Leveraging on Source Routing for Scalability and Robustness in Datacenters

Konstantinos Papadopoulos, Panagiotis Papadimitriou
University of Macedonia, Greece
{konpapad, papadimitriou}@uom.edu.gr

***Abstract*—The transition towards 5G leads to higher in-network processing demands, which, in turn, raise more severe dataplane requirements, *e.g.,* in terms of datacenter switch forwarding state. Since legacy switches coupled with layer-2/3 forwarding are expected to introduce significant dataplane limitations, we leverage on source routing as means for a more scalable and robust switching fabric, especially for datacenters with large-scale network function deployments.**

In this respect, we present a practical implementation of source routing using OpenFlow-enabled switches. To assess the gains of source routing in virtualized infrastructures, we perform a comparative study between source routing and L2 forwarding, using our implementation in Mininet. Our evaluation results show significant forwarding state savings and hassle-free network updates, when the proposed source routing scheme is employed.

I. Introduction

5G aims at providing a wide range of service capabilities to address the various needs of vertical industries, such as automotive, entertainment, e-health, and manufacturing. In this respect, next-generation network architectures seek to facilitate the deployment of new functionality into the network [1], [2]. Network Function Virtualization (NFV) [2], [3], [4] and Software-Defined Networking (SDN) [5], [6] are considered as key enablers towards 5G, as they empower the dynamic deployment and chaining of customisable service elements in virtualized infrastructures.

An envisaged large-scale deployment of virtualized network functions (VNFs) would require a very large amount of forwarding state in datacenter switches in order to steer traffic through a sequence of VNFs in a specified order (*i.e.,* known as service chaining). Thereby, L2 forwarding, supported by Ethernet switches, in conjunction with their Forwarding Information Base (FIB) restrictions (due to the small size of the switch TCAM) introduce significant dataplane limitations. This could eventually restrict the network capabilities in terms of multi-service and multi-tenancy, as sought by 5G.

Source routing has been seen as a potential solution to this problem [7], [8], [9], [10], [11], as this routing scheme enables the encoding of the path into the packet header, which significantly reduces the forwarding state in switches. In particular, with source routing, switches are required to maintain only a small number of flow-independent flow entries. Besides forwarding state reduction, source routing can facilitate network updates, since path updates can be encoded into the packet header, obviating the need for consistent flow tables updates [12].

Since source routing is not supported by legacy Ethernet switches, we investigate ways for a practical implementation using commodity programmable switching hardware (*i.e.,* OpenFlow switches). In particular, we discuss the implementation of a port switching technique for source routing, certain processing operations of which (*i.e.,* label matching, advancement of pointer to the next label) require careful attention for their correct handling with OpenFlow. Our source routing datapath is implemented and validated using OpenvSwitch [13] in datacenter network topologies emulated with MiniNet [14]. We further conduct an evaluation study of source routing, quantifying its gains in datacenters with respect to (i) forwarding state and (ii) network updates. Our evaluation is conducted separately for two common cases: (i) an enterprise datacenter, and (ii) a multi-tenant datacenter serving as an NFV infrastructure.

The remainder of the paper is organized as follows. In Section II, we discuss the functionality and the main benefits of source routing. In Section III, we provide a detailed description of our source routing implementation using OpenFlow. In Section IV, we evaluate the efficiency of source routing in terms of forwarding state and network update, using our implementation. Section V provides an overview of related work. Finally, in Section VI, we highlight our conclusions.

II. Source Routing

Source routing has been seen as a viable approach for the reduction of forwarding state in datacenters (DCs) with large-scale service (*e.g.,* NFV) deployments. For example, L2 forwarding with Ethernet leads to switch flow entries proportional to the end-points. Considering the massive level of consolidation in multi-tenant DCs and the widespread adoption of NFV, the limited Ternary Content Addressable Memory (TCAM) space in commodity switches is certainly insufficient to store the required amount of state.

Source routing can achieve a significant degree of TCAM saving in switches, as it commonly requires a minimal set of flow-independent entries in the switch flow table. This stems

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE (project code: T1EDK-02947).

978-1-7281-3627-1/19/$31.00

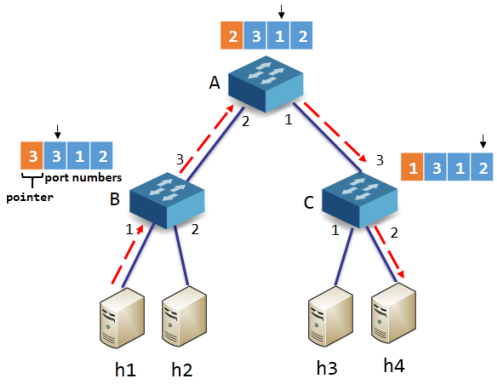

Fig. 1: Source routing example.

from the encoding of the path into the packet header, which allows source routing datapaths to perform simplistic packet forwarding. Such minimal and nearly static forwarding tables open up new opportunities and also facilitate certain network operations. More specifically, source routing allows for smaller TCAM, simpler packet processing, and thus, cheaper switching hardware. Furthermore, source routing greatly simplifies path updates. Whereas path updates across DC switches are typically handled with consistent flow table updates, which, in turn, requires appropriate primitives and abstractions [12], source routing allows for much simpler and hassle-free path updates. In essence, source routing handles path updates, by encoding the new path into the packet header, *i.e.,* an operation associated with a specific switch. In Section IV, we demonstrate these benefits of source routing, using our implementation.

We hereby describe the main functionality of source routing, before elaborating on its implementation in Section III. Source routing mainly involves two processing operations: (i) path encapsulation and (ii) port switching. Path encapsulation consists in the encoding of a set of labels into the packet header. These labels correspond to the sequence of switch ports that the packet has to to traverse, without necessarily matching the port numbers. In a virtualized DC, path encapsulation is expected to take place in an edge or a virtual switch (*e.g.,* OpenvSwitch). For path encapsulation, we assume the presence of a centralized (*e.g.,* SDN) controller, which has network-wide visibility, performs path computation, and extracts the sequence of ports along each computed path [10].

Port switching is carried out in source routing datapaths with minimal forwarding state. More precisely, the switches extract the next label from the packet header, and subsequently, perform a table lookup to identify the corresponding output port. Since multiple labels are encoded into the packet header, a pointer is required, so that a switch can identify the label to be extracted. Since the pointer occupies more space in the header, source routing requires a careful selection of the header fields that will be used for the labels and pointer encoding. Additional header space can be utilized from VLAN/MPLS headers, at the expense of extra transmission overhead. Fig. 1 illustrates an example of such a source routing scheme.

III. Implementation

In this section, we first discuss alternative ways to implement source routing (SR) using OpenFlow-enabled switches, and, subsequently, provide the details of our own implementation. Inline with the implementation options discussed in [7], [10], we have identified the following ways for SR implementation:

Label Sequence and Pointer. The most straightforward way to implement source routing is to encode the sequence of labels into header fields, and also use a separate header field for the encoding of the pointer to the next label. Candidate fields for storing labels (without extra transmission overhead) are source/destination MAC address and IP(v6) address. In this case, multiple labels will have to be encoded into the packet header, which requires matching over certain bits in the header field. This feature (bitmasking) is supported by the latest OpenFlow versions.

Label Sequence and TTL. An alternative way to the previous implementation is to use TTL instead of a dedicated field for the pointer to the next label. In particular, the deviation of TTL from the initial value gives the number of hops that the packet has traversed, and, hence, can implicitly serve as the pointer. This leads to header space conservation, provided that switches can handle TTL operations correctly.

Switch IDs. Another option is to define the path as a sequence of switch IDs, assuming globally-unique switch IDs. This merely requires a static match table with the IDs of neighboring switches in each switch. This eliminates the need for a pointer, since a switch can recognize its own ID in the sequence. This implementation scheme is highly dependent on the number of switches (since it affects the length of the switch ID).

After careful inspection of the alternative implementation schemes, we chose the *Label Sequence and TTL*, since it allocates less header space than *Label Sequence and Pointer* and is also more suitable than *Switch IDs* for large-scale networks. We particularly use source and/or destination MAC addresses for the encoding of the labels (corresponding to the sequence of switch output ports). In particular, MAC address fields are populated with a sequence of output ports that each switch in the path will steer each packet. We also use the TTL field, as a pointer to the next port number encoded in the MAC address.

The process is explained in the example of Fig. 1. Assume that host *h1* wishes to send a packet to *h4*. The packet is forwarded to the Top-of-the-Rack (ToR) switch B, which inserts the port sequence to *h4*, based on the destination IP address. TTL is set with an initial value matching the hop count (*i.e.,* 3 in the case of Fig. 1), and as decremented along the path, it always points to the next port number encoded in the MAC address. Therefore, Switch B reads the TTL and forwards the packet to the corresponding port 3, towards switch A and decrements the TTL. Subsequently, Switch A reads the following port number based on the current value

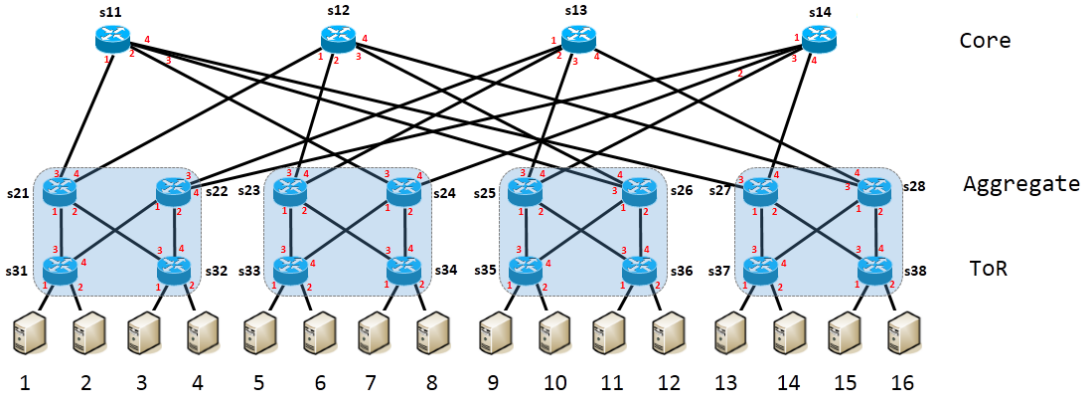

Fig. 2: Fat-tree data center network topology.

of TTL (*i.e.,* 2), and forwards the packet to port 1 towards switch C. Finally, switch C decodes the next port from the packet header, sending the packet to the destination host *h4*. We note that the whole SR process is transparent to the end-hosts. In case the destination MAC address is used for path encapsulation as well (*i.e.,* to encode a larger number of ports), the ToR switch delivering the packet to the destination can rewrite the destination MAC address, hiding SR from the destination host.

To illustrate the details of our OpenFlow implementation, we use a region of a fat-tree DC network with k = 4 pods, as shown in Fig. 2. In this example, we encode the path into the source MAC address, which is sufficient for this network diameter. We also use 4 bits for each port, which is suitable for switches with up to 16 ports (another two bits can meet the requirements for port densities up to 48). This essentially permits the encoding of 12 ports (*i.e.,* 12 hops) into the source MAC address.

To install the appropriate flow entries, we rely on the OpenFlow Extensible Match (OXM) capabilities of OpenFlow 1.3. All required flow entries are proactively inserted in the switches. More precisely, two types of flow entries have to be installed: (i) path insertion entries that insert the source routing path to the MAC address field, and (ii) SR forwarding entries that are used to forward traffic based on the encoded source routing path. In the following, we provide examples of these two types of flow entries.

A. Path Insertion Entries

Hereby, we show an OpenFlow entry that inserts the appropriate port sequence to packets in switch *s31* with host 6 as the destination.

```
priority=10,ip,dl_dst=00:00:00:00:00:06,
nw_ttl=64,actions=load:0xc->NXM_NX_IP_TTL
[],mod_dl_src:32:12:00:00:00:00,output:3
```

Assume that the hosts connected to switch *s31*, ignorant of the SR routing process, send a packet to a destination based on its MAC address. The corresponding OpenFlow field in the flow entry (*dl_dst*) is used to match the destination MAC address[1] of host 6. The default initial TTL value (64) is used to identify a non-SR packet, sent from the connected hosts 1 or 2. Assume that the port sequence from switch *s31* to host 6 is 3-3-2-1-2 (Fig. 2). The flow entry will forward the packet to port 3 towards switch *s21*. As such, there is no need to encode this port in the packet header. The remainder of the port sequence (*i.e.,* 3-2-1-2) is inserted into the source MAC address. The TTL field is set to 12, which corresponds to the maximum hop count of the SR path (*i.e.,* 12, as explained above). This flow entry has a high priority, ensuring that path insertion entries are applied over forwarding entries (*i.e.,* in the case of multiple matches). In our example, 16 path insertion entries have to be installed in switch *s31*, *i.e.,* one for each host.

B. SR Forwarding Entries

Forwarding entries are installed in SR switch datapaths to perform forwarding based on the SR header. With respect to SR forwarding, the main challenge is how to extract the correct port number from the sequence of ports encoded into the packet header field (*i.e.,* source MAC address, in our example). In our case, the TTL contains the pointer to the next port number. Since the corresponding switch port is not in the same position for all incoming packets, a separate flow entry needs to be installed for every possible pointer position. As such, in our example (*i.e.,* 4-bit port numbers encoded into the source MAC address), 12 SR forwarding entries per SR datapath are required. Below, we append a subset of these 12 forwarding entries.

```
priority=1,ip,nw_ttl=12 actions=dec_ttl,
output:NXM_OF_ETH_SRC[44..47]
priority=1, ip,nw_ttl=11 actions=dec_ttl,
output:NXM_OF_ETH_SRC[40..43]
priority=1, ip,nw_ttl=10 actions=dec_ttl,
output:NXM_OF_ETH_SRC[36..39]
...
priority=1, ip,nw_ttl=1 actions=dec_ttl,
output:NXM_OF_ETH_SRC[0..3]
```

[1]The destination IP address could be used instead for forwarding and matching.

In each of these forwarding entries, a separate bit range (quartet of bits) is extracted from the source MAC address field, depending on the TTL value. For example, if TTL is 12 then the current port is retrieved from the 4 most significant bits (*i.e.,* bits 44–47). Besides forwarding to the correct port, the set of actions also includes TTL decrement to advance the pointer to the next port. Forwarding entries have a lower priority than path insertion entries, since path encapsulation precedes.

IV. Evaluation

In this section, we discuss our evaluation results. Our main goal is to assess the potential gains of source routing in terms of switch forwarding state and network updates. Our evaluation is conducted on Mininet [14], a widely used network emulation environment. Our evaluation tests are carried out on a three-layer fat-tree DC topology, with the same characteristics of the topology depicted in Fig. 2. L2 forwarding (*i.e.,* switches store L2 forwarding entries for all hosts in the DC) comprises the baseline in our measurements. We compare the forwarding state between SR and L2 in two cases: (i) an enterprise datacenter, at which all hosts communicate with each other, and (ii) a multi-tenant datacenter which hosts service chains (*i.e.,* chains of virtualized network functions). We also compare the response time during network update in the enterprise datacenter case.

A. Forwarding State

We initially compare the forwarding state in switches with L2 and SR in the enterprise scenario (*i.e.,* with all-to-all communication). Fig. 3 illustrates the number of flow entries in (i) switches with L2 forwarding, (ii) ToR switches that perform path insertion with SR, and (iii) core and aggregation switches that perform packet forwarding with SR. In each case, Fig. 3 presents the average number of flow entries across each switch category. We note that these results are extrapolated from measurements with fewer hosts (using our implementation in Mininet), based on a smaller region of the same fat-tree topology.

Fig. 3 shows the forwarding state for 100, 1000, and 10000 hosts. We use 6 bits for the encoding of each port, whereas we also allocate both source/destination MAC addresses to support up to 16 hops per path. Since L2 requires a number of flow entries proportional to the number of hosts, in all three cases, the L2 state exceeds by far the state maintained in core and aggregation switches with SR. SR yields higher state requirements in ToR switches, at which the path is encapsulated. This amount of the state in ToR switches is comparable with L2 switches. However, it is possible to delegate path insertion into virtual switches (*e.g.,* OpenvSwitch), alleviating edge switches from this burden, thereby, enabling simplistic forwarding across all switching layers.

B. Network Updates

In the enterprise datacenter case, we further seek to assess the gains of SR during network updates. In particular, the

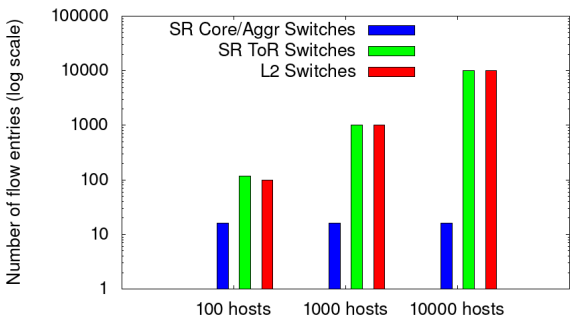

Fig. 3: Forwarding state per switch type with SR and L2 forwarding.

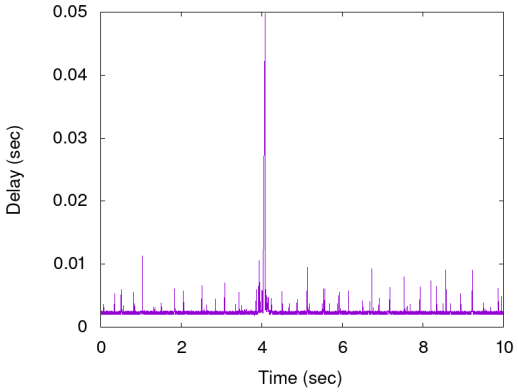

Fig. 4: Packet inter-arrival time with L2.

redirection of a flow to a new path may lead to transient implications, such as pack loss, retransmissions, increased delays, and/or out-of-order packets. To investigate such potential implications during network updates, we measure the packet inter-arrival times with UDP and TCP traffic. Specifically, we generate UDP traffic between two hosts in our fat-tree topology using D-ITG [15]. The generated traffic rate is 500 packets/sec with packet size of 500 bytes. In the 4^{th} second, a path update occurs. In L2 forwarding, the path update is handled by the installation of flow entries on all switches along the two paths. In SR, the path insertion entry is updated with a new entry that encapsulates the new path.

Figs. 4 and 5 show the packet inter-arrival times with L2 forwarding and SR, respectively. In L2, inter-arrival time increases by a factor of 23 in the 4^{th} second that the path update takes place. Besides this implication, 17 packets are lost on average. In contrast, no perceptible traffic disruptions occur with SR (Fig. 5). We have obtained similar results with TCP traffic, *i.e.,* 14 lost packets and an increase in packet inter-arrival time by a factor of 21. Essentially, these results corroborate the efficiency of SR during network updates.

C. Forwarding State with Service Chaining

We further conduct an additional comparative study of SR against L2 forwarding, in multi-tenant DCs hosting service chains. A service chain consists of a sequence of VNFs assigned to servers. The path length of a service chain is highly

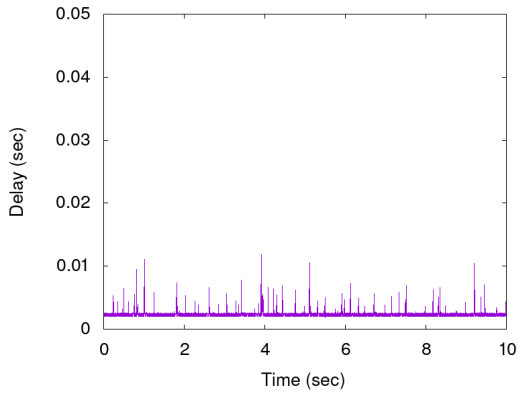

Fig. 5: Packet inter-arrival time with SR.

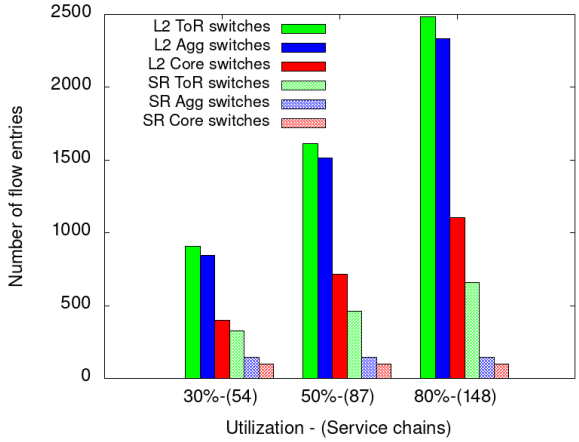

Fig. 6: Flow entries per switch type (128 hosts).

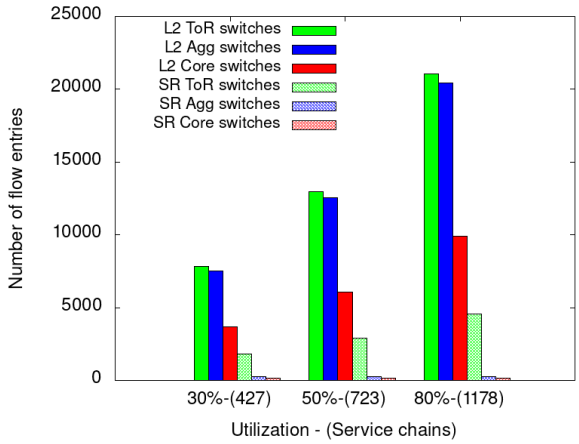

Fig. 7: Flow entries per switch type (1024 hosts).

dependent on the number of VNFs and the degree of VNF co-location. More specifically, large service chains with a small degree of VNF co-location require a larger hop-count, and consequently, more state in DC switches. Since this scenario is more challenging, we conduct additional measurements to quantify the gains in terms of forwarding state in this case. In this respect, we have implemented a service chain generator, which creates chains with 3 to 8 VNFs. Each VNF in the chain is associated with certain computational demands expressed in CPU units. We further use a topology generator, which, in conjunction with the service chain generator, permit the placement of VNFs onto DC servers. In terms of placement, we developed a heuristic algorithm with VNF anti-collocation constraints, which allows us to perform tests with service chains that occupy longer paths (for near-optimal service chaining embedding, see our previous work in [3]).

We use this VNF placement method in order to compute the forwarding state required for service chaining with L2 forwarding and SR. We conduct separate measurements for three different DC utilization levels (*i.e.,* 30%, 50%, and 80%). In L2 forwarding, we insert flow entries for each service chain in the switches along the chain's path. Flow insertion with SR is more complicated, especially when the chain hop-count exceeds the maximum number of hops that can encoded in the packet header (*i.e.,* using the source/destination MAC address with 6 bits per port, the maximum number of ports is 16). This is observed in chains with 8 VNFs, at which adjacent VNFs are placed in different pods. To overcome this limitation, we decompose SR paths into pathlets. When the packet reaches the end of the first pathlet, the ToR switch uses an additional flow entry that inserts the next pathlet in the packet header, so that the packet can continue its route along the chain's path. The end of a pathlet is easily identified by the TTL value.

Our forwarding state results are depicted in Figs. 6, 7, 8 for a diverse number of hosts. In Fig. 6, L2 flow entries are higher by a factor of 3 to 5, even for a low utilization level of 30% (which corresponds to 54 service chains). The margin in the state between L2 and SR increases in higher utilization levels, which imply a larger number of deployed chains. The state savings with SR are even more profound in the DCs with 1024 (Fig. 7) and 9826 hosts (Fig. 8). In the latter, L2 requires more than 200K entries in ToR and aggregation switches, which is infeasible with commodity switching hardware. Instead, SR requires a minimum amount of state in aggregation and core switches, whereas its increased state (due to path insertion entries) in ToR switches is still significantly lower than the corresponding state with L2. Consequently, the severe dataplane limitations of L2 forwarding mandate alternative routing fabrics with a significant level of forwarding state conservation, such as source routing.

V. Related Work

Source routing has recently gained significant attention, due to its simplistic and reduced-state packet forwarding. SecondNet [8] employs source routing for DC network virtualization, and in particular, relies on port switching using MPLS. Since connecting pairs of virtual machines yields smaller hop-count than service chaining, SecondNet deals with shorter paths and, hence, does not face some of the restrictions we discussed in the context of service chaining. *Jyothi et al.* [7] propose a source-routed fabric for DC networks towards improved flexibility and performance. This work discusses alternative implementations for source routing and further quantifies the throughput gains of the proposed source-routed fabric compared to traditional forwarding in leaf-spine DC

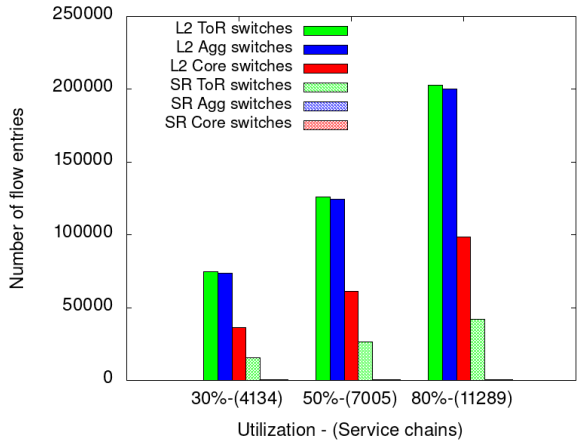

Fig. 8: Flow entries per switch type (9826 hosts).

topologies. *Hari et al.* [9] have also proposed a source routing fabric for DCs, which is in accordance with our port switching scheme and the source routing mechanisms exemplified in [7]. The work in [9] further provides an extensive analysis of the path encapsulation technique, as well as techniques for the compression of the path header.

In our previous work, [11] we present the design of a source routing fabric for long paths to meet the extensive requirements for service chaining. To achieve this goal, we couple source routing with path switching. This work lacks an implementation and experimental evaluation of source routing [11]. An implementation of a port switching scheme using Click Modular Router [16] is presented in [10]. However, the scope of this work is limited to software-based datapaths implemented with Click.

In contrast to all these aforementioned approaches, we provide a practical implementation of source routing with OpenFlow, which has very wide adoption. We further quantify the forwarding state savings with SR in NFV infrastructures, which has not been studied using a SR implementation.

VI. Conclusions

In this paper, we leveraged on source routing for increased scalability and robustness in datacenters, which will become more imminent with the prevalence of 5G network services, which are expected to generate more demand for in-network processing capabilities. In this respect, we presented in detail a practical SR implementation using OpenFlow switches, which tend to be deployed in multi-tenant DCs. Using this implementation, we uncovered a surprisingly large degree of state reduction with SR, especially in the case of service chaining. In NFV infrastructures of large scale, L2 forwarding generates a massive amount of state (*i.e.,* in excess of 100K entries with medium/high utilization levels), which yields network service deployment infeasible. In contrast, source routing with its reduced-state forwarding can be seen as an enabler for the in-network processing requirements of next-generation network services.

References

[1] K. Katsaros, V. G. P. Papadimitriou, G. Papathanail, D. Dechouniotis, and S. Papavassiliou, "Meson: Facilitating cross-slice communications for enhanced service delivery at the edge," in *European Conference on Networks and Communications*, June 2019.

[2] M. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batall, H. Koumaras, D. Dietrich, A. Ramos, J. F. Riera, J. Bonnet, A. Pietrabissa, A. Ceselli, and A. Petrini, "T-nova: An open-source mano stack for nfv infrastructures," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 586–602, Sept 2017.

[3] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, "Multi-provider service chain embedding with nestor," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 91–105, March 2017.

[4] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker, "E2: A framework for nfv applications," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15. New York, NY, USA: ACM, 2015, pp. 121–136. [Online]. Available: http://doi.acm.org/10.1145/2815400.2815423

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[6] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[7] S. A. Jyothi, M. Dong, and P. B. Godfrey, "Towards a flexible data center fabric with source routing," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. New York, NY, USA: ACM, 2015, pp. 10:1–10:8. [Online]. Available: http://doi.acm.org/10.1145/2774993.2775005

[8] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: A data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International COnference*, ser. Co-NEXT '10. New York, NY, USA: ACM, 2010, pp. 15:1–15:12. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921188

[9] A. Hari, T. V. Lakshman, and G. Wilfong, "Path switching: Reduced-state flow handling in sdn using path information," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15, 2015, pp. 36:1–36:7.

[10] A. Abujoda, H. R. Kouchaksaraei, and P. Papadimitriou, "Sdn-based source routing for scalable service chaining in datacenters," in *Wired/Wireless Internet Communications*, L. Mamatas, I. Matta, P. Papadimitriou, and Y. Koucheryavy, Eds. Cham: Springer International Publishing, 2016, pp. 66–77.

[11] C. Papagianni, P. Papadimitriou, and J. S. Baras, "Towards reduced-state service chaining with source routing," in *2018 14th International Conference on Network and Service Management (CNSM)*, Nov 2018, pp. 438–443.

[12] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for network update," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 323–334, Aug. 2012. [Online]. Available: http://doi.acm.org/10.1145/2377677.2377748

[13] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015, pp. 117–130.

[14] "Mininet," http://www.mininet.org.

[15] "D-ITG," http://www.grid.unina.it/software/ITG/.

[16] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," *SIGOPS Oper. Syst. Rev.*, vol. 33, no. 5, pp. 217–231, Dec. 1999. [Online]. Available: http://doi.acm.org/10.1145/319344.319166

European Union
European Regional Development Fund

HELLENIC REPUBLIC
MINISTRY OF ECONOMY & DEVELOPMENT
SPECIAL SECRETARY FOR ERDF & CF
MANAGING AUTHORITY OF EPAnEK

EPAnEK 2014-2020
OPERATIONAL PROGRAMME
COMPETITIVENESS ENTREPRENEURSHIP INNOVATION

ΕΣΠΑ 2014-2020
ανάπτυξη - εργασία - αλληλεγγύη
Partnership Agreement 2014 - 2020

Co-financed by Greece and the European Union