


RESEARCH

Open Access



# Automated testing of NFV orchestrators against carrier-grade multi-PoP scenarios using emulation-based smoke testing

Manuel Peuster<sup>1\*</sup> , Michael Marchetti<sup>2</sup>, Gerardo García de Blas<sup>3</sup> and Holger Karl<sup>1</sup>

## Abstract

Future large-scale network function virtualization (NFV) environments will be based on hundreds or even thousands of NFV infrastructure installations, the so called points of presence (PoP). All their resources and the services deployed on top of them will be controlled by management and orchestration (MANO) systems. Such large-scale scenarios need to be automatically tested during the development phase of a MANO system. This task becomes very challenging because large-scale NFV testbeds are hard to maintain, too expensive, or simply not available. In this paper, we introduce the concept of emulation-based smoke testing, which enables automated, large-scale testing of MANO systems. We show that our test platform prototype can easily emulate up to 1024 PoPs on a single physical machine and that it can reduce the setup time of a single test PoP by a factor of 232× compared to a DevStack-based PoP installation. In a case study, we test and compare two versions of a state-of-the-art MANO solution, namely ETSI's Open Source MANO (OSM), in large-scale scenarios using our prototype. The issues we found in this study would not have been discovered with existing, lab-scale test environments.

**Keywords:** NFV, Management and orchestration, Testing, Emulation, Automation, Multi-PoP scalability

## 1 Introduction

Network softwareization and its underlying technologies, like software-defined networks (SDN) and network function virtualization (NFV), are one of the key concepts of the upcoming fifth generation of networks (5G). Those technologies are expected to introduce a new level of agility into our networks, including the on-demand deployment of new network services within a few minutes or even seconds [1]. In addition, these software-based services are expected to move towards the network edge and execute on top of many small, spatially distributed cloud infrastructure installations, the so called NFV infrastructures (NFVI). Such deployments are called multi-point-of-presence (multi-PoP) environments, where each point of presence (PoP) provides NFVI resources as well as a virtual infrastructure manager (VIM), like an OpenStack installation [2], which offers interfaces to request

resources, e.g., start a virtualized network function (VNF). Multiple of these VNFs are then chained together, possibly across multiple PoPs, to build more complex network services, a concept called service function chaining (SFC).

The key component in such NFV environments is the management and orchestration (MANO) system that controls the deployment and configuration of individual VNFs as well as complex SFCs. A MANO system can connect to multiple PoPs by consuming the north-bound interfaces of their VIMs and has a global view of the NFV infrastructure under its control. Several of these MANO systems are currently developed, some as commercial products, others as open-source community or research projects, like SONATA [3], ONAP [4], or Open Source MANO (OSM) [5].

The complex and distributed nature of these multi-PoP scenarios lead to a big challenge for the development of MANO systems, which obviously need to be tested against such large-scale environments. But large multi-PoP environments are costly, hard to set up, and usually just not available to MANO developers. Even if they were available, it is often too expensive to use

\*Correspondence: [manuel.peuster@upb.de](mailto:manuel.peuster@upb.de)

<sup>1</sup>Computer Networks Group, Paderborn University, Warburgerstr. 100, 33098 Paderborn, Germany

Full list of author information is available at the end of the article

them in automated test pipelines for continuous integration (CI) setups, which would occupy resources whenever a developer submits code. This raises the question how to efficiently test MANO systems in large multi-PoP environments.

In this paper, which extends our previous work [6], we present a solution for this: an emulation-based test platform that emulates multiple NFVI environments on a single machine, enabling automated tests of MANO systems in large multi-PoP scenarios. The presented solution is inspired by a concept called *smoke testing* [7] which focuses on testing only the main functionalities of a system and skips unimportant details to reduce test times (Section 1.1.2).

Our contributions are as follows: First, we introduce our novel *emulation-based smoke testing* concept in Section 2.1 before presenting the prototype of our open-source multi-PoP test platform in Section 2.2. This prototype consists of two parts: the emulation platform, which is based on [8], and a novel test suite for MANO systems compliant to European Communications Standards Institute (ETSI)'s SOL005 standard [9] that is presented in this paper and made available as open-source project [10]. Both parts can be integrated into a fully automated CI pipeline as discussed in Section 2.2.3. Next, we analyze the scalability of our platform and show how we can emulate up to 1024 NFVI PoPs (10 times as much as in our previous work [6]) on a single physical machine or virtual machine (VM) in Section 3.1. Finally, we present a case study in which we test and compare OSM rel. THREE and OSM rel. FOUR [5] using our platform and test suite. In this case study, presented in Section 3.2, we discovered some interesting insights and bugs that would not have been found with existing, lab-scale NFVI testbeds offering only a handful of PoPs. We discuss these insights in Section 4 and conclude in Section 5.

## 1.1 Background

Before presenting our solutions, we first analyze components and interfaces required to test MANO systems and give deeper insights into the *smoke testing* concept.

### 1.1.1 Management and orchestration for NFV

MANO systems are complex software systems and represent the main control entity in NFV-enabled network infrastructures. Besides basic lifecycle management (LCM) tasks, MANO systems are also responsible for performing more complex orchestration tasks, like scaling, self-healing, or failover management [11]. Those tasks are often automated and part of a closed control loop, which uses monitoring data as inputs to trigger orchestration decisions based on pre-defined policies or service-/function-specific management algorithms [12]. To provide all these functionalities, MANO systems

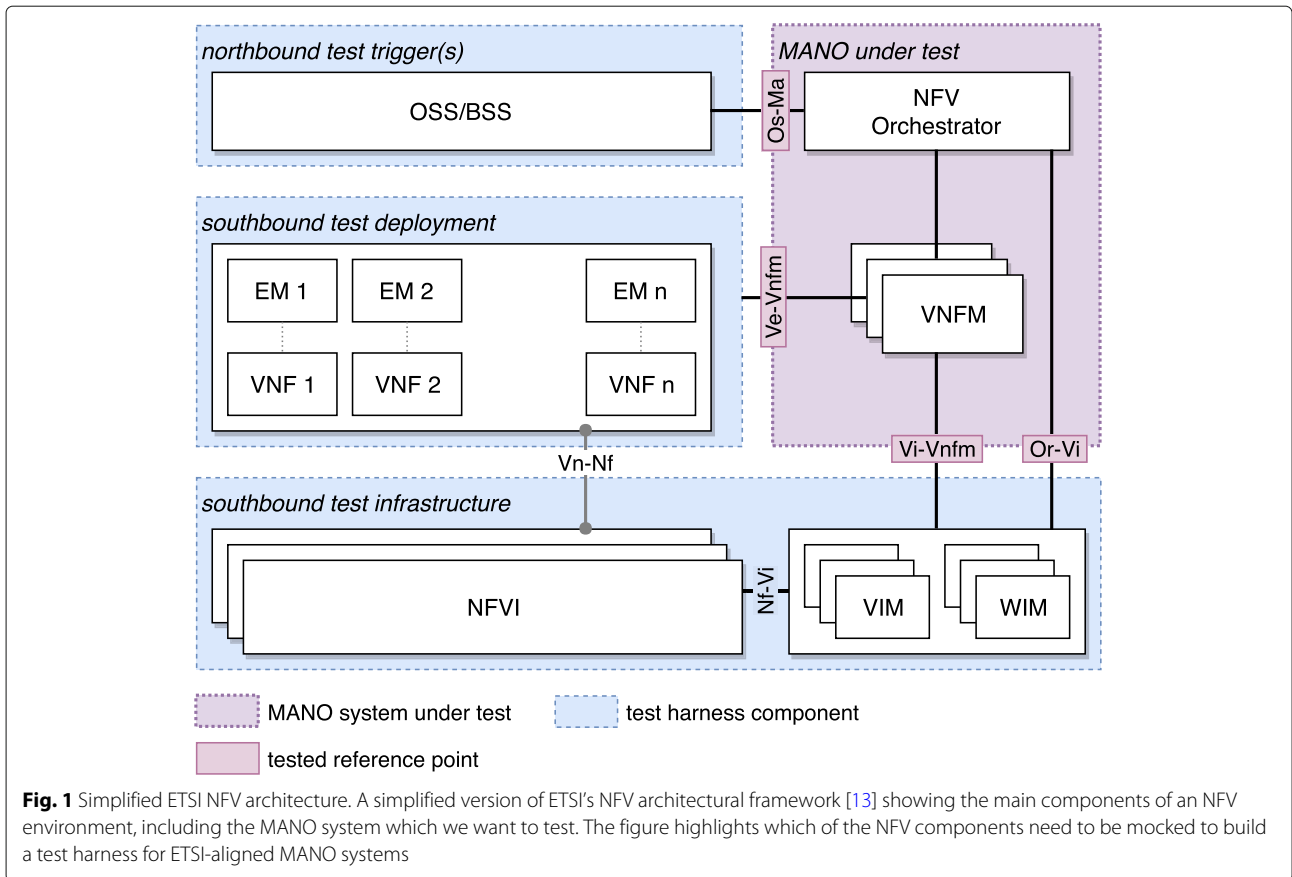
usually interface with a high number of external components, as shown in Fig. 1. The figure shows a simplified version of ETSI's NFV architectural framework [13] and highlights the MANO system and its interfaces to external components.

In general, the interfaces of a MANO system can be categorized into northbound and southbound interfaces. The northbound interfaces are those interfaces used by service providers, platform providers, or operation/business support systems to trigger LCM actions, like service instantiation. They are consolidated within the *Os-Ma* reference point in the ETSI architecture (Fig. 1). The southbound interfaces of a MANO system are considered to be those interfaces that connect to the underlying NFVI and the corresponding management components, like VIMs and WAN infrastructure managers (WIMs). Those interfaces are part of the *Vi-Vnfm* and *Or-Vi* reference points. In addition, the interfaces that connect to the instantiated VNFs and services, e.g., for configuration and monitoring tasks, are also considered part of the southbound interfaces of a MANO system. They are represented by the *Ve-Vnfm* reference point.

When looking at this complex environment, it becomes clear that testing MANO systems in isolation, e.g., using unit tests, is not sufficient to ensure that they behave as desired. More specifically, testing solutions are needed that efficiently test the interoperability of a given MANO system in different environments, e.g., a high number of connected VIMs in multi-PoP scenarios. Our proposed solution offers exactly this by providing a lightweight *test harness* for MANO systems. Figure 1 shows which of the components in the ETSI architecture need to be mocked to build a full test environment for a MANO system. The first component contains the *test triggers* which connect to the northbound interface of a MANO system and trigger OSS/BSS actions. The second—and most important—component of the test harness is the *test infrastructure*, which is connected to the MANO's southbound interface and can be used by the MANO system to test NFV deployments without requiring one or more full-featured NFVI installations. Those deployments are the third component of the test harness, called *test deployments*, for example, lightweight NFV services or service mockups.

### 1.1.2 Smoke testing

The term *smoke testing* was originally introduced by the electrical engineering community and describes a very basic test to see if the tested device catches fire (smokes) after it is plugged into a power source. Later, the term *smoke testing* was taken up by the software testing community and used to describe rapid tests that verify that the most basic but critical functions of a system work as they should [14]. They are also called *build verification tests* and should be executed whenever a new build of a



system (or of a subcomponent of that system) becomes available. They can be considered as a preliminary testing stage that is used to qualify builds for further, more complex tests, like regression or integration tests. The important thing here is that smoke tests do not substitute regression or integration tests which are still needed to test every detail of a system. The main goal of smoke tests is to ensure that the basic functionality of a software product works, e.g., the program can be started and the default usage path does something meaningful. Using this, broken builds with major bugs are rejected early before more time and resource intensive tests are deployed and executed [7].

We noticed that those smoke testing concepts perfectly match the problem of testing complex NFV MANO systems where testing suffers under the high resource demands of end-to-end tests due to the needed NFVI infrastructures. Our main idea is to use a more lightweight NFV environment, including a very lightweight NFVI, that allows to test the basic functionalities of a MANO system, e.g., service on-boarding and initial instantiation, before testing the MANO system and all its detailed functionality in a full-fledged NFV environment, which might not even be available to each individual developer of a MANO system. Section 2.1 presents our smoke testing concepts for NFV in more detail.

### 1.2 Related work

Automated testing of NFV deployments is still a novel research direction with a limited amount of solutions. Most of them focus on testing NFVIs and their corresponding data planes or the corresponding VIMs, e.g., the test tool collection of OPNFV with projects like Yardstic, Functest, or NFVperf [15]. They neither consider testing of VNFs, complex network services, nor MANO solutions, which makes those solutions complementary to our work. Some recent work focuses on end-to-end testing in 5G networks [16] or the verification and validation of network services and network function [17]. Even though [17] considers the case of applying integration tests in the NFV domain to test interoperability between different VNFs, none of them explicitly considers the need of testing the core part of NFV deployments: the MANO system. In the software engineering community, smoke testing has already been established since several years, providing the ability to quickly integrate new versions of different software components [7], which is what our solution introduces for NFV MANO systems.

Another related research direction are automated performance tests of either VNFs, network services, or NFVIs. A handful of solutions have been proposed for performance testing of VNFs with the goal to characterize

their performance behavior under different configurations or in different environments [18, 19]. Some solutions focus more on end-to-end performance tests for complete services, like [20], arguing that testing the performance of a single VNF in isolation does not yield representative results. All of these solutions require an end-to-end deployment of the tested VNFs and services during their tests, but none of them does these deployments with a real, production-ready MANO system. They all use custom-tailored, often hardcoded, deployment scripts for their solutions, making them unsuitable for MANO testing.

A straight-forward solution to setup those NFVIs for testing is to use testbed installations. Testbeds can either be installed locally, e.g., lab-scale installations, or third party testbeds can be used remotely. Keller et al. [21] propose a locally installed multi-cloud testbed based on a handful of physical machines, each representing a single cloud site, i.e., a small OpenStack installation. Those machines are then interconnected, and traffic shaping solutions are added to emulate realistic delays between the sites. The problem with local installations, like [21], are their resource limits which prevent large-scale test cases, e.g., with a high number of PoPs. Remote testbeds, like [22–24], may offer the required NFV infrastructure and interfaces, but their main focus is the development, experimentation, and evaluation of network services, rather than being infrastructure for automated test pipelines. Most of their infrastructure deployments and management functionalities are fixed, e.g., the used SDN controllers, VIMs, and MANO solutions, offering limited space for custom-tailored MANO tests. In addition, they are shared between many users which means they may not always be available to quickly execute automated tests on them. In general, these testbed solutions are complementary to our presented approach and should be used for final, manually deployed integration tests rather than for automated smoke testing.

Another option for automated smoke tests is using locally available network emulation approaches, like Mininet [25], CORE [26], or VLSP [27]. Unfortunately, these solutions focus on prototyping and evaluation of new protocols or network management paradigms rather than on interactions with production-ready MANO solutions. None of these solutions offers de-facto standard VIM northbound interfaces for easy MANO system integration, like our solution does with its OpenStack-like interfaces. Even if VLSP focuses on MANO-like experiments in the NFV domain, it lacks the ability to execute real-world VNF software, which is possible in our platform that uses lightweight container solutions to run VNFs in an emulated environment.

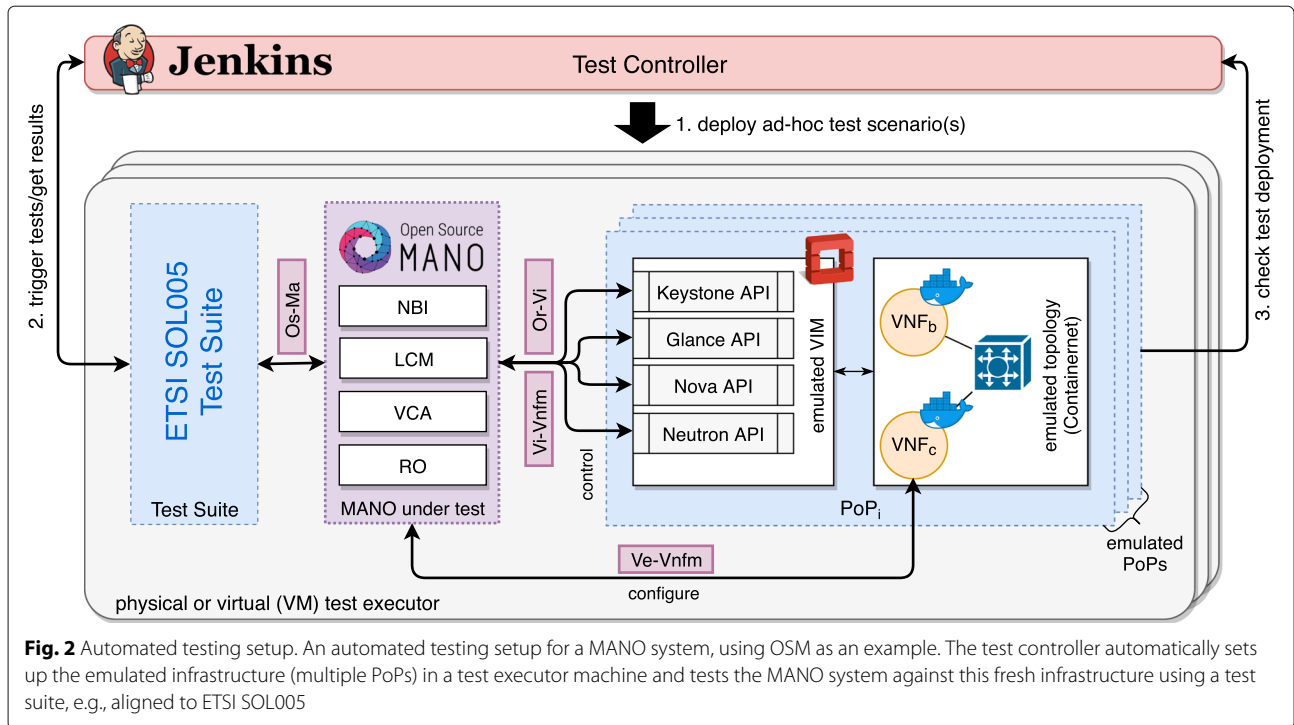
## 2 Methods

We introduce *emulation-based smoke testing*, which substantially reduces the required resources to perform realistic test scenarios with real-world MANO solutions as the key concept behind our solution (Section 2.1). After that, we present our prototype in Section 2.2.

### 2.1 Emulation-based smoke testing

Emulated test infrastructure provides some major benefits when compared to a real NFV multi-PoP deployment, e.g., based on OpenStack. First, the state of emulated VIMs and NFVIs can be made volatile, which ensures that tests are always executed in a clean environment. For example, there are no zombie VMs left in a PoP resulting from a former test execution in which the used environment and infrastructure was not cleaned properly. Such a cleanup would take much longer with real-world VIM systems and might even require their reinstallation and reinitialization. Second, the setup of an emulation platform can be expected to be much quicker and the needed resources are far less than for a full-featured VIM, e.g., an OpenStack installation and the configuration of the attached compute, storage, and networking infrastructure. More importantly, an emulation platform can even be executed on a single machine (physical or VM), making it a much better fit for existing test pipelines, e.g., based on Jenkins [28]. It also allows parallelization by using multiple VMs, each containing its own emulated NFV deployment. Third, emulated infrastructure can be easily scaled to hundreds (or even thousands) of PoPs, whereas a fully automated setup of hundreds of interconnected OpenStack installations is very challenging and may be even infeasible in a short time, as we show in Section 3.1.

Figure 2 shows the proposed testing setup in which a test controller, e.g., Jenkins [28] or a simple shell script, automatically sets up an environment that emulates a pre-defined multi-PoP topology (1). This setup can either be done on a physical machine or a VM, the so-called *test executor*. Once this is done, the test controller configures the MANO system to be tested and connects it to the VIM interfaces of the emulated PoPs. In the figure, we use OSM as an example for a MANO system under test; we emphasize again that other MANOs can be used. After that, the test controller triggers the test cases against the MANO's northbound interface (2), e.g., deploying a test service. To do so, either custom test suites or pre-defined standard-compliant test suites, e.g., our test suite for ETSI NFV's SOL005 [9] MANO northbound interface specification, introduced in Section 2.2.2, may be used. Those tests should trigger the main functionalities of a MANO system, starting from VNF and service on-boarding, followed by browsing the elements of a MANO's catalog, to the instantiation and scaling of a VNF or a service. By doing so, the MANO system is tested end-to-end. Once



a test service is instantiated, the test controller checks if the resulting deployments and configurations on the emulated infrastructure, done by the MANO system during the tests, are correct (3). For example, it checks if the number of VNFs deployed on the PoPs can be retrieved and if the correct configuration values have been applied to them. Once all tests are done, the test controller destroys the emulated infrastructure by stopping the emulation environment and freeing the test executor machine. It can then start a new emulation instance, e.g., with a different multi-PoP topology, for further tests.

As expected, there are also a couple of limitations when using an emulation-based infrastructure for testing. First, not all features of the original OpenStack APIs will be supported by an emulated platform. This behavior is intentional and helps to achieve the goal of a very lightweight substitution of a full-featured NFV infrastructure. In our prototype implementation, presented in the next sections, we focused on the API endpoints required to let typical MANO solutions, like OSM, believe that they talk to a real OpenStack installation, namely the OpenStack *Keystone*, *Nova*, *Glance*, and *Neutron* endpoints. Nevertheless, new endpoints can easily be added to our prototype. Second, an emulated infrastructure will not be able to deploy VNFs as full-blown VMs; instead, it is limited to lightweight container technologies, like Docker in our prototype. This limitation is required to keep the emulation lightweight and to be able to run it on a single machine and execute the test cases within seconds rather than within minutes

or hours. Third, the total available resources of the emulated infrastructure is limited. However, the lightweight emulation design still allows to emulate hundreds of PoPs as shown in Section 3.1.

These limitations must be kept in mind when using our *emulation-based smoke testing* concept in a testing pipeline. In general, *emulation-based smoke tests* should not be considered as a full replacement of a final integration test against a real multi-PoP environment but as a much faster, intermediate testing stage that can easily be executed for each new commit to the MANO system's code base—something that is certainly not feasible with existing setups based on real-world PoP testbed installations.

### 2.2 Prototype

We built a prototype of the described testing platform to validate our design and to check the feasibility of the proposed testing approaches. The core of our prototype is based on our open-source emulation platform that was initially designed to rapidly prototype network services or experiment with different MANO solutions on a local machine [8] as described in Section 2.2.1. After extending the emulation platform to support large-scale MANO test scenarios with many emulated PoPs, we added a test suite for MANO systems to it (Section 2.2.2). Finally, we integrated the entire system with existing testing solutions to be able to automatically run them within existing CI pipelines as shown in Section 2.2.3.



### 2.2.1 Multi-PoP emulation platform

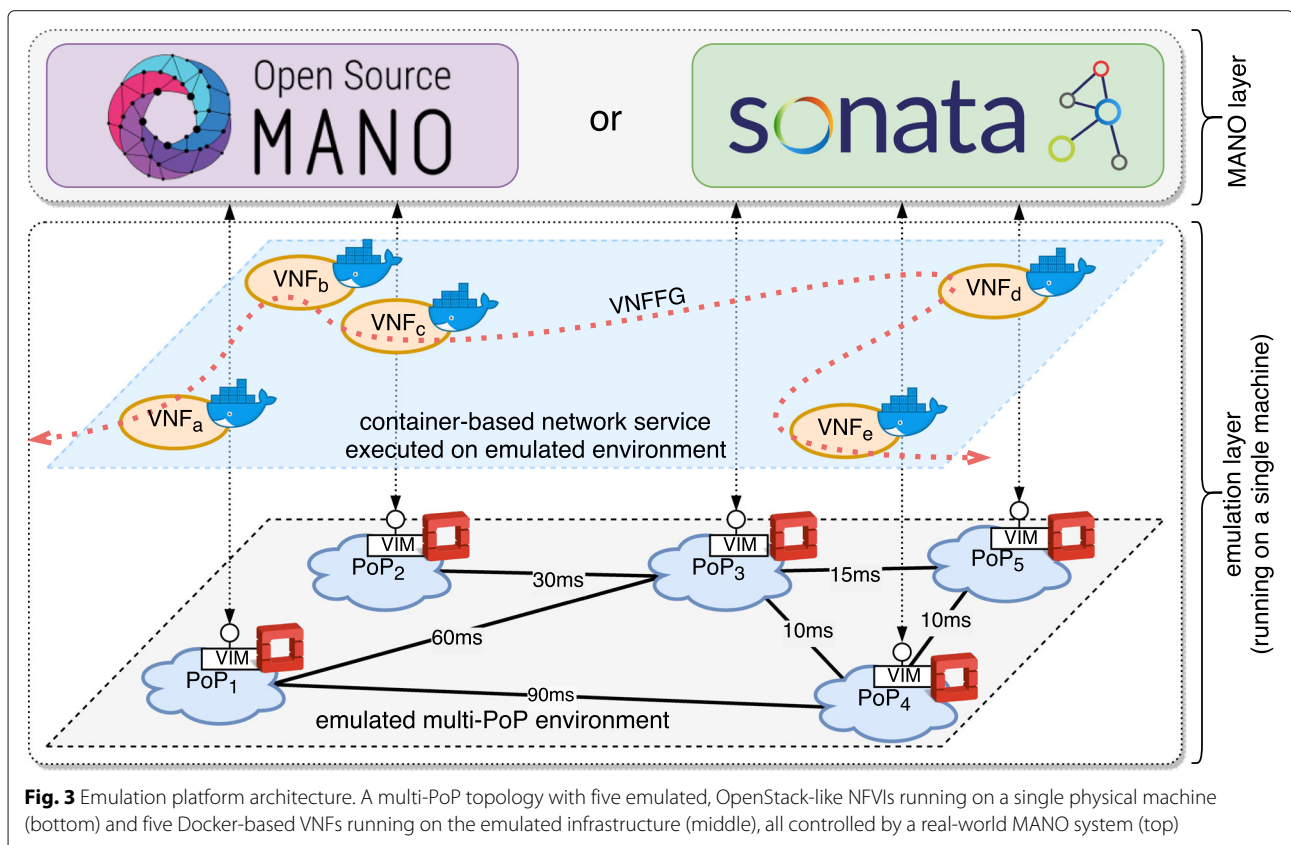
Our emulation platform consists of three main components as shown in Fig. 3: first, the network emulation part, which is based on Containernet [29], a Mininet extension [25], and shown as the bottom layer in the figure. Containernet allows to execute network functions inside Docker containers that are connected to arbitrary, user-defined network topologies [8]. Those topology definitions are based on Python scripts, and the network links can have parameters (like delay, loss, and data rate) which are used to artificially delay, drop, or rate-limit the traffic between the Docker containers of the emulated network.

The second part of our platform is the *VIM emulation* part which creates an abstraction layer for the network emulation and lets a user define arbitrary topologies with emulated NFVI PoPs instead of single networking hosts. Each of these emulated NFVI PoPs then represents a single VIM endpoint and allows to deploy, terminate, and configure VNFs executed inside the emulated PoP. This allows the emulation platform to emulate realistic, distributed NFVI deployments, e.g., by adding artificial delays to the links between the PoPs. We utilize this to allow the emulator to automatically load topologies from the *Internet Topology Zoo* (ITZ) library [30], as we show in Section 3.2. The *VIM emulation* layer deploys or terminates single VNFs, in form of Docker containers, inside

each of the emulated PoPs at runtime, just like it would be possible in a full-featured, cloud-based PoP. Once the VNFs are running, traffic can be steered through multiple VNFs by using the emulator’s chaining API to support service function chaining (SFC) [8] for the test deployments.

The third part, which is one of the main contributions of this paper, are additional APIs on top of the emulation platform. These APIs mimic the original OpenStack APIs for each of the emulated PoPs and translate OpenStack requests, e.g., `openstack compute start`, into requests that are executed by the emulation platform, e.g., start a Docker-based VNF in one of the emulated PoPs. We opted to mimic the OpenStack APIs because OpenStack is currently the de-facto standard VIM and supported by most MANO systems. However, all these API endpoints are designed as small, pluggable components and can easily be replaced by endpoints that mimic the APIs of other VIM solutions.

Figure 3 shows a usage scenario in which our emulation platform (bottom layer) emulates five interconnected PoPs, each offering its own OpenStack-like northbound API. This emulated infrastructure can be controlled by any real-world MANO system that is able to use OpenStack, e.g., OSM [5] or SONATA [3] (top layer). The MANO system is used to instantiate a complex, distributed network service, consisting of five VNFs, on



**Fig. 3** Emulation platform architecture. A multi-PoP topology with five emulated, OpenStack-like NFVIs running on a single physical machine (bottom) and five Docker-based VNFs running on the emulated infrastructure (middle), all controlled by a real-world MANO system (top)

top of the emulated infrastructure (middle layer). With this setup, the emulated infrastructure and the instantiated services look like a real-world multi-PoP NFVI deployment from the perspective of the MANO system consisting of multiple data centers deployed at different geographical locations.

### 2.2.2 Standard-compliant MANO test suite

Once the emulated NFVI is up and running and the MANO system that is supposed to be tested is installed, running, and configured, everything is ready for test execution. The only missing piece in such a setup are the actual test cases as well as mechanisms to invoke the tested MANO system during the tests. One option to implement test cases for this scenario is to create test cases that are custom-tailored to the MANO system under test. This approach makes a lot of sense if very specific aspects of a single MANO solution should be tested, e.g., a proprietary management interface. However, the goal of NFV is to establish an open environment with well-documented and standardized interfaces. An example for this is the *Os-Ma-Nfvo* reference point defined by ETSI [13] and its interface specification ETSI NFV-SOL005 [9]. Even though ETSI recently established a special task force (STF) activity with the goal to build API conformance tests for their NFV interface specifications [31], the resulting tests are not yet complete, neither they are able to work against real-world MANO implementations, like OSM.

Motivated by this, we started to design a standardized test suite for ETSI's *Os-Ma-Nfvo* reference point, implemented it as part of our prototype, and released it under Apache 2.0 license [10]. To make this test suite as reusable as possible, we used a two-layered design. The top layer, which is based on Python's `unittest` library, implements the abstract test logic according to the written interface specifications of ETSI SOL005. Those tests then call the bottom layer of our test suite which contains pluggable connection adapters, abstracting MANO-specific connection details that are not part of the interface specification, e.g., authentication mechanisms. Our prototype comes with an example MANO adapter that supports OSM rel. FOUR and uses OSM's client libraries to access OSM's northbound interface.

Table 1 presents an overview over the implemented tests. It shows different operations of the tested interfaces grouped by the resources they manipulate. The table also shows the availability of each interface endpoint in ETSI SOL005 and its implementation status in OSM rel. FOUR. Some endpoints, e.g., the endpoints to manipulate the VIMs that are connected to a MANO system, are only available in OSM's interface but not in ETSI's specification. Those differences to the written specification usually originate from additional requirements of practical implementations. Other endpoints, e.g., network

**Table 1** Interfaces covered in our test suite (TS)

	ETSI	OSM	TS	Runtime
Resource: VIMs				
Create		•	•	1.31 s ± 0.31 s
List		•	•	2.58 s ± 0.54 s
Resource: Single VIM				
Show		•	•	1.26 s ± 0.37 s
Update		•		
Delete		•	•	1.16 s ± 0.33 s
Resource: NSDs				
Create	•	•	•	0.52 s ± 0.07 s
List	•	•	•	0.55 s ± 0.04 s
Resource: Single NSD				
Show	•	•	•	0.54 s ± 0.09 s
Update	•	•		
Delete	•	•	•	0.53 s ± 0.11 s
Resource: VNFD				
Create	•	•	•	0.24 s ± 0.06 s
List	•	•	•	0.40 s ± 0.07 s
Resource: Single VNFD				
Show	•	•	•	0.24 s ± 0.06 s
Update	•	•		
Delete	•	•	•	0.23 s ± 0.06 s
Resource: NS				
Create	•	•	•	9.35 s ± 0.51 s
List	•	•	•	9.56 s ± 0.43 s
Resource: Single NS				
Show	•	•	•	9.44 s ± 0.44 s
Update	•			
Scale	•	•		
Create Alarm		•		
Export Metric		•		
Heal	•			
Terminate	•	•	•	9.44 s ± 0.43 s
Resource: VNF				
List		•	•	9.81 s ± 0.48 s
Resource: Single VNF				
Show		•	•	9.67 s ± 0.42 s

service healing, are defined by ETSI but are not yet available in OSM. To keep the table short, it does not show the *network service performance and fault management* interfaces defined by ETSI, since they are not yet available in OSM. In general, our open-source test suite already contains tests for all major endpoints of OSM; additional

ones, like the scaling endpoint, are in preparation and will evolve together with the available endpoints of the tested MANO solutions. Finally, the table presents average run-times of each test further described in Section 3.2.3.

### 2.2.3 CI pipeline integration

One of the key points in modern software testing is *automation*. Today, most software projects, including MANO system projects, use CI approaches to automatically execute tests whenever a developer commits new code to the code base. Those tests are organized in so-called test pipelines that start with static code style checks, continue with detailed unit tests, and end with basic integration tests between the project's components. Once all these tests passed, the resulting software artifacts have to be tested in more complex environments to check their compatibility with external components, e.g., different VIM solutions, to find integration issues.

The main problem of those complex tests is the required test infrastructure, e.g., to setup multiple OpenStack-based VIMs and to maintain them. Another problem with those tests is their scalability: Even if some lab-scale OpenStack installations are available, they can only be used to execute a limited number of test cases at a time that easily becomes a bottleneck if the number of developers and thus the number of contributions increases. A common solution for this is to reduce the frequency of complex tests by not executing them for each new commit, but only once a day. At this point, our emulation-based smoke testing solution can help and improve the test workflow because it can be used as an intermediate test stage between frequent basic tests and complex integration tests in real environments.

More specifically, our emulation-based solution provides some characteristics which make it a perfect fit for a frequent execution in CI pipelines. First, the entire platform can be started and configured with a single command. This includes the setup of hundreds of emulated VIMs, which is not feasible with real-world VIM solutions. Second, our platform always starts in a clean state. There is no need to manually cleanup the environment after a test has been executed: whenever the emulation platform is restarted, all deployed services, the underlying emulated infrastructure, and all emulated VIMs are removed, and new ones are started. Third, the emulator can be packaged and executed within a Docker container (nested Docker deployment) or a virtual machine which make distribution, initial setup, and integration with existing test environments easy. It also allows highly parallelized test setups because multiple VMs, each running one emulation platform instance, can be deployed on an existing test infrastructure and used completely independently from each other. This feature should be particularly helpful for multi-branch test pipelines. Finally, the

resource footprint of the emulation platform is very small, and it can be (re-)started within seconds (or minutes if hundreds of PoPs should be emulated) as we show in the following section.

## 3 Results

The evaluation of the proposed smoke testing concepts and our platform prototype can be split into two parts. First, we evaluated the scalability of our emulation platform in Section 3.1 using the same approach as in [6] but using scenarios ten times larger to push the platform to its limits. Second, we conducted a case study using OSM as a state-of-the-art MANO solution and tested it against our platform using real-world topologies in Section 3.2. In this case study, we not only tested two major release of OSM, namely OSM rel. THREE and OSM rel. FOUR, and compared them, but also analyzed the runtimes of our novel, ETSI-compliant test suite executed against OSM.

### 3.1 Emulation platform scalability

To get a first idea about the setup time savings that can be expected from emulated PoPs, we compared the setup times of our emulation platform configured to emulate a single OpenStack-like PoP with the setup times of a single-node OpenStack DevStack [2] installation, which can be considered as the most simple way to install a fully featured OpenStack in a PoP. We executed both setup procedures 10 times on a single physical machine with Intel(R) Core(TM) i5-4690 CPU @ 3.50 GHz and 16 GB memory and found a mean setup time for a single emulated PoP of 2.48 s compared to a mean setup time of 576.42 s for a fresh DevStack installation, which is more than 232 times slower. This comparison makes sense since we want to ensure that we always test against a clean environment, and thus, a fresh installation of DevStack would be always required.

Users of production-ready, carrier-grade MANO systems, like OSM, are not only interested that the MANO system works well with a single PoP but expect that these systems scale well with the number of attached PoPs and the number of deployed services. To test this, our emulation-based approach is a perfect fit since it is able to emulate many PoPs and allows to deploy many lightweight services on the emulated infrastructure. All this can be done on a single machine, whereas similar DevStack-based testbed installations would require much more resources, i.e., 2 CPU cores, 4 GB memory, and about 15 GB disk space per PoP [2].

To quantify the scaling abilities of our emulation platform, we did a set of experiments to study its behavior when topologies with many PoPs are emulated or when hundreds of service instances are deployed on the emulated infrastructure. This experiment and all following experiments have been executed on a single physical



machine with Intel(R) Xeon(TM) E5-1660 v3 CPU with 8 cores @ 3.0 GHz and 32 GB memory and have been repeated 10 times. In the first experiment, we analyzed the startup and configuration time of the emulation platform for different synthetic topologies with different numbers of PoPs. Figure 4 shows the setup time breakdown for up to 1024 PoPs using four topologies. It shows how much time is used by which of the four phases of the emulation setup procedure: initialization, PoP setup, link setup, and emulation start. The *linear* topology connects all PoPs into a long chain, the *star* topology connects all PoPs to a single central PoP, and the two randomized (*rnd*) topologies get the number of PoPs  $|V|$  and a factor  $k$  as inputs. They then interconnect the PoPs with  $|E| = k|V|$  links where  $|E|$  is the number of created links. Those  $|E|$  links are picked uniformly at random from the set of possible links between all involved PoPs. All error bars in this paper show 95 % confidence intervals.

The results show that in all topologies, 128 PoPs can be set up in between 91.8 and 197.7 s, which is a huge improvement when compared to 128 DevStack installations. Even the maximum tested number of 1024 PoPs can, on average, be created in 3,704.0 s using the *rnd*( $k=0.5$ ) topology. The results of the randomized

topologies indicate that the number of links which have to be established in the topology has a non-negligible impact on the overall setup time. Further, the plots indicate a non-linear relationship between number of PoPs and total setup times. We identified the Open vSwitch daemon (*ovs-vswitchd*), which runs always on a single CPU core, to become the bottleneck in large deployments as it has to manage one vSwitch instance per PoP.

We also analyzed the memory consumption for these four topologies and directly compared their total setup times (Fig. 5). The figure shows that the total memory used by the tested environment increases proportionally to the number of PoPs in the topology. In general, not more than 5 Gb of memory is used, even with large topologies, which shows that our emulation platform can easily be executed on existing test nodes or locally on a developer’s laptop.

Finally, we studied the time required to deploy a large number of VNFs on top of the emulated infrastructure. We again used our *liner*, *star*, *rnd*( $k=0.5$ ), and *rnd*( $k=1.5$ ) topologies with either 8 or 128 PoPs and deployed up to 256 VNFs on those PoPs (randomly placed). The used VNFs are based on the default Docker `ubuntu:trusty` images and do not run any additional software, since

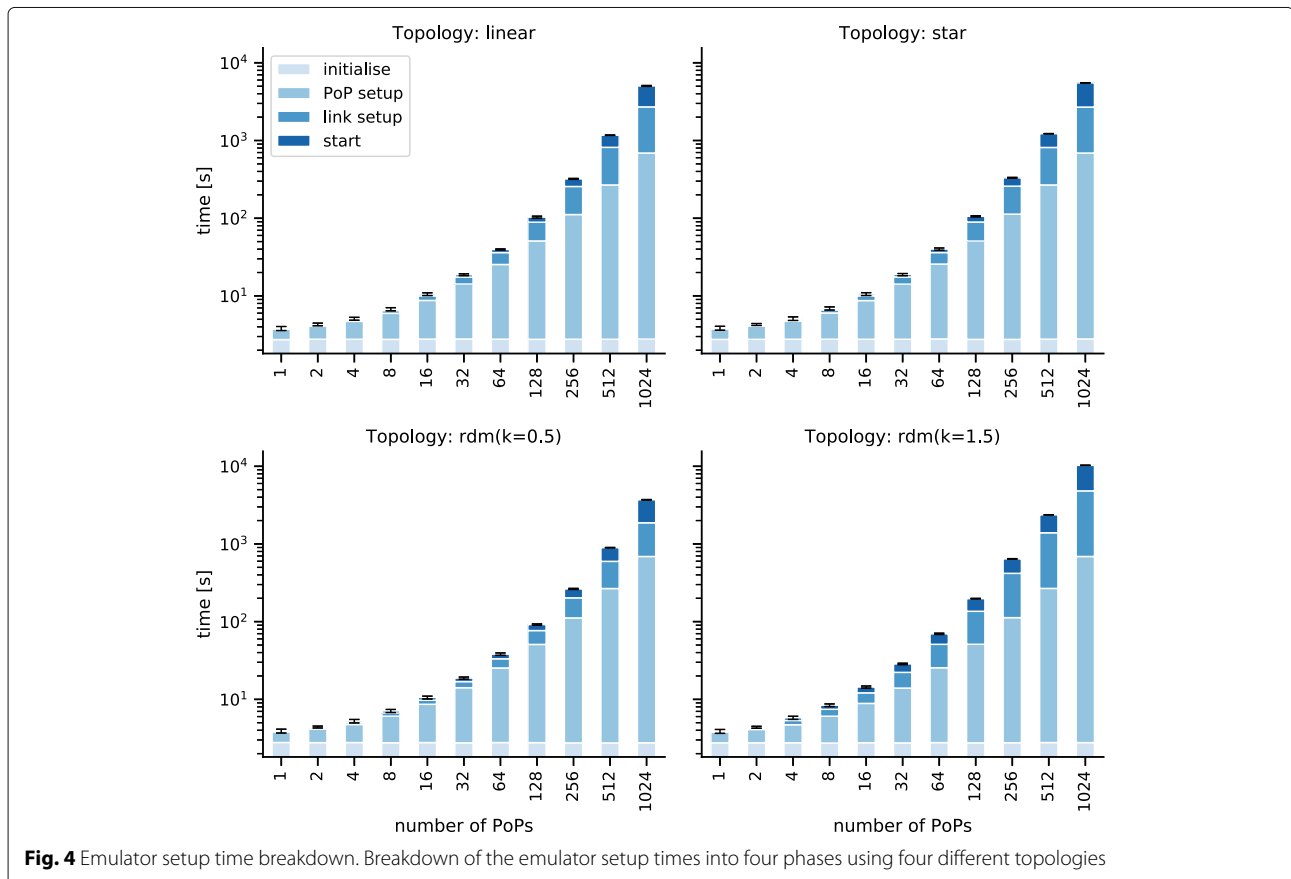
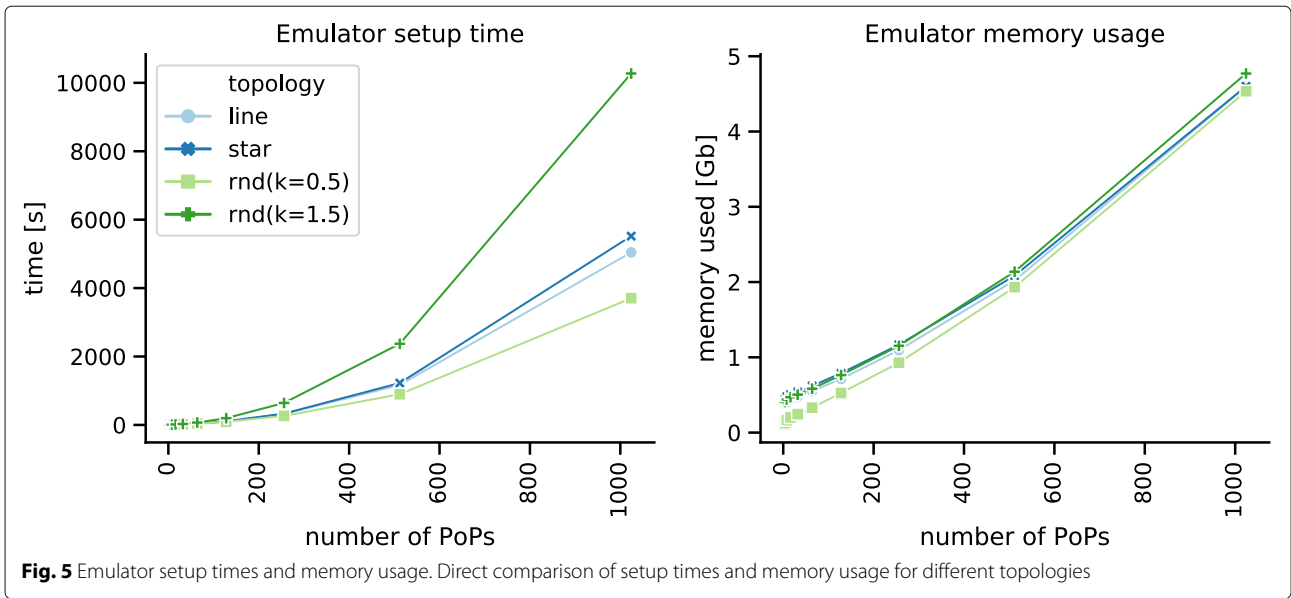


Fig. 4 Emulator setup time breakdown. Breakdown of the emulator setup times into four phases using four different topologies



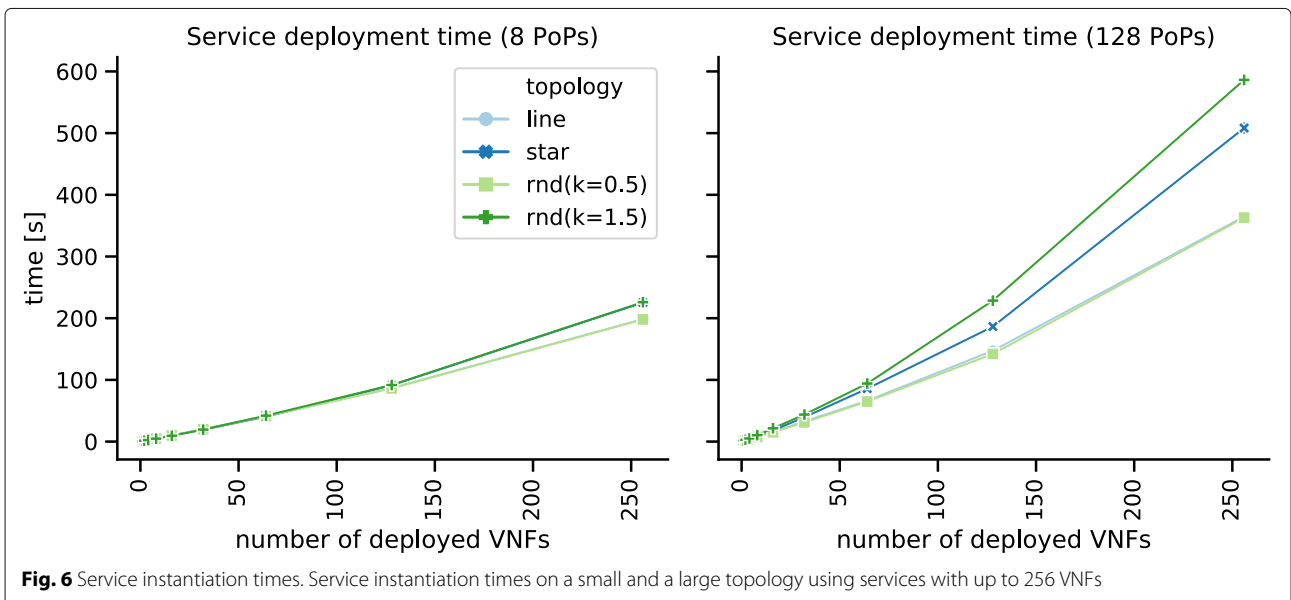
we are only interested in the bare instantiation times. Figure 6 shows that the instantiation times scale proportionally with the number of VNFs and that the instantiation process takes longer in larger topologies and is also influenced by the number of links in a topology. Please note that most error bars are hidden behind the markers of the plots because of the small deviation observed between the experiments. It can be seen that with our platform, hundreds of VNFs can be quickly deployed on a single machine, enabling fast tests of large deployment scenarios.

**3.2 Case study: OSM rel. THREE vs. OSM rel. FOUR**

In our case study, we decided to compare OSM rel. THREE and OSM rel. FOUR [5] because OSM

rel. FOUR is the latest release at the time of writing and the internal architecture of OSM has completely changed between those two major releases. Where OSM rel. THREE had a more monolithic design, with three or four large components using fixed APIs for communication, OSM rel. FOUR follows a micro-service-based architecture using many small components communicating over a common message bus. Besides the improved flexibility, this design also promises better scalability and performance, which we verify with our experiments.

The setup for the study was the same as described in Fig. 2, but we used a scripted test controller that automatically performs a series of experiments and collects additional data. Besides the general functionality of the

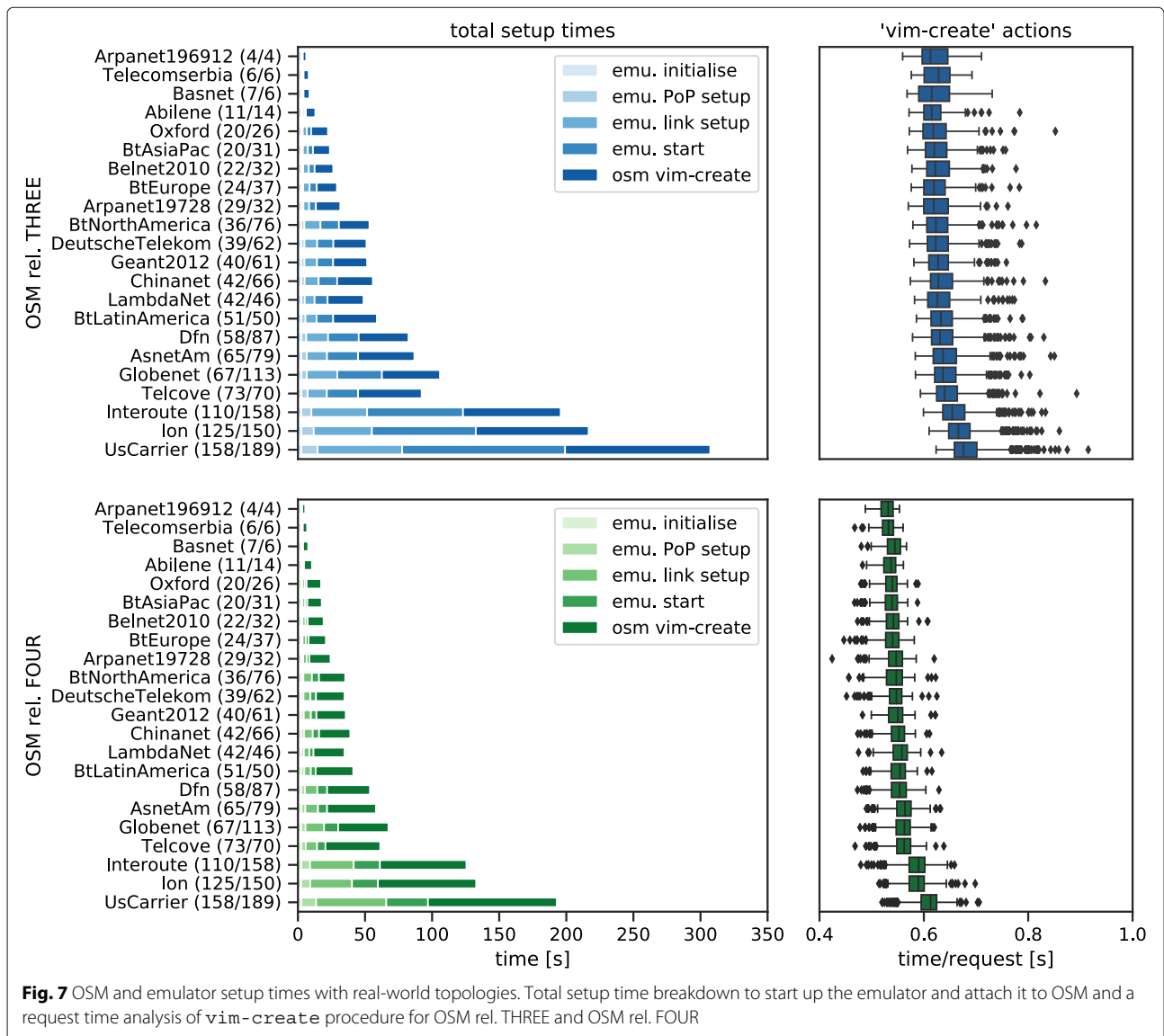


VIM attachment procedure, we investigated the behavior of OSM when it has to interact with large multi-PoP deployments and a high number of instantiated network services. To be more realistic, we used a set of real-world topologies with different sizes that are taken from the Internet Topology Zoo (ITZ) library [30]. In our case study, each node of a given topology is turned into a single PoP emulating an OpenStack VIM, resulting in topologies with 4 to 158 PoPs. The delays between the PoPs are calculated based on the geolocations provided by the ITZ dataset. These are test cases which are not covered by existing NFV testbed installations that usually only use a single PoP installation.

### 3.2.1 OSM in large multi-PoP environments

In the first set of experiments, we analyzed the VIM attach procedure, which is used to connect OSM to a single

PoP using the `osm vim-create < vim-endpoint >` command. Figure 7 shows the total setup time breakdown to start the emulated infrastructure and to attach all emulated VIMs to OSM. The numbers behind the topologies indicate the number of nodes and links in the topology. The results show that the time required to attach the VIMs to OSM uses most of the test environment's setup time, but the system can still be deployed and configured in between 200 and 330s, even if the largest topology with more than 150 PoPs is used. The figure also shows the request times for all `osm vim-create` requests. It indicates that the attachment procedure becomes slightly slower when larger topologies are used. Comparing the results between the two OSM releases, OSM rel. FOUR shows improved setup times and reduced request times to attach the VIMs. It can also be seen that the setup times of the emulation platform are smaller in the OSM rel. FOUR



case. The reason of this is the significantly smaller resource footprint of OSM rel. FOUR which is executed on the same physical machine as the emulation platform.

### 3.2.2 OSM service instantiation and termination

In the second set of experiments, we investigated OSM’s network service management behavior. More specifically, we tested the network service instantiation (`osm ns-create`), network service termination (`osm ns-delete`), and network service show (`osm ns-show`) operations. To do so, we used a test network service consisting of two linked VNFs. We requested OSM to sequentially create 64 instances of this service, which corresponds to 128 deployed VNFs. Later, these services are terminated one after each other. In each instantiation request, the service was randomly placed on the available PoPs of the three used topologies (Fig. 8). The given instantiation and termination times represent the time until the requested containers (the VNFs of the service) are started or stopped, not only the raw API response times.

The results show that a service instantiation takes between 7 and 12 s in most of the cases if OSM rel. FOUR is used. OSM rel. THREE, in contrast, shows instantiation times between 10 and 20 s. The results also show that the instantiation times in OSM rel. FOUR are more stable. The increased instantiation times shown by OSM rel. FOUR when the small Abilene (11 PoPs) topology is used are caused by the fact that more services are instantiated per emulated PoP. The analysis of network service termination operations, shown in the middle of Fig. 8, clearly shows that service termination is much faster and shows smaller variance in OSM rel. FOUR compared to OSM rel. THREE. Service termination times also

show only very small dependencies on the used topologies. Further, the request times to show details of a running network service instance have been improved in OSM rel. FOUR as shown in the right part of the figure. In general, `osm ns-show` requests are much faster than the other operations, since nothing in the actual service deployment is changed during a request.

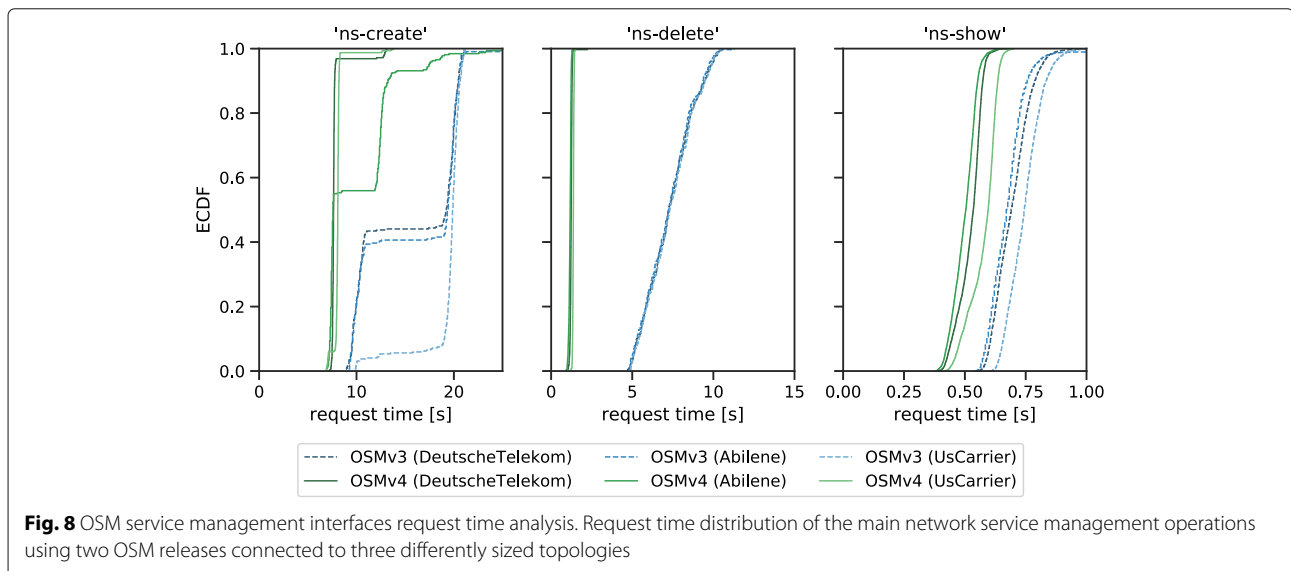
This test validates the design choices made in OSM rel. FOUR and shows that they improved its performance. More importantly, those large-scale test cases would not have been feasible without our presented testing platform and clearly show its usefulness for the NFV community and how it can support future 5G developments.

### 3.2.3 ETSI-compliant test suite

Using our ETSI SOL005-compliant test suite, presented in Section 2.2.2, we recorded the request times for more endpoints of OSM’s northbound interface. Table 1 shows the mean request times and standard deviation among 10 runs of the test suite against OSM rel. FOUR using an emulated PoP as a single connected VIM. The results show that the request times are all very stable and the use of our emulation platform allows to execute a complete test run in about 67.03 s, which is a result of the fast instantiation times of VNFs and network services. Similar test runs in cloud testbeds will take substantially longer. We do not present results for OSM rel. THREE, because of its missing ETSI SOL005 support and the resulting incompatibility with the test suite.

## 4 Discussion

The results of our case study show the evolution of OSM and how its performance was improved in the latest





release. Especially, the reduced resource requirements of OSM rel. FOUR contribute to a better performance when used with many PoPs.

During our case study, we found and reported some interesting issues, for example, a bug in OSM rel. THREE that prevents a user to instantiate a network service on the 101st or higher-numbered PoP. The reason for this is a hard-coded query limit that causes the OSM client to only fetch the first 100 PoPs that are attached to the system. This results in a *PoP not found* exception when a network service should be instantiated on, e.g., PoP 101. Based on our feedback, this issue is fixed in OSM rel. FOUR. We also noticed that for every `osm vim-show < pop x>` command, the entire VIM list is fetched by the OSM client, instead of only fetching the information of the requested PoP. This increases request delays when OSM is used with many attached PoPs.

It is important to note that such issues would not be discovered by today's NFV test deployments which usually do not use more than a handful of PoPs. But the 5G and NFV community envisions very large multi-PoP scenarios for future use cases, like Internet of Things (IoT). As a result, MANO systems need to be tested against such large multi-PoP networks. To do this, our platform provides a flexible and easy to apply test solution that allows to verify and improve the quality of MANO systems for use cases of future networks.

We are planning to test more MANO solutions, especially ONAP [4], as the second "big" player in the open-source MANO landscape, against our system and analyze their scalability and behavior in large multi-PoP scenarios. However, the majority of the available codebase of ONAP is, at the time of writing, not in a state to perform those experiments, e.g., because of limited APIs. Other reasons are the lack of automated installation procedures and the very high resource requirements. But we are confident that this will change in the next two or three release cycles. Another improvement we are planning to integrate into the presented platform is support for VNF and service configuration mechanisms, like Juju Charms [32]. This will allow VNF and service developers to use our platform to perform complex integration tests between their developed products and the MANO systems while having the benefits of a lightweight test platform that can be deployed locally.

## 5 Conclusions

Using emulation-based smoke testing as part of the automated test and integration pipeline, used by MANO software projects, contributes to the quality and production readiness of these complex software systems. The presented approach enables the pre-validation of future-readiness of MANO systems for upcoming, large-scale 5G scenarios with hundreds or thousands of PoPs.

This is not possible with today's lab-scale NFV testbed installations.

Our case study shows how our presented concepts are used to find bugs and to reveal the performance improvements between two major releases of OSM, one of the most prominent open-source MANO solutions today.

The presented platform was developed as part of the European H2020 project SONATA [33] and was recently adopted by the ETSI OSM project [5], where it is maintained under the name *vim-emu* as part of the *DevOps module development group*. Its future developments are supported by the 5GTANGO project [34]. It is open source and publicly available under Apache 2.0 license [35, 36]. The novel test suite for ETSI SOL005-compatible MANO northbound interfaces, presented in this paper, is also available under Apache 2.0 license [10]. It can be directly used by NFV researchers and MANO developers to test compatibility with the existing API specifications. Our results might be exploited and contribute to running test specification activities within standardization definition organizations (SDOs), like ETSI's STF557 [31]. The long-term vision for this project is to receive further contributions from the community and become an open-source reference test suite for NFV scenarios.

### Abbreviations

API: Application programming interface; BSS: Business support system; CD: Continuous deployment; CI: Continuous integration; ETSI: European Communications Standards Institute; IoT: Internet of Things; LCM: Lifecycle management; MANO: Management and orchestration; NFV: Network function virtualization; NFVI: NFV infrastructure; NS: Network service; NSD: Network service descriptor; OSM: OpenSource MANO; OSS: Operations support system; PoP: Point of presence; SDN: Software defined network; SFC: Service function chaining; VIM: Virtual infrastructure manager; VM: Virtual machine; VNF: Virtual network function; VNFD: Virtual network function descriptor; WAN: Wide area network; WIM: WAN infrastructure manager

### Acknowledgements

We want to thank the SONATA, 5GTANGO, and OSM project members for the many fruitful discussions that contributed to this study.

### Authors' contributions

MP came up with the initial concept of emulation-based smoke testing for NFV, the system design, and implemented the prototype. He wrote the majority of this paper. MM and GG both supported the case study with their deep knowledge about OSM. HK participated in the design process, gave feedback, and helped to improve the writing. All authors read and approved the final manuscript.

### Funding

This work has been partially supported by the SONATA project, funded by the European Commission under Grant number 671517 through the Horizon 2020 and 5G-PPP programs (<http://sonata-nfv.eu>), the 5GTANGO project, funded by the European Commission under Grant number H2020-ICT-2016-2 761493 through the Horizon 2020 and 5G-PPP programs (<http://5gtango.eu>), and the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

### Availability of data and materials

All software assets developed in this study are available under Apache 2.0 open source license [10, 35]. The datasets of all our measurements analyzed in this study are available in the following repository: <https://git.io/fAPLa>.

### Competing interests

The authors declare that they have no competing interests.

### Author details

<sup>1</sup>Computer Networks Group, Paderborn University, Warburgerstr. 100, 33098 Paderborn, Germany. <sup>2</sup>Sandvine, 11 Karl Court, Cambridge N3C3R4, Canada.

<sup>3</sup>Network Virtualization Initiative, Telefónica Investigación y Desarrollo, Zurbarán 12, 28010 Madrid, Spain.

Received: 19 September 2018 Accepted: 10 June 2019

Published online: 26 June 2019

### References

- H. Karl, S. Dräxler, M. Peuster, A. Galis, M. Bredel, A. Ramos, J. Martrat, M. S. Siddiqui, S. van Rossem, W. Tavernier, et al, DevOps for network function virtualisation: an architectural approach. *Trans. Emerg. Telecommun. Technol.* **27**(9), 1206–1215 (2016)
- OpenStack Project, DevStack. Online at: <https://docs.openstack.org/devstack/latest/>. Accessed Aug 2018
- S. Dräxler, H. Karl, M. Peuster, H. R. Kouchaksaraei, M. Bredel, J. Lessmann, T. Soenen, W. Tavernier, S. Mendel-Brin, G. Xilouris, in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*. SONATA: Service programming and orchestration for virtualized software networks (IEEE, Paris, 2017), pp. 973–978. <https://doi.org/10.1109/ICCW.2017.7962785>
- Linux Foundation, ONAP: Open Network Automation Platform. Online at: <https://www.onap.org>. Accessed Mar 2018
- ETSI OSM, Open Sorce MANO. Online at: <https://osm.etsi.org>. Accessed Feb 2018
- M. Peuster, M. Marchetti, G. G. de Blas, H. Karl, in *2018 European Conference on Networks and Communications (EuCNC)*. Emulation-based Smoke Testing of NFV Orchestrators in Large Multi-PoP Environments, (Ljubljana, 2018), pp. 1–9. <https://doi.org/10.1109/EuCNC.2018.8442701>
- E. Dustin, J. Rashka, J. Paul, *Automated software testing: introduction, management, and performance*. (Addison-Wesley Longman Publishing Co., Inc., Boston, 1999)
- M. Peuster, H. Karl, S. van Rossem, in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments (IEEE, Palo Alto, 2016), pp. 148–153. <https://doi.org/10.1109/NFV-SDN.2016.7919490>
- ETSI, Network Functions Virtualisation (NFV) Release 2; Protocols and Data Models; RESTful protocols specification for the Os-Ma-nfvo Reference Point (2018). Website. Online at [https://www.etsi.org/deliver/etsi\\_gs/NFV-SOL/001\\_099/005/02.04.01\\_60/gs\\_NFV-SOL005v020401p.pdf](https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/005/02.04.01_60/gs_NFV-SOL005v020401p.pdf). Accessed 18 June 2019
- M. Peuster, ETSI NFV SOL005 Test Suite. Online at: <https://github.com/mpeuster/etsi-nfv-sol005-test-suite>. Accessed Aug 2018
- C. Parada, J. Bonnet, E. Fotopoulou, A. Zafeiropoulos, E. Kapassa, M. Touloupou, D. Kyriazis, R. Vilalta, R. Muñoz, R. Casellas, R. Martínez, G. Xilouris, in *2018 European Conference on Networks and Communications (EuCNC)*. 5GTANGO: A Beyond-Mano Service Platform, (2018), pp. 26–30. <https://doi.org/10.1109/EuCNC.2018.8443232>
- H. R. Kouchaksaraei, S. Dräxler, M. Peuster, H. Karl, in *2018 European Conference on Networks and Communications (EuCNC)*. Programmable and flexible management and orchestration of virtualized network functions, (Ljubljana, 2018), pp. 1–9. <https://doi.org/10.1109/EuCNC.2018.8442528>
- ETSI, GS NFV 002: Network Functions Virtualisation (NFV): Architectural Framework (2014). Online at [https://www.etsi.org/deliver/etsi\\_gs/nfv/001\\_099/002/01.02.01\\_60/gs\\_nfv002v010201p.pdf](https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.02.01_60/gs_nfv002v010201p.pdf). Accessed 18 June 2019
- S. McConnell, Daily build and smoke test. *IEEE Softw.* **4**, 144–143 (1996)
- Linux Foundation, *OPNFV Project*. (Website. Online at <https://www.opnfv.org>. Accessed 18 June 2019
- A. F. Cattoni, G. C. Madueño, M. Dieudonne, P. Merino, A. D. Zayas, A. Salmeron, F. Carlier, B. Saint Germain, D. Morris, R. Figueiredo, et al, in *2016 European Conference on Networks and Communications (EuCNC)*. An end-to-end testing ecosystem for 5G IEEE, Athens, 2016), pp. 307–312. <https://doi.org/10.1109/EuCNC.2016.7561053>
- M. Zhao, F. L. Gall, P. Cousin, R. Vilalta, R. Muñoz, S. Castro, M. Peuster, S. Schneider, M. Siapera, E. Kapassa, D. Kyriazis, P. Hasselmeier, G. Xilouris, C. Tranoris, S. Denazis, J. Martrat, in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. Verification and validation framework for 5g network services and apps, (2017), pp. 321–326. <https://doi.org/10.1109/NFV-SDN.2017.8169878>
- L. Cao, P. Sharma, S. Fahmy, V. Saxena, in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference On*. NFV-VITAL: a framework for characterizing the performance of virtual Network Functions (IEEE, 2015), pp. 93–99
- R. V. Rosa, C. Bertoldo, C. E. Rothenberg, Take your vnf to the gym: a testing framework for automated nfv performance benchmarking. *IEEE Commun. Mag.* **55**(9), 110–117 (2017). <https://doi.org/10.1109/MCOM.2017.1700127>
- M. Peuster, H. Karl, in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFVSDN)*. Profile your chains, not functions: Automated network service profiling in DevOps environments, (Berlin, 2017), pp. 1–6. <https://doi.org/10.1109/NFV-SDN.2017.8169826>
- M. Keller, C. Robbert, M. Peuster, in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM(SIGCOMM '13)*. An evaluation testbed for adaptive, topology-aware deployment of elastic applications (ACM, New York, 2013), pp. 469–470. <https://doi.org/10.1145/2486001.2491689>
- B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, M. Bowman, PlanetLab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.* **33**(3), 3–12 (2003). <https://doi.org/10.1145/956993.956995>
- H2020 SoftFIRE consortium, SoftFIRE approach to experiment management: why and how. Online at <https://goo.gl/LxLflz>. Accessed Aug 2018
- H2020 Fed4Fire+ consortium, Fed4Fire: The largest federation of testbeds in Europe. Online at: <https://www.fed4fire.eu>. Accessed Aug 2018
- B. Lantz, B. Heller, N. McKeown, in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks(Hotnets-IX)*. A network in a laptop: rapid prototyping for softwaredefined networks (ACM, New York, 2010), p. 6. <https://doi.org/10.1145/1868447.1868466>
- J. Ahrenholz, C. Danilov, T. R. Henderson, J. H. Kim, in *MILCOM 2008 - 2008 IEEE Military Communications Conference*. CORE: A real-time network emulator, (San Diego, 2008), pp. 1–7. <https://doi.org/10.1109/MILCOM.2008.4753614>
- L. Mamas, S. Clayman, A. Galis, A service-aware virtualized software-defined infrastructure. *Commun. Mag. IEEE.* **53**(4), 166–174 (2015)
- Jenkins Project, Jenkins CI webpage. Online at: <https://jenkins.io>. Accessed Apr 2018
- M. Peuster, Containernet. Online at: <https://containernet.github.io>. Accessed Aug 2018
- S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo. *Sel. Areas Commun. IEEE J.* **29**(9), 1765–1775 (2011). <https://doi.org/10.1109/JSAC.2011.111002>
- G. Bernini, M. Zhao, E. Kraja, Specialist Task Force 557: NFV API Conformance test specification. Online at: <https://portal.etsi.org/STF/STFs/STFHomePages/STF557>. Accessed Apr 2019
- Canonical Ltd, Juju Charms. Online at: <https://www.juju charms.com>. Accessed Aug 2018
- SONATA Consortium, SONATA Project. Online at: <http://sonata-nfv.eu>. Accessed Jan 2018
- 5GTANGO Consortium, 5GTANGO Project. Online at: <http://5gtango.eu>. Accessed Aug 2018
- ETSI OSM, OSM vim-emu code repository. Online at: <https://osm.etsi.org/gitweb/?p=osm/vim-emu.git>. Accessed Aug 2018
- ETSI OSM, OSM vim-emu documentation. Online at: <https://goo.gl/XZNLVw>. Accessed Aug 2018

### Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.