Traffic Engineering with Segment Routing in ONOS Controller

May Thu Zar Win, Khin Than Mya, Yutaka Ishibashi
University of Computer Studies, Yangon.
maythuzarwin@ucsy.edu.mm, khinthanmya@ucsy.edu.mm, ishibasi@nitech.ac.jp

Abstract

Traditional per-flow routing increases communication complexity of both control and data planes because it requires a direct interaction between each node that is included in traffic paths and controller. Segment Routing (SR) only maintains per-flow states at the ingress node and no needs to install flow rules at the intermediate devices. Therefore, SR has been popular as an alternative traffic engineering (TE) solution in the software defined network. This paper implements SR-TE in ONOS controller and also demonstrates fine gained TE by using SR's tunnels and policies in the software defined network. Moreover, we discuss SR without policies and tunnels and SR-TE with delay variation of ICMP packet.

Keywords *- Traffic Engineering, ONOS, Segment routing, Software defined networking.*

1. Introduction

The concept of Software Defined Networking (SDN) is to organize and manage networks with software. SDN decouples control and data plane from networks through control protocols such as OpenFlow [1]. Moreover, SDN serves vendor neutrality with the abstraction of network devices such as routers and switches, etc. Network administrator can easily manage the large scale network with the software when SDN is employed into a large-scale, carrier-grade network. To manage networks with software, many SDN controllers have been introduced such as POX, NOX, RYU, FloodLight, OpenDayLight and ONOS controller. Most of them are open-source projects. Because SDN employs a centralized management scheme, it has a bottleneck problem in the control plane of large scale networks [2]. ONOS has a distributed nature and can resolve the control plane bottleneck problem.

In SDN scenario, when the first packet of a flow enters to the switch, it encapsulates and forwards the packet to the SDN controller that makes a decision whether the flow should be added to the switch flow table or not. Then, based on the flow table the switch forwards incoming packets to the appropriate port. The direct interaction between the controller and switches may increase and the switch's Ternary Content Addressable Memory (TCAM) may lead to storage consumption. SR doesn't need to install flow rules at the intermediate devices because SR only maintains per-flow states at the ingress node. Therefore, SR can reduce TCAM consumption and also decrease controller work load. In this paper, we implement and demonstrate SR-TE (Segment Routing with Traffic Engineering) by using ONOS controller. Moreover, we implement tunnels and policies to demonstrate fine gained TE and discuss ordinary SR and SR-TE with delay variation of ICMP packet.

The remainder of this paper is structured as follows. Section 2 describes the Traffic engineering with segment routing as well as SR architecture and ONOS controller. The experiment results are followed by section 3. Finally, section 4 gives conclusion.

2. Traffic Engineering with Segment Routing

Traffic engineering is an important network application which studies measurement and management of network traffic [3]. TE designed routing mechanism that guide network traffic to improve utilization of network resources, and better meets the requirements of network quality of service (QoS). TE permits the provider to ignore the shortest-path rule by sending traffic over longer but less congested links [4]. This helps to alleviate network congestion and enables the network provider to maximize the use of the existing network infrastructure.

SR has been proposed one of the effective solutions of TE in SDN, and it allows the network operator to specify a path from ingress to egress using a forwarding path that is completely abstract from the Interior Gateway Protocol (IGP) shortest path. Network operators attain low-latency or

disjoint paths, regardless of the normal forwarding paths by utilizing SR. SR achieves this without any additional signaling protocol like RSVP-TE (Resource Reservation Protocol - Traffic Engineering). Moreover, SR can give control over TE paths without increasing the control plane overhead at the transit nodes.

SR can be directly applied to the Multi-protocol Label Switching (MPLS) architecture with no change in the forwarding plane. A segment is encoded as an MPLS label. An ordered list of segments is encoded as a stack of labels. The segment to process is on the top of the stack. The related label is popped from the stack, after the completion of a segment. SR can also be applied to the IPv6 architecture, with a new type of routing header. A segment is encoded as an IPv6 address. An ordered list of segments is encoded as an ordered list of IPv6 addresses in the routing header. The active segment is indicated by the destination address of the packet. The next active segment is indicated by a pointer in the new routing header. In this paper, we use MPLS label in the SR domain [5].

In the SR domain, nodes and links are assigned Segment Identifiers (SIDs), which are advertised into the domain by each SR router using extensions to Intermediate System-Intermediate System/Open Shortest Path First (IS-IS/OSPF) [6]. The main types of SID are:

- Node SID: A unique SID or global SID for each switch/router in the network.
- Adjacency SID: A local SID and it is used to forward the packet over the corresponding adjacency.
- Service SID: A service SID for application service such as firewall and vpn. It is used to forward the packets to the service SIDs.

These SIDs allow an ingress node to choose a path through the network using either a single SID that represents the destination node or using a series of SIDs, called a segment list, which identifies a particular path through the network that the SR tunnel should traverse [7]. There are some references papers that prove the SDN based SR-TE work well. Luca Davoli et al. [8] proposed SR-based TE in SDN paradigm and also implemented a simple heuristic approach for flow allocation by implementing with Ryu controller and Mininet. They further showed better performance results for that work.

Francesco et al. [9] proposed an efficient segment list encoding algorithm that accounts ECMP (Equal Cost Multi-Path) and also guarantees the optimal path computation by reducing Segment Lists Depth (SLD) in SR based networks. They measured their performance in terms of SLD and described that the better performance is directly proportional to the minimum number of labels included in the segment lists. Lee et al. [10] proposed SR based bandwidth constraint routing algorithm for bandwidth requirement applications. They also improved SR-TE by considering and calculating link congestion index.

In this paper, we implements tunnels and policies for SR-TE implementation and demonstration by utilizing ONOS test environment and mininet. Moreover, we discuss ordinary SR and SR-TE with delay variation of ICMP packet. The paper also shows that SR-TE works well in the SDN environment.

2.1. ONOS Controller

ONOS (Open Network Operating System) provides the control plane for the SDN by managing network components, such as switches and links, and by running software programs or modules to provide communication services to end hosts and neighboring networks [11].

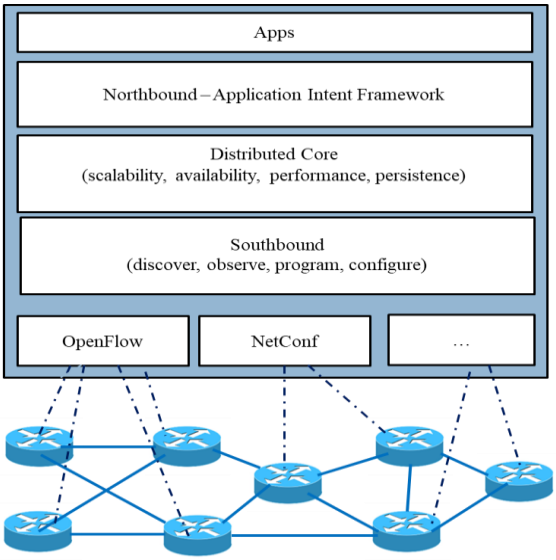

Figure 1. ONOS Architecture Tiers [12]

Moreover, it can run as a distributed system across multiple servers. ONOS's applications consist of customized communication routing, management, or monitoring services for software-defined networks. Figure 1 shows the architecture tiers of ONOS. The ONOS's kernel, core services and applications are written in Java as bundles that are loaded into the Karaf OSGi container. OSGi is a component system for Java that allows modules to

be installed and run dynamically in a single JVM. In this paper, we implement SR-TE in ONOS's SPRING-OPEN virtual environment.

3. Experiment and Testing

In order to create our test environment, a customized version of Open Network Foundation's (ONF) SPRING-OPEN project virtual demonstration environment was used. In SPRING-OPEN virtual environment, the segment routing application is developed on top of the ONOS controller that is provided SDN based control of an island of Open Segment Routers (OSRs).

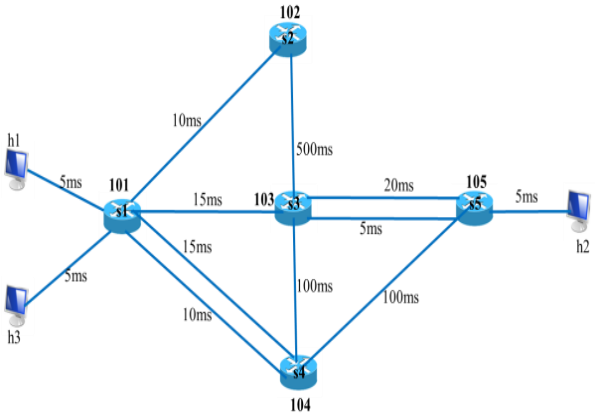

Figure 2. Custom Test Topology

The OSRs route unicast IPv4 packets with the standard MPLS operations that is taken by segment routing principles of globally significant labels and source routing [13]. The focus of this SPRING-OPEN virtual environment is to demonstrate segment routing features on existing hardware switches and stable version of the OpenFlow protocol (1.3). Therefore, we demonstrated TE-SR by utilizing ONOS's SPRING-OPEN virtual environment. The custom test topology includes five switches and three hosts. The switches s1 through s5 are assigned with SR labels, node SID 101 through 105 respectively, and set the different delay in each link as shown in Figure 2.

Table 1 shows all the possible paths and total delay for source hosts (h1, h3) and destination host (h3). By default, there are no SR paths or SR rules implemented; therefore packets pass through the network using ECMP and SPF (Shortest Path First) algorithms. The standard SPF path selection behavior can be seen by sending ICMP packets from host h1 to host h2 across the network. As shown in Figure 3, the ping time varies roughly between 85 ms and 270 ms.

Table 1. Possible Paths between (h1, h3) and h2

Source	Destination	Path	Total Delay(ms)
h1, h3	h2	p1 = s1-s2-s3-s5	540
		p2 = s1-s2-s3-s5	525
		p3 = s1-s3-s5	45
		p4 = s1-s3-s5	30
		p5 = s1-s3-s4-s5	225
		p6 = s1-s4-s3-s5	125
		p7 = s1-s4-s5	125
		p8 = s1-s4-s5	120

```
mininet> h1 ping h2
PING 10.10.2.102 (10.10.2.102) 56(84) bytes of data.
64 bytes from 10.10.2.102: icmp_seq=1 ttl=62 time=253 ms
64 bytes from 10.10.2.102: icmp_seq=2 ttl=62 time=168 ms
64 bytes from 10.10.2.102: icmp_seq=3 ttl=62 time=168 ms
64 bytes from 10.10.2.102: icmp_seq=4 ttl=62 time=160 ms
64 bytes from 10.10.2.102: icmp_seq=5 ttl=62 time=172 ms
64 bytes from 10.10.2.102: icmp_seq=6 ttl=62 time=155 ms
64 bytes from 10.10.2.102: icmp_seq=7 ttl=62 time=154 ms
64 bytes from 10.10.2.102: icmp_seq=8 ttl=62 time=97.4 ms
64 bytes from 10.10.2.102: icmp_seq=9 ttl=62 time=267ms
64 bytes from 10.10.2.102: icmp_seq=10 ttl=62 time=272 ms
64 bytes from 10.10.2.102: icmp_seq=11 ttl=62 time=100ms
64 bytes from 10.10.2.102: icmp_seq=12 ttl=62 time=104 ms
64 bytes from 10.10.2.102: icmp_seq=13 ttl=62 time=85.0 ms
64 bytes from 10.10.2.102: icmp_seq=14 ttl=62 time=99.2ms
64 bytes from 10.10.2.102: icmp_seq=15 ttl=62 time=119ms
```

Figure 3. Ping test result from h1 to h2

This is an excepted behavior because packets are routed to s5 either through s3 or s4. Paths p3, p4, p7 and p8 represent equal cost path from s1 towards s5. 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 delays that were configured earlier [14]. In this test environment, we use spring-open-cli tool to distribute SR rules to SR-enabled routers or switches and to instruct the ONOS controller. In order to implement a segment routed path in the network, a tunnel containing SR labels of the desired paths needs to be created. We need to implement a policy rule for each created tunnel. Like MPLS, SR tunnel and policy only effect to one direction, for instance: tunnel name, HIGH_BW tunnel only effect from s1 to s5 and the return traffic still uses SPF ECMP calculation. Therefore, the return traffic s5 to s1 needs to have a corresponding reversed path tunnel and policy set as well. In this test, we implement tunnels and policies according to the following assumptions:

1. Host h1 requires high bandwidth to h2 for file transferring.
2. Host h3 needs a low delay path to h2 for VOIP traffic.

3. Path p6 is a high bandwidth path from s1 through s5.
4. Path p3 is a low delay path from s5 through s1.

According to the above assumptions, we implement HIGH_BW tunnel for high bandwidth path p6 with the label path [101, 104, 103, 105] and label stack [103,105]. For return traffic, we also implement HIGH_BW_REVERSE tunnel with label path [105, 103, 104, 101] and label stack [104,101]. There are two links between s3 (103) and s5 (105). Therefore, we implement LOW_DELAY and LOW_DELAY_REVERSE tunnel by using adjacency SIDs (103005 and 105004) to reduce SR label stacks and demonstrate fine-gained TE. Figure 4 shows tunnels creating in SR domain and Figure 5 represents all the created tunnels in this test.

```
mininet-vm (config) # tunnel HIGH_BW
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) # tunnel HIGH_BW_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) # tunnel LOW_DELAY
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) # tunnel LOW_DELAY_REVERSE
mininet-vm(config-tunnel) # node 105
mininet-vm(config-tunnel) # node 103 adjacency 105004
mininet-vm(config-tunnel) # node 101
mininet-vm(config-tunnel) # exit
```

Figure 4. Tunnels creation is SR domain

```
mininet-vm(config)# sh tunnel
# Id              Policies Tunnel Path [Head-->Tail] Label Stack [Outer-->Inner]
-|---------------|--------|-----------------------|-----------------------
1 HIGH_BW                  [101, 104, 103, 105]     [[103, 105]]
2 HIGH_BW_REVERSE          [105, 103, 104, 101]     [[104, 101]]
3 LOW_DELAY                [101, 103, 103005, 105]  [[103005]]
4 LOW_DELAY_REVERSE        [105, 105004, 103, 101]  [[101]]
mininet-vm(config)#
```

Figure 5. Created SR tunnels

The creating SR policy is very similar to creating Access Control Lists (ACLs) with traditional routers. Figure 6 shows CLI commands to create SR policy. In this test, we use policy-type: *tunnel-flow*. The *flow-entry* command is used to match the criteria for the policy. The priority value decides which policy will be used, if multiple matches are found. The policy with a higher priority value is selected in a multiple match event. The priority value may be an integer value between 0 and 65535. In this test, we used priority value: 1000 for each policy. Figures 7 and 8 show policy implementing in SR domain and created policies for each tunnel, respectively.

```
mininet-vm(config-policy) # policy < name > policy-type tunnel-flow
mininet-vm(config-policy) # flow-entry ip <src_ip>/<prefix> <dst_ip>/<prefix>
mininet-vm(config-policy) # tunnel <name>
mininet-vm(config-policy) # priority <number>
```

Figure 6. CLI commands to create policy

```
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 HIGH_BW
mininet-vm(config-policy) # priority 1000
mininet-vm(config-policy) # 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 HIGH_BW_REVERSE
mininet-vm(config-policy) # priority 1000
mininet-vm(config-policy) # exit
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 LOW_DELAY
mininet-vm(config-policy) # priority 1000
mininet-vm(config-policy) # exit
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 LOW_DELAY_REVERSE
mininet-vm(config-policy) # priority 1000
mininet-vm(config-policy) # exit
```

Figure 7. Policies creation in SR domain

```
mininet-vm(config)# sh policy
# Policy Id          Policy Type Priority Dst Mac Src Mac Ether Type Dst IP        IP Protocol Src IP        Dst TcpPort
 Src TcpPort Tunnel Used
-|-----------------|-----------|--------|-------|-------|----------|-------------|-----------|-------------|-----------
|-----------|---------------
1 LOW_COST           TUNNEL_FLOW 1000     *     *     0x2048     10.10.2.102/32 *          10.10.1.0/24   *
 *          HIGH_BW
2 LOW_COST_REVERSE   TUNNEL_FLOW 1000     *     *     0x2048     10.10.1.0/24   *          10.10.2.102/32 *
 *          HIGH_BW_REVERSE
3 LOW_LATENCY        TUNNEL_FLOW 1000     *     *     0x2048     10.10.2.102/32 *          10.10.3.0/24   *
 *          LOW_DELAY
4 LOW_LATENCY_REVERSE TUNNEL_FLOW 1000     *     *     0x2048     10.10.3.0/24   *          10.10.2.102/32 *
 *          LOW_DELAY_REVERSE
mininet-vm(config)#
```

Figure 8. Created SR policies

First, we have sent ICMP requests from h1 to h2. According to LOW_COST and LOW_COST_REVERSE policy that is created in Figure 7, ICMP requests from h1 to h2 will route the network with high bandwidth tunnels, HIGH_BW and the ICMP replies from h2 to h1 through the network with HIGH_BW_ REVERSE tunnel. As shown in Figure 9, the ping time variation is significantly lower than that in Figure 3 and the variation time is nearly 30 ms, which is caused by load balancing behavior over the two links between nodes 103 and 105.

```
root@mininet-vm:~/mininet/custom# ping 10.10.2.102 -I h1-eth0
PING 10.10.2.102 (10.10.2.102) from 10.10.1.101 h1-eth0: 56(84) bytes of data.
64 bytes from 10.10.2.102: icmp_seq=1 ttl=61 time=157 ms
64 bytes from 10.10.2.102: icmp_seq=2 ttl=61 time=186 ms
64 bytes from 10.10.2.102: icmp_seq=3 ttl=61 time=157 ms
64 bytes from 10.10.2.102: icmp_seq=4 ttl=61 time=188 ms
64 bytes from 10.10.2.102: icmp_seq=5 ttl=61 time=159 ms
64 bytes from 10.10.2.102: icmp_seq=6 ttl=61 time=189 ms
64 bytes from 10.10.2.102: icmp_seq=7 ttl=61 time=169 ms
64 bytes from 10.10.2.102: icmp_seq=8 ttl=61 time=190 ms
64 bytes from 10.10.2.102: icmp_seq=9 ttl=61 time=163 ms
64 bytes from 10.10.2.102: icmp_seq=10 ttl=61 time=201 ms
64 bytes from 10.10.2.102: icmp_seq=11 ttl=61 time=178 ms
64 bytes from 10.10.2.102: icmp_seq=12 ttl=61 time=198 ms
64 bytes from 10.10.2.102: icmp_seq=13 ttl=61 time=163 ms
64 bytes from 10.10.2.102: icmp_seq=14 ttl=61 time=204 ms
64 bytes from 10.10.2.102: icmp_seq=15 ttl=61 time=180 ms
```

Figure 9. Pinging from h1 to h2

Finally, we have been carried out ICMP requests from h3 to h2. As the above assumptions,

h3 need low delay path. Therefore, the ICMP request from h3 to h2 through the network using LOW_LATENCY policy and the ICMP reply from h2 to h3 route the network using LOW_LATENCY_REVERSE policy. In Figure 10, the ICMP request and reply time is now considerably lower than the Figure 3 and Figure 9 because the packets through the network by utilizing low delay tunnels and policies.

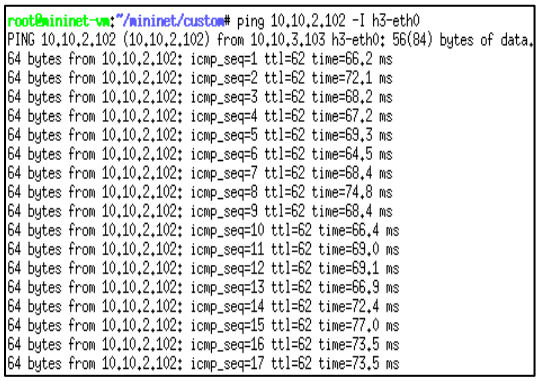

Figure 10. Pinging from h3 to h2

This result has now been demonstrated that the traffic is correctly routed in the test network using the configured SR tunnels and policies. SR can override default ECMP rules by using tunnels and policies functionality. Moreover, there is no need to install flow rules in intermediate nodes that involved in traffic paths. There is still some variance in the ping times, which is caused by the virtualized testing environment. The creation of policies and tunnels is very similar to the Cisco command structure and syntax.

4. Conclusion

In this paper, we studied segment routing and tested its traffic engineering capabilities in MPLS networks by utilizing a virtualized testing environment. By default, SR traverses the network by using ECMP and SPF algorithms. Therefore, the ping time variation is roughly varied. In SR-TE, we implement tunnels and policies for high bandwidth and low delay paths. By applying high bandwidth policies, the ping time variation is nearly 30 ms and it is lower than the default SR. We also got the lowest variation of ping test result and lowest delay time by applying the low delay policies. Moreover, this paper showed SR-TE work well in ONOS controller.

For our future work, we plan SR-TE not only to implement in other virtual environment but also to combine other features like application-aware engineering. We also plan to consider the failure recovery process in SR-TE.

References

[1] Y. Li and M. Chen, “Software-defined network function virtualization: A survey”, IEEE Access-3, 2015, 2542-2553.

[2] W. Kim, J. Li, JW. Hong and YJ. Suh, “OFMon: Openflow monitoring system in ONOS controllers”, *NetSoft Conference and Workshop*, IEEE, 2016 Jun 6, pp. 397-402

[3] Z. Shu, J. Wan, J. Lin, S. Wang, D. Li, S. Rho and C. Yang, “Traffic engineering in software-defined networking: Measurement and management”, *IEEE Access-4*, pp.3246-3256.

[4] https://blog.apnic.net/overcoming-traffic-engineering-challenges-sdn.

[5] https://tools.ietf.org/pdf/draft-ietf-spring-segment-routing-14.pdf

[6] Alcatel Lucent, “segment routing and path computation element”, Technology White Paper, [online] Available, http://www.resources.alctaellucent.com/SegmentRoutingandPathComputationElement.pdf.

[7] https://insight.nokia.com/benefits-segment-routing-and-path-computation.

[8] L. Davoli, L. Veltri, PL. Ventre, G. Siracusano and S. Salsano, “Traffic engineering with segment routing: SDN based architectural design and open source implementation”, *2015 Fourth European Workshop on Software Defined Networks*, IEEE, 2015 December.

[9] F. Lazzeri, G. Bruno , J. Nijhof, A. Giorgetti and P. Castoldi, “Efficient label encoding in segment-routing enabled optical networks”, *IEEE Optical Network Design and Modeling*, International Conference, 2015 May 11, pp. 34-38.

[10] MC. Lee and JP. Sheu, “An efficient routing algorithm based on segment routing in software defined networking”, *Computer Networks*, 2016 July, pp. 44-55.

[11] https://wiki.onosproject.org

[12] https://www.slideshare.net/opendaylight/onos-platform-architecture

[13] https://wiki.onosproject.org/display/ONOS/ProjectDescription

[14] L. Litmanen, “Segment routing”, Helsinki Metropolia University of Applied Science, Thesis, April 2017.