Profile Your Chains, Not Functions: Automated Network Service Profiling in DevOps Environments

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

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

***Abstract*—Benchmarking and profiling virtual network functions (VNF) generates input knowledge for resource management decisions taken by management and orchestration systems. Such VNFs are usually not executed in isolation but are often deployed as part of a service function chain (SFC) that connects single functions into complex structures. To manage such chains, isolated performance profiles of single functions have to be combined to get insights into the overall behavior of an SFC. This becomes particularly challenging in highly agile DevOps environments in which profiling processes need to be fully automated and detailed insights about a chain's internal structures are not always available.**

In this paper, we introduce a fully automatable, flexible, and platform-agnostic profiling system that allows to profile entire SFCs at once. This obviates manual modeling procedures to combine profiling results from single VNFs to reflect SFC performance. We use a case study with different SFC configurations to show that it is hard to model the resulting SFC performance based on single-VNF measurements and that performance interactions between real, non-trivial functions that are deployed in an chain exist.

I. Introduction

Network function virtualization (NFV) is expected to be the key enabler for agile network management in the upcoming 5th generation of networks. It allows to apply DevOps principles to network and service management processes with the overall goal to reduce turnaround times and time-to-market delays through the softwarization and virtualization of network components [1]. In such agile environments, network service function chains (SFC), chaining together virtualized network functions (VNF), are automatically managed by management and orchestration (MANO) systems. These systems decide how many virtualized resources are allocated to each function to meet pre-defined service level agreements (SLA) and ensure certain quality of service (QoS) levels. They are also responsible to re-configure SFCs and their functions at runtime, e.g., scale-up or scale-out.

To optimize their decisions, MANO systems can rely on online monitoring data or on pre-acquired knowledge about the relationship between available resource and resulting performance, so called performance profiles [2]. The second approach is especially favorable in very agile DevOps environments in which new versions of network services are directly deployed into production and thus no up-to-date monitoring data about the updated service is available.

Some initial profiling solutions have already been proposed by both the NFV and cloud computing research communities [3], [4]. However, most of the existing solutions are either tied to specific platforms on which the VNFs are profiled, or they require manual steps to setup and run profiling experiments. This complicates the integration of the profiling procedure into agile DevOps workflows. For example, it should be possible to automatically profile an SFC during its on-boarding procedure (i.e. uploading a new SFC version to an NFV platform).

Another important aspect for profiling in an NFV context is the need to consider complex SFCs and not only single, isolated functions. This is because the end-to-end performance of the entire SFC is the metric of interest, especially if the SFC is deployed on the path between end users and a customer service, e.g., between user and a content delivery network. One option to get the end-to-end SFC performance is to combine profiling results of single VNFs using a performance model of the SFC. The problem of this approach is that SFCs can consist of many VNFs that could possibly be chained in different ways, like different order of functions, different structures (branches), and different forwarding paths which makes it hard to find correct models. In addition, the required details about the SFC structure might not even be available, e.g., for proprietary (black box) SFCs. Performing this model-based approach for such proprietary SFCs would only be feasible if an SFC developer manually provides the correct models. A clear contradiction to the idea of automation. An alternative option is to develop profiling solutions that are able to profile entire SFCs end-to-end.

The contributions of this paper are two-fold. First, Section II introduces a novel NFV profiling system that addresses three shortcomings of existing approaches: *platform dependency*, *lack of automation* and *missing support for SFC profiling*. The presented solution is open-source [5] and designed to be completely platform-agnostic by using a novel descriptor-based experiment generation approach. The system can easily be adapted to be used with existing NFV platforms by providing *descriptor plugins* that generate target platform-specific service configurations and *driver plugins* used to deploy these service configurations on the target platforms. In our examples, we use the SONATA [5] description format that is aligned with the ETSI [6] NFV model.

Second, we use a case study to show that naive approaches that profile single VNFs in isolation and combine their results do not work and result in inaccurate predictions for the resulting SFC performance. Our findings, which also show that our fully automated end-to-end SFC black-box profiling approach works, are presented in Section III. Finally, Section IV compares our work with existing approaches and Section V concludes.

II. Automated Service Chain Profiling

Automated profiling of complex network services requires three key components. First, an approach to define profiling procedures. Second, an NFV platform with its infrastructure and MANO facilities used to deploy, configure, and execute the service under test (SUT). Third, a profiling controller that coordinates the profiling process and controls the test system by interfacing with the NFV platform. We identified the following requirements for such a profiling system. (R1) Automation: Allow fully automated (i.e. scriptable) profiling experiments without any human interaction after the experiment's initial description. (R2) Flexibility: Support many NFV platforms so that profiling experiments can be executed in different environments. This includes support for different control and monitoring interfaces as well as different description languages. (R3) Abstraction: Once a developer has described a profiling experiment it should be possible to execute this experiment on different target platforms. (R4) Integration: The profiling tool as such has to be integrable into different workflows, for example, to be executed inside a continuous integration and delivery (CI/CD) pipeline or to become part of a service platform's on-boarding procedure.

A. Profiling Controller Design & Workflow

We designed our profiling controller as a highly modularized system that allows to replace many of the components that interface with external systems so that it can be extended and used with any NFV platform. The key idea of this design is to utilize the available service description mechanisms offered by existing MANO systems to deploy and profile a SUT with different parameterizations and configurations instead of manipulating the target platform directly. This has the clear benefit that our system is transparent to the target NFV platforms and does not require interface changes.

Fig. 1 describes our system as well as the profiling workflow. In the first step, a user creates a *profiling experiment descriptor (PED)* that contains all necessary information to perform a profiling experiment. In particular, it references the network service that should be profiled, e.g., a service package or service descriptor, and it includes descriptions of all service configurations that should be tested, e.g., different resource assignments for the used VNFs (Sec. II-B). The PED is used to trigger our profiling system by either using its command line (CLI) or REST interfaces (step 1 in Fig. 1). These interfaces also allow the integration of our profiler into existing CI/CD workflows (R4). The profiler reads the PED and forwards the request to its *descriptor engine*. This module takes the network service referenced by the PED file and

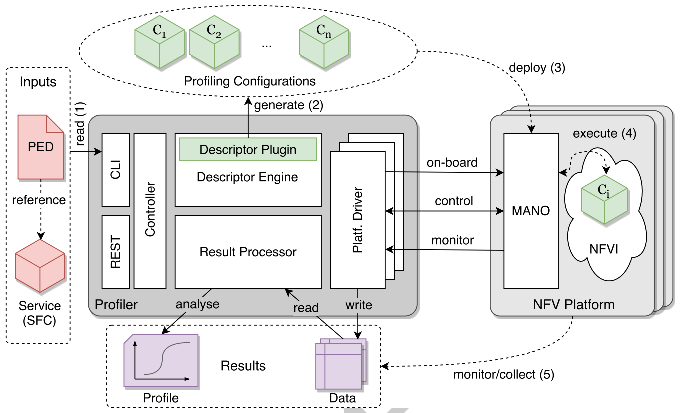

Fig. 1: System architecture of our profiling system interacting with several NFV platforms. The figure also shows the general workflow and generated artifacts, like *profiling configurations* and results.

embeds (or extends) it with additional measurement VNFs, called *measurement points* (MP). Our system offers default *measurement point* VNFs that contain standard networking test tools, like *iperf* or *hping*. A tester can replace these by any custom measurement VNF which may, for example, contain domain-specific, proprietary traffic generators. After the embedding step, one copy of the new service description, for each configuration specified in the PED, is generated (step 2). This includes resource configurations, like number of cores assigned to a VNF. The *descriptor engine* itself offers a plugin interface for service description generators so that our profiler becomes service descriptor agnostic and can be extended to further description formats, e.g. OSM [7] (R2).

In the third step, the previously generated service configurations are deployed one after the other on the target platform(s) using the *platform driver* modules (step 3). These drivers act as a client to the target platform and form an abstraction layer between specific MANO northbound interfaces and our internal control mechanisms (R3).

Once a service instance is up and running, the traffic generators in the additionally deployed *measurement point* VNFs are activated and start to stimulate the service. After this, the service instance is destroyed and removed from the platform before the next service configuration is deployed. We call the deployment, execution, and test of a single service configuration a *profiling round*.

During a *profiling round*, performance data is collected in two ways (step 4). First, service-internal performance metrics are monitored, including log files inside the *measurement points* and VNFs (if enabled). Second, platform metrics, like packet counters on virtualized interfaces, are collected through the platform's monitoring APIs. The latter are platform-specific and may not be available on each target platform.

As a last step (step 5), all measured data, collected from various sources, is aggregated and stored in a unified, table-based format. Each row in this table represents exactly one of the tested service configurations. These tables are then passed to a post-processing module which automatically triggers user-defined analysis scripts that can, for example, perform statistical analysis on the collected datasets (R1).

B. Describing Profiling Experiments

One of our key contributions is an easy-to-understand, human-readable description format used to describe profiling experiments. Listing 1 shows a small part of such a YAML-based descriptor. In the header (line 1-2), the descriptor contains general information and version fields. The PED also contains an URL to the service definition which can be in any format, for example, a SONATA service package (line 5).

```
1  descriptor_version: 0.1
2  name: "example-profiling-experiment1"
3  # (...)
4  service_experiments:
5    service_url: "file://data/example-service.son"
6    - name: "service_throughput_tcp"
7      repetitions: 8
8      time_limit: 120 # seconds
9      service_id: "eu.sonata-nfv.example-service.0.1"
10     measurement_points:
11       - name: "mp.input"
12         connection_point: "ns:input"
13         image: "default-mp:latest"
14         cmd_start: "iperf -c mp.output"
15       - name: "mp.output"
16         connection_point: "ns:output"
17         # (...)
18     resource_configurations:
19       - function_id: "eu.sonata-nfv.proxy-vnf.0.1"
20         cpu_time: "${0.05 to 1.0 step 0.05}"
21         cpu_core_set: "0, 1"  # core1 and core2
22         mem_max: "${64, 128, 256, 512}" # MByte
23         io_bw: null
24       - function_id: "eu.sonata-nfv.loadb-vnf.0.1"
25         # (...)
26   - name: "service_throughput_udp"
27     extends: "service_throughput_tcp"
28     # (...)
```

Listing 1: Part of an example PED that shows the main features of our experiment description approach.

After this, multiple experiment definitions can be specified. Each experiment has a unique name, number of repetitions (line 7), time limit for a single repetition (line 8), and an identifier of the service descriptor to be used (line 9). An experiment definition also includes the configuration of *measurement point* VNFs. These configurations must include a reference to a *connection point* of the network service so that the system knows how to combine and interconnect measurement point VNFs with the profiled service (line 15).

Finally, a set of resource configurations, for each of the VNFs contained in the service, can be specified. This includes, for example, the number of vCPU cores or assigned memory. More platform-dependent resource configurations, like the available CPU time of a container, are also possible.

To simplify the specification of complex parameter studies and significantly reduce the effort required to define new experiments, we added two features that are inspired by the configuration language of the well known *Omnet++*[1]

[1]https://omnetpp.org

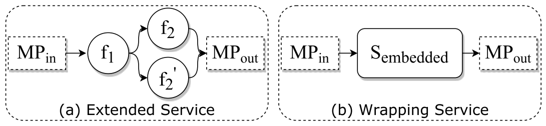

Fig. 2: Test service generation examples. Extended service descriptor (a) and embedded service S_{embedded} (b).

simulation framework. First, we support *parameter macros* which can specify value ranges (loops or lists) that should be tested for a specific parameter (line 20 and line 22). Second, we support inheritance for experiment descriptions (line 27). If a second experiment extends a first experiment, it inherits all configurations specified by the first experiment and can overwrite only the parameters that should be different.

Based on the experiment descriptors and the service specification, our profiling system will generate one service configuration for each combination of parameters that should be tested. This means that the *service_throughput_tcp* experiment in Listing 1 results in $repetitions \cdot |cpu\_time| \cdot |mem\_max| = 8 \cdot 20 \cdot 4 = 640$ different configurations, and thus profiling rounds, to be executed.

The generation of these configurations highly depends on the service descriptor technology used by the target platforms. These are often similar but in most cases not exactly the same. For example, the descriptors of SONATA [5] and OSM [7] are both based on the ETSI description model [6], but they differ in implementation details, like field names. This is why we use a plugin design to generate the descriptions which allows to implement specific generators for any description approach.

One of the main functionalities of these generators is to extend the network service descriptor of the SUT with additional measurement VNFs. This can be achieved with two different approaches shown in Fig. 2. The first approach extends the main service graph as such by appending the additional VNFs to the connection points specified in the PED (a). The second approach, in contrast, does not modify the network service descriptor as such but embeds it into another service descriptor that contains the measurement VNFs (b). As shown in the figure, the second approach has a much cleaner design and simplifies the generator implementation. However, it requires that the target platform supports hierarchical service structures.

C. Packaging Profiling Results

The profiling system collects different metrics during each profiling round and stores them in a table-based data format for further processing. The collection can either be done by using the outputs of the test tools executed inside the *measurement point* VNFs or with platform-specific monitoring systems accessible trough the *platform drivers*.

The collected results can finally be analyzed and normalized, for example, lookup tables can be created which are then bundled with the service and used by MANO algorithms, e.g., for service-specific scaling decisions. However, these analysis tasks highly depend on the planned use for the results. Thus, our system allows to plug-in arbitrary analysis scripts that are automatically executed at the end of an profiling experiment.

III. Case Study: Chain-based Profiling

We performed a series of experiments to test our profiling approach and to show that end-to-end SFC profiling can produce the same, or even better results than approaches that combine profiling results from single functions to model SFC performance behavior. This is especially interesting because our SFC-based profiling solution treats the entire SFC as a black box and does not require further information about its structure, except its incoming and outgoing connection points.

A. Scenarios and Approach

For our case study, we use a linear SFC consisting of up to three different VNFs, all acting as Layer 4 forwarding elements. The used VNFs are *Nginx*[2] (f_N) configured as TCP load balancer, the TCP relay *Socat*[3] (f_S) and *Squid Proxy*[4] (f_P) with disabled caching functionality to forward every packet. This SFC is deployed between a web-service (MP$_W$) and end users (MP$_U$). The goal is to demonstrate that, even in this simple linear setup, the fully automated end-to-end SFC profiling produces better results than solutions that combine single-function profiles. We measured the performance for each of the three isolated VNFs as well as for multiple setups of the full SFC (different order of VNFs). Table I shows a list of the used scenarios.

TABLE I: Used scenarios

isolated function f_N	$\mathrm{MP}_U \longleftrightarrow f_N \longleftrightarrow \mathrm{MP}_W$
isolated function f_S	$\mathrm{MP}_U \longleftrightarrow f_S \longleftrightarrow \mathrm{MP}_W$
isolated function f_P	$\mathrm{MP}_U \longleftrightarrow f_P \longleftrightarrow \mathrm{MP}_W$
SFC S_1	$\mathrm{MP}_U \longleftrightarrow f_N \longleftrightarrow f_S \longleftrightarrow f_P \longleftrightarrow \mathrm{MP}_W$
SFC S_2	$\mathrm{MP}_U \longleftrightarrow f_S \longleftrightarrow f_P \longleftrightarrow f_N \longleftrightarrow \mathrm{MP}_W$
SFC S_3	$\mathrm{MP}_U \longleftrightarrow f_P \longleftrightarrow f_N \longleftrightarrow f_S \longleftrightarrow \mathrm{MP}_W$

All scenarios have been deployed on our Mininet-based [8] NFV profiling platform presented in our previous work [2], [9]. This platform allows us to profile complex SFCs consisting of VNFs running inside containers on either a single physical machine or, in combination with Maxinet [10], on multiple physical machines. In both cases each VNF container was allocated to a single dedicated CPU core for isolation. To check the performance of our VNFs and SFCs under different resource configurations, we allocated different fractions of *CPU time* to each individual VNF container to emulate a large set of different resource configurations. It is important to note that the resulting performance numbers, generated by these experiments, should not be taken as absolute values but they allow us to compare our scenarios. Most experiments have been executed on a single machine with Intel(R) Core(TM) i7-960 CPU @ 3.20 GHz, 8 cores, hyper threading, and 24 GB RAM. For the distributed (Maxinet) setup, multiple of these machines, interconnected by 10G Ethernet interfaces, were used. In this setup, a dedicated machine was used for each deployed VNF container.

[2]http://nginx.org

[3]http://www.dest-unreach.org/socat/

[4]http://www.squid-cache.org

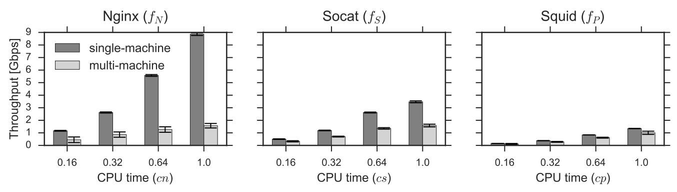

Fig. 3: Throughput of the three VNFs under different CPU time configurations executed on the single-machine and multi-machine (Maxinet) setup.

We focus on two metrics to measure VNF and SFC performance: Overall throughput and response time.

B. Throughput: Isolated Function vs. Service Chain

In the first set of experiments, we measure the total throughput between web service (MP$_W$) and end users (MP$_U$) by downloading big files with random content to simulate a vCDN service served through our VNFs and SFCs. In this setup we use *Apache2* running in MP$_W$ and *Apachebench* installed in MP$_U$ to run the downloads. Fig. 3 shows the results for single functions and different *CPU time* configurations measured in our single-machine and multi-machine platform setup. The results indicate a linear relationship between *CPU time* and throughput and show that f_N performs best. Moreover, it highlights that the absolute performance values of the single-machine setup are much better than in the multi-machine setup. The reason for this is that our profiling platform directly interconnects the VNF containers in the single-machine setup (virtual Ethernet pairs between containers). This means that there are no intermediate vSwitches that consume additional resources or tunnels between physical machines that need to be traversed. Because of this, single-machine setups are better suited to really focus on the VNF performance.

Using these results, we naturally model the expected throughput of a linear SFC with three functions and CPU configuration $C = (cn, cs, cp)$ as the minimum of the throughput measured in the single-function setups:

$$\tilde{T}_{\text{SFC}(C)} = \min\{T_{N(cn)}, T_{S(cs)}, T_{P(cp)}\}$$

We compared the SFC throughput to the results of our three SFC profiling scenarios in which the complete SFC was deployed and profiled end-to-end as shown in Fig. 4. All error bars indicate the 95% confidence intervals based on 10 repetitions. The experiments have been executed on our single-node setup and on our multi-node setup. Again, the overall performance of the sing-node setup was higher (up to 2.5 Gbps) compared to the multi-node case (1.2 Gbps). Both setups show that even though the modeled results are often near to the experimental results, there are some configurations in which the real performance of the SFC is much better than predicted by the model. The figure also indicates that SFC S_3 performs worse than the other two SFC configurations. It

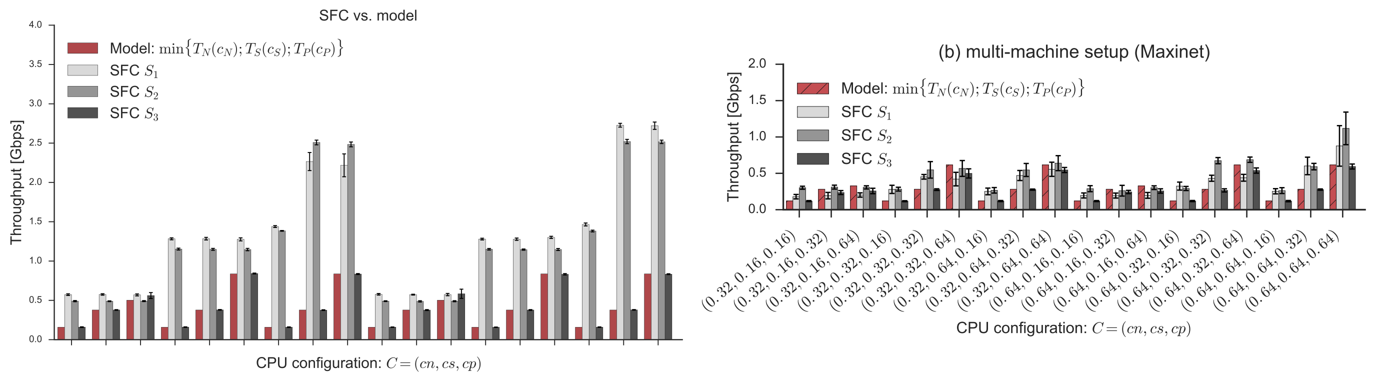

Fig. 4: Throughput of three SFC configurations under different CPU time configurations compared to the expected throughput modeled on basis of the results from our single-VNF measurements. Experiments have been execute in our single-node setup (a) and multi-node setup (b).

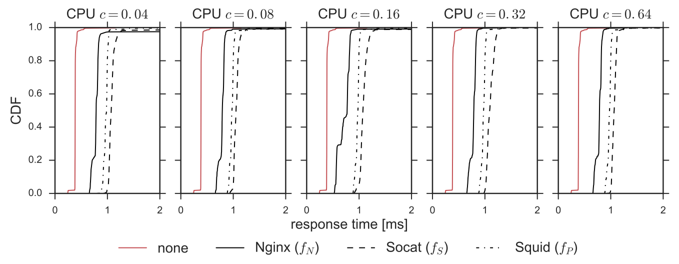

Fig. 5: Empirical response time CDFs for each VNF and a setup without VNF between MP$_U$ and MP$_W$.

especially shows that even the ordering of the same functions in the SFC has an effect on its end-to-end performance. This is something that can clearly not be captured by isolated single-function profiling approaches.

C. Response Time: Isolated Function vs. Service Chain

In the second set of experiments, we investigate the impact of different VNF and SFC configurations on the response time. This is done by performing 500 response time measurements (HTTP HEAD requests using *httping* installed in MP$_U$) in each profiling round. These experiments have only been performed in the single-machine setup because a multi-machine MaxiNet setup would have introduced a lot of bias to the response times caused by intermediate switches and network tunnels that need to be traversed. Fig. 5 shows the results for our single-function scenarios as well as for a scenario in which no VNF was deployed between MP$_U$ and MP$_W$. It shows that each VNF implementation has slightly different response times and that the allocated *CPU time* has only a small effect on the response times.

We again created a model to predict the behavior of our SFC scenarios based on single-function measurements. This model approximates the response time of an SFC by the sum of the response times measured in the individual VNF experiments. This assumption makes sense since we know that all our SFC scenarios use linear chains in which each packet has to traverse every function. Building the sum of the response times can be understood as summing up independent random variables and is done by computing the discrete convolution of the single-function measurements. Let $r_x(t)$ with $x \in \{N(cn), S(cs), P(cp)\}$ be the probability density function (PDF) derived from the results of the given single-function experiment x with a given CPU configuration. The approximated response time PDF of our SFCs with three functions is then defined as:

$$\tilde{r}_{\text{SFC}(C)}(t) = r_{N(cn)}(t) * r_{S(cs)}(t) * r_{P(cp)}(t)$$
$$= \sum_{i=0}^{t} \left(\sum_{j=0}^{i} r_{N(cn)}(j) r_{S(cs)}(i-j) \right) r_{P(cp)}(t-i)$$

This model can be used for all our linear SFC scenarios due to the commutative nature of the convolution. Fig. 6 compares these models to the measured results of our SFC experiments for changing CPU times of VNF f_P. It indicates that the SFCs do not behave like expected and show response times that are about twice as fast as the modeled response times. It can also be seen that S_1 performs worse in the 4.0% CPU time case and performs better than S_2 in the 8.0% CPU time case. This shows again that the order of the VNFs in our linear SFCs matters. It can be seen that modeling SFC performance behavior based on single-function measurements without detailed knowledge about the used VNFs and SFC structure does not lead to accurate results. As a consequence, SFC profiling solutions that support end-to-end measurements turn out to be a much better solution for fully automated environments in which black-box profiling is required.

IV. Related Work

NFV profiling is already considered by standardization bodies, like IETF [11] and ETSI [12] but the availability of real-world solutions is still limited.

A lot of work about profiling of virtualized applications has already be done by the cloud computing community, proposing a couple of solutions to profile single cloud applications [13], [14] and some solutions for complex, composed

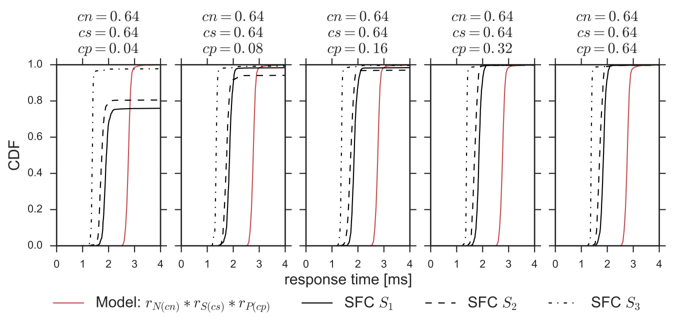

Fig. 6: Empirical response time CDFs measured for three SFC setups compared to the modeled response times derived from single-VNF measuremetns.

applications [15], [16]. Especially [14] is comparable to the resource-limited profiling approach used in our work [2]. However, these solutions cannot directly be applied to NFV scenarios due to their lack of knowledge about the required chaining functionality.

In the NFV community, in contrast, less work on profiling was done [3], [4], [17]. The first approach is called *VNF benchmarking as a service (VBaaS)* [3] and proposes a framework to profile NFV infrastructure as well as single VNFs, but lacks support to profile complex SFCs. The second approach is called *NFV-VITAL* [4] and introduces a VNF characterization framework based on an orchestrator component that allows a user to automatically profile SFCs. This approach is close to our solution but it is limited to services described by HEAT templates which offer only limited chaining support. In contrast to these approaches, the authors of [17] provide a theoretical model to estimate VNF performance. This model does not consider SFCs and requires detailed knowledge about elementary operations performed inside VNFs which is not necessarily available, e.g., for proprietary VNFs. None of the presented solutions focuses on the impact of SFC reconfiguration, e.g., re-odering, as it is done in our case study.

A highly automated DevOps environment is not explicitly considered by existing work. *NFV-VITAL* [4] provides some degree of automation but with limited flexibility compared to our experiment description and configuration generation approach. Finally, the solutions in [16] give interesting insights about reducing the number of configurations that have to be tested to profile a VNF. These ideas are compatible with our proposal an will be considered in future work to optimize the overall profiling process.

V. Conclusion

The presented profiling solution is a step toward a fully automated NFV DevOps toolchain. The results of our case study show that the structure of SFCs, and even the order of their functions, affect their end-to-end performance. This behavior cannot easily be simulated with the natural performance models used to combine single-function profiling results. Our study also demonstrates that end-to-end SFC profiling is a better solution for automated profiling setups since it removes manual modeling steps and allows black-box profiling of entire SFCs. We foresee such profiling procedures not only in the SFC development process but also as part of on-boarding procedures to give NFV platforms initial information about the behavior of SFCs prior their deployment. The prototype of the presented solution is available as part of SONATA's open-source NFV SDK [5].

Acknowledgments

We want to thank the participants of the SONATA project for the fruitful discussions. 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 (www.sonata-nfv.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] M. Peuster and H. Karl, "Understand Your Chains: Towards Performance Profile-based Network Service Management," in *5th European Workshop on Software Defined Networks (EWSDN'16)*. IEEE, 2016.
[3] R. V. Rosa, C. E. Rothenberg, and R. Szabo, "VBaaS: VNF benchmark-as-a-service," in *2015 Fourth European Workshop on Software Defined Networks*. IEEE, 2015, pp. 79–84.
[4] L. Cao, P. Sharma, S. Fahmy, and V. Saxena, "NFV-VITAL: A Framework for Characterizing the Performance of Virtual Network Functions," in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on*. IEEE, 2015, pp. 93–99.
[5] SONATA Cosortium, "SONATA Project," http://sonata-nfv.eu.
[6] ETSI GS NFV-IFA 014, "Network Function Virtualization (NFV); Management and Orchestration; Network Service Template Specification," 2016.
[7] ETSI OSM, "Open Sorce MANO," https://osm.etsi.org.
[8] 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.
[9] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid Prototyping of Production-Ready Network Services in Multi-PoP Environments," in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2016 IEEE Conference on*. IEEE, 2016.
[10] 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.
[11] A. Morton, "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure," IETF Internet-Draft https://tools.ietf.org/html/draft-ietf-bmwg-virtual-net-05, Tech. Rep.
[12] ETSI GS NFV-TST 001, "Network Functions Virtualization (NFV); Pre-deployment Testing; Report on Validation of NFV Environments and Services," 2016.
[13] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, Inc., 2008, pp. 366–387.
[14] J. Taheri, A. Y. Zomaya, and A. Kassler, "vmbbthrpred: A black-box throughput predictor for virtual machines in cloud environments," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2016, pp. 18–33.
[15] B. C. Tak, C. Tang, H. Huang, and L. Wang, "Pseudoapp: performance prediction for application migration to cloud," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 303–310.
[16] I. Giannakopoulos, D. Tsoumakos, N. Papailiou, and N. Koziris, "Panic: modeling application performance over virtualized resources," in *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 2015, pp. 213–218.

[17] M. Baldi and A. Sapio, "A network function modeling approach for performance estimation," in *Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), 2015 IEEE 1st International Forum on*. IEEE, 2015, pp. 527–533.