

Programmable and Automated Deployment of Tenant-Managed SDN Network Slices

M. Gharbaoui*, B. Martini*, P. Castoldi†

*CNIT, Italy ; †Scuola Superiore Sant'Anna, Pisa, Italy

Abstract—From tenant (i.e., vertical) perspective, one of the attractive feature of network slices is the possibility to use SDN solutions, i.e., network programmability, to facilitate dynamic connectivity in the slice of the application software components in a flexible and scalable way. In this paper, we present an automated deployment of tenant-managed SDN network slices that allows network operators to enhance the service offerings, and tenants to benefit of in-built SDN capabilities. The SDN slice is built as a Network Service automatically deployed by Open Source MANO as a set of VNFs dynamically configured as SDN nodes (switches and controller) connecting tenant software components. We describe how to automatically and programmatically set-up SDN capabilities in the slice through the use of Juju Charms and provide some preliminary performance results.

Index Terms—SDN; NFV; service chaining; MANO; Charms.

I. INTRODUCTION

The adoption of Network Function Virtualization (NFV) and Software-Defined Networking (SDN) technologies allows network operators to deploy network functions as software components running in virtual machines (i.e., Virtualized Network Functions (VNFs)) and to programmatically interconnect them via software network controllers [1]. This approach makes it possible to dynamically compose and orchestrate VNFs to deploy multiple end-to-end (E2E) Network Services (NSs) on top of a shared physical/virtual infrastructure (i.e., NFV Infrastructure (NFVI)) under to ETSI NFV Management and Orchestration (MANO) architectural framework [2]. The resulting NSs consist in logically independent infrastructure partitions, referred to as *network slices*, composed of virtual network, compute and storage resources tailored to serve vertical (i.e., tenant) applications with different requirements (e.g., security, end-to-end latency) [3].

Such a customized and tenant-oriented deployment of network slices in a composite NFV ecosystem involving both network operators, telco service providers and industry verticals leads to a multi-domain view that distinguishes in the first instance between infrastructure and tenant domains. According to [4], the infrastructure domain is within the NFVI and “*provides virtualized infrastructure resources such as compute, network, and storage or a composition of those resources ... and is responsible for the management and orchestration of those resources*”, whilst the tenant domain “*provides VNFs ... and is responsible for their management and orchestration, including their functional configuration and maintenance at application level*”.

If tenants are attracted by NFV/SDN-based network slices to flexibly deploy their service infrastructures as virtual assets, another attractive feature of network slices from the tenant perspective is the possibility to use SDN solutions to facilitate the delivery of connectivity among application software components (e.g., content caches, edge servers) in a flexible and scalable manner [5]. For instance, tenants may want to dynamically establish service chains among application service components. In this case, the service chaining is operated on top of a virtual network of nodes within the slice and, hence, within the tenant domain [6]. SDN can be effectively used to this purpose, with the tenant SDN controller controlling the data plane forwarding rules into SDN switches composing the virtual network established into the slice. The underlying NFVI resource settings to establish the connectivity among the VNFs to set-up the network slice in the infrastructure domain (i.e., across the NFVI-Point of Presence and/or the Wide Area Network) are under the responsibility of the infrastructure network controller, either using SDN or not [7].

In this context, if tenants willingly accepted network slices to reduce CAPEX and OPEX of their deployments, they definitely would also appreciate if SDN capabilities into the network slice are also automatically provided as part of slice deployment workflow. However, the challenge here is to combine tenant VNF deployments with proper dynamic configurations for the virtual SDN switches under the control of a tenant SDN controller.

In this paper, we present our initial efforts in implementing the automated deployment of tenant-managed SDN-enabled network slices using an ETSI NFV MANO platform (i.e., OpenSource MANO (OSM) [8]). More specifically, this work focuses on the dynamic enforcement and configuration of SDN capabilities while deploying network slices in a cloud using the OSM platform running on top of a cloud resource controller (i.e., OpenStack). The set-up reflects the most general SDN scenario in tenant domain, with the slices composed of SDN-based VNFs (deploying SDN switches and SDN controller) and general VNFs (deployed as empty VMs) connected to each others through the SDN switches. Depending on the use case, empty VMs may be either tenant applications or VNFs (e.g., access control systems). In particular, we describe how to automatically and programmatically set-up those SDN capabilities in the slice through the use of Juju Charms and provide some preliminary performance results.

II. RELATED WORKS

Recently, several research works focused on SDN usage within the NFV framework and in network slicing [9].

[10] proposes a "Bring Your Own Controller" (BYOC) paradigm, which offers an individual SDN controller to tenants to deploy their SDN apps and be able to manage a network consisting of their own VMs. To manage these controllers, the authors propose BYOC VISOR, which is a network virtualization platform tailored to IaaS clouds and allowing for customized services to tenants. The proposed work focuses on the integration of SDN with the NFVI management features instead of keeping them as separate domains. [11] proposes an integrated SDN/NFV architecture for multi-tenant networks used to deploy virtualized instances of SDN controllers instantiated as VNFs. The authors focus is on the virtualization of the control plane aiming at the dynamic management of the SDN controllers instances. They experimentally demonstrate the feasibility of virtual SDN controllers migration across heterogeneous infrastructures and show its benefits as a possible improvement of ETSI NFV-MANO.

Regarding network slicing in the tenant domain, the topic remains quite unexplored and few works tackled it. We can cite [6] where authors proposed a platform that provides tenants with latency-aware service function chaining management. The main focus of the work is the optimization of VNFs instance selection which allows for the minimization of the end-to-end latency while establishing service chains in the tenant domain. Although the approach is inspired by SDN, the set-up of the service chains does not rely on a tenant SDN controller but on a service function chaining-aware proxy exploiting Linux policy routing capabilities.

It is worth pointing out that the scope of our approach is completely different from the scope of network virtualization approaches such as FlowVisor [12]. In fact, while FlowVisor aims at dividing a physical network (composed of a set of physical switches) into multiple logical networks controlled by one or multiple SDN controllers, in this paper we aim at creating a virtual SDN network that does not depend on any hardware layer as part of an orchestrated network slice deployment on top of a virtualized infrastructure.

III. SDN CAPABILITIES IN THE TENANT DOMAIN: USE CASES

The capability to automatically deploy tenant-managed SDN network slices and configure them at run time brings important advantages in several use cases. Some possible examples are provided below.

A. Use Case 1: SDN-enabled Virtual Satellite Networks

Tenants like satellite network operator may want to exploit advantages from network slicing to deploy and operate satellite ground segment networks as a customized virtual assets. In [13] dynamic orchestration functions are presented to deploy a Virtual Satellite Networks established as a network

slice leveraging a service orchestrator and MANO-based functions. Together with others virtualized components (e.g., L4-L7 satellite network functions such as Performance Enhancing Proxy, security, FCAPS managers, and L2-L3 functions of satellite gateways), the Virtual Satellite Network will also comprise a number of (virtual) SDN switches under the control of an SDN controller. The SDN switches are used to support internal L3/L2 forwarding functions among the other (virtualized) satellite network components of the network slice, as well as for the internal or external interworking purposes, e.g., add/drop VLAN tags/ MPLS labels, enforce differentiated QoS treatments, etc. Further details can be found in [13].

This way satellite network operators can reduce both OPEX and CAPEX of their deployments and through SDN-based network programmability they can also accomplish advanced traffic control mechanisms (e.g., satellite Bandwidth on Demand [14], end-to-end Traffic Engineering [13], advanced failure recovery leveraging gateway diversity [15]) and facilitated integration of satellite networks with terrestrial networks [16], [17]. For instance, a case is presented in [13] where a mobile network (also embracing SDN) integrates an SDN-capable satellite network as part of the backhaul infrastructure so that the end-to-end path across the satellite and terrestrial components can be dynamically re-arranged at the flow level granularity upon link congestions.

B. Use Case 2: Tenant-side management of Service Function Chaining

Tenants like industry verticals or content providers can more benefit from NFV/SDN-enabled network slice deployments if they can easily establish their service delivery assets on top (e.g., cloud robotic facility in a factory, content delivery for a sport event with QoS). This is usually accomplished through dynamically-established chains of service components traversed by user traffic in a specified order based on user's requirements (e.g., quality of video) or status information (e.g., current load of users) [18] in order to best address QoE expectations. The dynamic selection of service components and context data acquisition is realized by service delivery platforms operated by the tenant typically based on Service Oriented Architecture (SOA) principles. However, SDN can be beneficial to provide the network programmability features to dynamically establish data forwarding rules into the network nodes to set-up service chain paths [19].

This way, tenants can focus on SDN settings for service delivery purposes in the domain of the application service that is delivered without the need to interwork with infrastructure elements assuming it is admitted by the infrastructure network providers. Indeed, depending on the infrastructure provider's service offering and security policy, service function chain-oriented capabilities (e.g., access to OpenStack Service Function Chaining APIs) may not be supported. In [6] a service-chaining platform is presented that is deployed in an NFVI slice and that provides tenants with latency-aware service function management (i.e., set-up and deletion) while minimizing support required by the infrastructure operators.

Although this approach is SDN-compatible, the implemented forwarding mechanisms are not natively SDN-capable. By deploying the network slice with virtual SDN switches and the relevant SDN controller, the offered network control capabilities can be fully programmable and also extended with additional features provided ad-hoc SDN applications enabling, e.g., automatic chain path adaptations based on link/switch load or upon link/switch congestion. Similarly, tenant-centric API can be also made available at the SDN controller northbound to support prescriptive service chain path requests (i.e., intents) [20].

IV. PROBLEM STATEMENT AND SYSTEM ARCHITECTURE

Based on the reference architecture defined by the European Telecommunication Standards Institute (ETSI) NFV group for the management and orchestration of NFV service [2], in Fig. 1 we present the reference architecture of the proposed SDN-enabled network slicing deployment system applying NFV concepts.

The framework consists of three main components: the NFV Management and Orchestration component (MANO), the NFV infrastructure (NFVI) and the tenant domain. MANO is composed of three functional blocks including: one or more Virtualized Infrastructure Manager (VIM)/WAN Infrastructure Manager (WIM) responsible for controlling and managing the NFVI resources; a VNF Manager (VNFM) responsible for the lifecycle management of the VNFs; and the NFV Orchestrator (NFVO) responsible for orchestrating operations for the set-up of the network slices in the form of NFV Network Services (NSs) while leveraging VIMs/WIM. The NFVI consists of the set of resources (i.e., compute, storage and networking) that are present within and across cloud domain(s) and on top of which the VNFs/NSs are created and then can run. As for network resources in the NFVI, the WIM and the VIMs can either leverage infrastructure SDN controllers or not to deploy connections in the NFVI (in the WAN and in the cloud network, respectively) [21] [22] [7].

As a result of the NSs establishments, multiple network slices can run in parallel on top of the shared NFVI for the benefit of tenants [7]. Tenants operate one or more network slice(s) to serve the application needs of the slice(s) in the tenant domain (e.g., set-up of a content delivery network to deliver media content during a special sport event). In this regards, SDN could be an attractive capability for tenants to flexibly deliver and manage the connectivity among virtual nodes in the slice with virtual nodes being either application-level VNFs (e.g., intrusion detection systems) or service components (e.g., content caches). Moreover, providing these SDN capabilities as an in-built feature of the offered slice would even be more attractive from the tenant perspective [23]. In fact, the deployment and the configuration of a subset of VNFs as the virtual Open Flow switches and as SDN controller (i.e., tenant SDN controller) are automatically managed by the NFVO during the deployment phase and as a result of the NS orchestration operations. Once the NS instance is activated, the network slice is ready to support SDN connectivity in the

tenant domain. In particular the tenant SDN controller can realize network services according to the tenant application needs (e.g., service chains among application service components).

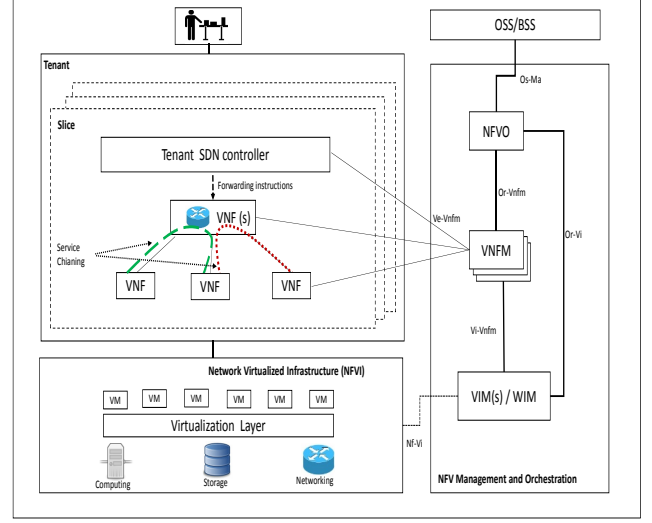


Fig. 1: System Architecture

V. SDN-ENABLED NETWORK SLICING DEPLOYMENT: SET-UP AND AUTOMATED CONFIGURATION

In this section, we detail the SDN-enabled network slicing deployment process and we present the concept of VNFs programmable configuration which represents a main challenge we face in this work.

A. System operation

In Fig. 2, we present the system that we set-up to experimentally validate our approach. We deployed two virtual machines on which we have installed OSM [8] (top right of Fig. 2) and Openstack [24] (bottom of Fig. 2). For setting up OSM, we used a VM with the following characteristics: 2 vCPUs, 8GB of RAM and 30GB for storage, with Ubuntu 18.04 as an operating system. We deployed the Release FIVE of OSM which is the most stable release to date. After having OSM up and running, we attached it to Openstack which represents the VIM in our setup. Openstack interacts with OSM through a REST API and allows for the computation, storage and network resources assignment. For the VIM, we used a second VM with the following characteristics: 3 vCPUs, 12GB of RAM and 60GB for storage, with Ubuntu 18.04 as an operating system.

OSM delivers a MANO software stack aligned with ETSI NFV framework and leverages on specific descriptors (i.e., templates) that describe the essential features of each entity of the network service to deploy, to coordinately and automatically instantiate and configure the needed resources. Details about other descriptors (e.g., VNFFGD, VLD) can be found in [25], while in the following, we give a brief description of the two main ones, i.e., the VNFD and NSD.

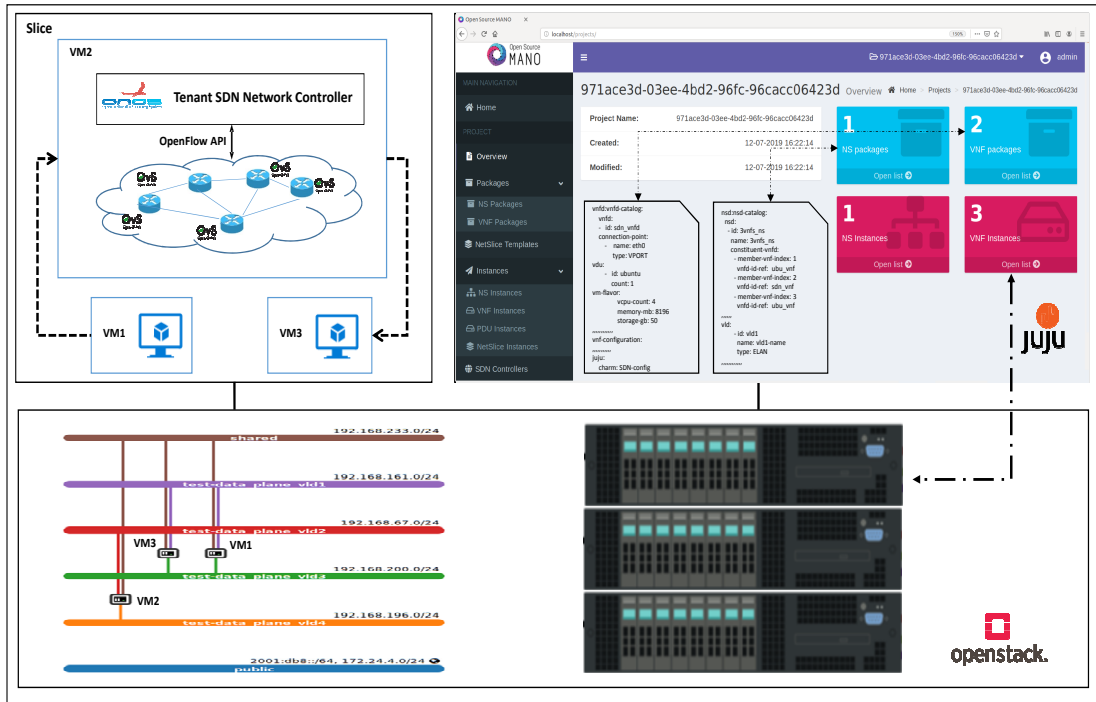


Fig. 2: Experiment Setup

- **Virtual Network Function Descriptor (VNFD):** consists in a deployment template which defines the behavior of a single VNF. The VNFD provides the following information: VNF image with all its necessary parameters (i.e., storage, memory, CPU and networking resources), connection points, Charms configuration details.
- **Network Service Descriptor (NSD):** it mainly defines the topology of the network service. It references one or more VNFD connection points (i.e., constituent VNFs identified by their unique id), lists the set of virtual interfaces, virtual links and establishes the data flow in the network.

The aim of this work is to establish a network slice with tenant-managed SDN capabilities and automatically and programmatically configure them. Accordingly, we built a NSD and VNFD that allow for a network slice to be deployed with VNFs running SDN components (i.e., an SDN controller connected to one or more OpenFlow switches) connecting other VNFs supposed to run the tenant application software modules. Since in this work we focus on providing SDN capabilities to tenant network slices whatever is the tenant application running in the slice, we disregard what those VMs might eventually run, and without lack of generality, we finally deploy simple (i.e., empty) VMs. As shown in the top right of Fig. 2, the NSD and the VNFD we prepared have been correctly managed by OSM and, the VNFs have been instantiated and automatically configured. In particular, VM2 has been configured to host the tenant SDN controller and a custom topology of OpenFlow switches as detailed below. In the next subsection, we describe how we automated and

programmed the VNFs configuration.

B. Programmable configuration

In this work, since we used OSM as the NFV MANO orchestrator, we rely on Juju as the VNFM for the configuration of the VNFs [26]. This solution brings more flexibility and simplicity since the configuration scripts can be directly brought within the VNFDs. Juju is a tool from Canonical which focuses on reducing the operation overhead of today's software by facilitating the quick deployment, configuration, scaling, and performance of operational tasks on a variety of cloud services [27]. The central mechanism behind Juju is called *Charms*. A Charm is a collection of YAML configuration files and a selection of scripts, called *hooks* that allow for installing software, starting/stopping a service, upgrading a Charm, etc. Through these files, a Charm provides all the needed information to configure a virtual instance and monitor its state.

In this work, we have considered simple VM images that once instantiated are completely configured through the Charms, which makes it easy to repeatedly deploy applications, then scale them as required with minimal effort. In OSM, the Resource Orchestrator (RO) starts by creating a virtual machine using the VM image specified in the VNF descriptor. When the VM is successfully created, the VNF Configuration and Abstraction (VCA) module, which is the OSM component in charge of the configuration of VNFs, generally aligned with the VNF manager entity of the ETSI NFV framework, presents an interface to Juju which allows for the direct communication with the VM to perform the *Day-0* and *Day-1*

configurations (both executed at the instantiation time) [26]. More specifically:

- *Day-0* configuration avoids the use of pre-built images thus allowing to save space since small standard images are considered (e.g., Ubuntu 16.04 in this work). During this phase, basic configurations are performed (e.g., SSH keys, hostname, etc.) which mainly permit the access and the communication with the VM.
- *Day-1* configuration performs the execution of the scripts defined within the Charms (i.e., initial configuration primitives) which mainly consist in downloading the necessary software packages and the execution of the desired service. At the end of this step, the VNF is fully working.

The detailed workflow of the programmable configuration is shown in the Fig. 3 where also *Day-0* and *Day-1* configurations necessary for establishing SDN capabilities in the network slice are highlighted and then further detailed.

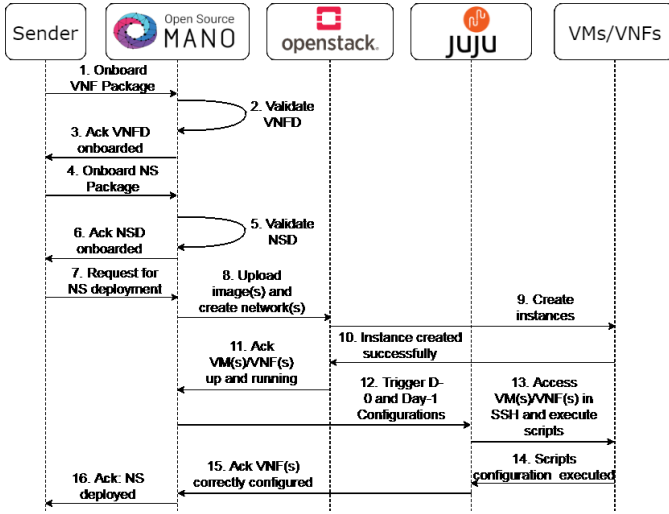


Fig. 3: Experiment Workflow

The workflow starts with the sender (i.e., OSS/BSS or any entity requiring the NS set-up on behalf of a tenant, including the tenant itself, e.g. through a web server) sending a request to the NFVO for onboarding the VNFD and NSD descriptors, detailed in the previous section. Once the descriptors are validated by OSM (i.e., check of the syntax, check of the availability of the resources (e.g., presence of VM images)), the NFVO submits the NS deployment request after selecting the target VIM (i.e., Openstack). The request is handled by the VNF Manager, which, through a REST API, directly communicates with the VIM to ask for the creation of the VMs according to the requirements specified in the descriptors. Once the VMs are instantiated, through the Juju Charms, *Day-0* and *Day-1* configurations are carried out. More specifically, since the required image in the VNFD is a simple ubuntu 16.04 image, through the scripts built in the Charms we install the necessary dependencies to execute our set-up. On the one hand, VM2 requires the installation of the the SDN controller (i.e., ONOS [28]) and the Mininet tool [29] which allows for

the creation of either a custom topology composed of a set of OpenVSwitches (OvSs) [30] or a simple OvS. On the other hand, VM1 and VM3 simply require the setup of the iperf package [31] which allows for sending/receiving traffic once the experiment is running.

With respect to the Juju charms which we adopted in this paper, another way for configuring the VNFs consists in deploying them as "empty" VNFs and then perform the configuration steps after the instantiation phase. To do so, cloud-init [32] is considered as the industry standard method for cloud instances initialization.

VI. EXPERIMENTAL EVALUATION

In this section, we present a set of preliminary results we obtained after carrying out a number of experiments to explore the performance of the presented automated deployment using Charms and its scalability properties. To this purpose, we performed successive deployments of a NS while varying the number of VNFs it contains and measuring the time required for each deployment. We repeated the cycle of the 5 deployment cases 10 times for each case.

In Fig. 4, we plot the average deployment time of the NSs while considering two different VIMs; Openstack and the VIM Emulator which is an emulation platform provided by OSM and created to support network service developers to locally test their network services in realistic scenarios [33]. Results show that the time required to deploy a NS with a single VNF is lower than 30s. By increasing the number of VNFs in the network service, the deployment time, as expected, linearly increases. Moreover, the time required for Openstack to finalize the instantiation of the VMs is significantly higher than for the VIM Emulator. This is mainly because the emulated platform is based on docker containers which are much faster than VM images.

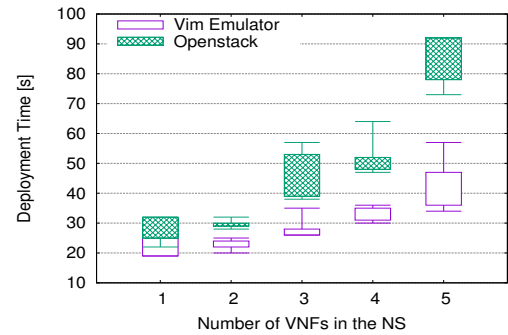


Fig. 4: Deployment time of the NS under two different VIMs

In a second experiment, we studied how, by using the Juju charms, the deployment time is affected by the automated configuration of the VNFs. For this purpose, we used the VIM emulator and we considered two different scripts. The first one is executed on only one VNF among the ones constituting the NS and allows for deploying the SDN network slice with all its components (i.e., SDN controller, Mininet topology). The second script is installed on all the other VNFs (in the case of a NS containing more than one VNF) and it aims at: (i) installing

the software necessary for the applications running on the VNF(s) (e.g., iperf package, packages necessary for sending video streamings, etc.) and (ii) performing the configuration necessary to connect the VNF(s) to the OpenFlow switches running on the SDN-based VNF.

Fig. 5 plots the obtained results that represent the time necessary for configuring all the VNFs composing the network slice. We can observe that the configuration time significantly increases when the VNFs are programmed automatically. The time also in this experiment increases linearly while augmenting the number of VNFs in the NS. This is mainly because the Charms are executed sequentially. It is worth pointing out that the plotted times also depend on the characteristics of the machine on which OSM is running (in this experiment, we used a VM with 4GB of RAM and 1 vCPU) and the Internet speed (used to download the packages necessary for the configuration) which could further slow down or improve the performance. Moreover, while the deployment of the network slice is considered completed only after the end of the configuration of all the VNFs if using the Juju charms, it is possible to move the configuration step to the initialization phase using cloud-init and then extremely decrease the deployment time of the slice (tens of seconds).

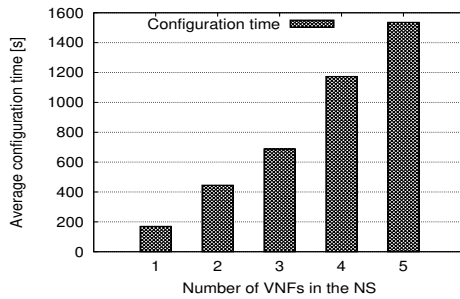


Fig. 5: Juju Charms configuration time

VII. CONCLUSIONS

In this work, we have reported the implementation of the automated deployment of SDN-capable network slices exploiting the advantages of programmable configuration of VNFs. We set-up a full MANO software stack on top of Openstack to validate the OSM descriptors and the VNFs, and to assess the performance of the approach. Preliminary results show that the overall deployment time of NSs strictly depends on the number of VNFs it contains. Results also quantify the impact of the programmability feature in terms of configuration time and confirm the overhead introduced by remote VIMs with respect to the one integrated with OSM.

ACKNOWLEDGEMENT

This work has been partially funded by the 5GINFIRE Project (grant no. 732497).

REFERENCES

[1] A. Hakiri *et al.*, “Leveraging SDN for the 5G networks: Trends, prospects and challenges,” in *preprint arXiv:1506.02876*, 2015.

[2] “Network functions virtualisation (NFV): Architectural framework,” *ETSI GS NFV*, vol. 2, no. 2, p. V1, 2013.

[3] D. Harutyunyan *et al.*, “Latency-aware service function chain placement in 5G mobile networks,” in *Network Softwarization (NetSoft), 2019 1st IEEE Conference*.

[4] “Network functions virtualisation (NFV); terminology for main concepts in NFV,” *ETSI GS NFV*, vol. 1.4.1, 2018.

[5] “Network functions virtualisation (NFV); ecosystem; report on SDN usage in NFV architectural framework,” *ETSI GS NFV*, vol. 1.1.1, 2015.

[6] G. Cuffaro *et al.*, “Tenant-side management of service function chaining: architecture, implementation and experiment on a future internet testbed,” in *Network Softwarization (NetSoft), 2019 1st IEEE Conference*.

[7] “Network functions virtualisation (NFV) release 3; evolution and ecosystem; report on network slicing support with ETSI NFV architecture framework,” *ETSI GS NFV-EVE 012*, vol. 3.1.1, 2017.

[8] <https://osm.etsi.org/>.

[9] I. Afolabi *et al.*, “Network slicing; softwarization: A survey on principles, enabling technologies; solutions,” *IEEE Communications Surveys Tutorials*, vol. PP, no. 99, 2018.

[10] H. Wang *et al.*, “Bring your own controller: Enabling tenant-defined SDN apps in iaaS clouds,” in *IEEE Conf. on Computer Communications (IEEE INFOCOM 2017)*, May 2017, pp. 1–9.

[11] R. Munoz *et al.*, “Integrated SDN/NFV management and orchestration architecture for dynamic deployment of virtual SDN control instances for virtual tenant networks [invited],” *IEEE/OSA J. Opt. Commun. Netw.*, vol. 7, no. 11, pp. B62–B70, Nov. 2015.

[12] R. Sherwood *et al.*, “Flowvisor: A network virtualization layer,” in *OpenFlow Switch Consortium, Tech. Rep 1*, 2009, p. 132.

[13] R. Ferrus *et al.*, “On the virtualization and dynamic orchestration of satellite communication services,” in *2016 IEEE 84th Vehicular Technology Conference (VTC-Fall)*, Sep. 2016, pp. 1–5.

[14] T. Ahmed *et al.*, “Towards SDN/NFV-enabled satellite ground segment systems: Bandwidth on demand use case,” in *IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017.

[15] T. Ahmed *et al.*, “Satellite gateway diversity in sdn/nfv-enabled satellite ground segment systems,” in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 882–887.

[16] L. Boero *et al.*, “Satellite networking integration in the 5g ecosystem: Research trends and open challenges,” *IEEE Network*, vol. 32, no. 5, pp. 9–15, Sep. 2018.

[17] F. Mendoza *et al.*, “Flexible capacity and traffic management for hybrid satellite-terrestrial mobile backhauling networks,” in *2016 International Symposium on Wireless Communication Systems (ISWCS)*, Sep. 2016.

[18] F. Paganelli, M. Ulema, and B. Martini, “Context-aware service composition and delivery in ngsons over sdn,” *IEEE Communications Magazine*, vol. 52, no. 8, pp. 97–105, 2014.

[19] B. Martini and F. Paganelli, “A service-oriented approach for dynamic chaining of virtual network functions over multi-provider software-defined networks,” *Future Internet*, vol. 8, no. 2, p. 24, 2016.

[20] M. Gharbaoui *et al.*, “Experimenting latency-aware and reliable service chaining in next generation internet testbed facility,” in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2018, pp. 1–4.

[21] “Network functions virtualisation (NFV); management and orchestration; or-vi reference point - interface and information model specification,” *ETSI, GS NFV-IFA 005*, vol. 2.1.1, 2016.

[22] “Network functions virtualisation (NFV) release 3; management and orchestration; report on management and connectivity for multi-site services,” *ETSI, GR NFV-IFA 022*, vol. 3.1.1, 2018.

[23] “Network functions virtualisation (NFV) release 3; evolution and ecosystem; report on network slicing support with ETSI NFV architecture framework,” *ETSI, GR NFV-EVE 012*, vol. 3.1.1, 2017.

[24] <https://www.openstack.org>.

[25] “Network functions virtualisation (NFV) release 2; protocols and data models; NFV descriptors based on TOSCA specification,” *ETSI GS NFV*, vol. 2.5.1, 2018.

[26] “Juju,” <https://jujucharms.com/>.

[27] [https://en.wikipedia.org/wiki/Juju_\(software\)](https://en.wikipedia.org/wiki/Juju_(software)).

[28] “Onos project official site,” <https://onosproject.org/>.

[29] “Mininet official site,” <http://mininet.org/>.

[30] “Open vswitch,” <https://www.openvswitch.org/>.

[31] <https://iperf.fr/>.

[32] <https://cloudinit.readthedocs.io/en/latest/>.

[33] https://osm.etsi.org/wikipub/index.php/VIM_emulator.