Ilpo Litmanen

Segment Routing

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

1 April 2017

Metropolia

Author Title Number of Pages Date	Ilpo Litmanen Segment routing 43 pages + 3 appendices 1 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Networking Technology
Instructor	Erik Pätynen, Senior Lecturer

The goal of this project was to study segment routing and test its traffic engineering capabilities in MPLS networks by creating a working virtualized testing environment. In addition, another goal was to produce clear instructions for setting up the testing environment and running tests in it.

The testing of segment routing capabilities was carried out by running a VirtualBox hypervisor with Open Network Foundation's SPRING-OPEN project virtual machine installed on a PC. The virtual machine included a segment routing testing environment, based on the Open Network Operating System and Mininet.

A documented, working testing environment was built using the SPRING-OPEN project virtual machine. In addition, a customized segment routing network topology was built using self-made scripts and configuration files. Basic segment routing features were analyzed and tested.

The results showed that segment routing can be used to introduce fine grained traffic control into networks with small changes into pre-existing network protocols. The testing environment performed well with basic tasks, but some issues existed with more advanced segment routing features.

Keywords	Segment Routing, ONOS, Mininet, Networking

Tekijä Otsikko Sivumäärä Aika	Ilpo Litmanen Segmenttireititys 43 sivua + 3 liitettä 1.4.2017
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Tietoverkot
Ohjaaja	Lehtori Erik Pätynen

Insinöörityön tarkoituksena oli tutustua segmenttireititykseen (segment routing) ja testata sen soveltuvuutta tietoliikenneverkon hallintaan. Segmenttireititys on tarkoitettu suurten tietoliikenneverkkojen tarpeisiin, silloin kun hallintaa ei pystytä toteuttamaan riittävän tarkasti nykyisin käytössä olevien reititysprotokollien avulla. Segmenttireitityksen on suunniteltu toimivan laajennuksena nykyisiin reititysprotokolliin, eikä sen käyttö näin ollen vaadi uusien verkkolaitteiden hankkimista.

Työssä tutustuttiin ensin segmenttireitityksen perusteisiin ja arkkitehtuuriin lyhyiden esimerkkien avulla. Tämän jälkeen konfiguroitiin virtualisoitu testiympäristö kotitietokoneelle käyttäen pohjana segmenttireitityksen demonstrointiin tarkoitettua virtuaalikonetta (Open Network Foundation SPRING-OPEN project virtual machine). Testiympäristön ja testausvaiheen konfiguraatioista laadittiin seikkaperäinen ohjeistus, jotta työn käytännönvaihe olisi mahdollisimman helppoa toistaa tarvittaessa toisella tietokoneella.

Virtuaalikoneen demonstrointiin tarkoitettujen esitysten koodia muokattiin niin, että testiosiossa pystyttiin helposti kokeilemaan segmenttireitityksen perusominaisuuksia. Testausvaiheessa määriteltiin virtualisoituun verkkoon segmenttireititystunneleita ja -käytänteitä, minkä jälkeen virtuaaliverkon liikennettä analysoitiin siihen tarkoitetulla työkalulla.

Testiympäristössä suoritetut testit osoittivat, että segmenttireititystä voidaan käyttää tietoliikenneverkkojen hallitsemiseen. Segmenttireitityksen aiheuttama hallintaliikenne (Control Traffic Overhead) on vähäistä verrattuna muihin vastaaviin ratkaisuihin.

Testiympäristö ei toiminut odotetusti silloin, kun virtualisoituun verkkoon aiheutettiin keinotekoisesti häiriöitä. Alun perin segmenttireitityksen demonstrointiin tarkoitettu ympäristö hidastui merkittävästi häiriötilanteessa, eikä se kyennyt itsenäisesti toimimaan segmenttireititysarkkitehtuurin mukaisesti.

Avainsanat	Lähdereititys, Segment Routing, MPLS, ONOS, Mininet

Metropolia

Contents

List of Abbreviations

1 Introduction 1

2 Network Control Architectures 2

2.1 Traditional Network Control 2
2.2 Programmable Network Control 2
2.3 Network Control through Source Routing 3

3 Segment Routing 4

3.1 Segment Routing Architecture 4
3.1.1 Data Plane 5
3.1.2 Control Plane 6
3.1.3 Traffic Engineering with Segments 7
3.2 Segment Routing in MPLS 10
3.3 Segment Routing in IPv6 11
3.3.1 Header Chaining 12
3.3.2 Segment Routing Header 13
3.4 Security Considerations 15

4 SR Demonstration Environment 17

4.1 Environment Requirements 17
4.2 Virtual Machine Setup 17

5 Customizing the Environment for SR Tests 19

5.1 Customizations 19
5.1.1 ONOS Controller Setup 20
5.1.2 Mininet Setup 21
5.2 Verify System Functionality 23
5.3 Traffic Engineering with Segment Routing 24
5.3.1 Implementing SR Rules 27
5.3.2 Testing the SR Implementation 31

6 Discussion 39

7 Conclusion 40

References 41

Appendixes

Appendix 1. ONOS Controller Configuration File

Appendix 2. Mininet Network Configuration Script

Appendix 3. Mininet Custom Topology Script

List of Abbreviations

ACL	Access Control List
CLI	Command Line Interface
ECMP	Equal-cost Multi-path
ICMP	Internet Control Message Protocol
MPLS	Multiprotocol Label Switching
ONOS	Network Operating System
PCE	Path Computation Element
SR	Segment Routing
SDN	Software-Defined Networking
SID	Segment Identifier
SLA	Service Level Agreement
SPF	Shortest Path First
SPRING	Source Packet Routing in Networking
SRH	Segment Routing Header
SSH	Secure Shell
TE	Traffic Engineering
TLV	Type Length Value
VM	Virtual Machine

1 Introduction

Computing and storage solutions have evolved rapidly during the past three decades [1] and the number of worldwide internet users has grown in just two decades from 16 million users tracked in 1995 [2] to 3.2 billion users in 2015 [3]. The number of internet connected devices has also grown rapidly, in the year 2015 there were 13.4 billion Internet of Things (IoT) devices [4]. Over the same time period, computer networking core architecture has remained largely unchanged. Demands for scalability and granularity in modern networks are on such a high level that the traditional networking approaches, where networking devices make independent routing decisions running several distributed protocols, have become a limiting factor in terms of innovation and growth in the industry. [1.]

A solution to this dilemma is the source routing paradigm, in which a sender of a packet can specify, either partly or completely, a route that the packet should take while traversing the network [5]. Source routing is the core design principle behind a new network protocol called segment routing (SR). It can be implemented to existing networks without costly hardware updates, through a small number of software extensions to routing protocols [6, 1-3].

The objective of this project is to study segment routing and test its traffic engineering (TE) capabilities in Multiprotocol Label Switching (MPLS) networks by building a virtualized testing environment. The plan was to create a virtualized testing network, where networking devices are controlled by a centralized controller. The controller implements SR rules to the networking nodes by creating network tunnels and policies.

In order to create the testing environment, a customized version of Open Network Foundation's (ONF) SPRING-OPEN project virtual demonstration environment was used. The environment is based on Open Network Operating System (ONOS) and Mininet – a network virtualization program.

2 Network Control Architectures

In this chapter I will describe how large network operators have used different network control architectures in the past to implement network control and traffic engineering characteristics into their networks. The scalability issues with these architectures will be discussed. Finally, I will introduce a new network tunneling technology called segment routing.

2.1 Traditional Network Control

Generally, the packet routing and switching architecture can be divided into data, control, and management planes to describe the core functionality of an intermediary network device. The data plane handles packet forwarding between ingress and egress interfaces according to lookup tables. The main role of the control plane is to generate routing lookup tables for the data plane. In order to do this, the control plane needs to compute active network topology jointly with different control protocols running in the network, using an onboard Path Computation Element (PCE). The management plane is used to configure and monitor the intermediary network device. [7, 7-8.]

Operators of large computer networks, typically IP and MPLS networks, anticipate increasing demand for services based on Service Level Agreements (SLA) [6]. In order to provide new SLA-bound, custom tailored service offerings for the clients, the operators require flexible TE capabilities. Resource Reservation Protocol (RSVP) and Label Distribution Protocol (LDP) are the two widely used protocols that offer granular network control over forwarding paths. However, scalability is an issue with these protocols due to increased control plane signaling and the requirement for midpoint state management. In practice RSVP and LDP offer only coarse traffic engineering capabilities. [8.]

2.2 Programmable Network Control

Software-defined networking (SDN) is an emerging network architecture that decouples the data and control plane functionality, which enables directly programmable and flexible network control. In the SDN architecture path computation is done via logically

centralized, software-based SDN controllers that maintain a global view of the network. This architecture uses an open standardized OpenFlow protocol as an interface between an SDN capable switch and a controller. The controller directs forwarding behavior in the network, by instructing the SDN switches to install new forwarding paths called flows in their flow tables. These flow tables are implemented by using Ternary Content Addressable Memory (TCAM) on the SDN switch hardware. In general, TCAM can support several thousand flow entries. Purely SDN based solutions offer fine-grained network controllability, but require keeping large quantity of network information (flows) in midpoint devices. SDN capable switches also require special hardware in order to support SDN and OpenFlow. [9; 10, 44-55.]

2.3 Network Control through Source Routing

Segment routing is a new network tunneling technology focused on providing a simplified, highly scalable and easy to operate solution for TE requirements in existing IP and MPLS networks [11]. In the SR architecture only an ingress network device needs to maintain forwarding information; hence the typical midpoint scalability issues in large networks are negated [6, 1-3].

A segment routing concept innovated by Cisco was proposed to operators in November 2012 [12] and the Internet Engineering Task Force (IETF) workgroup called Source Packet Routing in Networking (SPRING) was formed in October 2013 [13]. It began standardization of the SR architecture in November 2014 [14]. At the time of writing this thesis, the architecture is currently in revision 8 [14] and Cisco reports [12] that first SR capable systems were deployed in 2015. According to Cisco [12], segment routing has received strong operator endorsement and multi-vendor consensus in the networking community.

3 Segment Routing

This section describes the segment routing architecture and its main components. Data and control plane implementations in SR will be covered, along with SR terminology. A series of abstract examples will demonstrate the traffic engineering capabilities of SR. The MPLS and IPv6 data plane technology implementations of SR will be covered in detail and, finally, security considerations with the SR architecture will be discussed.

3.1 Segment Routing Architecture

An edge device, typically host or a router, acts as a steering entity in the SR design. This steering capability is achieved by using an ordered list of instructions called *segments*. The list is included in the packet that traverses the network by prepending a segment routing header (SRH) to it. A segment can act as an identifier either for a topological (path selection), or for a service instruction (packet delivery to a service). The segment list consists of these segment identifiers (SIDs). A SID can have either local or global significance to a node within an SR-domain. These SR capable nodes can be physically connected in the same network infrastructure or remotely connected, through a VPN tunnel for example. Since the forwarding information is prepended with an SRH to the packet at the ingress node, the midpoint nodes in the core network are not required to maintain state information. This, and the fact that SR supports the Equal Cost Multipath (ECMP) routing feature of the IP architecture, bring drastic gains in terms of network scalability. [6, 1-2.]

The SR architecture is a combination of traditional distributed network intelligence and centralized network control. In a typical use case, SR leverages existing intra-domain (IGP) and inter-domain (EGP) protocols for traditional routing of packets. A centralized network controller that has a global view of the network can be utilized to compute a path for a packet with an unknown destination or SLA requirement, when received by an ingress node to an SR-domain. [6, 2; 15.]

Main Components

The SR architecture has two main components: *data plane* and *control plane*, upon which the SR framework is implemented. The data plane of SR defines the encoding

process of the SRH to a packet and the instructions how each node on a packet's traversal path should process the packet. The control plane of the SR architecture defines how nodes in an SR-domain should be identified, and how devices should process the segment lists. [6, 2.]

3.1.1 Data Plane

The MPLS and IPv6 data plane technology implementations of SR are currently included in the SR standardization effort. An abstraction of SR data plane architecture is described below. An SRH of a packet contains an ordered list of segments. There are several kinds of segments that can be used. The four most fundamental segment types are:

- **Node-SID**, also referred to as IGP-Node, a unique identifier for a node in an SR-domain.
- **Adjacency-SID**, also referred to as IGP-Adjacency, an identifier for an egress data link with an adjacent node. Typically advertised as a locally significant segment.
- **Service-SID**, an identifier for a service, provided by a node processing the packet, for example a security inspection service. Advertised as a locally significant segment.
- **Anycast-SID,** an identifier which does not identify a specific node, but a group of nodes that have a common identifier. This functionality enables ECMP routing.

In order to support SR, a node needs to be capable of performing the following SR data plane actions:

- NEXT – Moves an SRH segment pointer to point to the next active SID in the ordered segment list after executing the previous segment.
- PUSH – Updates the segment list with a new segment. The segment goes top of the list and the pointer is set to point to the newly added segment, making it the active segment.
- CONTINUE – Forwards the packet without any changes to the segment list or the SRH pointer target. [6, 2;16, 6.]

The SRH segment pointer is a pointer that points to an active segment in the list of segments.

3.1.2 Control Plane

The SR control plane defines how communication of SID information is conducted within an SR-domain. Node and Adjacency-SIDs are advertised in the network via IGP protocols. The IGP protocol used for SR control plane signaling requires an extension to the protocol that supports SR. The two most used link state IGP protocols that support SID advertisement are Intermediate System to Intermediate System (IS-IS) and Open Shortest Path First (OSPF). Border Gateway Protocol (BGP) SR extension is also undergoing a standardization process. [6, 3; 17; 18; 19.]

The SR control plane also directs how an ingress node should select a traversal path for a packet with an unknown destination or SLA requirement. The following three methods are available in the SR architecture:

- **Static configuration.** A static SR tunnel can be manually configured on an SR enabled node. This method is typically used in troubleshooting or testing situations as a temporary aiding solution.
- **Distributed Constrained SPF (CSPF) path computation.** An ingress node uses its own internal SR PCE to calculate the shortest path to a destination and then generates an SRH for the associated packet.
- **Utilizing centralized controller.** An SR capable node can request for a route to an unknown destination from a known centralized controller (e.g. an SDN oriented OpenDaylight controller). The controller, with a global view of the network, then computes the best path to the destination in accordance with any SLA requirements received from the path requesting node. The computed path is then returned to the requesting node, which installs the received path to its routing table. Furthermore, the node generates an SRH for the associated packet.

All these methods described above can be run one at a time or in parallel in any combination, depending on the requirements of a network in question. The centralized

controller approach of SR offers great traffic engineering capability, without increasing control plane overhead at the midpoint nodes. [6, 3.]

3.1.3 Traffic Engineering with Segments

The path of a packet in an SR-domain depends on the segments and their types included in the SRH segment list. Four segment types (*Node Segment, Adjacency Segment, Service Segment* and *Anycast Segment*) are introduced through abstract examples in this section. It should be noted that in the following examples each node forwards packets using the normal SPF behavior and the default weight value is set to 10, unless explicitly told otherwise.

If a Node-SID is added to a packet's SRH segment list, the packet will have to travel to the specified node first. Figure 1 demonstrates an example, where packets from node N1 have their destination set as node N6.

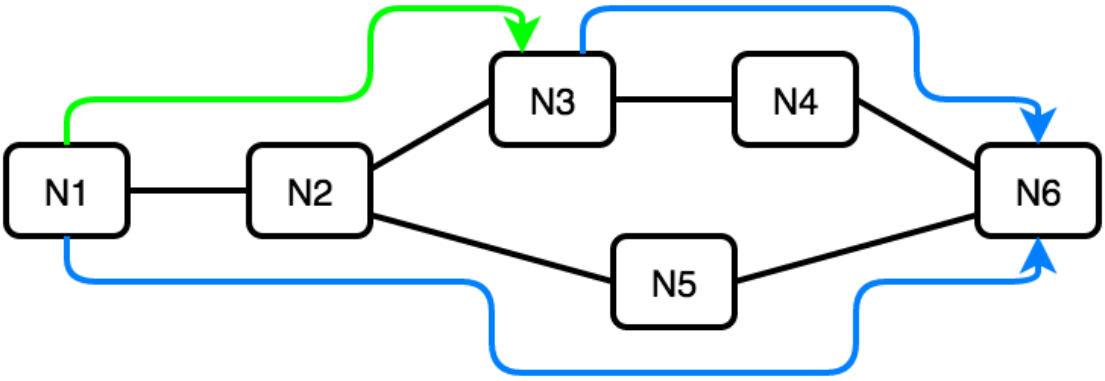

Figure 1. A comparison between normal SPF routing and *Node Segment* steering.

The blue arrows in Figure 1 show the normal SPF routing behavior. The shortest path from N1 to N6 goes through N2 and N5. If the packets are prepended with an SRH that has a Node-SID pointing to N3, the packet flow will be forced to reach N3 first, as shown with the green arrow. From N3 onward towards N6, the packets will be routed using the normal SPF behavior, which will result in an N1-N2-N3-N4-N6 path. With SR, N1 has now control over the packet's path in the SR-domain.

More granular TE can be implemented by using *Adjacency Segments.* Figure 2 depicts a scenario where Adjacency-SIDs can be used to specify which link should be utilized between two nodes, if multiple links exist.

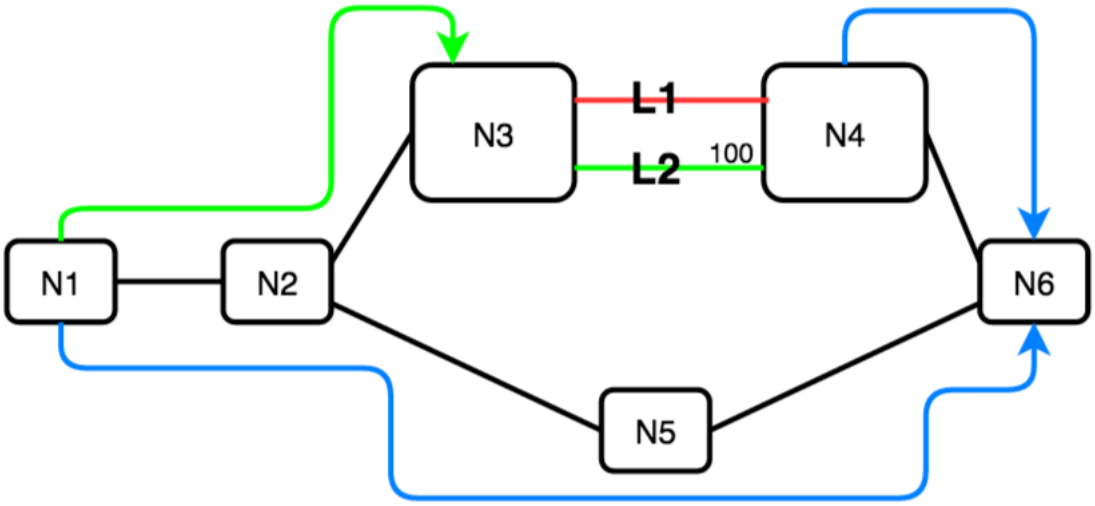

Figure 2. An *Adjacency Segment* implementation.

In Figure 2 there are two links L1 (red line) and L2 (green line) between nodes N3 and N4. The link L2 has a higher weight value of 100. Without SR, packets from node N1 to N6 will traverse through N5 (lower blue arrow). If the packets have an SRH that has a Node-SID pointing to N3, the packets will first traverse to N3 and then through the link L1 to N4 and finally to N6. L1 is being utilized, because it has a lower SPF weight value than L2. An Adjacency-SID L2 can be added to the packet's SRH segment list to force utilization of an alternative link between N3 and N4. N3 will notice the active Adjacency-SID and the packets will flow through the link L2. The Adjacency-SIDs are mostly advertised with local significance, as global uniqueness is usually not required [16, 6].

Service Function Chaining

Service provider grade networks often offer services that are beyond simple connectivity and routing services. These additional services are normally implemented through using a variety of appliance devices, which are independent of the other networking devices that handle the normal routing of traffic. This approach has proven inflexible in large networks, the appliance devices cause a significant amount of

management overhead, which in turn results in high operational expenditure (OPEX). [6, 4.]

In 2012, a group of operators and Network Equipment Manufacturers (NEMs) jointly proposed a new architecture, called Network Functions Virtualization (NFV), to alleviate the known scalability issues with the old additional services approach [20]. The new NFV approach aims to virtualize the various additional services required in large networks. Even though NFV provides the operators more control on how to implement the additional services, it does not solve the problem of how to implement these services into networks without causing high implementation and management overhead [6, 4]. A new technique, called Service Function Chaining (SFC), was proposed to reduce the OPEX of the added network services in large networks [21]. The SFC architecture instantiates an ordered list of services in the packet's header, thus removing the constraints associated with the physical location of the additional services. The SR architecture can support SFC through *Service Segments.* [6, 4.]. Figure 3 depicts how SFC can be implemented in SR capable networks.

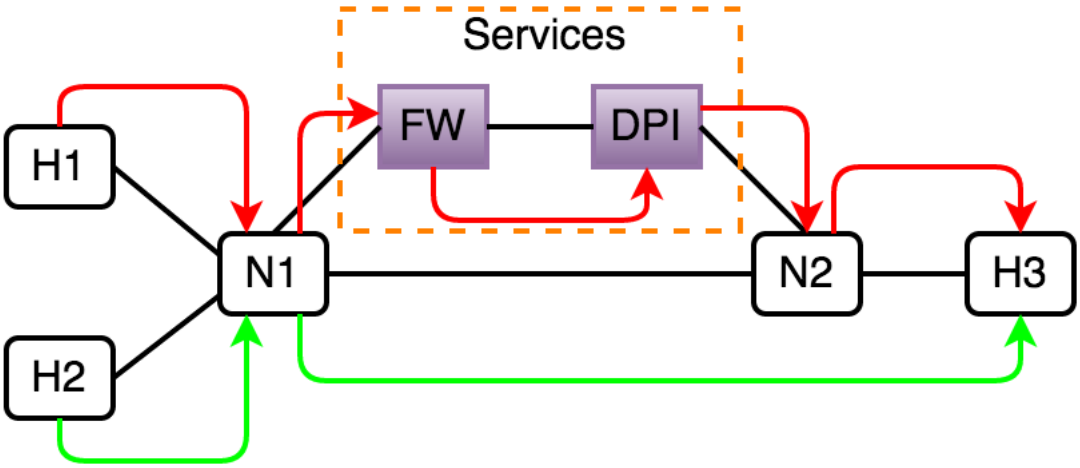

Figure 3. SFC in segment routing using *Service Segments.*

There are two hosts, H1 and H2 that are trying to send packets to host H3 depicted in the example given in Figure 3. H1 is an untrusted host and H2 is a trusted host. When H1 sends packets to SR-domain ingress node N1, the node will add an SRH that has two Service-SIDs for firewall (FW) and deep packet inspection (DPI) services and a Node-SID labelled N2. The packets will go through the security inspection services first and only then will the packets be allowed to traverse towards the inner nodes of the

SR-domain. The packets originated from H2 will receive an SRH that contains only the Node-SID N2, since H2 is trusted by node N1 in this imaginary example. The Service-SIDs can be advertised as locally significant, since only N1 needs to know about the additional services from a data plane perspective.

The nodes in an SR-domain can have one or more Node-SIDs assigned and two or more nodes can share the same Node-SID. These *Anycast Segments* make it possible to load-balance the traffic to two or more nodes. In Figure 4 two nodes have an additional, shared Anycast-SID.

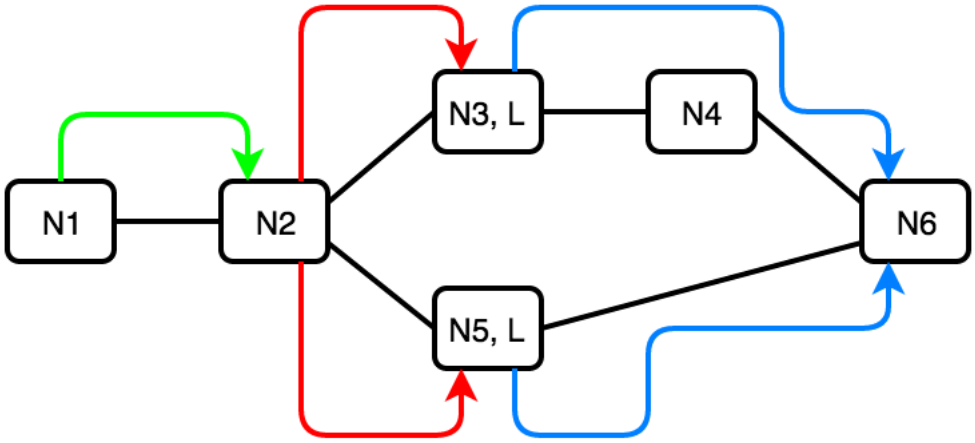

Figure 4. ECMP routing support in SR using *Anycast Segments.*

In the example topology shown in Figure 4, an operator of the network can instruct ingress node N1 to prepend a packet, from N1 to N6, with an SRH containing a Node-SID N2 and an Anycast-SID L. Once the prepended packets arrive to node N2 (green arrow), it will notice that there are two paths available to reach the next segment L. Both paths have the same cost from the N2 perspective, so the node will use ECMP routing to load-balance the traffic over the two links reaching the L segment.

3.2 Segment Routing in MPLS

The MPLS architecture uses labels (a 4-byte identifier) to route traffic within an MPLS domain, instead of protocol dependent (e.g. IPv4 or IPv6) addresses. The core idea of MPLS is to be able to transport multiple protocols in one unified network infrastructure (an MPLS domain). [22, 5-7.]

SR can be applied on top of the traditional MPLS data plane without making any changes to the plane itself. In the MPLS implementation of SR, a segment is encoded as an MPLS label and an ordered list of segments as a stack of MPLS labels. MPLS with the SR extension offers simplified service tunneling, by running the ISIS or OSPF protocol without any additional TE protocols that cause scalability issues. If required, SR can co-exist with other MPLS signaling protocols (e.g. LDP and RSVP). The required SR control plane operations can be directly mapped to normal MPLS actions, as depicted in Table 1. [23, 3; 6, 2.]

Table 1. SR operations mapping to MPLS operations. Copied from [6, 2].

SR	**MPLS**
SR Header	Label Stack
Active Segment	Topmost Label
PUSH Operation	Label Push
NEXT Operation	Label POP
CONTINUE Operation	Label Swap

Table 1 shows mapping of each SR control plane operation with a corresponding MPLS operation. The basic functionality of each corresponding operation in both MPLS and SR are the same.

3.3 Segment Routing in IPv6

As mentioned before, SR can be encoded into the IPv6 architecture with a new type of routing extension header. The SR extension to the IPv6 architecture is an evolution of the routing header type 0 (RH0), which has been deprecated due to security issues [24]. In the IPv6 implementation of SR, a segment is directly encoded as an IPv6 address and the SRH segment list as an ordered list of IPv6 addresses. There is no need to introduce a new SR identifier (label), because the SIDs can be presented as IPv6 prefixes in the control plane. As with the MPLS implementation of SR, the IPv6 SR implementation only requires addition of extensions to the existing IGPs. [25.]

3.3.1 Header Chaining

The fixed header of IPv6 features a *Next Header* field. This 8-bit field is used to indicate the type of the header that immediately follows the outer IPv6 header. It can indicate either the upper layer protocol (e.g. TCP) used by the packet's payload or which IPv6 extension header follows next. A request has been sent to IANA, to assign a new routing type value (suggested value 4) for SRH. [26, 4-8; 25, 8.]

In the IPv6 implementation of SR, the SRH is implemented as shown in Figure 5.

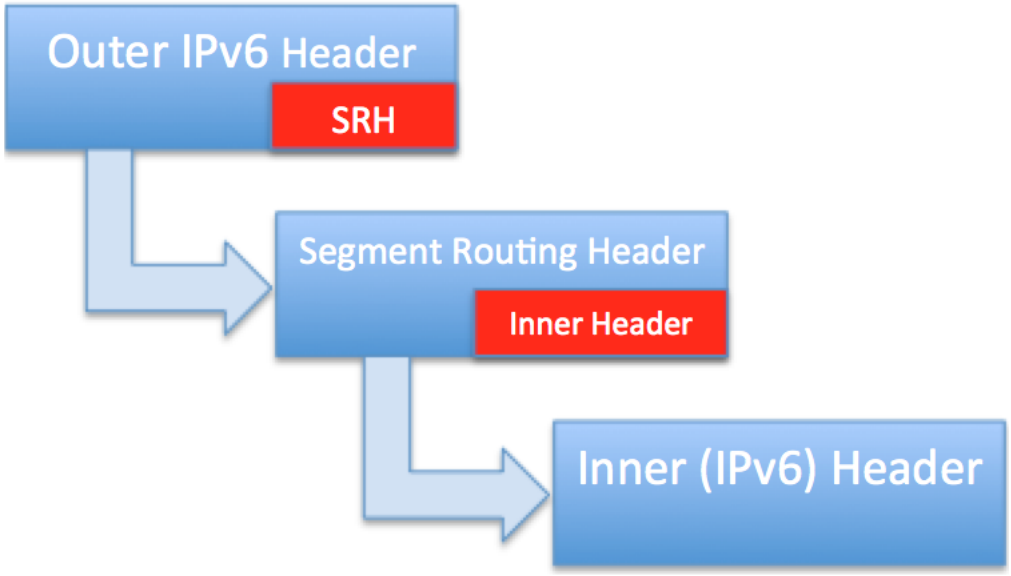

Figure 5. IPv6 extension header chaining in the IPv6 SR implementation.

The header chaining process shown in Figure 5 is used when a packet that requires SR is received by an ingress node to an SR-domain. **First,** the packet's header is prepended with an SRH that has the next header field set to indicate the type of the original packet's header. **Second,** the SRH and original packet (including the inner header) is encapsulated inside an outer IPv6 header, which in turn has the next header field set to indicate that an SRH type header follows next. It should be noted that the IPv6 SR implementation is designed to also allow transportation of other protocols than IPv6 (e.g. IPv4) as the inner payload packet. [25, 5.]

3.3.2 Segment Routing Header

The SRH format shown in Figure 6 is an extended version of the old RH0 format defined in the IPv6 specification [26, 14].

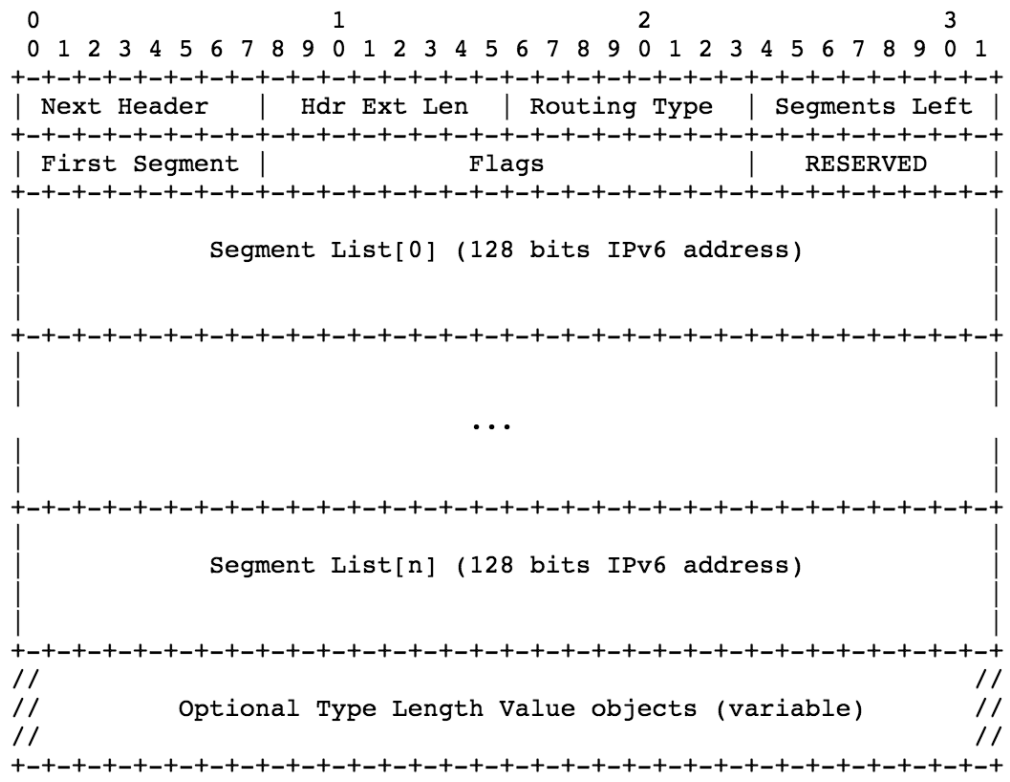

Figure 6. Format of SRH in the IPv6 implementation. Copied from [25, 8].

Figure 6 shows the different fields in the SRH that are either required or optional. The fields of the SRH are:

- **Next Header**, an 8-bit selector value. Identifies the inner header as explained in section 3.3.1.
- **Hdr Ext Len**, an 8-bit value. Used to indicate length of an SRH.
- **Routing Type**, an 8-bit value. Identifies the type as SRH. To be assigned by IANA.
- **Segments Left**, a decremental index number. Points to the current active segment.
- **First Segment**, an index number that points to the last element on the segment list. The last element of the list is the first segment on the SR traversal path.
- **Flags**, a 16-bit value. The values will be described after the listing.

- **RESERVED**, no planned use for now, should be unset and ignored.
- **Segment List [n]**, an ordered segment list of 128-bit IPv6 addresses.
- **Optional Type Length Value (TLV) objects**, the objects will be described later in this section. [25, 8-9.]

Flag Field Values

Flags are used to inform the processing node about SRH's conditions and of any additional content an SRH might have. The structure of the flag field is shown in Figure 7.

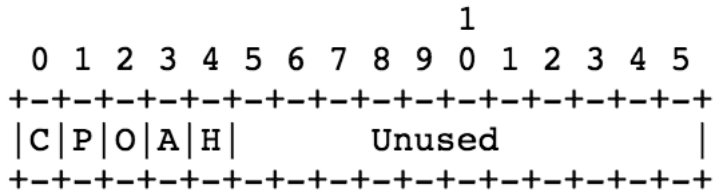

Figure 7. The SRH flag field structure. Copied from [25, 9].

Each flag shown in Figure 7 can have the bit value set as 1 or 0. If the bit value of a flag is 1, then the flag is set to be true. There are currently five defined flags, which are:

- **C-flag**, a clean-up flag. If this flag is set and there are no more segments left (Segments Left value has reached 0), the node processing the SRH needs to remove the Outer IPv6 encapsulation and the SRH header before forwarding the original packet.
- **P-flag**, a protected flag. Indicates whether the packet has been rerouted by an SR endpoint node or not.
- **O-flag**, used as an indicator for Operations and Management (OAM) packets.
- **A-flag**, An alert flag. Indicates that the SRH includes one or more Type Length Value (TLV) objects.
- **H-flag**, a hash-based message authentication code flag. Indicates presence of an HMAC TLV security object.

The unused flag bits are reserved for future use. [25, 9.]

TLV Objects

There are several optional TLV objects defined in the SR-IPv6 standardization draft and additional TLVs may be defined in the future. The primary use of TLVs is to carry information about the source routed packet's traversal path within the SR-domain (e.g. the ingress point and expected egress point). Each TLV type has its own format and several TLVs can be encoded into an SRH. The following five TLVs have been defined in the standardization draft:

- **Ingress Node TLV**, identifies the ingress node through which the packet entered the SR-domain.
- **Egress Node TLV**, identifies the egress node where the packet is expected to exit the SR-domain.
- **HMAC** TLV, contains HMAC and HMAC key ID. Always placed as the last TLV of the SRH. More details are given in the next section.
- **Opaque Container TLV**, a 128-bit data container. The object is used to transport data not relevant to the routing layer.
- **Padding TLV**, used to align the SRH on an 8 octet boundary.

The A-flag indicates the presence of one or more TLVs in an SRH. [26, 8-14.]

3.4 Security Considerations

The MPLS implementation of SR does not introduce any new security concerns since the SR architecture leverages the pre-existing MPLS data plane and its operations [25, 19]. Source routing in the IPv6 data plane has proven challenging from the security perspective in the past. The old RH0 source routing architecture had to be deprecated due to a severe amplification attack vulnerability in the architecture [24]. The SR IPv6 implementation renders the amplification attack vulnerability invalid by introducing optional SRH validity check at the SR-domain edge.

The SR-domain edge nodes can be configured to accept packets with an SRH explicitly from nodes within the SR-domain and discard all other outside source packets that have an SRH included. In addition, the SRH architecture features HMAC signatures that can be used to authorize and validate legitimate outside source packets with an SRH. [25,18-19.]

Adding the SRH validity checking has a small impact on routing performance. The originating source that crafts the SRH needs to compute the HMAC TLV and the ingress node to an SR-domain needs to validate it. The transit nodes downstream are not required to inspect the HMAC field. [25, 20-21.]

Figure 8 outlines the basic functionality of an edge node that acts as an ingress node to the SR-domain.

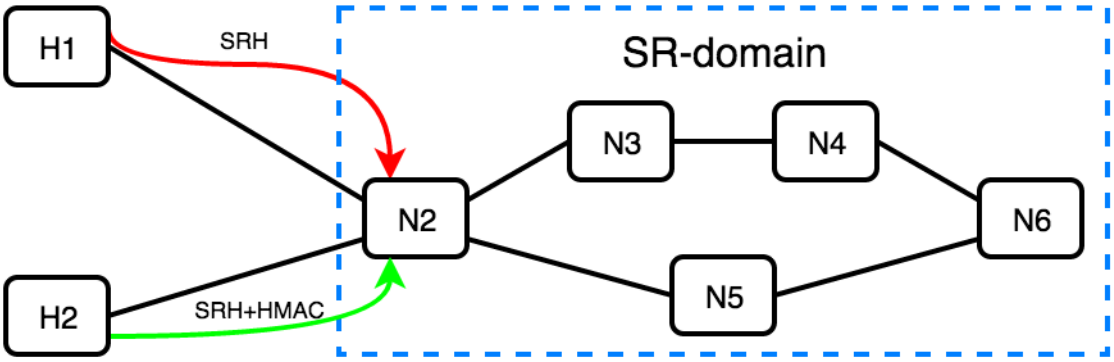

Figure 8. Validating outside source packets at the SR-domain edge.

In Figure 8 host H1 tries to send a packet with an SRH to N2, but N2 drops the packet according to local policy. N2 can be configured to refuse all outside source packets that include an SRH by default. To allow legitimate outside source packets with SRH, the N2 ingress node can be configured to do so. In order to do this, the outside source host H2 extends the SRH with an HMAC TLV and sets the H-flag to inform N2 that the SRH contains the HMAC TLV that needs to be verified first. The HMAC TLV is computed using a pre-shared key (known by both H2 and N2) and of the text which is a concatenation of:

- The source IPv6 address
- The first segment field
- A special bit octet crafted from the C-flag
- HMAC Key-ID
- All IPv6 addresses in the segment list. [25, 20.]

4 SR Demonstration Environment

In this part I will describe how to set up a segment routing virtual demonstration environment and how to configure a virtual network between SR capable software routers and hosts. I chose to use ONF's SPRING-OPEN project [27] virtual demonstration environment based on ONOS. In addition, I created a custom network topology for testing purposes. The project's virtual machine image includes an ONOS software controller, which provides an SDN based PCE for an island of SR capable software routers run in Mininet. The routers exchange unicast IPv4 packets with standard MPLS operations, which can be utilized to implement SR features in the network. The goal of this section is to provide a comprehensive guide on how to install and configure the demonstration environment VM.

4.1 Environment Requirements

The ONOS SPRING-OPEN project VM image has a 64-bit Ubuntu installed as the guest OS and the VM is preconfigured to run with two virtual CPUs and two GB of RAM. These are the minimum requirements to run the environment. I used my Desktop PC that has Microsoft Windows 8.1 OS and Oracle's VirtualBox hypervisor installed. The system was run with six core 4.6 GHz Hyper Threaded CPU and 32 GB of RAM. It should be noted that the VM and the tests can be run with significantly lower spec hardware if need be.

I decided to upgrade the default VM configuration slightly, by raising the CPU core count to six cores and the system memory to 12 GB RAM. The upgrades to the configuration were done due to abundant resources available in the host system and in hope of having better use experience.

4.2 Virtual Machine Setup

After starting up the VirtualBox hypervisor program (version 5.1.2 at the time of writing), the ONOS SPRING-OPEN project VM image [28] should be imported and the aforementioned configuration changes can be made to it. In addition, another

virtualized network interface should be added from VM network settings as shown in Figure 9 to allow SSH connection from the host OS to the guest OS.

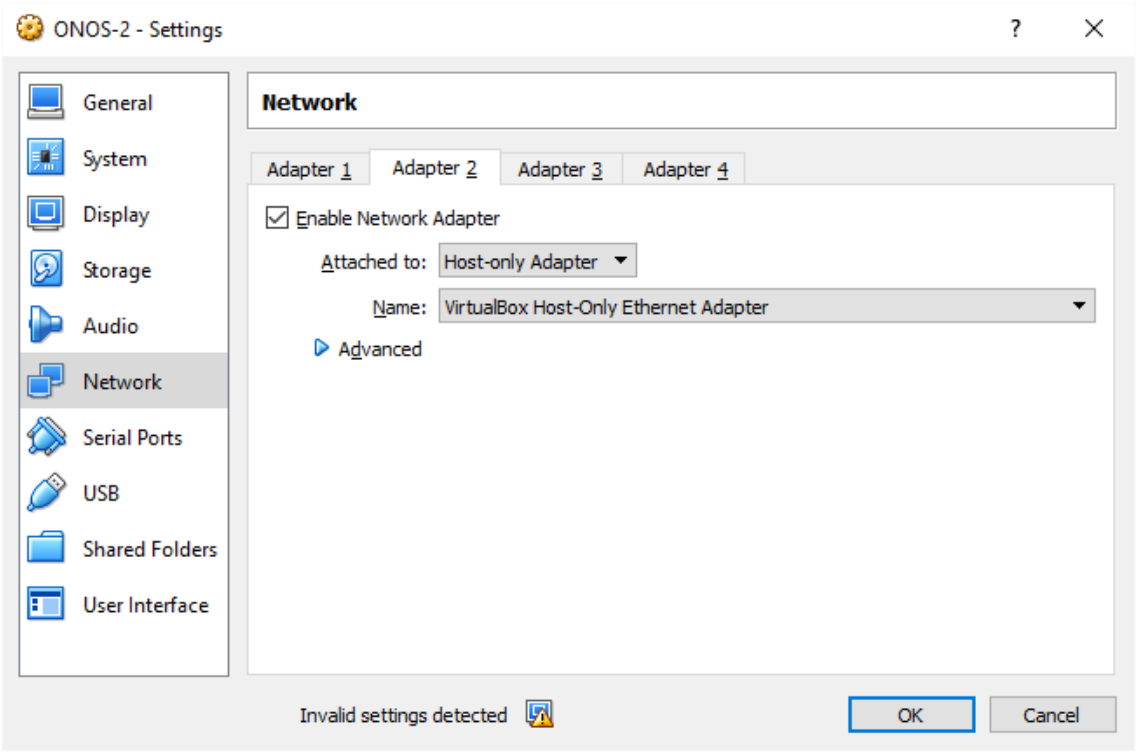

Figure 9. Addition of another network adapter. Screenshot [1].

After the VirtualBox VM import task and the additional configurations shown in Figure 9 are done, the VM should be started. The VM is preconfigured with an SSH server allowing local connections to it. VirtualBox has preconfigured port forwarding settings to allow SSH connection to the guest VM through localhost IP 127.0.0.1 and port 2222. From this step forward, there is no need to open a console connection to the VM within the hypervisor. All further actions can be made over an SSH terminal connection to the VM. How to form an SSH terminal connection to the VM is not explained here as it is not in the scope of this project. At this stage, the VM setup is completed.

5 Customizing the Environment for SR Tests

This section includes the setup and verification of a custom SR test application, utilizing Mininet and the ONOS controller. Once verified, I will describe in detail how to create tunnels and policies to implement SR in MPLS data plane.

The practical tests will include a SR path selection process according to specific simulated application demands (VOIP and file transfer traffic flows). Simple ICMP ping echo requests and replies will be used to simulate the traffic flows in the network. The tests will include the use of Node and Adjacency-SIDs to guide the traffic flows.

5.1 Customizations

I decided to create a custom network topology for segment routing tests. The VM image includes a few sample topologies for testing, but they lack link delay settings. By adding link delay, it is easier to visualize how TE with SR can be useful. By creating a customized test environment, I also gained inner sight into how the topologies are created in the ONOS virtual environment and thus have a better overall understanding of the system.

I chose to create a topology of three hosts and five SR capable routers. The topology has a loose resemblance to the real world. The routers (s1 through s5) have been named after internet exchange points in Finland (FICIX), Estonia (TLLIX), Germany (DECIX), Sweden (STHIX) and the Netherlands (AMSIX) as shown in Figure 10.

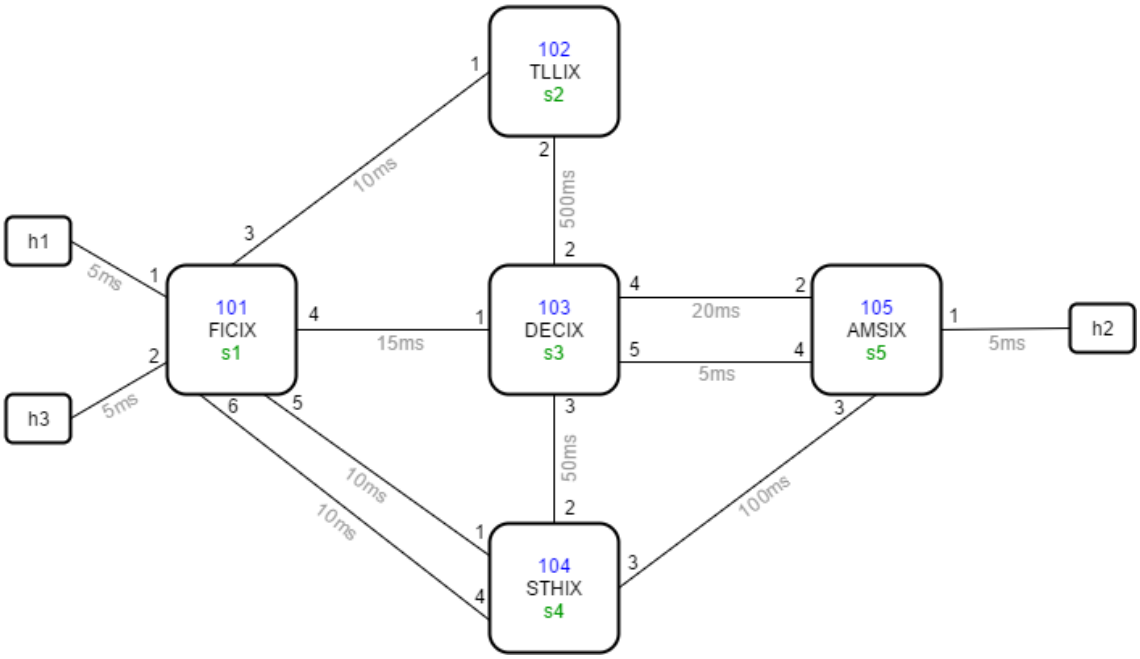

Figure 10. Project topology for segment routing demonstration.

The routers shown in Figure 10 have also been assigned with SR labels between 101 and 105, starting with s1 as 101 and so on. In the following test scenarios host h1 and h3 will ping host h2.

In order to implement the custom topology, a configuration file for the ONOS controller, available in appendix 2, had to be created and also two script files, available in appendix 3 and 4, were required to setup the topology in Mininet. There will be more details on the configuration and script files in the following sections.

5.1.1 ONOS Controller Setup

The ONOS controller acting as the PCE for the SR-domain needs to be started with the correct configuration file: *sr-thesis-controller.conf* (appendix 2). In order to do this, the configuration file should be saved in the ~/spring-open/conf/ directory. The configuration file basically instructs the controller where in the topology the nodes and links reside, and which MAC addresses and ports are assigned. The configuration file also labels the nodes and links with SR labels. Next, the ONOS controller startup configuration script *onos.properties* needs to be modified to start the controller with the proper configuration file. The *onos.properties* file is located in the same directory as the

previously saved file. The last line in *onos.properties* should be modified with a text editor to point to the previously saved configuration file as shown in Listing 1.

```
# Specify a network configuration file to be used by the
NetworkConfigManager
net.onrc.onos.core.configmanager.NetworkConfigManager.netwo
rkConfigFile = conf/sr-thesis-controller.conf
```

Listing 1. A caption of onos.properties file content.

Listing 1 shows the line that needs to be edited. After this, the controller can be started with the following command: **~/spring-open/onos.sh start**. Once the startup script has run its course, the status of the controller can be checked with the following command: **~/spring-open/onos.sh status.** The output should look similar to what is shown in Figure 11.

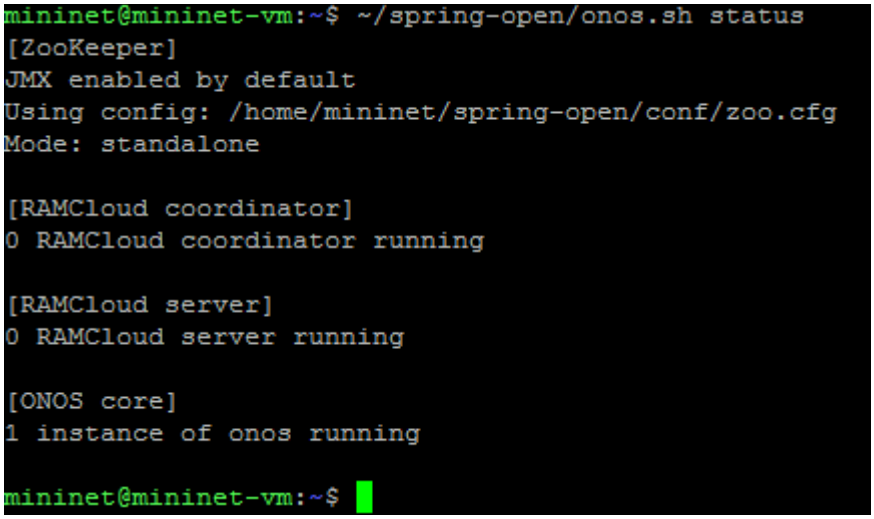

Figure 11. Status output of an ONOS controller running without errors. Screenshot [2].

The status command output should report one instance of the ONOS controller running as shown in Figure 11.

5.1.2 Mininet Setup

After starting the ONOS controller, it is time to setup the virtualized network in Mininet. First, the two configuration scripts *sr-mn-script.py* (appendix 3) and *sr_thesis_topo.py* (appendix 4) should be saved in the ~/mininet/custom/ directory. Both scripts guide

Mininet to create the network as outlined in Figure 10. These two script files need to be executable. Therefore the user rights for the files should be modified accordingly, as shown in Figure 12, to allow execution of the script files.

```
mininet@mininet-vm:~$ sudo chmod 751 ~/mininet/custom/sr-mn-script.py
mininet@mininet-vm:~$ sudo chmod 751 ~/mininet/custom/sr_thesis_topo.py
```

Figure 12. Script file's user rights modification to allow execution. Screenshot [3].

Figure 12 shows the two commands required to modify the user rights accordingly. It should be noted that the commands will not generate any output, if entered correctly.

The virtualized testing network should be started next by executing the following command: **sudo ~/mininet/custom/sr-mn-script.py**. The resulting output after command execution should be the same as shown in Figure 13.

```
mininet@mininet-vm:~$ sudo ~/mininet/custom/sr-mn-script.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(5ms delay) (5ms delay) (h1, s1) (5ms delay) (5ms delay) (h2, s5) (5ms delay) (5
ms delay) (h3, s1) (10ms delay) (10ms delay) (s1, s2) (15ms delay) (15ms delay)
(s1, s3) (10ms delay) (10ms delay) (s1, s4) (500ms delay) (500ms delay) (s2, s3)
 (50ms delay) (50ms delay) (s3, s4) (20ms delay) (20ms delay) (s3, s5) (100ms de
lay) (100ms delay) (s4, s5)
*** Configuring hosts
h1 h2 h3
(10ms delay) (10ms delay) (5ms delay) (5ms delay) *** Starting controller
*** Starting 5 switches
s1 s2 s3 s4 s5
*** Starting CLI:
mininet>
```

Figure 13. Mininet output after successful script execution. Screenshot [4].

The output of the command entered in Figure 13 should show addition and configuration of three hosts and five switches.

After these steps, the network and controller are ready for testing. An application called *spring-open-cli*, which is a Command Line Interface (CLI) tool, will be used in the next section to verify network functionality. The tool comes preinstalled in the VM image.

5.2 Verify System Functionality

The ONOS controller can be controlled and monitored with the *spring-open-cli* tool. It is more convenient to start the tool in another terminal window for better usability later on. The tool can now be started by executing the commands shown in Figure 14 in the ~/spring-open-cli/ directory.

```
mininet@mininet-vm:~/spring-open-cli$ source ./workspace/ve/bin/activate
(ve)mininet@mininet-vm:~/spring-open-cli$ sudo make start-sdncon
if ! lsof -iTCP:8000 -sTCP:LISTEN >/dev/null; then \
        ( \
          cd /home/mininet/spring-open-cli/sdncon; \
          [ -d /home/mininet/spring-open-cli/workspace/ve/cassandra/data/sdncon
] || python manage.py syncdb --noinput; \
          /home/mininet/spring-open-cli/build/start-and-wait-for-port.sh -p 8000
 -l /home/mininet/spring-open-cli/workspace/ve/log/sdncon.log python manage.py r
unserver 0.0.0.0:8000; \
        ); \
        fi
+++ sdncon running
(ve)mininet@mininet-vm:~/spring-open-cli$ cd cli/
(ve)mininet@mininet-vm:~/spring-open-cli/cli$ ./cli.py
version200
default controller: 127.0.0.1:8000, SDN OS 1.0 - custom version
mininet-vm>
```

Figure 14. Commands to start the spring-open-cli tool. Screenshot [5].

As a result of executing the commands shown in Figure 14, the tool to manage the ONOS controller is now operational. To check that the controller sees all five routers, the following CLI command is required:

```
mininet-vm> show router
```

The command prints information from the controller about the connected routers as shown in Figure 15.

```
mininet-vm> show router
# Router DPID             Router Name Router IP       Router Mac        Edge Router Node SId
-|----------------------|-----------|---------------|-----------------|-----------|--------
1 00:00:00:00:00:00:00:01 FICIX       172.16.101.1/32 00:00:00:00:00:01 true        101
2 00:00:00:00:00:00:00:02 TLLIX       172.16.102.1/32 00:00:00:00:00:02 false       102
3 00:00:00:00:00:00:00:03 DECIX       172.16.103.1/32 00:00:00:00:00:03 false       103
4 00:00:00:00:00:00:00:04 STHIX       172.16.104.1/32 00:00:00:00:00:04 false       104
5 00:00:00:00:00:00:00:05 AMSIX       172.16.105.1/32 00:00:00:00:00:05 true        105
mininet-vm>
```

Figure 15. Output of **show router** CLI command. Screenshot [6].

The Edge Router column in Figure 15 shows if a router has directly connected subnets. All edge routers in the SR-domain use both MPLS and IP protocols, whereas the core routers only use MPLS.

Link information from the controller can be queried with the CLI command *show link.* The command execution can be seen in Figure 16.

```
mininet-vm> show link
# Src Switch DPID                Src Port Dst Switch DPID                Dst Port Type
-|------------------------------|--------|------------------------------|--------|------
1 00:00:00:00:00:00:00:01 (FICIX) 4        00:00:00:00:00:00:00:03 (DECIX) 1        packet
2 00:00:00:00:00:00:00:02 (TLLIX) 1        00:00:00:00:00:00:00:01 (FICIX) 3        packet
3 00:00:00:00:00:00:00:03 (DECIX) 2        00:00:00:00:00:00:00:02 (TLLIX) 2        packet
4 00:00:00:00:00:00:00:03 (DECIX) 3        00:00:00:00:00:00:00:04 (STHIX) 2        packet
5 00:00:00:00:00:00:00:03 (DECIX) 4        00:00:00:00:00:00:00:05 (AMSIX) 2        packet
6 00:00:00:00:00:00:00:04 (STHIX) 1        00:00:00:00:00:00:00:01 (FICIX) 5        packet
7 00:00:00:00:00:00:00:04 (STHIX) 3        00:00:00:00:00:00:00:05 (AMSIX) 3        packet
8 00:00:00:00:00:00:00:04 (STHIX) 4        00:00:00:00:00:00:00:01 (FICIX) 6        packet
9 00:00:00:00:00:00:00:05 (AMSIX) 4        00:00:00:00:00:00:00:03 (DECIX) 5        packet
mininet-vm>
```

Figure 16. Execution and output of **show link** CLI command. Screenshot [7].

The output of the command executed in Figure 16 shows MAC addressing and port numbering information of each link's end points.

5.3 Traffic Engineering with Segment Routing

By default, there are no SR path rules implemented, so packets between the hosts traverse the network using ECMP and the SPF algorithm. The standard SPF path selection behavior can be seen by conducting series of ping requests from host h1 to host h2 across the network. Executing ping between hosts h1 and h2 in Mininet is done with the following command:

```
mininet> h1 ping h2
```

Execution and output of the ping command is shown in Figure 17.

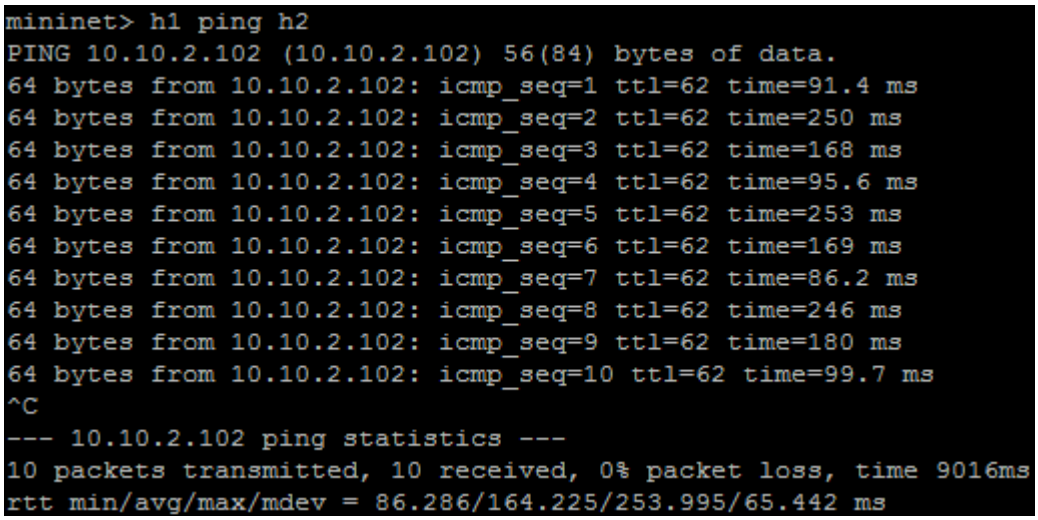

Figure 17. SPF ECMP routed ping times between hosts h1 and h2. Screenshot [8].

The ping time varies roughly between 80ms and 270ms as shown in Figure 17. This is expected behavior, because packets are routed to AMSIX either through DECIX or STHIX. Paths through DECIX and STHIX nodes represent an equal cost path from the point of view of FICIX towards AMSIX, as can be seen in Figure 10 topology. It should be noted that Mininet adds small variance and internal latency for the software routers to mimic real networks; thus the ping time is slightly higher compared to summing up the given link delay values that were configured earlier.

For the sake of demonstrating and testing SR features in the testing environment, I invented some path characterization for the network as shown in Figure 18.

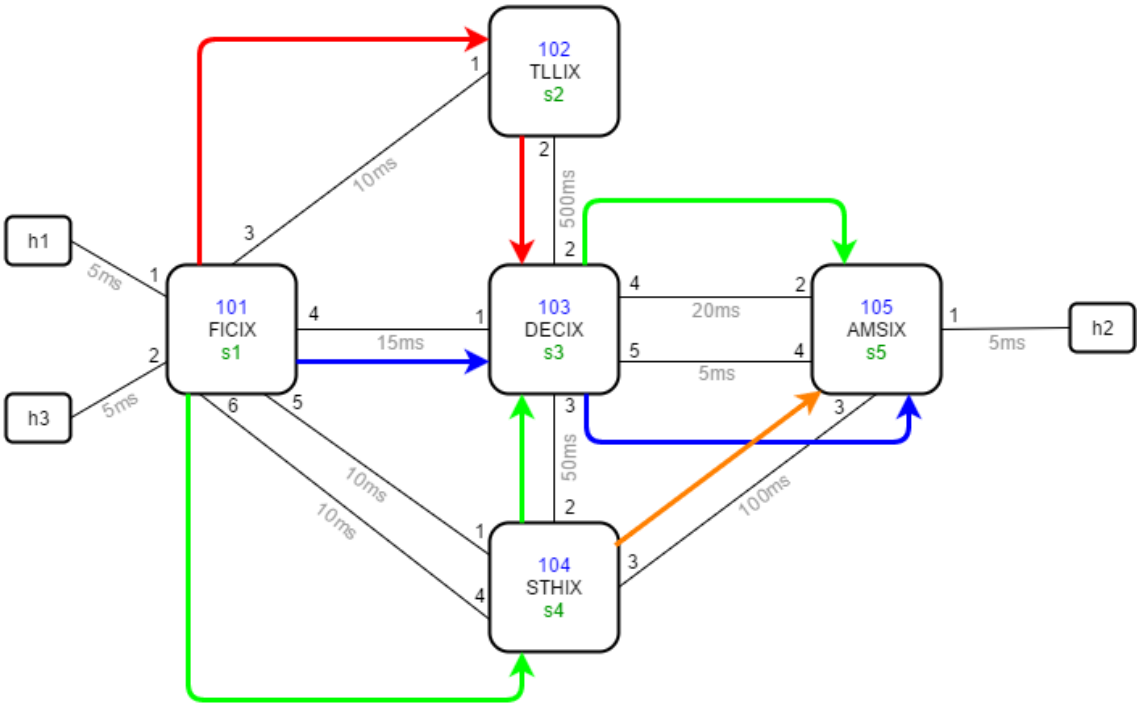

Figure 18. Test topology path characterization.

The following assumptions, to characterize the test topology shown in Figure 18, are made:

- The cheapest, high bandwidth path from FICIX to AMSIX goes through STHIX and DECIX, illustrated with green arrows in Figure 18.
- The low latency path from FICIX to AMSIX goes through DECIX and through the low latency link between DECIX and AMSIX, depicted with blue arrows in Figure 18.
- The path through TLLIX, illustrated with red arrows in Figure 18, is a high latency path and should be used only as a backup path in a network failure event.
- The orange arrow in Figure 18 illustrates a direct link between STHIX and AMSIX. This path is a high latency, low cost path and should be used only as a backup path in a network failure event.
- Host h1 has no special network demands between it and the destination host h2 (simulates file transfers).
- Host h3 requires a low latency path to h2 (simulates VOIP calls).

5.3.1 Implementing SR Rules

The *spring-open-cli* tool can be utilized to instruct the ONOS controller to redistribute SR rules to the SR-domain edge routers. First, the commands shown in Figure 19 should be executed.

```
mininet-vm> enable
mininet-vm# configure
mininet-vm(config)#
```

Figure 19. Entering configuration mode in the *spring-open-cli*. Screenshot [9].

The commands executed in Figure 19 will change the *spring-open-cli* tool into configuration mode.

In order to implement a segment routed path in the network, a **tunnel** containing SR labels of the desired path needs to be created. In addition, a **policy** rule needs to be created that will match the desired traffic.

Tunnel Creation

To create the green path in Figure 18 topology, a tunnel needs to be created with the *tunnel <name>* command. Then in the tunnel configuration mode, the SR labels of routers on the traversal path are entered with *node <SID>* command, starting with ingress node and ending with egress node. To exit a configuration menu, the **exit** command can be used. The created tunnel can be viewed with the **show tunnel** command. A new tunnel called GREEN for the green path can be created and then shown by using the commands shown in Figure 20.

```
mininet-vm(config)# tunnel GREEN
mininet-vm(config-tunnel)# node 101
mininet-vm(config-tunnel)# node 104
mininet-vm(config-tunnel)# node 103
mininet-vm(config-tunnel)# node 105
mininet-vm(config-tunnel)# exit
mininet-vm(config)# show tunnel
# Id    Policies Tunnel Path [Head-->Tail] Label Stack [Outer-->Inner]
-|-----|--------|------------------------|--------------------------
1 GREEN          [101, 104, 103, 105]      [[103, 105]]
mininet-vm(config)#
```

Figure 20. Commands to create and show a tunnel with the CLI. Screenshot [10].

The Tunnel Path column in Figure 20 shows the configured SIDs and the Label Stack column shows which SIDs will be included with the packets leaving the ingress node FICIX. Labels 101 and 104 are not required to be included with the packet, because the first node that requires to check the Label Stack for pathing instructions is STHIX (SID 104). The Policies column lists all policies that are active and point to a tunnel.

Policy Creation

Creating an SR-policy is very similar to creating Access Control Lists (ACLs) with traditional routers. To create a policy and enter the policy configuration menu, the following CLI command is used:

```
mininet-vm(config)# policy <name> policy-type tunnel-flow
```

The only fully implemented policy-type at the time of writing this thesis is tunnel-flow. Matching criteria for the policy is given with the *flow-entry* command as shown in Listing 2.

```
mininet-vm(config-policy)# flow-entry ip <src_ip>/<prefix>
<dst_ip>/<prefix>
```

Listing 2. *Flow-entry* command structure.

The *flow*-entry command illustrated in Listing 2 supports IP, TCP and UDP matching in source and destination parameters. Next, the tunnel where matched packets should be sent is specified with the *tunnel <name>* command. A new policy named LOW_COST needs be created for the green path traffic with the commands shown in Figure 21.

```
mininet-vm(config)# policy LOW_COST policy-type tunnel-flow
mininet-vm(config-policy)# flow-entry ip 10.10.1.0/24 10.10.2.102/32
mininet-vm(config-policy)# tunnel GREEN
mininet-vm(config-policy)# priority 1000
mininet-vm(config-policy)# exit
```

Figure 21. Policy creation and matching with a tunnel. Screenshot [11].

A priority value needs to be given for the policy as shown in Figure 21. The priority value determines which policy will be used, if multiple matches are found. The policy with a higher priority value is chosen in a multiple match event. The command to set priority is *priority* <*number*>. The priority value needs to be an integer value between 0 and 65535. The created policies can be viewed with the *show policy* command.

It should be noted that the previously created policy and tunnel only affect the traffic leaving FICIX, the return traffic still uses SPF ECMP calculation. The return traffic from host h2 to the h1 subnet needs to have a corresponding reversed path tunnel and policy set as well. Use the configuration commands shown in Figure 22 to create the reverse tunnel named GREEN_REVERSE and policy named LOW_COST_REVERSE.

```
mininet-vm(config)# tunnel GREEN_REVERSE
mininet-vm(config-tunnel)# node 105
mininet-vm(config-tunnel)# node 103
mininet-vm(config-tunnel)# node 104
mininet-vm(config-tunnel)# node 101
mininet-vm(config-tunnel)# exit
mininet-vm(config)# policy LOW_COST_REVERSE policy-type tunnel-flow
mininet-vm(config-policy)# flow-entry ip 10.10.2.102/32 10.10.1.0/24
mininet-vm(config-policy)# tunnel GREEN_REVERSE
mininet-vm(config-policy)# priority 1000
mininet-vm(config-policy)# exit
```

Figure 22. Reverse tunnel and policy for the green route. Screenshot [12].

As shown in Figure 22, the reverse tunnel uses the same commands as the forward tunnel did. The direction of the tunnel and policy is simply reversed and new name is given to the tunnel and policy.

To implement rules for the blue path in Figure 18, a tunnel with node labels 101, 103 and 105 needs to be created. In addition, there needs to be a local Adjacency-SID added for the low latency connection between nodes 103 and 105. The Adjacency-SIDs are automatically created in the test system by the ONOS controller. To find out the correct Adjacency-SID, the following CLI command should be used on node 101:

```
mininet-vm(config-policy)# show router <router MAC> port
```

The configuration of the blue path tunnel and a corresponding reverse path tunnel are shown in Figures 23 and 24 below. It should be noted that the Adjacency-SID is

extended to the associated Node-SID in the CLI command syntax. The commands shown in Figure 23 can be used to implement and show the blue path tunnel named BLUE.

```
mininet-vm(config)# tunnel BLUE
mininet-vm(config-tunnel)# node 101
mininet-vm(config-tunnel)# node 103 adjacency 103005
mininet-vm(config-tunnel)# node 105
mininet-vm(config-tunnel)# exit
mininet-vm(config)# show tunnel
# Id            Policies         Tunnel Path [Head-->Tail] Label Stack [Outer-->Inner]
-|-------------|----------------|-------------------------|---------------------------
1 BLUE                           [101, 103, 103005, 105]   [[103005]]
```

Figure 23. Commands to create and show blue tunnel. Screenshot [13].

The Label Stack is reduced to a single Adjacency-SID as shown in Figure 23. Only a single Adjacency-SID travels with the packet from the SR-domain edge node in this case. The reverse tunneled packets also carry a single label, the Node-SID 101. Use the commands in Figure 24 to implement and show the reversed blue path tunnel, named BLUE_REVERSE.

```
mininet-vm(config)# tunnel BLUE_REVERSE
mininet-vm(config-tunnel)# node 105 adjacency 105004
mininet-vm(config-tunnel)# node 103
mininet-vm(config-tunnel)# node 101
mininet-vm(config-tunnel)# exit
mininet-vm(config)# sh tun
# Id            Policies         Tunnel Path [Head-->Tail] Label Stack [Outer-->Inner]
-|-------------|----------------|-------------------------|---------------------------
1 BLUE                           [101, 103, 103005, 105]   [[103005]]
2 BLUE_REVERSE                   [105, 105004, 103, 101]   [[101]]
```

Figure 24. Commands to create and show the reversed blue tunnel. Screenshot [14].

Likewise, the reversed tunnel requires only a single Node-SID in the Label Stack, as shown in Figure 24.

The commands shown in Figure 25 should be used to configure policies named LOW_LATENCY and LOW_LATENCY_REVERSE for the blue path.

```
mininet-vm(config)# policy LOW_LATENCY policy-type tunnel-flow
mininet-vm(config-policy)# flow-entry ip 10.10.3.0/24 10.10.2.102/32
mininet-vm(config-policy)# tunnel BLUE
mininet-vm(config-policy)# priority 1000
mininet-vm(config-policy)# exit
mininet-vm(config)#
mininet-vm(config)# policy LOW_LATENCY_REVERSE policy-type tunnel-flow
mininet-vm(config-policy)# flow-entry ip 10.10.2.102/32 10.10.3.0/24
mininet-vm(config-policy)# tunnel BLUE_REVERSE
mininet-vm(config-policy)# priority 1000
mininet-vm(config-policy)# exit
```

Figure 25. Policy configurations for the blue path. Screenshot [15].

The commands in Figure 25 combine the created tunnels with policy rules and as a result, the traffic specified in the set policies will now use SR to traverse the network.

The required tunnel and policy configurations are now done for the green and blue paths depicted in Figure 18. The set policies can be viewed with the *show policy* command. Next the configuration will be tested and verified.

5.3.2 Testing the SR Implementation

A new SSH terminal connection window should be opened to the guest-VM running the SR virtual machine. The next task is to assign a free IP address to the *s1-eth1* adapter from the subnet where host h1 resides. Then an SSH connection should be made to host h1. The required commands are shown in Figure 26.

```
mininet@mininet-vm:~$ sudo ifconfig s1-eth1 10.10.1.250
mininet@mininet-vm:~$ ssh 10.10.1.101
```

Figure 26. Commands required to establish SSH connection to host h1. Screenshot [16].

After the execution of the commands shown in Figure 26, it should be noted that the command prompt will not change in any way. The **ifconfig** command can be used to check if the correct node has been entered in the network. The command will output the network interfaces of the connected node.

An infinite ping request should be launched towards host h2 from the host h1 SSH connection window as shown in Figure 27.

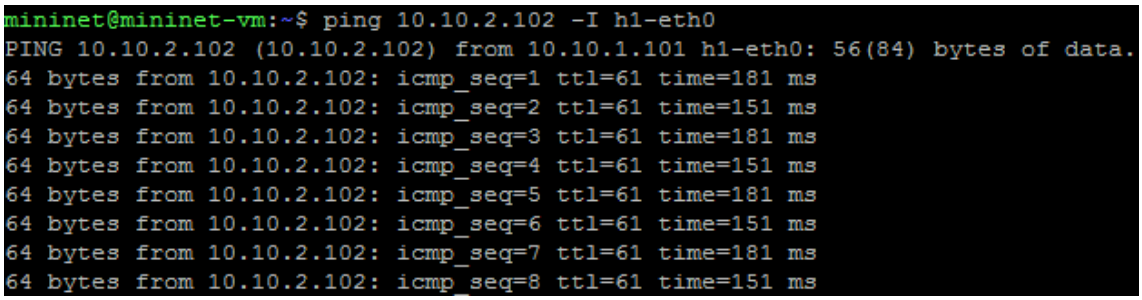

Figure 27. Ping requests launched from host h1 towards h2. Screenshot [17].

It should be now visible that the ping time variation is significantly lower than before, as shown in Figure 27. In fact, the variation should be close to 30m, which is caused by the load balancing behavior over the two links between nodes 103 and 105.

Inspecting MPLS Packet Headers

The outer MPLS headers of the SR routed packets should be inspected in order to understand how the ICMP traffic, from host h1 to h2, traverses the network according to set SR-policy *LOW_COST* and its associated tunnel *GREEN*. To do this, the ping requests should be left running in the host h1 SSH session window.

Next, the spring-open-cli SSH session window should be made active, which was opened earlier. First, a packet should be captured from the node 101 interface, which is connected to host h1. The correct interface to capture can be found by executing the following command: **show router <node MAC> port**. This command needs to be executed in the spring-open-cli SSH session window as shown in Figure 28.

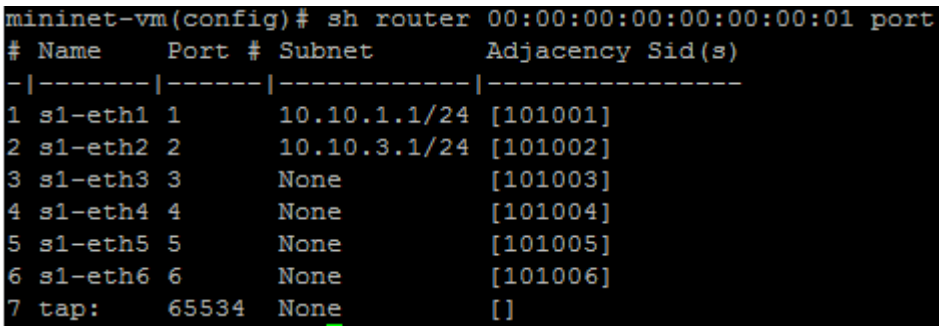

Figure 28. Port and interface configuration of node 101. Screenshot [18].

By using the command shown in Figure 28, it should be possible to determine the correct interface for packet capturing. The output of the command should be compared with Figure 18, where links have been labeled with associated port numbers.

To capture a packet, a packet capturing command line tool is used, included in the testing environment, called *tcpdump*. The following command show in Listing 3 needs to be executed in the Mininet CLI SSH session window to capture an ICMP packet from host h1.

```
mininet> s1 tcpdump -vi s1-eth1 -c1 ‘icmp and src host
10.10.1.101’
```

Listing 3. *Tcpdump* command structure for packet capture.

The command syntax illustrated in Listing 3 is constructed in such manner that *tcpdump* only captures ICMP packets from host h1 on the node 101’s interface directly connected with host h1. A captured packet is shown in Figure 29.

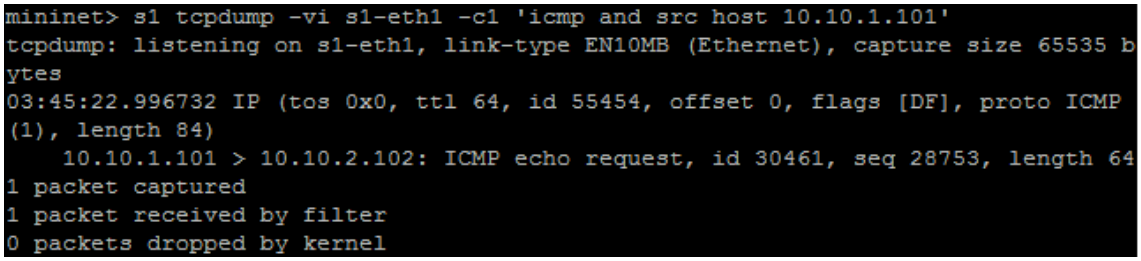

Figure 29. A packet captured from node 101 port connected to host h1. Screenshot [19].

The command output in Figure 29 shows that one ICMP packet was captured. According to the output, the ICMP packet was sent from host h1 IP address and it was destined to the host h2 IP address as expected.

Next, an MPLS encapsulated packet leaving node 101 from one of its interfaces connected to node 104 is expected to be captured. In order to capture and examine such a packet, the MPLS packet capturing command shown in Figure 30 is required.

```
mininet> s1 tcpdump -vi s1-eth6 -c1 mpls
tcpdump: WARNING: s1-eth6: no IPv4 address assigned
tcpdump: listening on s1-eth6, link-type EN10MB (Ethernet), capture size 65535 b
ytes
03:46:43.097836 MPLS (label 103, exp 0, ttl 63)
        (label 105, exp 0, [S], ttl 63)
        IP (tos 0x0, ttl 64, id 55534, offset 0, flags [DF], proto ICMP (1), len
gth 84)
    10.10.1.101 > 10.10.2.102: ICMP echo request, id 30461, seq 28833, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 30. A packet leaving node 101 towards node 104. Screenshot [20].

As can be seen in Figure 30, there are two MPLS labels present in the packet. The first node segment points towards node 103 and the second towards node 105 as expected.

According to the LOW_COST policy, the ICMP packets from host h1 to h2 should leave the node 104 on the interface that is connected to node 103. Node 104 is also expected to remove the first node segment 103 from the Label Stack. The next active SID is 105 in the Label Stack. The command shown in Figure 31 needs to be used in order to capture and inspect MPLS packets leaving node 104.

```
mininet> s4 tcpdump -vi s4-eth2 -c1 'mpls && src host 10.10.1.101'
tcpdump: WARNING: s4-eth2: no IPv4 address assigned
tcpdump: listening on s4-eth2, link-type EN10MB (Ethernet), capture size 65535 b
ytes
03:48:01.262435 MPLS (label 105, exp 0, [S], ttl 62)
        IP (tos 0x0, ttl 64, id 55612, offset 0, flags [DF], proto ICMP (1), len
gth 84)
    10.10.1.101 > 10.10.2.102: ICMP echo request, id 30461, seq 28911, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 31. A packet captured leaving the node 104. Screenshot [21].

The output of the command executed in Figure 31 shows that the packet was captured as expected and that there is now only one MPLS label 105 left.

Node 103 is the *penultimate hop* to SR-domain edge node 105. At this point, the MPLS encapsulation is removed from the packet and the packet is sent as normal ICMP packet to node 105. The command shown in Figure 32 can be used to inspect the packet leaving node 103 towards node 105.

```
mininet> s3 tcpdump -vi s3-eth4 -c1 'icmp and src host 10.10.1.101'
tcpdump: WARNING: s3-eth4: no IPv4 address assigned
tcpdump: listening on s3-eth4, link-type EN10MB (Ethernet), capture size 65535 b
ytes
04:06:01.070655 IP (tos 0x0, ttl 61, id 56690, offset 0, flags [DF], proto ICMP
(1), length 84)
    10.10.1.101 > 10.10.2.102: ICMP echo request, id 30461, seq 29989, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 32. A captured from node 103 port connected to node 105. Screenshot [22].

Figure 32 output shows that the packet is now indeed sent as an ICMP packet to its final destination as expected.

The ping reply from host h2 is routed back to host h1 according to the SR-policy *LOW_COST_REVERSE* and its associated tunnel *GREEN_REVERSE*, which was set earlier. To see how the node 105 has encapsulated the ping reply packet, the following command shown in Figure 33 needs to be used.

```
mininet> s5 tcpdump -vi s5-eth2 -c1 mpls
tcpdump: WARNING: s5-eth2: no IPv4 address assigned
tcpdump: listening on s5-eth2, link-type EN10MB (Ethernet), capture size 65535 b
ytes
04:18:20.314944 MPLS (label 104, exp 0, ttl 63)
        (label 101, exp 0, [S], ttl 63)
        IP (tos 0x0, ttl 64, id 6852, offset 0, flags [none], proto ICMP (1), le
ngth 84)
    10.10.2.102 > 10.10.1.101: ICMP echo reply, id 30461, seq 30727, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 33. An ICMP reply packet leaving node 105. Screenshot [23].

As shown in Figure 33 output, the MPLS Label Stack has the labels 104 and 101 present and the encapsulated ICMP reply packets will traverse the network back to host h1 according to the set policy.

Fine Grained Traffic Control with SR

Finer grained traffic control can be achieved by implementing the local specific *Adjacency Segments*. In the example scenario outlined in Figure 18, the low latency dependent traffic between hosts h3 and h2 is guided by the blue path's policies and tunnels set in the previous section.

A new terminal connection should be established to the guest-VM running the SR virtual machine. The next task is to assign a free IP address to the *s1-eth2* adapter from the subnet where host h3 resides. In addition, the guest-VM needs to be instructed which adapter is connected to the subnet. Then an SSH connection should be opened to the host h3. The required commands are shown in Figure 34 below.

```
mininet@mininet-vm:~$ sudo ifconfig s1-eth2 10.10.3.150
mininet@mininet-vm:~$ sudo ip route add 10.10.3.0/24 dev s1-eth2
mininet@mininet-vm:~$ ssh 10.10.3.103
```

Figure 34. Commands required to establish SSH connection to host h3. Screenshot [24].

As shown in Figure 34, the host system of Mininet needs to be instructed with an additional *ip route* command that will bind the subnet with corresponding interface. This additional step was not required before in Figure 26, because Mininet adds route to the first connected subnet automatically.

Once connected to host h3, an infinite ping request needs to be launched towards host h2. The command to do so is shown in Figure 35.

```
mininet@mininet-vm:~$ ping 10.10.2.102 -I h3-eth0
PING 10.10.2.102 (10.10.2.102) from 10.10.3.103 h3-eth0: 56(84) bytes of data.
64 bytes from 10.10.2.102: icmp_seq=1 ttl=62 time=65.6 ms
64 bytes from 10.10.2.102: icmp_seq=2 ttl=62 time=88.1 ms
64 bytes from 10.10.2.102: icmp_seq=3 ttl=62 time=84.1 ms
64 bytes from 10.10.2.102: icmp_seq=4 ttl=62 time=86.6 ms
64 bytes from 10.10.2.102: icmp_seq=5 ttl=62 time=69.2 ms
64 bytes from 10.10.2.102: icmp_seq=6 ttl=62 time=69.2 ms
64 bytes from 10.10.2.102: icmp_seq=7 ttl=62 time=68.0 ms
64 bytes from 10.10.2.102: icmp_seq=8 ttl=62 time=71.6 ms
```

Figure 35. Ping requests launched from host h3 towards h2. Screenshot [25].

Figure 35 shows that the ping time is now considerably less than what it was with the green path that was examined earlier. There is still some variance in the ping times, which is caused by the virtualized testing environment.

Inspecting Adjacency Segment Operation

The LOW_LATENCY policy and its associated BLUE tunnel were implemented to route packets from host h3 to h2. The BLUE tunnel Label Stack was reduced to a single Adjacency-SID 103005. The ICMP packets from host h3 to h2 should leave the SR-

domain edge node 101 towards node 103. The commands shown in Figure 36 need to be run in the Mininet CLI session window to see how the node 101 has encapsulated the ICMP packet from host h3.

```
mininet> s1 tcpdump -vi s1-eth4 -c1 'mpls and src host 10.10.3.103'
tcpdump: WARNING: s1-eth4: no IPv4 address assigned
tcpdump: listening on s1-eth4, link-type EN10MB (Ethernet), capture size 65535 bytes
02:10:41.071701 MPLS (label 103005, exp 0, [S], ttl 63)
        IP (tos 0x0, ttl 64, id 22867, offset 0, flags [DF], proto ICMP (1), length 84)
    10.10.3.103 > 10.10.2.102: ICMP echo request, id 3662, seq 1370, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 36. A packet leaving node 101 towards node 103. Screenshot [26].

As shown in Figure 36, node 101 sends an MPLS routed packet towards node 103 with a single Adjacency-SID 103005 in the Label Stack as expected. Once again, the node 103 is the *penultimate hop* to the SR-domain edge node 105. At this point, the MPLS encapsulation is removed from the packet and the packet is sent as normal ICMP packet to node 105, using the lower latency link between the nodes. The command shown in Figure 37 can be used to inspect the packet leaving node 103 towards node 105 using the lower latency link.

```
mininet> s3 tcpdump -vi s3-eth5 -c1 'icmp and src 10.10.3.103'
tcpdump: WARNING: s3-eth5: no IPv4 address assigned
tcpdump: listening on s3-eth5, link-type EN10MB (Ethernet), capture size 65535 bytes
02:27:01.235786 IP (tos 0x0, ttl 62, id 23845, offset 0, flags [DF], proto ICMP (1), length 84)
    10.10.3.103 > 10.10.2.102: ICMP echo request, id 3662, seq 2348, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 37. A captured from node 103 low latency link with node 105. Screenshot [27].

As can be seen in Figure 37, the MPLS encapsulation has been removed and the packet is sent via the low latency link as an ICMP packet.

The ping reply packet from host h2 to h3 is encapsulated into an MPLS packet that has again only a single Node-SID 101 in the Label Stack. The process of reducing the Label Stack into a single label was explained in section 5.3.1. The MPLS encapsulated packet leaving node 105 can be examined with the commands show in Figure 38.

```
mininet> s5 tcpdump -vi s5-eth4 -c1 'mpls and dst 10.10.3.103'
tcpdump: WARNING: s5-eth4: no IPv4 address assigned
tcpdump: listening on s5-eth4, link-type EN10MB (Ethernet), capture size 65535 bytes
04:06:23.856069 MPLS (label 101, exp 0, [S], ttl 63)
        IP (tos 0x0, ttl 64, id 45770, offset 0, flags [none], proto ICMP (1), length 84)
    10.10.2.102 > 10.10.3.103: ICMP echo reply, id 3662, seq 8297, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 38. A ping reply packet from node 105 to node 103. Screenshot [28].

The Figure 38 output shows that there is a single Node-SID 101 in the Label Stack. The packet was captured on the s5-eth4 interface, which is connected to the low latency link between the nodes 103 and 105. The ping reply packets from node 105 are sent using the low latency link as expected.

Next, node 103 strips the MPLS encapsulation from the packet and sends it out through s3-eth1 interface to node 101 as an ICMP packet as shown in Figure 39.

```
mininet> s3 tcpdump -vi s3-eth1 -c1 'icmp and dst 10.10.3.103'
tcpdump: WARNING: s3-eth1: no IPv4 address assigned
tcpdump: listening on s3-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
04:15:23.018727 IP (tos 0x0, ttl 62, id 46308, offset 0, flags [none], proto ICMP (1), length 84)
    10.10.2.102 > 10.10.3.103: ICMP echo reply, id 3662, seq 8835, length 64
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Figure 39. A packet captured leaving node 103 towards node 101. Screenshot [29].

The packet capture in Figure 39 shows that the encapsulation has been removed and the packets are sent to host h3 using the ICMP protocol as expected.

It has now been demonstrated and tested that the traffic is correctly routed in the test network using the configured SR tunnels and policies. This chapter concludes the practical tests of the project.

6 Discussion

The test results show that the project was successful in demonstrating traffic engineering with segment routing. The testing system was built in a virtualized testing environment, using a centralized ONOS software controller and SR capable software switches run in Mininet. It was relatively easy to implement SR tunnels and policies with the inbuilt CLI tool, due to similarities with Cisco CLI commands for setting up tunnels and ACLs.

The chosen ONF's SPRING-OPEN project virtual demonstration environment worked well for basic testing of the SR features. However, the test system was struggling, when network failure events were introduced to the test network by shutting down nodes or links. The system slowed down significantly and traffic rerouting did not seem to follow any foreseeable logic, such as SPF routing or configured SR policies and tunnels. In a network failure event the traffic was sent through suboptimal paths, seemingly at random.

7 Conclusion

The aim of this project, to study segment routing and test its traffic engineering capabilities in MPLS networks by building a virtualized testing environment, was met. SR architecture combines the source routing paradigm, existing routing protocols, and SDN features in a unique way that is at the same time cost effective and relatively simple to implement.

Even though SR is undergoing a standardization process, it seems to be a prominent technology which can answer the rapidly growing scalability demands in large networks, without compromising security. It will be interesting to see if the IPv6 implementation of SR could turn out to be the most popular implementation. SR is even simpler to implement with the IPv6 extension compared to the MPLS extension as it requires no additional labels to operate.

The selected virtualized demonstration environment was a little challenging to use at first. The detailed instructions in the study should provide an excellent starting point for more advanced SR tests in the future. The creation of policies and tunnels is very simple due to close resemblance with the Cisco OS command structure and syntax.

However, it should be noted that more advanced TE tests with the demonstration environment will likely require deep understanding of the Python language. The environment had severe issues when artificial network failures were introduced into the network and will require fixes and improvements to the ONOS controller code base.

References

1 Open Networking Summit. Why SDN? [online]. URL: http://opennetsummit.org/archives/mar14/site/why-sdn.html. Accessed 30 May 2016.

2 University of Virginia. 14 Years of web statistics at U.Va! [online]. July 2009. URL: http://www.virginia.edu/virginia/archive/webstats.html. Accessed 30 May 2016.

3 International Telecommunication Union. ITU releases 2015 ICT figures [online]. May 2015. URL: http://www.itu.int/net/pressoffice/press_releases/2015/17.aspx#.V0ww85N95E4. Accessed 30 May 2016.

4 Juniper research. 'Internet of Things' connected devices to almost triple to over 38 billion units by 2020 [online]. URL: http://www.juniperresearch.com/press/press-releases/iot-connected-devices-to-triple-to-38-bn-by-2020. Accessed 30 May 2016.

5 Juniper Networks. Understanding IP source route options [online]. URL: http://www.juniper.net/documentation/en_US/junos12.1x44/topics/concept/reconnaissance-deterrence-attack-evasion-ip-source-route-understanding.html. Accessed 1 June 2016.

6 Filsfils C, Nainar NK, Pignataro C. The segment routing architecture [online]. 2015. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7417124&tag=1. Accessed 6 June 2016.

7 Göransson P, Black C. Software defined networks: A comprehensive approach. Waltham, MA, USA: Elsevier; 2014.

8 Bookham C, Jansen A, Raj A. Benefits of segment routing and path computation [online]. September 2015. URL: https://techzine.alcatel-lucent.com/benefits-segment-routing-and-path-computation. Accessed 1 June 2016.

9 Open Networking Foundation. Software-defined networking (SDN) definition [online]. URL: https://www.opennetworking.org/sdn-resources/sdn-definition. Accessed 6 June 2016.

10 Lee M-C, Sheu J-P. An efficient routing algorithm based on segment routing in software-defined networking [online]. July 2016. URL: http://www.sciencedirect.com/science/article/pii/S1389128616300871. Accessed 6 June 2016.

11 Cisco. Segment routing: Prepare your network for new business models white paper [online]. July 2015. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/application-engineered-routing/white-paper-c11-734250.html. Accessed 1 June 2016.

12 Nehib G, Duvivier B. Transforming evolved programmable networks [online]. May 2015. URL: https://www.ciscoknowledgenetwork.com/files/518_05-05-15-AER_CKN_v6.pdf. Accessed 2 June 2016.

13 IETF. WG Action: Formed source packet routing in networking (spring) [online]. October 2013. URL: https://www.ietf.org/mail-archive/web/ietf-announce/current/msg12052.html. Accessed 2 June 2016.

14 Filsfils C, Previdi S, Bashandy A, Decraene B, Litkowski S, Horneffer M, Shakir R, Tantsura J, Crabbe E. Segment routing architecture draft-ietf-spring-segment-routing-00 [online]. November 2014. URL: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-00. Accessed 2 June 2016.

15 Filsfils C. Segment routing: Deployment experience and technology update [online]. March 2016. URL: https://www.youtube.com/watch?v=VORGW0bBHLs. Accessed 8 June 2016.

16 Filsfils C, Previdi S, Bashandy A, Decraene B, Litkowski S, Shakir R. Segment routing architecture draft-ietf-spring-segment-routing-08 [online]. May 2016. URL: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-08. Accessed 8 June 2016.

17 Previdi S, Filsfils C, Bashandy A, Gredler H, Litkowski S, Decraene B, Tantsura J. IS-IS extensions for segment routing draft-ietf-isis-segment-routing-extensions-06 [online]. December 2015. URL: https://tools.ietf.org/html/draft-ietf-isis-segment-routing-extensions-06. Accessed 8 June 2016.

18 Psenak P, Previdi S, Filsfils C, Gredler H, Shakir R, Henderickx W, Tantsura J. OSPF extensions for segment routing draft-ietf-ospf-segment-routing-extensions-08 [online]. April 2016. URL: https://tools.ietf.org/html/draft-ietf-ospf-segment-routing-extensions-08. Accessed 8 June 2016.

19 Previdi S, Filsfils C, Ray S, Patel K, Dong J, Chen M. Segment routing BGP egress peer engineering BGP-LS extensions draft-ietf-idr-bgpls-segment-routing-epe-05 [online]. May 2016. URL: https://tools.ietf.org/html/draft-ietf-idr-bgpls-segment-routing-epe-05. Accessed 16 June 2016.

20 European Telecommunications Standards Institute. Network functions virtualisation [online]. October 2012. URL: https://portal.etsi.org/NFV/NFV_White_Paper.pdf. Accessed 21 June 2016.

21 Halpern J, Pignataro C. Service function chaining (SFC) architecture [online]. October 2015. URL: https://tools.ietf.org/html/rfc7665. Accessed 21 June 2016.

22 Luc De Ghein. 2007. MPLS fundamentals. Indianapolis, IN, USA: Cisco Press.

23 Filsfils C, Previdi S, Bashandy A, Decraene B, Litkowski S, Horneffer M, Shakir R, Tantsura J, Crabbe E. Segment routing with MPLS data plane draft-ietf-spring-segment-routing-mpls-04 [online]. March 2016. URL: https://tools.ietf.org/html/draft-ietf-spring-segment-routing-mpls-04. Accessed 15 June 2016.

24 Abley J, Afilias, Savola P, Neville-Neil G. Deprecation of type 0 routing headers in IPv6 [online]. December 2007. URL: https://tools.ietf.org/html/rfc5095. Accessed 15 June 2016.

25 Previdi S, Filsfils C, Field B, Leung I, Linkova J, Aries E, Kosugi T, Vyncke E, Lebrun D. IPv6 Segment routing header (SRH) draft-ietf-6man-segment-routing-header-01 [online]. March 2016. URL: https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-01. Accessed 15 June 2016.

26 Deering S, Hinden R. Internet protocol, version 6 (IPv6) specification [online]. December 1998. URL: https://tools.ietf.org/html/rfc2460. Accessed 16 June 2016.

27 Das S, Shin S, Vavilapalli S, Khan FN, Rajagopalan B, Virk J, Ganapathiraman N, Singal P, Ghasemian S, Park C. ONF's SPRING-OPEN project [online]. December 2014. URL: https://wiki.onosproject.org/display/ONOS/Project+Description. Accessed 25 July 2016.

28 Das S, Shin S, Vavilapalli S, Khan F, Rajagopalan B, Virk J, Ganapathiraman N, Singal P, Ghasemian S, Park C. ONF SPRING-OPEN project's pre-built VM image [online]. December 2014. URL: http://downloads.onosproject.org/spring-open/SPRING-OPEN.ova. Accessed 26 July 2016.

ONOS Controller Configuration File

configuration *sr-thesis-controller.conf*

```
{
  "comment": " SR thesis 5 router and 3 hosts description and configuration",
  "restrictSwitches": true,
  "restrictLinks": true,

  "switchConfig":
          [
            { "nodeDpid": "00:01", "name": "FICIX", "type": "Router_SR", "allowed": true,
              "latitude": 80.80, "longitude": 90.10,
              "params": { "routerIp": "172.16.101.1/32",
                        "routerMac": "00:00:00:00:00:01",
                        "nodeSid": 101,
                        "isEdgeRouter" : true,
                        "subnets": [
                                  { "portNo": 1, "subnetIp": "10.10.1.1/24" },
                                  { "portNo": 2, "subnetIp": "10.10.3.1/24" }
                                  ]
                        }
              },

            { "nodeDpid": "00:02", "name": "TLLIX", "type": "Router_SR", "allowed": true,
              "latitude": 80.80, "longitude": 90.10,
              "params": { "routerIp": "172.16.102.1/32",
                        "routerMac": "00:00:00:00:00:02",
                        "nodeSid": 102,
                        "isEdgeRouter" : false

                        }
              },

            { "nodeDpid": "00:03", "name": "DECIX", "type": "Router_SR", "allowed": true,
              "latitude": 80.80, "longitude": 90.10,
              "params": { "routerIp": "172.16.103.1/32",
                        "routerMac": "00:00:00:00:00:03",
                        "nodeSid": 103,
                        "isEdgeRouter" : false
                                  }
              },

                                  { "nodeDpid": "00:04", "name": "STHIX", "type":
"Router_SR", "allowed": true,
              "latitude": 80.80, "longitude": 90.10,
              "params": { "routerIp": "172.16.104.1/32",
                        "routerMac": "00:00:00:00:00:04",
                        "nodeSid": 104,
                        "isEdgeRouter" : false
```

```
                }
            },

                        { "nodeDpid": "00:05", "name": "AMSIX", "type":
"Router_SR", "allowed": true,
            "latitude": 80.80, "longitude": 90.10,
            "params": { "routerIp": "172.16.105.1/32",
                    "routerMac": "00:00:00:00:00:05",
                    "nodeSid": 105,
                    "isEdgeRouter" : true,
                    "subnets": [
                            { "portNo": 1, "subnetIp": "10.10.2.1/24" }
                            ]
                }
            }
        ],

 "linkConfig":
        [
           { "type": "pktLink", "allowed": true,
             "nodeDpid1": "01", "nodeDpid2": "02",
             "params": { "port1": 3, "port2": 1 }
             },

           { "type": "pktLink", "allowed": true,
             "nodeDpid1": "01", "nodeDpid2": "03",
             "params": { "port1": 4, "port2": 1 }
             },

           { "type": "pktLink", "allowed": true,
             "nodeDpid1": "01", "nodeDpid2": "04",
             "params": { "port1": 5, "port2": 1}
             },

           { "type": "pktLink", "allowed": true,
             "nodeDpid1": "01", "nodeDpid2": "04",
             "params": { "port1": 6, "port2": 4 }
             },

                        { "type": "pktLink", "allowed": true,
             "nodeDpid1": "02", "nodeDpid2": "03",
             "params": { "port1": 2, "port2": 2 }
             },

                        { "type": "pktLink", "allowed": true,
             "nodeDpid1": "03", "nodeDpid2": "04",
             "params": { "port1": 3, "port2": 2 }
             },

                        { "type": "pktLink", "allowed": true,
             "nodeDpid1": "03", "nodeDpid2": "05",
             "params": { "port1": 4, "port2": 2 }
             },
```

{ "type": "pktLink", "allowed": true,
"nodeDpid1": "03", "nodeDpid2": "05",
"params": { "port1": 5, "port2": 4 }
},

{ "type": "pktLink", "allowed": true,
"nodeDpid1": "04", "nodeDpid2": "05",
"params": { "port1": 3, "port2": 3 }
}
]
}

Mininet Network Configuration Script

Script *sr-mn-script.py*

```
#!/usr/bin/env python
'''
A script to connect 5 router & 3 host topo

'''

from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.node import UserSwitch, RemoteController, OVSSwitch
from mininet.topolib import TreeNet
from mininet.topo import SingleSwitchTopo
from mininet.net import Mininet
from mininet.link import TCLink
from functools import partial
from sr_thesis_topo import SRTopo
from alterableNet import alterableCLI
from alterableNet import alterNet

def setDefaultRoute(node, ip, intf=None):
    """

    Modified node.setDefaultRoute that sets a default gateway IP.

    """
    if not intf:
        intf = node.defaultIntf()
    node.cmd('route add -net 0.0.0.0 gw %s %s' % (ip, intf))

if __name__ == '__main__':
    setLogLevel( 'info' )
    topo = SRTopo()
    # load the topology specified in sr_thesis_topo.py, use the cpqd1.3 switch and point
to a controller running
    # in this VM
    # link parameter enables addition of custom link delays
    net                 =                 alterNet(topo=topo,                 switch=UserSwitch,
controller=partial(RemoteController,ip='127.0.0.1'), link=TCLink)

    # adding extra links between routers and delays
    s1, s2, s3, s4, s5 = net.switches
    net.addLink( s1, s4, delay='10ms' )
    net.addLink( s3, s5, delay='5ms' )
    net.start()

    # end-host configuration
    h1, h2, h3 = net.hosts
```

```
# host h1 is attached to router s1 / FICIX
h1.setIP("10.10.1.101/24")
h1.setMAC("00:00:00:00:01:02")
setDefaultRoute(h1, "10.10.1.1")

# host h2 is attached to router s5 / AMSIX
h2.setIP("10.10.2.102/24")
h2.setMAC("00:00:00:00:02:24")
setDefaultRoute(h2, "10.10.2.1")

# host h3 is attached to router s1 / FICIX
h3.setIP("10.10.3.103/24")
h3.setMAC("00:00:00:00:03:12")
setDefaultRoute(h3, "10.10.3.1")

alterableCLI(net)
net.stop()
```

Mininet Custom Topology Script

Script *sr_thesis_topo.py*

```
#!/usr/bin/env python
"""Custom topology

5 routers in 5 cities, 3 hosts.

"""

from mininet.topo import Topo

class SRTopo( Topo ):
    "Simple topology."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        host1 = self.addHost( 'h1' )
        host2 = self.addHost( 'h2' )
        host3 = self.addHost( 'h3' )

        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')
        s5 = self.addSwitch('s5')

        # Add links for hosts
        self.addLink( host1, s1, delay='5ms' )
        self.addLink( host2, s5, delay='5ms' )
        self.addLink( host3, s1, delay='5ms' )

        # Add links between switches
        # Only a single link will be added here between any pair of switches
        # Extra links will be added by the script that imports this topo file
        # See sr_mn_script.py
        self.addLink( s1, s2, delay='10ms' )
        self.addLink( s1, s3, delay='15ms' )
        self.addLink( s1, s4, delay='10ms' )
        self.addLink( s2, s3, delay='500ms' )
        self.addLink( s3, s4, delay='50ms' )
        self.addLink( s3, s5, delay='20ms' )
        self.addLink( s4, s5, delay='100ms' )

topos = { 'mytopo': ( lambda: SRTopo() ) }
```