Service Function Chaining Leveraging Segment Routing for 5G Network Slicing

Davide Borsatti, Gianluca Davoli
DEI, University of Bologna & CNIT, Italy
{davide.borsatti, gianluca.davoli}@unibo.it

Walter Cerroni, Franco Callegati
CIRI ICT, University of Bologna & CNIT, Italy
{walter.cerroni, franco.callegati}@unibo.it

***Abstract*—In this manuscript we describe an experimental work that integrates the NFV-MANO framework with segment routing to support 5G network slicing. The aim is to implement Service Function Chains spanning several cloud domains and the related interconnection transport network in a coordinated way. The manuscript shows the feasibility and the performance effectiveness of this approach, reporting numerical results from practical experiments.**

***Index Terms*—SDN, NFV, 5G, SFC, NFV-MANO, Segment Routing, IPv6.**

I. Introduction

The recent evolutionary trends in cloud and networking technologies and the increasingly relevant role played by software components in communication infrastructures are completely reshaping the way network services are designed, developed, provisioned, and managed. Network Function Virtualization (NFV) and Software Defined Networking (SDN) allow the design and implementation of complex and composite network services by concatenating one or more physical/virtualized components [1], achieving what is commonly known as Service Function Chaining (SFC) [2].

Such a flexible service provisioning matches with the 5G promise to make future mobile networks able to adapt to the specific requirements of *vertical applications*, such as autonomous driving, massive IoT, broadband multimedia communications, etc. [3]. According to the *5G network slicing* paradigm, a single network infrastructure is partitioned in virtual sub-networks tailored to the specific requirement of a vertical application, thus achieving an optimal tradeoff between cost (shared) and performance (dedicated) [4]. Slices are deployed as the sum of components running in different cloud platforms and properly interconnected by the transport network. A suitable SFC links the various components in the proper order.

An effective network slice lifecycle management is thus crucial for the success of 5G [5]. In this paper we address the slice lifecycle management issue by means of proper SFC configuration. We adopt the ETSI NFV Management and Orchestration (MANO) framework [6] as the reference standard model of operation to manage service chain components [7], as well as segment routing [8] to control how data traffic traverses service components, which promises to overcome scalability issues affecting other approaches such as OpenFlow [9]. We propose and validate an implementation of the network slice lifecycle management, integrating on the Open Source MANO (OSM) platform [10] with segment routing (SR) techniques. The main contributions of this work are:

- definition of a network scenario where an instance of OSM is in charge of network slice lifecycle management in the form of end-to-end service function chains across multiple cloud domains leveraging SR;
- definition of the required management operations that the NFV-MANO components must put in place in order to support the deployment and dynamic reconfiguration of SFCs through SR;
- pilot implementation of the proposed approach using Open Source MANO (OSM), OpenStack [11] and leveraging IPv6 segment routing (SRv6);
- experimental validation of the proposed approach on a real test bed, demonstrating the feasibility of network slice lifecycle management, also assessing the system response time.

Although previous attempts were made to implement SFCs through segment routing [12], to the best of our knowledge this is the first time that a full integration with the NFV-MANO framework is reported and experimentally demonstrated.

The remainder of the paper is structured as follows. Section II reports relevant work on NFV applied to segment routing and SFC. Then, in Section III we present the proposed network scenario for slice lifecycle management with segment routing, including the definition of the required management operations. In Section IV we describe the implementation of the proposed approach based on the OSM platform. In Section V we report the experimental validation and performance assessment. Finally in Section VI we draw some conclusions.

II. Related Work

The general segment routing architecture has been recently standardized [8], laying the basis for both the MPLS and IPv6 implementation versions. On top of that, the required data plane functionalities have been defined in order to achieve service programming [13]. Furthermore, this architecture has been extended to support network programmability using SRv6 (i.e. Segment Routing with IPv6) [14], defining the list of functions needed to enable advanced networking functionalities, such as overlay network, to support service programming. Thanks to these extensions, the segment routing

978-3-903176-24-9

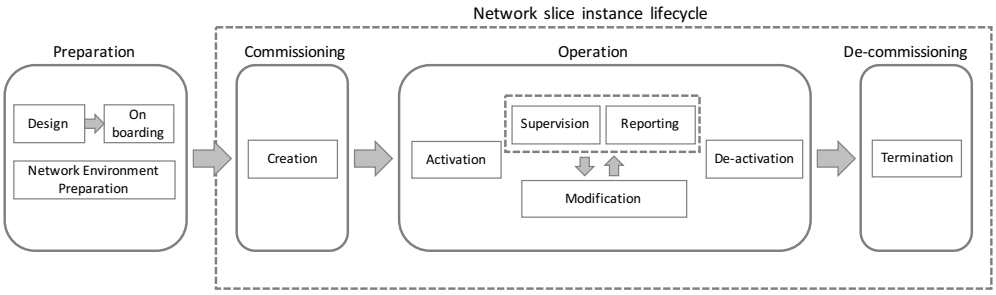

Fig. 1. Schematic of the 5G network slice lifecycle.

architecture has become one of the most relevant solutions to realize SFC.

A first solution to integrate SRv6 inside an NFV infrastructure has been proposed in [12]. However, the solution that we are presenting here goes beyond that, as we discuss the operations to be performed by the different NFV-MANO components. We also propose a possible implementation with well known open source software platforms. Several performance metrics have ben introduced regarding different southbound API solutions to configure SRv6 devices, both in terms of time required to send a command for a new SRv6 route and in terms of CPU utilization [15]. The impact of an insertion of a new steering rule on packet loss is also evaluated. An open source project that grants the possibility of deploying a Linux Node acting as an SFC proxy [2] for unaware service functions inside a segment routing enabled infrastructure has also been developed [16].

III. The Network Scenario

The lifecycle of a 5G network slice is decomposed in a number of sub-phases [5], as sketched in Fig. 1. In this work we focus on the most relevant ones: preparation, creation, activation, and modification. In this section we describe how to integrate NFV-MANO and segment routing to implement those phases. Without loss of generality, the network scenario considered here is depicted in Fig. 2. The NFV-MANO framework deploys the Virtual Network Functions (VNFs) composing the network service, possibly across different cloud domains. The VNFs are interconnected by a segment-routing-enabled network.

A. NFV-MANO Components

As per the definition given by ETSI [6], the NFV-MANO framework includes a variety of components: Virtualized Infrastructure Managers (VIMs), in charge of managing the resources offered by multiple cloud infrastructure domains; Wide Area Network Infrastructure Managers (WIMs), controlling the communication infrastructures interconnecting cloud domains; Virtualized Network Function Managers (VNFMs), in charge of the lifecycle management of the VNFs instantiated in the cloud domains; a NFV Orchestrator (NFVO), supervising the whole system, monitoring the set of resources provided

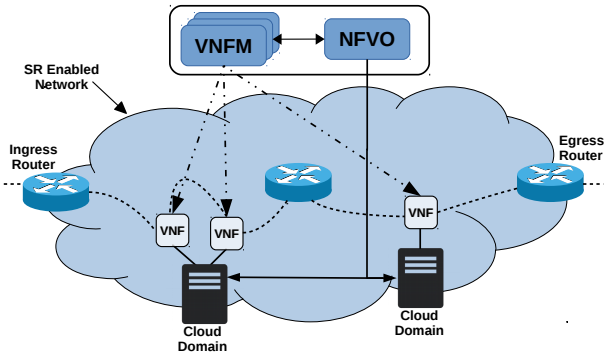

Fig. 2. Network scenario for NFV-MANO and segment routing integration.

by the underlying infrastructures, and controlling the VNF lifecycle through relevant interfaces towards VIMs, WIMs and VNFMs.

B. Segment Routing

A programmable data plane is required in order for SFC to take advantage of dynamic traffic steering and achieve adaptive service composition. To this purpose, SDN enables the use of several solutions for controlling how data traffic traverses service components, including packet tagging, additional header insertion and/or use of programmable switches [17]. Segment routing (SR) is one of such techniques, based on the well-known source routing concept, coupled with the SDN paradigm. By inserting an ordered list of Segment IDs (SIDs) identifying the VNFs to be traversed, it is possible to ensure an end-to-end connection that crosses the desired VNFs between the ingress and egress router of the network.

With reference to Fig. 2, the ingress router classifies incoming packets and inserts in their header the SIDs corresponding to the ordered list of cloud domains where the VNFs needed for the particular service are deployed. Once the packets reach a given cloud domain, all the forwarding operations inside that domain are performed by inserting an additional stack of SIDs representing the ordered list of VNFs that need to be traversed. For instance, the ingress router of the SR network may classify the flow corresponding to a given service

as <CD1,CD2,ER>, meaning that the required VNFs are hosted in "Cloud Domain 1" and "Cloud Domain 2". This does not require the knowledge of the exact SIDs associated to the single VNF, which can be kept local to each cloud domain. Then at the ingress of each cloud domain, after a local classification step, an additional stack of SIDs is added to incoming packets, containing the list of VNFs the packets have to traverse inside of the cloud domain, for instance <F1,F2>. This secondary list is then removed before the packets leave the cloud domain.

The described stacking operations require that each cloud domain is equipped with SR ingress and egress routers that work in a similar way as the ones in the SR enabled network. Several solution allow to achieve this. For example, it is possible to use a single physical router at the edge of the cloud domain acting both as ingress and egress router for all services. Alternatively, a VNF dedicated to a given service (or class of services) can be used as ingress and/or egress router. In the latter case, the operations of inserting and removing the local SIDs can be easily and dynamically controlled by the relevant VNFM. This solution can be further improved by using more than one SID for each cloud domain: reserving a SID to each tenant (or customer) of the cloud domain allows to keep their SID lists separated, enabling differentiated SFC management.

C. NFV-MANO Operations

To deploy and dynamically reconfigure the SFC within each cloud domain, the NFV-MANO components must perform a number of configurations. We describe these operations adopting the *Day 0/1/2* terminology commonly used in network automation. *Day 0* configurations are those related to the initial state of the VNF instance, including information such as the image to be used, its computing characteristics (e.g. RAM, storage, CPUs), and the initial network configuration. *Day 1* configurations include the sequence of operations to be performed immediately after launching the instance. For example, configuring additional network features, enabling system parameters, installing packages and applications setup. Finally, *Day 2* configurations relate to any additional reconfiguration made during the lifecycle of the instance.

With reference to our network scenario, *Day 0* configurations are made jointly by NFVO and VIM that select the required VNFs and their characteristics based on the service requirements. The NFVO is able to get this information from the Network Slice Template (NST) derived from the NEtwork Slice Type (NEST), which is defined according to the Service Level Agreement (SLA) required by the customer of that particular service, as expressed by GSMA [3].

Day 1 configurations instead can be performed by either the VNFM(s) or the VIM depending on the specific solution adopted. In this specific scenario it is required to enable the processing of SR packets in each VNF. An alternative *Day 1* configuration consists in the insertion of the first SIDs of an SFC that the user wants to deploy immediately.

Lastly, *Day 2* configurations are typically performed by the VNFM(s), that can add, remove or modify the list of SR functions present in each VNF, as well as act on the routing table of the node, if necessary.

D. Mapping with 5G network slice lifecycle

All the operations described before can be easily mapped inside the Network Slice Infrastructure (NSI) lifecycle [5]. Referring to Fig. 1, *Day 0* configurations are part of the *Creation* step in the *Commissioning* phase, whereas *Day 1* and *Day 2* are the *Activation* and *Modification* steps inside the *Operation* phase, respectively.

IV. Test bed Implementation

In this section we describe how we implemented the network slice lifecycle management using open-source software tool currently available, such as Open Source MANO and OpenStack. We applied our configurations on the vanilla versions of those tools, without any change to the source code.

A. OpenSource MANO

For this work we made use of Open Source MANO (OSM) [10], an ETSI-hosted open source project that allows to develop an Open Source NFV Management and Orchestration (MANO) software stack compliant to the ETSI specifications [6]. OSM consists of different functional blocks.

The most important ones, shown in Fig. 3, are the NBI (Northbound Interface), the LCM (Life Cycle Management), the RO (Resource Orchestrator) and the VCAs (VNF Configuration Adapter).

The NBI is in charge of receiving requests from the user, from either the GUI or the command line, checking whether they are compliant with the Information Model (IM) of OSM, and passing them on to the LCM. The LCM is the component that supervises the whole process of creation, management and deletion of the different network services. It takes the requests from the NBI and it interacts with the other functional blocks to serve them. The RO is the component that interacts directly with the VIMs (e.g. OpenStack, AWS or VMware vCD) requesting or freeing up the resources needed by the network service. Finally, the VCAs are the components used to perform *Day 2* configurations on the VNFs of the service. More specifically, these components are implemented leveraging JUJU proxy charms. JUJU is an open source project backed by Canonical which aims at simplifying the deployment and configuration of applications over different types of infrastructure [18]. To interact with the different components of an application, JUJU uses *charms*, which are a collection of actions (i.e., on-demand functions) employed to perform *Day 1* and *Day 2* configurations for the application. JUJU supports different types of charms, however OSM uses only one of them, the *proxy charms*. This type of charms can work both with Physical and Virtualized Network Functions. Moreover the whole charm logic runs in separate LXD containers, one for each VNF, running on the machine hosting OSM. The required commands needed for *Day 1*

and *Day 2* configurations are exchanged between the LXD container and the relative VNF through SSH.

In OSM, composition of network services is obtained by using two types of descriptors, both written in the YAML format. The first one is the VNF descriptor [19] which defines the characteristics of the virtual function, including quantity and type of interfaces, name of the image for the virtual machine, *Day 1* configuration files (e.g., cloud-init [20]) and a list of possible actions that can be launched when needed (i.e., *Day 2* configurations). The other type of descriptor is the Network Service Descriptor [21] which defines the list of VNFs composing the service, and their interconnection.

With the JUJU tools it is possible to build the proxy charm package containing all the actions a VNF needs. In addition, it is possible to define the set of commands (e.g., bash commands) that has to be launched in the VNF whenever a specific event takes place.

The actions and their parameter defined in the proxy charm must be included in the descriptor of the VNFs where we want to use them. The syntax is structured as follows:

```
vnfd:vnfd-catalog:
   vnfd:
   ...
      vnf-configuration:
         juju:
            charm: CHARM_NAME
         config-primitive:
         -  name: ACTION_NAME
            parameter:
            -  name: PARAMETER_1
               data-type: STRING
               default-value: ''
               ...
```

The `CHARM_NAME` must correspond to the one of the charm included inside the package of the VNFD. The same applies also to `ACTION_NAME` and its parameter, that must match the ones in the `actions.yaml` file of the proxy charm folder. It is important to notice that this definition is completely independent from the characteristics chosen for the VDU of the VNF. After the definition of the VNFD it is possible to use it to compose NSDs. For this particular implementation we defined a VNFD able to support all the available SRv6 functionalities [22].

B. Hardware configuration

To recreate the scenario described before, we used three bare-metal servers hosted by the CloudLab facilities [23]. Two of them were used both to host an Openstack node (Stein release via Devstack) acting as controller and compute node. The last server was used to run OSM Release SIX, an open source solution compliant with the ETSI MANO framework. The same node was also used to emulate the network interconnecting the two different cloud domains. Finally, for simplicity we placed the ingress and egress router inside Cluster1 and Cluster2 respectively. Each one of the virtual machines deployed in the Openstack clusters used a clean image of Ubuntu 18.04.2 LTS with kernel version 4.15.0-50-generic, which supports the creation of SRv6 functions using iproute2.

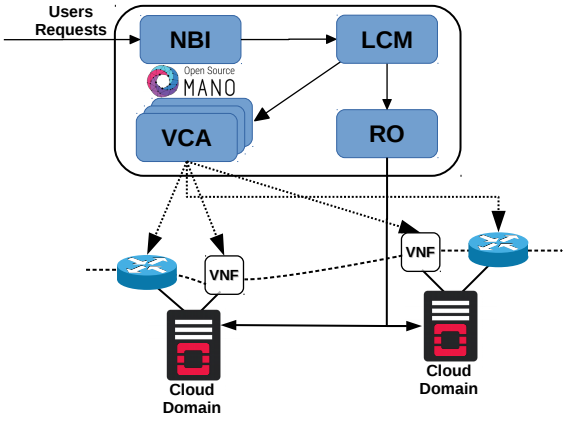

Fig. 3. Proposed Scenario.

V. Experimental Results

A. Validation

To prove the feasibility of the proposed solution, we present a simple scenario composed by two services with different priorities. Both consist of three different VMs, one acting as source of the traffic, another one as destination, and the last one as an example of a possible VNF required from the service. Recalling the scenario described in Section IV, the first two VMs can be considered as the ingress and egress point of the whole cloud domain, respectively. Our solution should also work in deployments with just one ingress and egress point shared by all services hosted by the cloud domain.

In our test scenario, the bandwidth of the service with lower priority must be reduced when the one with higher priority is instantiated. This can be achieved simply by adding to the segment list of the service with lower priority another VNF, acting as a Traffic Shaper, which can then be removed when the other service finishes transmitting its traffic. This way it is possible to validate both the setup of new SFCs and their dynamic reconfiguration. Figures 4 and 5 show the behavior in terms of bandwidth of the low priority service and of the high priority one, respectively. The former reports measures obtained from the destination of the packets (LP-D), the VNF of the service (LP-VF) and the VNF acting as a traffic shaper (LP-TS), which is active only when the higher priority service is present. The latter reports measurements obtained from the destination (HP-D) and from the VNF (HP-VF) of the higher priority service. For the first 50 seconds, the lower priority service is the only one active, therefore it is saturating all the available bandwidth. Then the higher priority service starts and the two services share the channel for about 5 seconds, which is the time needed from the system to reconfigure the segment list of the lower priority service in order to include the

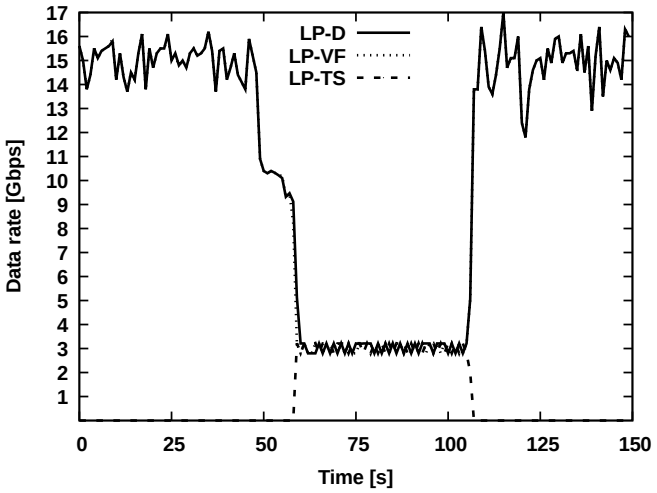

Fig. 4. Throughput achieved by the lower priority service.

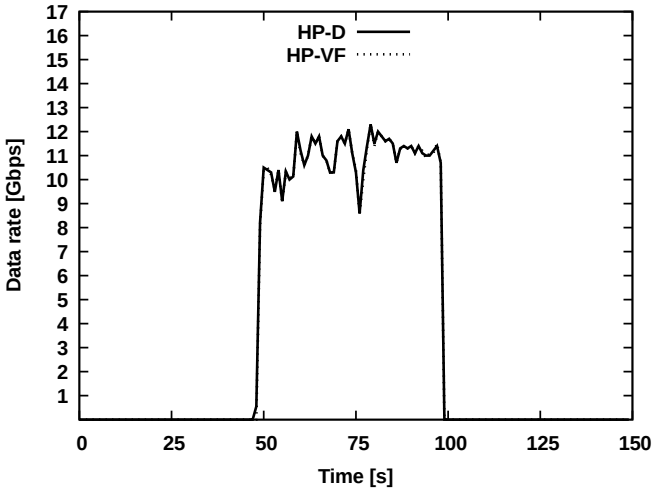

Fig. 5. Throughput achieved by the higher priority service.

Traffic Shaper, which limits the throughput to 3 Gbit/sec. A zoomed-in version of this process can be seen in Figure 6. After the end of the higher priority service, the other one can go back to using the whole available bandwidth. This is accomplished by updating the segment list of the chain again, and this also explains the delay between the end of the higher priority service and the rise in achieved throughput of the lower priority one.

B. Performance Evaluation

We evaluated the amount of time needed to deploy chains of increasing length inside a single cloud domain. It is important to recall the different steps required by OSM to setup the instances of the service and their VCA container. The LCM of OSM first instantiates each one of the Ubuntu-based containers (t_1) that will run the VCA in charge of controlling the VNF of the service. Then, it proceeds with the installation of the proxy charms components (at t_2) and the installation of an ssh key

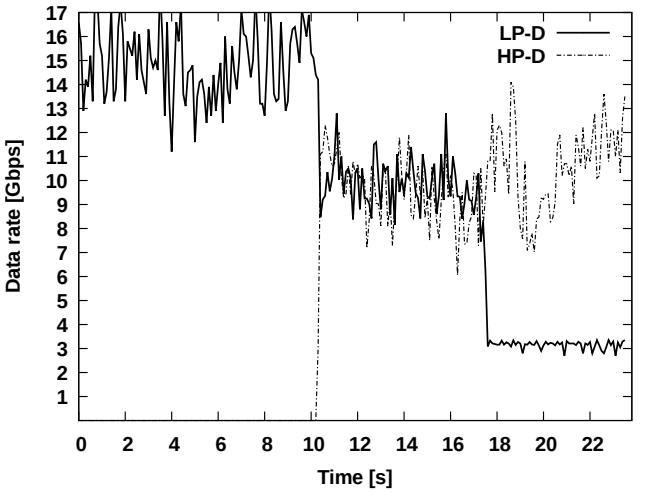

Fig. 6. Throughput achieved by the two services during the update of the segment list.

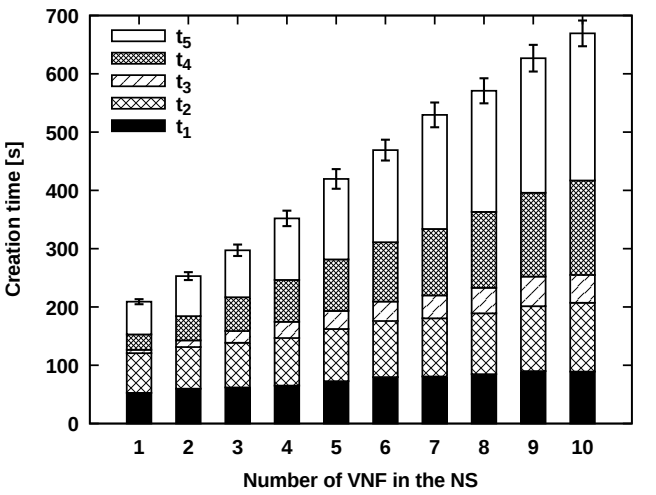

Fig. 7. Average creation time (with 95% confidence intervals) of a NS as a function of the number of VNF in it. The histogram shows the contribution of each of the five steps required to setup the NS.

inside each one of them (at t_3). Then the LCM contacts the RO (at t_4). Finally, the VCA containers obtain addresses on the management network, and the LCM verifies their reachability, the correctness of the SSH parameters defined in the VNF descriptors, and oversees the application of the desired *Day 1* configurations (at t_5).

By analysing the results we can see how much the setup of the LXC containers impacts on the time needed to deploy the overall service. However, it is important to highlight the fact that this affects only the deployment phase, as the additional configurations do not require to go through the whole process again. In fact, after the deployment, all the configurations will be made through commands launched from the VCA container to its related VNF via SSH, therefore the only factor will be the time needed by the packets traveling between the two entities.

In the test bed, the server hosting OSM and the one hosting Openstack were located inside the same data center, and the round trip time between the two was around $0.4\ ms$. Bearing this in mind, we measured the total time needed by OSM to perform an action, and the average measured time was $5.28\ s$, which is in line with the behavior shown in Fig. 4.

C. Comparison with Openstack SFC

The solution for deploying SFC natively supported by OSM leverages directly the "SFC-plugin" [24] of Neutron (the networking component of OpenStack), but with some limitations. These include the fact that OSM only allows to instantiate chains that make use of the Network Service Header (NSH) as encapsulation method, therefore lacking the support for MPLS encapsulation, and does not allow the instantiation of SFC-unaware VNFs. Contrarily, the solution proposed here can be easily expanded to support SFC-unaware VNFs, for example by creating an image, at VIM side, containing a modified Linux Kernel that supports SR-proxy [16] and by adding the actions needed to configure it to the charm. We recall from Section V-B that the time needed to deploy a service increases by adding the framework for proxy charms. By using the native solution for SFC this overhead would not be present, therefore decreasing the overall deployment time. In other words, we would notice only the time marked as t_4 in Fig. 7, plus an additional time contribution needed from OpenStack to insert the required OpenFlow rules in its internal switches. Assessments of this new contribution can be found in [25]. Nonetheless, the use of proxy charms allows to dynamically reconfigure the chains, a feature the native solution does not have, requiring the re-instantiation of the whole service. Moreover, once the proxy charm framework is up and running, it can be also used for other application-specific configurations, simply by implementing the required actions. Therefore, the adoption of proxy charms introduces an overhead in the instantiation of the service, but at the same time it grants the possibility of realizing SFP with SRv6 inside different cloud domains, making this an alternative solution to automate the creation of SFC with OSM as an orchestrator.

VI. Conclusion

This manuscripts reports an original proposal to integrate NFV-MANO and segment routing to implement Service Function Chains spanning over different technological domains, aimed at supporting the lifecycle management of 5G network slices. The Open Source Mano platform was integrated into a test bed with SRv6 (segment routing over IPv6) with the goal of implementing dynamic SFCs and the related interconnection network at once. The results presented prove the effectiveness of the proposed approach, supported by a performance evaluation over an actual test bed implementation.

Acknoledgments

This work was partially supported by CNIT through the LASH-5G project (Grant Agreement no. 732638), funded by the European Commission under the H2020 Fed4FIRE+ initiative, and by CIRI ICT, University of Bologna through the I4S project (Contract no. E91F18000250009), funded by Regione Emilia Romagna under the POR-FESR 2014-2020 program.

References

[1] F. Callegati, W. Cerroni, C. Contoli, R. Cardone, M. Nocentini, and A. Manzalini, "SDN for dynamic NFV deployment," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 89–95, Oct. 2016.
[2] J. M. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," RFC 7665, IETF, Oct. 2015.
[3] "From vertical industry requirements to network slice characteristics," The GSM Association, Tech. Rep., 2018.
[4] "An introduction to network slicing," The GSM Association, Tech. Rep., 2017.
[5] "5G; management and orchestration; concepts, use cases and requirements," 3GPP, Tech. Rep. 3GPP TS 28.530 ver. 15.1.0 rel. 15, 2019.
[6] "Network Functions Virtualisation (NFV); Management and Orchestration," The European Telecommunications Standards Institute (ETSI), 2014.
[7] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *Journal of Network and Computer Applications*, vol. 75, pp. 138–155, Nov. 2016.
[8] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, IETF, Jul. 2018.
[9] Z. N. Abdullah, I. Ahmad, and I. Hussain, "Segment routing in software defined networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 464–486, First quarter 2019.
[10] Open Source MANO. [Online]. Available: https://osm.etsi.org/
[11] OpenStack. [Online]. Available: https://www.openstack.org
[12] 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 *2017 IEEE Conference on Network Softwarization (NetSoft)*, Jul. 2017, pp. 1–5.
[13] F. Clad, X. Xu, C. Filsfils, D. Bernier, C. Li, B. Decraene, S. Ma, C. Yadlapalli, W. Henderickx, and S. Salsano, "Service Programming with Segment Routing," IETF, Internet-Draft draft-xuclad-spring-sr-service-programming-02, Apr. 2019.
[14] C. Filsfils, P. C. Garvia, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, "SRv6 Network Programming," IETF, Internet-Draft draft-ietf-spring-srv6-network-programming-01, Jul. 2019.
[15] P. L. Ventre, M. M. Tajiki, S. Salsano, and C. Filsfils, "SDN Architecture and Southbound APIs for IPv6 Segment Routing Enabled Wide Area Networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1378–1392, Dec. 2018.
[16] A. Mayer, S. Salsano, P. L. Ventre, A. Abdelsalam, L. Chiaraviglio, and C. Filsfils, "An Efficient Linux Kernel Implementation of Service Function Chaining for legacy VNFs based on IPv6 Segment Routing," in *2019 5th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, Jun. 2019, pp. 333–341.
[17] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi, "Traffic steering for service function chaining," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 487–507, First quarter 2019.
[18] JUJU. [Online]. Available: https://jaas.ai
[19] Virtual Network Function Descriptor Information Model. [Online]. Available: http://osm-download.etsi.org/repository/osm/debian/ReleaseSIX/docs/osm-im/osm_im_trees/vnfd.html#
[20] Cloud-Init Documentation. [Online]. Available: https://cloudinit.readthedocs.io/en/latest/
[21] Network Service Descriptor Information Model. [Online]. Available: http://osm-download.etsi.org/repository/osm/debian/ReleaseSIX/docs/osm-im/osm_im_trees/nsd.html#
[22] SRv6 - Linux Kernel Implementation - Advanced Configuration. [Online]. Available: https://segment-routing.org/index.php/Implementation/AdvancedConf
[23] Cloudlab. [Online]. Available: https://www.cloudlab.us
[24] Service Function Chaining Extension for OpenStack Networking. [Online]. Available: https://docs.openstack.org/networking-sfc/latest/
[25] D. Borsatti, G. Davoli, W. Cerroni, C. Contoli, and F. Callegati, "Performance of service function chaining on the OpenStack cloud platform," in *1st Workshop on Segment Routing and Service Function Chaining (SR+SFC 2018), 14th International Conference on Network and Service Management (CNSM)*, Nov 2018, pp. 432–437.