Distributed Function Chaining with Anycast Routing

Adrien Wion, Mathieu Bouet, Luigi Iannone, Vania Conan

Distributed Function Chaining with Anycast Routing

Technical Report

Adrien Wion*†, Mathieu Bouet*, Luigi Iannone† and Vania Conan*
* Thales, † Telecom ParisTech
{firstname.name}@thalesgroup.com
{firstname.name}@telecom-paristech.fr

***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. We also present an implementation and an early evaluation, showing that we can easily steer traffic through available resources. This approach also offers promising features such as incremental deployability, multi-domain service chaining, failure resiliency, and easy maintenance.

I. 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 risen. 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.

Despite the fact that Internet Service Providers critically rely on middleboxes for security and policy compliance [31], most of existing NFV management solutions rely on a 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 network 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 centralized orchestration for service function chaining is not necessary. The same functionalities can be provided in a distributed way by directly augmenting the network routing layer. 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 distributed control-plane*. We propose a modular architecture, showing that we do not need complex elements and we remain 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. II. Then, we introduce in Sec. III the main concept of our proposal: namely *the service plane topology*. We detail the system architecture in Sec. IV and the implementation in Sec. V. Early results supporting our proposal are presented in Sec. VI, while Sec. VII provides an agenda about research worth to be performed with respect to our proposal. Sec. VIII concludes the paper.

II. 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 [25], [18], [15], [11]. For instance, Ghaznavi [17] proposes a centralized VNF (Virtual Network Functions) splitting and placement algorithm. Some solutions, such as Slick [9], go further proposing a programming language to define, on a central control point, high level policies. Such policies drive the decision being taken and enforced at runtime concerning chaining logic, flow forwarding, VNFs placement. Yet, such centralization, while simplifying VNF and path management, comes at the price of increased fragility (e.g., single point of failure, control loop delay, etc.) and high computation load on the central point.

While the Software-Defined Networking (SDN) paradigm helps in simplifying path management, as previously men-

tioned, it still struggles to achieve traffic steering with fine-grained forwarding rules. First, the size of forwarding state is limited by costly memory (TCAM). Second, dealing with forwarding rules installation when there are middleboxes (e.g., NAT – Network Address Translation – service) is another challenge to be tackled [13], [15], [26].

Another approach consists in using encapsulation to convey traffic along a service chain. Recent work at the IETF (Internet Engineering Task Force) proposes Network Service Header (NSH) as a dedicated encapsulation header for service chaining [27]. Segment Routing (SR) encapsulation has also been proposed to handle Service Function Chaining [8]. It is based on a source routing model to steer traffic segment to segment. Recent work has also made the case for session protocol to build service overlay [34]. Even if the approach allows dynamic chaining, it relies on extending/modifying TCP, which, in an ossified Internet, is a hard task [19].

Whether or not encapsulation is used, another main challenge in service function chaining is the coordination among flow path and middlebox state. Indeed, sometime VNF instances need to be created or reduced due to fluctuations in flow volume, migrated for resource optimization, or just recovered due to failure. Some solutions, like OpenNF [16] and Split/Merge [28], provide an open interface on middleboxes, so to allow a central coordinator to manage both forwarding and state. Dysco [34] proposes to solve these issues by consolidating forwarding and session state into middleboxes. Kablan et Al. [20] instead try to avoid such state coordination problem by splitting each VNF into a stateless processing part and a consistent back-end data store.

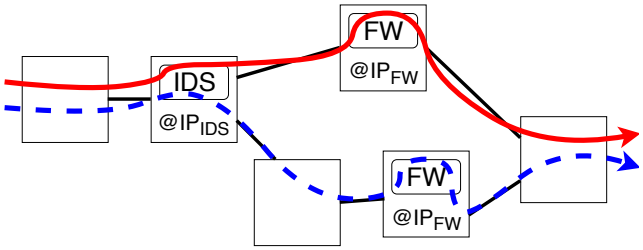
All these works strongly rely on a centralized orchestration or were only used in such context. In the rest of the paper, we make the case for orchestrating service chaining in a distributed manner.

III. Distributed Orchestration via Network Layer Routing Augmentation

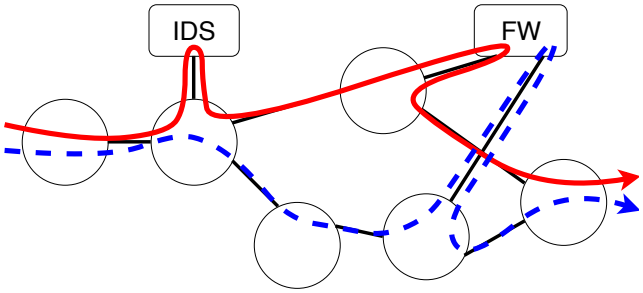
While so far service function chaining has relied on a holistic centralized orchestration to steer the traffic through sequences of virtual appliances, we believe that it can be done at the network layer routing in a distributed way.

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 nodes*. NFV nodes are physical appliances that run the IGP and host virtual middleboxes (i.e., VNFs). NFV nodes can be datacenters, Points of Presence, or routers with VNF hosting capabilities. The second type of node, named *VNF node*, corresponds to the VNF instances themselves. NFV nodes can provide different types of service: Deep Packet Inspection (DPI), Firewalling, NAT, stream encoding etc. These virtual nodes run on top of NFV nodes. Since the NFV nodes that host them run the IGP, they can directly use control packets to share information on their VNF instances. This way, the



(a) Network topology.



(b) IGP logical view.

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

VNF nodes are present in the IGP topology too. Consequently, each NFV node has a service plane view[1] to not only take chaining decisions, but also VNF instantiation, scaling, or deletion decisions.

The main feature of VNF instances is the service they provide. We thus propose to leverage on *anycast addressing* to announce service functions (i.e., VNF instances) on the network. Different VNF instances, that are potentially hosted on different NFV nodes, but that provide the same service, will be announced by the same prefix. This way, each function is mapped to a prefix. The IGP cost to reach such prefix, announced by each NFV node, can be based on the VNF state, its available capacities, its load, or any other relevant information.[2] A link between two NFV nodes represents a topological distance (network cost), while a link to a VNF node describes some state of the VNF instance (VNF cost). Thus, a routing decision makes a tradeoff among these metrics and can be designed so as to balance the load, differentiate nodes or chains, etc. By applying the routing algorithm associated to the IGP to such a service plane topology, nodes can populate their routing table, which now becomes service-aware, since it includes routes toward all the available VNF-based services. Indeed, VNF instances providing the same service share the same prefix, hence they can be discriminated by the prefix's

[1]We use the terms *service plane topology* and *service plane view* interchangeably.

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

attribute(s).

Figure 1 illustrates with a toy example the approach we propose. Figure 1a represents the network topology constituted of NFV nodes. Each VNF 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 prefix: $@IP_{fw}$. 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 VNF instance is at one hop from the NFV node that hosts the IDS instance. The NFV nodes that host the used VNFs 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 as well as dynamic path allocation for service function chains (Figure 1b). Notice that in Figure 1b, since the same prefix is announced but no adjacency is made, 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 VNF instance in the IGP, but in reality is running directly on a NFV node.

Combining this augmented IGP with anycast addressing allows to fully benefit from what is already done at the network layer routing: network layer information exchange and route computation. Indeed, the NFV nodes can be considered as classic routers that compute the next hop(s) for the best path(s), depending on the metrics, to the different prefixes, which are in our case different network services.

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

To actually drive flows through the service chain they are associated to, we need to leverage on an encapsulation approach. In both the hop-by-hop model and the source routing model the encapsulation header should provide the information necessary to steer the flows through the correct sequence of VNFs. As such, the header should include *i)* part or all of the service chain identified at the classification step at the ingress of the network and *ii)* the next service step in this chain. For instance, in the example in Figure 1, the NFV node 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.

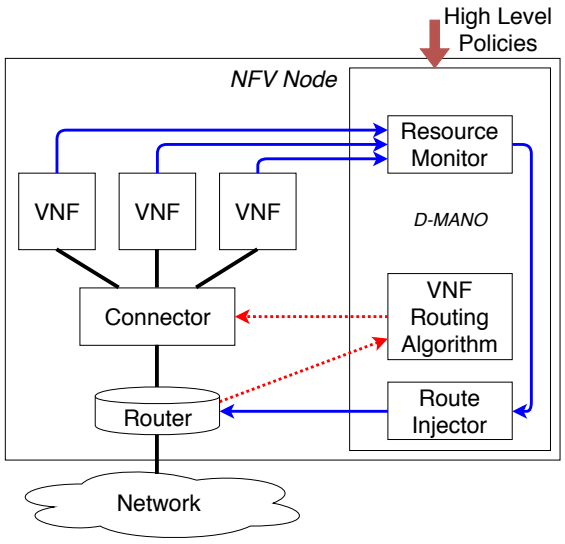


Fig. 2: NFV Node architecture. Doted arrows illustrate VNF routing control flow. Solid arrows show how VNFs state is monitored, transformed in a cost, which is then injected in the IGP.

IV. System Architecture

In this section, we describe the architecture of our system and design its main modules. A NFV node, as illustrated in Figure 2, is composed of a *router* providing underlay connectivity, a *connector*, which attaches the router to the different VNF instances, the *VNFs* themselves, providing the services, and a *Distributed MANagement and Orchestration (D-MANO)* component, which allows 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 VNF anycast prefixes, 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 VNFs. It enforces chaining decisions as follows. It forwards incoming packets to the intended VNF instance, based on the encapsulation header. Once the packets have been processed, the VNF forwards them back to the connector, which enforces a forwarding decision toward the next VNF 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. 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 VNF instance unicast address. This information is used by the connector to enforce chaining decisions for outgoing traffic, and locally balance the load among the VNF instances that provide the same service (same prefix).

VNF: VNF instances process service flow packets according to the service they provide. Once a packet has been processed, the VNF 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 node'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 VNF instances, deriving from them VNF 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.

V. Implementation

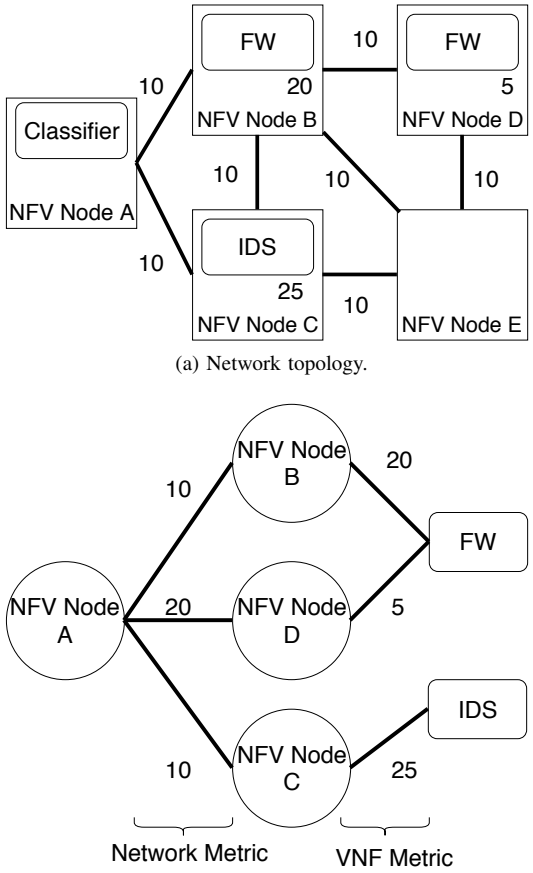
We started to implement our proposed solution, which we describe in the present section. Furthermore, we include the technical choices we made for each component of the architecture described in the previous section.

A. System-Level Choices

Encapsulation Header: Our implementation is based on the *Network Service Header (NSH)* protocol to allow steering the traffic through the different services [27]. Even if other encapsulations, such as Segment Routing v6 [8], could have been used, our choice is motivated by the fact that NSH is an IETF standard explicitly designed for service chaining and is widely used in many opensource frameworks (e.g., [5], [4], [3], [1]). 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: We build our implementation on top of an *Open Shortest Path First (OSPF)* underlay since this IGP is widely used and easily extensible thanks to opaque Link State Advertisements (LSA). Opaque LSAs are leveraged to propagate information about VNF instances and links. Even if flooding opaque LSAs increase control traffic overhead, it does not affect OSPF stability since they do not trigger shortest path algorithm computation. We thus define *VNF opaque LSAs* to convey 3 pieces of data: *i)* the anycast address of a VNF instance, *ii)* the associated VNF 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 VNF metric: the remaining processing capacity of the VNF instance. The NFV nodes use the provided information to build a graph linking VNFs and NFV nodes, each link being weighted with the associated cost (Fig. 3). Thus, each NFV node 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 VNF cost. As illustrated on Figure 3, we use network link costs and VNF costs to weight the paths to a VNF anycast prefix. 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. Such cost is used by WCMP to assign the flows on these instances. Since the VNF cost is regularly updated, WCMP adapts to the current load by distributing the traffic on the VNF instances that have the lower load (i.e., lower cost and thus higher WCMP weight). In Sec. VII we discuss more about the metrics.



(a) Network topology.

(b) Service plane view at the Node A.

Fig. 3: Each node builds its service plane view (example at Node A on Fig. (b)) with the Network costs and VNF costs so as to compute the next hop(s).

B. Node-Level Choices

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

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 VNF opaque LSAs.

Connector: We implemented the connector logic in *P4*, a language for programming the dataplane [10]. Our P4 code is run on the simple_switch target [6]. Its runtime CLI is

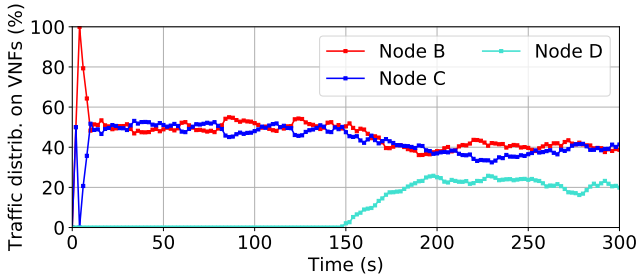
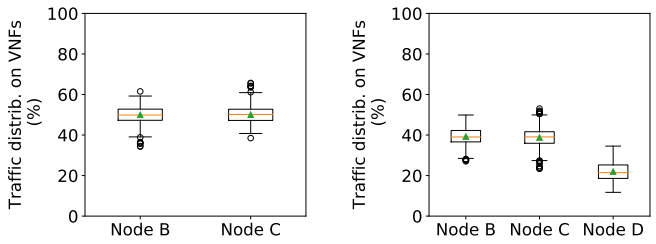


Fig. 4: Traffic distribution over time on the VNF instances. During the first 150s only two VNFs are running. At $t = 150s$, a third VNF is instantiated.



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

Fig. 5: Boxplot of the traffic distribution on the VNF instances during the two phases.

exposed to the D-MANO to configure the switch and populate the WCMP table at runtime.

VNF: VNFs 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 VNFs for the time being. The Python *psutil* library enables us to monitor the resources used by the VNF 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 VNF instances to build the related costs. The costs are then announced on the network with VNF opaque LSAs. Second, the D-MANO gets the VNF 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.

VI. Preliminary Results

In this section we evaluate a simple scenario to show how we can achieve load balancing on different VNF instances of the same type by using the proposed solution.

We consider a network topology that looks like Figure 3b, except that we use one single generic service. Moreover, the link cost between Node A and Node B and between Node A and Node C are set to 160 and between Node A and Node D it is set to 30. All the VNF instances have the same initial capacity (i.e., VNF cost) set to 150. It is the maximum number of packets per second a VNF 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 be steered through one of the VNF 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 VNFs. The arrival rate is of two flows per second. Our scenario evolves in 2 phases. *Phase 1:* only the VNFs on Node B and Node C are running. *Phase 2:* After 150 seconds, a third VNF is instantiated on at Node D, which leads to a redistribution of the traffic, since there are more VNF processing capacities now available in the service topology.

Figure 4 presents the traffic distribution over time on the VNF 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 VNF load is measured and advertised every 2s. We can see that, during the first phase, each VNF 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 VNF cost. Once Phase 2 starts, after the 50 seconds of transition, which lasts between $t = 150s$ and $t = 200s$, a new steady state is reached. Now the VNFs on Node B and C, each process 40% of the traffic, while the VNF on Node D roughly processes 20%. This distribution of traffic corresponds to the WCMP weights that consider links' cost and VNFs' 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 VNFs. 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.

VII. 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 as well a number of questions. We overview them in this section.

Traffic Engineering Constraints: Forwarding traffic in service function chains, fulfilling both Service Level Agreements and cost minimization, is a hard task. Some long-lived flows require QoS guarantees (e.g., small delay for VoIP), while short ones may suffer from an initial latency in path computation (e.g., DNS query). We believe that best effort traffic and short lived flows are best handled by precomputed hop-by-hop routing decisions. Conversely, traffic requiring resource reservation would be best served using the source routing paradigm. Our service plane topology provides support for both approaches, and enables enforcing high-level policies. However, such hybrid scenarios and related tradeoff need further investigation.

VNF Metrics: In our approach, service-aware routing involves two different types of entity: namely network links and VNF instances. While assigning a cost to a link is straightforward and normal operation (based on bandwidth, latency etc.), evaluating the *cost* of a VNF instance is an open research area. On the one hand, such a cost may be based on a plethora of VNF state parameters [12], [24]. On the other hand, the metric computation needs to be in the same order of magnitude of the links' metric, and, more importantly, it has to be additive, so to guarantee loop-free convergence even when taking into account multiple constraints [32], [23].

VNF/Resource Management: Resource allocation is already a hard problem when using a centralized approach [18]. Even if a distributed approach, like ours, improves the architecture resiliency and scalability, it adds coordination to the problem. In a distributed environment, each NFV node needs to take autonomous VNF provisioning decisions based on the exchanged information. Defining the needed information, their granularity, their update frequency, and the range of actions that each NFV node can take according to global resources availability remains to be explored. However, there is a great potential to use Machine Learning solutions that would make the network completely autonomous.

Service Modification: During a flow lifetime, the service chain it is associated to may be modified for numbers of reasons (e.g., a suspicious flow is redirected to a DPI). Such a service change implies a traffic redirection. In the source routing model, this modification could be easily handled since per flow state is concentrated at the edge. Conversely, in the hop-by-hop model, the chaining protocol should coordinate NFV nodes' state in order to modify the flow path (e.g., use Operations, Administration, and Maintenance for signaling). Existing work [34] identified challenges to keep end user sessions alive during reconfigurations. Coordination between VNF session state and routing decisions remains a challenging question to be explored.

Maintenance and Failure: To provide carrier grade network services, the impact of VNF unavailability on existing traffic should be minimized. With our approach maintenance can be easily handled through any existing loop-free graceful shutdown approach [14]. Furthermore, some VNF state migration use-cases can be dealt locally with on NFV nodes [28], [20], [33]. However, VNF migration to a remote NFV node is more challenging. Indeed, NFV state migration has to be coordinated with service topology update. Certainly existing fail-over mechanisms and make-before-break approaches can be considered, yet, the design of such mechanism is an open research area.

Security: Distributing the service chaining decision raises some inherent security questions. To state if a VNF announce is valid or not, NFV nodes should trust each other. Trustworthiness can be solved by key distribution and initial authentication. Moreover, since the chaining protocol may convey sensitive information in its header or metadata, it may be useful to use encryption between authenticated NFV nodes. For instance, we could use IPsec [30] as a transport encapsulation between NFV nodes. In general, since we are augmenting the network routing layer, without revolutionizing it, there is quite a number of existing security solutions that can be considered to provide security in the service topology.

Multi-domain SFC: Even if multi-domain SFC would open new business opportunities, service providers are reluctant to share information related with their network. We believe that a distributed design can ease multi-domain orchestration. Defining an IGP routing logic to provide distributed SFC decisions is a first step for the design of multi-domain services. Indeed, the next step to be investigated is the use of inter-domain routing, based on BGP [29] to provide chaining among different administrative entities, for instance based on the use of communities [22].

VIII. 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 an architecture based on this concept and 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] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 163–174, 2014.

[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] 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.

[19] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In *Proceedings of the ACM SIGCOMM Conference*, pages 181–194. ACM, 2011.

[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] K. Seo and S. Kent. Security Architecture for the Internet Protocol. RFC 4301, Dec. 2005.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[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.