SDN-Based Source Routing for Scalable Service Chaining in Datacenters

Ahmed Abujoda, Hadi Razzaghi Kouchaksaraei, Panagiotis Papadimitriou

SDN-based Source Routing for Scalable Service Chaining in Datacenters

Ahmed Abujoda, Hadi Razzaghi Kouchaksaraei, and Panagiotis Papadimitriou

Institute of Communications Technology, Leibniz Universität Hannover, Germany
{ahmed.abujoda, panagiotis.papadimitriou}@ikt.uni-hannover.de
razzaghi.kouchaksaraei@stud.uni-hannover.de

Abstract. The migration of network functions (NFs) to datacenters, as promoted by Network Function Virtualization (NFV), raises the need for service chaining (*i.e.,* steering traffic through a sequence of NFs). Service chaining is typically performed by installing forwarding entries in switches within datacenters (DCs). However, as the number of service chains in DCs grows, switches will be required to maintain a large amount of forwarding state. This will raise a dataplane scalability issue, due to the relatively small flow table size of switches. To mitigate this problem, we present a software-defined network (SDN) based source routing architecture for scalable service chaining, at which the NF-path path is encoded into the packet header obviating the need for any forwarding state and lookup in the switches. We assess the feasibility and efficiency of our architecture using a prototype implementation.

1 Introduction

Network Function Virtualization (NFV) is an emerging concept aiming to replace special-purpose and hardware-based middleboxes with software-based network functions (NFs) deployed on virtualized infrastructures [7] [8] [24] [21] [22]. Inspired by the success of cloud computing, NFV strives to introduce new business models to the network operators (*e.g.,* NF as a service (NFaaS)), enabling the migration of middleboxes to Network Function Providers (NFPs) in a pay-per-use manner. This can lead to a significant reduction in capital and operational expenses, while enabling elastic provisioning based on service demand. Opting to support NFV and offer NFaaS, many Internet Service Providers (ISPs) have already started to deploy micro-datacenters on their networks for NFaaS offerings to their clients [9].

To implement security and access control policies, middleboxes are deployed in certain locations in enterprise networks, such that traffic can traverse them in a particular order (*e.g.,* a firewall should be deployed before a load balancer to filter malicious traffic). This order needs to be maintained when migrating NFs to NFPs' datacenters (DCs). In other words, there is a need for steering traffic through NFs deployed in DCs to ensure compliance with the client's policy. The so-called *service chaining* can be performed in DCs via the installation of forwarding entries in switches, such that the traffic traverses the NFs in the exact order specified by the client (*i.e.,* as prescribed in the service chain [10], [20]). We refer to this approach as *rule-based service chaining.* Several studies have investigated rule-based service chaining in enterprise and wide-area networks [5], [19], [25]. However, an increasing number of service chains will require a substantial amount of forwarding state in switches, which, in turn, will raise a dataplane scalability issue for NFPs, due to the relatively small flow table size of switches (*i.e.,* a switch flow table can typically store a few thousand entries).

To mitigate this problem, we employ source routing to steer traffic through the NFs of a service chain. In particular, the NF-path is encoded into the packet header obviating the need for any forwarding state and lookup in the switches, *i.e.,* switches forward packets based on the information carried in the packet header. Source routing is an attractive solution for DCs where the number of hops is relatively small (typically there are three switches between access gateways and any server in a DC) in comparison to ISP and enterprise networks [23], [4], [11]. Furthermore, as opposed to the wide-area, source routing does not raise any serious security concerns within DCs, since only switches and hypervisors (on virtualized servers) managed by the DC network operator can insert or remove paths from packet headers. However, source routing raises a set of challenges in terms of scalability and performance. More specifically, the encoded path may increase the packet size beyond the maximum segment size (*e.g.,* 1500 bytes for Ethernet). As such, there is a need to minimize the packet header space used for source routing. Furthermore, source routing should yield packet forwarding rates as high as in rule-based forwarding.

Service chaining with source routing raises the following requirements: (i) the discovery of switch output ports based on the NF-path in order to compose the source routing header, (ii) the encapsulation of the source routing header into the incoming packets, and (iii) traffic steering based on the source routing header. To meet these requirements, we present a software-defined network (SDN) based source routing architecture for scalable service chaining in DCs. We rely on a centralized controller to collect information about the switch ports and subsequently derive the sequence of output ports for each service chain. In this respect, we assume the presence of service mapping techniques, *i.e.,* the assignment of service chains to DC networks [6], [1]. The controller uses OpenFlow [13] to insert the service routing header in the incoming packets at the core switches. We have further implemented a datapath using Click [12] for packet forwarding using the source routing header. We study the feasibility of our architecture using a prototype implementation. Our experimentatal results show that our source-routing based approach for service chaining yields high packet forwarding rates, low header insertion delay (*i.e.,* flow setup time), and a significant reduction in the control communication overhead in DCs.

The remainder of the paper is organized as follows. Section 2 presents our architecture for source routing within DCs using SDN. Section 3 discusses the implementation of our architecture components. In Section 4, we evaluate the performance of our source routing datapath and controller. Finally, Section 5 highlights our conclusions.

2 Service Chaining Architecture

In this section, we elaborate on our source routing based approach for service chaining. Consider the example in Fig. 1 where traffic needs to be routed through three NFs deployed on servers within a DC for service chaining. A straightforward approach is to install a forwarding entry in each switch on the path connecting the NFs, for each service chain. Despite its simplicity, this approach requires maintaining the state of each service chain on multiple switches (in this example, 7 entries for one chain are required) which restricts the number of service chains that can be deployed on a DC (given the small flow table size in switches). Using source routing instead, the path between the NFs can be encoded into the packet header at core switches, eliminating the need for inserting forwarding state on each switch on the path.

Along these lines, we propose to encode the sequence of switch output ports into the header of each packet. For instance, to steer traffic through NF1 and NF2, each packet should carry the port numbers 1, 3, and 5. As such, each switch can merely extract the output port number from the packet and forward it accordingly (*e.g.,* switch A will extract and forward to port number 1). To allow the switches to identify the corresponding port numbers (out of a sequence of port numbers encoded into the packet), we add a counter field to the routing header. This counter identifies the next port number to be read by the next switch on the path, *i.e.,* the counter field specifies the location of the corresponding port number from the beginning of the routing header. For example, when reaching switch B, the counter will carry the value of 6 indicating that switch B should read the sixth field on the port number starting from the beginning of the routing header (Fig. 1). The value of the counter is decremented at each switch to indicate the next port number to be read.

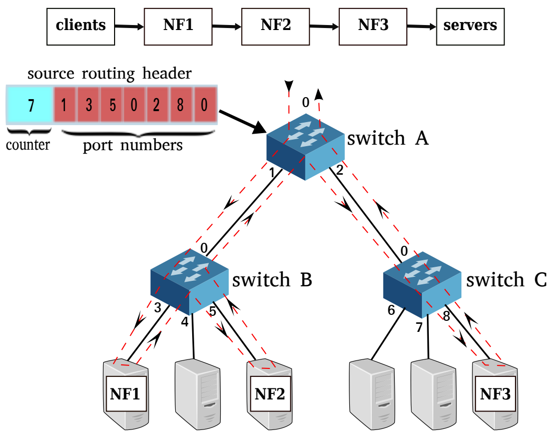


Fig. 1. Example of service chaining using source routing.

In the following, we describe a source routing based architecture for service chaining leveraging on SDN. Our architecture consists of four main components (Fig. 2):

– **Source Routing Encapsulation (SRE) switch** is an OpenFlow [13] switch which inserts the source routing header into each incoming packet based on the configuration provided by the source routing controller. In a DC we consider SRE switch functionality at the core level.

– **Source routing controller** provides support for: (i) topology discovery to keep track of the different links and switches on DC network, (ii) path-to-ports translation to identify the corresponding switch output ports for the NF-path, and (iii) flow table configuration on SRE switches for the insertion of source routing headers on incoming packets.

– **Source routing switch** extracts the output port from the source routing header and forwards the packet accordingly. This role is mainly fulfilled by aggregation and top-of-the-rack (ToR) switches.

– **Southbound interface** provides an interface for the controller to install flow entries on SRE switches and perform topology discovery (*i.e.,* collect information about the switch port numbers and active links).

– **Northbound interface** enables the NFV orchestrator to submit the service chain mapping to the source routing controller.

Inserting the port numbers of the entire path on the packet headers may significantly increase the packet size leading to high bandwidth consumption and/or packet sizes beyond the maximum segment size (MSS). For instance, a service chain consisting of 10 NFs which are assigned to different racks requires a routing header with 30 port numbers. Assuming each port number is carried on 8-bits field, a routing header adds 31 extra bytes (1 byte is needed for the counter field) to each incoming packet. This would lead to the consumption of 48% more bandwidth for packets with the size of 64 bytes and the fragmentation of packets with a size larger than 1470 bytes (with Ethernet).

To mitigate this problem, we use more than one SRE switches along the path. In this respect, the service chain is subdivided into pathlets where each pathlet starts and ends with an SRE switch. The location and number of SRE switches are identified by the DC network operator based on the DC network topology and the employed service mapping algorithm. Minimizing the number of racks used for each service chain reduces the number of hops and consequently, the number of SRE switches per chain.

An alternative approach is to deploy both SRE switch and source routing functionality within all DC switches, allowing the DC operator to dynamically install routing headers on packets in different locations of the network. To identify packets reaching the end of their pathlet, we use the zero value of the counter field (indicating no further ports to read). While this approach provides more flexibility, it increases the complexity of the switch design.

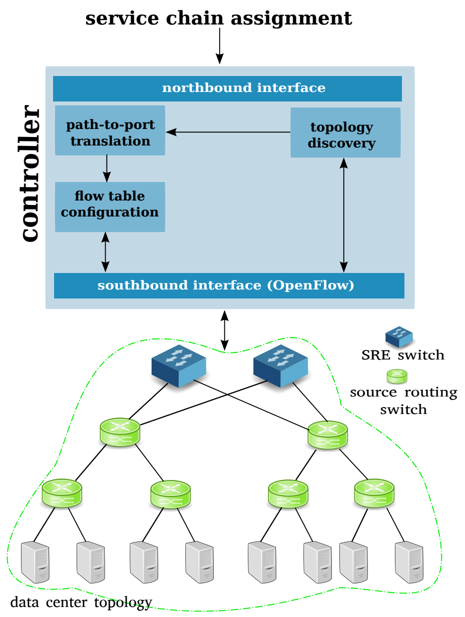


Fig. 2. Source routing architecture.

To quantify the benefits of source routing, we calculated the amount of forwarding state required to steer traffic across a service chain's path using source routing (with a single and multiple SRE switches – 1 SRE switch for 10 hops) and rule-based forwarding. According to Fig. 3, source routing with a single and multiple SRE switches achieves a significant reduction in the state compared to rule-based forwarding.

3 Implementation

In this section, we discuss the implementation of our architecture components:

– **Source routing header.** To insert the routing header into the incoming packets, we use the destination MAC address and further add a VLAN tag and a MPLS label stack to each packet. In particular, we encode the port numbers in the destination MAC address, the VLAN ID, and the MPLS label (OpenFlow 1.0, which we use for our implementation, does not allow modifying other VLAN and MPLS fields). By combining these fields, we can store 10 port numbers per packet, where each field is 8-bit long and supports switches with up to 256 ports. To store the counter value, we use the TTL field of the MPLS header. We can further increase the number of encoded ports by inserting more MPLS stack labels in each packet.

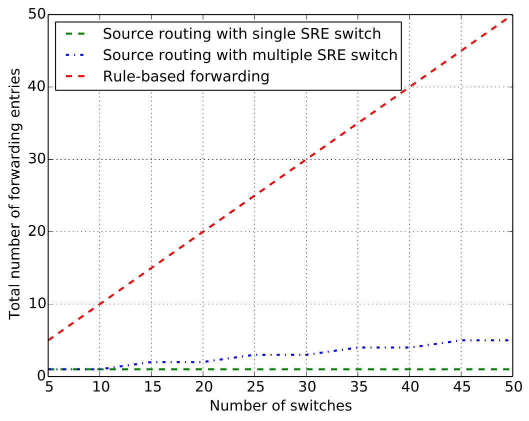


Fig. 3. Forwarding state with source routing and rule-based forwarding.

– **Source routing controller.** We use POX [18] to implement the different components of our controller. Using OpenFlow, the controller collects and stores the port numbers of each switch. Based on this information, the controller extracts the switch ports for each service chain path. Subsequently, the controller inserts the port numbers into a bit vector. This bit vector is further broken down into the MAC destination address, the VLAN ID and the MPLS label. Next, a flow entry is installed in the SRE switch using OFPT_VENDOR [15] message. This message carries the flow matching fields (*e.g.,* source/destination IP, source/destination port numbers and protocol), the VLAN tag, the MPLS stack, the destination MAC address as well as the output port number.

– **SRE switch.** We rely on OpenvSwitch [16] to insert routing headers on the arriving packets. OpenvSwitch exposes an OpenFlow interface to the controller to install forwarding rules. As such, the controller inserts the routing headers on the packets by adding a VLAN tag and a MPLS label stack to each packet and updating its destination MAC address.

– **Source routing datapath.** We extend Click Modular Router [12] with a new element, SourceRouter, which extracts the values of the VLAN ID, MPLS label, MPLS TTL, and the destination MAC address and subsequently combines them into the routing header (see Fig. 1). Based on the counter value, the element reads the corresponding port number through which the packet is forwarded. Combining our element with existing Click elements for packet I/O, we implement a source routing datapath (Fig. 4).

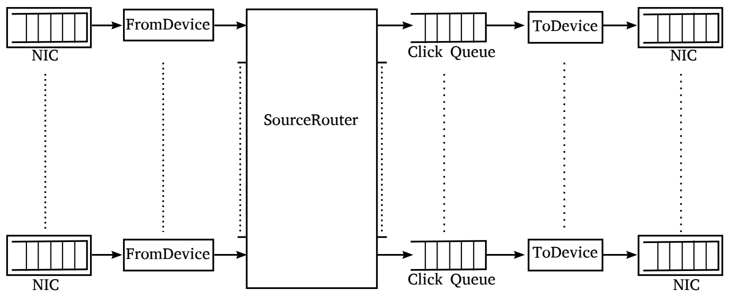


Fig. 4. Source routing datapath implementation with Click.

4 Experimental Evaluation

In this section, we evaluate the performance of our source routing datapath and controller. In particular, we use our emulab-based testbed to measure the packet forwarding rate and the computational requirements of our source routing switch, as well as the setup time and communication overhead of our controller. We run our experiments on 4 servers equipped with an Intel Xeon E5520 quad-core CPU at 2.26 GHz, 6 GB DDR3 RAM and a quad 1G port network interface cards (NICs) based on Intel 82571EB.

4.1 Source Routing Switch Performance

To measure the packet forwarding rate, we deploy our source routing datapath on one server and use two other servers as traffic source and sink. The switch and destination servers run Click modular router 2.0 in kernel space on Linux kernel 2.6.32. To generate traffic, we use a NetFPGA-based [14] packet generator which is installed on and controlled through our source server. Since the switch on our testbed filters packets with VLAN and MPLS header, we encapsulate our packets in IP-in-IP header. We measure the forwarding rate of our switch with packet sizes of 85 bytes including the source routing header and the IP encapsulation header. We further compare our switch performance with rule-based forwarding where we forward packets based on packets' destination IP address using a routing table with 380K entries. Fig. 5 shows that our source routing switch achieves more than 30% higher forwarding rate than rule-based routing. This performance is achieved using a single CPU core.

Furthermore, we use Oprofile 1.1.0 [17] to measure the computational overhead of our switch in terms of CPU cycles per packet. As depicted in Table 1, our source routing datapath consumes less than 8% of the total CPU cycles per packet. On the other hand, packet I/O operations represent the dominant share of the CPU cycles/packet. Similar results for packet I/O operations are shown in [2].

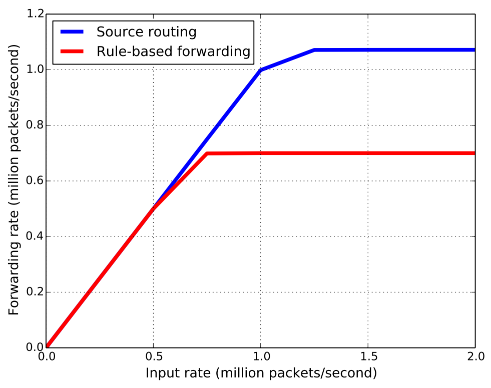


Fig. 5. Packet forwarding rate of our source routing datapath with a single CPU core.

Table 1. Average CPU cycles/packet

Element	**cycles/packet**	**Percentage**
Source Routing Switch	156	7.9%
Packet I/O	1008	51.3%
Encapsulation	259	13.2%
Decapsulation	81	4.2%
Others	458	23.4%
Total	**1965**	100%

4.2 Controller Performance

To evaluate the performance of our controller, we use 4 servers, two of which act as traffic source and sink, while the other two servers host the controller and the SRE switch. Initially, we measure the flow setup time (*i.e.,* the time required to insert the source routing header at the SRE switch) which we define as the time elapsed from the flow's first packet arrival at the SRE switch ingress port till its departure from the egress port.

As depicted in Fig. 6, the flow setup time does not change significantly across the various flow arrival rates. We further break down the setup time into multiple components. In particular, we measure the time required for the source routing header computation, the time the control packet takes to traverse the POX and kernel stack on the controller server in both directions (between the controller and the switch), the RTT (between the controller and the SRE switch) and the SRE switch processing time. As shown in Table 2, source routing header computation consumes less than 29% of the total setup time. Instead, most of the setup time is spent in POX, kernel stack, and the SRE switch processing.

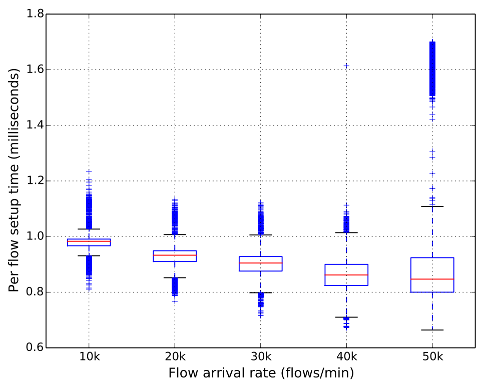


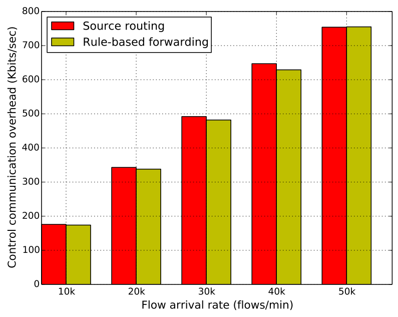
Fig. 6. Setup time per flow.

Table 2. Flow setup time

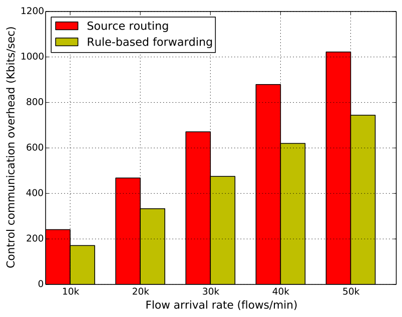
Element	**Time (milliseconds)**
Source routing header computation	0.25
POX processing + kernel stack	0.24
RTT	0.1
SRE switch processing	0.3
Total	**0.89**

We also compare source routing with rule-based forwarding in terms of control communication overhead. We first measure the communication overhead on a single switch for different flow arrival rates. In this respect, we measure the control traffic between the switch and controller per direction (noted as uplink and downlink overhead). Our measurements show that both source routing and rule-based forwarding consume the same amount of bandwidth at the uplink (Fig. 7(a)), since both approaches use the same packet size and format (*i.e.,* OFPT_PACKET_IN [15]) to transfer the packet fields to the controller. On the other hand, for downlink, source routing consumes more bandwidth than rule-based forwarding (Fig. 7(b)), due to the extra packet fields (*i.e.,* VLAN and MPLS) required to install the source routing header.

Using the results we obtained for a single switch, we further calculate the communication overhead with a diverse number of switches. For our calculation, we consider DCs with a fat-tree topology [3] (Fig. 8). The DC components in a fat-tree topology can be modeled based on the number of ports per switch. More precisely, a fat-tree topology with k-port switches has $(5(k^2))/4)$ switches where $(k/2)^2$ of these switches are core switches and k^2 are aggregation and edge switches.



(a) Uplink overhead



(b) Downlink overhead

Fig. 7. Controll overhead for a single switch.

We calculate the control overhead of source routing and rule-based forwarding for fat-tree topologies with a diverse number of ports per switch. As shown in Fig. 9, source-routing introduces significantly lower communication overhead in comparison to rule-based forwarding. We observe that the savings in communication overhead increase with the size of the DC network, since the source routing controller needs to communicate only with the core switches of the DC (Section 1).

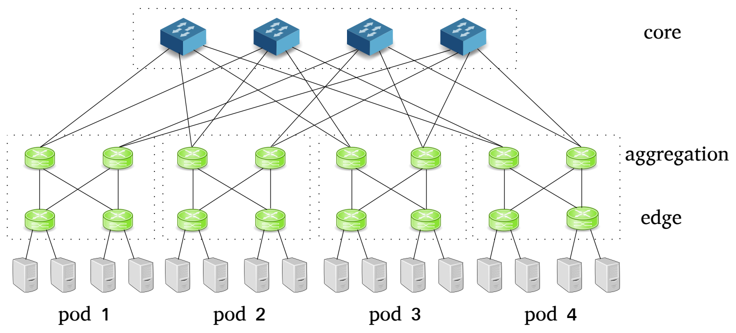
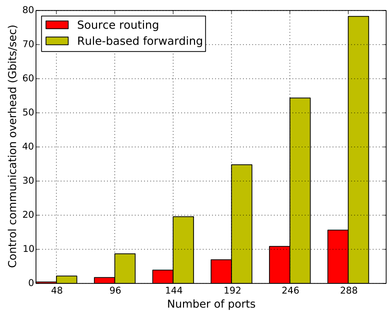


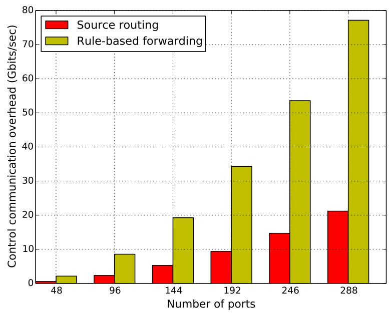
Fig. 8. Fat tree DC network topology.

5 Conclusions

In this paper, we presented a SDN-based source routing architecture for service chaining within DCs. Our architecture encodes the path of each service chain into the packet headers leading to significant reduction in the forwarding state per switch and better dataplane scalability. To insert the source routing header



(a) Uplink overhead



(b) Downlink overhead

Fig. 9. Control overhead for multiple switches.

into the incoming packets, we implemented a centralized controller that extracts switch output ports from the chain's path and subsequently encodes the port numbers into the packet headers by configuring SRE switches with OpenFlow. We further implemented a source routing datapath for packet forwarding based on the source routing header.

Using our prototype implementation, we showed that our source routing datapath achieves high packet forwarding rates and incurs low computational overhead. In addition, we showed that our controller incurs low flow setup delay and achieves a significant reduction in communication overhead in comparison with rule-based forwarding, for different DC network sizes.

In future work, we will seek to coordinate service mapping (*e.g.,* using our service chain-to-DC mapping algorithm in Nestor [6]) with SRE switch placement to achieve additional savings in terms of forwarding state and provide support for NFV deployments at massive scale.

6 Acknowledgments

This work was partially supported by the EU FP7 T-NOVA Project (619520).

References

1. Abujoda, A., Papadimitriou, P.: DistNSE: Distributed Network Service Embedding Across Multiple Providers
2. Abujoda, A., Papadimitriou, P.: Profiling Packet Processing Workloads on Commodity Servers. In: Tsaoussidis, V., J.Kassler, A., Koucheryavy, Y., Mellouk, A.(eds.) Wired/Wireless Internet Communication. LNCS, vol.7889, pp.216-228. Springer, Berlin Heidelberg (2013)
3. Al-Fares, M., Loukissas, A., Vahdat, A.: A scalable, commodity data center network architecture. ACM SIGCOMM Computer Communication Review, pp. 63-74, 38(4),(2008)

4. Ashwood-Smith, P., Soliman, M.: SDN State Reduction, https://tools.ietf.org/html/draft-ashwood-sdnrg-state-reduction-00
5. CISCO Policy-based routing. Technical report, http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/15-0SY/configuration/guide/15_0_sy_swcg/policy_based_routing_pbr.pdf
6. Dietrich, D., Abujoda, A., Papadimitriou, P.: Network service embedding across multiple providers with nestor. In: IFIP Networking Conference (IFIP Networking), IEEE (2015)
7. Dobrescu, M., et al.: RouteBricks: Exploiting Parallelism To Scale Software Routers. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp. 15-28, Big Sky, USA (October 2009)
8. ETSI Network Function Virtualization, http://www.etsi.org/technologiesclusters/technologies/nfv
9. Frank, B., et al.: Collaboration Opportunities for Content Delivery and Network Infrastructures. In: Haddadi, H., Bonaventure, O. (Eds.) Recent Advances in Networking. vol. 1, pp. 305-377, (2013)
10. Gember, A., et al.: Stratos: Virtual middleboxes as first-class entities. UW-Madison. 15, TR1771 (2012)
11. Hari, A., Lakshman, T. V., Wilfong G.: Path Switching: Reduced-State Flow Handling in SDN Using Path Information.
12. Kohler, E., et al.: The Click Modular Router. ACM Transaction on Computer Systems, 18(3), (2000)
13. McKeown, N., et al.: OpenFlow: Enabling Innovation in Campus Networks, In: ACM SIGCOMM CCR, 38(2), pp.69-74, (2008)
14. NetFPGA, http://netfpga.org
15. OpenFlow Switch Specification v1.0, http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
16. OpenvSwitch, http://openvswitch.org
17. Oprofile, http://oprofile.sourceforge.net
18. Pox, www.noxrepo.org/pox
19. Qazi, Z. A., et al.: SIMPLE-fying middlebox policy enforcement using SDN. In: SIGCOMM (2013)
20. Quinn, P., Nadeau, T.: Problem Statement for Service Function Chaining, RFC 7498, 2015, https://tools.ietf.org/html/rfc7498.
21. Sekar V., et al.: The Design and Implementation of a Consolidated Middlebox Architecture. USENIX NSDI, San Jose (April 2012)
22. Sherry, J., et al.: Making Middleboxes Someone Elses Problem: Network Processing as a Cloud Service. In: ACM SIGCOMM, pp. 13-24, Helsinki, Finland (August 2012)
23. Soliman, M., et al: Exploring source routed forwarding in SDN-based WANs. In: IEEE International Conference on Communications (ICC), pp. 3070-3075, Australia (2014)
24. T-NOVA Project, http://www.t-nova.eu
25. Zhang Y., et al., StEERING: A Software-Defined Networking for Inline Service Chaining. In ICNP (2013)