Hindawi
Security and Communication Networks
Volume 2019, Article ID 2425983, 11 pages
https://doi.org/10.1155/2019/2425983

WILEY | Hindawi

Research Article

Towards an Efficient Management and Orchestration Framework for Virtual Network Security Functions

Ignazio Pedone, Antonio Lioy, and Fulvio Valenza

Politecnico di Torino, Dip. Automatica e Informatica, Torino, Italy

Correspondence should be addressed to Ignazio Pedone; ignazio.pedone@polito.it

Received 9 May 2019; Accepted 20 September 2019; Published 12 November 2019

Academic Editor: Prosanta Gope

The recent years have witnessed a growth in the number of users connected to computer networks, due mainly to megatrends such as Internet of Things (IoT), Industry 4.0, and Smart Grids. Simultaneously, service providers started offering vertical services related to a specific business case (e.g., automotive, banking, and e-health) requiring more and more scalability and flexibility for the infrastructures and their management. NFV and SDN technologies are a clear way forward to address these challenges even though they are still in their early stages. Security plays a central role in this scenario, mainly because it must follow the rapid evolution of computer networks and the growing number of devices. The main issue is to protect the end-user from the increasing threats, and for this reason, we propose in this paper a security framework compliant to the Security-as-a-Service paradigm. In order to implement this framework, we leverage NFV and SDN technologies, using a user-centered approach. This allows to customize the security service starting from user preferences. Another goal of our work is to highlight the main relevant challenges encountered in the design and implementation of our solution. In particular, we demonstrate how significant is to choose an efficient way to configure the Virtual Network Security Functions in terms of performance. Furthermore, we also address the nontrivial problem of Service Function Chaining in an NFV MANO platform and we show what are the main challenges with respect to this problem.

1. Introduction

Cybercrime has grown faster in the past years, attacks are evolving rapidly, and organizations are forced to continuously update their cybersecurity and cyberdefence techniques. As reported in [1], in between 2014 and 2016, cyberattacks have evolved in two directions: growing number of threats and impact which the attack has on its target. The Verizon report explicitly describes [2], nowadays, cyberattacks are very tangible threats. Actually, about 76% of breaches were financially motivated (i.e., aimed at stealing secrets, intellectual properties, personal data, and sensitive information).

The *Internet Service Providers (ISPs)* that give connectivity to those devices are challenged by this scenario and need to provide more and more flexible, scalable, and customizable security solutions to their clients for protecting their data and privacy.

To reduce risks, various cybersecurity components are usually adopted, such as antimalware, firewall, virtual private network (VPN), and parental control functions. Several ISPs have started to offer their customers security controls implemented at the ISP premises (e.g., data centres). Of course, providing security services to hundreds of thousands or even millions of users is a challenging task which shows several limitations of traditional network technologies. *Network Function Virtualisation (NFV)* and *Software-Defined Networking (SDN)* are technologies that grant to the ISPs a reliable and scalable solution to address the end-user security problem.

NFV exploits a virtual infrastructure where network and security functions are implemented by software components called *Virtual Network Functions* (VNFs). NFV decouples software and hardware. For instance, VNFs can run in either virtual machines (VMs) or software containers, hosted on

standard high-volume servers (or ad hoc hardware), so that they can be easily deployed and removed on demand.

On another hand, the SDN paradigm provides a networking architecture where the control and forwarding planes are separated, and control functions are directly programmable. This migration of traffic control, from traditionally embedded and hard-wired individual network devices to programmable computing equipment, enables the abstraction of the communication infrastructure so that applications and services can be designed, implemented, and deployed by considering the network as a logical entity. In summary, SDN introduces unprecedented flexibility in the network, in particular by allowing dynamic fine-grained selection of arbitrary traffic flows that can be (re)routed through different network paths according to the control snap decisions when they are needed. These features can be leveraged to provide each user with the required security services; for instance, traffic flows concerning different users can be dynamically directed to different sets of security controls.

Even if these appealing technologies offer significant advantages in terms of flexibility, they also introduce new challenges such as the correct configuration of the network security functions [3].

Indeed, the impact of the "human error factor" on continuously growing networks is certainly not negligible, as several studies have proved. Nearly 60% of the security breaches, which occurred in 2015, originated from errors made by system and network administrators [4].

A new approach among ISPs is rising: the *Security-as-a-Service* (*SECaaS*) [5] paradigm. This allows to relief the user from complex security configurations and grants an always updated security service.

We started our work from [6], in which a complete user-centric framework was proposed to provide the personal security of a user. We applied this approach to an NFV Infrastructure and implemented an evolution of that framework. In the process, we addressed the following three main problems: the integration of a policy manager in an NFV environment, the configuration of *Virtual Network Security Functions* (vNSFs), and the Service Function Chaining in *NFV Management and Orchestration* (NFV MANO).

The first problem addressed in this research is strongly related to the concept of policy-based management discussed in [7]. Indeed, the ETSI standards clearly express the need for an integrated security policy manager, which defines the way to choose the VNFs and to link and configure them. Our framework gives an answer to that requirement and tries to bring to light the major problems encountered in this process of integration.

The second problem is related to configuration. This is a nontrivial issue considering the number of tools available on the market and the differences between the vendors that develop the VNFs. The choice for the most efficient and reliable tool could make the difference in contexts in which scalability and flexibility are not expendable. We have explored the major tools on the market, and we have chosen the ones more suitable for the problem.

In the end, we addressed the last problem, which some NFV MANO platforms do not even consider (e.g., Open Baton): the steering of the traffic leveraging *Service Function Chaining* (SFC) and/or *VNF Forwarding Graphs* (VNFFGs). We proposed and tested a solution which integrates in our framework this feature, giving full support for VNFFGs.

The rest of the paper is structured as follows. Section 2 briefly recalls the main works in literature that focus on SECaaS in an NFV environment, SFC, VNFFGs, and NFV MANO. Section 3 presents the proposed architecture and framework, while Section 4 discusses the configuration of the vNSFs and, finally, Section 5 examines the creation of Service Function Chains with these vNFSs. Finally, Section 6 evaluates our framework and Section 7 concludes the paper.

2. Background and Related Work

The focus of this paper is centered on the new NFV Management and Orchestration technologies and the use of the SECaaS paradigm in order to grant security to the end-user. We start by describing the background and the related works concerning those technologies and the SECaaS paradigm.

2.1. Security-as-a-Service in an NFV Environment. The concept of SECaaS takes place starting with another notion: the vNSF. In an NFV environment, indeed, it is possible to define a particular VNF [8] offering security functionalities (e.g., vFW, vIDS, and vDPI) and hence called vNSFs. SECaaS is a new approach based on NFV Orchestration which deploys security protections by means of vNSFs. The derived security services could offer dynamic response for a specific set of threats addressable with the available vNSFs; scalability based on the enterprise capabilities; and targeted security data monitoring and gathering at a specific point in the network for analysis and remediation.

In the literature, the SECaaS approach has been deeply investigated in the last few years. For instance, some authors [9] proposed a "user-centric" approach for the provision of virtualised security at the network edge. In that paper, a complete framework design is shown, in which an NFV Infrastructure is used to provide security services for the end-user terminals. The vNSFs, which in that case were called *Personal Security Applications* (*PSAs*), were driven by a policy-based management system. That system was in charge of gathering user policies (expressed in an high-level language: the *High-level Security Policy Language (HSPL)*) and the admin policies (expressed in a medium-level language: the *Medium-level Security Policy Language (MSPL)*); translating those policies in a service graph, which describes the needed PSAs and the links between them, and a set of low-level configurations for the PSAs used; and enforcing those low-level configurations on the PSAs deployed starting from the service graph provided.

Another SECaaS example is shown in [10], where the authors provide an extensible, adaptable, fast, low-cost, and trustworthy cybersecurity solution. This aims at delivering IT security as an integrated service of virtual network

infrastructures that can be tailored for ISPs and enterprise consumers. In this case, vNSFs are dynamically instantiated and configured after a malicious event or a threat. Furthermore, the choice of the attack remediation is driven by a *Data Analysis and Remediation Engine* (DARE); this engine features analytical components and is capable of predicting specific vulnerabilities and attacks. The DARE relies on continuous monitoring of the network traffic, using monitoring vNSFs, and translates their observation into a collection of data capable of inferring events which a single vNSF could not.

2.2. Threat Analysis. As specified by [11], there are many threats that involve the NFV Infrastructure itself, at the NFVI, VNF, and NFV MANO layers, respectively. However, our approach aims at solving threats that affect user's perspective, using a SECaaS approach. As claimed before, errors in network security configuration can create vulnerabilities that in turn affect the overall security, decrease the network throughput, and increase the maintenance costs.

2.3. Service Function Chain and VNFFG. IETF defines a Service Function Chain [12] as a set of abstract service functions and ordering constraints that must be applied to packets and/or frames and/or flows selected as a result of *classification*. The classification is the process of matching the traffic flows against the policy for the subsequent application of the required network service functions. The *Classifier* is the element that performs the classification. In other words, the SFC is the mechanism which allows forwarding the traffic of an end-to-end service and defining in which VNFs the latter must pass through.

ETSI, instead, defines a similar object called *VNF Forwarding Graph (VNFFG)* [13], a graph whose nodes represent the VNFs and the logical links define the connection between them; the latter could be unidirectional, bidirectional, multicast, and/or broadcast. The simplest example of VNFFG is a single chain of VNFs. We have also two additional elements: the *Forwarding Service Path (FSP)* and the Classifier; the first one is the equivalent of an SFC; the second is the Classifier which is associated with the FSP in order to classify the traffic on the right path.

In [14], a use case of SFC applied to NFV and SDN technologies is reported. In particular, an SFC Orchestrator capable of dynamically updating the Service Function Paths at any time has been proposed. The utility of this approach is clear during a fault or a decrease in the performance of a VNF. In that case, all the other VNFs in the chain suffer as far as the previous one is a bottleneck. A remediation to this problem is scaling out the affected VNF though we need to reconfigure also the SFPs in order to cleverly redistribute the traffic along the new instances.

One of the challenges which we aim to address regarding the SFC is partially discussed in [15]. Mechtri et al. proposed a new End-to-End SFC Orchestration Framework, which is compliant with ETSI NFV MANO, in order to overcome at least two problems: giving a harmonized service abstraction and description languages for NFV and SDN requirements and automating end-to-end service production for agile NFV services. The main reason why we consider another solution is because we had the aim of completely leveraging on the existing platforms that have already addressed this problem without implementing a new NFV MANO.

2.4. Management and Orchestration in NFV MANO. Management and Orchestration of the NFV Infrastructure plays a central role in modern computer networks. Centralizing the deployment, management, and monitoring of the *Virtual Network Functions (VNFs)*, which give the possibility to have a complete end-to-end *Network Service (NS)*, simplifies the way the service providers reach their users. Our focus in this paper is on the configuration of the VNFs and the support for Service Function Chaining which an NFV MANO should provide. In the following, we analyse the main open source NFV MANO platforms on the market:

(i) *Open Source MANO (OSM)* (available online at https://osm.etsi.org) is an ETSI-hosted project to develop an Open Source NFV MANO framework stack aligned with ETSI NFV standards. It offers a multiple *Virtual Infrastructure Manager (VIM)* support, which means it could be used with multiple Infrastructure-as-a-Service technologies (e.g., OpenStack, AWS, OpenVIM, and VMware) for the resource orchestration. OSM allows also to combine them with different SDN Controller technologies (e.g., ONOS, floodlight, and ODL) to manage the underlying connectivity. A monitoring system and an experimental support for VNFFG are provided as well. In order to manage the VNFs instance, OSM adopts Juju of Canonical as VNF Manager.

(ii) *Open Baton* (available online at https://openbaton.github.io) is an NFV MANO solution compliant with the ETSI NFV MANO specifications. It has a modular architecture, in which a message broker (RabbitMQ) grants the communication between a set of different orchestration and supplementary services. The main services offered are the orchestration of resources, the monitoring system, and a set of drivers which allow to use this platform over multiple VIM technologies. In order to extend Open Baton for supporting other VIMs, it is required to create a new driver plug-in and a specific VNFM for this technology. This approach gives an interesting flexibility to the VIM support; indeed, Open Baton has been the first NFV platform to give support to Docker Engine as VIM. So far, this platform, however, does not support SFC or VNFFGs mechanisms.

(iii) *OpenStack Tacker* (available online at https://docs.openstack.org/tacker/latest/) is an additional service for the OpenStack framework (available online at https://www.openstack.org). This service provides NFV Orchestration functionalities (e.g., VNFs/NSs management), leveraging the services included in

the OpenStack IaaS platform (e.g., Neutron, Nova, and Heat). The main advantage of using this platform is the full support for SFC with the service named networking-sfc and VNFFGs; in the *Network Service Descriptor (NSD)*, we could provide a high-level description of FSPs and Classifiers as well.

Although NFV MANO technologies are growing fast, there are still different challenges to address. As the authors of [16] report, one of the most significant ones is the dependability level of NFV. This depends on the NFV MANO platform for two different reasons: the NFV MANO manages the available resources to provide optimal service; therefore, it may be used to deal with failures of network elements efficiently and to improve service dependability; on the other hand, since the NFV MANO maintains a global view of the NFV system, it may affect the entire network as result of a mismanagement. Moreover, that paper provides a tutorial on these NFV MANO challenges and gives a clear insight into further research on how to address those problems. In our paper, we focused our attention on vNSF configuration and Service Function Chaining, so we chose the MANO platforms that better address those challenges and compare them: OSM and Tacker.

2.5. Configuration Tools. We explore in this section the main tools for virtual instance configuration and give an overview of the options available on the market:

(i) *Juju* (available online at https://jujucharms.com) is a tool from Canonical which provides instantiation, management, monitoring, and scaling of cloud applications in an efficient and rapid way. We could perform those actions on different platforms (e.g., OpenStack, AWS, Google Platform, and LXD) or even physical machines (Metal-as-a-Service (MaaS)). The way Juju works is a collection of Python or Bash scripts called "Charm." A Charm provides all needed information to deploy and configure a virtual instance and the tools for its monitoring. We need the Charm running on the instance in order to configure it.

(ii) *Ansible* (available online at https://www.ansible.com) is a tool developed by the Ansible Community and Red Hat. It performs all the required operations in order to deploy, manage, and configure a virtual instance via SSH, leveraging a Python interpreter. The operation to be performed on a virtual instance could be expressed in the form of a YAML descriptor, which is called a "playbook." All the operations performed by Ansible do not require any software daemon on the instance.

(iii) *Chef* (available online at https://www.chef.io) is another tool for deployment and management automation developed by the Chef company. This is one of the first tools for this purpose and is used by providers like Facebook, AWS, and Prezi. In this case, we need a client daemon on the instance in order to perform the configuration on the virtual instance.

(iv) *Puppet* (available online at https://puppet.com) is a tool for deployment and management automation developed by Puppet Company. As in the case of Chef, it requires a software daemon on the instance in order to execute the actions.

(v) *SaltStack* (available online at http://www.saltstack.com) is a tool for the management of virtual instances as the others described above. In this case, we could have a daemon on the instance (salt minion) or not (salt ssh) depends on the version. The features offered are the same as in Ansible, but the support for different platforms and the power of syntax is better in Ansible.

Considering the listed features provided by the analysed configuration tools, we can conclude that Ansible is the more suitable platform for use in our case.

3. Framework Design and Architecture

Starting from the premises expressed in the last section, we now present the whole framework architecture; this was built to provide the *Network Security Service (NSS)* for the end-user. The main aspects described in this section are as follows:

(i) High-level architecture: an overview of the whole solution architecture

(ii) Security Service Manager (SSM): the management and orchestration framework for the NSS

(iii) NFV Infrastructure: the infrastructure for the vNSF deployment

(iv) Solution workflow: a summary of the solution workflow based on an ISP use case

3.1. High-Level Architecture. Our high-level architecture composed of two main elements: the Security Service Manager and the NFV Infrastructure (Figure 1). The Security Service Manager is in charge of the whole NSS orchestration. The starting point is given by the user, which could submit a specific set of high-level security policies to the SSM. The latter is in charge of the translation of those policies to a precise collection of configurations and an overall description of the Network Security Service in the form of an NSD. The configurations are generated for a different set of vNSFs, which are implemented in specific technologies. The NSD, instead, is designed for the target NFV MANO platform (in our case, OSM) and has the aim to characterize the vNSFs required and the links between them. After the policy translation process, a service provider could deploy the NSS of the specific user on its own infrastructure according to the given policies. The SSM is also in charge of this deployment and its management. For what concerns the infrastructure, instead, it composed of different Security Sites; Security Site is a generic *Point-of-Presence (PoP)* (e.g., data centre), which typically consists of a VIM and different

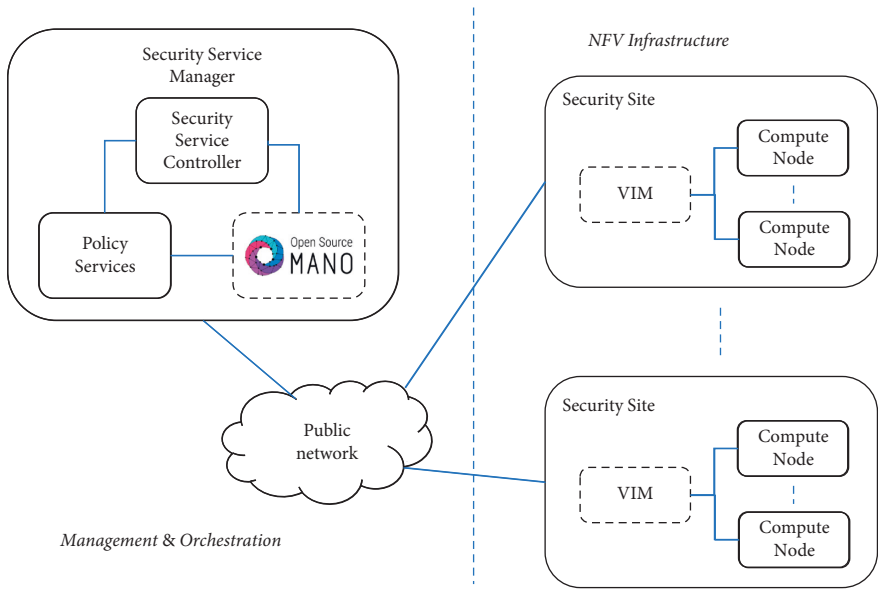

Figure 1: Framework architecture.

Compute Nodes. A Security Service Manager is able to manage different geographically distributed Security Sites; indeed, OSM offers a complete support for the multi-VIM management and orchestration.

3.2. Security Service Manager. In Figure 1, the main components of the Security Service Manager are depicted: Open Source MANO, the *Security Service Controller*, and *Policy Services*. These elements are designed and implemented to fit in a fully "containerized" environment; indeed, we adopt LXD (available online at https://linuxcontainers.org/lxd/introduction/) as virtualisation technology to deploy the services linked to our solution. LXD is a next-generation container technology based on Linux Containers and offers a user experience similar to virtual machines. This virtualisation solution provides also advanced features regarding security, scalability, network management, resource control, and device management in a Linux Container context (e.g., unprivileged containers, supports for multiple storage backends, and bridge creation/configuration). In our proposed design, the OSM Release Three architecture was extended adding new containers in charge of providing each specific service needed by our framework, forming three sets of containers matching the SSM components. The "Policy Services" containers handle the policy management; in particular, it provides the following functionalities:

(i) Policy Reconciliation Service: this service provides the aggregation of the policies from different actors (e.g., different users and admins).

(ii) Policy Refinement and Optimization: it handles the translation from high-level security policies to medium-level security policies; the latter depend on the types of vNSFs used but not on the specific vendor technologies. Besides, this service is in charge to draw up the service graph which defines the vNSFs to use for the NSS and the links between them [17, 18].

(iii) Policy Analysis Service: this module is in charge of the policy analysis in order to find any anomalies among them [6, 19].

(iv) User Policy Repository (UPR): this is the storage for all levels of policies and the service graphs.

(v) Consul and Vault: this provides service discovery functionality and access control for the other services.

The second set of containers is the *Security Service Controller (SSC)* which is in charge of giving the high-level functionalities for both end-user and service provider side (e.g., policy definition and NSS deployment). The SSC allows to upload new security policies from the user side and, at the same time, it manages the automatic instantiation of a new NSS for a specific user from the service provider side.

In the end, we have the last set of containers, which are linked to OSM, providing the NFV MANO functionalities: VCA, SO, and RO. The *VNF Configuration & Abstraction (VCA)* is a *VNF Manager (VNFM)* abstraction; the latter provides the life cycle management of the vNSFs leveraging Juju functionalities. The *Resource Orchestrator (RO)* is in charge of the resource orchestration (e.g., deploying vNSFs and managing the VIM). The *Service Orchestrator (SO)* is the component in charge to the NSS life cycle management (e.g., launch, stop, and configure a NSS) and also performs monitoring tasks on the Network Security Service.

3.3. NFV Infrastructure. The infrastructure in our solution consists of different Security Sites. This requires the possibility to abstract the management and orchestration layer from the infrastructural one. Indeed, this means we could use different VIM technologies at the same time. Our architecture allows to maintain this separation and use

multiple VIMs to deploy the instances. In our framework, we implement a test Security Site with OpenStack as VIM; OpenStack is a standard de facto for what concerns the open source IaaS platform and almost every NFV MANO framework is compliant with it. OpenStack is also capable of giving every resource required by Open Source MANO (e.g., compute, network, and storage) leveraging Nova and Neutron API.

3.4. Solution Workflow. In Figure 2, we showcase a typical use case consisting of an ISP which provides a Network Security Service to an end-user. All start with the user itself, during the step (1), who defines the high-level policies through the Policy GUI, which is a Graphical User Interface exposed by SSC in order to allow the user to manage his policies. After this action, the high-level policies are stored in the UPR and an ISP could start the NSS for this specific user. In order to complete this task, the ISP operates the *NSS Deployer* (2) which leverages the Policy Services to reconcile and refine the user policies (3). In particular, in this phase, we have the Policy Reconciliation, the translation from HSPL to MSPL and the provision of the service graph. Thereafter, the NSS Deployer translates the service graph in a consistent Network Service Descriptor for the NFV MANO platform and sends it to the SO together with the MSPL policies. The SO at this point (5) leverages the RO in order to deploy the virtual instance required for the NSS and the network configuration for the underlying connectivity. In the end (6), the SO through the VCA enforces the policies on the vNSFs and so the NSS could be defined "online." The translation in this process from MSPL to low-level configuration is performed by the vNSF itself.

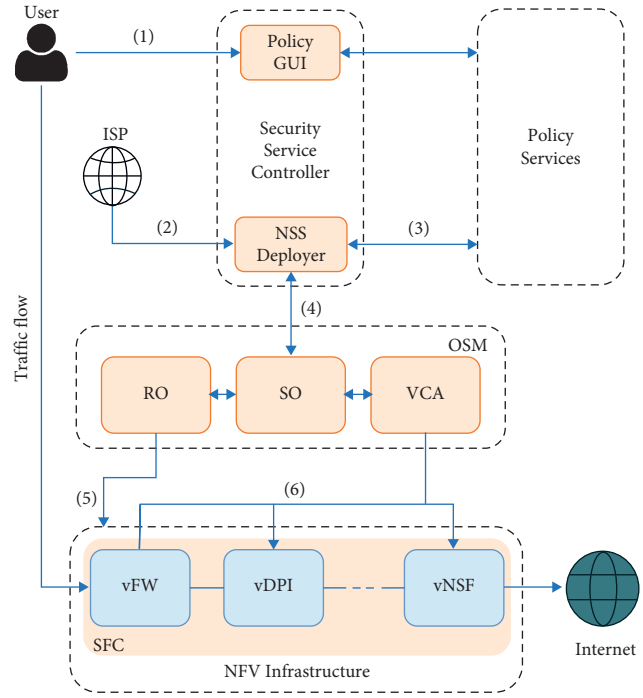

Figure 2: Solution workflow related to an ISP use case.

4. vNSF Configuration

One of the most relevant challenges for the NFV Orchestration is the configuration and the life cycle management of the vNSFs. Since we used in our solution OSM as NFV MANO, the vNSF configuration relies on Juju as VNF Manager (VNFM); adopting this solution (Juju with OSM) gives advantages in terms of flexibility for the vendor, who could embed a series of configurations scripts called "Proxy Charms" directly in the OSM *VNF Descriptor*. The complication in this case lies in the overhead that is introduced by Juju in the process of vNSF creation, at least in a limited resource scenario. We have discussed this strategy and proposed a new approach for the configuration in order to avoid this drawback: using Ansible as VNFM.

4.1. First Approach. Starting a Network Security Service implies the launch of all the required vNSF instances and the proper configuration of them. In our solution, we perform this operation in two different steps:

(i) Day-0 configuration: during this step, we provide all the software and configurations needed by the vNSF via cloud-init technology, starting from an Ubuntu cloud image; at the end of the process, we deliver a fully working vNSF which gives the desired service. With this peculiar technology, we are able to install the software packages required, inject the SSH keys, run arbitrary scripts, and configure the network interfaces of the instance.

(ii) Day-1 configuration: since the first type of configuration stops after the initialisation of the vNSF, we need to provide a second configuration mechanism for the Virtual Network Security Function life cycle management. In order to achieve this result, we adopted the VNFM suggested and supplied by OSM framework: Juju.

The Day-0 configuration has been provided as a technique to avoid the use of prebuilt software images. This approach allows to save space on the software image repository since we need for this new approach just a small set of a base cloud image (e.g., Ubuntu 16.04 LTS). On top of them, as described before, we could configure all the software requirements and configurations via cloud-init. The Day-1 configuration, instead, is crucial to avoid the reinitialisation of the vNSFs after a change in the behaviour of the security service. A typical example could be a firewall reconfiguration during the life cycle of the security service. With this type of configuration, we could change the settings of the specific vNSF, acting in this case like a firewall, without reinitialising neither the vNSF nor the Network Security Service.

The way we provide this functionality is via Juju, as described before. Juju allows to perform a set of actions triggered by OSM on every specific vNSF (e.g., start, stop, and set rules), leveraging the mechanisms of the Proxy Charm (Figure 3). A Proxy Charm is a virtual instance

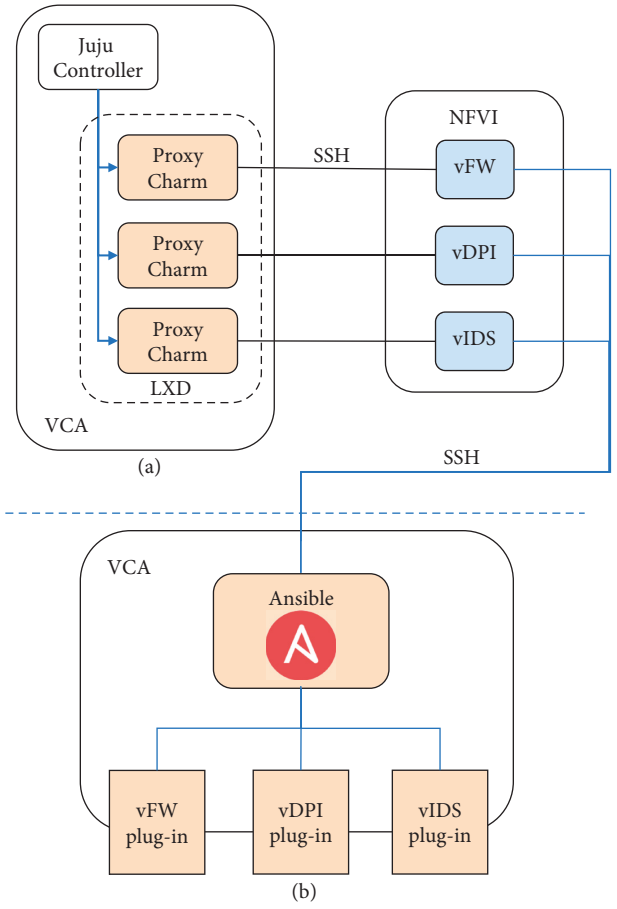

Figure 3: VNF Managers: Juju vs. Ansible comparison.

(typically an LXD container) which is in charge of the management operations on the vNSF. This virtual instance embeds a series of Python scripts and libraries which allow to communicate via SSH to the final vNSF instance and to execute arbitrary python code in order to manage the vNSF. The set of scripts needed from the Proxy Charm have to be loaded in advance in the VNF Descriptor (VNFD) package. With this approach, every single vNSF has his own Proxy Charm and the vendor has the flexibility to customize the vNSF configuration, without changing anything at the VNFM layer, directly in the description of the vNSF. Even though this approach is very flexible and scalable, it has a consistent drawback: we need a virtual instance for each single vNSF. This requirement could be a significant obstacle especially for NFV MANOs with limited infrastructure resources.

4.2. Evolution. The Juju solution for the second-step vNSF configuration could appear as a real breakthrough, but the problem lies in the overhead introduced for the new instances and their configuration. For our purpose, as in many real use cases, it is not mandatory to adopt Juju as VNFM; this is because we need a high-level controller (the SSC) to perform automatically the actions on the vNSF after some event on the platform (e.g., remediation to an attack and change of the user policies). This means we could rely on any system which allows us to configure the instance during its life cycle (e.g., Puppet, Chef, Ansible, and SaltStack). We choose Ansible as a replacement because it does not require a daemon on the instance to perform the configurations; other choices (e.g., Chef and Puppet) would lead us to require that daemon and call for further configurations and resources during the virtual instance launching process. For this reason, we had started to explore Ansible as VNF Manager evolution for our framework and we proposed and tested an alternative plug-in mechanism with Ansible "playbooks" (Figure 3).

A playbook is a file descriptor representing the language used by Ansible to describe the operations to be performed on a virtual instance. The single operation is called *task*, and the playbook is essentially a collection of them. One of the most relevant advantages of using Ansible is the powerful semantics given by the playbook and its tasks; they offer an easy way to install software packages, to copy files, and to perform all the typical required operations on a virtual instance in the form of a YAML descriptor. Besides, another advantage is no need for a dedicated virtual instance for every single vNSF as the Juju case; in that case, it was needed an LXD container per Proxy Charm; now it is required only an LXD container hosting Ansible framework. The only additional operation to be performed is in the upload process of a new vNSF Descriptor (e.g., a new supported vNSF) and is the loading of the new plug-in for the vNSF in the Ansible Container. The plug-ins we designed are a collection of playbooks capable of performing every "Day-1 configuration" needed by the vNSF.

The Security Service Controller is in charge of calling the proper configuration through Ansible; thus, we could reconfigure the vNSFs during their life cycle leveraging this mechanism. Ansible is also extensible for use with Docker (available online at https://www.docker.com); therefore, this solution is also suitable in the container case. As we will see in the next section, we also addressed the problem of the automatic traffic steering, leveraging another NFV MANO platform rather than OSM. Even in that case, the two-step configuration with cloud-init and Ansible is applicable without substantial changes.

5. Service Function Chaining

Another challenging problem of the NFV Orchestration is the steering of the traffic. Many of the best known NFV platforms (e.g., Open Baton) address the problem of the NFV Management and Orchestration without the use of the Service Function Chaining or the VNF Forwarding Graph. This means that the traffic between the vNSFs is configured manually in every single vNSF or at least it needs to be provided an external controller to manage the traffic flows. Our goal is to configure the whole security service starting only from the NSD; thus, the traffic steering should be automatically configured between the vNSFs as well. Another question about the traffic steering is the support for SFC and VNFFGs given by the NFV

MANO. In order to classify the different traffic flows, an NFV Orchestration framework needs to support this feature and the libraries offered by the specific VIM adopted. For instance, Open Source MANO gives an experimental support to VNFFGs only since the OSM release FOUR, while other platforms like Tacker offer a more stable support for this kind of technologies. In this section, we will show how we could integrate Tacker in our framework.

Leveraging the VNFFGs support given by Tacker, we proposed an extension of our framework which allows to define different paths for the traffic flows. As shown in Figure 4, for example, we could replicate the scenario of the user who needs a security service in order to access the Internet. In this scenario, we have different types of users (e.g., based on the type of access) and different types of traffic as well (e.g., different transport or application protocols). Based on this information, we can classify the traffic flows and redirect the traffic itself only through the needed vNSF instances. This approach could be a real breakthrough in terms of flexibility in different scenarios, in particular in which the network is divided based on the type of traffic and the specific business case (e.g., the network slicing in 5G).

OpenStack Tacker is an orchestration platform that fully supports the VNFFGs and Service Function Chaining for a specific VIM, in this case OpenStack. Tacker leverages the OpenStack service Heat for the Resource Management of the vNSFs, while it uses the OpenStack service networking-sfc to support the VNFFG and instructs the virtual switches, which are implemented in *Open vSwitch (OVS)* technology, on how they have to manage the traffic. Tacker also has two types of descriptor as the OSM case: VNF Descriptor and NS Descriptor. In the last one, it is possible to define not only the vNSFs which are required for the service, but also the FSPs and the Classifiers for each path. Our idea is to start from the high-level policies and give in their translation information about the way we need to classify the traffic, in the form of the final NSD.

At the end, in this descriptor, we have all the information about the traffic steering, the different FSPs, and the Classifiers. The only change to be made in our framework is the translation of the service graph to another NSD model. We have also another advantage in using Tacker: the lower requirements in terms of infrastructure resources. With OSM, we need to install a complete NFV Orchestration framework: the VNFM Juju, the monitoring services, the Resource Orchestrator, and the Service Orchestrator. All those services rely on the specular VIM services: OpenStack Nova and Heat for the resource management, Ceilometer for the monitoring and the metrics, and networking-sfc for the Service Function Chaining. Using Tacker, instead, allows to add one only service to the OpenStack VIM platform and to rely on a complete NFV MANO framework. A reason for supporting OSM could be his potential support for different VIMs. The problem until now is that the best VIM platform supported from OSM is OpenStack, and there are still some services (e.g., networking-sfc) for which it gives an experimental support.

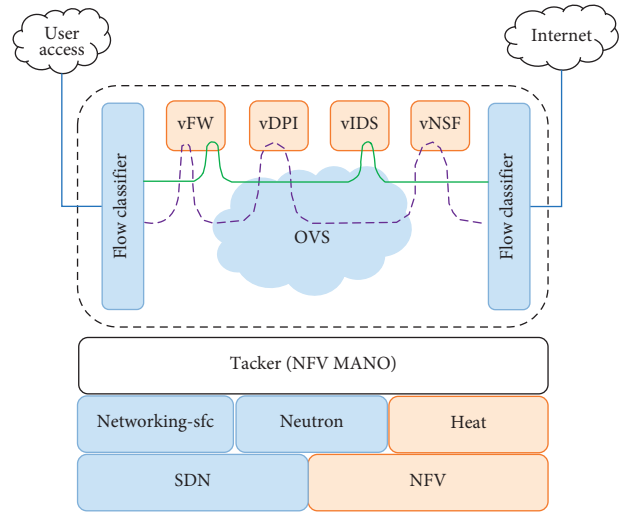

Figure 4: Service Function Chaining: a SECaaS use case.

6. Evaluation

We evaluated our framework in different scenarios, based on the different technologies used. In particular, we have defined two different scenarios:

(i) VM scenario: in this scenario, we explored the instantiation time of the Network Security Service depending on the number of the required vNSFs. More specifically, we used virtual machines on KVM as virtualisation technology and OpenStack as VIM. For what concerns the configurations, we leveraged both Day-0 and Day-1 configurations; the last one was performed with Juju.

(ii) Container scenario: in this scenario, we still have evaluated the time instantiation of the NSS, but using Docker as virtualisation technology and vim-emu (available online at https://osm.etsi.org/wikipub/index.php/VIM emulator) as VIM. In particular, in this scenario, Day-0 configuration was performed with Dockerfile and no Day-1 configurations are intended to be performed.

In these test scenarios, we used two physical machines, one acting as Security Site (VIM and Compute Node) and the other acting as a SSM (SSC, Policy Services, and OSM). Both the machines had an Intel Core i7 (quad-core 2.6 GHz) CPU, 16 GB of RAM, 512 GB of SSD storage, and Linux Ubuntu server (16.04.3 LTS) as Operating System. In the first scenario, we tested the first solution with OSM and Juju, but we also discussed the possible result enhancement using the Ansible solution. In the second scenario, we leveraged an experimental VIM provided by OSM as described below.

6.1. VM Scenario. In this scenario, we used virtual machines and OpenStack as VIM. We evaluated the instantiation time of the NSS depending on the number of vNSFs required. We also performed the two types of configuration described above. We need to point out that the Day-0 configuration

with cloud-init is mandatory in our framework due to the use of cloud images. For this reason, we show in Figure 5 the trend of the instantiation time according to the number of vNSFs used in the NSS itself. The dotted line indicates the time for instantiation of the resources needed by the VMs and the initialisation time of the cloud image. The dashed line instead shows the configuration time due to the Day-0 configuration. The continuous line reports the total time needed for the security service to be at disposal. As we can understand from the graph, the configuration time in this case is in the same range (1–10 s), as the cloud-init configuration is performed in parallel within all the vNSFs. This time, depending also on the specific requirements for the vNSF, we use random vNSFs (vFW, vDPI, and vIDS) with different requirements, but this finally does not depend on the number of vNSFs instantiated. The time of initialisation instead depends on the number of the vNSFs, and it is almost linear as the number of vNSFs increases. In another scenario, we also test the deployment of the same security services on two different OpenStack VIMs, where the second one is an exact clone of the first described with the same physical resources.

We experimented a constant decrease in terms of time needed to initialise the NSS, and it is shown in Figure 6. With this first configuration (Day-0), we had a complete working NSS without the possibility to reconfigure it during its life cycle. If we want this feature, we need to rely on something different as the Juju. We tested this scenario evaluating the time needed to deploy and configure the virtual instances (LXD containers) which have to run the Proxy Charm in charge of the reconfiguration of the vNSFs. As shown in Figure 7, we notice that the time needed in this case (continuous line) to instantiate the security service significantly diverges as the number of vNSFs increases. We also evaluated the overhead introduced by Juju (dashed line) in the instantiation time, comparing it with the previous case (dotted line). The graph shows how Juju introduces a significant delay and is not the best solution for the Day-1 configuration. We proposed as evolution Ansible which needs no instantiation time for every single vNSF and gives the same functionality as Juju in the reconfiguration of the vNSFs. Using the framework evolution, we could save the overhead time introduced by Juju and bring the whole instantiation time to the one provided in the first results with only Day-0 configuration.

6.2. Container Scenario. With respect to the Container Scenario, we used an experimental feature of OSM, which was created in order to obtain the Docker container orchestration within OSM. We deployed a set of containerized vNSFs and evaluated the time needed for the instantiation and the Day-0 configuration. In this case, we operated the Dockerfile as Day-0 configuration and no tools in order to perform the Day-1 configuration. As shown in Figure 8, we had a significant increase in terms of performance in the instantiation time. The dashed line represents the Docker instantiation time while the continuous one represents the instantiation time with the virtual machines technology. For

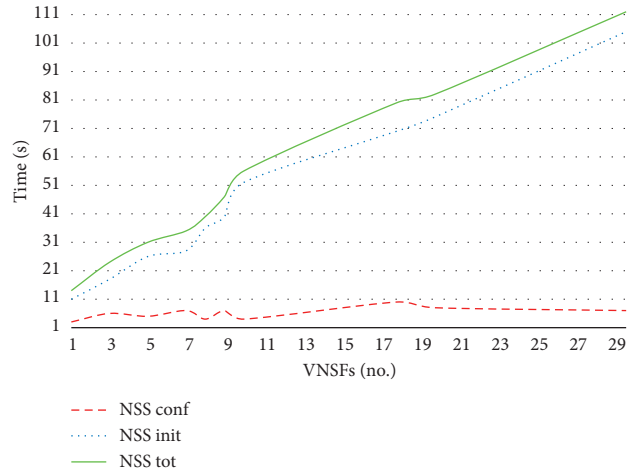

Figure 5: NSS instantiation time: initialisation and Day-0 configuration.

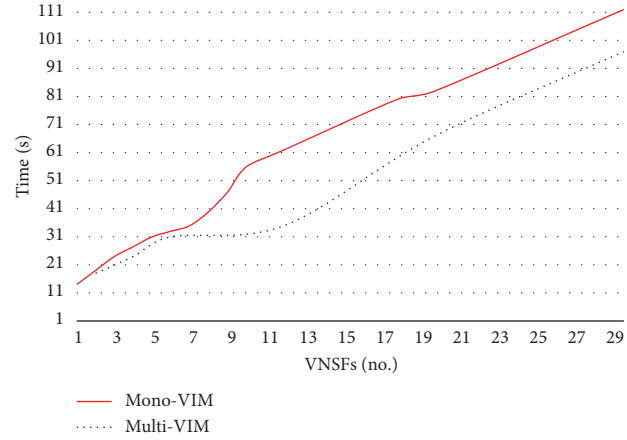

Figure 6: NSS instantiation time: monosite vs. multisite.

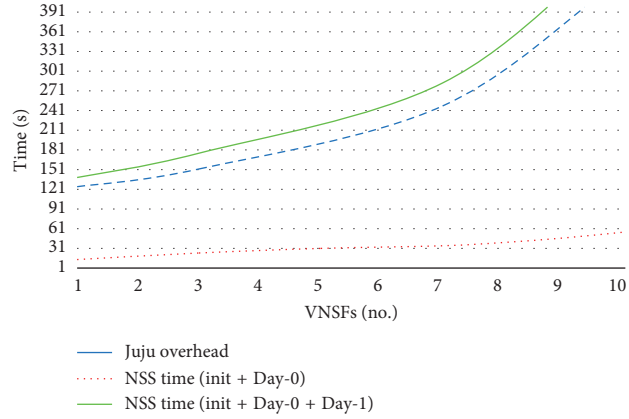

Figure 7: NSS instantiation time: initialisation, Day-0 configuration, and Day-1 configuration.

what concerns the Day-1 configuration, we could use two approaches in the future concerning the container case: the first approach is to restart the whole NSS due to the shorter time to instantiate the container instead the VMs, and the

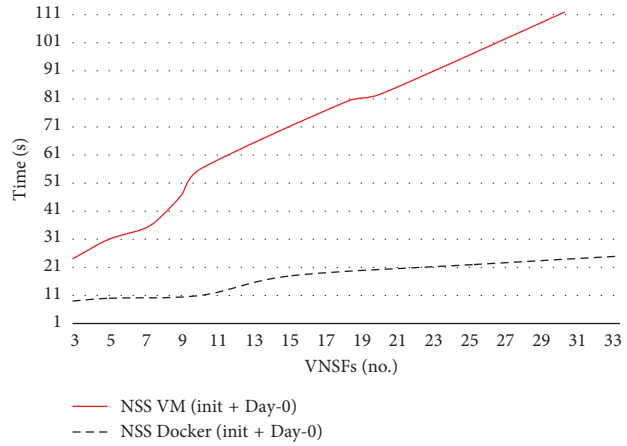

Figure 8: NSS instantiation time: virtual machines vs. containers.

second one is to use Ansible to reconfigure it with the same technique of the VM case, but leveraging the *Docker Deamon API* instead of the SSH connection to the instances. This scenario is experimental and represents an interesting evolution for the SECaaS paradigm on NFV technologies. The aim of this test is to show how promising could be a future integration and NFV Orchestration with container technologies.

7. Conclusions

In this paper, we have proposed a practical implementation of the SECaaS paradigm, leveraging NFV and SDN technologies. The way we stated the main challenges and proposed solution to them supplies an improvement on the state of the art about what technologies are more reliable and useful in order to implement this approach by an ISP. Furthermore, the configuration of the vNSFs and the Service Function Chaining are two critical points, which are widely pointed out in ETSI's standards, and they require to be deeply investigated in the time to come. Part of this work has been focused to address this question. Another contribution is about the tests made on the framework and on the infrastructure presented in both a virtual machine and container case scenario. These tests show how the adoption, or at least the partial adoption, of lightweight virtualisation could dramatically improve the Network Security Service performances. In conclusion, this paper outlines some of the future challenges and evolution of this work, underlining the need for new resource optimization techniques and a further investigation about the container configuration technologies.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work has been partly supported by the SHIELD project, cofunded by the European Commission (H2020 grant agreement no. 700199).

References

[1] IBM, "IBM X-force threat intelligence index 2018 notable security events of 2017, and a look ahead," 2018, https://www.ibm.com/downloads/cas/MKJOL3DG.

[2] Kevin Townsend, "Verizon: 2019 data breach investigations report," Verizon, Tech. Rep., IETF Datatracker, 2019.

[3] L. Durante, L. Seno, F. Valenza, and A. Valenzano, "A model for the analysis of security policies in service function chains," in *Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–6, Bologna, Italy, July 2017.

[4] C. Basile, D. Canavese, A. Lioy, C. Pitscheider, and F. Valenza, "Inter-function anomaly analysis for correct SDN/NFV deployment," *International Journal of Network Management*, vol. 26, no. 1, pp. 25–43, 2016.

[5] ETSI, "GR NFV 001—V1.2.1—network functions virtualisation (NFV); use cases," 2017, https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[6] F. Valenza, C. Basile, D. Canavese, and A. Lioy, "Classification and analysis of communication protection policy anomalies," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2601–2614, 2017.

[7] ETSI, "GS NFV-SEC 013—V3.1.1—network functions virtualisation (NFV) release 3; security; security management and monitoring specification," 2017, https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx.

[8] ETSI, "GS NFV-SWA 001—V1.1.1—network functions virtualisation (NFV); virtual network functions architecture," 2014, http://portal.etsi.org/chaircor/ETSI_support.asp.

[9] D. Montero, M. Yannuzzi, A. Shaw et al., "Virtualized security at the network edge: a user-centric approach," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 176–186, 2015.

[10] G. Gardikis, K. Tzoulas, K. Tripolitis et al., "SHIELD: A novel NFV-based cybersecurity framework," in *Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–6, Bologna, Italy, July 2017.

[11] M. Pattaranantakul, R. He, A. Meddahi, and Z. Zhang, "Secmano: towards network functions virtualization (nfv) based security management and orchestration," in *Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 598–605, Tianjin, China, August 2016.

[12] J. Halpern and C. Pignataro, "Service function chaining (sfc) architecture," *in Internet Requests for Comments, RFC 7665*, IETF Datatracker, 2015.

[13] ETSI, "GS NFV-MAN 001—V1.1.1—network functions virtualisation (NFV); management and orchestration," 2014, http://portal.etsi.org/chaircor/ETSI_support.asp.

[14] A. M. Medhat, G. A. Carella, M. Pauls, and T. Magedanz, "Orchestrating scalable service function chains in a NFV environment," in *Proceedings of the 2017 IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5, Bologna, Italy, July 2017.

[15] M. Mechtri, C. Ghribi, O. Soualah, and D. Zeghlache, "NFV orchestration framework addressing SFC challenges," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 16–23, 2017.

[16] A. J. Gonzalez, G. Nencioni, A. Kamisinski, B. E. Helvik, and P. E. Heegaard, "Dependability of the NFV orchestrator: state

of the art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3307–3329, 2018.

[17] C. Basile, F. Valenza, A. Lioy, D. R. Lopez, and A. P. Perales, "Adding support for automatic enforcement of security policies in NFV networks," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 1–14, 2019.

[18] A. Basile, C. Pitscheider, F. Risso, F. Valenza, and M. Vallini, "Towards the dynamic provision of virtualized security services," in *Cyber Security and Privacy Forum*, pp. 65–76, Springer, Berlin, Germany, 2015.

[19] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Lioy, "A formal model of network policy analysis," in *Proceedings of the 2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, pp. 516–522, Turin, Italy, September 2015.