TÉCNICO LISBOA

Analysis and Evaluation of Segment Routing and Virtual Network Functions in Network Softwarization Scenarios

Vasco Resendes Moniz da Cunha

Thesis to obtain the Master of Science Degree in

Telecommunications and Computer Engineering

Supervisor: Prof. Fernando Henrique Corte Real Mira da Silva

Examination Committee

Chairperson: Prof. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Fernando Henrique Corte Real Mira da Silva
Member of the Committee: Prof. Rui Jorge Morais Tomaz Valadas

January, 2021

Acknowledgments

I would like to pay my special regards to my professor, Dr. Fernando Mira da Silva, for his consistent support throughout the entire project, his insights, and guidance. I would also like to thank my parents, as well as my sister, who offered me unlimited moral support and helped me to persevere during these hard times and provided me with the emotional support I needed to accomplish this goal. Special thanks to my university, Instituto Superior Técnico, for providing me with the opportunity to develop my research and providing me with the environment to learn and better myself.

Resumo

Esta tese aborda as tecnologias de Segment Routing, Software Defined Networks, e Network Function Virtualization na tentativa de demonstrar a importância da conjugação das três nas redes de telecomunicações atuais.

Network Function Virtualization (NFV) tenciona trazer flexibilidade e programabilidade às redes de informação, permitindo substituir o hardware traditional por componentes de software, a correr em servidores virtuais. Especificamente, o NFV separa os serviços de rede da infraestrutura de hardware, permitindo o controlo e execução de software executado em servidores virtuais. Por outro lado, o Segment Routing (SR) é um novo paradigma de encaminhamento de pacotes em cenários de IPv6 e MPLS, baseando-se na ideia de "source-routing", onde um nó de origem (source) adiciona a um pacote uma sequência ordenada de de segmentos, que deve ser respeitada para o pacote chagar ao seu destino.

Com a atenção dos Fornecedores de Acesso à Internet (ISPs) se afastam das implementações de redes que se focam em hardware especializado, as Redes Definidas por Software (SDN) ganham cada vez mais relevância. Os benefícios que apresentam são vários, para ambos clientes e fornecedores, tais como a redução de custos e a maior velocidade das redes. Mesmo assim, é possível que novas soluções possam trazer com elas algumas desvantagens ou defeciências, que possam ter ou não solução já estudada.

Esta tese tem como objetivo testar a implementação do Segment Routing, em ambiente de SDN, nos quais o SR tem ganho tração, tais como centros de dados (Datacenters), integrando também serviços de NFV.

Palavras-chave: SDN, SR, NFV, SREXT, Spine-Leaf, NG-SDN

Abstract

Network Function Virtualization (NFV) aims to bring flexibility and programmability to the network layers, enabling the replacement of traditional network appliances for software components, running in virtual servers. Specifically, NFV decouples the network services from the underlying (dedicated) hardware infrastructure, enabling the designing, managing and execution of complex network services on software running on virtualized servers. On the other hand, Segment Routing (SR) is a new paradigm for routing in IPv6 and MPLS scenarios and relies on the source routing concept, where a source node adds to a packet an ordered sequence of segments that must be followed in order to the packet reach its final destination.

As focus from ISPs and network providers stray from hardware focused network implementations, Software Defined Networks have begun to make their way into the spotlight. The presençe of SDN and SR is expanding into Database centers, and its influence is assumed to reach worldwide networks with the emergence of 5G. The benefits entailed are various, both for the providers and the customers, such as cost reduction and network speed. Still, it is common for state-of-the-art solutions to reveal some shortcomings at some point that may or may not have been thought about beforehand.

This thesis manages to reproduce a successful case of the usage of segment routing alongside SDN in a datacenter, an environment where its benefits can be exploited. There was also an attempt to apply the same technologies to a broader network, as well as introducing Virtual Function Chaining in the network.

Keywords: SDN, SR, NFV, SREXT, Spine-Leaf, NG-SDN

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The complexity of modern networks has caused enterprises to incorporate network virtualization models into their traditional networks. Such an approach offers an easier management and control of a network, along with the tempting benefit of reducing network costs, both Capital and Operational Expenses (CAPEX and OPEX, respectively) [2]. It also shows promise in regards to increasing speed, agility, flexibility, and other parameters that are desired in every network [3]. But this potential is not always fulfilled . Cases have arisen where unexpected delays have occurred, and recent research suggests that such performance incoherences can sometimes be traced back to subpar combinations of Virtual Network Functions (VNF) in Service Function Chaining environments (SFC) [15]. For these technologies to fully be able to bring about their full benefits and potential, there is a need to understand how SFC and virtual environments in general are influenced by the virtual functions running in the network.

Having found out how the presently most used SDN version of policy-based routing, SFC, can bring about limitations to the network, efforts have been and are continually made to find a way to prevent the network from being affected by these setbacks. Researchers have put forward several attempts to do so, such as altering the function chaining process or creating tools to better understand what is causing performance degradation and where it is happening [15] [10]. Based on the recent research gathered and the possible improvements to the identified problem, the development of this thesis will revolve around the usage of Segment Routing to improve the routing capabilities of the present SFC in regards to VNF dissemination. The development of this thesis will have as backbone the usage of Software Defined Networks (SDN) and Segment Routing (SR) technologies, specifically ONOS. An attempt to integrate Network Function Virtualization (NFV) into the SDN + SR environment will be made, to see how the sequencing of the VNFs could be of use to the network and its overall performance.

1.2 Goals

The goal of this work is to answer the following research questions:

- How far has Segment Routing been developed and what are its use cases presently?
- Can enterprises leverage the use of Segment Routing? How can they benefit?
- What are the advantages of using Segment Routing with NFV? Is there a performance and/or latency increase?

During this thesis, an attempt was made to study the impacts of SR in SDN environments, more specifically, how the usage of Segment Routing in an IPv4 MPLS infrastructure can impact the network. SR in IPv6 has been a target of Virtual Network Function Chaining as a possible solution for achieving a better performance, but no conclusions were drawn regarding its stability [19]. Other relevant questions may surface, and there is the possibility that this work may delve further into the advantages (or disadvantages) of using Segment Routing in the multiple scenarios in which the routing technique will be implemented.

1.3 Structure

In the second chapter, this introductory report to the thesis presents an informative description of the backbone technologies and state-of-the-art of the tools and networking concepts which are essential to the development and understanding of the thesis.

The following chapter, ”Segment Routing Applications” focuses on how the solutions obtained try to answer the questions raised previously. It makes use of the achievements during the implementation of the project and provides a conceptual depiction of how said solutions can be used to answer those questions.

Chapter 4, ”Segment Routing Implementations”, addresses the implemented solutions, tests and results obtained during the development of the thesis, as well as the setbacks and challenges faced.

The document ends with chapter 5, expanding on the significance of the thesis regarding present technological needs and achievements, as well as what can be and is being done towards the development of similar implementations and the technologies involved.

Chapter 2

Background

Network softwarization has already captured the interest of both researchers and networking enterprises all over the world. Although the speed at which its core ideas are accepted and viewed as a significant improvement when comparing with the present hardware-based networks, the research into the less obvious particularities of the subject can still be considered somewhat superficial.

One of the most obvious benefits of the migration from legacy networks into a centralized approach are the expenses an ISP will face in the long term management of said networks, regarding scalability and flexibility [1].

Although a consensus has been reached about some of the more straightforward benefits of progressing into these kinds of software-based networks, the analysis of the optimal usage of this technology is still ongoing. Researchers have fiddled with many topics such as resource management, software interoperability and cost efficiency [2], and although it is globally accepted that the usage of SDN results in the reduction of costs, scalability improvement, and provides greater ease in managing the network [3], the technology's main potential lies in the possibility of its integration with other technologies, such as Segment Routing and Network Function Virtualization.

MPLS is a tunneling mechanism used in today's networks, providing a traffic steering functionality and secure encapsulation. MPLS VPN is an example of such encapsulation, while the famous RSVP-TE (Resource Reservation Protocol - Traffic Engeneering) is responsible for steering traffic in the network [7]. ISPs have found success in providing affordable VPNs and secure connection with the usage of MPLS, and the usage of VPN usage is still growing.

Segment Routing is considered to be an improvement over the classical MPLS encapsulation. Amongst some differences that give SR a reasonable edge over the classical protocol (which will be approached later in this paper), the most obvious difference between both implementations is the absence of the Label Distribution Protocol, which is close to being the defining feature of MPLS. SR has been a subject of interest for quite some time, and even more so when talking about centralised network controlling and

SDN [4]. Although some of the most commonly used hardware can face compatibility issues with SR, that lack is not really a major concern to the development of the thesis, as no implementation will not be dependant on a physical topology. Considerable research has been conducted into SR and a few tutorials have been made available (mostly by Cisco and Juniper) to help with its implementation, using both open source or copyrighted appliances.

2.1 Segment Routing Technology Overview

SR is a source routing technique allowing for both edge routers and hosts in a network to send data through several segments composed of multiple hops [7]. It aims to address the drawbacks of IP/MPLS networks, by enhancing packet-forwarding behaviour.

As stated previously, strictly speaking Segment Routing is not a new technology, and can be described better as a routing-based tunneling technique derived from the long-established MPLS encapsulation mechanism. The reason for its recent rise to stardom can mostly be attributed to the mutual compatibility of the routing technology and the development of network softwarization. Although Segment Routing can be used to leverage networks without centralized controllers, the technology's improvements over classic MPLS and the Reservation Protocol with Traffic Engineering (RSVP-TE) fully manifest themselves in a network with centralized management.

2.1.1 Multi Protocol Label Switching

To better understand SR, we must first delve into one of its bootstrap technologies: MPLS encapsulation.

Traditionally, MPLS makes use of the Label Distribution Protocol (LDP) as its main mechanism to encapsulate the routed packets by an Interior Gateway Protocol (IGP) such as "Open Shortest Path First" (OSPF) and "Intermediate System to Intermediate System" (IS-IS) , throughout the MPLS-enabled network. As the name suggests, the protocol enables the routing of packets based on assigned labels to the routing paths, providing some QoS (quality of service) that simple IGPs do not have. More so, it can escalate into both MPLS VPN (Virtual Private Network) and MPLS TE (Traffic Engineering).

When evaluating presently employed networks with traffic engineered MPLS, we conclude that although it is a clear improvement over the pure IP networks, there are still problems that come with the encapsulation protocol[1]. To start with, large networks can be very complex, and although MPLS hides the complexity, it does not make it go away, making them more expensive and harder to maintain. Secondly, the overall view of the network is hidden by the encapsulation protocol, limiting the manageability of the network regarding unexpected traffic situations. Lastly, the usage of heavy signaling protocols such as LDP and RSVP-TE lead to a sub-optimal performance of the network. As is the case, this heavily used protocol has started to be outperformed by a more recent solution.

2.1.2 Segment Routing over MPLS

With the evolution of the internet progressively shifting its focus towards cloud, virtual and application-centric platforms integration, the need for both flexible, scalable, and simpler to manage networks is ever increasing. MPLS can no longer sustain the mentioned needs of worldwide operators regarding application engineered routing, and since SR can be implemented over MPLS without changing the forwarding plane, it has become a rather attractive technology to ISPs. Not only that, the prospects of migrating from an IPv4 into an IPv6 data-plane give SR an extra boost in popularity, since the technologies benefits can be manifested without the need of MPLS.

Researchers have delved into the architecture of this new technique. Their findings conclude that SR removes the need for heavy protocols such as LDP and RSVP-TE, improving the scalability and flexibility of the networks [8]. Moreover, by decreasing the number of protocols inside the network, it becomes more scalable and displays performance gains.

All network devices are composed by 3 major architecture planes: Management, Control and Data planes. Here, the Control and Data planes will be focused on, as the difference of how SR deals with them in comparison to the traditional devices is pivotal to the improvements SR offers, along with the decoupling process of SDN which will be approached later on.

The **Data Plane** dictates the processing of packets in the network, based on the information in each packet's header - a list of segments. These segments represent subpaths that form a complete route with instructions on how the packet should be forwarded. In that segment list there is one active segment - the instruction being ran at the moment. Each segment has an identifier (SID), and these can be differentiated into 3 main types [9]:

- **Node SID -** Forward to a certain node with the referred SID using the shortest path.
- **Adjacency SID -** Forward through a certain path considered as an adjacency by the running IGP.
- **Service SID -** Forward a packet to a service with the established SID.

The **Control Plane** defines how the SID information is shared throughout the network. SR makes use of the running IGP to handle segment distribution inside a local network. The most commonly used IGPs are IS-IS and OSPF, and the extentions developed for both these protocols make it possible for any SR compatible router in the network to maintain an SID database, as well as providing end-to-end encapsulation without the need of the LDP protocol.

Another role of the Control Plane in SR is the selection of the forwarding path. Static configuration is possible, but except for specific cases such as troubleshooting it is obviously sub-optimal in regards to performance and scalability. Thus, the main methods for selecting a forwarding path are the **Distributed Constrained Shortest Path First (CSPF)** calculation, or the implementation of a SDN-based approach integrated with a controller centralized network.

With CSPF, an ingress router (router placed at the edge of the SR network that first receives the packet and forwards it throughout said network) calculates the shortest path to a destination, and matches said path with a SID sequence referring to it. The shortest path may or may not be subjected to extra decision-making parameters with traffic engineering purposes.

With an SDN-based approach however, broader options beyond shortest path calculation are available. With the centralized controller, a network manager can better analyze traffic inside the network, and can directly act on it by providing traffic engineering commands specifically design to deal with the current situation in real time.

2.1.3 Segment Routing with IPv6 (SR-IPv6)

Contrary to an IPv4 network, IPv6 networks do not need the help of neither LDP or MPLS to use SR routing, making it possible to implement SR over IPv6 networks without the mentioned protocols. IPv6 headers are flexible and contain supplementary information that can be used by the network. This allows for the usage of SR in environments where MPLS is not usually found, such as datacenters, and in networks where the MPLS is not implemented. If need be, SR-IPv6 can be used alongside IPv4 SR-MPLS. This has been proven to be useful in cases where there are IPv6 routers trying to communicate through a core network compatible only with IPv4 [21].

In SR-IPv6, a segment ID (SID) is defined as a packet header in place of the MPLS Data Plane label. This makes the advertisement of anything other than the IPv6 packet prefixes obsolete, as the simple advertisement of said prefix is the same as the SID. All the information regarding the network segments are edited right at the header of the IPv6 Packet.

Much like with SR-MPLS, the IPv6 version takes advantage of a controller to compute the best transmission paths composed by lists of segments. A Segment Routing Header (SRH) is placed in the packet's header, containing the segment list, next segment, and active segment. With this information, the network is capable of routing the packet through the segments in the SRH and do so in a specifically ordered sequence [22].

2.2 SDN Overview

Software Defined Networking is a paradigm that has emerged with the desire to reduce the impact of the limitations present on current networks. The current vertically integrated networks mostly rely on a rigid network infrastructure, possessing several nodes that take it upon themselves to individually deal with both the control and data planes. Although this offered resilience to the network, any change could turn out to be a daunting task, as modifications regarding the control plane would have to be made individually on each of the managed routers, switches and other specialized infrastructure hardware [11], SDN decouples the network's control plane from the underlying hardware, and attributes it to a controller that can manage it in a collective manner. This defining behaviour provides a broader viewpoint of the entire network that results in the possibility to manage the control of the network's hardware in its entirety, improving the network's manageability and flexibility [10]. The separation of data and control planes also allows for generic middleboxes to be used in place of the traditional specialized hardware, as the SDN controller takes charge of forwarding and logical decisions in the network. This would directly result in a lower cost of network maintenance.

In summary, SDN is defined by 4 fundamental pillars:

1. The functionality of the control plane is removed from network devices' responsibility (i.e. routers, switches, etc.), simplifying them into packet forwarding nodes.
2. That same functionality is given to an entity (SDN controller) possessing of an abstract network view, allowing for broader management options that are easier to implement.
3. Contrary to traditional IP networks, forwarding decisions are no longer destination-based. They are instead flow-based - a packet stream between a source and a destination with identical forwarding services, managed by the SDN controller.
4. Through an API (Application Program Interface) running on top of the controller, the network becomes programmable, interacting directly with the network's infrastructure devices.

The SDN architecture can be broken down into 3 distinct layers. Along with the Control and Data layers, there is also the Application layer [10]. The approach to this architecture will be bottom-up, that is, starting with the infrastructure layer (equivalent to the Data plane) ([1][11]).

Infrastructure Layer

The bottom-most layer of the architecture. It is composed of all the hardware found in traditional networks. It is responsible for executing packet forwarding and communication between nodes in the topology, receiving the instructions to do so from the Control plane, since the devices do not possess autonomous decision making.

Control Layer

Considered by some as the most relevant layer of the three, it is the mastermind behind most benefits brought to the table by the SDN paradigm[1]. The control layer manages the forwarding tables and the logical decisions that would traditionally be made by each device. It communicates with the Infrastructure layer through the Southbound Application Program Interfaces (APIs). These interfaces enable the

dynamic changes in real-time events of all network devices in the first layer.
This layer also has possesses Northbound APIs, which link it to the third and topmost layer, the Application layer. These APIs differ from the Southbound ones as they focus on receiving information on the running applications' needs such as, but not limited to, bandwidth and storage. The exchange of this information allows the automation of network applications (for example, firewalls and other security services) in the SDN network.

Application Layer

Responsible for the network applications that take action in the network. It provides the controller with all the needed information for it to coordinate the forwarding logic needed for the data plane to transmit packets throughout the network.

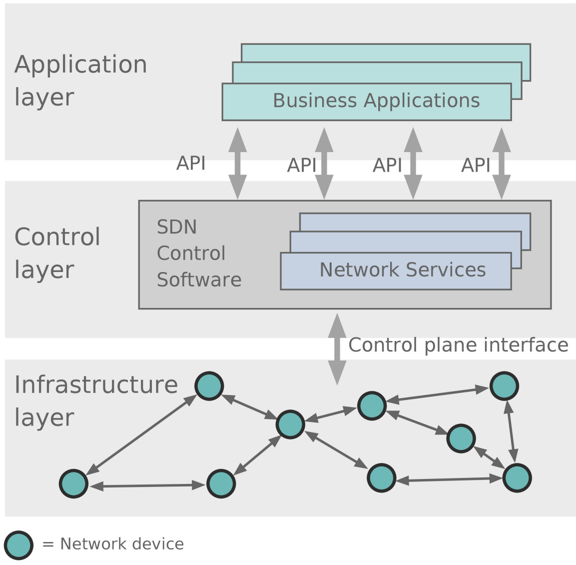


Figure 2.1: The 3 layers of the SDN Architecture.

With the segregation of the Control and Application layers and the information exchanged between the applications running on top of the network and the network itslef through the Northbound APIs, SDN can provide the network with an application-based network management. This means that each application is aware of the network state, resulting in a network capable of operating in accordance with each of the applications needs [23].

2.3 Network Function Virtualization

The motivation behind the enforcement of NFV revolves around the desire to horizontally segregate Virtual Network Functions (VNFs), and replace the traditional dedicated hardware with virtualized software, providing improved manageability while reducing capital expenditures and operation expenses. These VNFs are software implementations of traditional network functions, decoupled from the hardware. The isolation of said functions facilitates the identification of points of failure in the network. This allows the network to evolve to one where failure of a function (for example, a firewall) can quickly be identified and resolved.

Network Function Virtualization (NFV), is the overall concept of running software defined network functions along with the virtualization of the network.

Research regarding the efficiency of such virtualization techniques demonstrate though that the performance, flexibility, and other important metrics have unexpected and sometimes undesired values, depending on the type of functions and services being implemented. It was noted that for the same functions, performance varied depending on the order of the execution, as well as the type of service being provided (for example, network infrastructure compared to cloud services) [14] [15] [16]. Eventually, it was established that depending on the service set to be implemented, one had to manage VNFs specifically with the services to be provided in mind. But through this thought process another problematic scenario arose: VNFs requiring managing increased, and consequently, the complexity of the overall NFV management. As such, Management Orchestration (MANO) started being developed, to provide a platform to simplify handling the increasing complexity of NFV.

MANO systems are normally tasked with the management of virtualized infrastructures and VNFs (often implemented as virtual machines or software container images). Being able to provide better automation, high-availability and flexibility to those components are some of the factors that highlight the usefulness of MANO systems.

There are different MANO projects being developed. The purpose differs from each project: some are more academic-centered, others aim at business environments. Most MANO project adhere to the ETSI MANO framework model. Most MANO frameworks are supported by the **European Telecommunications Standards Institute (ETSI)**, commonly referred to as **ETSI MANO**. The framework used is viewed as one of the most relevant NVF MANO frameworks available, if not the most relevant. The platform focused on this thesis is the OSM platform.

For this project, open source MANO (OSM) will be used to provide a dedicated framework to view and manage the entirety of virtual functions being ran in the network. As such, Management Orchestration (MANO) started being developed, to provide a platform to simplify handling the increasing complexity of NFV.

The MANO framework in question is composed of three essential function modules [14]:

- **Virtualized Infrastructure Manager (VIM) -** Responsible for management of virtual machines and containers (VNFs), handling the virtual links between them. In this project, that will be achieved through usage of SDN.

- **Virtual Network Function Manager (VNFM) -** Its focus lies on dealing with network services, that is, controlling the VNFs' life-cycles, separately. The manager is charged with more than just the automation of the VNFs, delving into the configuration, the start of a function, and it's death.

- **Network Function Virtualization Orchestrator (NFVO) -** Ensures the integrity of the overall service provided by the interaction of all the VNFs in the system. It is in charge of all the data required to ensure the end-to-end integrity of the service. External applications communicate directly with the orchestrator when in need of critical information regarding the entities involved in the service (for example, VNFs, network services, and available resources).

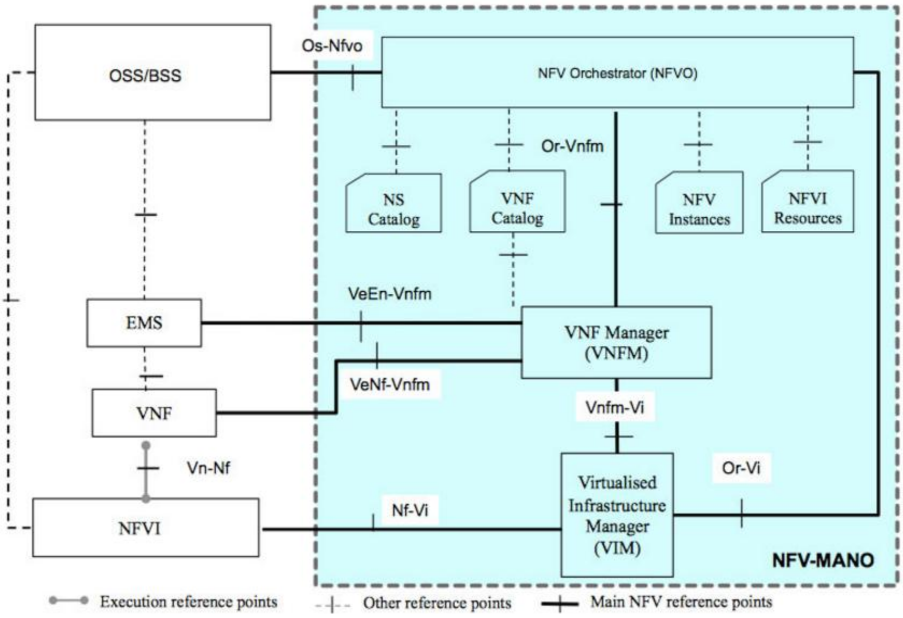


Figure 2.2: The NVF MANO framework based on the ETSI architecture.

Note that at this time, the ETSI MANO framework is still in expansion, and subject to improvements and further development.

2.4 ONOS Controller Platform

The Open Network Operation System (ONOS) is an open source project that leverages a network controller that can be used alongside SDN networks. It provides the control plane of the SDN network, and it is this platform that will be used in the development of this project.

ONOS was built with one goal in mind: becoming the SDN platform for controlling service provider networks. This does not preclude the system from being used in different networks. It just means that a set of requirements needed to be met in order to achieve the main goal, regardless of the other types of networks' needs.

For starters, the system had to be built with the scale, performance, and critical nature of the service provider's needs in mind. Thus, the system needed to possess high levels of **scalability**, **Availability**, and **Performance**.

Next, since the focus of ONOS was to provide a SDN platform with all its components, a strong level of **abstraction** was paramount.

Lastly, **Modularity** was significant elevating the platform's flexibility and possible growth. By separating its components, the potential complexity that would be born out of the platform's improvements were cut short.

This resulted in the clear separation of the overall ONOS into several subsystems. Although every subsystem is essential in some manner towards the functioning of ONOS and the overall network being managed, they are fairly independent from one another in terms of each of their functions, as some of them work within the northbound scope, while others are related to the devices and southbound API.

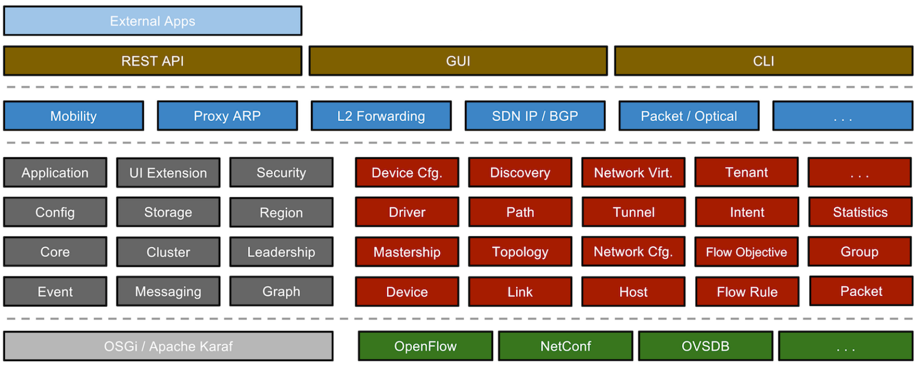


Figure 2.3: Depiction of ONOS' core subsystems

The ONOS subsystems can be separated into 6 main fields (not counting the external apps), each coloured differently in the figure above. The colour brown is attributed to Northbound APIs, acting as application interfaces. Right below, the blue ones represent network applications that would, in traditional devices, be equivalent to the functions the device would take in the network. Dark grey subsystems are related to several core subsystems that are not network related, opposed to the red subsystems. The colour green encompasses Southbound API modules. Finally, the light gray module refers to a framework that helps to manage services composed of several functions. The OSGi and Apache Kernel frameworks are extensive and will not be approached in this paper as the technology's operating method is not relevant to this research.

When referencing the Southbound API managing the connection between the ONOS Controller and the devices in the Infrastructure Layer, it is worth noting that ONOS Southbound API is not limited by any specific protocol, and supports several different implementations, namely OpenFlow, NETCONF/YANG, and SNMP. As mentioned in the SDN section, ONOS makes use of its Southbound API to communicate with the devices in the infrastructure layer. Subsystems like **Flow Rule**, which are responsible for managing and enforcing the rules for network forwarding on devices, are directly involved with the information sent to the devices by the controller through it's southbound interfaces[12].

Regarding the Northbound API connecting to the application layer, ONOS takes advantage of its own **"Intent Framework"**. This subsystem allows for the applications to declare their management needs to the controller, based on a pre-existing policies of the applications in question. This "intent-based" networking is a way for the applications to simply state their needs to the controller and letting it handle all the work, believing that these needs will be met. This is the foundation for the automation offered by ONOS' SDN approach, resulting in a much more scalable network [13] than a traditional non-SDN network.

2.5 OSM - Open Source MANO

After the specification of NFV MANO frameworks, specific projects started to emerge from each framework. This paper makes use of OSM, an expansion project based on the ETSI MANO architecture previously mentioned.

OSM aims to deliver the automation and modelling of enterprise-grade services. By implementing a virtualized network supervised by OSM, the integration of NFV infrastructures and VNFs is meant to be simplified, providing a stable approach to the emergence of virtualized networks. The OSM project delivers a VIM-independent product, compatible with multiple SDN technologies and capable of managing all types of VNFs [17].

The first factor contributing to the automation of Network Functions and Services is the **Information Model (IM)**. This model generates tree representations of the various Network Functions (not limited to virtual ones) managed by the system and automates their lifecycles at instantiation and proceeds to do so throughout their daily operation. Any given element can be instantiated independently of the VIM module, as well as any SDN software, in use. During the implementation phase of this project, it is planned that OSM will make use of ONOS' SDN software in its Virtualized Infrastructure Manager module. Another factor that is offered by OSM is its feasibility of integration in brownfield environments: by providing one single **Northbound Interface (NBI)** integration point, which allows for the handling of both physical and virtual assets/functions simultaneously, guaranteeing the proper handling without the need to make any distinction between said assets.

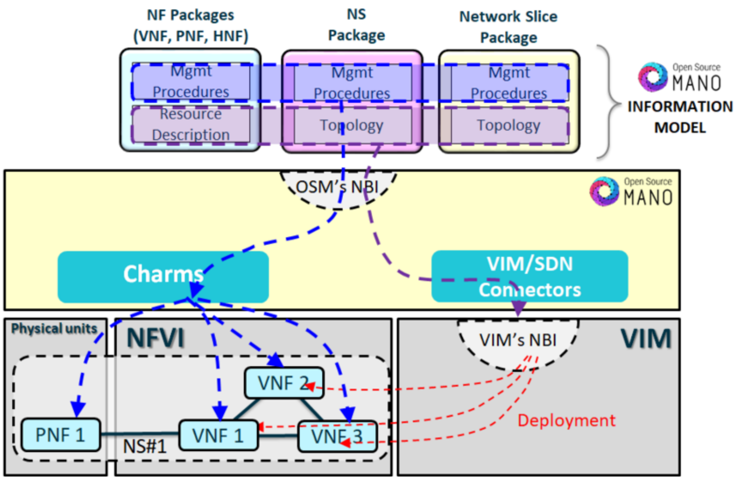


Figure 2.4: IM, Northbound Interface and VIM in network function management

In Figure 2.4, we have the depiction of how Network Functions, Network Services, and Network Slices are deployed and maintained by the IM and NBI interaction. The IM communicates with the VIM through the single Northbound Interface of OSM. In turn, the OSM communicates with the VIM which deploys the functions present in the IM in the Network Function Virtualization Infrastructure (NFVI). These functions are in turn managed by the OSM along with the Physical units, through the VNF Configuration Adapter (VCA). This entity is basically a controller used for the deployment of Proxy Charms: a collection of scripts that help with the reliability and repeated deployment and management of applications, in this case, the Network Functions in the IM.

2.6 Service Function Chaining

The future for networks seems to aim towards virtualization, and the escalation of NFV environments employing multiple VNFs comes as no surprise. To support this escalation, the Service Function Chaining (SFC) paradigm was proposed to deal with the sequential traffic routing between VNF instances. Automation and improved performance are two of the giveaway benefits of SFC, but while there are well documented benefits to this approach, there are also some concerns regarding its deployment[18]. One of such concerns is a recurrent performance uncertainty, originating from the usage of VNFs as opposed to hardware dedicated functions. This issue is commonly looked at as a question of improving performance, leading to the research and development of faster and more powerful NFV tools. Still, there are different perspectives on this matter that approach the problem not as a lack of performance, but regarding the reasons behind the performance variations for multiple cases in the same system.

Studies have comprised possible triggers such as uneven CPU usage by VNFs, bad handling of said resources by the managing tools, and the routing of SFC of throughout a network with multiple VNFs and VNF instances where the specific sequence of running instances matter [15] [18]. One conclusion of such research revolves around the need to implement a mechanism to be able to spot bottlenecks and irregularities in various sequences of instances and different environments, that can identify and act on said instabilities locally, regardless of the network implementation. One such tool is Probius[15] that aims to provide an abnormal behaviour detector based on several performance features according to the VNF's architecture, matching said abnormalities with possible performance variation triggers.

Chapter 3

Segment Routing Applications

In this chapter it will be explained how our solutions can help to reach the goals proposed in section 1.2, as well as the reason behind the choice of said solutions.
With Segment Routing gaining popularity, and SDN along with NFV looking to be the future of programmable networks, it seemed a good idea to expand on the idea of integrating the 3. The plan was to use an already existing tool, by the name SREXT (found in [19]), as an extension to the mininet capabilities, by conferring its virtual hardware the capability to route VNFs with Segment Routing.
The following use case is related to datacenters. The reason for that choice is based on the biggest challenges and needs that datacenters have not been able to overcome. They have faced and still face a problem regarding the usage of non-commodity hardware, and the reliance on in-site installation of dedicated hardware, which are direct causes of its high OPEX and CAPEX. Thus, the lack of agility and programmability is an issue that many enterprises wish to tackle and overcome, as it will drastically reduce costs. With this in mind, the first use case elaborated in this thesis is a Spine-Leaf network, much like a Datacenter network, where Segment Routing is enabled and a demonstration of its functioning is achieved.
One open source community that is in the forefront of SDN development is ONOS. They have many projects ongoing regarding different advantages and use cases for software defined networks. One of such projects is SPRING-OPEN. This project aims to demonstrate maturity, readiness and scaling capabilities of Segment routing and SDN usage in already available hardware in professional environments and enterprises. This seemed like a noble pursuit, and this thesis aimed to replicate the functioning of SPRING-OPEN.

Chapter 4

Segment Routing Implementations

The focus of this section is on the implementation of the Segment Routing Solutions found. It will start with an in depth explanation about the set-up of the environment and tools used. It will then describe each solution including the achieved results, implementation variations, and faced challenges throughout the development of each.
It will expand on 3 implementations: a Linux-based NFV Infrastructure, a Spine-Leaf solution with an ONOS controller, and the also ONOS controlled SPRING-OPEN solution.

4.1 Environment Set-up

4.1.1 Minimum Requirements

As minimum requirements for the Ubuntu VM to run ONOS, we need to allocate two virtual CPUs and two GBs of RAM. The native OS on which the VM operates was, in my case, Microsoft Windows 10, on which the VMware hypervisor was installed. The hardware used has an Intel i7-7500U Hyper Threaded CPU possessing 2 cores with 2.7 3.5 GHz CPU, as well as 16 GBs RAM. It should be noted that these specs are enough for the intended work, as the VM requires at least 2 virtual cores to be allocated. Since the CPU is Hyper Threaded, it is possible to use 2 virtual cores for each physical core. With this in mind, I decided to upgrade the default VM configuration by raising the CPU virtual core count to 4, and the system memory to 12 GBs, in hope of having better user comfort.

4.1.2 Used Software

This project was centered in the usage of the ONOS Software. To utilize this software, it resorted to other tools for support, such as Docker, mininet, and VMware (as mentioned previously). ONOS' latest version at the beginning of the project development was 2.2.0, and thus this version was used. Using the 64-bit Ubuntu Bionic 18.04 LTE version VM with VMware, ONOS was instaled using the Docker image made publicly available.

Docker can be installed by inputting the commands found in [24] into the Ubuntu terminal. Installing ONOS using the Docker environment is much easier than installing it as a service. It allows you to use an already made installation of the service as if it was in your system. This way you can bypass some errors that may occur regarding Java and ONOS versions' incompatibilities. It is worth to note that some variables and important keys are already predefined, and before using the software, we should be aware of what they are. Port values used for communication with Openflow (6653), OVSDB(6640) the GUI (8181), ONOS CLI (8101), and more information can be found in the Dockerfile, available on Github [25]. With this in mind, it is possible to start ONOS by using the terminal command *docker run -p 6653:6653 -p 6640:6640 -p 8181:8181 -p 8101:8101 -d –name onos onosproject/onos:2.2.0*.

4.2 Implementing VNF Chaining in a Linux-based NFV Infrastructure

The initial plan of this project was to utilize the already developed Linux-based NFV Infrastructure from the research paper ”Implementation of Virtual Network Function Chaining through Segment Routing in a Linux-based NFV Infrastructure” [19]. The developers behind this tool called ”SREXT” had programmed the default networking appliances offered by the Linux environment, conferring extra configuration options. With these improved appliances, it would be possible to implement not only IPv6 Segment Routing, but also NFV function chaining. Another benefit offered by the tool was the possibility of chaining SR-unaware VNFs. This means that it would be possible for VNFs that are not designed specifically to be used with Segment Routing would still function in this environment, making it so that generic VNFs could still be used in an SR network. A tutorial version was developed and made available by the authors, where a basic version of the utilities of the tool were demonstrated.

Although SREXT was functional at the time, after the Ubuntu kernel 5.2 update, the installation of the tool was no longer possible. This problem was raised by multiple users and was indeed acknowledged by the developers, but not fixed. According to the authors of SREXT, there will be no update to the tool, making it so that whoever wants to use the tool cannot do so if the Ubuntu version of their machine uses the kernel version 5.2 or higher.

Another problem arose out of this situation, related to the use-case tutorial provided by the authors. The tutotial was a testbed provided in a VirtualBox using Vagrant, a tool for managing virtual machine environments. This testbed broke due to the same reason, and this problem was documented as an issue on December 9, 2019, by the community. Both issues are still unresolved at this time.

4.3 ONOS Implementation - Spine-Leaf Solution

After the SREXT implementation setback, the project suffered some changes, and the development was centered around the ONOS platform. ONOS allowed for the development of the Segment routing network on top of the mininet network emulator in which the standard Linux network software is ran.

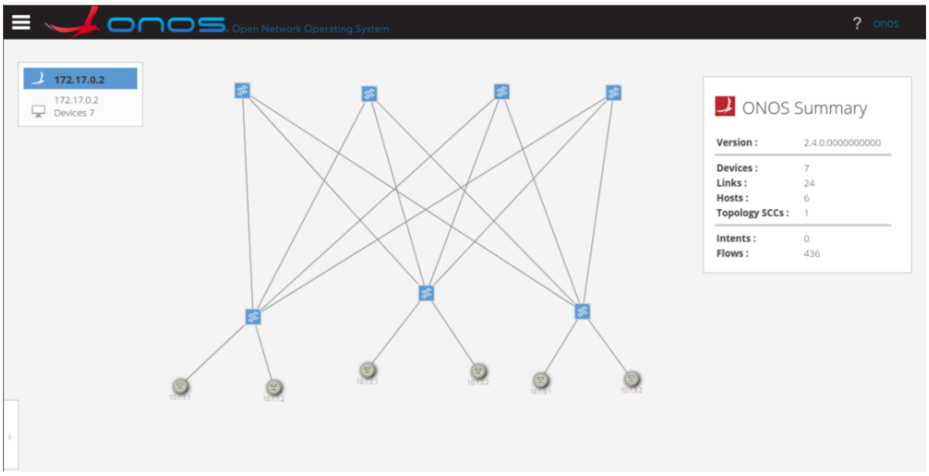


Figure 4.1: Leaf-Spine topology of the network using the ONOS GUI.

The implementation of Segment Routing using the ONOS in this project platform can be divided into 2 solutions: Spine-Leaf and SPRING-OPEN.
Spine-Leaf is the name given to a popular type of network architecture that is used in datacenters. Its is composed of 2 layers of switches, the leafs, connected to servers and spines, and the spines, connected only to leafs. A leaf connects to all spines, the same way as a spine connects to every leaf. This architecture is specially popular in datacenters with heavy server-to-server communication, as it minimizes latency and bottlenecks.
The objectives of this solution are twofold: to look into the functioning of SR in the Spine-Leaf architecture, and to see how it is possible to apply traffic engineering measures with the usage of the ONOS controller.
As seen in figure 4.1, the Spine-Leaf topology is composed by 4 Spine nodes and 3 Leaf nodes, while each Leaf node has 2 hosts directly connected. There is redundancy in place as every Spine is connected to each Leaf, but not connected to other Spines. This means that for a Spine to communicate with another, it must have a leaf as a medium.

The topology was developed using a python script called **TopoSR.py**, and is run with the help of **mininet**, the network emulator used in this project.
The python file creates all nodes of the network and assigns them several attributes necessary for the functioning of segment routing and IP forwarding, such as IPv4 addresses, MAC addresses, SIDs, and others. For the Spine nodes' SID, we attribute the number "20x" where x = the Spine number (i.e. Spine 1 = 201). The IP addresses of the Spine nodes are 192.168.0.i, where i = SID number. An identical process is used for the Leafs, where instead of using the value "20x", "10x" is used.
The hosts are attributed an IPv4 address based on the Leaf to which they are connected.. The mapping of host IDs is as follows: Leaf1 connects to Host1 (10.1.1.1) and Host2 (10.1.1.2). Leaf2 connects to Host3 (10.1.2.1) and Host4 (10.1.2.2). Leaf3 connects to Host5 (10.1.3.1) and Host6 (10.1.3.2).
The python fie also defines the address where the ONOS GUI is connected, along with the port. The full link to access the GUI is `http://'172.17.0.2':8181/onos/v1/network/configuration/`, and the

authorization parameters are the username **onos**, and password **rocks**.

4.3.1 Starting the environment

the following steps must be followed:

- Start ONOS by opening a terminal window in ubuntu and input the command **sudo docker run –name onos –rm -p 6653:6653 -p 8181:8181 -p 8101:8101 -d onosproject/onos**. This command will connect to ONOS' Openflow, OVSDB, GUI and CLI to the ports 6653, 6640, 8181, and 8101, respectively.

- Login to the GUI using the username ”onos” and password ”rocks”.

- Start Mininet by openign a second terminal window and run the command **sudo mn –custom TopoSR.py –link tc –topo=TopoSR '–controller = remote,ip=172.17.0.2,port=6653'**. This wil start the topology found in the file TopoSR.py and connect to the controller found at 172.17.0.2 through port 6653.

- Activate the required applications to use Segment Routing. These applications can be found in the GUI. Start by activating **OpenFlow Provider Suite** and cjeck if the topology is correct and if you can ping the hosts. Next, activate **network config link provider**, **network config host provider**, and **segmentation routing**.

After following these steps, communication using segments should be active.

4.3.2 Simple Traffic Engineering and Access List usage

In hopes of demonstrating the possibility of deploying extra configurations with Segment Routing regarding packet forwarding and Traffic Engineering, the application Multicast Traffic Control was used. This application enables the implementation of Segment Routing's Equal Cost Multi-path (ECMP) to redirect traffic so that a pair of communicating hosts talking between themselves do not use the same path. This is a relevant addition to a network. There is a growing demand for network bandwidth in many network management cases, and with this Traffic Engineering measure in place, the network should possess a better bandwidth management. Since each file transfer can create a new flow across the network, it is natural for collisions to occur when new flows use the same paths, but in this case, said collisions will be avoided.
A simple Access List was also developed to act as a Firewall that denies ICMP communications between Host1 (10.1.1.1) and Host5 (10.1.3.1). To add the ACL rule to the ONOS Controller, simply open a terminal window and run the python file **post.py**. This file will introduce the rule defined in the file txt.JSON to the **acl/rules** directory of ONOS where the ACL rule is defined.

4.4 Spine-Leaf Experimental Results

The implementation of Segment Routing in the Spine-Leaf topology was successful. The SIDs for each Spine, Leaf and Host, along with the attributes necessary for developing the topology were defined in the **TopoSR.py**, that can be fund in the annex. This implementation was fundamentally used to demonstrate the functioning of Segment Routing in ONOS, and to analyse the content of the packets being forwarded in the network.

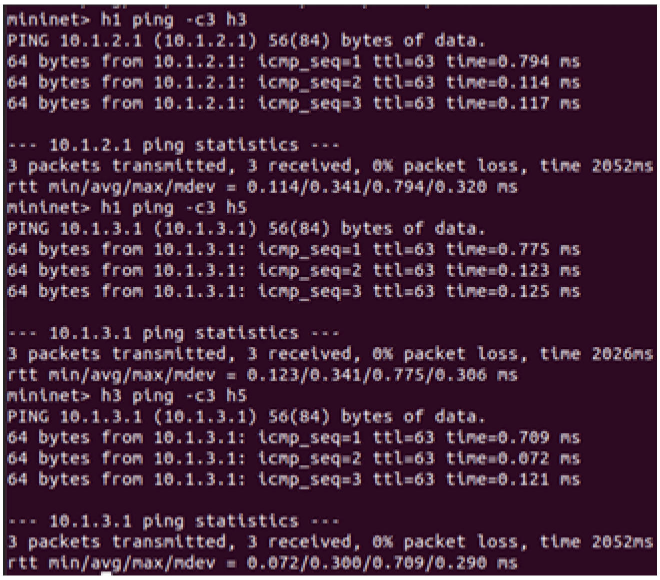


Figure 4.2: Ping verification for H1-H3, H1-H5 and H3-H5.

The connectivity verification was the first step, to see if Segment Routing was working properly. As depicted in figure 4.2, a host from a certain Leaf is capable of communicating with another host in a different Leaf, no matter which Leaf or host. The higher time value in the 1st message of each ping is expected: since it was the first time the hosts ever spoke to each other, the remote controller needed more time to calculate the route which the communications would make use of. After calculating said route, the travel time is reduced substantially.

After making sure that there was connectivity between hosts, wireshark was used to analyze the content of the packets being forwarded. A Wireshark capture was made involving the packets forwarded with the command **iperf h1 h5**. This command establishes communication from h1 to h5, and from h5 to h1. The Wireshark captures were made regarding the packets originating from H1 (figure 4.3) and H5 (figure 4.4).

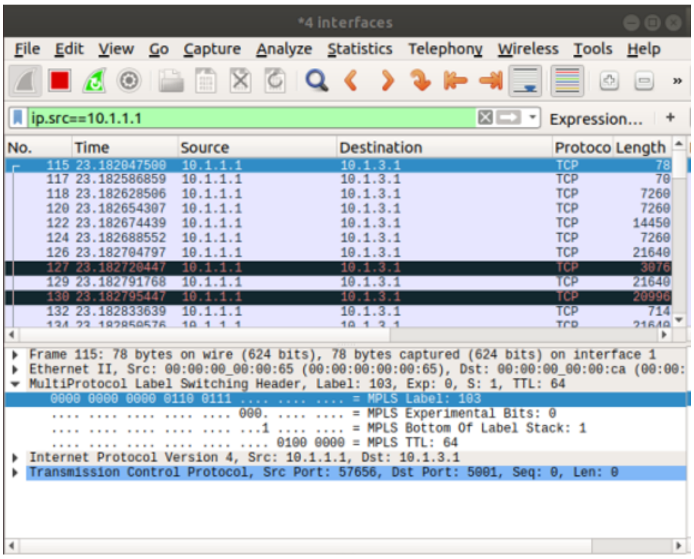


Figure 4.3: Wireshark capture to confirm SR behaviour (source in H1).

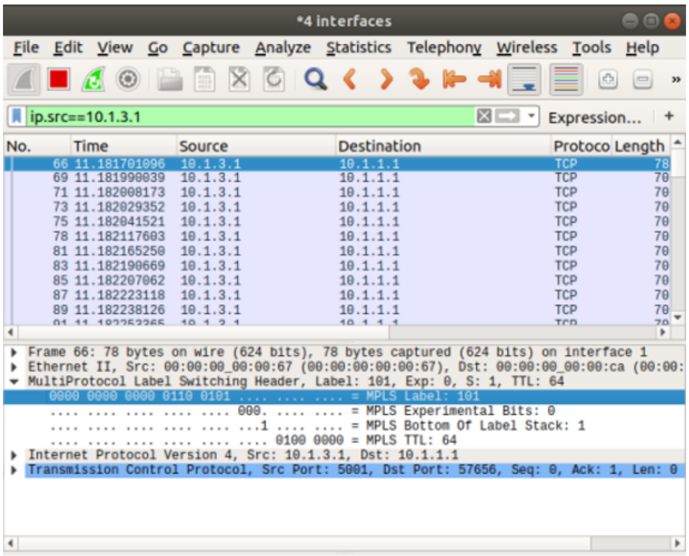


Figure 4.4: Wireshark capture to confirm SR behaviour (source in H5).

Through the analysis of figure 4.3, we can see that the packet originating from H1, which is connected to Leaf1, possesses the MPLS Header with the SID 103 in the label, which is the SID belonging to Leaf 3, to which H5 is connected. This demonstrated that Segment Routing is enabled and working in the network. Figure 4.4 also reinforces this claim, as it demonstrates that the same happens with the reverse routing path.

Furthermore, these captures also contribute to the confirmation that the Node SID forwarding has been properly configured: If simple MPLS was implemented, the label in the header would belong to a Spine node (20x type SID) due to the fact that no Leaf is directly connected to another Leaf.

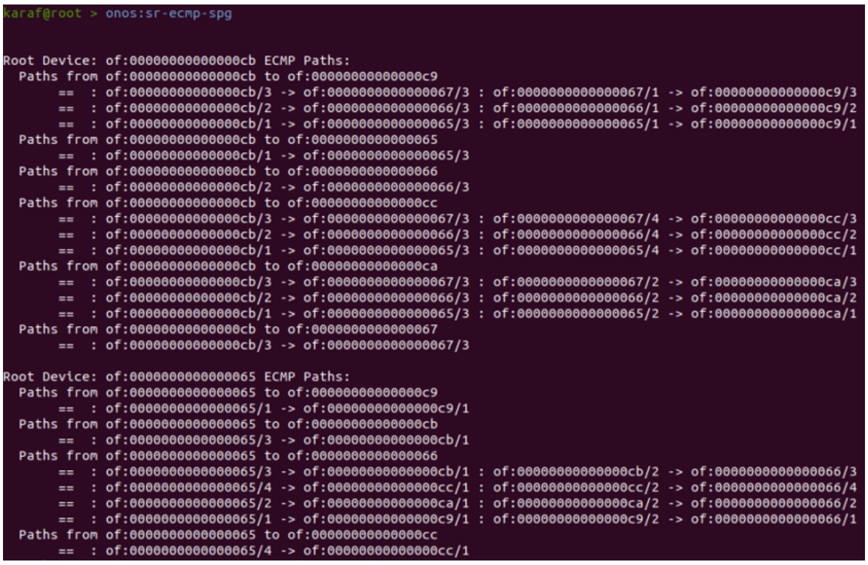


Figure 4.5: ECMP paths of Spine203(cb in hexadecimal) and Leaf101(65 in hexadecimal).

There was some consideration about the need to ascertain which path was used for forwarding, but due to the availability of 4 equally costly Shortest Paths (2 hops) the choice of pathing was random. It was then decided that focus would be placed into the ECMP (Equal Cost Multipathing), a traffic engineering measure innate to Segment Routing. When ECMP was enabled in the network, paths from each node to every other node were created and stored. Some of these paths can be seen in figure 4.5.

These paths were calculated through the Shortest Path, and all the paths with the lowest cost are shown. Each device is represented by a hexadecimal number correspondent to its SID in decimal. By using the paths defined, and knowing which paths are being used or not, the network can better allocate its resources to forward packets using links that have higher bandwidth values available and managing possible collisions.

Figures 4.6 and 4.7 show an example of a packet from Leaf1 that to go to H5 takes the interface eth2, connected to Spine 201, while when the response comes back from H5 to H1, the interface used is eth3, which is connected to Spine 202. This shows that the message sent by H1 and the answer by H5 do not take the same path, and not using the same route for the H1-¿H5 and H5-¿H1 communication.

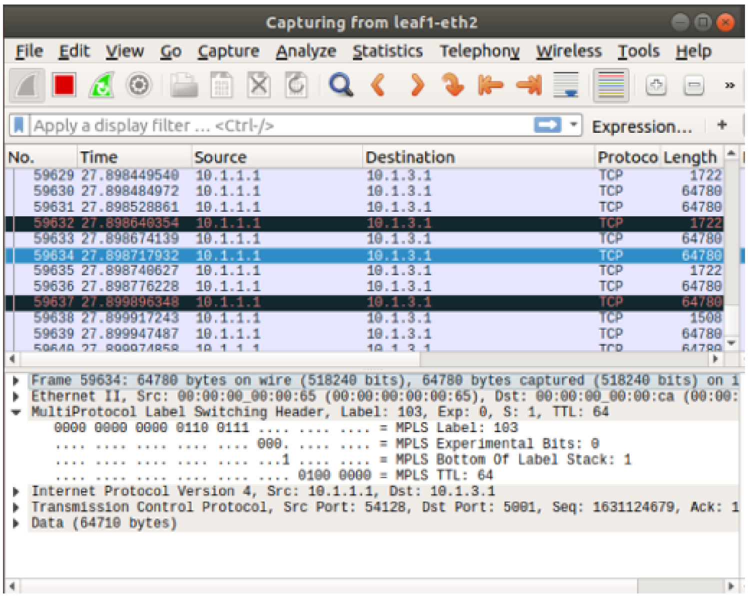


Figure 4.6: Wireshark capture of outgoing packet on interface 2 of Leaf1.

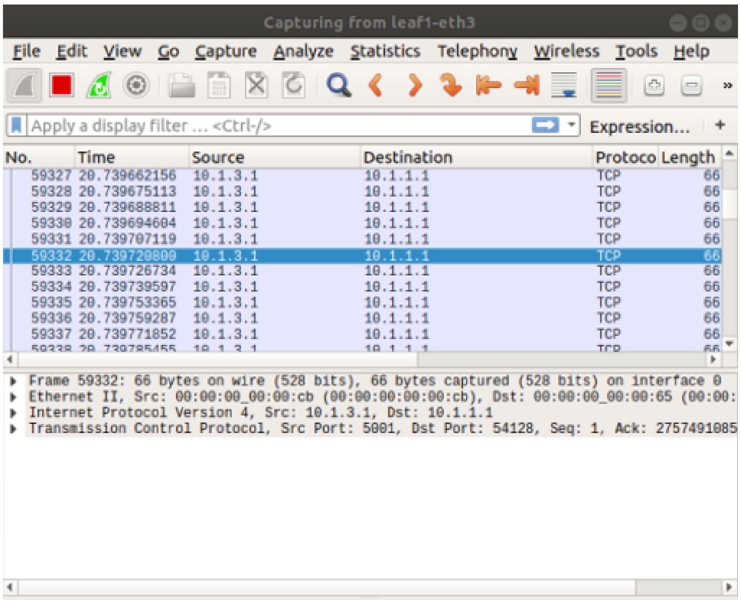
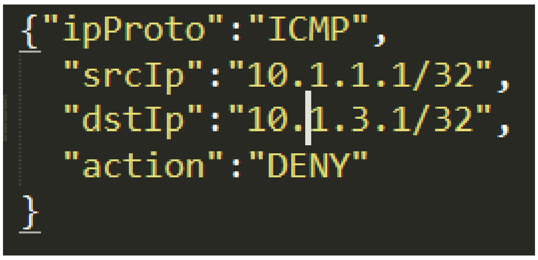
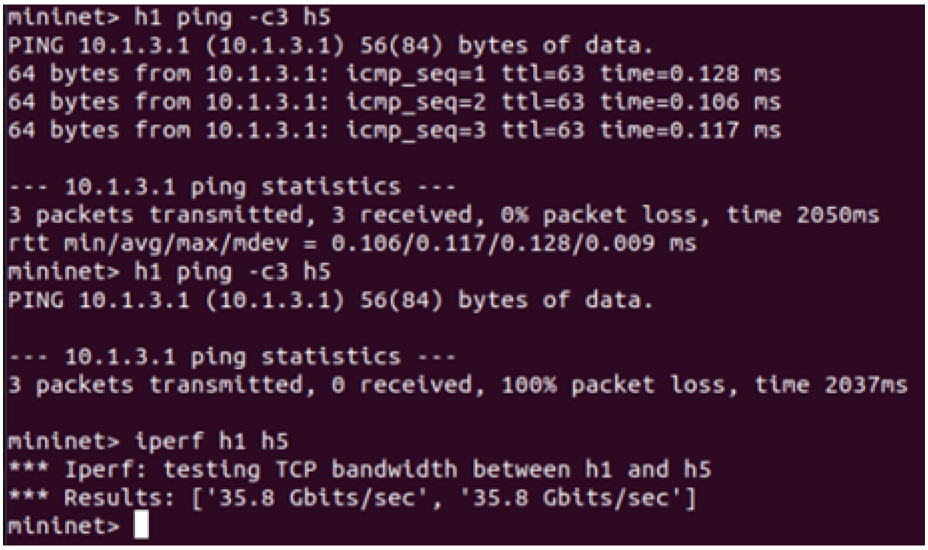


Figure 4.7: Wireshark capture of incoming packet on interface 3 of Leaf1.



```
{"ipProto":"ICMP",
  "srcIp":"10.1.1.1/32",
  "dstIp":"10.1.3.1/32",
  "action":"DENY"
}
```

Figure 4.8: ACL rule denying ICMP communication from H1 to H5.



```
mininet> h1 ping -c3 h5
PING 10.1.3.1 (10.1.3.1) 56(84) bytes of data.
64 bytes from 10.1.3.1: icmp_seq=1 ttl=63 time=0.128 ms
64 bytes from 10.1.3.1: icmp_seq=2 ttl=63 time=0.106 ms
64 bytes from 10.1.3.1: icmp_seq=3 ttl=63 time=0.117 ms

--- 10.1.3.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.106/0.117/0.128/0.009 ms
mininet> h1 ping -c3 h5
PING 10.1.3.1 (10.1.3.1) 56(84) bytes of data.

--- 10.1.3.1 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2037ms

mininet> iperf h1 h5
*** Iperf: testing TCP bandwidth between h1 and h5
*** Results: ['35.8 Gbits/sec', '35.8 Gbits/sec']
mininet>
```

Figure 4.9: Ping usage before and ping plus iperf usage after implementing ACL rule.

Another feature implemented with the Spine-Leaf solution was a simple firewall appliance, in the form of an Access List rule. The simplicity is due to the fact that the network itself is not very complex and has limited options on how a more elaborate firewall could have any impact in it. Still, the possibility of having a firewall is something deemed important, and thus it was decided that the implementation of such an example was beneficial.

The ACL rule is declared in a json file called **txt.json**. This file can contain multiple rules and can be added to the network at any time.In this case the only rule can be seen in figure 4.8. Rules can be added at any given time by running a JSON file. After the rule was implemented, it was confirmed that the ping command (composed of ICMP packets) stopped working, while the **iperf** command remained functional. This is depicted in figure 4.9

It can therefore be concluded that the firewall is working as intended. Other attributes can also be configured in order to create more complex rules, such as MAC addresses and ports.

4.5 ONOS implementation - SPRING-OPEN solution

The objective of using this solution was to implement a more comprehensive test-case of the Segment Routing capabilities, capable of incorporating VNF chain routing. ONOS SPRING-OPEN is a use-case developed by the ONF, relying on the ONOS controller, that can demonstrate the possibility of using SDN and Segment Routing along with already existing hardware, in a professional environment. The ONOS website provides a Virtual Machine build regarding the SPRING configuration for the ONOS controller. Along with the Virtual Machine and an installation guide, there are also some tutorials that explain the capabilities of the tool specifically for Segment Routing.

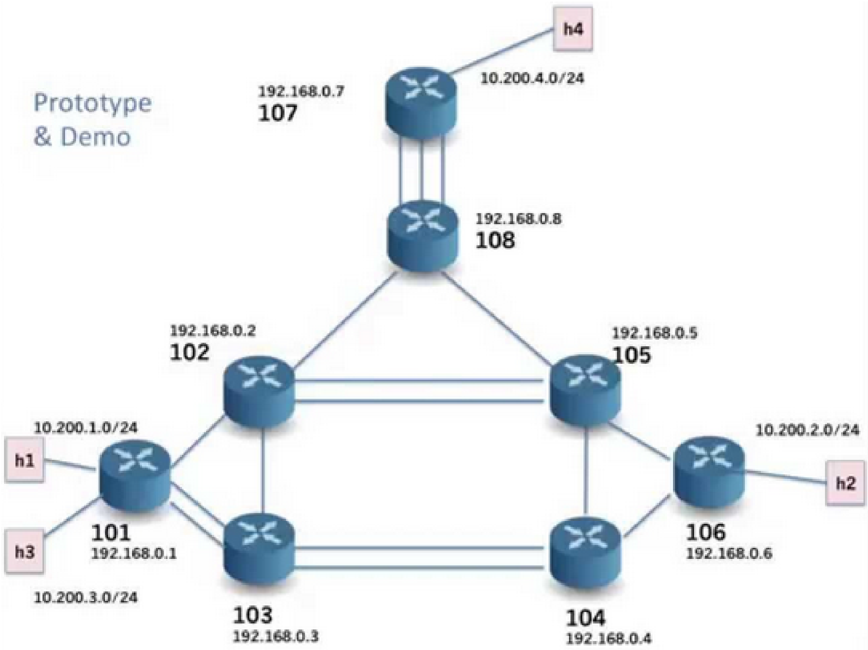


Figure 4.10: Network architecture use din the SPRING OPEN tutorial.

The network used was configured by the ONOS team responsible for the SPRING-ONOS project. The configuration file, **sr-3node.conf** can be found in the ONOS website, `https://wiki.onosproject.org/pages/viewpage.action?pageId=2130918`.

This solution would be used to demonstrate other advantages beyond the ones demonstrated in the Spine-Leaf solution. Instead of just demonstrating the functioning of the protocol, more specific use-cases can be demonstrated, such as Fine Grain Traffic Steering, Load-Balancing without the use of ECMP, and Segment stitching.

These functionalities are not native to the ONOS controller by its own, meaning that such use-cases would not be able to function without extra configurations for ONOS. These configurations can be found in the same web link as the network configuration file mentioned above, under the name **onos.properties**, as well as a small description on where to add the extra configurations.

Following the documentation and tutorials provided by the ONOS developers, the target was to replicate the functioning of a small scale intercity IP network, configuring it with Segment Routing and applying the above mentioned advantages and capabilities available with the routing protocol by following the available tutorials in the ONOS webpage regarding SPRING-OPEN, https://wiki.onosproject.org/pages/viewpage.action?pageId=2130908.

Sadly, the SPRING-OPEN project was archived and deprecated, despite its usefulness. The Virtual Machine containing the ready for use project was made unavailable, and the documented procedure to build the project from source is no longer functional. The main reason for the decommission of the project was the lack of integration offered by the SPRING-OPEN project regarding new complementary technologies and ideas. As mentioned, SPRING-OPEN relied on the heavy configuration of a specific ONOS version, which limited the configuration capabilities of the ONOS controller CLI, which in turn could only handle tunneling and other routing policies. All needed startup-configuration needed to be arranged via a configuration file loaded at startup. This means that [26]. The network would have limited potential to adapt "on the fly" to other changes beyond tunneling and routing policies. With a combination of SR and SDN, both which brag about flexibility and ability to adapt, this kind of limitation is counter productive.

In its stead, ONOS adapted a different project, less reliant in the startup configuration of the controller. the Next Generation SDN Platform (NG-SDN) leverages 3 technologies at its core: µONOS Stratum, and Trellis.

4.6 NG-SDN outline

The successor to the SPRING-OPEN project is called the new generation SDN project, or NG-SDN for short. Like its predecessor, it is an open source platform that focuses on the development of SDN networks, but unlike the previous project, it manages to integrate multiple technologies otherwise incompatible with the ONOS controller. For example, NG-SDN makes use of the P4 language and P4 Runtime protocol, which is growing in popularity among the tools utilized in the development and controlling of Software Defined Networks. [27]

One of the main components of NG-SDN that differentiates it from SPRING-OPEN is the new upgraded controller, µONOS. This upgrade focuses on the ONOS controller, conferring zero-touch provisioning, extending its capabilities in configuration, control, and monitoring, and also opening up the possibility to configure the network in real time and "on the fly" in ways not previously possible. The performance of the controller is also upgraded, as the improved µONOS is also aimed at 5G networks [28].

The second core technology used by NG-SDN is Stratum. This technology is one of the two core additions that differentiate NG-SDN from SPRING-ONOS. Stratum is essential in conferring hardware independence and top down progammability in the network, along with Fine-grained control and measurement, by providing an intelligent and cooperative data plane [29].

The third core component of this new project is Trellis. Trellis cannot be considered an upgrade on the old ONOS' SPRING-OPEN project, as its utilities go beyond the scope of the previous project. Trellis is a platform to create open-source multi-purpose L2/L3 switching fabrics, such as Spine-Leaf and NFV

switching fabrics, specifically developed for integration with μONOS and Stratum [30].
There are other platforms currently being developed to be used with the upgraded μONOS, for different uses and objectives. With the use of the 3 mentioned, the integration of NFVs with a Segment Routing capable network would be much less troublesome. For one, there would be no need to develop from scratch software to integrate NFVs into the network like the case with SREXT, in section 4.2. The fact that this project is still expanding but already possesses functional cases is also a plus.

Chapter 5

Conclusions

SDN threatens to become a core feature of networks in the futures. It leverages many benefits and even more so when paired with other technologies such as SR and NFV. This thesis demonstrates a successful use case where these technologies can be implemented, as well as bright prospects for future development and possible adoption by enterprises in professional environments.

Segment Routing presents benefits for networks that are tailored for today's needs, and show promise to secure its position alongside SDNs in the future. From datacenters to broad international networks, several solutions are in development and many strive to be mature enough to be adapted into professional environments. It is the case that, as shown, some use cases are already capable to hold its own in the networks of today, with others following the same footsteps.

Network Virtualization is a fairly recent technology that has taken off and is expected to grow into a major player in the networking field. Relevant technologies such as 5G make use of virtualization techniques, and it is desirable that these technologies operate to the best of their abilities. Unfortunately, its development is still in the early stages, and the integration with other relevant technologies is still problematic, although projects have been manifesting themselves during the last year.

The integration of Segment Routing and Service Function Chaining as a VNF dissemination tool is a scarcely explored improvement to the already existing software in charge of VNF control and transmission. In this thesis, it was not possible to demonstrate the supposed synergy that the integration of NFV, NFV and SR could accomplish, but the project leaves a challenge for the future regarding the usage of the µONOS, Stratum and Trellis to achieve exactly what was not possible this time.

Bibliography

[1] D. Kreutz *et al*, “Software-defined networking: A comprehensive survey,”Proc. IEEE, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] R. Minjubi *et al*, ”Network function virtualization: state-of-the-art and research challenges,” IEEE COMMUNICATIONS SURVEYS AND TUTORIALS. 18(1). p.236-262, 2016

[3] P. Veitch, M. J. McGrath, and V. Bayon, “An instrumentation and analytics framework for optimal and robust NFV deployment,” Communications Magazine, IEEE, vol. 53, no. 2, pp. 126–133, Feb 2015

[4] OpenDaylight. BGP LS PCEP: Helium Release Notes, 2014, [online] Available: `https://wiki.opendaylight.org/view/BGP_LS_PCEP:Helium_Release_Notes`”.

[5] Md. Humayun Kabir, ”Software Defined Networking (SDN): A Revolution in Computer Network,” IOSR Journal of Computer Engineering (IOSR-JCE)e-ISSN: 2278-0661, p-ISSN: 2278-8727 Volume 15, Issue 5 (Nov. -Dec. 2013), PP 103-106 www.iosrjournals.org

[6] T. Das, A. Kushwaha, A. Gumaste, M. Gurusamy” Leveraging optics for network function virtualization in hybrid data centers,” 2018 International Conference on Optical Network Design and Modeling (ONDM), IEEE, pp. 71–76, May 2018, Dublin, Ireland

[7] C. Filsfils, *et al*, ”The Segment Routing Architecture,” 2015 IEEE Global Communications Conference (GLOBECOM), IEEE, 6-10 Dec. 2015, DOI: 10.1109/GLOCOM.2015.7417124

[8] E. Rosen A. Viswanathan *et al*, ”Multiprotocol Label Switching Architecture,” IETF RFC 3031, January 2001

[9] Introduction to Segment Routing, `https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/seg_routing/configuration/xe-3s/segrt-xe-3s-book/intro-seg-routing.pdf`. Last accessed 18 November 2019

[10] H. Kim and N. Feamster, “Improving network management with software defined networking,”Communications Magazine, IEEE, vol. 51,no. 2, pp. 114–119, 2013

[11] E. Haleplidis *et al*, ”Software-Defined Networking (SDN): Layers and Architecture Terminology,” RFC, vol.:7426, pag. 1-35, 2015

[12] https://wiki.onosproject.org/display/ONOS/Flow+Rule+Subsystem
Last accessed 26/11/2019

[13] https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2018_2/onos/arquitetura.html
Last accessed 25/11/2019

[14] F. Yousaf *et al*, "NFV and SDN—Key Technology Enablers for 5G Networks," IEEE Journal on Selected Areas in Communications, Volume: 35 , Issue: 11 , Nov. 2017, IEEE, 06 October 2017, DOI: 10.1109/JSAC.2017.2760418

[15] J. Nam *et al.*, "Probius: Automated Approach for VNF and Service Chain Analysis in Software-Defined NFV," SOSR Mar. 2018, pag. 1-13, DOI: 10.1145/3185467.3185495

[16] M. Taylor, "The Myth of the Carrier-Grade Cloud, 2015, http://www.metaswitch.com/the-switch/the-myth-of-the-carrier-grade-cloud. Last Accessed 27/11/2019

[17] https://osm.etsi.org/images/OSM_EUAG_White_Paper_OSM_Scope_and_Functionality.pdf
Last Accessed 02/12/2019

[18] J. Herrerra and J. Botero, "Resource Allocation in NFV: A Comprehensive Survey," IEEE Transactions on Network and Service Management, vol. 13, no. 3, pag. 518 - 532, 2016

[19] A. AbdelSalam *et al.*, "Implementation of Virtual Network Function Chaining through Segment Routing in a Linux-based NFV Infrastructure," IEEE Conference on Network Softwarization, Bologna, Italy, 3-7 July 2017.

[20] https://github.com/netgroup/SRv6-net-prog/ Last accessed 23/12/2019

[21] https://www.youtube.com/watch?v=frgbYpl2-cM - "Demystifying IPv6 over MPLS: Tackling the challenge of connecting IPv6 islands". Last Accessed 03/01/2020

[22] https://www3.lacnic.net/eventos/lacnic23/jueves/gianpietro-cisco-segmentrouting-ipv6-flip6.pdf. Last Accessed 03/01/2020

[23] S. Das, *et al.*, "Application-aware aggregationand traffic engineering in a converged packet-circuit network,"inOptical Fiber Communication Conference and Exposition(OFC/NFOEC), 2011 and the National Fiber Optic EngineersConference. IEEE, 2011, pp. 1–3.

[24] https://docs.docker.com/engine/install/ubuntu/. Last accessed 06/12/2020

[25] https://github.com/opennetworkinglab/onos/blob/master/Dockerfile. Last accessed 06/12/2020

[26] https://wiki.onosproject.org/pages/viewpage.action?pageId=2130918. Last accessed 22/12/2020

[27] https://wiki.opennetworking.org/display/COM/NG-SDN. Last accessed 23/12/2020

[28] https://opennetworking.org/wp-content/uploads/2019/09/2-pm-%C2%B5ONOS-Project-Overview.pdf. Last accessed 23/12/2020

[29] https://opennetworking.org/stratum/. Last accessed 23/12/2020

[30] https://wiki.opencord.org/display/CORD/Trellis. Last accessed 23/12/2020

Appendix A

TopoSR.py

Python code:

```
from mininet.topo import Topo\newline
from mininet.node import Host\newline
from requests import post\newline
from json import dumps\newline
from os import environ\newline

class IpHost(Host):\newline
  def __init__(self, name, gateway, *args, **kwargs):\newline
    super(IpHost, self).__init__(name, *args, **kwargs)\newline
    self.gateway = gateway

  def config(self, **kwargs):
    Host.config(self, **kwargs)
    mtu = "ifconfig "+self.name+"-eth0 mtu 1490"
    self.cmd(mtu)
    self.cmd('ip route add default via %s' % self.gateway)

class LeafSpine( Topo ):

  def __init__( self, spine, leaf, host ):

    Topo.__init__( self )

    netcfg = {"ports":{},"devices":{}, "hosts": {}, "links": {}}
    hostcfg = {}
```

```
spines = {}

for s in range(spine):
  sid = s+1+200
  name = 'spine%s' % (s+1)
  print(name+str(sid))
  ip = '192.168.0.%s' % sid
  mac = '00:00:00:00:00:%s' % format(sid,'02x')
  dpid = format(sid,'016x')
  netcfg["devices"]["of:%s" % dpid] = {
    "segmentrouting":{
      "ipv4NodeSid":sid,
      "ipv4Loopback":ip,
      "routerMac":mac,
      "isEdgeRouter":False,
      "adjacencySids":[]
    },
    "basic":{
      "name":name,
      "driver":"ofdpa-ovs"
    }
  }

  spines[s] = self.addSwitch(name, dpid=dpid)

for l in range(leaf):
  sid = l + 1 + 100
  name = 'leaf%s' % (l+1)
  print(name+str(sid))
  ip = '192.168.0.%s' % sid
  mac = '00:00:00:00:00:%s' % format(sid,'02x')
  dpid = format(sid,'016x')
  netcfg["devices"]["of:%s" % dpid] = {
    "segmentrouting":{
      "ipv4NodeSid":sid,
      "ipv4Loopback":ip,
      "routerMac":mac,
      "isEdgeRouter":True,
      "adjacencySids":[]
```

```
  },
  "basic":{
    "name":name,
    "driver":"ofdpa-ovs"
  }
}

leafSwitch = self.addSwitch(name, dpid=dpid)

for s in range(spine):
  spineSwitch = spines[s]
  netcfg["links"]["%s/%s-%s/%s" % (dpid,s+1,format(s+1,'016x'),l+1)] = {
    "basic": {}
  }
  netcfg["links"]["%s/%s-%s/%s" % (format(s+1,'016x'),l+1,dpid,s+1)] = {
    "basic": {}
  }

  self.addLink(leafSwitch,spineSwitch,port1=s+1,port2=l+1)

for h in range(host):
  hid = (l*host)+h+1
  name = 'h%s' % hid
  ip = '10.1.%s.%s' % (l+1, h+1)
  cidr = '%s/24' % ip
  router = '10.1.%s.254' % (l+1)
  mac = '00:00:00:00:01:%s' % format(hid,'02x')
  port = spine + h + 1
  location = "of:%s/%s" % (dpid, port)

  netcfg["hosts"]["%s/-1" % mac] = {
    "basic": {
       "ips": [ ip ],
       "locations": [ location ]
    }
  }

  endHost = self.addHost(name, cls=IpHost, mac=mac, ip=cidr, gateway=router)
```

```
        netcfg["ports"][location] = {
          "interfaces": [
            {
              "ips":["%s/24" % router],
              "vlan-untagged":1
            }
          ]
        }

        self.addLink(leafSwitch,endHost,port1=port)

    onos = environ.get('ONOS','172.17.0.2')
    post('http://'+onos+':8181/onos/v1/network/configuration/', auth=('onos','rocks'), json=netcfg).

topos = { 'TopoSR': ( lambda spine=4,leaf=3,host=2: LeafSpine(int(spine),int(leaf),int(host)) ) }
```

Appendix B

post.py

```
import requests
import json

site="http://172.17.0.2:8181/onos/v1"
f=open('txt.json')
data=f.read()
f.close()
req=requests.post(site+"/dhcp/mappings",data=data,auth=('onos','rocks'))
print(req.status_code)
print(req.text)
print(req)
```