A Generic Emulation Framework for Reusing and Evaluating VNF Placement Algorithms

Stefan Schneider, Manuel Peuster, Holger Karl
Paderborn University, Paderborn, Germany
{stefan.schneider, manuel.peuster, holger.karl}@uni-paderborn.de

***Abstract*—In recent years, a variety of different approaches have been proposed to tackle the problem of scaling and placing network services, consisting of interconnected virtual network functions (VNFs). This paper presents a placement abstraction layer (PAL) that provides a clear and simple northbound interface for using such algorithms while hiding their internal functionality and implementation. Through its southbound interface, PAL can connect to different back ends that evaluate the calculated placements, e.g., using simulations, emulations, or testbed approaches. As an example for such evaluation back ends, we introduce a novel placement emulation framework (PEF) that allows executing calculated placements using real, container-based VNFs on real-world network topologies. In a case study, we show how PAL and PEF facilitate reusing and evaluating placement algorithms as well as validating their underlying models and performance claims.**

I. Introduction

Management and orchestration (MANO) of network services are an integral part of network function virtualization (NFV). MANO systems deploy network services, consisting of interconnected virtual network functions (VNFs), by placing the individual VNFs at different nodes in the network. To tackle this placement problem, a wealth of optimization problems and algorithms have been formulated in recent years [1], [2], [3], [4].

Currently, it is difficult to use these algorithms in practice since their usage (e.g., inputs and outputs) and capabilities vary strongly. Some algorithms focus on the mapping of VNFs to the substrate network and routing of traffic between them [1], [2], [5], others also consider scaling [4], [6], i.e., dynamically deciding the number of VNF instances (horizontal scaling) and their assigned resources (vertical scaling). Selecting among the available algorithms can be challenging since their realistic performance cannot be easily tested and validated. While performance can, in principle, be realistically evaluated using network emulators, most network emulators [7], [8], [9] do not support important aspects of NFV such as service function chaining (SFC). Moreover, it typically requires significant manual effort to start the calculated placements of different algorithms in emulation as it requires retrieving the relevant placement information, starting the network emulation, deploying VNFs, and interconnecting them correctly. Hence, only few authors of placement algorithms use emulation for their evaluation [3], [10].

Similarly, it is difficult for researchers to reproduce and compare results from different placement algorithms since preparing inputs and analyzing outputs in the custom format of each algorithm is cumbersome and time-consuming. Furthermore, the various algorithms use different model assumptions, e.g., regarding characteristics of network services or the network infrastructure, leading to different, incommensurable performance claims that cannot be meaningfully compared. For example, the end-to-end delay of placed services is often assumed to be equal to the sum of edge delays along the paths of connected VNFs [1], [2], [4]. Other placement algorithms use more sophisticated models, taking more real-world effects into account, e.g., VNF processing [5], scheduling [11], queuing and virtualization [12], or synchronization [13]. Hence, even for the same calculated placement, different algorithms may claim different end-to-end delays. It is non-trivial to validate these model assumptions and their resulting claims since existing simulation or emulation platforms do not easily support realistic evaluation of calculated placements.

To overcome these shortcomings, we present two main contributions. First, we define a *simple, generic placement abstraction layer (PAL)* with a northbound and a southbound interface (Sec. III). The northbound interface defines a consistent structure of inputs and outputs for interacting with arbitrary placement algorithms while hiding their internal complexity. The interface is simple, independent of a placement algorithm's functionality, and easy to implement. At the southbound interface, evaluation back ends connect to PAL to support realistic simulation, emulation, or even real testbed deployment of calculated placements. PAL coordinates the exchange of information (e.g., calculated placements) between the placement algorithms and evaluation back ends, masking their internal technical details.

Our second main contribution is a novel, open-source *placement emulation framework (PEF)* that integrates with PAL as possible evaluation back end (Sec. IV). PEF allows to test calculated placements with real, container-based VNFs in emulated networks using real-world network topologies. The deployed service placements can then be exposed to synthetic or trace-based traffic that is sent through the service chain to obtain realistic performance measurements. In a case study, we use PAL and PEF to evaluate placements of three different algorithms (Sec. V). By comparing the emulation measurements with the algorithms' performance claims, we perform a "reality check" of the calculated placements and the model assumptions of the different placement algorithms.

II. Related work

In recent years, the problem of VNF placement has been formalized and tackled by many authors, leading to a variety of different optimization problems and algorithms [14]. So far, there is no generic interface that abstracts the internals of these placement algorithms while clearly and consistently defining their inputs and outputs. Only the SONATA MANO system [15] internally defines an interface for the communication with placement algorithms through its placement plugin. However, this interface is tailored to the needs of SONATA, e.g., using their specific service descriptors. As opposed to this, the northbound interface of our proposed PAL is more generic and enables simple interaction with any placement algorithm. Similarly, our interface allows much smaller and simpler service specifications than the standardized approach by ETSI [16]. Their service specification is very comprehensive, trying to define all possible service details, but the resulting service descriptors tend to be complex and often contain details that are irrelevant for VNF placement algorithms (e.g., monitoring information). Our approach focuses on a minimal specification of the most relevant information for VNF placement but can be extended flexibly. For example, this also allows to include information from an existing ETSI network service descriptor or a reference to it.

Furthermore, PAL allows connecting generic back ends to evaluate calculated placements. As an example, we provide PEF, a powerful placement emulation framework that extends the MeDICINE emulation platform [17], [18]. In contrast to other generic network emulators [7], [8], [9], MeDICINE focuses on typical NFV scenarios with multiple points of presence (PoPs) on which arbitrary VNFs can be deployed. It supports the execution of real-world VNFs, given as Docker containers that are interconnected into complex forwarding graphs, on arbitrary, user-defined multi-PoP topologies.

Due to missing support for placement algorithms, only few authors have expended the effort to use generic network emulators to evaluate their placement algorithms. For example, Ma et al. [3] have built a custom emulation environment for an extensive evaluation of their placement algorithm, using Mininet [7] and implementing their own placement module for an SDN controller. Gebert et al. [10] demonstrate their placement algorithm focusing on wireless LTE networks and using a proprietary emulator by Nokia. These placement emulation prototypes are tailored to the specific use case of the authors and are not accessible to others. In contrast, our PEF is openly available under Apache 2.0 license [19] and can easily be used with any placement algorithm through the abstraction of PAL.

III. Placement abstraction layer (PAL)

Fig. 1 provides an overview of the architecture and the workflow with our proposed PAL. Attached placement algorithms that implement the northbound interface can be triggered in a uniform manner through PAL. PAL sends placement requests and the algorithms return a placement response with the calculated placement. Leveraging the consistent structure

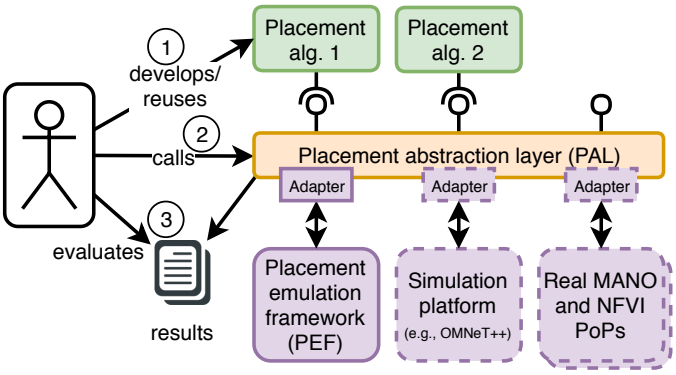

Fig. 1. Architecture and workflow overview

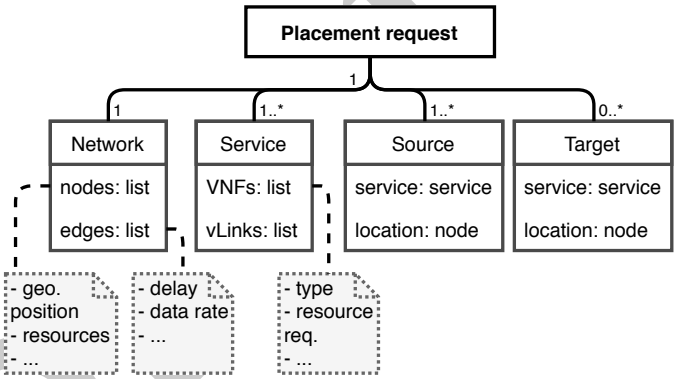

Fig. 2. Structure of a placement request

of placements, different evaluation back ends can easily be connected to PAL through the southbound interface. As an example evaluation back end, we present an open-source placement emulation framework in Sec. IV.

A. Northbound: Interaction with placement algorithms

Based on an analysis of various existing placement algorithms, we define placement requests and responses to contain only few mandatory fields that represent the quintessence of the varying information required by different algorithms (e.g., involved VNFs and their interconnections). This allows to keep placement requests and responses simple and ensures a consistent structure of inputs that are minimally required to perform VNF placement. Hence, the same placement request can be used to trigger different algorithms. The mandatory information can be extended by adding further, optional fields that are only required by some algorithms (e.g., certain VNF details). Algorithms can then simply ignore any additional information that they do not need. Conversely, they should inform users if they need more details to be included in a placement request.

1) Placement request: Fig. 2 shows the structure of a placement request sent through PAL to trigger a placement algorithm. Each request must specify a network, at least one service, and at least one source. The gray notes illustrate optional annotations containing further input details.

The underlying *network* $G = (V, E)$ consists of nodes V and edges E. Nodes represent NFVI PoPs at different geographical locations, interconnected by edges. Additional, optional fields may specify the geographic location or compute capacities of nodes and the data rate or delay of edges. For implementation, we suggest using the GraphML format, which is used by the popular libraries TopologyZoo [20] and SNDlib [21] and can easily be converted into other formats.

Network services consist of VNFs that are interconnected by virtual links (vLinks). For placement algorithms that support scaling, a specification of different involved VNF components is sufficient. For algorithms without scaling support, the service specification needs to include a list of already scaled VNF instances. VNFs can be annotated with VNF type, resource requirements, etc. vLinks specify which VNFs are interconnected, thus defining the VNF forwarding graph. Each vLink is unidirectional with a source and a destination VNF. To specify bidirectional connections, two vLinks must be used.

Sources, e.g., users or sensors, are located at different nodes in the network, requesting a service. Thus the list of sources contains the location and the requested network service per source. Additionally, they can define traffic characteristics, e.g., the rate and type of requests leaving each source or their combined data rate. Placement requests can also contain further optional inputs. This allows, for example, to specify fixed target locations where placed service chains need to end.

2) Placement response: After receiving a placement request, placement algorithms calculate a placement response and return it to PAL. Fig. 3 shows the structure of such a placement response, mapping VNF instances to network nodes and specifying to which network service they belong. Since the number of VNF instances and their interconnections (vLinks) may be decided dynamically by placement algorithms that support scaling, the placement response also needs to specify the source and destination of these vLinks. This mandatory information about VNF mapping and vLinks may be extended by further annotations, e.g., vLinks may specify the calculated routes between interconnected VNFs.

Placement responses may contain performance claims about the calculated placement, e.g., specifying the expected delay between connected VNFs and on the end-to-end service chain. Placement algorithms compute these performance claims based on their internal model assumptions. These claims can then be validated using an evaluation back end.

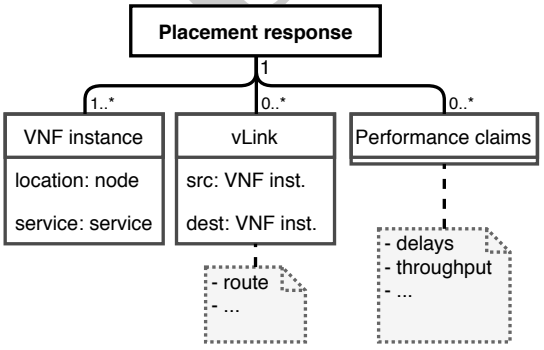

Fig. 3. Structure of a placement response

B. Southbound: Interaction with evaluation back ends

PAL allows attaching generic back ends to its southbound interface that can be used to evaluate calculated placements. These evaluation back ends are isolated from the placement algorithms through PAL such that they do not have to interact with the algorithms directly. Instead, users choose which placement algorithm and evaluation back end to use and PAL coordinates the exchange of placement information.

Each evaluation back end connects to PAL through a small but important adapter component that starts the calculated placements on the back end. It does so by translating the consistently defined VNF mapping and interconnections into corresponding instructions for the back end. For example, the adapter of our novel PEF starts VNF containers and configures their network (see Sec. IV). The adapter can also use optional fields, e.g., the VNF type, but should provide default values if a field is not available in a placement response. In doing so, the placement algorithms do not have to deal with technical details that are required to turn an abstract placement result into an executable emulation scenario.

Many evaluation back ends are conceivable in addition to PEF, e.g., simulation platforms, like OMNeT++ [22]. Even globally distributed testbeds, like SoftFIRE [23], or custom testbeds based on MANO systems, like OSM [24], could be connected to enable placement experiments on real NFVI PoPs. Hence, PAL allows to flexibly combine different placement algorithms with various back ends to evaluate the calculated placements.

IV. Placement Emulation Framework (PEF)

PEF is a novel, multi-PoP placement emulation framework that integrates as possible evaluation back end for PAL. PEF emulates calculated placements, enabling realistic measurements (Sec. IV-A). To this end, we extended our existing network emulation platform [17] by providing an adapter that processes the received placement requests and responses, automatically start and interconnect the involved VNFs, and coordinates performance measurements (Sec. IV-B). We provide an open-source prototype of PEF available under Apache 2.0 license on GitHub [19].

A. Emulation and measurements

After receiving a placement from PAL, the adapter starts the emulation using the underlying multi-PoP NFVI emulation platform [17] as shown in Fig. 4. The bottom layer of the emulation platform is able to emulate arbitrarily complex multi-PoP topologies, including realistic inter-PoP delays and bandwidth limits, e.g., taken from TopologyZoo [20]. Each node in an emulated topology represents an NFVI PoP and offers interfaces to start and stop container-based (Docker) VNFs. The use of lightweight container technology allows the platform to run hundreds of VNFs on a single physical

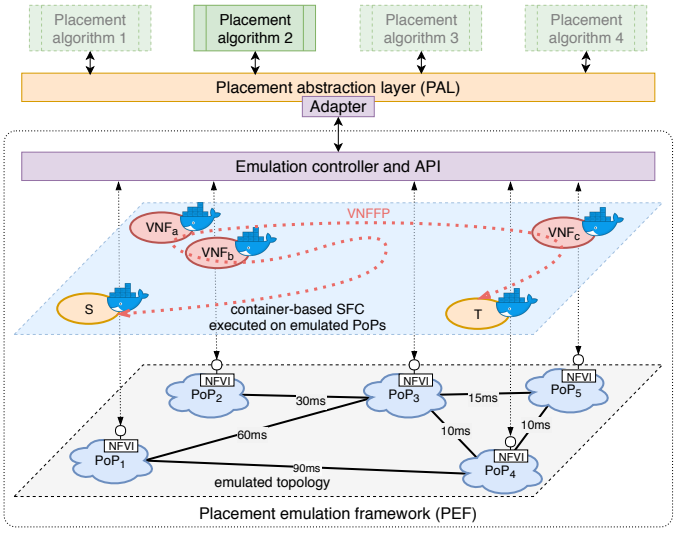

Fig. 4. PEF uses the multi-PoP NFVI emulation platform *MeDICINE* [17] and connects to PAL through an adapter.

machine or VM [25]. Running real Docker containers, it allows to execute arbitrary, real-world VNF software, e.g., firewalls, proxies, or intrusion detection systems.

Once the VNFs are started inside the emulated PoPs, they are connected to complete service function chains using the APIs of the emulation platform (Fig. 4, middle layer). This chaining process is based on the custom routing computed by the used placement algorithm. Our chaining approach uses VLAN tagging to isolate different VNF forwarding paths and allows to create chains that reach across any number of PoPs. To do so, PEF automatically configures the involved SDN switches along the specified paths. Hence, the underlying emulation platform is a perfect fit for placement experiments because it provides full control over the location of VNFs as well as the traffic steering and chaining – going beyond existing emulation solutions.

After PEF starts and interconnects the VNFs in the emulation platform, users can execute arbitrary commands on the running VNFs, even installing new software packages. In doing so, users can perform any kinds of measurements with the placed network service, e.g., to confirm its functionality or to assess its performance. For example, *iperf* allows to generate traffic, *ping* can be used to measure delays between placed VNFs, and *httping* measures the end-to-end delay of a placed service using HTTP requests. These measurements provide realistic insights as they take side effects into account, e.g., introduced by the involved protocol stacks and VNF software components, which are often not considered in the models of placement algorithms. Users may even imitate and observe the effect of failures by deliberately removing individual VNFs or vLinks of a running service deployment.

B. Adapter for placement processing

To start the calculated placements on the emulation platform, PEF comes with an adapter that processes the placements and corresponding network topology as follows.

For realistic network emulation, all network edges of a given topology need to specify an edge delay. Alternatively, PEF can use annotated geographical node positions from real-world network topologies (e.g., from TopologyZoo [20]) to calculate the distance between interconnected nodes and then derive the corresponding edge delays. If neither are available, PEF defaults to edge delays of 0 ms.

According to the specified type of placed VNFs, PEF selects corresponding Docker containers to execute in the emulation (e.g., firewalls, forwarders, or web servers). PEF comes with a pre-configured Socat[1] layer-4 forwarding VNF and an Apache[2] web server, which can be used to build service chains of variable length if no VNF type is specified.

PEF automatically configures the network of the started VNFs if no configuration details are specified in the calculated placement. If a placement does not specify routes between interconnected VNFs, PEF automatically computes and uses the shortest paths according to edge delays.

Finally, the adapter coordinates automatic performance measurements, by running commands like *ping* on the different deployed VNFs, logging the results, and converting them into convenient YAML format.

V. Case Study

In the following case study, we use three different placement algorithms with varying inputs to illustrate how PAL simplifies the evaluation process of different algorithms (Sec. V-A). We also demonstrate the capabilities of PEF by emulating the calculated placements using real VNFs running on real-world network topologies. We measure the delays of the deployed services in emulation and compare these realistic delays with the algorithms' performance claims (Sec. V-B).

A. Simplified evaluation process

Whether developing new placement algorithms or adjusting an existing one, the simple northbound interface of PAL can be implemented with little effort since it is independent of the algorithm's functionality. We demonstrate this by integrating three placement algorithms with PAL's interface: One greedy placement algorithm that minimizes delays between interconnected VNFs and one dummy algorithm that places VNFs randomly. Furthermore, we adjust the existing placement heuristic by Dräxler et al. [4] (called bjointsp) to implement the interface – again, only having to adjust the processing of inputs and outputs, not the algorithm's functionality itself (affecting less than 10% of the algorithm's code).

As underlying network, we use three real-world network topologies taken from TopologyZoo [20]: A small network with 11 nodes and 14 edges, a medium-sized network with 16 nodes and 37 edges, and the largest network in the TopologyZoo with 197 nodes and 245 edges. PEF's adapter automatically calculates and sets the delays of all edges. Furthermore, we emulate resource limits at the involved PoPs

[1]Socat man page: https://linux.die.net/man/1/socat (accessed May 4, 2018)
[2]Apache HTTP Server: https://httpd.apache.org/ (accessed May 4, 2018)

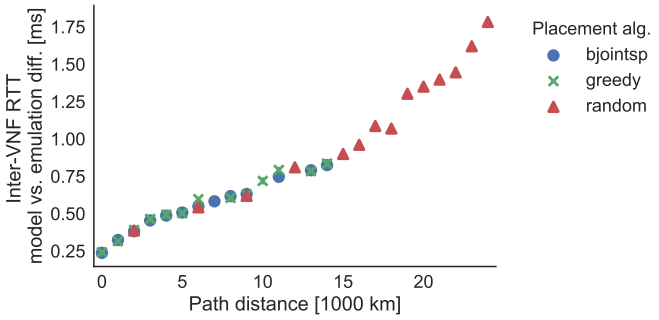

Fig. 5. With increasing distance, the difference increases between measured inter-VNF RTTs in emulation and claimed RTTs based on the model.

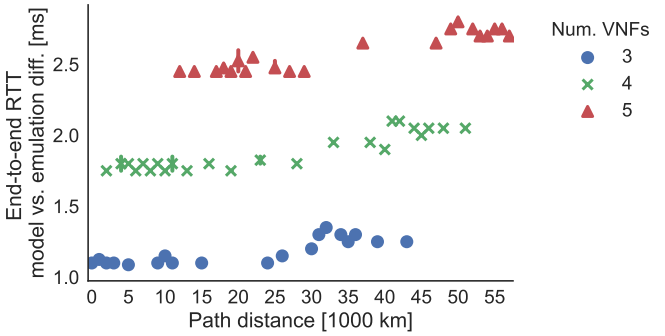

Fig. 6. The emulation vs. model difference for the end-to-end RTT mostly depends on the number of VNFs in the deployed SFCs.

by setting node capacities such that each node can only host at most one VNF at a time.

As network services, we use a linear chains of layer-4 forwarding VNFs and a web server VNF at the end of the chain (three to five VNFs in total). For each service and network, we evaluate placements calculated by each algorithm with sources at different random locations.

To execute the case study, we used a machine with an Intel Xeon E5-1660 v3 processor and 32 GB RAM. On this machine, the time to boot and configure our emulation framework was always on the order of seconds or minutes. Even scenarios with TopologyZoo's largest network with 197 nodes could be started in 2-3 min.

Instead of the numerous manual steps that are typically required for evaluation (e.g., preparing separate inputs for each algorithm, collecting and parsing results, manually starting the emulation, etc.), PAL and PEF automated and greatly simplified the whole evaluation process.

B. Placement emulation insights

In this case study, we do not focus on the quality of the calculated placements but investigate the realism of the algorithms' underlying models and corresponding performance claims. Here, the algorithms model the round-trip times (RTTs) between interconnected VNFs as the sum of edge delays. In fact, the RTTs between interconnected VNFs based on this model are very similar to the measured emulation RTTs, i.e., their absolute difference is very small (0-2 ms in the medium network). While this means that the claimed RTTs based on the model are already quite realistic for the evaluated service placements, there are still systematic differences between the modeled RTTs and the measured RTTs in emulation.

Fig. 5 shows this difference for the RTT between interconnected VNFs in relation to their distance (for placements in the medium-sized network). For VNFs placed close together, the RTTs measured in emulation are very similar to the claimed RTTs based on the model, i.e., the difference is close to 0 ms. However, for increasing distance between two VNFs, model and emulation RTTs diverge, leading to an increasing difference. For placements with faraway VNFs, mostly calculated by the random placement algorithm, the

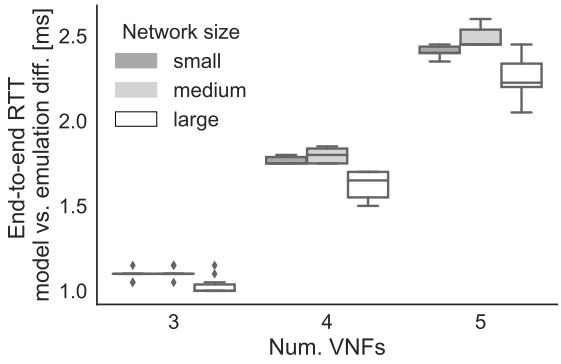

Fig. 7. In all three networks, the model vs. emulation difference for the end-to-end RTT clearly increases for SFCs with more VNFs.

inter-VNF RTT measured in emulation is 1-2 ms higher than the modeled RTT. This difference is caused by the additional PoPs and switches that are traversed in longer paths, which are not considered by the model but lead to higher RTTs in emulation.

Considering the overall end-to-end RTT of the deployed service function chains (SFCs), there are also small but systematic differences between the measured emulation RTTs and the modeled RTTs based on the sum of edge delays. Fig. 6 shows the emulation vs. model difference for the end-to-end RTT in relation to the total interconnection distance of each deployed SFC. It also indicates the number of involved VNFs in each SFC (here, 3, 4, or 5 VNFs). As previously observed for connections between two VNFs, longer SFCs (in km) also lead to a slightly higher emulation vs. model RTT difference. However, the main cause for divergence between end-to-end emulation and model RTT is the number of VNFs in an SFC. As each VNF introduces some processing delay, which is not considered in the model used here, the end-to-end RTT difference increases step-wise with a higher number of VNFs in an SFC. As illustrated in Fig. 7, this effect can be confirmed for placements in all three network topologies used in this case study (here produced by the greedy algorithm).

VI. Conclusion

The presented case study illustrates an example use case of PAL and PEF, emulating placements produced by different placement algorithms and comparing the emulation measurements with the algorithms' performance claims. For the given service placements, the claimed delays based on the sum of edge delays were already quite close to the emulation measurements. Nevertheless, the evaluation still revealed systematic differences between claimed and measured delays. This insight can be used to improve the model assumptions, e.g., by including appropriate VNF processing delays or additional delays for longer paths between VNFs. Especially when dealing with more complex models or services (e.g., considering various queuing, processing, or virtualization delays), the emulation with PEF is helpful to either confirm the correctness of the used models or to uncover shortcomings that lead to divergence between claimed performance and emulation measurements.

PEF allows researchers to gain practical hands-on experience when working with their calculated placements. This allows to identify possibly unexpected side effects of real deployments. For example, the TCP 3-way handshake leads to additional delay when setting up new flows but is typically neglected in the models of placement algorithms. Similarly, operators interested in using a placement algorithm for VNF deployment, can easily test the algorithm and validate its performance using PEF. While the emulation with PEF already allows realistic placement evaluation, integrating other evaluation back ends with PAL surely provides further interesting evaluation insights.

Acknowledgments

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

References

[1] S. Mehraghdam, M. Keller, and H. Karl, "Specifying and placing chains of virtual network functions," in *IEEE CloudNet*, 2014.

[2] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *IFIP/IEEE CNSM*, 2015.

[3] W. Ma, J. Beltran, Z. Pan, D. Pan, and N. Pissinou, "SDN-based traffic aware placement of NFV middleboxes," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 14, 2017.

[4] S. Dräxler, S. Schneider, and H. Karl, "Scaling and placing bidirectional services with stateful virtual and physical network functions," in *IEEE NetSoft*, 2018.

[5] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE IM*, 2015.

[6] S. Dräxler, H. Karl, and Z. A. Mann, "JASPER: Joint optimization of scaling and placement of virtual network services," *IEEE Transactions on Network and Service Management (TNSM)*, 2018.

[7] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

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

[9] 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.

[10] S. Gebert, D. Hock, T. Zinner, P. Tran-Gia, M. Hoffmann, M. Jarschel, E. Schmidt, R. Braun, C. Banse, and A. Köpsel, "Demonstrating the optimal placement of virtualized cellular network functions in case of large crowd events," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, 2014.

[11] L. Qu, C. Assi, and K. Shaban, "Delay-aware scheduling and resource optimization with network function virtualization," *IEEE Transactions on Communications*, vol. 64, no. 9, 2016.

[12] D. B. Oljira, K.-J. Grinnemo, J. Taheri, and A. Brunstrom, "A model for QoS-aware vnf placement and provisioning," in *IEEE NFV-SDN*, 2017.

[13] S. Schneider, A. Sharma, H. Karl, and H. Wehrheim, "Specifying and analyzing virtual network services using queuing petri nets," *ArXiv e-prints*, 2018.

[14] J. Herrera and J. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, no. 3, 2016.

[15] SONATA Consortium, "SONATA H2020 Project," online at: sonata-nfv.eu, 2018.

[16] ETSI NFV ISG, "Network functions virtualisation (NFV); management and orchestration; network service templates specification," ETSI, Group Specification ETSI GS NFV-IFA 014 V2.1.1, 2016.

[17] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *IEEE NFV-SDN*, 2016, pp. 148–153.

[18] 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 *IEEE NetSoft*, 2017, pp. 1–3.

[19] S. Schneider and M. Peuster, "Placement emulation framework prototype," https://github.com/CN-UPB/placement-emulation, 2018.

[20] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[21] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *3rd International Network Optimization Conference (INOC)*, 2007. [Online]. Available: sndlib.zib.de

[22] A. Varga, "OMNeT++: Discrete event simulation system," in *European Simulation Multiconference*, 2001. [Online]. Available: https://omnetpp.org/

[23] SoftFIRE consortium, "Constructing a federated testbed and an orchestrated virtualisation infrastructure," Technical Overview, 2017.

[24] A. Israel, A. Hoban, A. T. Sepúlveda, F. J. R. Salguero, G. G. de Blas, K. Kashalkar, M. Ceppi, M. Shuttleworth, M. Harper, M. Marchetti, R. Velandy, S. Almagia, and V. Little, "OSM release three," Open Source MANO, Technical Overview, 2017.

[25] M. Peuster, M. Marchetti, G. García de Blas, and H. Karl, "Emulation-based smoke testing of NFV orchestrators in large multi-PoP environments," in *IEEE EuCNC*, 2018.