Abstract—One of the biggest challenges for the real-world application of network function virtualization (NFV) is reducing the development complexity of both virtualized network functions (VNF) and service functions chains (SFC). This is in particular important for the upcoming 5th generation of networks in which service agility is one of the key concepts to allow quick development and integration cycles and to reduce costs. Still, the availability of tools to support VNF and service developers is limited and existing solutions mainly focus on packaging support and static validation of descriptors.

To change this, we introduce *Containernet 2.0* a novel, open-source NFV prototyping platform supporting the creation and local execution of complex SFCs. *Containernet 2.0* is the first prototyping tool that explicitly supports hybrid SFCs composed of both container-based and virtual machine-based VNFs that are combined in a single chain. During our demonstration, the end-to-end SFC prototyping workflow, including composition, execution, and configuration, is shown.

I. INTRODUCTION

In network function virtualization (NFV), complex network services are composed of multiple, chained virtualized network functions (VNFs) as so-called service function chains (SFC) [1]. Such SFCs are usually defined by network service descriptors that are static files describing the interrelationship and chaining of the involved VNFs. In addition, VNFs are defined with VNF descriptors (VNFD) that describe how a VNF and its deployment units (VDU), e.g., a virtual machine image or a container, should be provisioned and deployed on top of the available NFV infrastructure (NFVI) [2]. All these artifacts are usually developed by a network service developer who writes down the descriptors, provisions new or re-uses existing VNFs, and configures these VNFs according to the needs of the network service.

Today, most parts of this development process are manual, complicated, and error-prone steps with very limited tool support. Existing NFV development support solutions, so called *NFV service development kits (SDK)*, mainly focus on descriptor creation or generation as well as static syntactical and semantical checks among them [3], [4]. These tools help to identify bugs and errors in the static descriptors, like a missing link in an SFC definition. But they do not offer support for developers when VNFs and their contained software components should be integrated and configured. For example, a firewall software that should be installed and configured inside an existing VM or container image. In practice, this means that a developer needs to set up a local NFVI testbed on which the developed service is deployed and manually configured and tested. Once everything works and is tested, the developer has to export the VM or container images to ship them—a process that is far too complicated for an agile environment.

To solve this issue, rapid prototyping platforms are required that allow the local execution and configuration of complex SFCs on a developer’s laptop. First solutions use single-node NFVI deployments [5] or Java-based VNF proxy functions in simulated environments to create lightweight prototyping platforms [6]. Others offer explicit debugging support, but focus more on software-defined networking (SDN) and not on the integration of real-world VNFs [7]. Another solution, which was introduced in our previous work [8], is called *MeDICINE* and supports rapid prototyping of container-based VNFs in multi-PoP environments. *MeDICINE*, however, focuses mainly on the integration between orchestration systems and the developed network services rather than on the network services and the VNFs as such [9]. More importantly, none of these solutions supports hybrid SFCs that are composed of both container-based and VM-based VDUs at the same time, which will be a common scenario for 5G deployments [10].

In this demonstration, we introduce *Containernet 2.0* [11], the first rapid prototyping platform that supports the execution of hybrid SFCs consisting of container- and VM-based VNFs (Sec. II). Our platform can be installed and executed locally on a developer’s laptop and comes with an intuitive service composition GUI that allows developers to compose service prototypes within minutes. During our demonstration, an example network service is composed, configured, and executed on the prototyping platform. Further, live interactions and reconfigurations of the involved VNFs through Containernet’s interactive command line interface (CLI) will be shown. Finally, the service will be tested with video streaming traffic as described in Sec. III.

II. RAPID-PROTOTYPING OF HYBRID NETWORK SERVICES

One key requirement for rapid prototyping is the availability of an execution environment in which the developed components can be quickly deployed and tested by a developer. To build this execution environment for complex SFCs, we initiated the *Containernet* [11] project, a fork of the famous *Mininet* network emulator [12]. Containernet allows to execute VNFs in form of Docker containers and to interconnect them to arbitrary complex topologies. Containernet 2.0 extends
Containernet, which uses pipes to directly connect to the TTYs In contrast to the CLI interaction scheme used in Mininet and with all nodes (Mininet hosts, Docker container, and VMs) Containernet’s interactive CLI that should allow a user to interact with all nodes (Mininet hosts, Docker container, and VMs) in the emulated network through a common CLI interface. In contrast to the CLI interaction scheme used in Mininet and Containernet, which uses pipes to directly connect to the TTYs of the emulated hosts or containers, a direct interaction with VMs is not possible. To solve this, we opted for a network-based solution that adds a management network interface to each VM and connects to it using SSH\textsuperscript{2}. This solution solves the problem and gives seamless access to all VMs in the emulated topology (see Fig. 1). The downside of this approach is that it introduces the requirement that all used VMs need to have SSH installed and their access credentials have to be available to Containernet 2.0. We argue that this is an acceptable requirement since the majority of existing NFV and cloud orchestration solutions rely on such management interfaces in any case.

```python
net = Containernet()
# add Mininet host, Docker host, VM-based host
h1 = net.addHost("h1", ip="10.0.0.1")
d1 = net.addDocker(
   "d1", ip="10.0.0.2", image="ubuntu:trusty")
v1 = net.addLibvirtHost(
   "v1", ip="10.0.0.3", image="/ubuntu1604.qcow2")
# connect hosts to switches
s1 = net.addSwitch("s1")
net.addLink(h1, s1)
net.addLink(d1, s1)
net.addLink(v1, s1)
# start the emulation
net.start()
```

Listing 1: Example Containernet 2.0 topology with Mininet host (h1), Docker host (d2), and VM-based host (v1).

B. Extending MiniEdit for Containernet

To simplify the composition of complex SFCs, we extended Mininet’s graphical editor, called MiniEdit, as shown in Fig. 2. In particular, we added support for Docker-based and VM-based hosts as well as support for multihoming and direct interconnections between all types of nodes. The multihoming feature is especially important since VNFs usually have multiple network interfaces, like data input, data output, and management/control. Having these features in place, our platform can be used to prototype complex SFCs including their data plane and control plane.

III. DEMONSTRATION

The objective of the planned demonstration is three-fold. First, we demonstrate how a complex network service can be composed with our graphical user interface. The created network service consists of both container-based and VM-based VNFs creating a hybrid SFC. Second, we demonstrate how our prototyping platform can be used to run production-ready network services on a developer’s laptop. Finally, we show how the developer can interact and reconfigure the running service as well as test its functionality.

A. Demonstrated Scenarios

Our demonstration comes with a set of example VNFs that operate on different layers of the networking stack, i.e., a proxy based on Squid\textsuperscript{3} deployed in a Docker container acting on

\textsuperscript{1}Libvirt project: https://libvirt.org

\textsuperscript{2}Secure Shell protocol: https://www.ssh.com/ssh/

\textsuperscript{3}Squid: http://www.squid-cache.org
and VMs at the same time. It is in particular useful to test and configure container and VM images before they are deployed on real NFVI testbeds or production platforms. Our graphical composer drastically lowers the barrier for new VNFRD and service developers. The presented tools are complementary and compatible to existing NFV SDKs and bridge the gap between static descriptor creation on the developer’s laptop and the actual execution of the service and its VNFs. Containernet 2.0 is open source and available on GitHub\(^6\) [11]. There is a video available that shows parts of the described demonstration\(^7\).

ACKNOWLEDGMENTS

This work has been partially supported by 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).

REFERENCES