A Prototyping Platform to Validate and Verify Network Service Header-based Service Chains

Manuel Peuster
Paderborn University
manuel.peuster@upb.de

Stefan Schneider
Paderborn University
stefan.schneider@upb.de

Frédéric Tobias Christ
Paderborn University
ftchrist@mail.upb.de

Holger Karl
Paderborn University
holger.karl@upb.de

***Abstract*—Network service header (NSH) is considered to be a key enabler for the wide adoption of service function chaining (SFC), but the availability of NSH-enabled components, such as network functions, forwarders, or classifiers, is still very limited. One reason for this is the lack of experimentation and prototyping platforms for NSH.**

This paper aims to rectify this shortcoming by introducing a novel prototyping platform for NSH. Our platform uses an emulation-based approach to build a lightweight environment for containerized network functions and allows to create complex SFC experiments and prototypes, as we show in our evaluation. The presented platform is publicly available and aims to be a rapid prototyping tool for researchers and developers.

I. Introduction

A key concept to deploy complex network services using network service virtualization (NFV) is service function chaining (SFC) as defined by IETF [1]. SFC allows to combine multiple, possibly virtualized network functions to larger services. Those functions are then called service functions (SF) and are traversed by packets according to the SFC configuration using one or multiple service function paths (SFP) [2]. To simplify forwarding along an SFP, Quinn et al. [3], [4] introduced network service header (NSH) as a possible protocol to encapsulate and mark packets according to the SFPs they are assigned to. The main benefits of NSH are the possibility to create topology-independent SFPs and its ability to pass arbitrary metadata between the involved SFs, e.g., classification information.

However, in a recent survey, Medhat et al. [5] identify the lack of NSH support in current switch and SF implementations as one of the key challenges for SFC, which is limiting its applicability and leading to unnecessary complexity in today's NFV solutions. We argue that one reason for the lack of practical implementations are missing prototyping environments in which NSH-enabled network functions can be easily developed and tested. Even if major virtual infrastructure managers (VIM), like OpenStack, are currently integrating SFC solutions into their platforms, their availability for prototyping is still very limited. One reason for this is that they focus on production-grade systems rather than on systems that offer development support, e.g., debugging. Other major drawbacks are their high resource requirements and complicated setup procedures, making simple and fast local deployments, e.g., on a developer's laptop, infeasible.

To this end, we introduce a novel, lightweight prototyping platform for NSH-enabled SFCs as the main contribution of this paper. Our platform allows researchers as well as function and service developers to quickly test their NSH-enabled components and to validate and verify their functionality before putting them into production. The presented platform can be used in three different ways. First, an SF developer can use our platform to validate and verify that an SF behaves correctly in an NSH-enabled SFC setup. Second, SFC integrators can test complex service chains with many SFs in a controlled environment. Third, management and orchestration (MANO) solutions can use our platform as an experimentation and test backend to test their service orchestration functionality, e.g., to verify that their chaining logic requests correct forwarding paths.

To simplify the prototyping process, our platform also comes with a set of pre-packed SFC components such as a generic NSH-enabled SF and a traffic generator for NSH encapsulated traffic.

The remainder of this paper is organized as follows. We first discuss existing work and the lack of suitable prototyping tools in Sec. II. In Sec. III, we present and discuss the design of our open source prototyping platform. We evaluate the functionality of our prototype in Sec. IV and conclude in Sec. V.

II. Related Work

Besides the RFCs mentioned in Sec. I, the software-defined networking (SDN) and NFV communities investigate a variety of different aspects within SFC, e.g., regarding resilience against failures [6], [7] or dynamic readjustment to changed service requests [8]. So far, these research findings have been mostly evaluated using simulation [6], [8] or heavyweight testbeds [7], which are not available to every developer and hard to set up. Future evaluations can benefit from the lightweight prototyping solution proposed in this paper.

Similar to our approach, Davoli et al. provide an implementation of an SFC control plane using NSH [9]. However, our approach focuses more on enabling quick prototyping, e.g., using lightweight Docker containers instead of complete virtual machines. Furthermore, our platform reuses existing Open vSwitches (OvS) for the data plane rather than deploying separate service function forwarders (SFFs).

We build our NSH-based prototyping platform on our *vim-emu* emulation platform, formerly known as *MeDICINE* [10]. This platform allows realistic emulation of arbitrary network topologies with multiple Points-of-Presence (PoPs) as well as the execution of real network services consisting of container-based SFs. This emulation platform is highly scalable and can efficiently emulate hundreds of PoPs [11]. This makes it a perfect fit for lightweight NSH prototyping and large-scale NSH experiments. Other emulation platforms like Mininet [12], Maxinet [13], or VLSP [14] are not suitable for prototyping of SFC approaches as they do not emulate NFV infrastructure, to which hosts can be added or removed at runtime and be chained dynamically. In contrast to *vim-emu*, they do not provide standard VIM interfaces, which allow using MANO systems for managing chained network services. While VLSP provides some support for attaching MANO systems, it does not allow executing real-world SFs.

Pelle et al. [15] provide a framework for troubleshooting and debugging SDN, but it does not focus on quick prototyping of SFC. Similarly, ESCAPE [16] is not suitable for quick prototyping of SFC approaches as it focuses more on orchestration between non-emulated PoPs. Finally, simulation approaches [17]–[19] do not support real SF implementations and can thus not be considered as prototyping environments.

III. A Prototyping Platform for NSH-enabled Functions and Services

To solve the problem of missing prototyping platforms for NSH-enabled SFs, we first collected the requirements that such a platform should fulfill to properly support developers: (i) The platform has to be able to quickly deploy the prototyped SFs. This includes the execution of SFs written in different programming languages using arbitrary frameworks and libraries. (ii) A developer should be able to configure arbitrary network topologies in which SFs or complex SFCs can be tested. (iii) The created test networks should allow to transport and deliver real network traffic and implement the correct forwarding behavior of NSH-encapsulated packets. (iv) A prototyping platform should seamlessly integrate with other tools commonly used in the NFV landscape, e.g., MANO systems.

Based on these requirements, we developed our novel prototyping platform that uses an emulation-based approach to create arbitrary complex network topologies with multiple PoPs. We based this platform on our NFV emulator *vim-emu* initially presented in [10]. This emulator allows to emulate large multi-PoP NFV scenarios on a single physical or virtual machine and can be deployed and started within seconds [11]. The emulation part is based on Containernet [20] and the links between the involved PoPs can be configured with user-selected delays, bandwidth, and loss rates. SFs can then be deployed as lightweight Docker containers in each of the emulated PoPs and execute arbitrary software as long as it can be packaged inside a Docker container. These features satisfy our previously defined requirements (i) and (ii). On top of this, *vim-emu* provides an API endpoint for each emulated PoP that mimics the API of an OpenStack-based VIM and can directly be consumed by existing MANO solutions like OSM [21]. This feature provides the basis to satisfy requirement (iv).

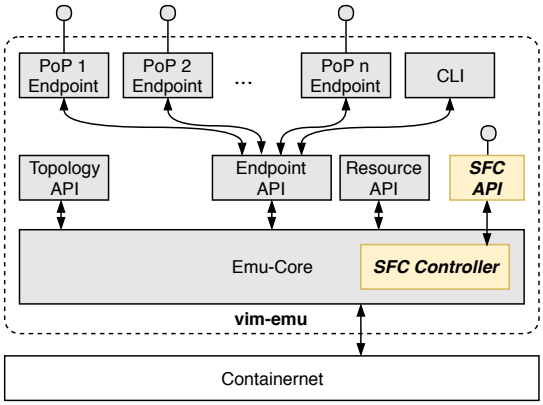

Fig. 1: Extended *vim-emu* architecture with additional SFC controller and API.

However, the existing version of *vim-emu* has no support for NSH-based SFC and only provides a very simplistic chaining model using VLAN tags statically assigned to SF ports. Hence, we address requirement (iii) by extending *vim-emu* as shown in Fig. 1 and adding support for NSH [22]. We describe those extensions in the following sections.

A. Adding NSH Support to a Multi-PoP Emulator

Starting from the bottom, we first look at the networking layer of *vim-emu* and identify the required changes to support NSH. Each emulated network topology consists of multiple PoPs and each PoP in a *vim-emu* emulation is represented by a single virtual SDN switch instance. Thus, every PoP simplifies its internal network using a big-switch abstraction, turning an emulated multi-PoP topology into a much simpler network of virtual switches as shown in Fig. 2. All these switches can be controlled by a single SDN controller, fitting to the IETF SFC architecture and NSH design, which expects to be deployed in a single control domain [3].

The next question is whether the used switches support NSH encapsulated packets, i.e., match the NSH fields, and can be used as service function forwarders (SFF) [2]? Fortunately, OvS already comes with experimental support for NSH starting with version 2.9. It supports the installation of NSH-specific rules using either its command-line client (`ovs-ofctl`) or OpenFlow using the extensible match (OXM) feature introduced in OpenFlow 1.2. Since *vim-emu* is based on Containernet [20], which is a fork of Mininet [12], OvS switches can be used within the emulated networks. After manually updating the used switches, NSH support is available in *vim-emu* networks.

Directly using the NSH support of the involved switches comes with the benefit that our platform can use Ethernet as transport protocol for the NSH encapsulated packets as indicated in Fig. 2. It does not need to use more complex tunnel setups like production-ready solutions do, e.g., OpenStack.

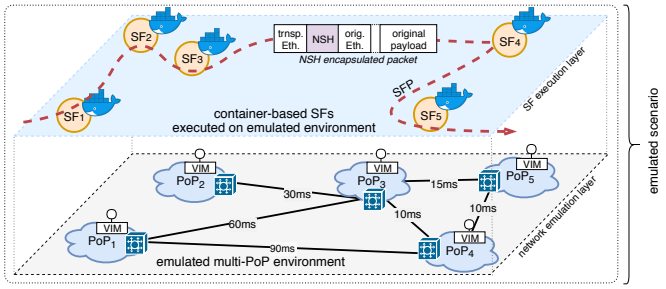

Fig. 2: Emulated network scenario with five interconnected PoPs and five Docker-based SFs deployed among them.

This is enabled by the previously mentioned simplifications of the emulated networks, putting all PoPs and their switches into a single Layer 2 domain. This comes with the benefit that a developer can easily dump and check the network traffic at all interfaces involved in an experiment. However, the drawback of this simplification is that debugging of tunnel setups requires some additional manual work. But this is usually out of scope for function and service developers who consume the transport functionality offered by target platforms.

Once NSH is supported by the underlying network emulation and requirement (iii) is satisfied, the control layer of *vim-emu* can be extended as shown in Fig 1. This extension is twofold. First, an *SFC controller* component is added to the emulator core which translates high-level chaining requests into low-level rules and installs them into the involved switches as described in Sec. III-B. Second, we extended *vim-emu* by additional SFC APIs that allow a user to create and configure SFPs between the deployed functions (Sec. III-C).

B. SFC Controller

The *SFC controller* is added to the emulator core and receives chaining requests from the *SFC API*. These requests contain the identifiers of the ports, i.e., network interfaces of the SFs, to be chained. The controller then translates those high-level chaining requests and installs the resulting rules in the switches of the emulated topology. This process uses the available knowledge about the topology to calculate the shortest path (using number of hops or emulated link delays) between the two SFs that are about to be chained. It generates the so-called rendered service path (RSP) [2], containing a list of all SFs and SFFs that should be traversed by the packets assigned to the path. Our current prototype simplifies the SFC model at this point by only using the first available path between two SF ports, instead of supporting multiple paths and additional load balancing features like defined by the IETF [2].

In our prototype, we opted for a custom implementation for this SFC controller based on the Ryu SDN controller [23]. We picked this approach over existing SFC implementations provided by SDN controllers like OpenDaylight [24], firstly because they are heavyweight and would destroy the lightweight nature of the presented platform. And secondly, their SFC implementations focus on production-grade systems and are tailored to interface with existing cloud deployments, e.g., they deploy their own OvS instances as SFFs which does not fit to our needs.

C. SFC Application Programming Interface (API)

As *vim-emu* already comes with APIs that mimic the OpenStack northbound interface to start, stop, and configure SFs inside the emulated PoPs, we aligned our chaining API to OpenStack as well. As a result, we added a new REST interface to *vim-emu* that offers OpenStack-like API endpoints to create, list, update, and delete `port_pairs`, `port_pair_groups`, as well as `port_chains`. Using OpenStack's SFC model [25], a user of our prototyping platform can configure SFPs by creating `port_pairs` between the network interfaces of the deployed SFs, group them to `port_pair_groups`, and finally chain them to a `port_chain`. With this, we satisfy requirement (iv) and make our platform compatible to existing MANO solutions that can consume this OpenStack-like SFC API. To support inter-PoP chaining, we provide a single chaining API endpoint for all PoPs of an emulated network, instead of providing one endpoint per PoP.

D. Simplified Prototyping using Pre-packed SFC Components

Having a prototyping platform for NSH-enabled SFs available is already helpful. But we noticed that some easy-to-use, pre-packed NSH-enabled SFs would support developers to quickly setup NSH experiments and to experiment with complex SFCs. As a result, we added a pre-packed NSH-enabled forwarder SF as well as a basic NSH traffic generator SF to our prototype. The forwarder receives NSH-encapsulated packets on its network interface, logs it for debugging and analysis purpose, decrements the service index (SI) field in the NSH according to the IETF defined behavior [3], and outputs the packets on its network interface. The traffic generator SF can be used to generate flows of NSH-encapsulated packets with a given rate and configurable NSH contents. Both SFs are available as Docker images and can directly be deployed on our prototyping platform. We used them for the evaluation presented in the next section.

IV. Evaluation

Using our platform prototype, we performed a qualitative evaluation to verify that our platform works and correctly forwards NSH-encapsulated traffic. To do so, we created a complex SFC that consists of one traffic generator (TG) and five forwarder SFs which are deployed on two PoPs as shown in Fig. 3. The traffic generator is located at the beginning of the SFC and mimics an SFC classifier [2] that encapsulates incoming traffic with NSHs. We then establish a total of six different SFPs among the available SFs as shown by the colored lines in Fig. 3: SFP1 (red), SFP2 (green), SFP3 (blue), SFP4 (red dashed), SFP5 (blue dashed), and SFP6 (green dashed). These paths are grouped (by color) to pairs and the dashed paths represent alternatives to the solid paths. They are used to show how our platform can redirect packets when their SFP identifier is changed during an experiment. SF3 plays a

special role in this setup, since this SF can reclassify packets from SFP2 to SFP6 and thus redirect the traffic from SF4 to SF5. We exploit this feature in our evaluation to demonstrate reclassification at runtime and the use of NSH metadata in our platform.

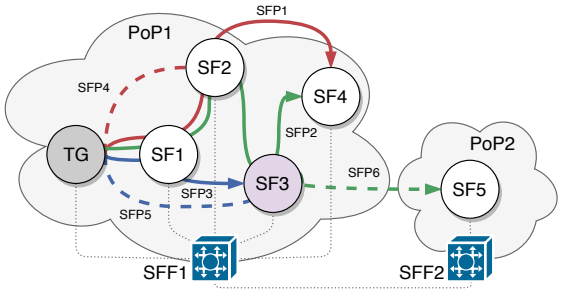

Fig. 3: Evaluation experiment setup over two PoPs: TG injecting packets into SFC with five SFs and six different SFPs.

During an experiment, the traffic generator generates a pre-defined amount of packets with a given rate for each of the SFPs and sends them to the SFC. Detailed numbers about the generated traffic are shown in Tbl. I.

TABLE I: Generated traffic per SFP

color	path id.	rate	packets sent	∑ packets sent
red	SPF1 SPF4	8 pkt/s 8 pkt/s	60 240	304
blue	SPF3 SPF5	24 pkt/s 24 pkt/s	380 200	580
green	SPF2 SPF6	16 pkt/s 16 pkt/s	440 0	451

In each SF, received packets are identified by their SFP identifier and the number of packets seen per SFP is logged. This allows us to verify that the right number of packets, belonging to the right SFP, traverses the correct set of SFs. Fig. 4 shows these counters for each SF and verifies that the expected amount of packets has been seen (indicated by horizontal lines). It shows that during the experiment, no packets were lost nor forwarded to the wrong SFs. It is important to note that this experiment focuses on verifying the correct forwarding behavior of the platform. We intentionally do not measure the maximum forwarding performance of the SFC because this only depends on the performance of the used OvS switches in the system and is out of the scope of this paper. A user of our platform can in general expect that the forwarding performance of the tested SFCs is similar to the performance achieved by Containernet [20] or Mininet [12] experiments not using NSH that are executed on a similar machine.

Once verified that the total number of packets seen by each SF is correct, we investigate a more dynamic scenario in which we change the assigned SFPs of the generated traffic and activate the reclassification in SF3 at pre-defined points in time. We identify those changes with the following events:

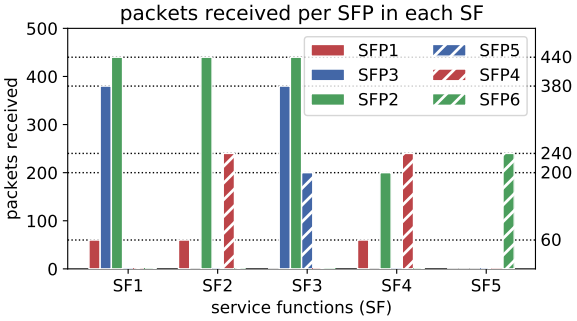

Fig. 4: Total packets received per SFP in each SF during the experiment and the expected values (dotted lines).

e_1: The traffic generator stops the generation of packets for SFP1 and starts to generate packets for SFP4 ⇒ SF1 does not see red packets anymore. SF2 and SF4 still see the red packets.

e_2: SF3 starts to reclassify SFP2 packets to SFP6 packets ⇒ SF4 does not see green packets anymore and SF5 starts to receive green packets.

e_3: Generation of SFP3 packets is stopped and SFP5 traffic is generated instead ⇒ SF1 does not see blue packets anymore and SF3 still sees blue packets.

Fig. 5 contains these events and shows the counters for the packets seen per SFP over the total number of generated packets for each of the SFs in the experiment. It verifies that the correct number of packets arrives at the correct SFs. It also shows how the three events impact the flow of the packets in the system, e.g., how packets marked with SFP1 and SFP2 disappear in SF1 at event e_1 and e_3, respectively.

A special case in this experiment is marked by event e_2. At this point in time, SF3 starts to reclassify packets marked with SFP2 and changes their identifier to SFP6. As a result, the involved SFFs forward the packets to SF5 instead of SF4. This also explains why we do not need to generate SFP6 traffic as shown in Tbl. I. To trigger the event in SF3, we exploit the metadata field of NSH: Once SF2 has seen more than 200 packets of SFP2, it sets a flag in the NSH metadata field of the following packet. SF3 then reacts to this flag in the metadata field and starts the reclassification. This example shows how an SFC with dynamic reclassification mechanisms can be prototyped in our platform and how developers can easily play with the advanced features of NSH, e.g., metadata transport between SFs. Finally, our evaluation shows that the developed prototype works correctly and can be used to locally prototype complex SFCs.

V. Conclusion

NSH can be considered as one of the key enablers for the wide adoption of SFC. Using our novel prototyping platform, researchers and developers are enabled to quickly prototype NSH-enabled components or test novel service management systems against an easy-to-deploy, NSH-enabled platform. The

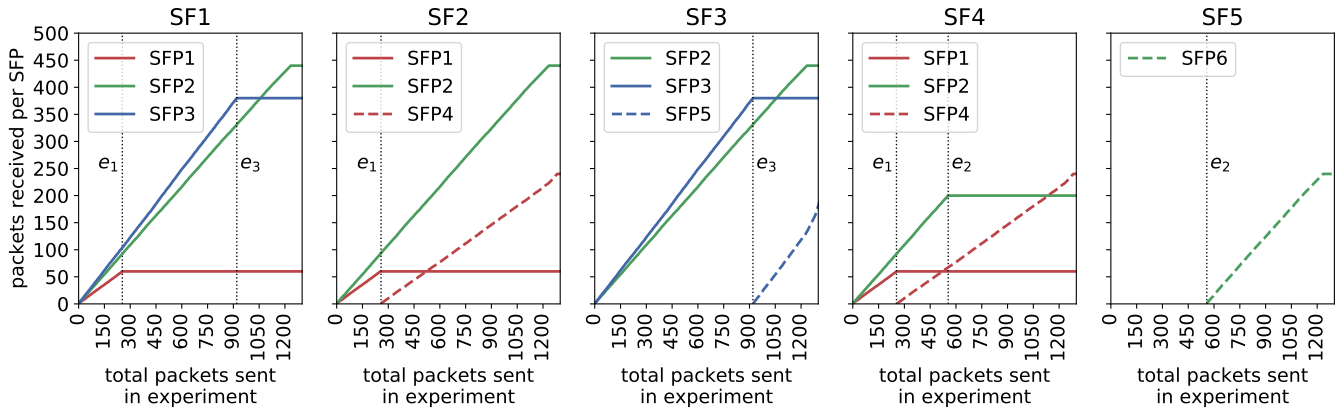

Fig. 5: Packets received per SFP over the total number of packets sent to the experiment SFC. One plot per SF and vertical markers for events e_1, e_2, and e_3.

presented platform is very lightweight and still allows for experiments with complex SFCs using advanced NSH features, such as metadata-triggered reclassification, as shown in our evaluation. We published[1] our platform under Apache 2.0 license and plan to continue its development, e.g., by adding support for load-balanced SFPs.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. H2020-ICT-2016-2 761493 (5GTANGO), and the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901).

References

[1] T. Nadeau and P. Quinn, "Problem Statement for Service Function Chaining," IETF, Internet Request for Comments RFC 7498, 2015.

[2] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," IETF, Internet Request for Comments RFC 7665, 2015.

[3] P. Quinn, U. Elzur, and C. Pignataro, "Network Service Header (NSH)," IETF, Internet Request for Comments RFC 8300, 2018.

[4] P. Quinn and J. Guichard, "Service Function Chaining: Creating a Service Plane via Network Service Headers," *IEEE Computer*, vol. 47, no. 11, pp. 38–44, 2014.

[5] A. Medhat, T. Taleb, A. Elmangoush, G. Carella, S. Covaci, and T. Magedanz, "Service Function Chaining in Next Generation Networks: State of the Art and Research Challenges," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 216–223, 2017.

[6] M. Beck, J. Botero, and K. Samelin, "Resilient Allocation of Service Function Chains," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[7] A. Medhat, G. Carella, M. Pauls, M. Monachesi, M. Corici, and T. Magedanz, "Resilient Orchestration of Service Functions Chains in a NFV Environment," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[8] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On Dynamic Service Function Chain Deployment and Readjustment," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 543–553, 2017.

[9] G. Davoli, W. Cerroni, C. Contoli, F. Foresta, and F. Callegati, "Implementation of Service Function Chaining Control Plane through OpenFlow," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017.

[10] 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)*, 2016, pp. 148–153.

[11] M. Peuster, M. Marchetti, G. G. de Blas, and H. Karl, "Emulation-based Smoke Testing of NFV Orchestrators in Large Multi-PoP Environments," in *2018 IEEE European Conference on Networks and Communications (EuCNC)*, 2018.

[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.

[13] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. H. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *Networking Conference, 2014 IFIP*. IEEE, 2014, pp. 1–9.

[14] L. Mamatas, S. Clayman, and A. Galis, "A service-aware virtualized software-defined infrastructure," *Communications Magazine, IEEE*, vol. 53, no. 4, pp. 166–174, 2015.

[15] I. Pelle, T. Lévai, F. Németh, and A. Gulyás, "One tool to rule them all: A modular troubleshooting framework for sdn (and other) networks," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, ser. SOSR '15. ACM, 2015, pp. 24:1–24:7.

[16] A. Csoma, B. Sonkoly, L. Csikor, F. Németh, A. Gulyás, W. Tavernier, and S. Sahhaf, "Escape: Extensible service chain prototyping environment using mininet, click, netconf and pox," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 125–126, 2015, https://sb.tmit.bme.hu/escape/.

[17] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[18] W. Zhao, Y. Peng, F. Xie, and Z. Dai, "Modeling and simulation of cloud computing: A review," in *Cloud Computing Congress (APCloudCC), 2012 IEEE Asia Pacific*, 2012, pp. 20–24.

[19] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, 2008.

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

[21] ETSI OSM, "Open Source MANO: Open Source NFV Management and Orchestration (MANO) software stack aligned with ETSI NFV," Website, 2016, online at https://osm.etsi.org.

[22] F. T. Christ, "Docker-based Emulation of Service Function Chains Using Network Service Header," Bachelor's Thesis, Paderborn University, 2018.

[23] Ryu SDN Framework Community, "Ryu," https://osrg.github.io/ryu/, 2017.

[24] Linux Foundation, "OpenDaylight," https://www.opendaylight.org, 2018.

[25] OpenStack Project, "OpenStack SFC API," https://docs.openstack.org/networking-sfc/latest/contributor/api.html, 2018.

[1] vim-emu with NSH support: https://git.io/vim-emu-nsh