- **Terminating SFP:** the traffic must return to the network when the SFC is finished, i.e. when the packet has traversed through the last SF in the SFC. This information

is available in the encapsulation which contains the next hop information.

2.3.5 SFC Proxy

To support SFC's unaware SFs, a SFC proxy must be provided. It is placed between SFs and SFF. The SFF redirects the traffic to the proxy instead of the SF. The proxy receives the packets, removes the encapsulation and delivers the packet to the SF attached to it. When the packet leaves the SF, the proxy reclassifies the outgoing packets and re-encapsulates before sending it to the SFF.

2.4 Software Defined Networks

Traditional IP networks, although widely used, can be extremely complex to configure, as they use distributed control and require the configuration using proprietary software, usually, in a low-level interface. The reconfiguration and automatic response mechanisms are practically nonexistent in today's IP networks. Applying the necessary policies in such a dynamic environment, as the current data center networks, is highly challenging.

Software Defined Networks (SDN) appears in an attempt to overcome the limitations of current networks. To achieve this goal, as can be seen in Figure 9, there is a separation between the network control logic (the control plane) of the underlying routers and switches that forward the traffic (the data plane) (KREUTZ et al., 2015).

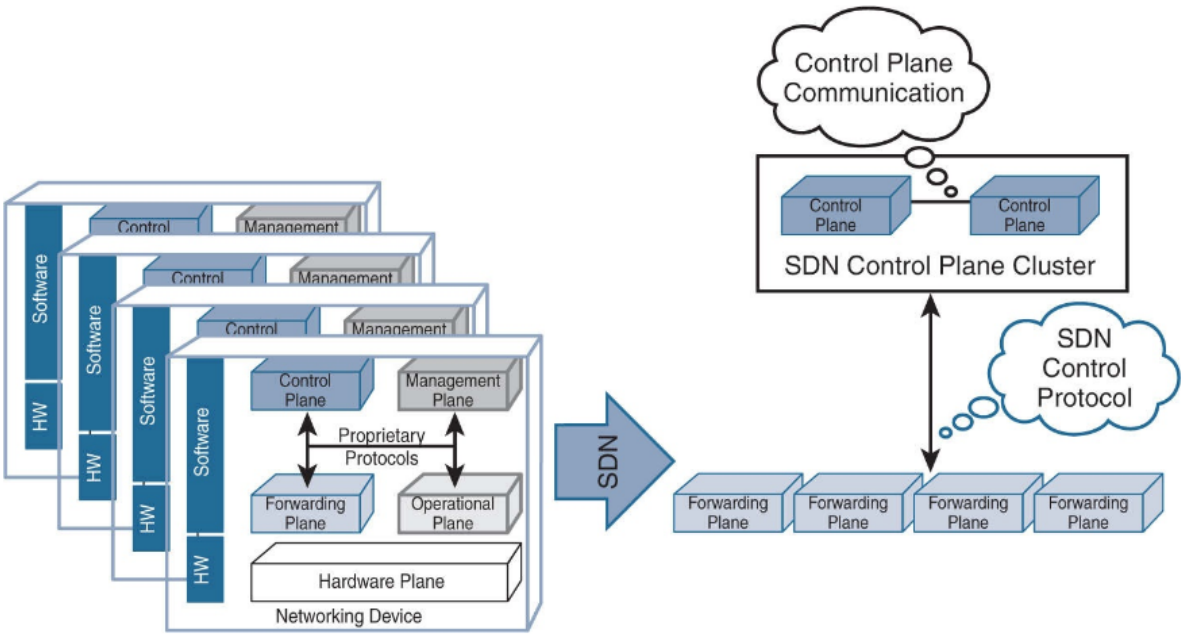


Figure 9 – Transition from traditional network technology to SDN paradigm. (CHAYAPATHI SYED F. HASSAN, 2016)

Separation is possible through a well-defined programmable interface between the switches and the SDN controller. The controller performs direct control over the switches in the data plane using an API. The most notable example of such an API is OpenFlow (MCKEOWN et al., 2008).

2.4.1 OpenFlow

OpenFlow was the first open source protocol for communication between SDN controllers and data plane equipment. OpenFlow provides a protocol for modify flow tables, which defines the forwarding of the message.

Figure 10 shows a high level view of a OpenFlow switch components. According to (MCKEOWN et al., 2008), an OpenFlow Switch consists of at least three parts:

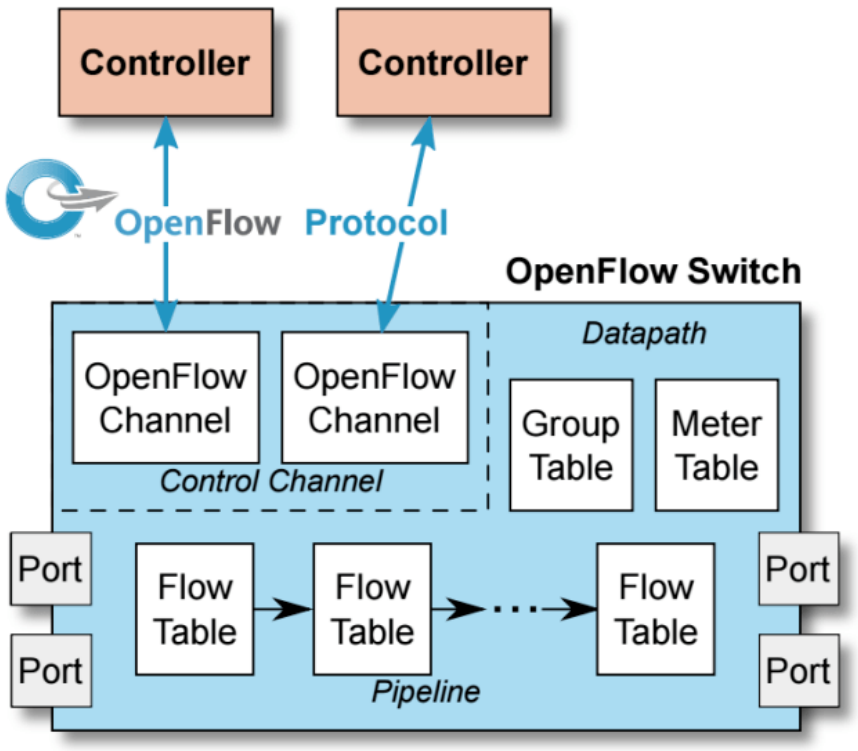


Figure 10 – Main components of a OpenFlow switch. (FOUNDATION, 2020a)

1. A flow table with an action associated with each entry in the table. Thus, the switch is told how to process each flow;

2. A secure channel between the switch and the controller allowing packet and command traffic and;

3. The OpenFlow protocol itself that contains the enabling tool for communication between the controller and switch.

When a packet arrives at the switch, it searches the chain of flow tables for any rules that the packet fits into and performs the corresponding action. If the packet does not fit any rule, it is forwarded to a queue and an event is sent to the SDN Controller. The controller responds with a rule for this particular packet. Each entry in the table has a match, an action in case of agreement with the match and a counter that registers the number of received packets that are in accordance with that rule.

2.4.2 Ryu Framework

Ryu is a component-based network controller and has a number of predefined components. Components can be modified, extended and customized to create SDN applications (KHONDOKER et al., 2014). SDN applications are created and communicate through the Rest API available with the framework. The framework forwards requests to OpenFlow switches using the OpenFlow protocol. This scheme is shown in Figure 11.

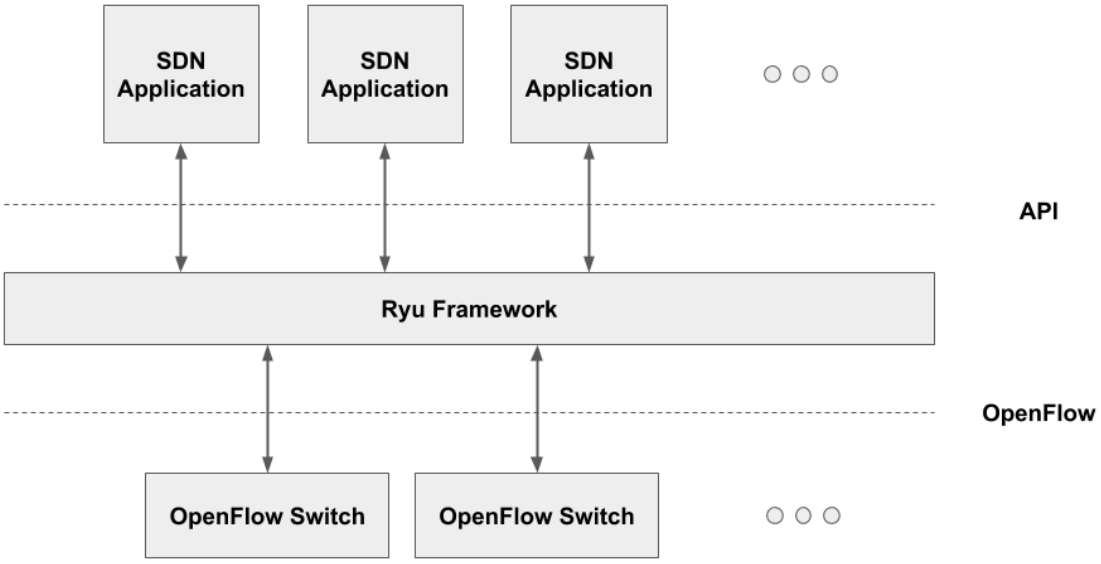


Figure 11 – Application architecture using Ryu network controller.

The main components of the Ryu controller are:

- **app_manager**: is the main component. It is responsible for starting the application, providing context to applications and route messages between applications;
- **controller**: the main component of the OpenFlow controller. Responsible for managing application connections with switches and for managing and forwarding events generated by Ryu's applications.

- **dpset**: manages the virtual switches assigned to the controller. In newer versions it has been replaced by the topology component.
- **ofp_event**: contains definition of events.
- **ofp_handler**: deals with OpenFlow negotiation.

The framework also contains several components that encode and decode OpenFlow messages in their different versions.

2.4.3 Open vSwitch

Open vSwitch is an open source virtual switch, widely used for enabling networks for virtualized environments. Virtual machines connect via bridged ports allowing them to communicate with each other, as well as with the physical network.

The software is basically written in the C programming language. It supports multiple virtualization platforms such as: Xen/XenServer (XENSERVER, 2017), KVM (KVM, 2016), and VirtualBox (ORACLE, 2011).

The main components of the Open vSwitch are:

- **ovs-vswitchd**, a daemon that implements flow-based switching of the switch, along with a Linux kernel module.
- **ovsdb server**, a lightweight database server that *ovs-vswitchd* queries for configuration information.
- **ovs-dpct**, a tool for configuring the switch's kernel module.
- **ovs-vsctl**, a utility to query and update the configuration of *ovs-vswitchd*.
- **ovs-appctl**, a utility that sends commands to run the Open vSwitch daemons. (FOUNDATION, 2017)

Figure 12 shows the components that make the packet switching of the virtual *switch*. When receiving a new flow and there is no specific action, the package goes up to the user level (userspace) where the most specific rule possible is consulted. After this step, the rule is cached at the kernel level.

As Open vSwitch is generally used in SDN environments, you can insert a controller that communicates using the Openflow protocol (Section 2.4.1). The controller can add, delete, monitor and obtain statistics about the flow in the Open vSwitch instances.

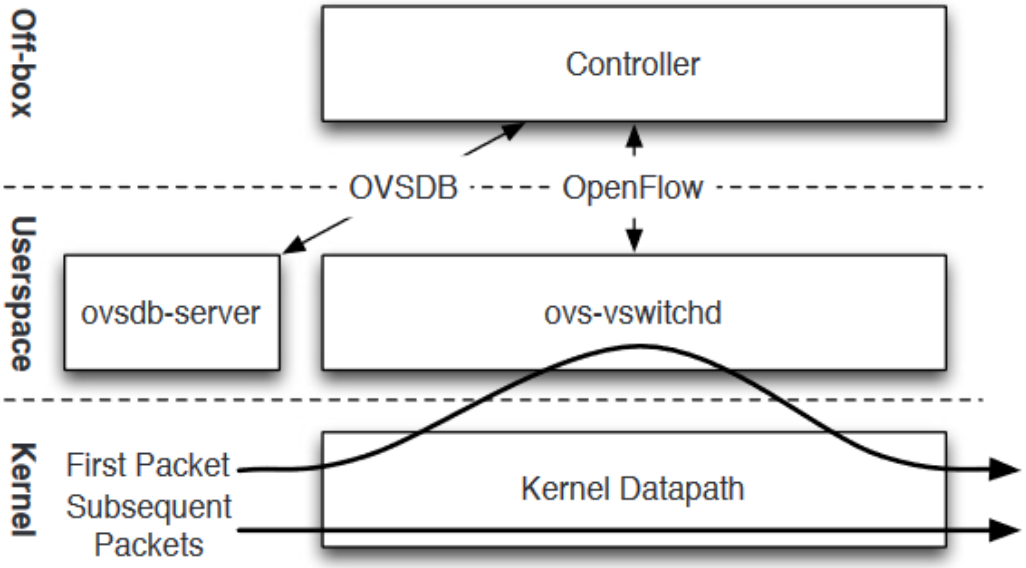


Figure 12 – Open vSwitch components. The first packet of a stream goes up to the userspace due to the occurrence of a miss. The decision is then saved for possible packets in the same flow.(FOUNDATION, 2017)

2.5 Network Slicing

The Next Generation Mobile Networks (NGMN) Alliance published a white-paper in 2015 that introduced the network service deployment concept of Network Slicing. The network slicing concept consists of 3 layers: 1) Service Instance Layer, 2) Network Slice Instance Layer, and 3) Resource layer, as shown in Figure 13.

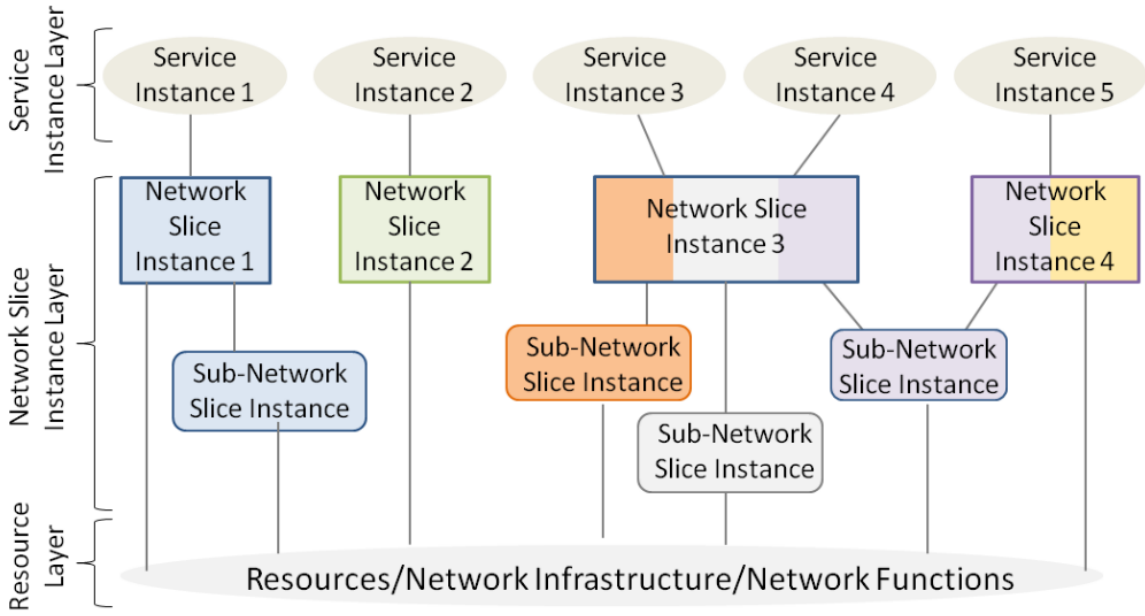


Figure 13 – The NGMN network slicing concept.(AFOLABI et al., 2018)

The **Service Instance Layer** represents the network services that clients hire from Service Providers. Each service needs to be specified by some pattern or blueprint that contains the network requirements and resources that a service needs. This blueprint defines the implementation of service over the Network Slice Instance Layer. Therefore, a **Network Slice Instance** provides the network characteristics, which are required by a Service Instance. Figure 13 shows that different Service Instances share network Slices Instancesm and may be connected to none or many Sub-network Slice Instances. The **Resource Layer** specifies the physical and logical resources available in network infrastructures.

Regarding a service provided by multiple service providers, from the point-of-view of wireless services, there are two main scenarios: (i) roaming scenario: individual users move from one provider (i.e., Home network provider) to a network managed by another provider (i.e., Visited network provider); (ii) business verticals: when the capabilities of a single provider cannot meet a business vertical service request, the provider may harness the necessary capabilities from another provider, based on an SLA (Service Level Agreement) between the two providers. (NGMN, 2016)

According to (AFOLABI et al., 2018), network slicing implementations need to follow seven main principles that shape the concept and related operations:

- Automation: enables an on-demand configuration of network slicing without the need for fixed contractual agreements and manual intervention.
- Isolation: is a fundamental property of network slicing that assures performance guarantees and security (to defend network openness to third parties) for each tenant even when different tenants use network slices for services with conflicting performance requirements.
- Customization: assures that the resources allocated to a particular tenant are efficiently utilized to meet the respective service requirements.
- Elasticity: is an essential operation related to the resource allocated to a particular network slice to assure the desired SLA.
- Programmability: allows third parties to control the allocated slice resources.
- End-to-end: is an inherent property of network slicing for facilitating a service delivery all the way from the service providers to the end-user/customer(s).
- Hierarchical abstraction: is a property of network slicing that has its roots on recursive virtualization. The resource abstraction procedure is repeated on a hierarchical pattern with each successively higher level offering a greater abstraction with a broader scope.

2.6 Related Works

This section aims to position this proposal concerning related works in literature. Related Works were classified and subdivided into three categories in this section:

- Source Routing;
- SFC mechanisms for a single data center;
- SFC mechanisms for multiple data centers.

2.6.1 Source Routing

Source Routing is not a new concept, it was proposed for the first time by (SUNSHINE, 1977) in 1977 and discussed later in 1978 (SUNSHINE, 1978). In Source Routing, the entire path at switch level is presented in each packet, which is different from traditional methods that use a hop-by-hop approach.

Source Routing has gained traction as data center networks move towards fabric-based architectures and the demand for optimizations increases, especially for the task of forwarding in the core of DC networks. According to (JYOTHI et al., 2015; SUNSHINE, 1978; VALENTIM et al., 2019), there are several advantages when using source routing for forwarding packets in a data center, such as:

- **Higher throughput:** by allowing more path diversity than hop-by-hop-based methods, source routing increases the network utilization;
- **Smaller forwarding tables:** source routing requires relatively fewer flow entries in DCN;
- **Nearly static forwarding tables:** the addition of new nodes does not require updates in the forwarding tables because the flow entries depend only on the packet information;
- **Fast response to failures:** with a centralized controller source routing can react to link failures without the need to wait for distributed algorithms to converge;
- **Architectural solution for constant update:** the constant update of switch forwarding can result in security failures or inconsistent states. On the order hand, source routing does not require updates in the core, reducing the number of updates on forwarding tables and reducing the possibility of errors;
- **Path provenance:** source routing makes it easier to track the source of packets since each link in the path can be retrieved from the packet header;

- **Ease of verification:** it is hard to verify the state of a hop-by-hop-based network (IP for example) since the routing algorithm is distributed and any verification requires to build models and simulations. Source routing forwarding tables are easier to verify since that are fewer entries compared to IP-based ones;
- **Easy to attend to flow requirements:** with source path selection it is possible to choose a path that supplies flow requirements, such as: response time, latency, bandwidth, jitter, and others.

Loose Source Routing (LSR) (Bhagwat; Perkins; Tripathi, 1996) is an option supported in IP, which can also be used to perform address translation operation. The destination of the packet is replaced with the next router the packet must visit.

Source Routing (SSR) allows a flow source to list the specific routers that a packet must visit on the way to its destination. No deviation from this list is permitted. The traditional way of performing SSR is to insert a stack (or an ordered list) of ports or addresses in the packet header that details the entire path to be taken by the packet in the data center infrastructure. Thus, each forwarding element performs a *pop* operation on this stack to discover the exit port, without the need to query forwarding tables. Examples of this approach include: SecondNet (GUO et al., 2010), Segment Routing (CLAD et al., 2018) and Sourcey (JIN et al., 2016).

Another alternative SSR approach explores the mathematical concept of the Residual Number System (RNS). In this approach, the exit port on each node is defined by the remainder of division over the route identifier (MARTINELLO et al., 2014). An RNS-based SSR scheme can exploit properties to provide network functions that cannot be covered by the traditional list method, such as forwarding without packet rewriting and resilience (DOMINICINI et al., 2020).

Related works have already explored this type of SSR approach to provide rapid reaction to failures (GOMES et al., 2016) and network programmability (MARTINELLO et al., 2014). Some works have also investigated techniques to improve the scalability of the number of bits of the route identifier and demonstrated its efficiency in fulfilling latency restrictions for multicast in data centers (JIA, 2014). Finally, the `KeySFC` (DOMINICINI et al., 2020) scheme applied SSR to the SFC problem in a single datacenter.

The majority of SSR implementations assume a centralized controller with a global view of the network topology. Some implementations can add routing information in any field of the packet. Most solutions use OpenFlow to make flow classification and add routing information.

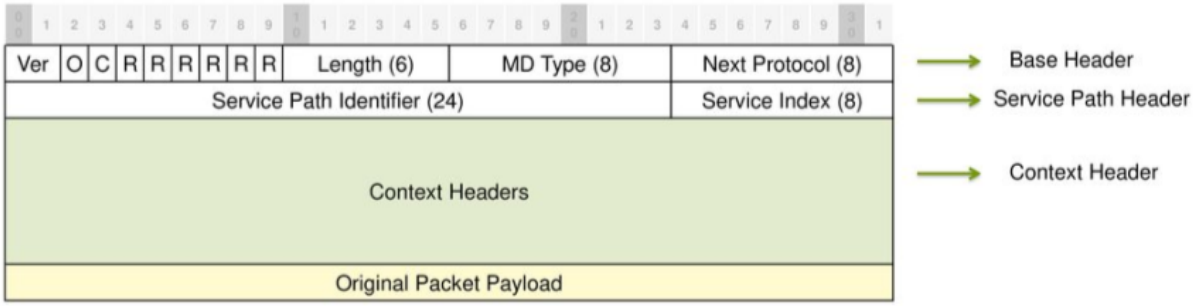


Figure 14 – NSH Header.

2.6.2 SFC mechanisms for single data center

NSH (QUINN; GUICHARD, 2014) is the standard SFC encapsulation header. The NSH protocol specifies a header (Figure 14) that includes two chain identifiers: the first indicates the selected chain and the second indicates the current position in the chain. The forwarding elements in the path use the information from these identifiers to search a table and decide which VNF to run at any time. Then, the address of the next network element is encapsulated in a more external header, relying on the underlying routing methods to deliver the packet between those elements. Although very flexible, this scheme requires table queries for each hop both to discover the next VNF and to route between functions in the physical network, which leads to scalability and agility problems related to the number of states in the network. Besides, pushing headers expands the packet size, the increase of traffic and possible problems with fragmentation

Another popular method is Segment Routing (CLAD et al., 2018), where an ordered list of segments is encoded as a stack of MPLS labels. The next segment to be processed is extracted from the top of the stack and a forward table search operation is performed at each hop. Although the source node may need to send a long list of segments to obtain optimal paths, most MPLS devices support a limited stack size (about 3 to 5 labels), which can lead to inefficient distribution traffic (Abdullah et al., 2019). Therefore, although the Segment Routing scheme can be used to enable SFC with SSR, it generally uses a hybrid scheme that specifies a list of a few network elements that the packet must pass through and delegates the routing between those elements to the underlying network, which employs shorter paths using ECMP (Equal-Cost Multi-Path routing), for example.

`KeySFC` (DOMINICINI et al., 2020) enables SFC using RNS-based SSR. In contrast to the Segment Routing and NSH schemes, `KeySFC` does not restrict the selection of paths, it reduces the number of states on the network, especially in scenarios with multiple clouds, and eliminates the tables present on the devices in the core of the networks.

2.6.3 SFC mechanisms for multiple data centers

Several important works have already addressed the problem of linking network functions in multi-cloud environments from the point of view of an optimization problem. One can quote (BHAMARE et al., 2017), whose focus is to decrease the flow between *data centers* through the efficient positioning of VNF chains. (SUN et al., 2019) also addresses the positioning problem using Integer Linear Programming (ILP, or *Integer Linear Programming*) and optimizes energy consumption with a focus on meeting chain requirements.

v (DIETRICH et al., 2017) solves the routing problem, through optimization by ILP, proposing the partitioning of chains in multiple domains connected by virtual *gateways* with the strategic positioning of the VNFs in these domains. (GUPTA et al., 2017) also optimizes the positioning of VNFs in multi-cloud environments using a predictive algorithm to minimize the latency of function chains. However, although some optimization solutions for the multi-cloud SFC problem have been proposed in the literature, few studies focus on the development of SFC mechanisms to allow the implementation of these resource allocation solutions in network infrastructures data center.

As most implementations of SFC mechanisms in single data centers use NSH as an encapsulation protocol, one approach to pass context between different clouds is to use the stacking of NSH headers. (VU; KIM, 2016) implements a Hierarchical SFC (hSFC), a sub-problem of multiple domains chaining. However, this approach carries the same problems as the NSH protocol concerning low scalability and high complexity in the management of network states, with the addition of extra headers for passing context between domains and, consequently, greater overhead in packet forwarding and increase in the number of states on the network.

2.6.4 Contributions Highlights

In summary, considering the state of the art and the relevance of the related works presented in this section, it is possible to affirm the relevance and timeliness of the problem addressed in this work. There exists a gap in the literature regarding the proposal of referral mechanisms that implement SFC in multi-datacenter environments in a flexible and efficient way. Also, the novel use of SSR to perform network slicing and the introduction of a gateway to abstract the interconnection between infrastructure, which are the most important highlights of our proposal.

3 Proposal of Architecture

We presented the motivation and the research problem in Chapter 1. This Chapter explains the solution towards building an architecture to end-to-end network slicing using tableless strict source routing.

The network slicing problem has three layers of abstraction. When we consider only the Network Slice Instance Layer, the orchestration of a network slice instance requires a set of run-time network functions, and resources to run these network functions, forming a complete instantiated logical network to meet certain network characteristics required by the Service Instance (NGMN, 2016). These are the same requirements to implement a Service Function Chaining which is a ordered set of abstract service functions and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of classification (PIGNATARO; CARLOS, 2015).

The similarity between the two tasks is useful, because it allows the use of the same tools developed to solve SFC problems to be applied in network slicing problems. However, our goal is to create a complete solution that goes beyond SFC and is able to provide the virtualized network service covering the management of physical resources and service instances. Therefore, we need to adapt the solutions to consider the provisioning of physical resources in the Resource Layer and the management of network services in the Service Layer.

Following SDN principles we proposed an architecture with a logical separation between data and control planes shown in Figure 15. The Data Plane contains both physical and virtual elements that are responsible for hosting and connecting the network functions that compose the network services. These elements are distributed in different infrastructures and managed via control interface by the elements in the control plane.

The architecture represents the control plane at a functional level and it does not imply any specific implementation. Multiple functional blocks may be merged internalizing the reference point between them. The communication between the control plane and the data plane follow specific protocols and must be assured in order to guarantee the proper functioning of the infrastructure.

In the following, we detail each architectural element dividing the approach by control and data plane elements.

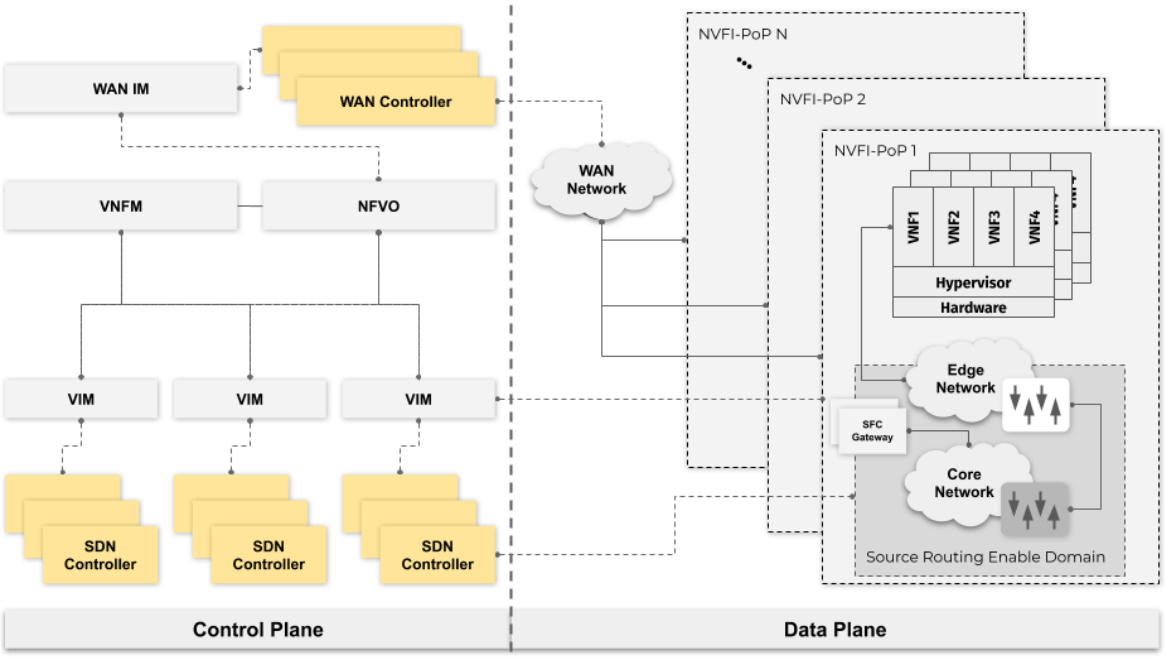


Figure 15 – Proposed architecture for Control and Data planes.

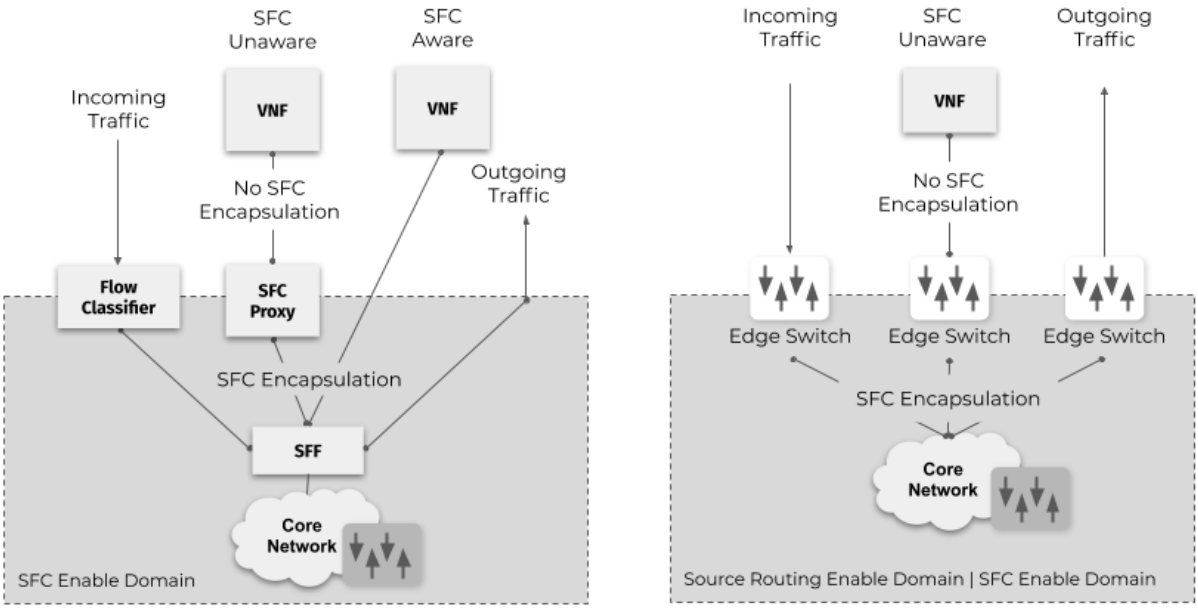
3.1 Data Plane Architectural Elements

The Data Plane designed has a dual approach. It uses the separation between core and edge proposed by KeySFC (DOMINICINI et al., 2020) and uses the RFC 7665 specifications for the SFC functional components with the effort to adapt to a tableless strict source routing context. The RFC 7665 specify functional blocks which can be merged and implemented together. The KeySFC major specification is the separation between core and edge elements and each element has very defined tasks. This Section makes the correspondence between the RFC7665 functional blocks and the KeySFC architectural elements. The correspondence is summarized in Figure 16.

3.1.1 Architectural components

Encapsulation. The SFC encapsulation allows the orchestrator to indicate the next VNF in the SFC. In this proposal, it was chosen to use Strict Source Routing which allows the SFC encapsulation not just to specify the next VNF, but also the complete network underlay path. Therefore, this proposed data plane architecture uses transport-dependent encapsulation which differs from the RFC7665. The understanding of the RFC is that by opting for a transport-independent encapsulation, it is possible to achieve maximum retro compatibility with existing data center deploys. On the other hand, KeySFC states that by using a new approach for data center networks, it is possible to achieve high performance.

Therefore, in this work we are going to use a transport-aware encapsulation. The



(a) (b)

Figure 16 – (a) Logical architecture proposed by RFC7665. (b) Functional architecture proposed by KeySFC Architecture.

SSR implementation may vary and the only constraint is that at any underlay hop the outgoing port is defined. Examples are Sourcey (JIN et al., 2016), KeySFC (DOMINICINI et al., 2020) or even MPLS (ROSEN; VISWANATHAN; CALLON, 2001). The use of SSR increases path diversity and can optimize network usability. In KeySFC proposal, the responsibility to add and remove encapsulation is to the edge elements. Core elements forward packets based on labels in the encapsulation.

Service Function Forwarder (SFF). The SFF is responsible for forwarding packets received from the network to one or more SFs associated with a given SFF. It uses information conveyed in the SFC encapsulation. Traffic from SFs eventually returns to the same SFF, which is responsible for injecting traffic back onto the network. In KeySFC the SFF is an edge switch and it is transport-aware. In this architecture, all SFs are assumed to be SFC-unaware. Therefore, every time an SFF wants to deliver a packet to a SF, it first forward to an SFC Proxy that will modify the packet removing any SFC-related modification.

Service Function (SF). A Service Function in this architecture is assumed to be SFC-unaware. A Service Function can be a PNF (Physical Network Function) or a VNF (Virtual Network Function). In both cases, it is required that an SFC packet goes through a SFC Proxy before the SF.

Classification and Reclassification: In Figure 16 (a), there is an interface delimiting a SFC-enable domain. A packet necessarily enters in this domain through a

Flow Classifier. In Figure 16 (b), the element that performs as a Flow Classifier is the Edge Switch. The Edge Switch is also the entry point to the SR-enabled domain. Every time a packet enters an SF, it is required to leave the SR-enabled domain, which is why the SF is SFC-unaware. For that reason, it is required a reclassification every time a packet leaves an SF. As stated in the RFC 7665, this task is executed by an SFC Proxy, that reclassifies the packet and forwards it to the SFF and the element responsible for the classification is the Edge Switch.

Transport Network and Network Components: Underneath the SFF there are elements responsible for performing transport forwarding. In the original RFC 7665, they do not consult the SFC encapsulation or inner payload for performing this forwarding. However, in this proposal, they do consult encapsulation to perform transport hops. RFC7665 treat the underneath elements as a black-box, in this design, they will act as white-boxes with its behavior defined by the control plane. The lack of specificity can lead to loss of performance in experimental environments, but this is avoided by the use of programmable hardware (e.g SmartNICs, P4 programmable switches and NetFPGA) even dedicated hardware for production deployments.

SFC Proxy: The SFC Proxy is a functional block defined by the RFC7665, in the KeySFC architecture it is implemented in the Edge Switches of the data center network. The SFC Proxy responsibilities are: (i) remove SFC-encapsulation; (ii) restore transport information to an SFC packet and deliver to the SF attached to it and; (iii) reclassify packets which egress from SFs and forward to the SFF.

3.1.2 SFC Gateway

Some Network Services requires the provision of network functions to be in different data centers, or to forward some flow from one infrastructure to another. In these cases, administrators need an efficient way to stitch the network slices and build the hired services. This problem also exists for SFC, however, the literature focus on the optimization of path and positioning of VNFs, while few works have addressed to the mechanisms to connect the SFC segments.

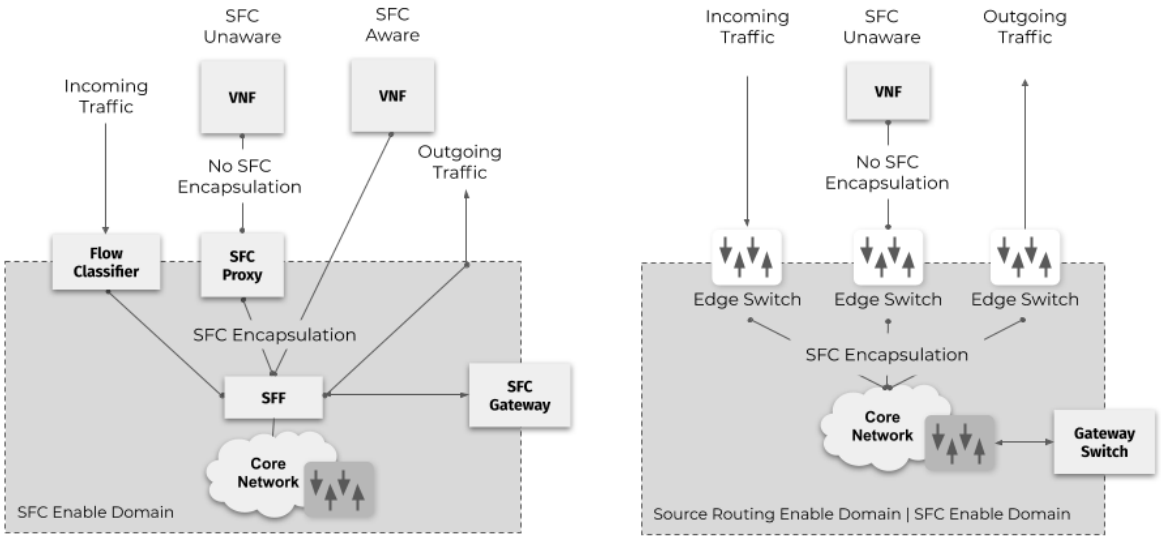
Still on SFC, the RFC7665 (PIGNATARO; CARLOS, 2015) do not specify any protocol to be used in the transport of packets in a hop between two Service Functions. The RFC proposes the use of NSH encapsulation and relies on traditional transport network to deliver a packet to the next SF. This method is not easily done in hops that require inter-datacenter communication. The configuration of a transport network between geo-distributed data centers should be tedious and it involves hiring several TSP (Telecommunication Service Provider) to connect each data center.

Another solution explores the use of VPNs endpoints virtualized as Service Func-

tions that implement a Layer 3 connectivity between two segments of SFC located in different Infrastructure Domains. In this option, the user is responsible for the configuration of the connections and the operator loses the control over this stage of the process.

In this context, we propose an SFC Gateway, which is an architectural element in the data plane. The SFC Gateway manages VPNs on-demand and abstracted from the administrator the creation of such networks. Those connections are Layer 3 VPNs, though, from the point-of-view of the SFC are only Layer 2 hops. The delivery of packets to theses SFC Gateways are integrated to the protocol not differing from other host interfaces.

Figure 17 shows the new element in the reference architecture. We developed the SFC Gateway to receive packets from other infrastructures and to forward flows to other infrastructures. The forwarding respects the encoded information in the SFC tunneling. Therefore, the orchestrator process the required route for an flow, anticipates the need to deliver a flow to another infrastructure, and incorporate the forwarding information in the encapsulation.



(a) (b)

Figure 17 – (a) Addition of a SFC Gateway element to the logical architecture proposed by RFC7665. (b) Addition of a Gateway Switch to the functional architecture proposed by KeySFC Architecture.

The SFC Gateway is a specific contribution of this work and it is capable of simplify the management of distributed network slices. The SFC Gateway can be distributed in the infrastructure and the Orchestrator might considers to forward the flows to a specific SFC Gateway instance considering aspects such as latency, traffic congestion and path usage. The responsibilities of an SFC Gateway are to: (i) (re)classify incoming packets to the proper SFF; (ii) forward packets to the destination data center according to the VNF FG; (iii) add and remove VPN encapsulation.

3.2 Control Plane Functional Blocks

Currently, implementations for management and orchestration of network functions are focused on centralized approaches, especially in scenarios where services spawn over several administrative domains (MIJUMBI et al., 2016). ETSI proposed a control plane architecture for the composition of virtualized services. In ETSI documentation, there is an extensive list of tasks each block should perform. The following sections refer to some of ETSI definitions and include the responsibilities and capabilities required to adapt the standard architecture to handle with source routing enabled Infrastructure Domains.

3.2.1 NFV Orchestrator (NFVO)

The NFVO's main responsibilities are to: (i) orchestration of NFVI resources across multiples VIMs and; (ii) life-cycle management of Network Services. To fulfill its responsibilities, the NFVO needs to perform several tasks. The following is a non-exhaustive list of capabilities provided by the NFVO in a multi-domain context:

- Network Service instantiation and its life-cycle management;
- Management of the Network Service instances topology (e.g. create, update, query, delete VNF Forwarding Graphs);
- Collect topological information from every Infrastructure Domain on the orchestrator responsibility;
- Underlay path management according to polices (e.g shortest path, the bandwidth available, etc).

3.2.2 VNF Manager (VNFM)

The VNF Manager is responsible for managing the life cycle of VNF instances. Every VNF should be on the responsibility of a VNF manager. Usually, the VNF Manager is agnostic about the VNF type, however, in this implementation, the VNF type is important for the VNF FG use-cases when the VNF may change the nature of flow and compromise the reclassification in the traffic steering. The following is a non-exhaustive list of capabilities provided by the VNFM in a multi-domain context:

- VNF instantiation, including VNF configuration if required by the VNF deployment template;
- VNF instance modification;
- VNF instance scaling out/in and up/down;

- Retrieve VNF instance information on events.

The VNFM should have a Virtualized Network Function Descriptor (VNFD) catalog. The VNFD is a VNF deployment and operation abstraction. The catalog stores all the supported VNFs in the Infrastructure Domain. This is an important aspect for any NFV solution implementation, however, this proposal will not extend the subject since is a proof-of-concept. Moreover, a VNF, when used in a VNF FG, can be aware or unaware concerning the traffic steering. The VNFM should attend to this characteristic since it affects not just the VNFM but also the NFVO.

3.2.3 Virtualized Infrastructure Manager (VIM)

The Virtualized Infrastructure Manager (VIM) is responsible for controlling and managing the NFVI compute, storage, and network resources, usually within one operator's Infrastructure Domain (e.g. all resources within an NFVI-PoP, resources across multiple NFVI-PoPs, or a subset of resources within an NFVI-PoP). The following is a non-exhaustive list of capabilities provided by the VIM in a multi-domain context:

- Maintain hardware repository (storage, computing, and network);
- Keep a record of the allocation of virtualized resources;
- Interact and manage with the hypervisor;
- Keep a record of hardware usage and status;
- Management of virtual networks.

Some literature has been referring to the VIM as Cloud Frameworks which possess the same responsibilities to act as a layer between virtualized resources correspondents. However, to enable cloud frameworks to support NFV usually it is necessary to use plugins or extensions.

3.2.4 SDN Controller

The SDN Controllers are responsible for providing a programmatic way to interact with the network elements in the Infrastructure Domain. Basic connection, or Network as a Service is provided by the VIM. Hence, the Controllers are complementary to the Infrastructure Domain NFV capabilities. Figure 15 shows several instances of controllers connected to different elements in the Domain.

The following is a non-exhaustive list of capabilities provided by the Network Controllers in a multi-domain context:

- Perform topological discovery covering all the Infrastructure Domain;
- Manage the flows that performs the network slicing and the partitioning of network resources.
- Perform flow propagation which is responsible for steering, receiving the information from a centralized and globally agent such as NFVO;
- Resource management and tracking of network resources and attributes such as bandwidth, jitter, delay, and etc;
- Implement southbound interfaces to the compute and network resources providing the connectivity services to create overlay tunnels (e.g. VXLAN, NVGRE, MPLS over GRE) or network partitions (for example using OpenFlow);
- Abstract the information exposed by the underlying NFVI network via various southbound interfaces.

3.2.5 Wide Area Network Infrastructure Manager (WAN IM)

The WAN IM is responsible for the management of virtual connections between NFVI-PoPs. NFVO may require virtual connections to the WAN IM that establishes the connection and manages its life-cycle.

The design of this proposal leverages from a Software-Defined WAN (SD-WAN) approach. SD-WAN is proposed to apply software-defined techniques in networking connections covering a wide geographical area, and it achieves the purpose of software control using the philosophy that is different from Software Defined Network (YANG et al., 2019). Therefore, the connections between NFVI-PoPs are virtual and based on software switches. Also, the connections between Infrastructure Domains are based on Layer 3 protocols, which allow the implementations to use the Internet or any routed third-party connections.

The WAN IM manages the SFC Gateways of its infrastructures. These elements perform the stitching of segments of slices from different infrastructures. The following is a non-exhaustive list of capabilities provided by the WAN IM in a multi-domain context:

- Resource management and tracking of network resources and attributes such as bandwidth, jitter, delay;
- Implement southbound interfaces to the compute and network resources providing the connectivity services to create overlay tunnels (e.g. VXLAN, GRE, MPLS over GRE) or network partitions (for example using OpenFlow);
- Abstract the information exposed by the underlying NFVI network via various southbound interfaces.

4 Implementation and Evaluation

This chapter presents the testbed used to evaluate the proposed architecture. It also details the Data Plane and Control Plane implementation presenting scenarios to illustrate how the orchestrator enables network slicing and the stitching of slices using tableless source routing.

4.1 Testbed Description

The prototype consists of 5 servers with 2.4 GHz Intel Xeon E5-2620 processor, 16 GB of memory and 6 Gbps Ethernet NICs and CentOS version 7 operating system. Figure 18 shows a representation of the implemented prototype.

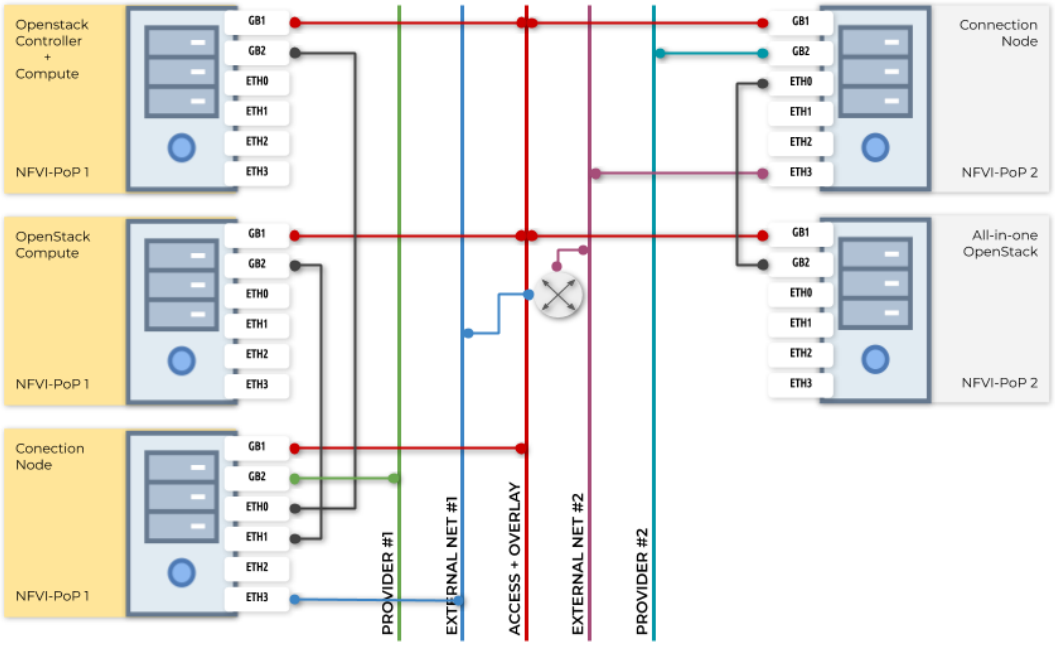


Figure 18 – Prototype view at server level with networks connecting each other.

As we want to demonstrate the stitching of slices, we need to use a scenario with multiple NFVI-PoP. The implementation has two different infrastructure domains: NFVI-PoP 1 and NFVI-PoP 2. NFVI-PoP 1 has two host nodes, and NFVI-PoP 2 has only one node. The two remaining nodes in Figure 18 host instances of software switches that complement the infrastructure and connect the NFVI-PoPs to the data center network. The WAN connection between the two NFVI-PoP is a 100 Mbps connection which is a realistic bandwidth value.

The networks follow the requirements for the view topology, and cross-cables are used to connect the Compute Nodes (OpenStack nodes that host VMs) to the named Connection Nodes. Connection Nodes are servers equipped with virtual switches used to connect with the data center network and interconnect the nodes in the same cloud.

Each OpenStack cloud requires at least three networks: Access, Overlay, and Provider. The access network is the network that is important to perform cloud management. In our prototype, we merged the Access network with the Overlay network. The Overlay network provides the means to OpenStack to create virtual networks using overlay protocols such as VLAN, VxLAN, GRE, and others. These virtual networks interconnect VMs and provide traffic isolation between tenants in the cloud. There is only one Access + Overlay for both clouds since we are using just for SSH access, and we do not create virtual networks.

The Provider is a network that OpenStack does not have control over it, and the data center operator is responsible for the management. Usually, every node in the cloud would have direct access to this network. However, in our implementation, access to the Provider network is through the Connection Node. Since the cloud now uses tableless source routing as the transport protocol, it cannot communicate with the Provider network without some translation. Therefore, the Connection Node manages the compatibility between this network traffics. The Provider network is essential for the VMs to have access to the internet.

The External network is the network with access to the WAN. As both our clouds are in the same data center, we created a router between the External network from NFVI-PoP 1 and the External network from NFVI-PoP 2.

Figure 19 shows a representation of our prototype showing both the Control Plane and Data Plane and simplifying the network representation.

Considering the premise of using solutions available on the market, the implementation chooses *OpenStack* as Virtualized Infrastructure Manager (VIM). The choice is justified because it is a modular cloud framework that manages storage, network, and computing resources. Through the programming interface available in the *OpenStack SDK* it is possible to access the resources of the data center infrastructure available in *OpenStack*.

The option for *OpenStack* as VIM is convenient since the structure of virtual switches used to deliver connectivity to virtual machines is compatible with both the ETSI architecture and `KeySFC`. As can be seen in Figure 19, the hosts have Open vSwitch bridge (***`br-int`***, ***`br-int`***) instances compatible with the forwarding mechanism proposed in `KeySFC`. OpenStack uses these bridges to deliver Layer 2 connectivity to virtual machines, and the SFC data plane leverage this structure to build the proposed architecture.

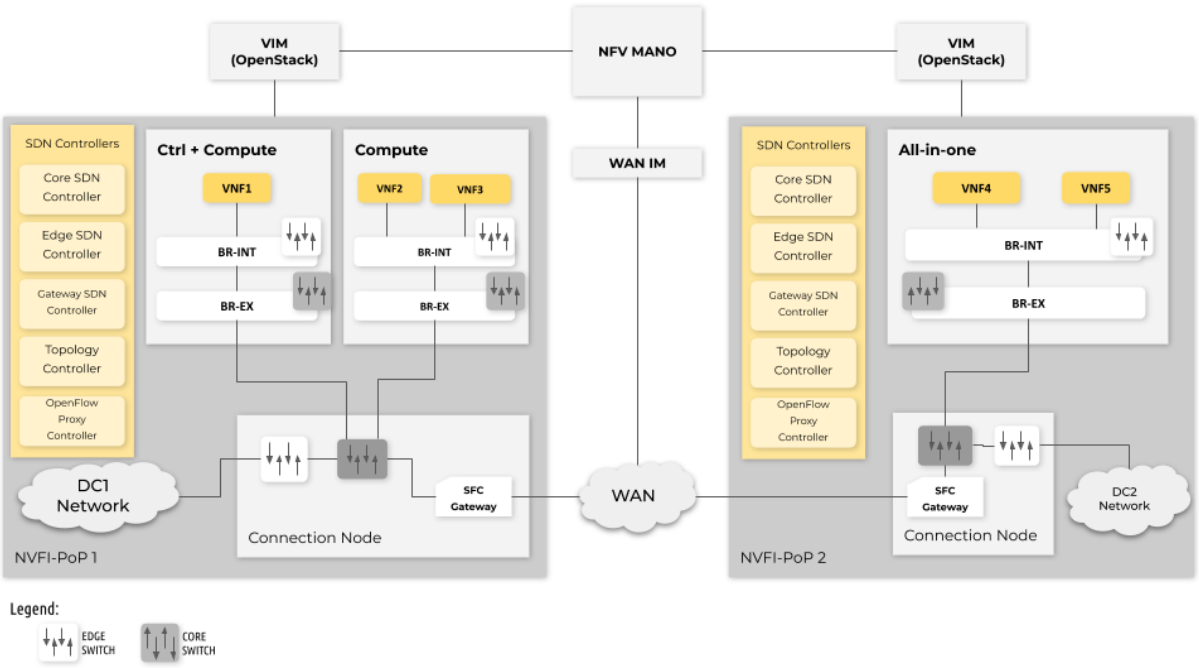


Figure 19 – Representation of the multi-domain SFC solution prototype.

4.2 The Data Plane implementation

The Data Plane is composed of two active elements following the KeySFC architecture: Core Switches and Edge Switches. This proposal adds another element (Gateway) to enable inter-NFVI-PoP SFC, which is a contribution of this work. These three elements together implement all the functionalities required by the architecture in multiples data centers. The next sections explore each element in the proposed network slicing architecture.

4.2.1 SFC Encapsulation

The SFC encapsulation is required not just to steer a packet through several SFs, but also it is required to isolate flows from different tenants and virtual services.

The **RNS-based method** follows KeySFC specifications (DOMINICINI et al., 2020). Routing in the `KeySFC` architecture uses the mathematical concept of the Residual Number System (RNS). In summary, consider $S = \{s_1, s_2, \ldots, s_N\}$ a set of N identifiers for core switches. Also, consider that these N switches belong to the path that the packet must follow from source to destination, and the values must be co-prime with each other. In addition, the set $P = \{p_1, p_2, \ldots, p_N\}$ represents the output ports to which packets will be redirected on each core switch, where p_i is the output port for the packet in s_i. Then, there is a number R, called the route identifier, which is a function of a set of core switches and their respective output ports for a specific path. The Chinese Remainder Theorem states that it is possible to reconstruct R from its residues. The forwarding process in each

core depends on a module operation (remainder of division) of the route identifier by the core node identifier.

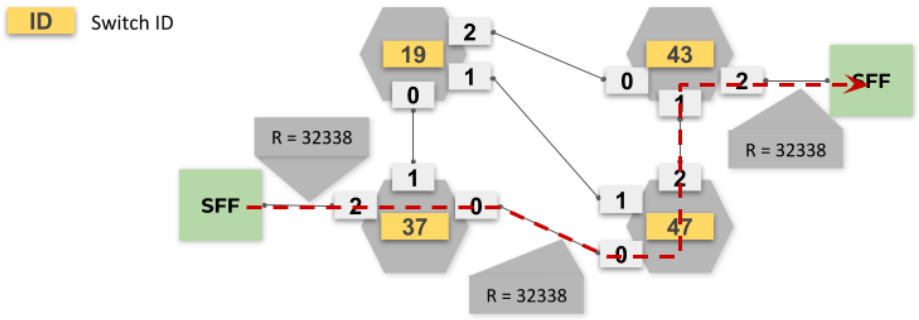


Figure 20 – Routing example using RNS.

Figure 20 shows an example of routing at the core network. It is assumed that the packet enters in the network via an edge switch where it receives the route identification, computed after collecting topological information. The switches that compose the route ($S = \{37, 47, 43\}$) and the output ports on each switch ($P = \{0, 2, 2\}$) are required. For this scenario the Route ID (R) is calculated as follows:

$$M = (37 * 47 * 43) = 74777$$

$$M_1 = M/s_1 = 2021 \qquad M_2 = M/s_2 = 1591 \qquad M_3 = M/s_3 = 1739$$

$$L_1 =< M_1^{-1} >_{37} = 23 \qquad L_2 =< M_2^{-1} >_{47} = 20 \qquad L_3 =< M_3^{-1} >_{43} = 34$$

$$R =< M_1 * L_1 * p_1 + M_2 * L_2 * p_2 + M_3 * L_3 * p_3 >_M$$

$$R =< 2021 * 23 * 0 + 1591 * 20 * 2 + 1739 * 34 * 2 >_{74777}$$

$$R = 32338$$

Upon reaching *Switch* 37 the module operation $< 32338 >_{37} = 0$ determines the output port. Then, the packet is forwarded to port 0, which is connected to Switch 47. Upon reaching Switch 47, the operation $< 32338 >_{47} = 2$ is performed. When exiting through port 2 of Switch 47 the packet arrives at Switch 43. Again, the current switch performs a module operation to find the next hop, the result of $< 32338 >_{43} = 2$ results in forwarding to port 2 that is connected to the destination, which can be an edge switch. In this edge switch, the packet needs to be restored to its initial state. When the packet exits the the core network and comeback to the edge network, the source MAC is restored to its original value. After restored, the packet goes to the SF destination.

We chose the field of the MAC address of the source to store the Routing Identifier. In other words, when the edge network forwards a packet to the core network, it contains

the route address on the field of the MAC address of the destination. This was an implementation choice since it does not require the creation of a new header. This field is 6 Bytes long. The first octet indicates the packet as an SFC RNS-based packet, and it is used by the network controller and during packet selection. The second and third octets store the Segment Identifier, which is the unique identifier that correlates the path with the service that the packet belongs. The remaining octets store the Routing Identifier.

The prototype was designed to primarily use RNS-based SFC encapsulation. In an effort to compare with other solutions we evaluate 2 other encapsulations: (i) list output ports and (ii) a push new MPLS header. Current solutions proposed in the literature and used in commercial implementations inspired the choice for these methods.

The second encapsulation method used is a **output port list**. The method implements a transport mechanism similar to Sourcey (JIN et al., 2016). In this method, labels are inserted into each packet to encode an explicit path to the destination. Switches in the path pop the output port from the encoding and use the label value as the output port. Figure 21 shows an example. Every switch pops the first label in the stack and forwards the packet to the corresponding switch port.

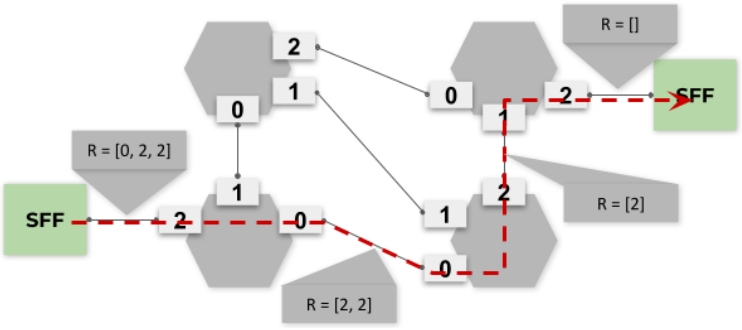


Figure 21 – Routing example using a list of output ports.

In the list-based method, the field MAC address of the source on the Ethernet header stores the routing information. From 6 Bytes of a MAC, the first octet indicates the packet as an SFC list-based packet, and the controller and the packet classification uses this information. The second and third octet store the Segment Identifier, which specifies the SFC that owns this packet, and the remaining 3 octets store the Routing Identifier. For this implementation, we set that the maximum number of ports for a Core Switch is 16, which is more than enough to connect all available physical interfaces and between switches. Consequently, it is only needed 4 bits to specify an output port. With 3 octets available to specify output ports, the maximum number of underlay hops available is 6, which is enough to represent the possibilities of path in the prototype.

The third method pushes a MPLS header to the packet. The MPLS header (Figure 22) contains, in the 20 bits label field, an identifier which specifies the SFC and path that the packet belongs. Using this unique identifier, flow table entries in specific

Edge and Gateway Switches select the packets of interest. The Edge and Gateway switches select the packet if the SFC hop's destination is attached to the switch. Then, the switch will forward the packet to the correct VNF or VM removing the encapsulation.

This procedure implements a transport mechanism similar to the Networking-SFC project (FOUNDATION, 2020), which is a tool that enables SFC in OpenStack clouds. MPLS SFC can't completely specify the underlay path in the same way as SSR methods. For this approach to work, the core network needs to acts as a Layer 2 hub. A Layer 2 hub forwards to all ports every arriving packet in core elements. Every hop in the SFC has an unique identifier, which is enough to deliver a packet from source to destination.

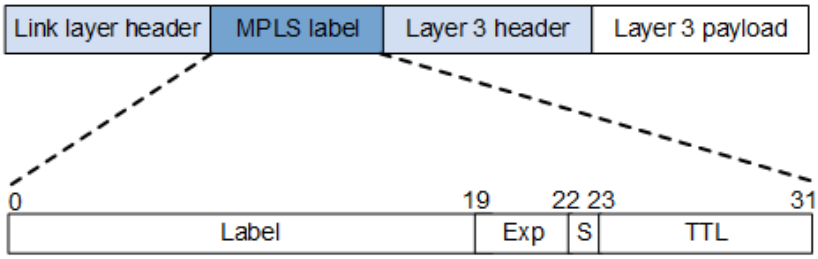


Figure 22 – MPLS header which is attached to every packet of flow in a SFC.

Table 1 summarizes the three methods, the encapsulation and the codification used to identify each packet.

Table 1 – SFC Encapsulations used in the prototype.

Method	Encapsulation	Codification
RNS	Source MAC	First octet 0x90
Output port list	Source MAC	First octet 0x91
MPLS	MPLS Header	Not applied

4.2.2 Edge Switches

An Edge Switch is responsible for dealing with the complexity of the SFC mechanism. It implements the flow classification and reclassification, SFC Proxy, SFF and encapsulation. OpenFlow flow entries are responsible for implementing these tasks. The Orchestrator manages the flow entries in the edge (see Section 4.3).

Every OpenFlow entry is composed of match and action fields. The match fields allow OpenFlow-enabled switches to select packets, and the actions act over the selected packets. The flow classification depends on the protocol. In this prototype, the Edge is capable of classifying the protocols: IP, TCP, UDP, ARP, and ICMP. Table 2 specifies the required fields and shows examples of OpenFlow entries for each of these protocols.

Table 2 – Supported protocols and its fields.

Protocol	Match Fields	Description
arp	arp_op	ARP opcode
	arp_spa	ARP source IPv4 address
	arp_tpa	ARP target IPv4 address
ip	ip	IP protocol value
	ipv4_src	IPv4 source address
	ipv4_dst	IPv4 destination address
udp	udp	IP protocol value
	ipv4_src	IPv4 source address
	ipv4_dst	IPv4 destination address
	udp_src	UDP source port
	udp_dst	UDP destination port
tcp	tcp	IP protocol value
	ipv4_src	IPv4 source address
	ipv4_dst	IPv4 destination address
	tcp_src	TCP source port
	tcp_dst	TCP destination port
icmp	icmpv4_type	ICMP type
	ipv4_src	IPv4 source address
	ipv4_dst	IPv4 destination address

After the classification, the next step is to deliver the packet to the SFF. The SFF is implemented using OpenFlow entries. The SFF has two options for arriving packets: if the destination SF is connected to the SFF, then the packet is forwarded to the correspondent SFC Proxy, otherwise, it encapsulates the packet with the chosen SFC Encapsulation and delivers to the network core which handles the packet transport. The packet will reach the SFF connected to the next SF in the SFC.

In this prototype, we always require the use of SFC Proxy. In this way, the NFV Orchestrator has full control over the whole process. Since we implemented the Edge Switch's functions using OpenFlow entries, we need to attach every switch to a set of controllers that implement these task. The set of Controllers is composed of: OpenFlow Proxy (Section 4.3.2.5), Topology Discovery (Section 4.3.2.4) and Edge Controller (Section 4.3.2.2).

4.2.3 Core Switches

Core switches are implemented using Open vSwitch bridges, in the same way of Edge Switches. However, Open vSwitch and OpenFlow tables do not support tableless Source Routing methods to forward packets. The control plane is responsible to, on demand, discovers the output port of packets and to install OpenFlow flow entries to forward the packets. The use of flow entries is necessary to avoid Open vSwitch software

customization. In production, the option could be to patch the Open vSwitch source code or to implement using FPGA, SmartNICs or programmable P4 switches.

Figure 23 describes the process when an SFC packet enters a Core Switch. First, it verifies if there is an entry that matches this flow. If more than one entry is valid for this flow, it chooses the entry that is more specific or with higher priority. If none entry matches, then the bridge drops the packet. Usually, flow tables have a default entry with priority 0 (lowest) that matches every packet. This default entry can forward packets in a broadcast mode or forward it to a controller.

Every packet has a unique identifier, for RNS-based and List-based we use the Source MAC to store it. For MPLS, we store the identifier at the MPLS Identifier located at the MPLS header.

In the MPLS-based method, the Core Network is configured in broadcast mode. The first action is to verify if there is a flow for this unique identifier. If not, it verifies the packet type using the source MAC first octet. This type verification is not required for MPLS packets, since for MPLS the core switches works in broadcast mode.

If the packet belongs to a SFC and there is no flow entry, the switch sends the packet to the Core Controller. The controller uses the encapsulation to build a flow entry and installs the flow entry in the Core Switch. After this process, there is a entry specific for this flow and the next packets will match this flow entry.

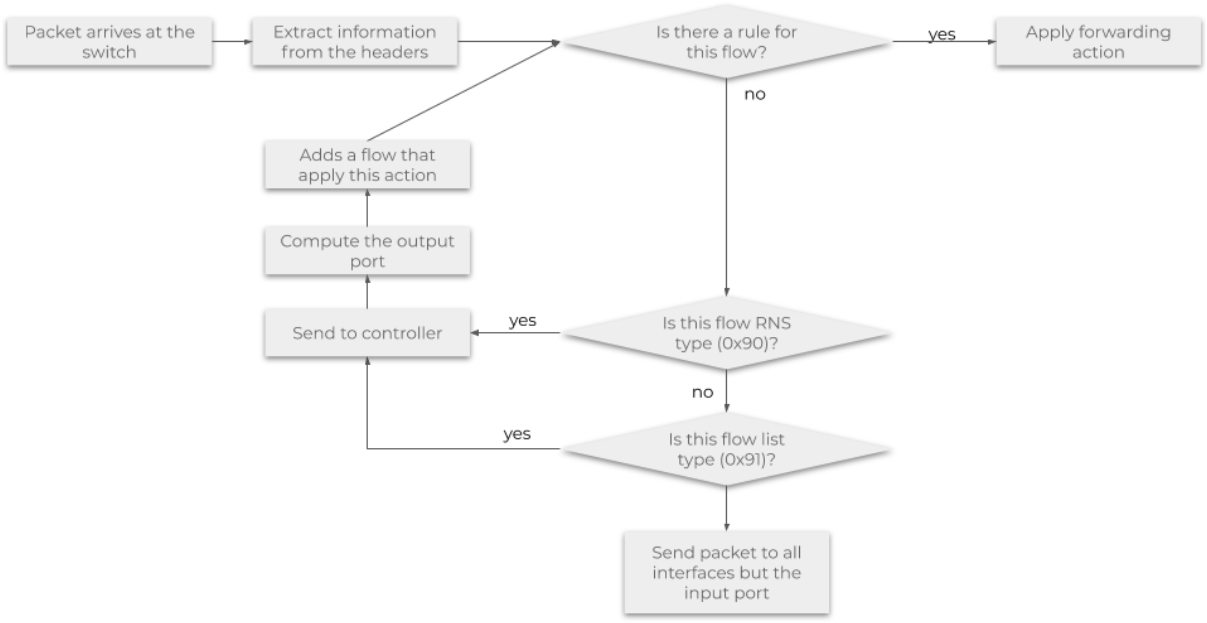


Figure 23 – Mechanism used by core switches to handle incoming packets.

4.3 The Control Plane implementation

The Control Plane proposed by ETSI is composed of Network Controllers, the NFV Management and Orchestration, and the Virtualized Infrastructure Manager (VIM). The next sections will detail implementation aspects for every component.

4.3.1 VIM

It was chosen OpenStack Cloud Framework to operate as VIM. OpenStack is a massive project the involves many companies that contribute to its source code. OpenStack is a framework that manage the virtualized infrastructure. It handles the virtualization of compute, network and storage resources.

The compute resources are provided by the interface that OpenStack does with each host hypervisor. OpenStack supports several hypervisors. In this implementation we chose KVM (KVM, 2016) since it is a open source project with notorious reliability. Therefore, the Service Functions are implemented using Virtual Machines.

Openstack also creates virtual networks that provide VM's connectivity. To create these networks, OpenStack uses Virtual Routers with Linux namespaces and for the L2 connection it uses Open vSwitch bridges that connect Virtual Routers to Host's physical interfaces and VM's virtual interfaces (also called taps).

While the Virtual Machines connect to the integration bridges (`br-int`), the physical interfaces connect to external bridges (`br-ex`). It is possible to leverage these characteristics and use the `br-int` bridges as Edge Switches and `br-ex` bridges as Core Switches. We connected the respective controllers to each bridge in every node, this connection bridge-controller defines the bridges' role in the topology.

OpenStack provides an extensive API that allows creating scripts to instantiate virtual machines, create networks, and other functionalities required by the Orchestrator. Therefore, OpenStack makes accessible an interface between physical resources and virtual resources dealing with the complexity, which are the responsibilities of a VIM in the ETSI architecture. In Figure 15, we can see two instances of OpenStack running. One has two nodes (one compute node and one compute and controller node), and the other has only one, which we call an All-in-one node (one node as compute and controller node).

The OpenStack deployment restores its original network settings, which is challenging, since our implementation modify some setting not native to OpenStack. We developed some scripts to automate the cloud configuration and also built containers to host the controllers. Consequently, we maintained a homogeneous deployment using the same configuration.

4.3.2 SDN Controllers

We used the Ryu framework to implement the network controllers of this prototype. Ryu is a framework written in Python and contains robust boilerplate code and allows quick deploy of controllers.

The prototype contains 5 types of Network Controllers with different responsibilities. We chose to create several instances because it facilitates development. Also, it makes it easier to classify the switches based on the controllers to which its associated, for instance, a bridge connected to a Core SDN Controller is a Core Switch.

Every controller is associated to an IP and two TCP ports, one for HTTP access of its API and other for controller communication. The virtual switches are *Open vSwitch 2.12 bridges*, which have support for *OpenFlow* from versions 1.0 to 1.5.

4.3.2.1 Core SDN Controller

As explained at Section 4.2.3, the legacy network based on OpenvSwitch bridges does not implement tableless strict source routing. Therefore, we use the Control Plane to perform module operations and to act on the core switches adding OpenFlow entries that apply the calculated output port for an specific flow.

In this context, the Core SDN controller has three responsibilities: (i) it provides an API that the NFV MANO queries to retrieve a list of Core Switches in the NFVI-PoP network; (ii) it adds a flow entry for RNS and List-based methods; (iii) for the RNS-based method, it stores the hash map which maps the Core Switch instance to the Switch Identifier. This map is used to compute the output port for each flow. The Core SDN Controller retrieves the routing identifier and together with the flow encapsulation computes the output port.

4.3.2.2 Edge SDN Controller

The Edge SDN Controller has 3 responsibilities: (i) it provides an API that the NFV MANO consults to query a list of Edge Switches in the NFVI-PoP network and; (ii) it implements a host discovery for the PNFs (Physical Network Functions).

The Host Discovery is applied when dealing with PNFs. These PNFs may not be capable of connecting to the Core Network using Source Routing protocols. One way to allow this compatibility is to connect an Edge Switch to a LAN in the DCN. The PNFs are also part of this LAN. However, under the same NFVI-PoP may exist several LANs and several different instances of Edge Switches interfacing both domains. When deploying an SFC, it is required to use this Edge Switch as SFF to steer packets through PNFs.

The Host Discovery can make ARP requests to the LANs attached to the Edge switches in the topology and find not just which switch but also which port of this switch

has a connection with the LAN where the PNF is located. The Orchestrator triggers the search for hosts via an API.

4.3.2.3 Gateway SDN Controller

The Gateway SDN Controller has two responsibilities: (i) it provides an API that the NFV MANO consults to retrieve a list of Gateway Switches in the NFVI-PoP network; (ii) it manages the life-cycle of flows entries responsible for the forwarding of packets to the WAN interface.

The Gateway Switches are under Gateway SDN Controller's responsibilities.

4.3.2.4 Topology Controller

Topology Discovery is a feature that concerns all switches(Edge, Core, and Gateway) and is a necessary task to implement SFC. Topology discovery is possible due to the LLDP protocol (*Link Layer Discovery Protocol*). Each *switch* discloses its identification to all its neighbors. The controller centralizes this information received from the network and builds the topology.

4.3.2.5 OpenFlow Proxy Controller

Finally, OpenFlow Proxy is a feature that allows the insertion of flow entries on topology switches via HTTP REST calls. The Proxy converts the rules received via REST calls to *OpenFlow* and sends them to the target *bridges*.

The OpenFlow Proxy Controller connects with Edge and Gateway switches because these are the switches that require population of flow entries which execute the functionalities presented in the SFC architecture. These entries perform flow classification, encapsulation or decapsulation, and forwarding.

4.3.3 NFV MANO

We merged the NFV Orchestrator with the VNF Manager creating the NFV MANO block. This block is responsible for the coordination of the interaction of all other functional blocks.

The NFV MANO implementation performs the management of life-cycle of Virtual Network Functions. VNF management requires the execution of the following tasks: (i) manage different domains; (ii) manage the life-cycle of VNFs; (iii) manage the life-cycle of Service Function Chaining. These three tasks require sub-tasks that make requests to the other functional blocks in the Control Plane architecture.

The Item (i) relates to the use of the API that the other blocks make available. Each domain expose APIs that allow the NFV MANO to perform some modifications in its infrastructure. The NFV MANO needs to know the endpoints (IP and port) for each Network Controller and the OpenStack Cloud credentials used to connect with the VIM.

The Item (ii) relates to the management of VNF's life-cycle. For this prototype, only creation and deletion are enabled. It was provided a Python API that allows an administrator to deploy VNFs. VNFs are virtual machines that host network applications. Figure 24 shows the steps towards positioning a VNF.

The NFV MANO receives from the administrator a requisition to create a VNF and then uses OpenStack's API to create the virtual machine. The application running in the virtual machine is defined by the image used to create the VM. The image is a file managed by the OpenStack module Glance. A VNF type is defined by an image and the number of virtual resources that this VM uses, which is called Flavor. The Flavor specify, among other information, CPU, RAM, and disk available to a VM.

The last information required to instantiate a VNF is the virtual network. In this prototype, we use a flat network without any tunneling. Adding tunneling would only complicate the task of maintaining retro-compatibility. The deletion of a VNF is the same as to delete a VM, which requires only the object obtained in the creation.

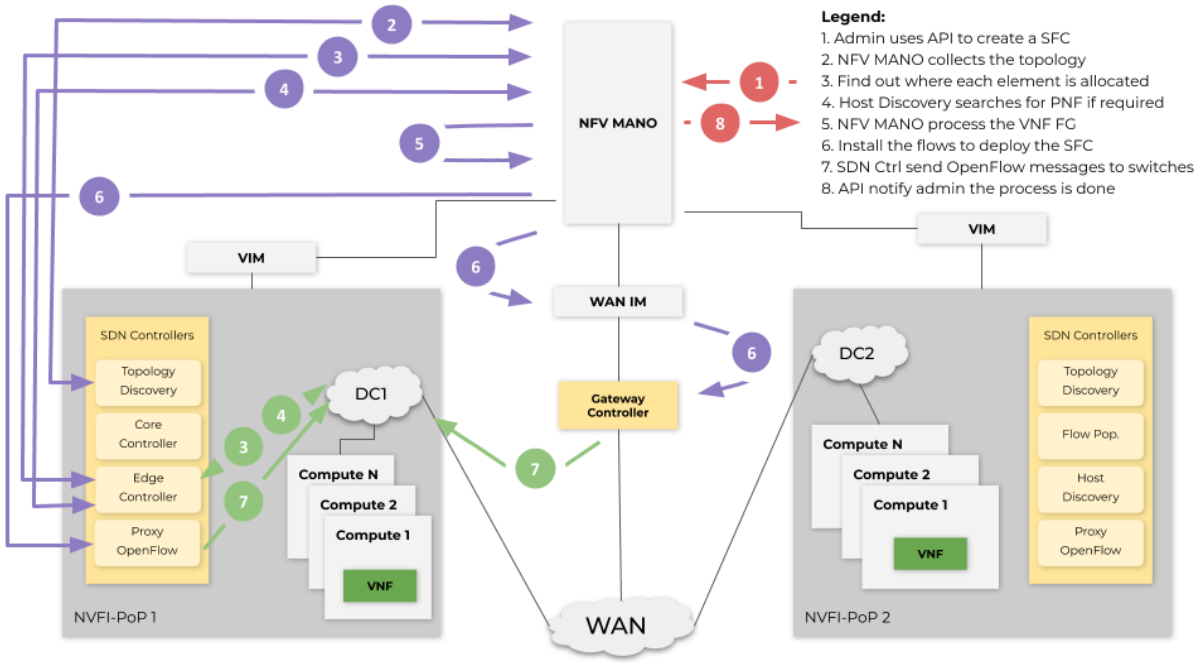


Figure 24 – Steps towards positioning a VNF.

The Item (iii) is the management of SFC's life-cycle. It is a more complex task that requires several steps to build a SFC that we show in Figure 25. First, the administrator specifies a Flow Classifier that selects a type of traffic to steer and the VNF FG, which is an ordered list of VNFs to which it desires to steer some flow to it.

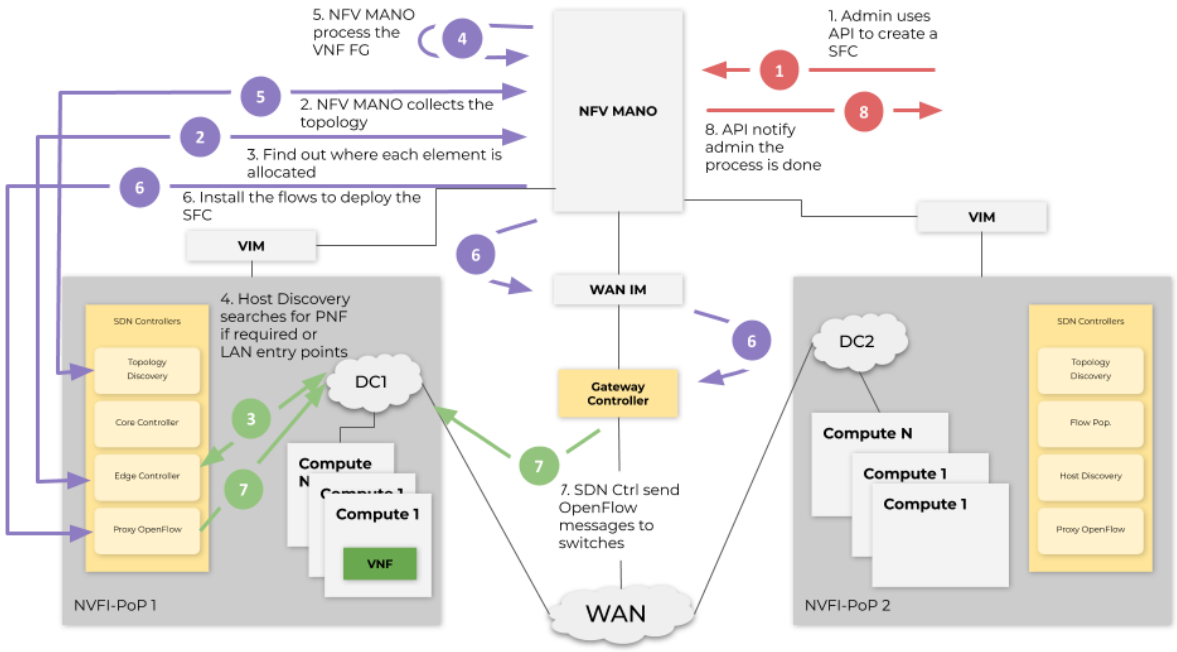


Figure 25 – Steps towards deploying SFC.

The NFV-MANO searches for the Edge Switch target for the Flow Classifier, the Orchestrator needs to know the entry points of the target flow. The procedure to find the target switch is to search if the source is a VM. In this case, the Orchestrator uses OpenStack API to find the node that hosts the required VM. Then retrieve the list of all Edge Switches in the NFVI-PoP target and obtain the switch related to this node.

If the source is a PNF, then it is required to make a Host Discovery. To make a Host Discovery, the NFV-MANO sends a Discovery requisition with the source IP to the Edge Controller. The Controller sends ARP requests to each port attached to itself with the solicited IP in the packet content. If it receives a response, then this Edge Switch is one target. We use the same procedure for the destination of the flow.

After the Orchestrator found the Edge Switches connected to the source and destination of the flow, the next task is to handle the sequence of SFs. In this prototype, we assume the SFs are only VNFs and, hence, only VMs allocated by the VIM. The NFV-MANO asks the appropriated NFVI-PoP VIM for the host of each VNF. The next step is to process the VNF FG received from the administrator.

Figure 26 shows a high-level understanding of how the Orchestrator processes an VNF FG. The administrator does not specify the entire path between two VNFs. Instead, it passes a high-level sequence of VNF objects obtained by the API. The VNF specified could be virtual machine already created in any NFVI-PoP, otherwise, it will be created.

The orchestrator iterates over the list of VNFs, splits the VNF FG in NFVI-PoPs. When the split process finishes, then the NFV-MANO attaches to each VNF the

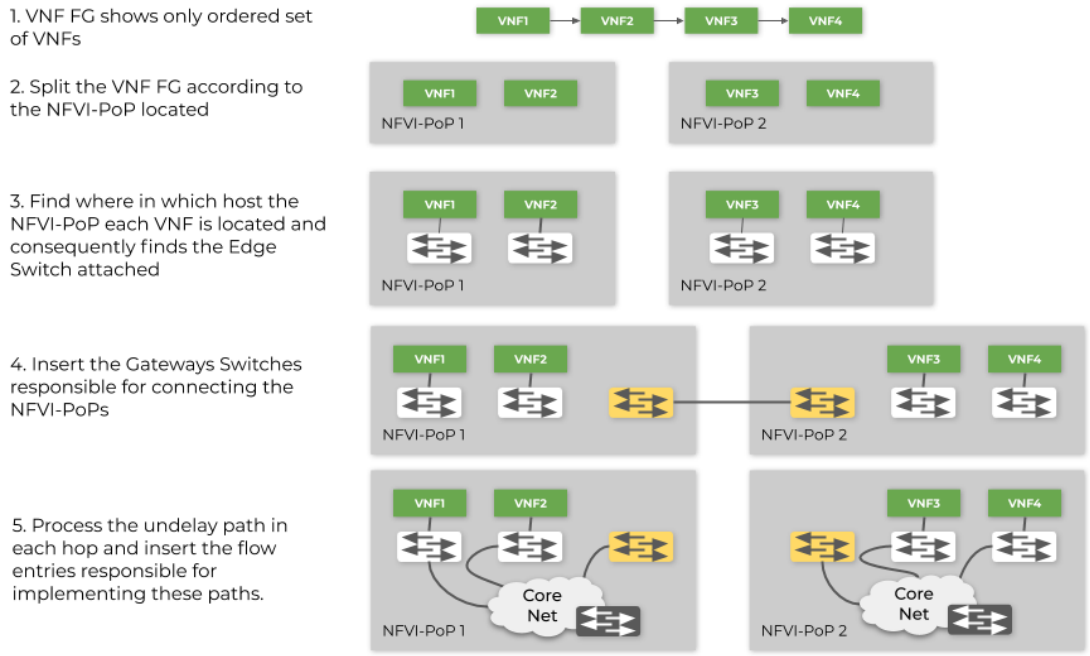


Figure 26 – Processing of a VNF FG by NFV MANO. In white, Edge Switches, in Dark Gray, Core Switches and in yellow, Gateway Switches.

correspondent Edge Switch found in the previous step.

Next, the NFV MANO selects the Gateway Switches that owns the interconnection between the NFVI-PoPs. The last step in the VNF FG processing is to build the path in the Core Network. For this prototype, the path selection uses the shortest path. Topology SDN Controller provides the topology, and, once acquired, the Orchestrator builds a connectivity matrix using Dijkstra's algorithm and generates a path in the core network. RNS-based and the list-based use the path to create the routing information in the encapsulation.

Then, the next step is to build the flow entries. The flow entries are almost the same for each protocol. But it is useful to show the generic configuration which is independent of the method. The flow entries can be categorized according to different scenarios:

1. **Scenario 1:** Source and destination are connected to the same Edge Switch;
2. **Scenario 2:** Source and destination are connected to different Edge Switches and are located in the same NFVI-PoP;
3. **Scenario 3:** Source and destination are connected to different Edge Switches and are located in different NFVI-PoPs;

Figure 27(a) shows the Scenario 1 which does not require to encapsulate the flow because the Edge Switch is capable of steering the traffic directly to the next VNF. In this scenario, the Orchestrator adds a unique entry that classifies the flow and deliver the

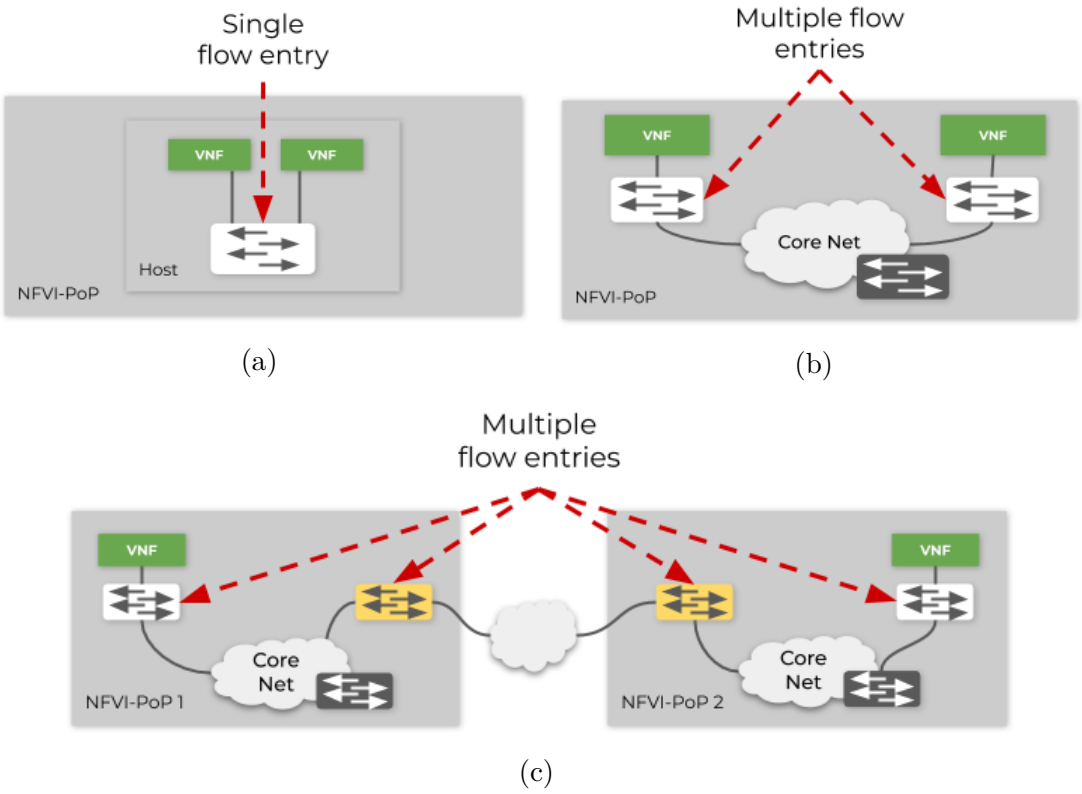


Figure 27 – Figures show scenarios for SFC. In white, Edge Switches, in Dark Gray, Core Switches and in yellow, Gateway Switches. (a) VNFs in the same Edge Switch. (b) VNFs in the different Edge Switches and in the same NFVI-PoP. (c) VNFs in the different Edge Switches and in different NFVI-PoPs.

packets to the SFC Proxy that rewrites the destination MAC with the VM's interface MAC. After changing the MAC, it delivers the packet to the VNF. By changing the destination MAC, the VNF is not aware of the steering.

For Scenarios 2 showed in Figure 27(b), the NFV MANO creates two flow entries. Figure 28 shows a high-level representation of flow entries in the scenario and the red line represents the process to which every packet is submitted. The first is installed at the source Edge Switch of the current hop and is composed of a classifier that selects the flow egress from a VNF or the flow's source. If the flow is classified when leaving the flow's source, then it is a classification. If it's leaving a VNF, then the correct term is a reclassification. The remaining of the first flow entry is the encapsulation, which changes the MAC Source for RNS and List-based or pushes an MPLS header for MPLS-based after the Edge switch sends the packet to the Core Switch attached to it.

The NFV MANO installs the second flow entry at the hop's destination. This flow entry match packets according to the unique identifier. For RNS and List-based, the unique identifier is the Source MAC, and in the case of MPLS is the ID field present in the protocol header. The Core SDN Controller, responsible for the core network, handles the flow entries at the transport network that allows the packet to reach its destination.

After Edge Switch matches the packets according to the unique identifier, it performs the SFF function and removes the SFC encapsulation and rewrites the Ethernet header with the correct information, which is the field of the destination MAC.

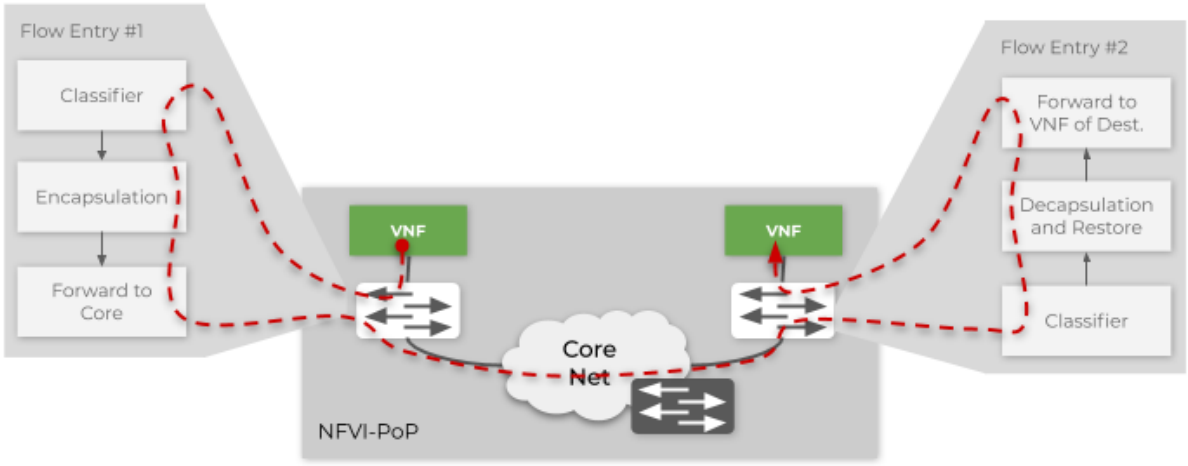


Figure 28 – Illustration of flows in Scenario 2.

Finally, for Scenario 3, showed in Figure 27(c), the Orchestrator needs to create four flow entries. Two at the NFVI-PoP 1 and two at the NFVI-PoP 2. At the source NFVI-PoP, the Orchestrator installs the flow entries at the Edge Switch connected to the flow's source and the other at the Gateway Switch that owns the virtual connection to the NFVI-PoP 2.

At the NFVI-PoP 2, a flow is installed at the Gateway and the Edge Switches. The flows installed at the Edge Switches in this scenario are the same as in Scenario 2. The flows installed at the Gateways are new and it is required to select the flow using the ID present at the header. With this ID, the Gateway redirects the flow without SFC encapsulation to the correct Virtual Link connected to the NFVI-PoP 2.

The Virtual Link between NFVI-PoPs is one contribution of the SFC Gateways to the architecture. The configuration allows the operator to ignore these connections and to trust the NFVO Orchestrator to use the connection in the SFC.

At the destination, the Gateway reclassifies the packets and applies the SFC encapsulation that transports the flow to the Edge Switch of destination in this hop. Technically, in this scenario, there are two hops, one from source Edge to Gateway switches and other from Gateway to Edge switches.

At the end of this process, the NFV MANO sends the flow entries via API to the OpenFlow Controller. The Proxy OpenFlow receives flow entries via HTTP requests in *JSON* format and converts to objects and send to the target switches.

5 Evaluation

This chapter presents the functional evaluation of the prototype and a discussion of the results. At the end, we discuss the possibilities that our proposal presents for load balancing strategies in different scenarios. This chapter aims:

- to present an experimental scenario used to evaluate the solution;
- to characterize the proposal using network metrics collected in the evaluation;
- to show that the solution is capable to attend to performance requirements;
- to explore the proposal in different load balancing scenarios inside a NFVI-PoP and between NFVI-PoPs.

5.1 Description

We instantiate the following chain of VM instances to demonstrate the operation of the proposed solution: SRC → DPI → NAT → Edge Firewall → DST, illustrated in Figure 29. The VNF type used is a simple forwarder, which means that all packets enter and leave a VM using the same network interface with no processing or modification. It is important to highlight here that it is not the objective of this work to evaluate the performance of different and specialized VNFs, since the focus of this work is on the evaluation of the SFC mechanism.

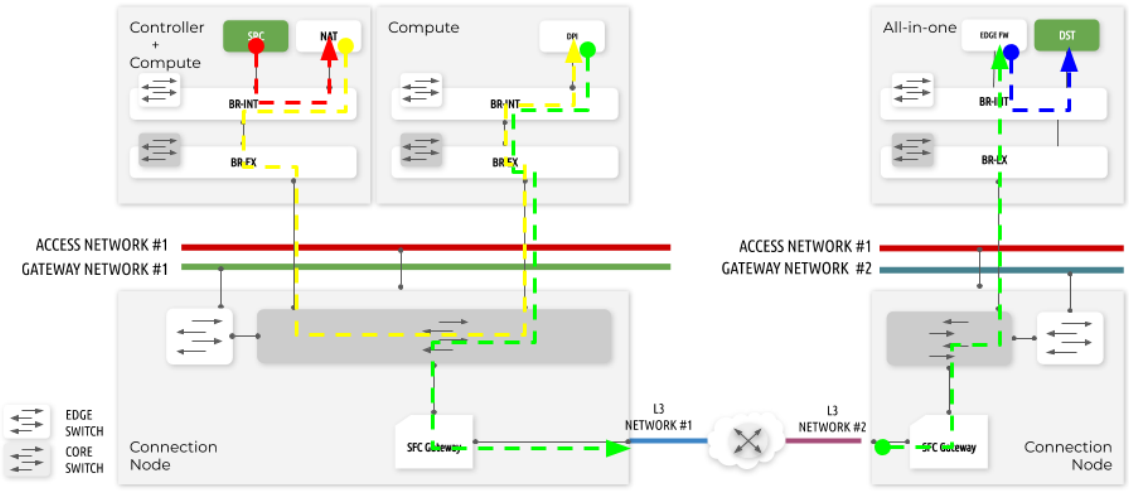


Figure 29 – SFC diagram with highlight for each hop.

This configuration was chosen to assess routing in different scenarios that explore all aspects of the solution, which are:

- 1st VNF Hop: between elements on the same server (SRC → NAT);
- 2nd VNF Hop: between VNFs on different servers (NAT → DPI);
- 3rd VNF Hop: between VNFs in different NFVI-PoPs (DPI → EDGE-FW) and;
- 4th VNF Hop: between a VNF and the flow destination (EDGE-FW → DST).

The three transport mechanisms for SFC presented earlier in Chapter 4 were implemented and evaluated. The NFV MANO and the script used to create the SFC are available on GitHub. Also, we made available an Ansible Script to deploy the network controllers and configure the physical interfaces and bridges according to an automatic inventory.

We collected latency, jitter and maximum throughput performance metrics in this scenario for each transport mechanism evaluated. These metrics are capable of characterizing the data center network and its capacity to provide an adequate infrastructure for SFC. It is a requirement for any architecture to do not degrade network metrics in order to proper host NFV applications. We also measured the same metrics without the SFC steering to understand its impact on the evaluated metrics.

Latency is a metric closely related, but different from Round Trip Time (RTT). Latency is not explicitly half of RTT, because the delay can be asymmetric between any two endpoints. However, since our objective is to understand the behavior and not in the value of the metric itself which it varies with the size of a chain or the distance between endpoints, therefore, we will refer RTT as latency in this text. We also performed 30 measurements of the RTT of an ICMP packet from VM SRC to VM DST. For the measurement of *jitter*, the `iperf` tool with UDP traffic was also executed 30 times. The results for latency are in Figure 30 and *jitter* in Figure 31.

The SFC adds latency which is expected since the packet has a longer path between source and destination. Another observation is that the greater the distance between the NFVI-PoPs, the greater the latency felt by the tunnel. For this prototype, both NFVI-PoP are in the same data center, which contributes to a smaller latency compared with geo-distributed data centers. In addition, as presented in Section 2.6, several publications study the problem of positioning to reduce the distance between VNFs when the service is sensitive to latency, and the network orchestrator can use these results.

For SFC scenarios, jitter tends to increase the longer the chain because the variance is proportional to the number of elements a packet needs to pass between source and destination. The results of *jitter* shows they are similar for every method. These jitter measured are small and don't compromise applications using network services. For instance, maximum jitter desired for VoIP calls is 30ms (CISCO, 2005) which is, approximately, 1000 times higher than the measured values.

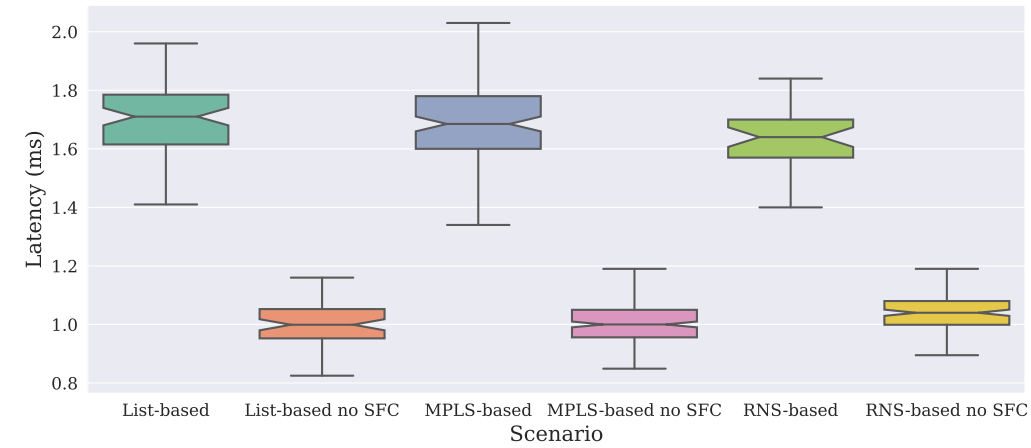


Figure 30 – Comparison between methods of latency measured between source and destination.

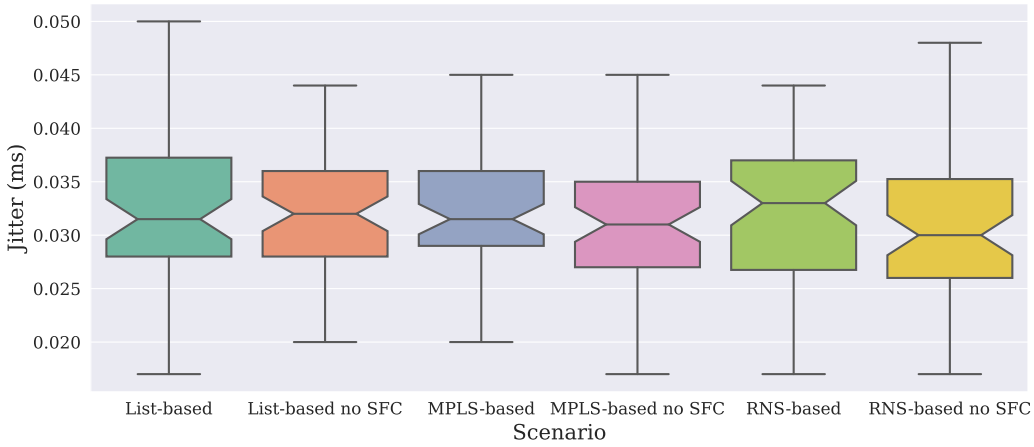


Figure 31 – Comparison between methods of jitter measured between the source and destination of the flow.

The results for the maximum throughput measured in the experiment are close to the nominal capacity of the tunnel which was set to 100 Mbps, as limited by the switch interface. This result shows that the encapsulation overhead in the proposal does not compromise the maximum throughput of the network.

The results presented in this section allow us to conclude that there is no latency and jitter degradation. Therefore, it is also possible to affirm that the proposal to perform chaining *inter NFVI-PoP* presented in this work is feasible to be implemented in real situations, using free and off-the-shelf software, widely used in the market.

Once the functional analysis was completed, we could explore the implications and possibilities from the adoption of the proposed architecture. The next section explores the implications and possibilities for inter and intra-NFVI-PoP traffic.

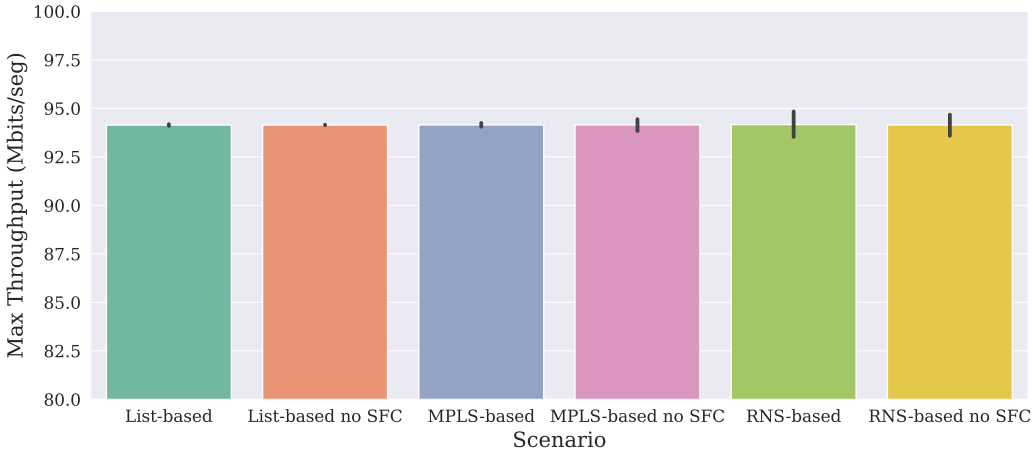


Figure 32 – Comparison between methods of maximum throughput SFC scenarios.

5.2 Discussion

In this section, we discuss load balancing inter and intra-NFVI-PoP and how these tasks can leverage from the proposed architecture. We enable some exciting load balancing possibilities, such as efficient intra- and inter- NFVI-PoP load balancing.

5.2.1 Intra NFVI-PoP load balance

A common characteristic between modern data center networks is the heavy traffic between the infrastructure servers. Regarding the flows coming from the servers. According to (KANDULA et al., 2009; BENSON et al., 2010): (i) at every $100ms$ new flows arrive at a data center; (ii) flows of up to 25 s are responsible for more than half of the traffic volume; (iii) and only 0.1 % of the total flows last more than 100 s, being responsible for almost 20 % of the data volume. Another important observation is that a small fraction of the links experiences losses much higher than the rest of the network. These characteristics in the nature of flows in a data center requires a flow balancing mechanism that is scalable, capable of handling the massive amount of incoming flows, and, also, able to improve the use of the resources by segregating elephant flows from other flows with different requirements. Segregation must occur dynamically, valid only during the life cycle of the flow. The migration of routes in the data center must occur transparently, causing the least possible interference to the quality of the experience of the users of the applications hosted in the data center.

In (VALENTIM et al., 2019), we proposed RDNA Balance as a load balancing solution that works on the data center networks architectures that use Source Routing as transport mechanism. By using Source Routing, it is possible to separate elephant flows from other flows dynamically. The network architecture divides the network in core and edge and use source routed transport. This separation between core and edge in both

architectures aims to improve the usage of the available bandwidth between the different paths of the data center and, at the same time, to guarantee the lowest possible latency for the different competing flows through a Load Balancing and Traffic Engineering solution.

The main features provided by the separation between core and edge together with source routing is the flexibility in the management of routes to implement load balance decisions in the network. We assume the network infrastructure is the same as the ones used in the architecture proposed in this work. The assumptions are: programmable edge and centralized control. Distributed load balancing approaches often make decisions based on local information about the use of network links (CAI et al., 2012) and are, therefore, not optimized (ALIZADEH et al., 2014). Centralized load balancing approaches have already been demonstrated in (ALIZADEH et al., 2014; RASLEY et al., 2014) as being the best options to optimize global use of the network.

The `RDNA Balance` mechanism handles flows inside a single data center. The idea is to optimize communication between VMs, however, this strategy fits for balancing each hop in an SFC. The discussion in this section shows that our load balancing strategy improves the availability of paths in the underlay network. An important aspect is that SFC flows are hired for the long term, and its nature (IP address and ports, protocols, bandwidth and latency required) can be previously and easily known in and SDN network. Therefore, it is easier to detect the nature of a flow and migrate in order to guarantee performance.

Moreover, routing at flow level using tables at the core of the network do not scale. Data centers receive a large number of new flows every moment, which implies an enormous amount of rules to be installed and states to be managed in each switch and router between the source and destination of a flow. This is one of the problems addressed in the `RDNA Balance` approach, as there is only a need to manage the states of the edge switches, with fewer elements to be managed. `RDNA Balance` is capable of installing the rules at the edge of the network with propagation time in the order of $1ms$ per rule, without significant losses in the migrated flow and gives high degree of programmability and versatility to the `RDNA Balance` solution.

In the following sections we tackle some evaluation scenarios where we can apply RDNA Balance to improve the virtualized services' performance. We have selected three scenarios where it is possible to improve network performance by flow migrations. These scenarios are: (i) elephant flow competing with mice flow; (ii) elephant flow competing with another elephant flow and; (iii) flow with strict reliability requirements, that is, low tolerance for packet loss. For each scenario we show that it is possible to improve one metric without affect the migrated flow.

We built a 2-tier fabric topology with two spine and three leaf nodes. This topology is more interesting than the topology used for functional evaluating since it presents

alternative paths between two servers. This network topology used for the evaluation differs from the prototype. In the evaluation prototype a single NFVI-PoP was deployed in a Source Routing Enabled Domain.

5.2.1.1 Elephant flow competing with mice flow

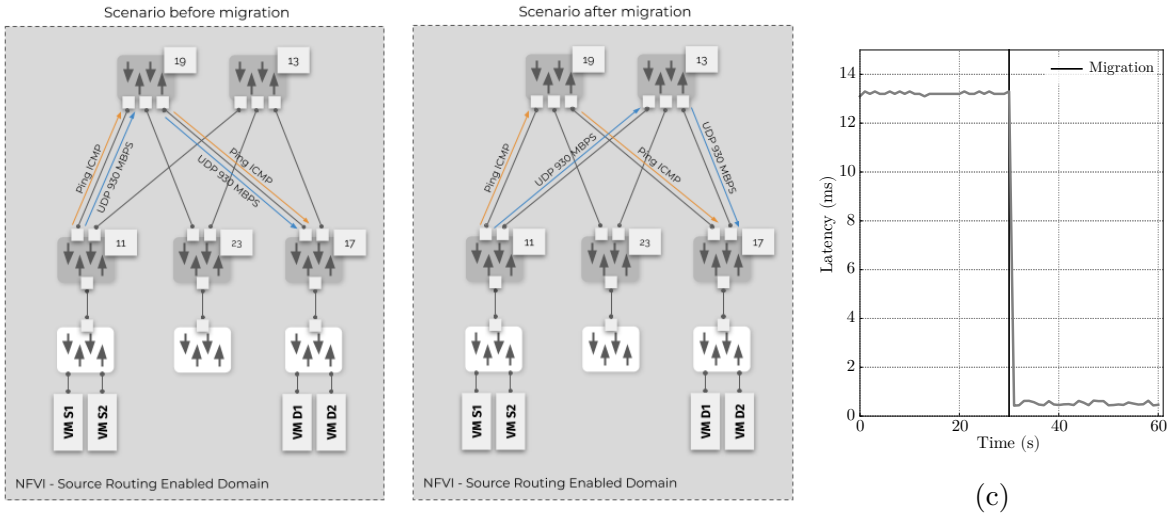
Consider the scenario proposed in Figure 33a. Two different flows share the same link: 1 UDP flow of $930Mbps$, characterized as an elephant flow and 1 ICMP message of type *ping*, sent every second, characterized as a mice flow. The UDP flow is generated by the *pktgen* tool, which is capable of generating packets of specific sizes at a constant rate. The *pktgen* generating the requested rate in the $81274pps$ experiment (packets per second) of $1518Bytes$ saturated the physical limit of the $930Mbps$ link. The rate and size of the package are in accordance with (BOLLA; BRUSCHI, 2006; POPOVICIU et al., 2008). Before migration, the UDP flow route is $VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD1$ and the ICMP flow is $VMS2 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD2$.

In the instant of time equal to $30s$, the action of the `RDNA Balance` solution is forced with the migration of the UDP flow route to the $VMS1 \rightarrow S_{11} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$. After the migration, the flows are separated, and there is no sharing of physical links between the two flows, as can be seen in Figure 33b. In this scenario, the bottleneck for latency is the physical link, as the virtual bus connecting the core switch with the edge has a much higher flow capacity. The tool *ping* obtains the measure of latency. The result is shown in Figure 33c with sample interval of $1s$. There is a relevant reduction of latency from 13 ms to 0.7 ms in the latency of ICMP traffic perceived in $VMD2$ at the time 30 s, due to the migration of the elephant flow to another route.

5.2.1.2 Elephant flow competing with another elephant flow

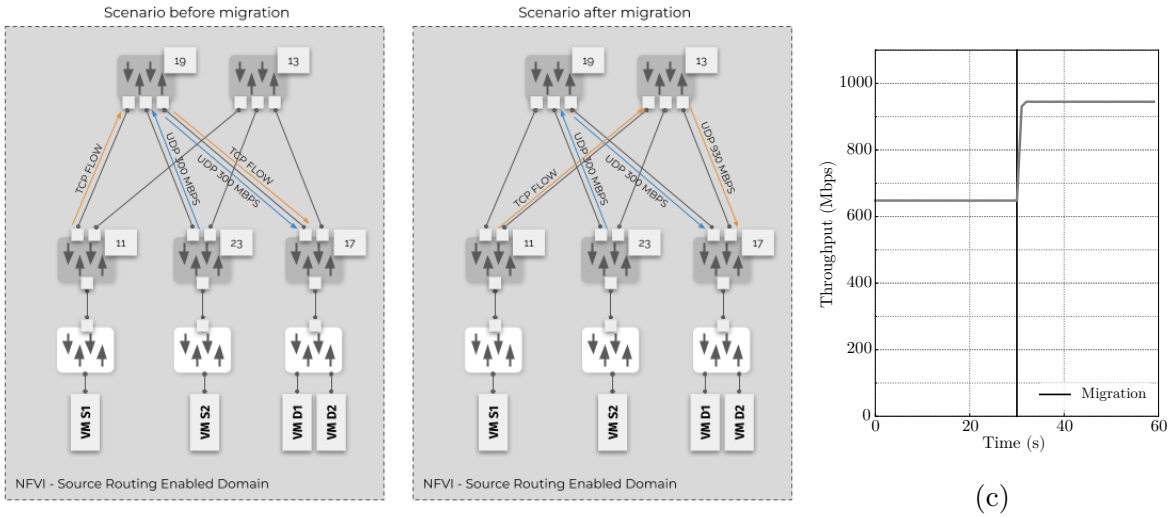
The test scenario in Figure 34a illustrates two elephant flows with different behaviors sharing the same link $S_{19} \rightarrow S_{17}$. In this scenario, the adaptive TCP flow ($VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD1$) is hampered by sharing the same link with the UDP flow ($VMS2 \rightarrow S_{23} \rightarrow S_{19} \rightarrow S_{17} \rightarrow VMD2$). The UDP flow occupies $300Mbps$ of the link bandwidth and the TCP flow occupies approximately $630Mbps$. The rate reached by the TCP flow is due to the limit of the physical interface of $1Gbps$ of the server and the rate of the UDP flow simulates an elephant flow. At $30s$, the `RDNA Balance` solution is forced into action and the TCP flow is migrated, isolating it from its competitor.

Figure 34c shows the perceived flow gain in $VMD1$ at the time $30s$ resulting from the migration. The absence of packet loss and re-transmission makes the flow rate stable during the migration would incur in a peak of latency during the migrations. The non occurrence of a peak proves the speed of the exchange mechanism that demonstrates a simple and efficient load balancing.



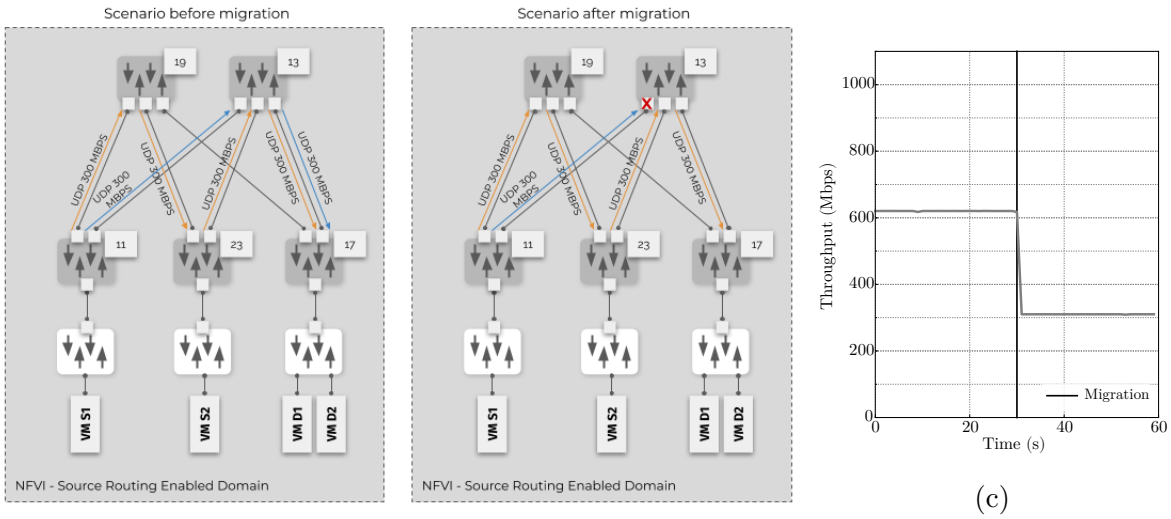
(a) (b) (c)

Figure 33 – Migration of a flow and the analysis of the impact on the perceived latency in $VMD2$ during the experiment. (a) Routes before migration (b) Routes after migration (c) Perceived latency in $VMD2$ during the experiment.



(a) (b) (c)

Figure 34 – Migration of a TCP flow and the analysis of the impact on the bandwidth. (a) Routes before migration (b) Routes after migration (c) Measure throughput in $VMD1$ (TCP traffic) during the experiment.



(a) (b) (c)

Figure 35 – Failure of a physical interface and the analysis of the impact on the bandwidth in a scenario of reliability. (a) Routes before the failure (b) Routes after the failure (c) Throughput perceived in $VMD1$ during the experiment.

5.2.1.3 Flow with delivery reliability requirements

In the experiment shown in Figure 35a, we decided to duplicate the flow by two distinct routes: $VMS1 \rightarrow S_{11} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$ and $VMS1 \rightarrow S_{11} \rightarrow S_{19} \rightarrow S_{23} \rightarrow S_{13} \rightarrow S_{17} \rightarrow VMD1$. At the time of 30 s, we simulated the failure of the physical interface, marked with an X in Figure 35b, making it impossible for one of the flows to arrive at the destination. As path redundancy was adopted, the flow in $VMD1$, which was previously 600 $Mbps$, started to receive 300 $Mbps$. The *iperf* tool generated this flow.

These functionalities of redundant paths are already supported by the architecture in the conception and are hard to implement in legacy networks and cannot offer such flexibility in the path selection.

5.2.2 Inter NFVI-PoP load balance

In the Section 5.2.1, we have discussed intra data center strategies for architectures that use source routing. However, since the connection between domains uses a virtual device with virtual connections, we can apply the concepts of SD-WAN to NFV, creating an inter-NFVI-PoP load balance strategy. A new trend in the industry is Software-Defined WAN (SD-WAN). SD-WAN leverages SDN to enable new features at WAN connections reducing costs. SD-WAN simplifies the application of bandwidth control and create private overlay networks over broadband shared connections.

The architecture for SFC in multiple data centers presented in this work introduces the concept of Gateway Switches. Gateway Switches are software-based components that allow fine-grained control over interconnections between different infrastructures.

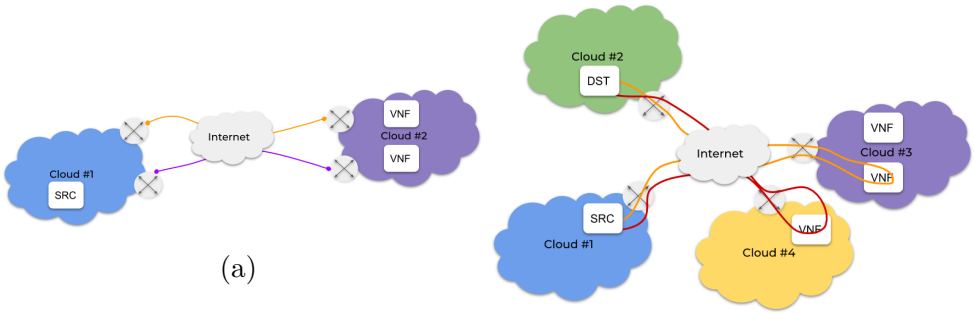


Figure 36 – Scenarios where multiple WAN connections are interesting. (a) Redundant connection between domains to improve link availability; (b) Redundant VNFs in different clouds.

Figure 36a shows a use-case of load balancing for WAN connections. The traffic between the two domains is critical when building distributed and virtualized services.

In this context, a good practice would be to have redundant WAN connections. However, managing which WAN connection to use adds complexity to the already hard task to manage a data center. Other remark is that, in traditional networks the control over the use of WAN connections is not capable of acting at flow level and involves the management of route entries in the routers. We propose the use of OpenFlow entries and the SFC Gateway to enable the control over the WAN connection for SFC flows.

An example would be some police that establishes a threshold latency for some SFC that contains VNF allocated in different data centers located in different geographic locations. The orchestrator chooses a WAN Gateway for this traffic and for some reason the latency in this WAN connection starts to increase. Upon reaching the established threshold, the load balance mechanism chooses another WAN connection available in the WAN-IM. The architecture proposes a centralized control to collect metrics and make decisions and the Gateway Switch is responsible for the flow selection and traffic steering to the correct WAN.

The same mechanism can be used to create redundant SFC segments instantiated in several data centers as shown in the Figure 36b. The redundancy might be necessary in the case of maintenance stops at the data center, for instance. In this case we expect that, during the migration, the SFC has minimum downtime, and our proposal can help in this task because there is no convergence time in the network when using source routing.

To prove the feasibility, we created two redundant connections between NFVI-PoP 1 and NFVI-PoP 2 using VLANs. From the point-of-view of the application, these connections are totally different. Figure 37 shows the prototype with the redundant connections and the flow using one of these links provisioned by the Orchestrator.

We want to show in practice a scenario where this approach benefits the application.

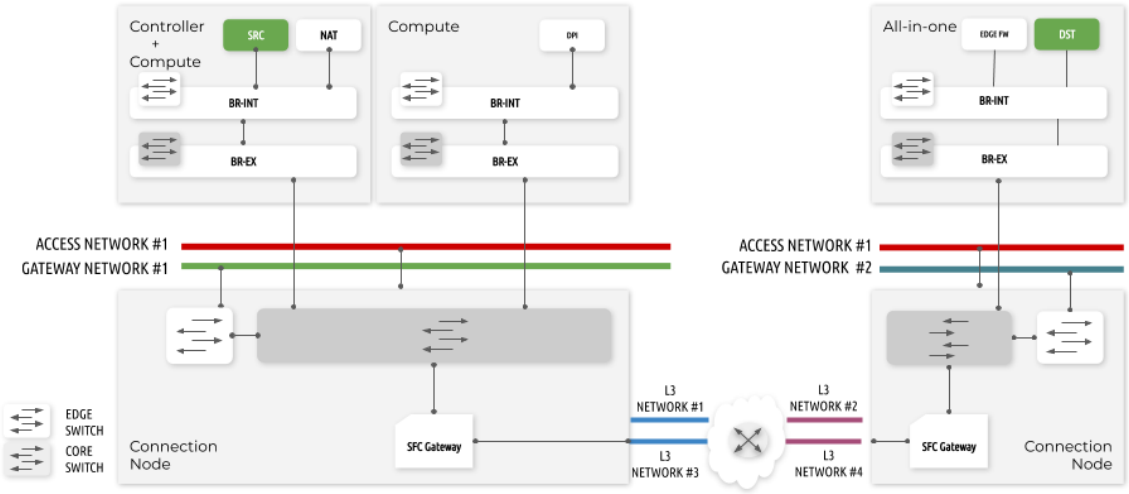


Figure 37 – Prototype with redundant connection between NFVI-PoPs.

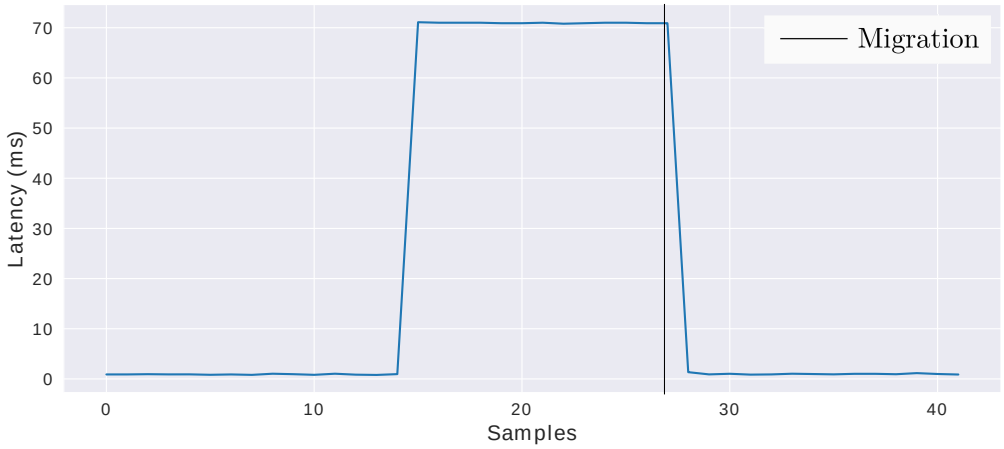


Figure 38 – Latency measured before and after WAN migration.

It is the case if a link failure or some problem with the provider causes some instability in the WAN link that incurs in an increase of latency measured by a monitoring service. Then, the NFV-MANO triggers a segment migration using the same mechanism discussed in the Section 5.2.1. Figure 38 shows the latency increasing that triggers the NFV-MANO, and the migration. After the migration, with minimum downtime possible, the latency is back to the same level before the link instability.

This result shows a shift of paradigm in load balancing for SFC between NFVI-PoPs, which enables sophisticated Traffic Engineering solutions. The proposal might enable other mechanisms like: duplicate a flow in different WANs to increase the chance of reaching the destination, or link fusion to increase the bandwidth available between two NFVI-PoPs.

6 Conclusions and Future Work

This chapter presents the conclusions of the current work and its contributions. Finally, its limitations and perspectives for future work are presented.

6.1 Conclusion

This work proposed, implemented, and evaluated an architecture for end-to-end network slicing using tableless strict source routing. Although the architecture proposes a solution for control and data planes considering the established standards from ETSI, it modifies and adapts some elements to build the solution. Our solution is able to deliver a single full-stack network slicing technology, and simplify the stitching of network slices from different cloud infrastructures. Also, the solution inherits from source routing methods the reduction of complexity in the core of the network, reducing the management of the states and the long convergence time, and speeding up the creation of new chains.

The main results of this work are: (i) development of an architecture that enables the stitching of network slices from multiple data centers in a efficient and flexible manner; (ii) development of an architecture that enables data centers to support tableless Strict Source Routing mechanisms as transport method and; (iii) implementation of both control and data planes in order to enable network slicing deployment into data center network infrastructures. Therefore, fulfilling all the established objectives.

As a limitation of our proof-of-concept implementation, we can point the use of table entries to perform the source routing. Although we avoided the use of custom software or hardware with this approach, we noticed some delay caused by the communication between the software switch and the SDN controller required to compute the output port. This process is required every time some new flow arrives in a core switch, but it can be completely avoided by the use of a (hardware or software) switch that natively supports source routing in the data plane.

Finally, this work is highly relevant mainly because it focuses on the mechanisms towards building virtualized network services in multiple data centers. In contrast, related works focus on optimization and positioning of network functions in multiple clouds, without proposing the actual SFC mechanisms for enabling end-to-end network slicing.

6.2 Publications

The research process performed in the context of the laboratory LabNERDS-UFES led to the publication of the following papers. The publications are listed below in chronological order:

1. **VALENTIM, R.**; DOMINICINI, C. K.; VILLACA, R.; RIBEIRO, M. R. N. ; MARTINELLO, MAGNOS ; MAFIOLETTI, D. . RDNA Balance: Balanceamento de Carga por Isolamento de Fluxos Elefante em Data Centers com Roteamento na Origem. In: SBRC, 2019, Gramado. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2019

2. **VALENTIM, R.**; DOMINICINI, C. K.; VILLACA, R.; RIBEIRO, M. R. N. ; MARTINELLO, MAGNOS ; MAFIOLETTI, D. . Roteamento na Origem como Facilitador do Encadeamento Multi-nuvem de Funções de Rede: Proposta e Implementação. In: SBRC, 2020, Gramado. Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, 2020 [**Accepted for publication**]

3. DOMINICINI, C. K. ; MARTINELLO, MAGNOS ; RIBEIRO, M. R. N. ; VASSOLER, G. L. ; VILLACA, R. S. ; **VALENTIM, R.** ; ZAMBON, E. . KeySFC: Agile Traffic Steering using Strict Source Routing. In: Symposium on SDN Research (SOSR), 2019, São Diego. Symposium on SDN Research (SOSR), 2019.

4. DOMINICINI, C. K. ; MARTINELLO, MAGNOS ; RIBEIRO, M. R. N. ; VASSOLER, G. L. ; VILLACA, R. S. ; **VALENTIM, R.** ; ZAMBON, E. . KeySFC: Agile Traffic Steering using Strict Source Routing for Enabling Efficient Traffic Engineering.

6.3 Future Works

For future works, it is essential to study methods for the real-time collection of network metrics. These metrics are essential to build a feedback system that allows the Orchestrator to migrate flows. Another interesting feature is to test other methods of tunneling. In this prototype, we used GRE. However, some other possibilities (VxLAN, Geneve, and GREoIPsec) can be explored.

There are several mechanisms and polices for the proposal of inter NFVI-PoP load balance that we did not explore. There is significant research related to this topic that involves the control and data planes and the algorithms to choose the best moment for a migration and algorithms to split the load in order to reach maximum utilization.

Lastly, the switches used in the network core of our proof-of-concept implementation were virtualized. There are several possibilities for offloading this solution to hardware.

Some examples are servers with SmartNICs and NetFPGAs. However, any offloading needs to address the communication of the hardware with the control plane implemented by the SDN controllers.

Bibliography

Abdullah, Z. N. et al. Segment routing in software defined networks: A survey. *IEEE Communications Surveys Tutorials*, v. 21, n. 1, p. 464–486, Firstquarter 2019. ISSN 1553-877X. Citado na página 39.

AFOLABI, I. et al. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys and Tutorials*, v. 20, n. 3, p. 2429–2453, 2018. ISSN 1553877X. Citado 3 vezes nas páginas 9, 35, and 36.

ALIZADEH, M. et al. CONGA: Distributed Congestion-Aware Load Balancing for Datacenters. *ACM SIGCOMM Computer Communication Review*, v. 44, n. 4, p. 503–514, aug 2014. ISSN 01464833. Disponível em: <http://dl.acm.org/citation.cfm?doid=2740070.2626316>. Citado na página 69.

BENSON, T. et al. Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 40, n. 1, p. 92–99, jan. 2010. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1672308.1672325>. Citado na página 68.

Bhagwat, P.; Perkins, C.; Tripathi, S. Network layer mobility: an architecture and survey. *IEEE Personal Communications*, v. 3, n. 3, p. 54–64, 1996. Citado na página 38.

BHAMARE, D. et al. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications*, Elsevier B.V., v. 102, p. 1–16, 2017. ISSN 01403664. Disponível em: <http://dx.doi.org/10.1016/j.comcom.2017.02.011>. Citado na página 40.

BOLLA, R.; BRUSCHI, R. Rfc 2544 performance evaluation and internal measurements for a linux based open router. In: IEEE. *High Performance Switching and Routing, 2006 Workshop on*. [S.l.], 2006. p. 6–pp. Citado na página 70.

CAI, Y. et al. *RFC 6754 - Protocol Independent Multicast Equal-Cost Multipath (ECMP) Redirect*. [S.l.], 2012. Citado na página 69.

CHAYAPATHI SYED F. HASSAN, P. S. R. *Network Functions Virtualization (NFV) with a Touch of SDN*. [S.l.]: Person, 2016. Citado 2 vezes nas páginas 9 and 31.

CISCO. *Troubleshooting and Debugging VoIP Call Basics*. 2005. <https://www.cisco.com/c/en/us/support/docs/voice/h323/14081-voip-debugcalls.html>. Citado na página 66.

CLAD, F. et al. *Segment Routing for Service Chaining*. [S.l.], 2018. Citado 2 vezes nas páginas 38 and 39.

DIETRICH, D. et al. Multi-Provider Service Chain Embedding With Nestor. *IEEE Transactions on Network and Service Management*, v. 14, n. 1, p. 91–105, 2017. ISSN 19324537. Citado na página 40.

DIMOLITSAS, I. et al. A Multi-Criteria Decision Making Method for Network Slice Edge Infrastructure Selection. p. 1–7, 2020. Citado na página 18.

DOMINICINI, C. K. et al. KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration. *Computer Networks*, v. 167, p. 106975, 2020. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S138912861930194X>. Citado 7 vezes nas páginas 19, 21, 38, 39, 42, 43, and 51.

FOUNDATION, L. *What Is Open vSwitch?* 2017. <http://docs.openvswitch.org/en/latest/intro/what-is-ovs/>. Citado 3 vezes nas páginas 9, 34, and 35.

FOUNDATION, L. *Open Network Operating System*. 2020. <https://www.onosproject.org>. Citado 3 vezes nas páginas 9, 18, and 32.

FOUNDATION, L. *Open Platform NFV Project*. 2020. <https://www.opnfv.org/>. Citado na página 18.

FOUNDATION, L. *OpenDaylight Project*. 2020. <http://www.opendaylight.org/>. Citado na página 18.

FOUNDATION, O. *OpenStack Documentation*. 2017. <https://docs.openstack.org/ocata/>. Citado 3 vezes nas páginas 9, 24, and 25.

FOUNDATION, O. *Service Function Chaining Extension for OpenStack Networking*. 2020. <https://docs.openstack.org/networking-sfc/>. Citado 2 vezes nas páginas 18 and 54.

GOMES, R. R. et al. Kar: Key-for-any-route, a resilient routing system. In: *2016 46th Annual IEEE/IFIP DSN*. [S.l.: s.n.], 2016. p. 120–127. Citado na página 38.

GUO, C. et al. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In: *Co-NEXT '10*. New York, NY, USA: ACM, 2010. p. 15:1–15:12. ISBN 978-1-4503-0448-1. Citado na página 38.

GUPTA, L. et al. COLAP: A predictive framework for service function chain placement in a multi-cloud environment. *CCWC 2017*, 2017. Citado na página 40.

Hermiyanty, Wandira Ayu Bertin, D. S. Multi-domain Network Virtualization. *Journal of Chemical Information and Modeling*, v. 8, n. 9, p. 1–58, 2017. ISSN 1098-6596. Citado na página 20.

JIA, W. A scalable multicast source routing architecture for data center networks. *IEEE Journal on Selected Areas in Communications*, v. 32, n. 1, p. 116–123, January 2014. ISSN 0733-8716. Citado na página 38.

JIN, X. et al. Your data center switch is trying too hard. In: *Proceedings of the Symposium on SDN Research*. NY, USA: ACM, 2016. (SOSR '16), p. 12:1–12:6. ISBN 978-1-4503-4211-7. Disponível em: <http://doi.acm.org/10.1145/2890955.2890967>. Citado 4 vezes nas páginas 16, 38, 43, and 53.

JYOTHI, S. A. et al. Towards a flexible data center fabric with source routing. In: ACM. *1st ACM SIGCOMM*. [S.l.], 2015. p. 10. Citado 2 vezes nas páginas 17 and 37.

KANDULA, S. et al. The nature of data center traffic. In: *Proceedings of the 9th ACM SIGCOMM conference on IMC '09*. New York, New York, USA: ACM, 2009. p. 202. ISBN 9781605587714. Disponível em: <http://portal.acm.org/citation.cfm?doid=1644893.1644918>. Citado na página 68.

KHONDOKER, R. et al. Feature-based comparison and selection of software defined networking (sdn) controllers. In: IEEE. *Computer Applications and Information Systems (WCCAIS), 2014 World Congress on*. [S.l.], 2014. p. 1–7. Citado na página 33.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, IEEE, v. 103, n. 1, p. 14–76, 2015. Citado 2 vezes nas páginas 15 and 31.

KVM. *Main Page — KVM,*. 2016. [Online; accessed 5-June-2020]. Disponível em: <https://www.linux-kvm.org/index.php?title=Main_Page&oldid=173792>. Citado 2 vezes nas páginas 34 and 57.

MAHMOODI, T. et al. Network Functions Virtualisation (NFV); Management and Orchestration. *IEEE Communications Standards Magazine*, v. 1, n. 4, p. 60, 2017. ISSN 24712825. Citado 3 vezes nas páginas 21, 28, and 29.

MARTINELLO, M. et al. Keyflow: A prototype for evolving SDN toward core network fabrics. *Network, IEEE*, v. 28, p. 12–19, 03 2014. Citado 2 vezes nas páginas 17 and 38.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 2, p. 69–74, 2008. Citado na página 32.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011. Citado na página 24.

MIJUMBI, R. et al. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, v. 54, n. 1, p. 98–105, 2016. ISSN 01636804. Citado 2 vezes nas páginas 15 and 46.

NGMN. Description of Network Slicing Concept by NGMN Alliance. *Ngmn 5G P1*, v. 1, n. September, p. 19, 2016. Disponível em: <https://www.ngmn.org/uploads/media/160113_Network_Slicing_v1_0.pdf>. Citado 3 vezes nas páginas 15, 36, and 41.

ORACLE, V. Virtualbox user manual. *URL: http://www.virtualbox.org/manual/: 2016-05-01)*, 2011. Citado na página 34.

PEI, J. et al. Efficiently Embedding Service Function Chains with Dynamic Virtual Network Function Placement in Geo-Distributed Cloud System. *IEEE Transactions on Parallel and Distributed Systems*, IEEE, v. 30, n. 10, p. 2179–2192, 2019. ISSN 15582183. Citado na página 20.

PIGNATARO, J. M. H.; CARLOS. *Service Function Chaining (SFC) Architecture*. [S.l.], 2015. v. 53, n. 9, 1689–1699 p. Disponível em: <https://www.rfc-editor.org/info/rfc7665>. Citado 4 vezes nas páginas 18, 29, 41, and 44.

POPOVICIU, C. et al. *RFC 5180 - IPv6 benchmarking methodology for network interconnect devices*. [S.l.], 2008. Citado na página 70.

QUINN, P.; GUICHARD, J. Service Function Chaining: Creating a Service Plane via Network Service Headers. *Computer*, v. 47, n. 11, p. 38–44, nov 2014. ISSN 0018-9162. Citado 2 vezes nas páginas 18 and 39.

RASLEY, J. et al. Planck: Millisecond-scale Monitoring and Control for Commodity Networks Jeff. In: *Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14*. New York, New York, USA: ACM Press, 2014. p. 407–418. ISBN 9781450328364. Disponível em: <http://dl.acm.org/citation.cfm?doid=2619239.2626310>. Citado na página 69.

ROSEN, E.; VISWANATHAN, A.; CALLON, R. Multiprotocol label switching architecture," rfc 3031. Citeseer, 2001. Citado na página 43.

SUN, G. et al. Energy-efficient and traffic-aware service function chaining orchestration in multi-domain networks. *Future Generation Computer Systems*, Elsevier B.V., v. 91, p. 347–360, 2019. ISSN 0167739X. Disponível em: <https://doi.org/10.1016/j.future.2018.09.037>. Citado na página 40.

SUNSHINE, C. A. Interconnection of computer networks. *Computer Networks (1976)*, v. 1, n. 3, p. 175–195, 1977. ISSN 03765075. Citado na página 37.

SUNSHINE, C. A. Source routing in computer networks. *ACM SIGCOMM Computer Communication Review*, v. 7, n. 1, p. 29–33, jan 1978. ISSN 01464833. Disponível em: <http://portal.acm.org/citation.cfm?doid=1024853.1024855>. Citado na página 37.

TSO, F. P.; JOUET, S.; PEZAROS, D. P. Network and server resource management strategies for data centre infrastructures: A survey. *Computer Networks*, v. 106, p. 209 – 225, 2016. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128616302298>. Citado na página 17.

VALENTIM, R. et al. RDNA Balance: Balanceamento de Carga por Isolamento de Fluxos Elefante em Data Centers com Roteamento na Origem. In: *SBRC 2019*. SBC, 2019. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/7418>. Citado 2 vezes nas páginas 37 and 68.

VU, A. V.; KIM, Y. An implementation of hierarchical service function chaining using OpenDaylight platform. *IEEE NETSOFT 2016*, p. 411–416, 2016. Citado na página 40.

XENSERVER. 2017. Disponível em: <https://xenserver.org/>. Citado na página 34.

YANG, Z. et al. Software-defined wide area network (sd-wan): Architecture, advances and opportunities. In: IEEE. *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. [S.l.], 2019. p. 1–9. Citado na página 48.

YI, B. et al. A comprehensive survey of Network Function Virtualization. *Computer Networks*, v. 133, p. 212–262, 2018. ISSN 13891286. Citado 3 vezes nas páginas 9, 27, and 28.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, Springer, v. 1, n. 1, p. 7–18, 2010. Citado na página 24.