This work has been accepted for publication in 2018 IEEE Conference on Network Softwarization (Netsoft 2018).
Copyright © 2018 by IEEE. ISBN: 978-1-5386-4633-5

Containernet 2.0: A Rapid Prototyping Platform for Hybrid Service Function Chains

Manuel Peuster
Paderborn University
manuel.peuster@uni-paderborn.de

Johannes Kampmeyer
Paderborn University
johannes.kampmeyer@uni-paderborn.de

Holger Karl
Paderborn University
holger.karl@uni-paderborn.de

***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-ues 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

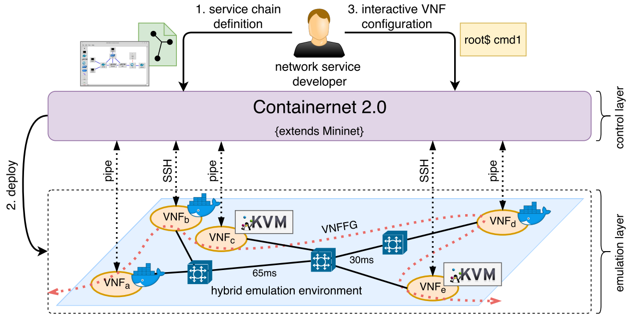


Fig. 1: Demonstration overview showing a network service developer prototyping a hybrid SFC consisting of three container-based and two VM-based VNFs, all running in an locally emulated network.

the existing project by adding execution support for VM-based VNFs to the platform. Fig. 1 shows the architecture of Containernet 2.0 and how a network service developer uses it. As a first step, the developer defines the SFC by using either Containernet's GUI editor or a script that calls Containernet's Python API (1). After this, Containernet deploys and interconnects the involved VNFs in its local, Mininet-based emulation environment (2). Once all VNFs are running, the developer uses Containernet's interactive CLI to interact with and configure the running VNFs that can either be Docker containers or full-featured VMs (3). To establish the network between the VNFs, the underlying Mininet is used. Our platform is fully backwards compatible to the original Mininet emulation API, e.g., it allows to emulate arbitrary delays, loss, and jitter on the links between VNFs. It also allows to include SDN switches and customized SDN controllers into the prototyped topologies, e.g., to build custom chaining solutions.

A. Supporting VMs in Containernet

Our main requirement for the integration of VMs into Containernet 2.0 is to be fully aligned with the existing Mininet and Containernet APIs. Our design allows a user to add a fully-featured VM to an emulation topology with a single Python command that expects the path to the VM image to be used as additional parameter, as shown in Listing 1 line 6 and line 7. Other parameters, like the node name or the IP addresses to be used, remain the same as in the existing implementations. This enables the seamless integration of VMs into existing emulation topologies. Additional parameters, like the hypervisor type, can be optionally passed to the underlying *libvirt*[1] implementation.

A more challenging problem was the integration with 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[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.

```
1  net = Containernet()
2  # add Mininet host, Docker host, VM-based host
3  h1 = net.addHost("h1", ip="10.0.0.1")
4  d1 = net.addDocker(
5      "d1", ip="10.0.0.2", image="ubuntu:trusty")
6  v1 = net.addLibvirthost(
7      "v1", ip="10.0.0.3", image="~/ubuntu1604.qcow2")
8  # connect hosts to switches
9  s1 = net.addSwitch("s1")
10 net.addLink(h1, s1)
11 net.addLink(d1, s1)
12 net.addLink(v1, s1)
13 # start the emulation
14 net.start()
```

Listing 1: Example Conatinernet 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

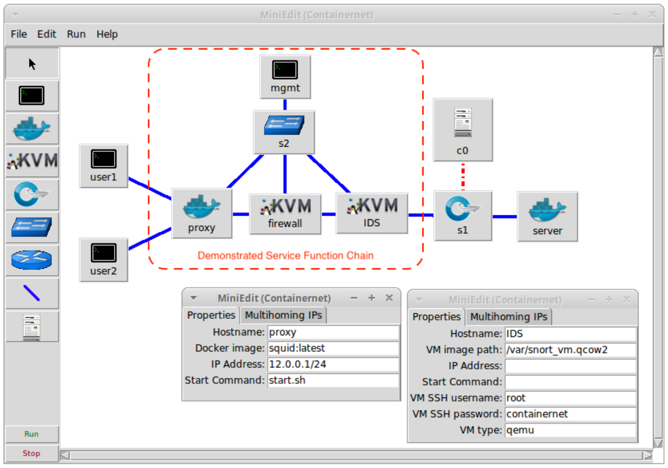
Our demonstration comes with a set of example VNFs that operate on different layers of the networking stack, i.e., a *proxy* based on Squid[3] deployed in a Docker container acting on

---

[1] Libvirt project: https://libvirt.org

[2] Secure Shell protocol: https://www.ssh.com/ssh/

[3] Squid: http://www.squid-cache.org



MiniEdit (Containernet)
Properties | Multihoming IPs
Hostname: proxy
Docker image: squid:latest
IP Address: 12.0.0.1/24
Start Command: start.sh

MiniEdit (Containernet)
Properties | Multihoming IPs
Hostname: IDS
VM image path: /var/snort_vm.qcow2
IP Address:
Start Command:
VM SSH username: root
VM SSH password: containernet
VM type: qemu

Fig. 2: Containernet's intuitive SFC composition editor showing our demo service consisting of a *proxy* (Docker), a *firewall* (VM), and a *IDS* (VM) VNF. It also shows additional nodes used to emulate users, management network, and content server to test the SFC.

L3, a *firewall* based on Iptables[4] deployed in a VM acting on L2, and an intrusion detection system (IDS) based on Snort[5] deployed in a VM acting on L2. Additionally, a Docker-based content server for video streaming is used to generate test traffic. During the demo, these example VNFs are combined to an SFC as shown in Fig. 2.

In particular, we show how the available VNFs are configured prior and after their deployment, e.g., we show how a developer can reconfigure a firewall or analyze the correctness of the IDS rules.

B. Demonstration Steps

The demonstration includes the following steps:

1) *Step 1:* Composition of a hybrid network service, consisting of VMs and containers, using Containernet's intuitive graphical composition interface (Fig. 2).
2) *Step 2:* Instantiation of the hybrid network service in the local emulation environment.
3) *Step 3:* Live interaction and reconfiguration of running VNFs through Containernet's interactive CLI.
4) *Step 4:* End-to-end verification of the service composition and configuration using test traffic, i.e., video streaming.

C. Demonstration Requirements

The demonstration can be executed either locally on a single laptop running the entire emulation platform or remotely on a server. It requires a power outlet for a laptop and one or two large screens to show the visualizations and executed services.

IV. CONCLUSION

Our novel prototyping platform simplifies the development of hybrid network services that are composed of containers 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 VNF 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

[1] 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," *Transactions on Emerging Telecommunications Technologies*, vol. 27, no. 9, pp. 1206–1215, 2016.

[2] J. Garay, J. Matias, J. Unzilla, and E. Jacob, "Service description in the NFV revolution: Trends, challenges and a way forward," *IEEE Communications Magazine*, vol. 54, no. 3, pp. 68–74, March 2016.

[3] S. V. Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "Introducing Development Features for Virtualized Network Services," *IEEE Communications Magazine*, vol. PP, no. 99, pp. 2–10, 2018.

[4] S. V. Rossem, M. Peuster, L. Conceio, H. R. Kouchaksaraei, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "A network service development kit supporting the end-to-end lifecycle of NFV-based telecom services," in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2017, pp. 1–2.

[5] OpenStack Project, "DevStack," online at: https://docs.openstack.org/devstack/latest/, 2018.

[6] S. Clayman, L. Mamatas, and A. Galis, "Experimenting with control operations in software-defined infrastructures," in *NetSoft Conference and Workshops (NetSoft)*. IEEE, 2016, pp. 390–396.

[7] T. Lévai, I. Pelle, F. Németh, and A. Gulyás, "EPOXIDE: a modular prototype for SDN troubleshooting," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 359–360.

[8] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 148–153.

[9] M. Peuster, S. Draxler, H. R. Kouchaksaraei, S. v. Rossem, W. Tavernier, and H. Karl, "A flexible multi-pop infrastructure emulator for carrier-grade MANO systems," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–3.

[10] R. Cziva, S. Jouet, K. J. S. White, and D. P. Pezaros, "Container-based network function virtualization for software-defined networks," in *2015 IEEE Symposium on Computers and Communication (ISCC)*, July 2015, pp. 415–420.

[11] Containernet Project, "Containernet a Mininet Fork adding Container Support to Network Emulations," online at: https://containernet.github.io, Paderborn University, 2017.

[12] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

---

[4] Iptables: http://www.netfilter.org

[5] Snort: https://www.snort.org

[6] GitHub: https://github.com/containernet/containernet

[7] YouTube Video: https://youtu.be/TODO