A Novel Compact Header for Traffic Steering in Service Function Chaining

Hajar Hantouti
Faculty of sciences
Moulay Ismail University
Meknes, Morocco
h.hantouti@edu.umi.ac.ma

Nabil Benamar
School of Technology
Moulay Ismail University
Meknes, Morocco
n.benamar@est.umi.ac.ma

Tarik Taleb
Aalto University
Espoo, Finland
tarik.taleb@aalto.fi

***Abstract*—Large-scale networking infrastructures such as service providers deploy complex services to deal with the growing network traffic demand, security concerns, and user preferences. Using Service Function Chaining (SFC), a set of networking and management operations, permits to steer the traffic through a list of intermediate services. Traffic steering for SFC is usually based on packet headers to share the SFC information, however such headers introduce encapsulation overhead and require service functions support. In this paper, we present a novel traffic steering technique based on a compact SFC header. The proposed header does not increase the packet size and allows network operators to deploy SFC using legacy service functions. We also present a new SDN architecture for SFC based on compact headers. Our proposal permits a scalable and a flexible SFC deployment in real-life infrastructures.**

Keywords—service function chaining; traffic steering; software-defined networking; network function virtualization

I. INTRODUCTION

Big networking infrastructures such as service providers afford complex services involving a significant number and type of service functions, A Service Function (SF) can be any networking middlebox, such as Network Address Translation, Deep Packet Inspection or firewall. Accordingly, Service Function Chaining (SFC) [1] is a set of operations that combines intermediate services to compose complex services. SFC relies on traffic steering configurations or protocols to direct traffic to/from the intermediate SFs. SFC assists network operators in customizing the composition of complex services to offer diverse customers services and optimizing infrastructure resources usage.

Current SFC deployments are usually static, relying on network configurations to stitch SFs together. One way to chain services is by configuring VLANs to steer traffic among the SFs. Recently, the new technologies of Software-Defined Networking (SDN) [2] and Network Function Virtualization (NFV) [3] permit to dynamically compose chains and make the accurate forwarding policies.

SDN is an emerging networking concept that separates the data plane from the control plane. It provides a centralized control and programming via southbound communication protocols and interfaces to the data plane networking devices. In the same way, NFV is a trending technology based on network virtualization that enables the development of network functions that can be deployed on commodity hardware. NFV, together with SDN, enhance the flexibility for implementing SFC.

SFC has gained significant interest from Academia and industry, and several SFC approaches have been proposed thus far [4]–[11]. However, the static SFC techniques and the recent dynamic SFC approaches are still not flexible enough to be deployed in real-life infrastructures [8]–[10]. On the one hand, the static SFC solutions that rely on physically stitching service functions together to compose Service Chains (SC) or configuring networks to create the service chain paths are costly, and require complex and error-prone configurations. On the other hand, new dynamic SFC solutions that use new headers such as Network Service Header (NSH) [9] involve service function and Service Function Forwarders (SFF) to support new headers. Furthermore, an additional network encapsulation overhead is produced and cannot be negligible for SFC deployment in large-scale infrastructures. Therefore, the deployment complexity in real-life infrastructures, as well as the scalability issues limit the efficiency of such SFC solutions. Indeed, the efficiency of the SFC deployment is based on the type of the traffic steering method used. Thus, the choice and the design of the accurate forwarding operations must be investigated.

The main contributions of this paper are the design of a new SFC header for encoding SFC forwarding information, and a traffic steering mechanism based on the existing packet header: source Mac address. Our solution meets the requirements of an SFC flexible solution, easily deployable without further SF/SFF support. Furthermore, the compact header described in this paper does not induce additional packet size or encapsulation overhead as compared to NSH[9].

The remainder of this paper is organized as follows. Section II presents the most related works to the approach described in this study. Section III describes the design choices of our SFC solution as well as the traffic steering method proposed, and section IV presents an evaluation of our proposal and its potential to scale. Finally, Section V concludes the paper and presents our future works.

II. Related Work

Substantial research efforts were provided to respond to the challenges of dynamic service function chaining. This section gives an insight on some recent SFC approaches.

A. SFC based on the definition of new headers

The first contributions for service function chaining began in IETF working groups (WG), mainly the service function chaining [12] and in ETSI NFV [13] WGs. Some of the proposals rely on packet headers, such as: Network Service Header (NSH) [9] and SFC IPv6 extension header (SFCEH) [14]. Particularly, NSH protocol received the most attention from academia and industry. NSH [9] [15] is a forwarding protocol for SFC; it ensures the traversal of network traffic among the SFs composing a service chain. Several SFC propositions are based on the NSH protocol. G. Li *et al.* [16] and Mehmeri et al. [17] used NSH as the traffic steering protocol. Furthermore, S. Kulkarni et al. [15] presented a modified version of the protocol to improve scalability. Despite the popularity of the NSH protocol, it worth noticing that NSH produces additional packet size and communication overhead, because of the encapsulation required for its header. Thus, this protocol adds considerable complexity in real life deployments. Moreover, it requires SFs and SFFs to support the header or to request additional intermediate proxies. For these reasons, we avoided using extra SFC headers and other tunneling techniques for the design of our proposal. Our approach is based on rewriting the source Mac address to encode an SFC compact header.

B. SFC based on the rewriting legacy packet headers

Besides the proposals discussed earlier, other SFC approaches choose to rewrite existing packet headers such as the L2 Ethernet MAC address, IP option field or IP DiffServ. Among these proposals, the works presented in [8], [18], [19]. Recently, Blendin et *al.* [19] suggested to rewrite the next SF's Mac address in the packet's destination Mac address. The association of SF address and the incoming port identify the service chain while IP addresses identify users. They choose to encode the SF address and retrieves the precise service chain. Likewise, Ding et *al.* [18] encoded the service chain identifier (SC-ID) in the packet's source Mac address and retrieve the accurate SFs. Conjointly, Abujoda *et al*. [8] encoded the forwarding path in a header that combines multiple fields: the destination MAC address, VLAN tags and MPLS label. These approaches encoded SFC information differently, however, tradeoff between sharing the necessary SFC information and encoding this information in moderate size headers must be studied. For this reason, our solution encodes the relevant SFC information in a compact header. The compact header rewritten to the Mac address field prevents additional packet size. Moreover, the proposed header allows traffic forwarding with less lookup by avoiding re-classification. By taking these factors into consideration, our solution can be easily deployed and responds to scalability and flexibility requirements.

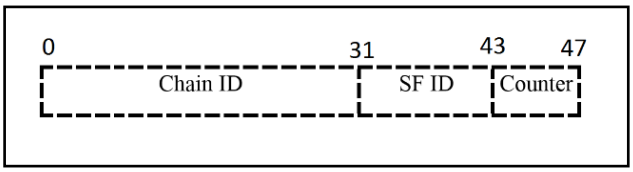

Figure 1: format of the SFC Compact header

III. The Proposed Approach Design

In this section, we present the design choices for our SFC proposal, the compact header format and the traffic steering operation.

A. Compact header proposal

The goal of Compact header proposal is to simplify SFC forwarding, reduce packet size and avoid re-classification of the traffic each time it visits an SFF. We choose to encode the SFC information in a Compact header that does not add size to packets, by re-writing the Mac source address to carry SFC forwarding information.

The mandatory SFC information, that helps SFFs to identify the traffic is: (i) the service chain identifier, (ii) the next service function in the path and (iii) a counter to position the path. The header format is depicted in Figure 1; it consists of an identifier of the service chain, the next service function that is updated after every SF by the SFF and a counter that is decremented after visiting the SF and initialized with the number of SFs that the traffic flow should traverse.

The header's chain ID field supports more than 4000 million chain identifiers, which supports fine granular policies and allows for very large-scale networks [20]. Moreover, The SF ID field scales for more than 4000 SFs. As for the counter, it serves to identify the position of the next service function and expires after visiting the last SFF, right before sending the packets to the destination. The counter field size is 4 bits and supports service chains with an order up to 15 SFs per chain.

B. Deployment

The design of SFC compact header is motivated by the need for more flexible and scalable solutions for production environments. Furthermore, the reduced overhead and complexity for deployment in real-life infrastructures are required. These requirements are considered for the design of our proposal:

- Flexibility: one of the advantages of SFC being deployed in an SDN environment is the flexibility gained from programming the configurations and pushing the

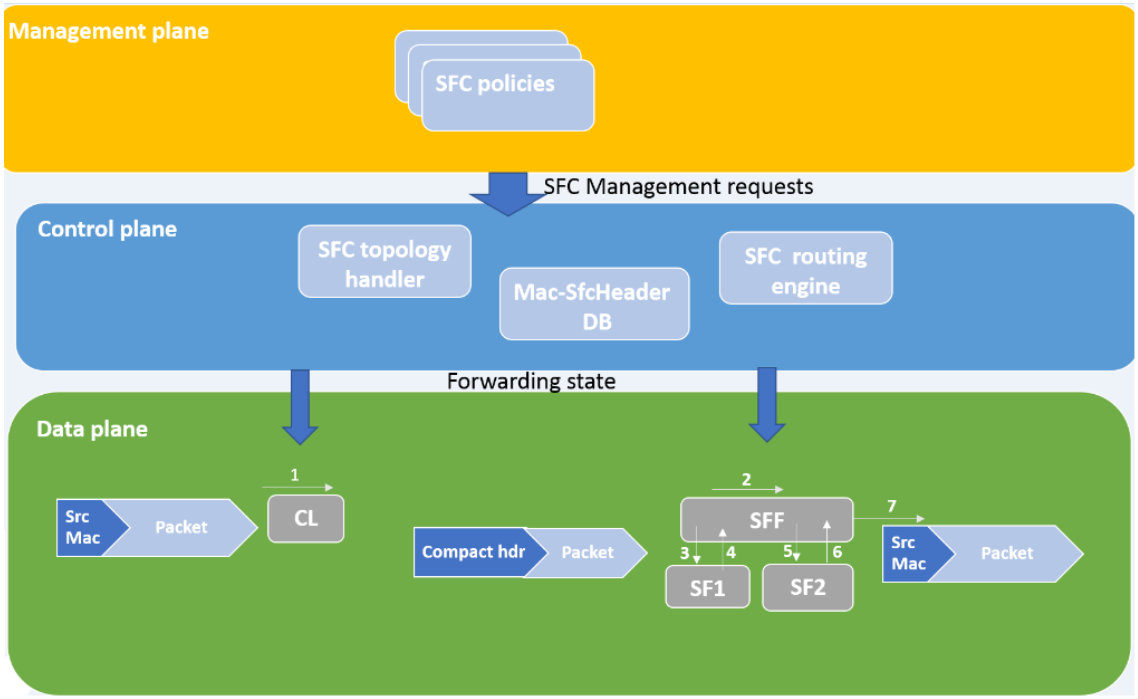

Figure 2: The proposed SDN based SFC architecture and traffic steering operation using compact header

forwarding state into the data plane elements. The forwarding state can be calculated proactively or reactively. Routes are calculated and inserted in the data plane components using southbound communication protocols such as OpenFlow [31]. Our approach can be implemented in an SDN controller to manage SFC forwarding rules in the switches, that can be software or hardware appliances.

- Scalability: A key requirement of an SFC solution is its ability to scale for medium to very large-scale networks, as the concept of SFC is more meaningful in the context of an enterprise, datacenter up to provider infrastructures. An SFC solution can scale if it supports high number of chains with acceptable overhead. The compact header proposed in this paper encodes the relevant information in Mac address to allow for simple matching. Thus, it does not add any bits to the packets, and at the same time, it provides a significant number of chains and functions, allowing fine granularity policies for an important number of users, applications or hosts.
- Overhead: among the limits of the current SFC approaches (e.g. NSH [4]) is the need for further tunneling. SFC and transport tunneling introduce overhead, mainly related to the increased packet size and encapsulation-decapsulation operations.
- Support: the compact header approach can be implemented in a controller and does not require modifications of the switches and the southbound communication protocols between the controller and the involved SFFs.

C. System architecture and traffic steering method

The architecture of our SFC proposal shown in Figure 2 is inspired by RFC 7665 [1]. It consists of a three-layered architecture composed of a management plane, control plane, and data plane. The management plane contains the northbound applications that describe the network policy and pushes requests for the control plane to translate to policy rules into forwarding rules. The forwarding state is sent to the data-plane components (mainly classifiers and service function forwarders) to ensure steering traffic to accurate service functions and final destination.

The traffic steering process is initiated by defining the policy in the management plane. The controller follows up by sending forwarding state to classifiers and SFFs. The traffic flows are first filtered by the classifier(s). The later associates the accurate SFC information to the packets by rewriting the Ethernet MAC source address. Usually, the Mac source address is not changed across the network. Deploying transparent SFs that does not modify packet headers, ensures that the SFC information follows the journey of a packet. Next, an SFC index and identifier of the first SF are inserted in the packet, as well as a counter initiated by the number of SFs in the chain.

As illustrated in Figure 2, once the classifier (CL) prepares the packets, they are sent to the accurate SFF that hosts the first SF in the service chain. Upon reception, an SFF looks at the SFC compact header and checks the SF identifier, if the SFF hosts the SF, it forwards the flow towards the accurate SF and backward, updates the next SF field and decrements the counter. In case the SFF does not host the SF, it sends it to the nearest SFF in the path. If the counter equals to 0, then the SF is the last SF in the

chain, the SFF rewrites the source Mac address to the original address and forwards the packet to the final destination based on legacy routing.

IV. SCALABILITY OF SERVICE FUNCTION CHAINING

In this section, we discuss how the use of a compact header can increase the capacity of an SFF (mainly switches) to handle a high number of SFs, chains and support chains with several SFs. First, we calculate the maximum number of chains that the switches can handle according to our compact header fields capacity while varying the number of SFs in a service chain and increasing the number of SFs connected to an SFF. Then, we discuss different factors that impact the SFF memory. For this discussion, we refer to switches within the interval of [12,48] ports.

A. *The scale of service chains*

The deployment of service function chaining is significant for medium to very-large-scale networks. Thus, it is important to consider the scalability requirements while designing an SFC header. Figure 3 illustrates the scale of service chain identifiers versus the size of the field where the identifier is encoded. Therefore, the capacity of a service chain identifier field should support a high number of chains. By considering the scalability requirement for service function chaining and as recommended in [20], a 32 bits field can encode service chain identifiers for a very large-scale network. With 2^32 combinations of service chain identifiers, compact header allows for more than 10^9 possible identifiers as shown in Figure 3. Moreover, compact header contains an 8 bits field for SF identifier which supports up to 255 SFs.

B. *The scale of service chains and service functions for an SFF*

Large-scale topologies deploy an important number of SFs that also impacts the scalability of an SFC solution. In this section, the scale of service chains and service functions for an SFF refers to the maximum number(*Max_SCs)* of chains where a switch (SW) is involved. *Max_SCs* depends on the order of service chains (*Osc)*. *Osc* is the number of SFs in a service chain, e.g., *Osc* =2 for SCs with two intermediate SFs. The order of SFs in SCs does not define the scalability of the network; it shows the complexity of chains and how strict the network policy is. Thus, large chains with a high SFC order indicate that the flows are subject to complex treatment by various SFs, which is the case for security SCs and sensitive infrastructures.

Moreover, nb_SFs refers to the number of SFs connected to a switch; it increases the number of combinations of SCs. In this paper, we choose to use the extreme case, where the SFs connected to the switch are all involved in SCs. We vary *Osc* and calculate the maximum of possible combinations of SFs in SCs, For example:

with *Osc*=2 and *nb_SFs*=10,

$Max\_SCs(Osfc) = nb\_SFs.(nb\_SFs - 1)$ =90

In general, we find that the number of possible combinations of service chains identifiers is:

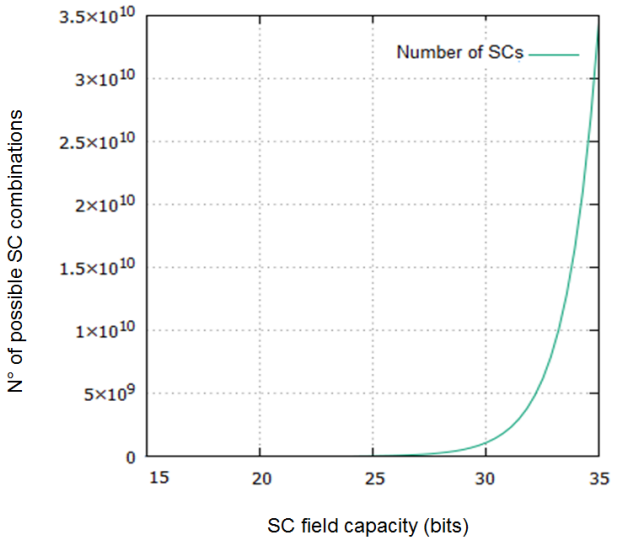

Figure 3: Compact header capacity requirements for service chain identifier versus the number of possible SC combinations

$$Max_{SCs(Osc)} = \prod_{i=0}^{i \le Osc} nb\_SFs - i \quad (1)$$

The number of combinations of SC identifiers depends on two parameters as shown in (1): *Osc* and the number of SFs connected to the switch. Table 1 shows the number of SC combinations calculated referring to (1). Therefore, we observe that by increasing the order of SCs, the number of SC combinations increases from tens to millions of chains. Similarly, while increasing the total number of SFs, the number of SCs increases to millions of SFC combinations.

C. *Impact of compact header on SFF memory*

SFFs can be either software or hardware appliances. Recently, the software switches such as Open Vswitch [21] introduced important improvement concerning the capacity to store a high number of flows because of the type of memory used, usually RAM. However, when dealing with hardware switches that use Ternary Content Addressable Memory (TCAM), the storage capacity is about few hundreds of flow rules [22]. Therefore, managing the switch memory is challenging, and strategies for reducing the required memory allocation for flow entries is a critical point for the scalability of an SFC solution.

Each flow requires a rule or a flow entry in the switch. The number of flow entries *nb_entries* required for the service chains linearly depends on the order of chains. Thus, each SF in the chain requires an entry in the flow tables. Thus, the larger the *Osc* is, the highest is the number of flows. Noting that the chains are not symmetric, as such reverse chains will require different flow entries. A reverse chain can be deployed by reversing the order of SFs and switching the flow identifiers (e.g., IP address, port number) and it requires different flow entries. The equation (2) represents the number of required flow entries for a given switch:

$$nb_{entries}(osc) = Osc \,.\, Max_{SCs} \quad (2)$$

Table II shows the required number of flow entries according to equation 2. The number of entries is per switch, assuming the extreme case where all the SFs of an SC are connected to the same switch and involving all the switch's ports. It shows that the number of flow entries increases while increasing the number of SFs and the SCs order. The number of flow entries varies from thousands to millions of rules.

The efficiency of a switch memory depends on the memory capacity as well as the management of flow entries allocation. As such the number of entries can be strengthened by reducing the amount of memory allocated per entry. The latter depends on the switch memory characteristics such as the memory blocks, banks and words [23]. It also depends on the complexity of match and action fields. OSI L2-L3 match fields require less memory than complicated matches L4-L5 or higher. Thus, by matching on simpler fields such as compact header, a switch can handle more flow entries than with complicated matches that increase the per entry memory allocation and consequently reduces the total number of entries and the scale of the switch to support a high number of flows, SCs and SFs.

Another factor that increases the flow entries is the number of hops in a given path [24]. This factor explains the increase of the number of entries in the SFC use cases, where chains with a higher order, that implies multiple SFs increase the number of chains and consequently leads to a higher number of flow entries. This result is also explained by the linearity of (1): the number of possible SCs linearly depends on the order of SCs and the total number of SFs connected to the switch.

Different ways allow to reduce the flow state and enable the flow table to support more flows and scale to larger topologies. Some solutions permits to compress the number of flow entries for an SDN Switch such as MINNIE [25]; MINNIE allows to find the possible compressions and wildcards in a flow table. Deploying less complex match and actions, using the minimum number of hops or reducing the order of necessary SFs per SC can reduce the forwarding state as well.

The allocated size of a flow entry depends on how complicated the flow rules are. It also depends on the characteristics and the type of the memory used. A flow entry is mainly composed of match headers, counters, and actions [26]. We can model the flow entry capacity as a factor of *E*: the memory allocation for an entry, *M*: the memory capacity for match headers, *A*: the memory capacity for match headers and α: the additional memory allocated for other flow entry fields. Therefore:

$$E= M+A+\alpha \quad (3)$$

Another factor that impacts the number of flow entries in a switch is the type and characteristics of each switch. Virtual switches use RAM which supports an order of millions of rules. Hardware switches support fewer rules due to the limited and restricted capacity of TCAM that only supports hundreds of rules. TCAMs are different and can have different depths (e.g. 36,72). Thus, different TCAMs may support a different number of entries. Thus, a flow entry capacity must be a multiple of TCAM widths, where *x* is the number of *TCAM_width* required for an entry.

$$E=x \, . \, TCAM\_width \quad (4)$$

TABLE I: THE NUMBER OF SERVICE CHAINS FOR DIFFERENT ORDER AND NUMBER OF SFS

N° SFs / Osc	12	24	48
2	132	552	2256
3	1320	12144	103776
4	11880	255024	4669920
5	95040	5100480	205476480

TABLE II: THE NUMBER OF FLOW RULES FOR DIFFERENT ORDER AND NUMBER OF SFS

	N° SFs =12	N° SFs =24	N° SFs =48
Osc=2	264	1104	4512
Osc=3	3960	36432	311328
Osc=4	47520	1020096	18679680
Osc=5	475200	25502400	1027382400

β is an additional overhead of empty drops of memory for each entry allocation. Assuming exact match rules where the same headers are matched (M), the same additional fields are present (α) and the same memory overhead is added (β), a flow table with capacity C is represented in (6).

$$C=\sum(M+A+\alpha+\beta) \quad (6)$$

As equation (6) is linear, each of the factors can increase or decrease the switch memory to support less or more rules, respectively.

Consequently, different factors impact the number of flow rules that a switch can store especially for hardware appliances where the flow table size is an issue. Indeed, managing flow tables can extend the switch's flow table capacity. The types of headers an SFF can match, and the resulted actions allow a switch to extend its flow table [27]. Thus, scaling to bigger infrastructures or supporting fine granular policies. For this reason, the compact header solution deploys simple OSI-L2 matches and simple modification actions. Compact header requires flow entries with regular capacity and memory allocation.

V. CONCLUSION AND FUTURE WORK

In this paper, we presented a Compact SFC header, that allows to share SFC information without increasing the packet size. Our solution encodes the compact header in the Mac source address and permits to reduce the encapsulation overhead, which makes it supported by service functions. We analyzed SFC scalability, memory constraints for SFFs and showed that our proposal does not impose memory constraints. Furthermore, we introduced a traffic steering method and SFC architecture based on compact header.

The traffic steering method described in this paper is suitable for transparent SFs, that does not update the packet headers. However, the SFC information could be lost after visiting

opaque SFs, that alter the packet headers. Our next step is to investigate on better ways to restore SFC header after visiting opaque SFs. We plan to develop an SFC routing engine in Floodlight SDN controller, to automatically translate the policies into forwarding state based on our traffic steering method.

ACKNOWLEDGEMENT

This work was partially funded by the Academy of Finland Project CSN – under Grant Agreement 311654 and also partially supported by the European Union's Horizon 2020 research and innovation program under the 5G!Pagoda project with grant agreement No. 723172.

REFERENCES

[1] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," *IETF RFC 7665*, 2015. [Online]. Available: https://tools.ietf.org/html/rfc7665.

[2] Open Networking Foundation, "SDN Architecture Overview," 2014. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf.

[3] ETSI GS NFV, "NFV-Architectural Framework," *Etsi*, 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[4] A. M. Medhat, T. Taleb, A. Elmangoush, G. A. Carella, S. Covaci, and T. Magedanz, "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges," *IEEE Commun. Mag.*, pp. 2–9, 2016.

[5] T. Taleb, S. Member, A. Ksentini, S. Member, M. Chen, and S. Member, "Coping With Emerging Mobile Social Media Applications Through Dynamic Service Function Chaining," *IEEE Trans. Wirel. Commun.*, vol. 15, no. 4, pp. 2859–2871, 2016.

[6] F. Z. Yousaf and T. Taleb, "Fine-Grained Resource-Aware Virtual Network Function Management for 5G Carrier Cloud," *IEEE Netw. Mag.*, vol. 30, no. Mars, pp. 110–115, 2016.

[7] P. Zave, X. K. Zou, and J. Rexford, "Dynamic Service Chaining with Dysco," in *SIGCOMM '17*, 2017.

[8] A. Abujoda, H. R. Kouchaksaraei, and P. Papadimitriou, "SDN-based source routing for scalable service chaining in datacenters," in *Mamatas L., Matta I., Papadimitriou P., Koucheryavy Y. (eds) Wired/Wireless Internet Communications. WWIC 2016. Lecture Notes in Computer Science*, vol. 9674, Springer, 2016, pp. 66–77.

[9] P. Quinn, "Network Service Header," *IETF-Draft*, 2017. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-nsh-28.

[10] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags," *11th USENIX Symp. Networked Syst. Des. Implement. (NSDI 2014)*, pp. 543–546, 2014.

[11] E. Datsika, A. Antonopoulos, N. Zorba, and C. Verikoukis, "Software Defined Network Service Chaining for OTT Service Providers in 5G Networks," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 124–131, 2017.

[12] IETF, "Service Function Chaining (sfc)," *IETF Working Group*. [Online]. Available: https://datatracker.ietf.org/wg/sfc/documents/.

[13] ETSI, "European Telecommunications Standards Institute." [Online]. Available: http://www.etsi.org/about/what-we-are.

[14] M. Boucadair and C. Jacquenet, "An IPv6 Extension Header for Service Function Chaining (SFC)," *IETF-Draft*, 2016. [Online]. Available: https://tools.ietf.org/html/draft-jacquenet-sfc-ipv6-eh-00.

[15] S. Kulkarni, "Neo-NSH : Towards Scalable and Efficient Dynamic Service Function Chaining of Elastic Network Functions," in *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference*, 2017, p. 5.

[16] G. Li *et al.*, "Fuzzy Theory Based Security Service Chaining for Sustainable Mobile-Edge Computing," *Mob. Inf. Syst.*, p. 13, 2017.

[17] V. Mehmeri, X. Wang, Q. Zhang, P. Palacharla, T. Ikeuchi, and I. T. Monroy, "Optical Network as a Service for Service Function Chaining across Datacenters," in *Optical Fiber Communications Conference and Exhibition (OFC)*, 2017, p. 4.

[18] W. Ding, W. Qi, J. Wang, and B. Chen, "OpenSCaaS: An open service chain as a service platform toward the integration of SDN and NFV," *IEEE Netw.*, vol. 29, no. 3, pp. 30–35, 2015.

[19] J. Blendin, J. Ruckert, N. Leymann, G. Schyguda, and D. Hausheer, "Position paper: Software-defined network service chaining," *Proc. - 2014 3rd Eur. Work. Software-Defined Networks, EWSDN 2014*, pp. 109–114, 2014.

[20] M. Boucadair and C. Jaquenet, "Service Function Chaining: Design Considerations, Analysis & Recommendations," *IETF-Draft*, 2014. [Online]. Available: https://tools.ietf.org/html/draft-boucadair-sfc-design-analysis-03.

[21] B. Pfaff *et al.*, "The Design and Implementation of Open vSwitch," *Nsdi*, pp. 117–130, 2015.

[22] A. Vishnoi, R. Poddar, V. Mann, and S. Bhattacharya, "Effective switch memory management in OpenFlow networks," *Proc. 8th ACM Int. Conf. Distrib. Event-Based Syst. - DEBS '14*, pp. 177–188, 2014.

[23] P. Bosshart *et al.*, "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," *Acm Sigcomm*, pp. 99–110, 2013.

[24] X. N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "OFFICER: A general optimization framework for OpenFlow rule allocation and endpoint policy enforcement," *Proc. - IEEE INFOCOM*, vol. 26, pp. 478–486, 2015.

[25] M. Rifai, N. Huin, and C. Caillouet, "Too many SDN rules ? Compress them with MINNIE," in *Global Communications Conference (GLOBECOM), 2015 IEEE*, 2015, pp. 0–6.

[26] B. Heller, "OpenFlow Switch Specification 1.0.0," *Open Networking Foundation*, 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[27] S. Banerjee, "Tag-In-Tag : Efficient Flow Table Management in SDN Switches," in *Network and Service Management (CNSM), 2014 10th International Conference*, 2014, pp. 109–117.