

Distributed Function Chaining with Anycast Routing

Adrien Wion, Mathieu Bouet, Luigi Iannone, Vania Conan

▶ **To cite this version:**

Adrien Wion, Mathieu Bouet, Luigi Iannone, Vania Conan. Distributed Function Chaining with Anycast Routing. Symposium on SDN research, Apr 2019, San Jose, United States. 10.1145/3314148.3314355 . hal-02163976

HAL Id: hal-02163976
https://hal.science/hal-02163976

Submitted on 24 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed Function Chaining with Anycast Routing

Adrien Wion
Thales/Telecom ParisTech

Mathieu Bouet
Thales

Luigi Iannone
Telecom ParisTech

Vania Conan
Thales

ABSTRACT

Current networks more and more rely on virtualized middleboxes to flexibly provide security, protocol optimization, and policy compliance functionalities. As such, delivering these services requires that the traffic be steered through the desired sequence of virtual appliances. Current solutions introduce a new logically centralized entity, often called orchestrator, needing to build its own holistic view of the whole network so to decide where to direct the traffic.

We advocate that such a centralized orchestration is not necessary and that, on the contrary, the same objectives can be achieved by augmenting the network layer routing so to include the notion of service and its chaining.

In this paper, we support our claim by designing such a system called NFV Router. We also present an implementation and an early evaluation, showing that we can easily steer traffic through available resources. The proposed approach offers as well valuable features such as incremental deployability, multi-domain service chaining, failure resiliency, and easy maintenance.

CCS CONCEPTS

• **Networks** → *Network architectures*; *Routing protocols*;

KEYWORDS

Service chaining; IGP; NSH; NFV; Distributed orchestration

1 INTRODUCTION

Network services used to be built as an ordered set of physically wired hardware appliances that processed traffic for security or optimization purpose. With Network Functions Virtualization (NFV), middleboxes are more and more software-based running on top of virtualization-enabled equipment, thus allowing cost reduction and network flexibility. Nevertheless, with this new paradigm, new challenges have arisen. Indeed, the service function chains are completely separated from the physical topology, and virtual functions are more ephemeral and dynamic in nature. Steering traffic through these sparsely located virtual entities, without compromising end-users' sessions and Quality of Service (QoS), is therefore a complex challenge.

Even though Internet Service Providers critically rely on middleboxes for security and policy compliance [30], most of existing NFV management solutions rely on an omnipotent logically centralized entity, generally named orchestrator. Such centralized approaches, as they require a holistic view of the whole network to perform service chaining, introduce control reactivity and resiliency (e.g., single point of failure) issues. Also, this may be quite costly for operators, since it requires the deployment of a whole new management and control infrastructure. In addition, the control part, which is meant to modify the configuration so to accommodate the orchestrator decisions, tend to be poorly interoperable with legacy appliances and is thus hard to deploy incrementally.

We believe that centralizing every orchestration decision for service function chaining is not necessary. Indeed, different decisions (e.g., in time scale or complexity) call for different control loops. Sporadic long-term configuration can afford a remote central controller. Local event-driven decisions (e.g., depending on flow arrival) gain from a distributed design. To that extent we propose to augment the network routing layer to make it service-aware. In particular, *we argue in this paper that it is possible to leverage on any Interior Gateway Protocol (IGP), anycast addressing, and any service chaining encapsulation, to construct a distributed service-aware control plane* . We propose a modular architecture that we name NFV Router (NFV-R). We show that it does not require complex elements and remains interoperable with legacy appliances. We also implemented such architecture, and early evaluation shows that our system successfully steers traffic through the intended service chain, distributing it over different instances according to available resources.

The reminder of this paper is organized as follows. First, we overview related work in Sec. 2. Then, we introduce in Sec. 3 the main concept of our proposal: namely *the service plane topology*. We detail the system architecture in Sec. 4 and the implementation in Sec. 5. Early results supporting our proposal are presented in Sec. 6, while Sec. 7 provides an agenda about research worth to be performed with respect to our proposal. Sec. 8 concludes the paper.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SOSR '19, April 3–4, 2019, San Jose, CA, USA

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6710-3/19/04…$15.00
https://doi.org/10.1145/3314148.3314355

2 RELATED WORK

So far, NFV frameworks have been built on top of centralized cloud-based management system, which has simplified the use and implementation of resource allocation algorithms [12, 16, 19, 25]. For instance, Ghaznavi et al. [17] propose a centralized VNF (Virtual Network Functions) splitting and placement algorithm. Some solutions, such as Slick [10], go further proposing a programming language to define, on a central control point, high level policies, which drive service chaining and VNF placement. Even if such centralized system provides simple and flexible VNF and path management, it increases its fragility (e.g., single point of failure, control loop delay). In addition, such a coupling between flow and VNF placement is prone to costly remapping in case of bursty traffic. Once more, these coupled decisions are taken at different timescale, hence we propose to decouple them: service chains should be steered in a fast loop so to enforce traffic engineering policies on existing VNFs, while a slower decision loop should adapt VNF provisioning to traffic changes.

Several traffic steering techniques have been proposed to tackle service chaining. Early work proposes to use fine-grained forwarding rules populated on every network appliance along a flow path [34]. This approach encounters several limitations. First, forwarding state is limited by costly memory, hardening deployment in large networks [18]. Second, forwarding rules become more complex when middleboxes modify packet headers (e.g., NAT – Network Address Translation) [14, 16, 26]. To reduce flow state on network appliances without losing path flexibility, recent works propose to leverage on waypoint routing and encapsulation. These approaches see service function chains as an ordered set of waypoints (encoded in packet headers) and rely on a distributed routing protocol so to ensure waypoint reachability [8, 18, 33]. Our proposal follows this line of thought to build robust service function chains while distributing the chaining decision. Several encapsulation formats have been proposed for service chains [8, 27, 33]. In Segment Routing v6 [8] and Dysco [33], an ingress node is in charge of setting a list of locations to reach before being delivered using IPv6 and TCP extensions respectively. Recent work at the Internet Engineering Task Force (IETF) proposes Network Service Header (NSH) as a dedicated encapsulation header for service chaining [27]. These protocols are generally used in a centralized approach.

3 DISTRIBUTED ORCHESTRATION VIA NETWORK LAYER ROUTING AUGMENTATION

While so far service function chaining has relied on a holistic centralized orchestration to steer the traffic through a sequence of virtual appliances, we believe that it can be done at the network layer routing in a distributed way.

Indeed, *any network Interior Gateway Protocol (IGP) can be directly leveraged* to convey the location, the type, and the necessary information associated to a virtual appliance and build an augmented network view.

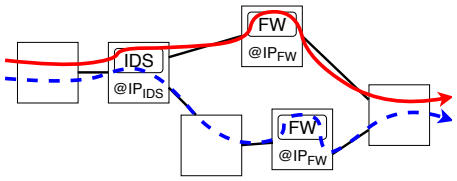
Such a view, which we call the *service plane topology*, is formed by two different types of nodes. The first type is *NFV Routers* (NFV-R): physical appliances that run the IGP and host virtual middleboxes (i.e., VNFs). These IGP-speaking machines can be normal IP routers with VNF hosting capabilities, Points of Presence, or even datacenters. The second type of node, named *virtual Service Function* (vSF), corresponds to the VNF instances themselves. vSFs can provide different types of service: Deep Packet Inspection (DPI), Firewalling, NAT, stream encoding etc. These virtual functions run on top of NFV-R. Since NFV-R, hosting vSF, also run the IGP, they can directly inject information on their vSF instances into the IGP. This way, the vSFs are present in the IGP topology too. Consequently, each NFV-R has a service plane topology that not only take chaining decisions, but may as well decide about VNF instantiation, scaling, or deletion decisions. Note that regular routers, which do not host VNFs, see NFV-R nodes as classical router and vSFs as network stubs in the IGP topology.

The main feature of vSFs instances is the service they provide. We thus propose to leverage on *anycast addressing* to announce instances providing the same service on the network. Thus, different vSF instances that are potentially hosted on different NFV-R, but that provide the same service, will be announced by the same anycast address. In this way, each specific service function (e.g., Firewall, IDS) can be mapped to a specific anycast address.[1] A link between two NFV-R represents a topological distance (network cost), while a link to a service function (i.e., the anycast address) describes some state of the vSF providing the service on the NFV-R (vSF cost). The weight that each NFV-R associate to the announced prefix, i.e. the IGP cost to reach such an address, can be based on the vSF state, its available capacities, its load, or any other relevant information.[2] A routing decision makes a tradeoff among these metrics and can be designed so as to balance the load, differentiate nodes or chains implementations (e.g., for different traffic pattern optimization), etc.

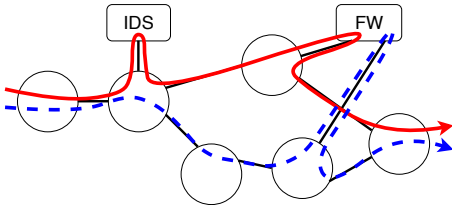
A routing decision maps vSF on NFV-R IP location, thus building a vSF routing table and ensuring that a flow always goes through the same vSF instances. New incoming flows are matched against this table to get the next(s) NFV-R(s). This decision is then cached to be consistent for all packets

[1] Note that, we are not proposing to use BGP anycast. While it would be definitely possible, this is left as future work. In this paper the focus is on an intra-domain distributed orchestration.

[2] In Sec. 7, we discuss more about how to calculate such a metric in a meaningful and rigorous way, so to guarantee loop-free routing convergence.



(a) Network topology.



(b) IGP logical view.

Figure 1: Network topology composed of 6 NFV-R, with 3 hosting a vSF instance (Fig. 1a). The IGP views the two FW instances as a single entity, since they announce the same anycast IP address (Fig. 1b). A first flow (plain red line) is routed through the IDS and the top FW instance. A second flow (dashed blue line) is routed through the IDS and the bottom FW instance as the top FW instance is already loaded with the first flow.

in a flow. Path between NFV-R could be chosen so to fulfill additional constraints. This topic is further discussed in Sec. 7

Figure 1 illustrates with a toy example the approach we propose. Figure 1a represents the network topology constituted of NFV-Rs. Each vSF instance of a given type is announced on the network with the same anycast address. In particular the two Firewall (FW) instances announce the same address: $@IP_{fw}$. Flows have to be processed here by a unique chain: $IDS + FW$. The first flow is thus routed through the IDS instance and then through the top FW instance. Indeed, in this example, this vSF instance is at one hop from the NFV-R that hosts the IDS instance. The NFV-Rs that host the used vSFs advertise their neighbors with the new experienced load or any other relevant information. When the second flow arrives, the Firewall instance at the bottom is preferred, resulting in load balancing among the FW instances (Figure 1b). Notice that in Figure 1b, since the same address is announced but no adjacency is made between the vSF (the two Firewall instances in our example), the flows that use a link to reach a service function (drawn as boxes) have to use the same link to go out of it. However, note as well that this link is only *virtual*, since it is the representation of the vSF instance in the IGP, but in reality, is running directly on a NFV-R.

Augmenting the IGP modularly allows to fully benefit from what is already done at the network layer routing. Anycast addressing leverages IGP information sharing to build the augmented topology. Based on this topology a routing decision maps vSF type to the appropriate next(s) NFV-R(s) based on network and instances metric. Finally, the IGP gives us robust IP connectivity between NFV-Rs to steer flows through the correct set of instances. Notice that the IGP protects us against flow remapping in case of link failure. Indeed, once the IGP has converged, connectivity to NFV-R is restored without any change in cached routing decisions.

As for any IGP, *high level policies* can be used to control the decision-making at each NFV-R. They are common to all the nodes and can be enforced by a central controller for easy-manageability. Such policies include flow classification rules, to map traffic to the needed service chain. High level policies also concern routing decisions since all NFV-Rs must share the same routing objectives. Based on the service plane topology, the NFV-Rs can use any path computation algorithm, to choose which instance of the next vSF of the chain the flow will go through. Additionally, high level policies can define as well how to compute vSFs' IGP costs, stating which data to use and the function to translate such data in a cost.

We use an encapsulation approach to convey the necessary information so to drive flows through the associated service chain. This information can be used to take a routing decision at the source or at every NFV-R processing the flow (hop-by-hop). This header should include *i)* part or all of the service chain identified at the classification step at the ingress of the network, *ii)* the next service step in this chain and *iii)* a consistent flow identifier to cache the routing decision. For instance, in the example in Figure 1, the NFV-R that hosts the IDS instance must have a mean to know that a packet belonging to a specific flow has been assigned to the service chain $IDS + FW$, that the next service to apply is FW, and which of the FW instances it actually has to go through.

4 SYSTEM ARCHITECTURE

In this section, we describe the architecture of our system and design its main modules. A NFV-R, as shown in Figure 2, is composed of an IP *router* providing underlay connectivity, a *connector*, attaching the router to the different vSF instances, the *vSFs* themselves, providing the services, and a *Distributed MANagement and Orchestration (D-MANO)* component, allowing local autonomous management of the node.

Router: The router provides both underlay connectivity and participate in the network IGP. It exposes a control interface used by the D-MANO to inject or remove vSF anycast addresses, announcing the services available on the node and the associated costs. This control interface is also used to get the IGP topology to build the service plane topology.

Connector: The connector allows dispatching traffic to the vSFs. It enforces chaining decisions as follows. It forwards incoming packets to the intended vSF instance, based on the

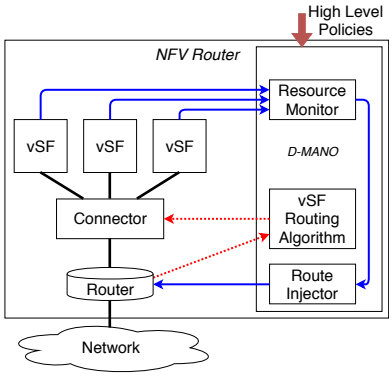


Figure 2: NFV-R architecture. Doted arrows illustrate vSF routing control flow. Solid arrows show how vSFs state is monitored, transformed in a cost, which is then injected in the IGP.

encapsulation header. Once the packets have been processed, the vSF forwards them back to the connector, which enforces a forwarding decision toward the next vSF instance location (i.e., its connector) according to the service topology. These forwarding decisions are cached in the connector, indexed by a hash computed using flow-related information, thus ensuring flow affinity. The connector also exposes a control interface, used by the D-MANO, to populate the service-aware routing table and the mapping between service function and vSF instance unicast address. This information is used by the connector to enforce chaining decisions for outgoing traffic, and locally balance the load among the vSF instances that provide the same service (same prefix).

vSF: vSF instances process flow packets according to the provided service. Once a packet has been processed, the vSF instance updates the chaining encapsulation header to point to the next service. Each instance is monitored by the D-MANO.

Distributed MANO: The D-MANO controls and manages the other NFV-R's components. It is configured with high level policies, which guide its autonomous orchestration decisions. It has two essential control functions (illustrated in Figure 2). The first one consists in monitoring vSF instances, deriving from them vSF costs, and then injecting such costs in the IGP, via the router. The second function consists in getting IGP information from the router to build the service plane topology, computing the service-aware routing table and then pushing it in the connector.

5 IMPLEMENTATION

We started to implement our proposed solution, which we describe in the present section. Furthermore, we include the technical choices we made for each component of the architecture described in the previous section.

5.1 System-Level Choices

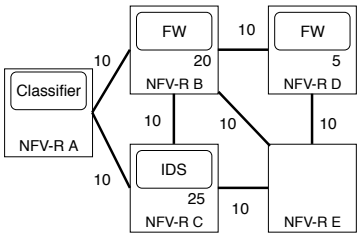
Encapsulation header: Our implementation uses the *Network Service Header (NSH)* to steering the traffic through the different services [27]. Our choice is motivated by the fact that NSH is an IETF standard explicitly designed for service chaining and is widely used in many open-source frameworks (e.g., [1, 3–5]). In NSH, the Service Path Identifier (SPI) field uniquely identifies a set of abstract service functions (i.e., the Service Function Chain), while the Service Index (SI) points to the next function the packet has to be delivered to in the SPI set. NSH also provides extensible metadata fields that we leverage to convey the hash value used to consistently identify a flow along its chain. Such hash value is computed at the classification step with the 5-tuple of the original packet.

IGP: As IGP protocol we use *Open Shortest Path First (OSPF)*, since it is widely used and easily extensible, thanks to opaque Link State Advertisements (LSA). Opaque LSAs are leveraged to share information about vSF instances and links. Even if flooding opaque LSAs increases control traffic overhead, it does not affect OSPF stability, since they do not trigger shortest path re-computation. We use *vSF opaque LSAs* to convey 3 pieces of data: *i)* the anycast address of a vSF instance, *ii)* the associated vSF cost, and *iii)* the NSH endpoint IP address (i.e., the IP address of the next Connector). In our initial implementation, we choose to use a simple vSF metric: the remaining processing capacity of the vSF instance. The NFV-Rs use the provided information to build a graph linking vSFs and NFV-Rs, each link weighted with the associated cost (Fig. 3). Thus, each NFV-R is able to build the service plane topology based on the information shared via OSPF.

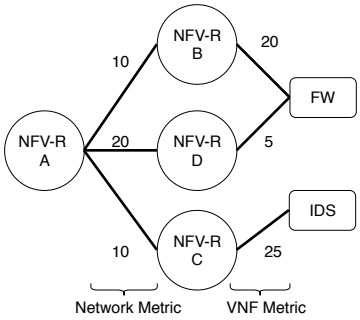
Service-aware path computation algorithm: We choose to use *Weighted Cost MultiPath (WCMP)* [35] to compute nodes' service-aware routing table. It is particularly suited for our anycast-based approach as it allows balancing the traffic based on the vSF cost. As illustrated on Figure 3, we use network link costs and vSF costs to weight the paths to a vSF anycast address. In this example, we show the service topology as seen by node A. It is easy to see that the cost to reach the *FW* instance on node B is 30 and to reach the one on node D is 25. WCMP combines network and vSF cost in order to assign weight to the differents vSF instances. This weight corresponds to the probability for a new flow to be sent to a given vSF instance. Since the vSF cost is regularly updated, WCMP adapts to the load by distributing the traffic on the lightly loaded instances (i.e., lower cost, hence, higher WCMP weight). In Sec. 7 we discuss more about the metrics.

5.2 Node-Level Choices

We build our NFV-R using Linux and use network namespaces to isolate the components.



(a) Network topology.



(b) Service plane topology seen at the NFV-R A.

Figure 3: Each NFV-R builds its service plane topology (example at Node A on Fig. (b)) with the Network costs and vSF costs so as to compute the next hop(s).

Router: In our implementation, we use *FRRouting* [2], an open source IP routing protocol suite, to implement our OSPF router. In particular, we use the OSPF API offered by FRRouting to mirror the Link-State Database (LSDB) in the D-MANO and to inject vSF opaque LSAs.

Connector: We implemented the connector logic in *P4*, a language for programming the dataplane [11]. P4 has been chosen for several reasons: it can support any header (e.g. NSH) and it is stateful (registers), allowing us to cache routing decision so to handle flow affinity. Our P4 code is run on the simple_switch target [6]. Its runtime CLI is exposed to the D-MANO to configure the switch and populate the WCMP table at runtime.

vSF: vSFs are implemented as simple processes (using *scapy* [7]) parsing incoming packets, decrement their NSH SI field, and forward them back to the connector. The focus of the initial implementation being on the different components of the proposed approach, we purposely choose simplistic vSFs for the time being. The Python *psutil* library enables us to monitor the resources used by the vSF processes.

D-MANO: The D-MANO has been implemented in Python. Its main loop runs as follows. First, it polls the resource use of the local vSF instances to build the related costs. The costs are then announced on the network with vSF opaque LSAs.

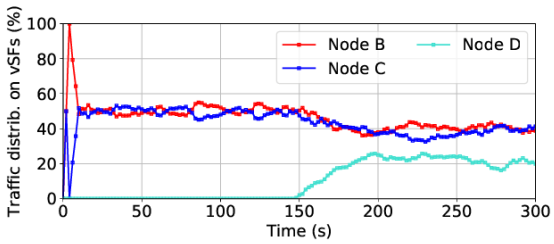


Figure 4: Traffic distribution over time on the vSF instances. During the first 150s only two vSFs are running. At $t = 150s$, a third vSF is instantiated.

Second, the D-MANO gets the vSF announces from its mirrored LSDB. With these data, it builds a service view (see Fig. 3b). Based on this topology, it computes WCMP weights and updates them on the connector.

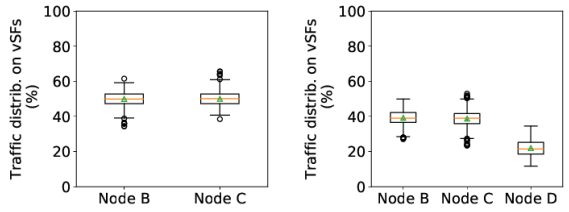
6 PRELIMINARY RESULTS

In this section we evaluate a simple scenario to show how we can achieve load balancing on different vSF instances of the same type by using the proposed solution. We selected the parameters in order to assess that our system successfully steers traffic through the target service chain. More complete evaluations are let for future work.

We consider a network topology that looks like Figure 3b, except that we use one single generic service. The link cost between NFV-R A and NFV-R B and the link cost between NFV-R A and NFV-R C are set to the same value. Link cost between NFVR-R A and NFV-R C is set to a different value so to be less preferred than the previous ones. All the vSF instances have the same initial capacity (i.e., vSF cost). It is the maximum number of packets per second a vSF instance can process, normalized so to have the same range of values as the link costs. We use Mininet to emulate this topology [21].

Traffic has to go through one of the vSF instances and then toward the egress. We generate constant bit-rate flows on a source connected to Node A. Each flow lasts 50 seconds and consumes 2 of processing units at the vSFs. The arrival rate is of two flows per second. Our scenario evolves in 2 phases. *Phase 1:* only the vSFs on NFV-R B and NFV-R C are running. *Phase 2:* After 150 seconds, a third vSF is instantiated on NFV-R D, leading to traffic redistribution.

Figure 4 presents the traffic distribution over time on the vSF instances. Since each flow lasts 50s, during the first 50s of the experiment, the system load rises until it reaches its steady state. Note that, in this example, a measure of each vSF load is measured and advertised every 2s. We can see that, during the first phase, each vSF instance receives in average the same amount of traffic. Indeed, they do have the same network cost from the ingress point of view and the same initial vSF cost. Once Phase 2 starts, after the 50 seconds of



(a) Phase 1 (50-150s). **(b) Phase 2 (200-300s).**

Figure 5: Mean traffic distribution on the vSF instances during the two phases.

transition, which lasts between $t = 150s$ and $t = 200s$, a new steady state is reached. Now the vSFs on Node B and C, each process 40% of the traffic, while the vSF on Node D roughly processes 20%. This distribution of traffic corresponds to the WCMP weights that consider links' cost and vSFs' cost.

Figure 5 presents the mean traffic distribution on the instances for the steady state of the two phases of the scenario. They result from 20 runs of the experiment. We can observe that our solution is able to balance the load among the available vSFs. The mean and median loads are centered on the values we can compute from WCMP: $50/50\%$ in Phase 1 and $40/40/20\%$ in Phase 2. In addition, 50% of the loads are less than 3 points from the median value, while the max and min values are at most 10 points from it. Such limited variation shows that the system remains quite stable.

7 RESEARCH AGENDA

Our preliminary results illustrate how service chaining can indeed be achieved by augmenting the network layer routing and applying high level policies. However, while opening interesting perspectives, it opens several questions as well. We overview them in this section.

Augmented Network Layer Routing: We have shown that we can steer traffic through a service function chain by augmenting the network routing layer. Nonetheless, finding feasible path with or without constraint is a hard problem [9]. Precomputed hop-by-hop routing decisions could follow simple and fast heuristic (shortest path to the next vSF) to steer best effort traffic. Yet, we believe that flows, which require QoS guarantees (e.g., VoIP), would be best served using the source routing paradigm (e.g., with segment routing) so to enforce on-demand optimized path. Even if our service plane topology provides support for both approaches, such hybrid scenarios and related tradeoff need further investigation.

vSF Metrics: To take service-aware routing decisions, two different types of entities are involved: network links and vSF instances. While assigning a cost to a link is straightforward (based on bandwidth, latency etc.), the *cost* of a vSF instance is an open research area. This cost may be based on a plethora of vSF state parameters [13, 24], but should also be in the same order of magnitude of the links' metric. More importantly, it has also to be additive, so to guarantee loop-free even when considering multiple constraints [23, 31].

Multi-domain SFC: In this paper, we define an augmented IGP routing logic to provide distributed SFC decisions. This is a first step towards the design of multi-domain services. Indeed, our proposal can be extended to inter-domain routing with BGP [29]. With the use of communities [22], operators could choose the information to share to build multi-domain SFC thus opening new business opportunities.

Service Chain Management: Our proposal can leverage on existing works to support vSF maintenance, failure or even chain modification. Indeed, with our approach maintenance can be easily handled through any existing loop-free graceful shutdown mechanism [15]. Furthermore, some vSF state migration use-cases can be locally dealt with on NFV-Rs [20, 28, 32]. However, service management operation involving several NFV-Rs is more challenging, since state (vSF session state, routing cache entries...) has to be coordinated. Such operations are needed when a service is modified (e.g., suspicious flows redirected to an IDS), or when a vSF is migrated to a distinct NFV-R. Existing work [33] identified challenges and possible solution to keep end-user sessions alive during reconfigurations, which can also be used.

Distributed Orchestration Decisions: NFV-R decouples traffic steering from orchestration. However, it hardens vSF resource management problem which is already difficult when decisions are taken by a centralized orchestrator [19]. Even if distributed approaches like ours improve the architecture resiliency and scalability, how autonomous nodes could take orchestration decisions (e.g., instantiating a new vSF instance to balance the global load) using on our augmented topology is another problem to be tackle.

8 CONCLUSION

In this paper, we have made the case for orchestrating service chaining in a distributed manner. We proposed to augment the network layer routing by using anycast addressing for VNF so to build what we call the service topology, allowing embedding service chaining into routing. We designed our NFV Router whose architecture is based on this concept and we implemented a first prototype. Early evaluation performed with our implementation shows that flows can be successfully driven through the chain of services according to available resources. Our approach sets itself apart from previous work, and as such it still needs to be thoroughly investigated. To this end we provide a research agenda highlighting the different aspects that need to be tackled. However, what comes out as well is quite promising and opens interesting perspectives.

REFERENCES

[1] fd.io. https://fd.io/.

[2] Frrouting. https://frrouting.org/.

[3] Onos. https://onosproject.org/.

[4] Opendaylight. https://www.opendaylight.org/.

[5] Opnfv. https://opnfv.org.

[6] P4 software switch. https://github.com/p4lang/behavioral-model.

[7] Scapy. https://github.com/secdev/scapy.

[8] A. Abdelsalam, F. Clad, C. Filsfils, S. Salsano, G. Siracusano, and L. Veltri. Implementation of virtual network function chaining through segment routing in a linux-based NFV infrastructure. In *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, 2017.

[9] S. A. Amiri, K.-T. Foerster, R. Jacob, and S. Schmid. Charting the algorithmic complexity of waypoint routing. *ACM SIGCOMM Computer Communication Review*, 48(1):42–48, 2018.

[10] B. Anwer, T. Benson, N. Feamster, and D. Levin. Programming slick network functions. In *Proceedings of the ACM SIGCOMM Symposium on Software Defined Networking Research*, page 14, 2015.

[11] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[12] A. Bremler-Barr, Y. Harchol, and D. Hay. OpenBox: a software-defined framework for developing, deploying, and managing network functions. In *Proceedings of the ACM SIGCOMM Conference*, pages 511–524, 2016.

[13] L. Cao, P. Sharma, S. Fahmy, and V. Saxena. Nfv-vital: A framework for characterizing the performance of virtual network functions. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 93–99, 2015.

[14] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, pages 19–24, 2013.

[15] P. Francois, M. Shand, and O. Bonaventure. Disruption free topology reconfiguration in OSPF networks. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, pages 89–97, 2007.

[16] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar. Stratos: A network-aware orchestration layer for middleboxes in the cloud. *CoRR*, abs/1305.0209, 2013.

[17] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba. Distributed service function chaining. *IEEE Journal on Selected Areas in Communications*, 35(11):2479–2489, 2017.

[18] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. In *ACM SIGCOMM computer communication review*, volume 45, pages 15–28. ACM, 2015.

[19] J. G. Herrera and J. F. Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[20] M. Kablan, A. Alsudais, E. Keller, and F. Le. Stateless network functions: Breaking the tight coupling of state and processing. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 97–112, 2017.

[21] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19, 2010.

[22] T. Li, R. Chandra, and P. S. Traina. BGP Communities Attribute. RFC 1997, 1996.

[23] J. J. M. Algorithms for finding paths with multiple constraints. *Networks*, 14(1):95–116.

[24] P. Naik, D. K. Shaw, and M. Vutukuru. NFVPerf: Online performance monitoring and bottleneck detection for NFV. In *Proceedings of the IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 154–160, 2016.

[25] 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 Symposium on Operating Systems Principles (SOSP)*, pages 121–136, 2015.

[26] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *Proceedings of the ACM SIGCOMM*, pages 27–38, 2013.

[27] P. Quinn, U. Elzur, and C. Pignataro. Network service header (NSH). RFC 8300, 2018.

[28] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/merge: System support for elastic execution in virtual middleboxes. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 227–240, 2013.

[29] Y. Rekhter, S. Hares, and T. Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, 2006.

[30] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proceedings of the ACM SIGCOMM Conference*, pages 13–24, 2012.

[31] Z. Wang and J. Crowcroft. Bandwidth-delay based routing algorithms. In *Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM)*, volume 3, pages 2129–2133, 1995.

[32] S. Woo, J. Sherry, S. Han, S. Moon, S. Ratnasamy, and S. Shenker. Elastic scaling of stateful network functions. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[33] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford. Dynamic service chaining with dysco. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 57–70, 2017.

[34] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, et al. Steering: A software-defined networking for inline service chaining. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

[35] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat. WCMP: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the European Conference on Computer Systems*, page 5, 2014.