Validation and Benchmarking of CNFs in OSM for pure Cloud Native applications in 5G and beyond

Adrián Pino*, Pouria Khodashenas*, Xavier Hesselbach‡, Estefanía Coronado* and Shuaib Siddiqui*
*i2CAT Foundation, Barcelona, Spain; Email: {adrian.pino, pouria.khodashenas,
estefania.coronado, shuaib.siddiqui}@i2cat.net
‡Department of Network Engineering, Universitat Politècnica de Catalunya (UPC),
Barcelona, Spain; Email: xavier.hesselbach@upc.edu

2021 International Conference on Computer Communications and Networks (ICCCN) | 978-1-6654-1278-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICCCN52240.2021.9522356

***Abstract*—Cloud Native (CN) in 5G systems has been identified as a pivotal candidate for operational and capital expenditure savings as well as for improvements in system agility and services role-out. CN telco is a step forward with respect to Network Function Virtualisation (NFV) aiming at embracing a microservice-based architecture. With this in mind, the European Telecommunications Standards Institute (ETSI) has evolved the ETSI NFV reference architecture to adapt to CN and fill the gap with the NFV framework, including containers and Zero-Touch, among other capabilities. Open-source Management & Orchestration (MANO) initiatives, such as Open Source MANO (OSM), are promoting this adoption giving support to CN solutions based on containers. However, at this early stage deployments are currently non-standalone and embedded in VNF-based solutions such as OpenStack. In this context, this paper presents a proof of concept of a full container technology deployment -via Kubernetes- in a NFV architecture. First, a full CN NFV environment is set with the help of OSM MANO, for which we describe the implementation to enable native kubernetes-based Container Network Functions (CNFs) and analyse their performance, limits, advantages and drawbacks. Finally, our solution for CNFs is benchmarked against a typical OSM-OpenStack setup where VNFs are deployed. The results obtained in this work can help to further encourage users and operators to use CNFs and get the most out of containerisation in NFV.**

***Index Terms*—NFV, 5G, Cloud Native, OSM, K8s**

I. Introduction

Network Function virtualisation (NFV) has been one of the most groundbreaking networking concepts in recent years, yielding attention from both industry and academia, and leading to a change of generation on how network elements and functions are managed. By replacing the proprietary hardware with software network functions via Network Services (NSs) based on Virtual Network Functions (VNFs), service providers count with an unprecedented flexibility, faster deployment and more efficient use of physical resources [1], [2]. With the evolution of virtualisation technologies through solutions such as containers, and the advent of Cloud Native (CN), these capabilities can be further enhanced, introducing practices coming from the IT world like DevOps, and giving great importance to development automation with methodologies such as Continuous Integration and Continuous Delivery (CI/CD).

The NFV architectural framework, defined by the European Telecommunications Standards Institute (ETSI) [3], conveys NFV's basic requirements and groups its architecture into three main blocks: the NFV Infrastructure (NFVI) managed by the Virtual Infrastructure Manager (VIM), the NFV Management and Orchestration (MANO) and the set of VNFs. Notice that this architectural framework does not specify any detailed implementation. Thus, many implementations of different blocks can be formed and are left to the vendor choice. Currently, several commercial and open-source solutions are available for each of these architectural blocks. On the one hand, VMware vCloud Director, OpenStack, and OpenVIM are some of the most extended VIMs. On the other hand, Open Network Automation Platform (ONAP), Open Source MANO (OSM) and Cloudify MANO are some open-source MANO's systems, while Ericsson Cloud Manager, and CloudBand (Nokia) are vendors' proprietary MANO's solutions. Finally, some examples of VNFs are DHCP servers, routers or firewalls.

Cloud Native refers to the approach to design, build and run applications and virtual functions that fully exploits the benefits of the cloud computing model. The CN approach refers to the way applications are created and deployed, and not only where they are executed [4]. CN is based on the principle of decomposing applications into a set of microservices that can be developed and deployed independently to accelerate and optimise DevOps lifecycle of software systems. These microservices are packaged into lightweight containers that are scheduled to run on compute nodes by a container orchestrator. To boost the benefits of CN, several associations and foundations have been formed. One of the most important is the Cloud Native Computing Foundation (CNCF) [5], which is an open-source software foundation under the umbrella of Linux Foundation (LF) promoting container technology and aligning the tech industry around its evolution.

Due to the evolution in virtualisation technologies and networking, the CN principles have also arrived in the telco domain, promoting a new paradigm known as Cloud Native telco. This change of perspective involves a radical approach to NFV, which is highly needed to satisfy the demands of 5G and 6G verticals [6]. It should be remarked that no single technology by itself will be able to meet all 5G Key Performance Indicators (KPIs) [7], but that not all of these requirements will be demanded by every 5G single application. Moreover, to satisfy the stringent latency needs of such 5G verticals, the application deployment is moving

978-1-6654-1278-0/21/$31.00 ©2021 IEEE

from powerful data centers to the network's edges through the Multi-access Edge Computing (MEC) paradigm [8]. While MEC enables much reduced latency, it is also characterised by more limited resources. For these reasons, Cloud Native in 5G systems has been identified as a major candidate for cost saving and improvement in system agility and service deployment. In this regard, ETSI has published specifications to enable MEC deployments in NFV environments, where MEC applications are deployed as VNFs, managed by ETSI NFV MANO components, along with an optional layer of intelligence providing network information for MEC apps optimisation. The virtualisation infrastructure is deployed as in NFV and managed by the VIM, allowing operators to deploy simultaneously MEC and NFV [9].

In this context, one of the MANO systems attracting most of the interest in the NFV industry is OSM. OSM is an ETSI-hosted initiative providing an open-source NFV MANO software stack entirely aligned with ETSI NFV specifications. OSM acts as NFV Orchestrator (NFVO) and is responsible for the deployment of NSs and VNFs. It also plays the role of Virtual Network Function Manager (VNFM) by monitoring the lifecycle management of virtualised network functions. Additional layers, such as service orchestration, are also required for operators to fully enable NFV services. Therefore, OSM is a very interesting tool, because it contains two of the key components of the ETSI NFV architecture and strictly follows the ETSI NFV stack. In view of its recent support for containers (since OSMr7), new NFV paradigms partially or fully composed of containers can be formed. OSM gives container support through Kubernetes (i.e., K8s) to deploy Container Network Functions (CNFs)[1] following two approaches as sketched in Figure 1:

1) **K8s embedded on a second VIM**: This method can be performed via two methods. The first method employs a K8s cluster running on Virtual Machines (VMs) inside the VIM and connecting all the VMs to the VIM network. Alternatively, the second method consists on spinning up a bare metal K8s cluster connected to the VIM network. In both options, NSs based on VNFs, CNFs, or even hybrid ones can be instantiated, giving high flexibility to the network operators. Currently the only VIM supported is OpenStack.
2) **Pure standalone K8s**: This approach follows a pure CN scenario, as no VMs or additional VIMs are needed in the solution. OSM directly interact with K8s, and NSs can be instantiated in the form of CNFs running leveraging containers. To do so, a K8s cluster with some OSM's addons needs to be set.

The first solution has the disadvantage that to use CNFs it is necessary to have an OpenStack cluster acting as an intermediary between OSM and K8s, which could be challenging due to the lack of resources or expertise in this complex cloud computing platform. This constraint could thus lead to many users being reluctant to employ CNFs. By contrast, in the second solution, a user can deploy CNFs without having OpenStack knowledge and/or resources. By leveraging only OSM and K8s, developers and users could work with CNFs in a more direct way, favoring a faster and more efficient advancement of container-based NSs. Nevertheless, the main issue of this last approach is the lack of validation. To the best of the authors' knowledge, at the time of writing no work in the literature can be found regarding the deployment and validation of this OSM-isolated K8s architecture.

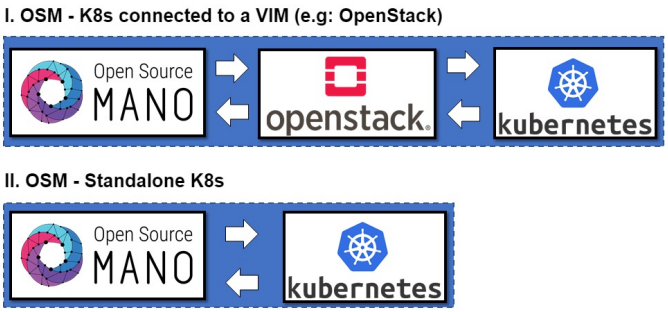

Fig. 1. OSM's support for the deployment of CNFs (via K8s).

In this context, this work has as main aim to introduce a proof of concept and validation of an standalone OSM-isolated K8s environment for CNF deployment. In particular, the contribution of this paper is threefold. First, we provide the details to enable native CNFs on this NFV architecture considering some of the gaps encountered, and validate and study its limits, advantages and drawbacks. Second, we discuss some of the issues encountered during this work and propose some solutions in order to better support OSM in standalone K8s deployments. Finally, we benchmark the CNFs performance using a OSM-K8s' testbed in comparison with an OSM-OpenStack environment, where VNFs are deployed.

The remainder of this paper is structured as follows. Section II discusses the related work. An analysis of the NFV evolution and the NFV architecture followed on this research is provided in Section III. Container support in OSM is discussed in Section IV along with some comments regarding its implementation. In Section V K8s-based VNFs are evaluated using some KPIs against a classic NFV scenario based on OSM and OpenStack. To conclude, some future work lines and our conclusions are drawn in Section VI.

II. Related Work

In the recent years, the promising performance argued by Docker containers as a lightweight virtualisation option has led to a huge body of literature aiming at analysing their capabilities in different contexts and deployment options [10], [11]. Similarly, several experimental studies have been conducted to evaluate container orchestration technologies, such as Kubernetes [12], as well as to compare the benefits of containers w.r.t. other virtualisation options such as VMs [13]–[16] and unikernels [17]–[20]. All these benchmarks demonstrate the

[1]NFs instantiated on containers receive either the name of Container Network Functions (CNFs) from 5GPPP side or Kubernetes-based virtual Network Functions (KNFs) from OSM's official documentation. Nonetheless, for being a more standard term, in this paper we refer to them as CNFs.

high performance of Docker applications in comparison to their competitors. While on the one hand, VMs is the per-excellence virtualisation option, container-based deployments are able to reduce startup and end-user serving time. Moreover, despite the attention attracted by Unikernels, its development, deployment and heavy debugging issues, have made this technology to stay behind containerised environments.

In the specific context of 5G systems, SDN, NFV and MEC have been identified as key cornerstones to enable efficient network management and orchestration. NFV and SDN have been proved essential in enabling network flexibility and programmability by decoupling control plane from user plane, and by allowing the detachment of network functions from the underlying hardware. In this respect, multiple works have study the usefulness of lightweight container solutions for deploying NFV environments and analysed how this choice determines network performance [16], [21], [22]. In [16] the authors address performance gaps associated with a VNF running on container and VM platforms (via KVM) in terms of CPU, memory consumption and deployment time. Similar conclusions are reach in [21] using LXC, Docker and rkt as container technologies. Different from the rest, the authors of [22] showcase a container-based 5G setup for CNF deployment of network services faster than NVFs. Nonetheless, the work is focused on performance prediction on the Kubernetes VIM but does not take into account a full MANO system [23].

Despite the high number of works focusing on virtualisation technologies, most of them take into account only cloud deployments where resource availability and power suppliant are usually insured. Nevertheless, with the emergence of edge computing to deploy latency-sensitive applications close to the end user, lightweight virtualisation technologies have become essential, given the more limited storage and performance capacity available [24], [25]. Moreover, a good part of the current research fail to consider MANO platforms or overlook the complexity of the whole orchestration system. In this respect, some limited efforts can be found in [26], which overviews several orchestration solutions from a theoretical point of view. Conversely, a full MANO platform evaluation is introduced in [27] by relying on two versions of OSM to perform orchestration tasks. Nonetheless, the authors do not consider light-weight virtualisation technologies and do not offer a comparison with other tools.

Some papers in the literature study both virtualisation technologies and MANO systems in a MEC context [28], [29]. The authors of [28] discuss that even though containerisation can bring greater advantages as a lightweight virtualisation technology, it is not mature enough to support the service migration required in several MEC scenarios to support user mobility in 5G networks and argue the need of more efficient orchestration tools. Conversely, the work in [29] evaluate two important MANO platforms such as OSM and Open Baton under diverse VIM environments such as OpenStack and AWS. Although Docker containers are also considered, the authors only analyse this environment under Open Baton, and no results are offered with respect to OSM.

Regarding NFV, some papers with a similar scope than this work have been recently published. On the one hand, the authors of [30] investigate and compare two of the most widely used MANO platforms, such as OSM and ONAP. The evaluation is performed in operational terms (resources footprint and deployment delay), and functional terms (supported VIMs, scaling support, etc.), by fixing OpenStack as the VIM and instantiating different VNFs that implement an open-source virtual Customer Premises Equipment (vCPE). However, no containerised functions, and hence CNFs, are taken into consideration in this work. On the other hand, similar to our work, the authors of [31] showcase some of the first use cases on CNFs using the 5GTANGO service platform [32] in a smart manufacturing domain. However, this last work is limited to a validation of the 5GTANGO along with K8s, without actually getting to study the behaviour of this system nor comparing it with the classical scenario based on VNFs, which would have helped to make tangible how beneficial the adoption of containerisation in NFV could be.

Nevertheless, to the best of the authors' knowledge, there is no work in the literature introducing a proof of concept and benchmarking of Kubernetes-based deployment for CNFs orchestrated by the ETSI-hosted OSM. The work presented in this paper is aligned with the ETSI-NFV architectural framework and takes OSM as the MANO platform to natively deploy CNFs. To this end, a pure standalone Kubernetes deployment is setup in OSM (meaning that no intermediate VIM such as OpenStack is used), in which Kubernetes provides the functionality of the VIM itself, and both OSM and the NSs are deployed via containers. Furthermore, the performance of this architecture has been compared with the classical one, the deployment relying on a VM-based VIM (i.e., in this case, Openstack) running NSs (via VNFs) in VMs.

III. ETSI NFV VISION

A. NFV Evolution

The telco sector is moving towards Cloud Native due to an evolution in the underlying technologies. The immediate example is the use of containerisation as a natural evolution from classic virtualisation. However, this process will take place gradually. VNFs running on VMs will still be used in production for some time due to the lack of maturity of the container technology from the telecommunications point of view (highlighting networking and security) and the recent equipment expenditure from operators to adapt to VMs scenarios, which still have to be made profitable. In this context, to make Cloud Native telco a reality, a transition from VNFs to CNFs must take place. This is well reflected in [33], which conveys the point of view of the European Commission and the European ICT industry, and provides some guidelines on how this technology will evolve in the near future.

This evolution can be seen in Figure 2, which showcases the path from the classic solution based on VNF implementations running on VMs to a new architecture based on CNF implemented on containers. As stated above, this shift is a process that cannot be done immediately, but instead a gradual

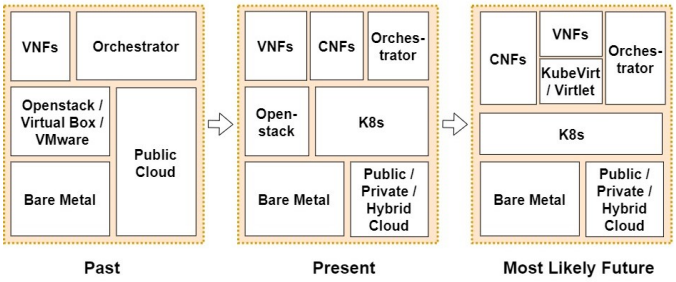

Fig. 2. Evolution of the virtualisation solutions used in telcos.

evolution is required. Therefore, we are currently experiencing a period of coexistence between VMs and containers that will last for a few years. More specifically, the various phases of this evolution can be described as follows:

- **Past.** The classic solution is based on VMs running on bare metal or public clouds. Although this approach has coped with the services demanded in 4G and lower generations, it falls short to meet the requirements of 5G in terms of latency, scalability and ultra-reliability. AWS, Microsoft Azure, or Google Cloud are examples of relevant public cloud providers. Hypervisors such as VMware or VirtualBox have been widely adopted. Openstack has been used as the de-facto cloud computing platform.
- **Current status.** The only feasible approach for the Cloud Native Telco is to offer solution enabling the evolution from VNFs (e.g., a Firewall, a DHCP server, etc.) running on VMs to CNFs leveraging containers. It mirrors how enterprises are moving their monolith architectures based on VMs to containers orchestrated by K8s and then refactoring them into microservices. Besides the gradual evolution required, many legacy solutions cannot be implemented yet with K8s, reaching a coexistence period in terms of infrastructure between VMs and containers.
- **Future.** The most likely future approach is based entirely on container cluster management. As a matter of fact, the CNFC has adopted Kubernetes as the default open-source management tool for micro-service oriented applications. Therefore, no longer room would remain for OpenStack. To address uses cases where VMs are a must (where some VNFs have not been ported to CNFs yet or cannot be ported), some software such as KuberVirt or Virtlet could be used on top of Kubernetes.

The path towards orchestration including containerisation states the basics of edge computing, which is another key concept in 5G. Edge computing, in a nutshell, argues the deployment of applications and services closer to the users to mitigate the latency and bandwidth limitations of today's communications. To reach a landscape with more distribution across multi-clouds and the inclusion of heterogeneous infrastructures, orchestration platforms with container support have become a must. The tree main orchestration frameworks for both VMs and/or containers promoted for the edge domain by 5GPPP are Kubernetes, OSM and ONAP [34].

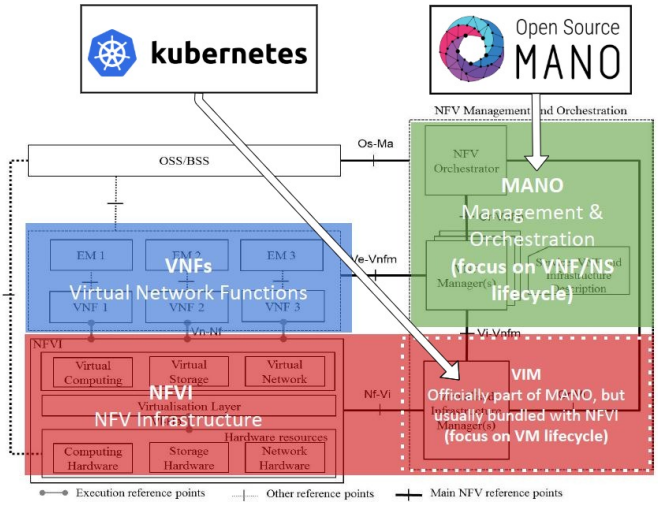

Fig. 3. NFV architecture layout implemented by OSM and K8s.

B. Our NFV Architecture

The ETSI NFV architectural framework provides the functional definition of the components involved but it does not specify any implementation approach [3]. Therefore, different combinations forming different architectures are possible. In this context, benchmarking works targeting the evaluation and analysis of the implementation choices are very useful and interesting contributions. The NFV network model studied in this paper is reflected in Figure 3 along with the ETSI NFV architecture. This model is divided into three main blocks, namely the VNFs themselves, the NFVI and the MANO layer.

At the MANO layer, the NFVO is implemented using OSM, which is responsible for the deployment of NS and CNFs on top of containers. Furthermore, OSM also acts as VNFM, and it is in charge of monitoring the lifecycle management of these virtualised network functions. The main reason for selecting OSM as the MANO platform is due to its maturity, performance, low resource print and wide adoption [30].

At the NFVI side, a K8s cluster manages the computing, storage and network resources, acting as both NFVI and VIM components. This decision was given by the fact that K8s is currently the only container management tool supported by OSM. Moreover, as previously mentioned, K8s is considered a major candidate to be the future orchestrator in B5G networks by both academia and industry because of its vast capabilities and flexibility. However, current NFV implementations make a tiny use of the real potential of this orchestration engine, since MANO systems consider K8s as a "dumb" NFVI [34].

The next section provides further details on the implementation details considered to naively enable the deployment of CNFs taking as a reference the architecture presented above.

IV. NFV Architecture for CNF Support

A. K8s support in OSM

As introduced in Section I, there exist two approaches to deploy a Kubernetes cluster along with OSM. The first

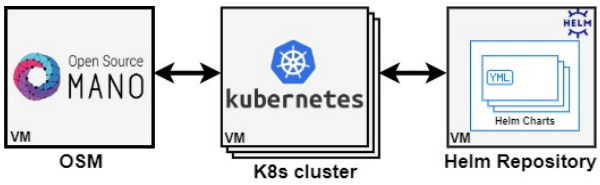

Fig. 4. Implementation of the OSM-Standalone K8s architecture

method entails a K8s cluster connected to a second VIM (e.g., Openstack), which in turn is also connected to OSM. This setup could be implemented via two methods: the first one consist of a K8s cluster running on VMs inside the VIM and connecting them to the VIM network, while he second method relays on spinning up a baremetal K8s cluster and connecting it again, to the VIM network. The second method regards a pure standalone K8s approach. In this pure NFV CN scenario (based only on containers) no additional VIM between OSM and K8s is needed. Therefore, network operators can deploy CNFs without requiring an operative extra VIM, which could be challenging due to resources and/or expertise.

Taking into account its very recent support for K8s environments, OSM is more inclined in the first phase of this adoption towards the deployments based on connecting K8s with an additional VIM. The reason for this is that this option is easier to take to a production environment from the current status (i.e., VNFs deployments using VIMs such as Openstack), since its use helps manage networks and facilitates the connection to the infrastructure [35]. Therefore, this option has available a richer documentation and different tested use cases.

By contrast, the standalone K8s approach is currently in a much more immature state and has less documentation resources. Given that OSM is an open-source solution, its progress is partly due to the activities of its community. Such activities include validation tasks and feature testing, which allows discovering and fixing issues and bugs. This is also one of the motivations for investigating and analysing this architecture, and that aims to contribute to the evolution of OSM with respect to lightweight virtualisation techniques.

Therefore, this work has as main objective to introduce a proof of concept and validation of a pure standalone OSM-K8s deployment in an NFV architecture in which CNFs can be instantiated. According to the official documentation of OSM, it is advised to deploy K8s associated with a VIM, giving the impression that they do not consider K8s as a VIM. Nevertheless, this is debatable, since K8s can be considered a VIM itself due to its functionality and capabilities. Furthermore, it should be stressed that the use of an intermediate VIM between OSM and K8s breaks the Cloud Native vision.

B. Setting a pure Cloud Native Testbed: OSM-Standalone K8s

The goal of this setup is to implement and validate the container-based NFV architecture represented in Figure 3. The implementation apart from the NFV components, OSM and K8s, includes a third component, a helm chart repository, as it is shown in Figure 4. This repository, was added in order to be able to store and test our personalised K8s apps. This architecture is based on OSM Release 8 (OSM-8), a K8s cluster (K8s v1.19.2) and a Helm Chart repository (Helm v2).

In the helm chart repository component is where K8s apps, in the form of helm charts, are stored. A helm chart is a collection of files that describe a related set of Kubernetes resources [36]. In a few words, a helm-chart could be defined as a large YAML file that fully describes the K8s deployment. It is important to highlight that the container image specified in the helm chart is not stored in the helm chart repository. Helm charts are at the same level than K8s, while container images are at a lower one, being stored in Docker registries. Therefore, it is an interesting option to use the same node as helm chart repository and Docker registry.

In the implementation of the NFV CN testbed, a K8s-all-in-one cluster has been set. This means that the same node acts as master node and worker node. This works main aim is to test the NFV CN architecture and therefore overlooks concepts such as high availability and disaster recovery that would be a must in a production environment (that would have of course a greater number of master, etcd and worker nodes). For instance, adding more worker nodes to the cluster would be as easy as installing kubelet (K8s node agent) in a new machine and adding it to the cluster by using a token signed by the master node. By default, a K8s' cluster does not schedule pods on the control-plane node for security reasons. To allow it, it is necessary to untaint the master following the steps specified in K8s official documentation [37].

Figure 5 presents a sequence diagram of the main steps required to implement a OSM-Standalone K8s environment. Note that this UML diagram includes the operations for both the *preparation of the NFV environment* and the *instantiation of a NS formed by CNFs*, and that will be detailed below.

Preparation of the NFV environment. This phase includes the tasks for the creation of the K8s cluster and the deployment of OSM to ensure a successful connection and communication both solutions. On the one hand, the process of adding some OSM addons to the K8s cluster: (i) a load balancer (i.e., metallb); and (ii) a storage class [38]. For Kubernetes clusters >v1.15, it is also needed a special permission of Tiller, which is the server portion of helm (in Helm v2). On the other hand, some steps in OSM must be also performed. First of all, due to current OSM requirements, to be able to work with a OSM-K8s isolated cluster, a dummy VIM must be added due to the lack of native support for Kubernetes. This dummy VIM does not perform any task and it is simply a OSM's VIM definition that contains dummy information in the different fields. It is reasonable to think that in future OSM releases this step will be deleted. Secondly the credentials and the location of the K8s cluster need to be provided to OSM, and thirdly, the details of the helm-chart repository must be also included.

Instantiation of a NS formed by CNFs. OSM is at this point ready to instantiate container-based NFs. In the deployment process, OSM communicates with the K8s cluster, which downloads the helm-charts specified in the CNF Descriptors (CNFD) and implements the configuration defined in them. An

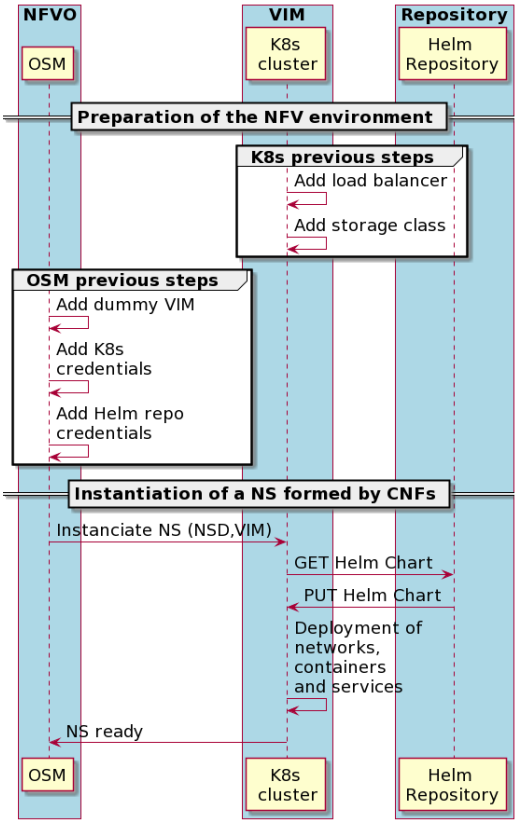

Fig. 5. Sequence diagram of the main steps while setting the CN testbed.

```
vnfd-catalog:
    schema-version: '3.0'
vnfd:
-   connection-point:
    -   name: mgmt
    id: cirros_knfd
    name: cirros_knfd
    short-name: cirros_knfd
    description: KNF with a helm-chart that contains
                 one K8s deployment (1 pod and 1 rs)
                 running a CirrOs image.
    vendor: i2CAT
    version: '1.0'
    mgmt-interface:
        cp: mgmt
    k8s-cluster:
        nets:
        -   external-connection-point-ref: mgmt
            id: mgmtnet
    kdu:
    -   helm-chart: i2cat-helm-repo/i2cat-cirros
        name: i2cat-cirros
```

Fig. 6. Example of a CNFD written in YAML

example of a CNFD written in YAML is provided in Figure 6. In the last part of the descriptor the helm chart repository and the helm chart name are specified, where information such as container images, network configuration, Role-Based Access Control (RBAC) policies, etc. is provided.

C. OSM-Standalone K8s architecture: lessons learnt

As stated earlier, the progress of OSM is partly due to the activities of its community. In this respect, the deployment and implementation given in the previous section has allowed identifying various issues regarding the instantiation of CNFs in pure CN OSM-K8s environments, that are listed below.

- **Lack of native support in the standalone Kubernetes approach.** To deploy this environment a dummy VIM that does not perform any task needs to be registered to OSM. Nonetheless, note that at the beginning of this work, no information regarding this dummy step was reflected in OSM's official documentation.
- **Lack of support for more than one K8s cluster.** OSMr8 does not support the selection of the K8s cluster where the NSs formed by CNFs are deployed. While on a scenario with a single K8s cluster OSM perfectly handles the deployment of container-based NFs, on a multi-k8s cluster it raises an error as it is not able to select the specific cluster.
- **Lack of information in errors related to Helm charts.** If any error on the helm chart or the helm chart repository in the CNFD occurs, OSMr8 does not provide any details about the error and limits it to a notification such as: "`Error: failed to download <Helm-Repo>/<Helm-Chart>`". Therefore, debugging becomes a very hard task as the error could come from different sources. Manual debugging on the OSM's pod is neither a solution, as due to permission issues is not possible to debug at such level. A solution for OSM could be to include the flag `--debug` in its internal helm-related commands. With it, a more complete error-reporting is provided, specifying the cause of the error.

V. Performance evaluation. CNFs vs VNFs

The performance evaluation reported in this section aims not only to validate the pure CN NFV framework for CNFs introduced in this work but also to compare it with the performance achieved by VNFs running on VMs. To this end, below we present the methodology followed (including testbed setup, images used, and experimentation details) and discuss the results obtained.

A. Methodology

1) Testbed setup: Two testbeds implementing the ETSI NFV architecture have been used for benchmarking in this work. The first testbed is based on native OSM-K8s, while the second one builds on OSM-OpenStack. The installation details and resources used by the components required by the two testbeds are specified in Table I. In particular, the MANO layer (including VNFM functionalities) of both testbeds is based on OSM. At the NFVI-VIM side, each testbed differs: the first one follows the containers philosophy, while the second one is based on a VM-driven architecture, which has been set for benchmarking purposes. Figure 7 graphically displays the configuration of the two aforementioned testbeds.

2) Images used: To fairly and equitably compare the behaviour of VNFs and CNFs, and hence of both testbeds, it is required to use an image with similar size and performance requirements. However, not many images meeting these two

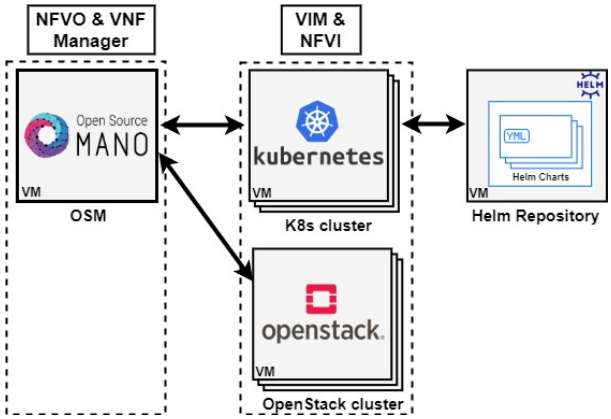

Fig. 7. Benchmarking testbeds: OSM-K8s & OSM-OpenStack.

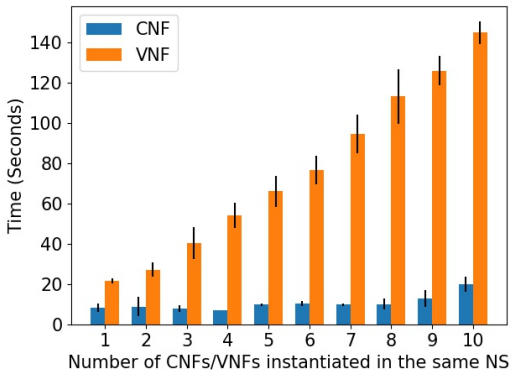

Fig. 8. NSIT of a NS for an increasing number of VNFs/CNFs.

TABLE I
INSTALLATION DETAILS & RESOURCES USED BY THE TWO TESTBEDS

	Installation details & resources used	
Comp.	**Testbed 1: OSM-Standalone K8s**	**Testbed 2: OSM-OpenStack**
OSM	OSM v8.0.1, installation based on K8s. Installed in a VM running Ubuntu 18.04 with 2 vCPUS, 6GB of RAM and 60GB of storage	
K8s	K8s v1.19.2 all-in-one cluster installed with kubeadm. Installed in a VM running Ubuntu 18.04 with 2 vCPUS, 8GB of RAM and 60GB of storage	-
Helm Chart Repo	Helm v2.20. VM running Ubuntu 18.04 with 2 vCPUS, 8GB of RAM and 60GB of storage	-
Open-Stack	-	Devstack v5.3.1 all-in-one cluster. Installed in a VM running Ubuntu 18.04 with 2 vCPUS, 8GB of RAM and 60GB of storage

conditions are available for both VMs and containers. Container images, due to its nature, use to have lighter weight. For instance, Ubuntu's 18.04 image in cloud format weights more than 300MB, while the docker image takes around 30MB. Therefore, it is not a good candidate for a fair benchmarking. On the other hand, CirroS images [39], which are minimal Linux distributions designed for testing, are available in both formats and have similar size (12.6MB in docker image format and 15.58MB in cloud format). Hence, these images guarantee a fair analysis and benchmarking for VNFs/CNFs. It should be also noted that the results of the performance evaluation are subject to the images and the resources used in the scenarios.

3) Experimentation details: To perform the analysis under a controlled environment in both testbeds and evaluate their behaviour when the complexity of the NSs grows, a set of NSs composed of a varying number of VNFs/CNFs has been previously onboarded in OSM. To this end, the number of replicas for each NS (i.e., the number of VNFs/CNFs forming each NS) has been increased at each iteration mimicking the scaling up process in a NS on this type of systems. We remind the reader that the VNFs/CNFs forming those NSs implement the same function, i.e., running CirroS cloud/-docker images, respectively. The number of VNFs/CNFs per NS is increased until they are composed of 10 VMs/containers, respectively. The maximum number is given by the constraint of our OpenStack testbed in which no more than 10 VMs can be simultaneously instantiated, as it will be detailed in Section V-B3. Once the onboarding is completed adding all the VNF/CNF and NS descriptors to OSM, the helm chart described in the CNF descriptors needs to be stored in the helm chart repository. Assuming that the steps described in Figure 5 have been performed without issues, the NSs are ready to be instantiated. Each experiment (i.e., deployment of NSs, increasing its VNFs/CNFs) has been repeated 4 times.

B. Performance Analysis

The next subsections discuss the tests performed for benchmarking using some KPIs proposed in the literature such as Network Service Instantiation time (NSIT), CPU and RAM consumption, and maximum number of VNFs/CNFs supported with the same resources.

1) KPI 1 - Network Service Instantiation Time (NSIT): It defines the time needed to deploy and instantiate the VNFs/CNFs of a NS [40]. This process involves the provision, instantiation and configuration of the VMs/containers. The creation of virtual links (if necessary) is also considered. These actions follow the specifications provided in the VNFs/CNFs and NS descriptors. This time is a key indicator as it reflects the time necessary to scale a NS, or even to re-instantiate it due to any issue. This metric is collected from the NFVO, which has a global view of the underlying NSs state. The idea behind this KPI is also considered in other related works [30], where the authors separate this concept into Deployment Process Delay (DPD) and On-boarding Process Delay (ODP).

Figure 8 showcases the NSIT taken by different NSs with regard to the number of VNFs/CNFs. Each VNF/CNF contains a single Virtual Deployment Unit (VDU)/Kubernetes deployment Unit (KDU). Its noticeable the huge difference between the NSIT taken by NSs based on VNFs (running on VMs) and NSs based on CNFs (running on containers). For instance,

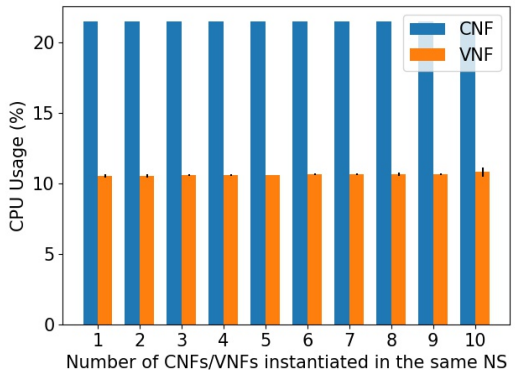

Fig. 9. CPU usage of a NS for an increasing number of VNFs/CNFs.

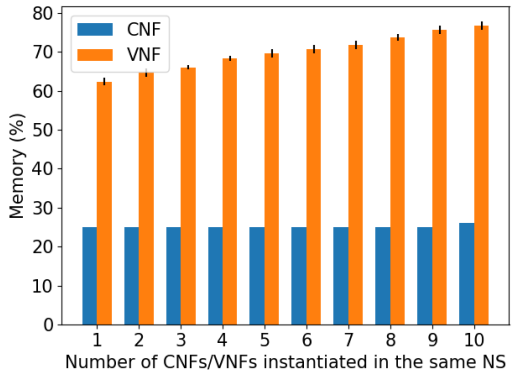

Fig. 10. RAM Usage of a NS for an increasing number of VNFs/CNFs.

when the NS is based on 1 CNF/VNF, the VNF takes more than 2.5 times to be up and running. This difference increases at each iteration, and with 10 VNFs/CNFs per NS, the VNF version takes more then 7 times longer than the CNF one. Notice that in the first iteration the docker image needs to be downloaded before instantiating the CNF, which imposes an extra time that is not present in the next experiments.

2) KPI 2 - CPU and RAM consumption: These are some of the must-do KPIs to be analysed in this type of infrastructures. In Figure 9 the percentage of CPU usage in both clusters is depicted, where it is clear that by default the K8s cluster consumes more CPU than the Openstack one. When instantiating the different NSs, the CPU used by each testbed evolves differently. The OpenStack testbed (in which VNFs are deployed) consume each iteration more CPU, even though no task is run with them, and is caused by the resource-allocation constraint of VMs. Conversely, containers use the resources that they need and, if they are not running any task, no impact is caused on the resources of the node where they are running. Note that, if required, K8s also allows the reservation and limitation of resources for a container using resource quotas. These parameters have been collected in both VIMs.

In terms of RAM consumption, Figure 10 shows how the OpenStack cluster consumes 60% of the memory only to run the OpenStack services. Moreover, with 10 VNFs running, the OpenStack cluster is almost at the limit, whereas the K8s cluster is almost unaffected. The K8s cluster consumes 25% of RAM to run the containers that implement the K8s architecture, and it only increases to 26% running 10 CNFs in parallel. Furthermore, these tests can be considered as a functional validation of both testbeds since they present logical results due to the nature of the type of virtualisation used.

3) KPI 3 - Maximum number of VNFs/CNFs supported simultaneously: As both Openstack and K8s all-in-one cluster's machines have the same resources, and the simulations have been run with images with similar size, a comparison between the maximum number of NFs running simultaneously in both testbeds (VNFs and CNFs respectively) is an interesting idea. Due to the nature of the virtualisation technique of each testbed, it is clear that the CNFs that can be instantiated in the K8s cluster outnumber the VNFs in the Openstack. Assuming that each CNF runs on a pod and each VNF runs on a VM, a 116-10 ratio is obtained, as shown in Table II.

TABLE II
MAX. NUMBER OF NFS RUNNING SIMULTANEOUSLY IN EACH TESTBED

Testbed	Maximum simultaneous instances
OSM-Standalone K8s	116 CNFs
OSM-OpenStack	10 VNFs

The main factor in favour of CNFs is that containers can consume less than 1vCPU if needed, e.g., a container that needs only 0.6 vCPU to run. VMs, on the other hand, need at least 1 vCPU to be instantiated. Thus, CNFs allow saving system resources and have a direct impact on CAPEX and OPEX. For the same resources per server, the number of applications that can be instantiated is 100 times higher, allowing users to deploy several NSs in parallel, or forming complex VNF-Forwarding Graphs composed of different NSs. However, coping with scenarios where a high volume of containers are interconnected could be a hard task due to the networking and troubleshooting complexity. To address this, a concept called service mesh is of great benefit. Open platforms such as Istio, implement service mesh, allowing users to achieve a reliable and secure communication between containers, facilitating service-to-service communications between microservices.

VI. CONCLUSION

The deployment of CNFs on lightweight devices provides to the NFV world an unprecedented flexibility and a crucial role, boosting the 5G verticals KPIs, such as in automotive, smart manufacturing, etc. Despite ETSI's efforts to adapt to CN, due to the early support of containers in the NFV architecture, there are still no stand-alone NFV environments in the literature that support pure CNFs. In this context, the main objective of this work is to introduce a proof of concept and validation of the NFV OSM-Standalone K8s architecture, which is fully aligned with CN principles as it is exclusively container-based. Facilitating this deployment aims also to encourage the use of CNFs from operators and developers. The results obtained provide a functional validation and demonstrate the advantages

w.r.t. VM-based deployments in terms of instantiation time, concurrent CNFs, and RAM/CPU consumption, which have a direct impact on CAPEX and OPEX.

As a future work, both NFV testbeds will be deployed in more resourceful bare-metal servers to analyse their behaviour in an environment closer to production. Moreover, we will perform the benchmarking with real network services with more complex and heterogeneous images, such as a firewall or a DHCP service with real traffic. At the time of writing, a new version of OSM was released (OSMr9), enabling resource orchestration in different scenarios and improved VNF/CNF lifecycle management supporting Helm 3. In this regard, future research work will include upgrading both testbeds with OSMr9 and benchmarking using Helm 3.

ACKNOWLEDGMENT

This work was supported by the European Union's H2020 research and innovation programme under the CARAMEL project (Grant agreement No.833611) and the 5GCity project (ref.no. TEC2016-76795-C6-2-R). This work was also supported by the Spanish National Project ONOFRE-2 (ref.no. TEC2017-84423-C3-3-P) and by the TRUE5G project (PID2019-108713RB-C51/AEI/10.13039/501100011033) funded by the Spanish National Research Agency.

REFERENCES

[1] ETSI, "Network Functions Virtualisation (NFV), Network Operator Perspectives on Industry Progress," European Telecommunications Standards Institute, Tech. Rep. NFV White Paper 02, Oct. 2013.

[2] R. Mijumbi, J. Serrat, J. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, pp. 236–262, 2015.

[3] ETSI, "Network Functions Virtualisation (NFV); Architectural Framework," December 2014.

[4] Cloud Native Computing Foundation (CNCF). CNCF Cloud Native Definition v1.0, https://github.com/cncf/toc/blob/master/definition.md, February 2020, Accessed on 01.03.2021.

[5] CNCF, Cloud Native Computing Foundation, https://www.cncf.io/, Accessed on 01.03.2021.

[6] 5GPPP, "5G Verticals," https://5g-ppp.eu/vertical, Accessed on 01.03.2021.

[7] 5GPPP, "5G KPIs," https://5g-ppp.eu/kpis, Accessed on 01.03.2021.

[8] ETSI, "MEC in 5G networks," European Telecommunications Standards Institute, Tech. Rep. ETSI White Paper 28, Jun. 2018.

[9] ETSI, "Mobile Edge Computing; Deployment of Mobile Edge Computing in an NFV environment," Sophia Antipolis, France, Feb 2018.

[10] R. Morabito, "A performance evaluation of container technologies on Internet of Things devices," in *Proc. of IEEE INFOCOM WKSHPS*, San Francisco, CA, USA, 2016.

[11] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, and R. Ranjan, "A Holistic Evaluation of Docker Containers for Interfering Microservices," in *Proc. of IEEE SCC*, San Francisco, CA, USA, 2018.

[12] A. Pereira Ferreira and R. Sinnott, "A Performance Evaluation of Containers Running on Managed Kubernetes Services," in *Proc. of IEEE CloudCom*, Sydney, Australia, 2019.

[13] A. Lingayat, R. R. Badre, and A. Kumar Gupta, "Performance Evaluation for Deploying Docker Containers On Baremetal and Virtual Machine," in *Proc. of IEEE ICCES*, Coimbatore, India, 2018.

[14] T. Salah, M. J. Zemerly, C. Y. Yeun, M. Al-Qutayri, and Y. Al-Hammadi, "Performance comparison between container-based and VM-based services," in *Proc. of IEEE ICIN*, Paris, France, 2017.

[15] R. K. Barik, R. K. Lenka, K. R. Rao, and D. Ghose, "Performance analysis of virtual machines and containers in cloud computing," in *Proc. of IEEE ICCCA*, Noida, India, 2016.

[16] D. Gedia and L. Perigo, "Performance Evaluation of SDN-VNF in Virtual Machine and Container," in *Proc. of IEEE NFV-SDN*, Verona, Italy, 2018.

[17] I. Mavridis and H. Karatza, "Lightweight Virtualization Approaches for Software-Defined Systems and Cloud Computing: An Evaluation of Unikernels and Containers," in *Proc. of IEEE SDS*, Rome, Italy, 2019.

[18] R. Behravesh, E. Coronado, and R. Riggio, "Performance Evaluation on Virtualization Technologies for NFV Deployment in 5G Networks," in *Proc. of IEEE NetSoft*, Paris, France, 2019.

[19] T. Kurek, "Unikernel Network Functions: A Journey Beyond the Containers," *IEEE Communications Magazine*, vol. 57, no. 12, pp. 15–19, 2019.

[20] J. B. Filipe, F. Meneses, A. U. Rehman, D. Corujo, and R. L. Aguiar, "A Performance Comparison of Containers and Unikernels for Reliable 5G Environments," in *Proc. of IEEE DRCN*, Coimbra, Portugal, 2019.

[21] J. Struye, B. Spinnewyn, K. Spaey, K. Bonjean, and S. Latre, "Assessing the value of containers for NFVs: A detailed network performance study," in *Proc. of IEEE CNSM*, Tokyo, Japan, 2017.

[22] S. Hirai, T. Tojo, S. Seto, and S. Yasukawa, "Automated Provisioning of Cloud-Native Network Functions in Multi-Cloud Environments," in *Proc. of IEEE NetSoft*, Ghent, Belgium, 2020.

[23] M. Gawel and K. Zielinski, "Analysis and Evaluation of Kubernetes Based NFV Management and Orchestration," in *Proc. of IEEE CLOUD*, Milan, Italy, 2019.

[24] J. Darrous, T. Lambert, and S. Ibrahim, "On the Importance of Container Image Placement for Service Provisioning in the Edge," in *Proc. of ICCCN*, Valencia, Spain, 2019.

[25] Z. Yu, J. Wang, Q. Qi, H. Sun, and J. Zou, "A Boundless Resource Orchestrator Based on Container Technology in Edge Computing," in *Proc. of IEEE ICCCN*, Hangzhou, China, 2018.

[26] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On Multi-Access Edge Computing: A Survey of the Emerging 5G Network Edge Cloud Architecture and Orchestration," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1657–1681, 2017.

[27] M. Peuster, M. Marchetti, G. García-de Blas, and H. Karl, "Automated testing of NFV orchestrators against carrier-grade multi-PoP scenarios using emulation-based smoke testing," *EURASIP Journal on Wireless Communications and Networking*, vol. 172, 2019.

[28] T. V. Doan, G. T. Nguyen, H. Salah, S. Pandi, M. Jarschel, R. Pries, and F. H. P. Fitzek, "Containers vs Virtual Machines: Choosing the Right Virtualization Technology for Mobile Edge Cloud," in *Proc. of IEEE 5GWF*, Dresden, Germany, 2019.

[29] N. Slamnik-Kriještorac, M. Peeters, S. Latré, and J. M. Marquez-Barja, "Analyzing the impact of VIM systems over the MEC management and orchestration in vehicular communications," in *Proc. of IEEE ICCCN*, Honolulu, HI, USA, 2020.

[30] G. M. Yilma, Z. F. Yousaf, V. Sciancalepore, and X. Costa-Perez, "Benchmarking open source NFV MANO systems: OSM and ONAP," *Computer Communications*, vol. 161, pp. 86 – 98, 2020.

[31] S. Schneider, M. Peuster, K. Hannemann, D. Behnke, M. Müller, P.-B. Bök, and H. Karl, ""Producing Cloud-Native": Smart Manufacturing Use Cases on Kubernetes," in *Proc. of IEEE NFV-SDN*, Dallas, TX, USA, 2019.

[32] G. Tango, "5G TANGO," https://www.5gtango.eu/, Accessed on 25.02.2021.

[33] 5GPPP Software Network Working Group, "Cloud Native and 5G Verticals' services," 5GPPP, Tech. Rep., Feb. 2020.

[34] 5GPPP Technology Board WG. 5G-IA's Trials Working Group, "Edge Computing for 5G Networks," 5GPPP, Tech. Rep., Jan. 2021.

[35] OSM official doc., "5.6. Using Kubernetes-based VNFs (KNFs)," https://osm.etsi.org/docs/user-guide/05-osm-usage.html#using-kubernetes-based-vnfs-knfs, Accessed on 01.03.2021.

[36] Helm official doc., https://v2.helm.sh/, Accessed on 01.03.2021.

[37] K8s official doc., https://kubernetes.io/, Accessed on 01.03.2021.

[38] OSM official doc., "ANNEX 7: K8s installation & req." https://osm.etsi.org/docs/user-guide/15-k8s-installation.html, Accessed on 25.02.2021.

[39] CirrOs' Image. DockerHub., https://hub.docker.com/_/cirros, Accessed on 25.02.2021.

[40] Project S. Xilouris (NCSRD) G. (Ed.). D6.3 Final Demonstration roadmap and Validation Results (2018), http://sonatanfv.org/content/d63-final-demonstration-roadmap-and-validation-results, Accessed on 25.02.2021.