Dynamic Provisioning of Network Services on Heterogeneous Resources

Hadi Razzaghi Kouchaksaraei*, Ashwin Prasad Shivarpatna Venkatesh†, Amey Churi†, Marvin Illian*, Holger Karl*
Computer Network Group
Paderborn University, Paderborn, Germany
{hadi.razzaghi, marvin.illian, holger.karl}@uni-paderborn.de*
{ashwin, amayc}@campus.uni-paderborn.de†

***Abstract*—To reduce the cost and enhance the flexibility of network functions, Virtual Network Functions (VNFs) are deployed on commercially-of-the-shelf (COTS) resources instead of specialized hardware. This, however, downgrades the performance of network functions. To mitigate this issue, leveraging acceleration hardware such as GPU and FPGA has been suggested. This is because accelerators provide better parallelization and pipelining for compute- and network-intensive VNFs. However, studies show that using accelerators are not beneficial for all cases. One of the reason is that accelerators are expensive resources, possibly increasing the service cost. Consequently, the use of accelerators needs to be decided for each situation individually. To address that, in this study, we have extended Pishahang (i.e., a MANO framework) to support dynamic usage of accelerators in the NFV environment. Using the extended Pishahang, then, we analysed the management overhead due to such dynamic usage of accelerators.**

I. Introduction

Softwarising network functions in Network Function Virtualisation (NFV) degrades the performance of network functions [1]. This is because, in NFV, commercially-of-the-shelf (COTS) resources are used instead of specialized hardware to reduce cost and enhance flexibility of network functions. To mitigate this issue, leveraging acceleration hardware such as Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA) has been proposed by many studies [2] [3] [4]. Such accelerators parallelize and pipeline better than General Purpose Processors (GPPs) that are used in COTS resources, improving the performance of compute- and network-intensive Virtual Network Functions (VNFs) [5].

Although such accelerators can enhance service performance, studies [3] [6] show that they are not the best solution for all cases. One of the reasons is that accelerators are more expensive than COTS resources, possibly increasing service cost. For example, we showed [6] that for a virtual transcoder VNF, GPUs are beneficial both performance-wise and cost-wise only when high-resolution or high-bitrate videos should be processed. When the input videos to the transcoder have low resolution or low bitrate, using GPUs is beneficial neither performance-wise nor cost-wise. Consequently, the use of accelerators needs to be decided for each situation individually.

For such dynamic, situation-based provisioning of VNFs over heterogeneous resources, VNFs need to be provided in different versions. This is because, for example, a VNF implemented to run on a GPU cannot be used to run on an FPGA. Versioning can be provided on two levels: (1) on the level of individual VNFs, which we call *multi-version VNFs* and (2) on the level of network services, which we call *multi-version services*. In multi-version VNFs, each version is an implementation of a VNF for specific hardware (e.g., GPU, FPGA). In the multi-version services, each version includes a specific combination of VNFs that run on heterogeneous resources. An example of multi-version services is shown in Fig. 1.

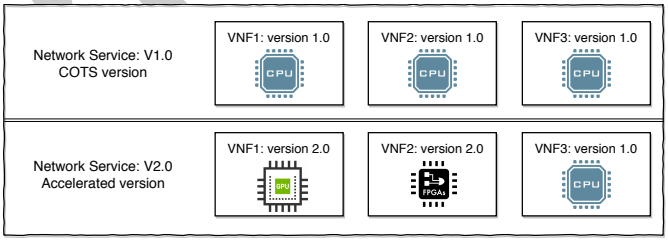

Fig. 1: An example of multi-version services

In our previous work, we evaluated multi-version services using simulation showing that the idea is promising in theory [7]. Next [6], we experimentally evaluated an example multi-version service, proving the applicability of multi-version services in real-word setups. This is the third paper in this series of work in which we focus on the management of multi-version services. To this end, we have extended a MANO framework called Pishahang [8] to support management and orchestration of multi-version services. Having such orchestration support, we measured the switching time of different versions of multi-version services to analyse the management overhead associated with these services. For this analysis, we implemented multiple versions of a virtual Transcoder (vTC), which we used as a case study for multi-version services.

The rest of the paper is structured as follows. In Section II, we review related work. The prototype implementation of orchestration support of multi-version services and the example multi-version service is described in Section III. Then, we discuss our evaluation results in Section IV, and, finally, we conclude the paper in Section V.

II. Related Work

Leveraging heterogeneous resources for NFV services has been mostly studied in the context of example virtual network functions. For example, Nobach et al. [5] studied the performance and cost of CPU-based Deep Packet Inspection (DPI) versus FPGA-based DPI. In this work, they implemented a DPI that uses COTS resources for all its tasks and another one that offloads network-intensive tasks onto FPGA resources. The results of their evaluation show a clear trade-off between performance in terms of throughput and latency and costs. Also, Araújo et al. [4] studied the use of GPUs to assist packet processing in DPIs. To this end, they implemented and evaluated a CPU-based and a GPU-assisted DPI. They evaluated these DPIs based on three metrics: processing time, resource utilization and overall latency. Their result shows that the using GPUs for DPI is not always beneficial and that there is a trade-off between resource utilization and total latency. As mentioned in Section I, we analysed the cost and performance trade-off of CPU-based vs. GPU-based vTCs [6].

Dräxler et al. [7] evaluated dynamic deployment of multi-version services using simulation. In this work, an algorithm has been implemented and used to analyse the benefit of multi-version services. The missing study here is the management of multi-version services. To manage multi-version service, a MANO framework capable of supporting multi-version services is required. Current NFV and Cloud management frameworks are limited in managing heterogeneous resources. For example, OSM[1], an NFV MANO framework, can only support VM-based VNFs running on CPU. Kubernetes,[2] which is usually used as Virtual Infrastructure Manager (VIM) in NFV MANO frameworks, supports acceleration resources like GPU as well[3] but it is limited to CN-based VNFs. Pishahang[4] (i.e., an NFV MANO) uses Kubernetes as a VIM and supports CN-based VNFs running on GPU and CPU along with VM-based VNFs. However, Pishahang is so far not capable of on-the-fly switching between different versions of a multi-version service – we rectify this shortcoming in this paper.

III. Implementation

A. Multi-version Services Orchestration

To support management and orchestration of multi-version services, we extended Pishahang [9], a multi-domain NFV MANO framework. Pishahang is based on a microservice architecture in which all functionalities that should be handled by the MANO framework are implemented using a set of loosely coupled microservices that are realised in containers. This simplifies extending Pishahang with new functionalities; this can be achieved by adding and integrating a new microservice (container) into the existing framework [10]. Another advantage of Pishahang is that it supports the provisioning of VNFs on both Virtual Machines (VMs) and Containers (CNs). Pishahang manages VMs and CNs using OpenStack[5] and Kubernetes, respectively. In Pishahang, services can be VM-based, which are provisioned only on OpenStack, CN-based, which are provisioned only on Kubernetes, or complex services, which are provisioned across OpenStack and Kubernetes. This is a valuable feature as it does not limit multi-version services to be implemented in a specific virtualisation environment. Having OpenStack and Kubernetes as VIMs allows scheduling Intel, AMD, and NVIDIA GPUs and consequently managing GPU resources. Therefore, using the Pishahang MANO framework, a wide range of GPU resources can be supported. Fig. 2 shows the high-level architecture of Pishahang containing relevant microservices and their interactions.

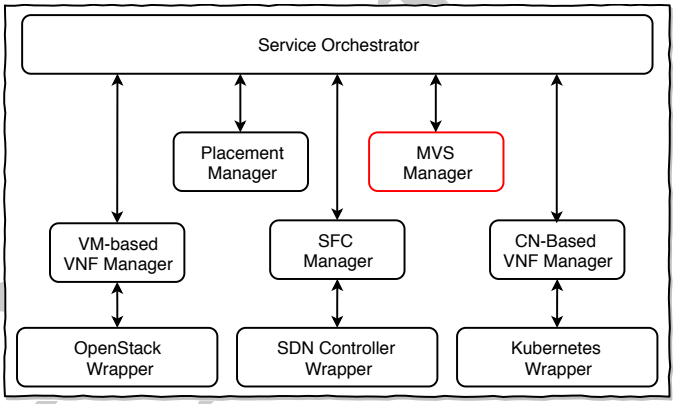

Fig. 2: The high-level architecture of Pishahang containing relevant microservices and their interactions

Although provisioning multiple versions of a service or VNFs is already supported by Pishahang, it still lacks support for multi-version services. To be exact, it cannot switch to different versions of a service/VNF on the fly, while the service/VNF is in use. To this end, we extended Pishahang with a new microservice, called MVS manager. This microservice decides when a service/VNF should be switched to which version. As it is illustrated in Fig. 3, the MVS manager consists of three main modules including message handler, version selector, and monitoring. The message handler module takes care of the communication of MVS manager modules with other Pishahang microservices such as service orchestrator. It processes the incoming messages to the MVS manager and hands them over to the respective module. The version selector module includes an algorithm taken from [7] that determines which version of the service/VNF should be used based on the service requirements, available resources, and service demand. Last but not least, the monitoring module, which is based on Netdata [6], monitors the performance of deployed VNFs along with the service demand.

Fig. 3 shows the workflow of the MVS manager and its interactions with other microservices in the Pishahang

[1]https://osm.etsi.org/, accessed Jan. 22, 2020
[2]https://kubernetes.io/, accessed Jan. 22, 2020
[3]https://github.com/NVIDIA/k8s-device-plugin, accessed Jan. 22, 2020
[4]https://github.com/CN-UPB/Pishahang, accessed Jan. 22, 2020
[5]https://www.openstack.org/, accessed Jan. 22, 2020
[6]https://www.netdata.cloud/, accessed Jan. 22, 2020

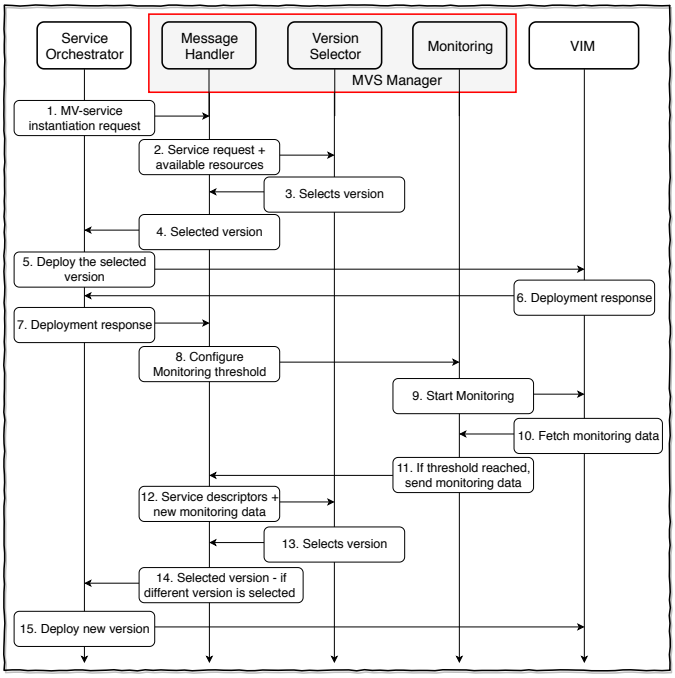

Fig. 3: The workflow of MVS manager and its interactions with other microservices in the Pishahang framework

framework. The initial multi-version service request is sent to the MVS manager by the service orchestrator microservice of Pishahang. The payload of this request includes the service descriptors (NS and VNF descriptors) and the available resources (e.g., CPU, GPU, Memory). The message is received by the message handler and then forwarded to the version selector. The version selector decides which version should be initially deployed based on the information given in the descriptors and available resources. Then, the selected version will be communicated with the service orchestrator. The service orchestrator deploys the selected version and informs the MVS manager about the result of the deployment. Once the selected version is successfully deployed, the message handler sends a request to the monitoring module containing the VNF records (e.g., name, IP address), monitoring metrics (e.g., data rate, CPU), and the metrics threshold. Then, the monitoring module starts fetching monitoring data and sends an alert to the message handler whenever a threshold is reached. The message handler, then, forwards the new monitoring data to the version selector and notifies the service orchestrator if the version selector decides to switch the version. If the decision is to switch the service version, the service orchestrator deploys the new version, redirects the packets to the new version, and removes the old version.

B. Multi-version Services Descriptors

The service descriptors model of Pishahang has also been extended to support the requirements of multi-version services. As it is shown in Fig. 4, there are three main descriptors in the multi-version service descriptor model: (1) MVS descriptor, (2) VNF descriptors and (3) Version descriptor. The MVS descriptor includes the high-level information of multi-version services such as the constituent VNFs, constituent service versions, and forwarding graph. Each service version in the MVS is represented by a forwarding graph. On lower layer, VNF descriptors are used to describe the high-level requirements of each VNF such as constituent VNF versions, their VIM type (OpenStack or Kubernetes), and version switching information (e.g, default version, threshold alerts). Each version is then described using a version descriptor which includes the VNF requirements such as Version types (COTS, accelerated), deployment units, resources, and images..

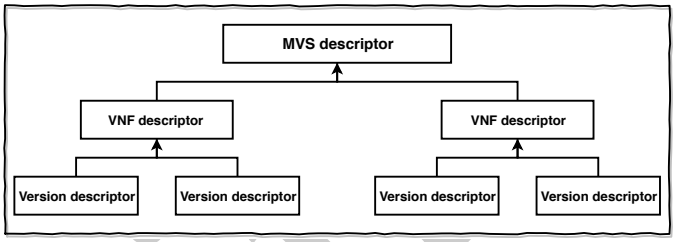

Fig. 4: The multi-version services descriptor model

C. Multi-version Virtual Transcoder

To validate the functionality of the MVS manager and analyse the version switching time, we also implemented an example multi-version service. This service consists of three versions of a virtual Transcoder (vTC). Transcoder has been chosen as it consists of compute-intensive processes that can be offloaded to acceleration hardware such as GPU. The three versions of the vTC consist of two commercial-of-the-shelf (COTS) versions namely a VM-based vTC and a CN-based vTC. These two versions only use CPU for all processes. The third version is an accelerated vTC that runs on a CN and offloads the compute-intensive processes to the GPU.

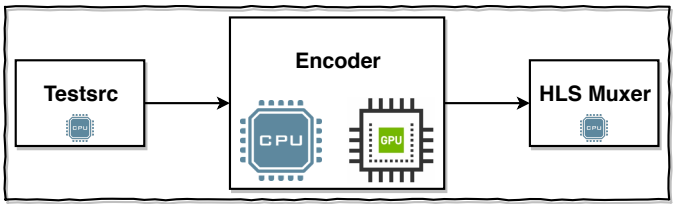

Fig. 5: The high-level architecture of virtual transcoder

Fig. 5 depicts the high-level architecture of the vTC. It is composed of three FFmpeg [7] components. FFmpeg is an open-source solution for recording, converting and streaming audio and video. The first component on the left is Testsrc, which generates test pattern video frames to mimic a live streaming source. Testsrc runs on CPU on all vTC versions and provides raw videos with different bitrates and resolutions for the second component which is the encoder. The encoder component converts the video format to H.264. This is the main component of the vTC. For the COTS vTC, no matter

[7]https://www.ffmpeg.org/, accessed Jan. 22, 2020

VM or CN, the encoder runs entirely on CPU, however, for the accelerated vTC, the encoder offloads its compute-intensive processes to GPU. The final component is the HLS Muxer that includes the output of encoder into the MPEG transport stream to be transmitted over the network. Like Testsrc, HLS Muxer also runs only on CPU in all vTC versions.

IV. Evaluation

We experimentally evaluated the switching time of different versions of multi-version services. Fig. 6 shows the set-up of our testbed. In this set-up, the extended version of Pishahang was running on a virtual machine equipped with 16 CPU cores of Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz and 32 GB of RAM. A single-node-cluster Kubernetes was used for CNFs. This cluster includes a workstation equipped with 8 cores of Intel(R) Xeon(R) W-2123 CPU @ 3.60GHz, 16 GB of RAM, and a GeForce RTX 2080 GPU. The OpenStack cluster also includes a single node, which is a workstation equipped with 20 cores of Intel(R) Xeon(R) W-2155 CPU @ 3.30GHz and 64 GB of RAM. Moreover, a Ryu controller [8] along with a Zodiac FX OpenFlow switch was used to redirect the traffic to the intended service version on the fly.

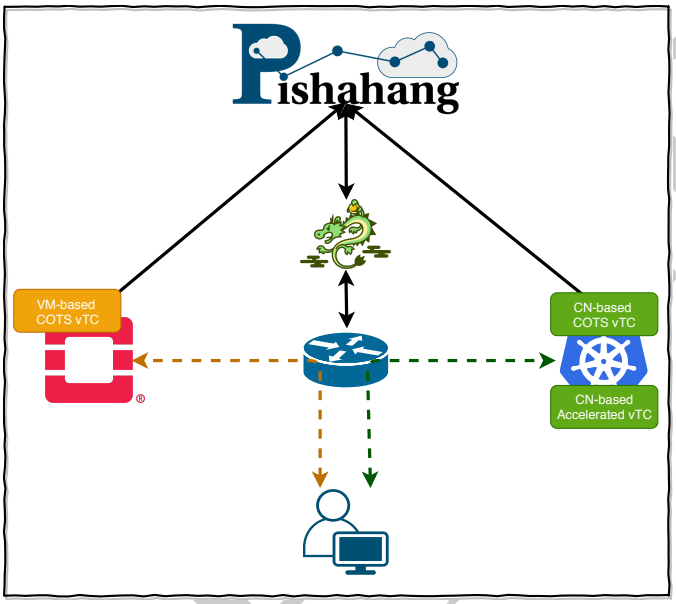

Fig. 6: The test-bed set-up used for the evaluation

As mentioned in Section III, we implemented a multi-version vTC for evaluation which consists of (1) CN-based COTS (CNC) vTC, (2) VM-based COTS (VMC) vTC, and (3) CN-based accelerated (CNA) vTC. We also used a virtual machine as a vTC user. This user sends requests to the vTC for different video resolutions and bitrates, which are the main metrics used by version selector in MVS manager to switch between different versions. These two metrics have been chosen based on the results in [6]: the accelerator-based vTC not only processes high resolution/bitrate videos faster than COTS-based vTC but also cheaper as it requires fewer resources. On the other hand, the CPU-based vTC is not only cheaper to process the resolution/bitrate videos but, in this case, also faster compared to an accelerator-based vTC. In this evaluation, we measured the time required to switch from one version to another. Having three versions of vTC, we had 6 cases of switching time, each of which was measured a hundred times. The switching time is the time between initiating the switching request in the version selector and receiving the switching response by the message handler in the MVS manager. This time includes the deployment of the new version, starting monitoring the new version, and termination of the old version.

Fig. 7 shows the cumulative distribution function (CDF) of the time to switch from VMC to CNA, CNC to CNA, CNA to CNC, and VMC to CNC versions of vTC. These results show that the switching to CN-based VNFs takes mostly between 3.2 and 4 seconds. The switching time from VM-based to CN-based vTCs is longer than CN-based to CN-based vTCs. This is because the termination of VM-based vTCs takes longer than CN-based vTC.

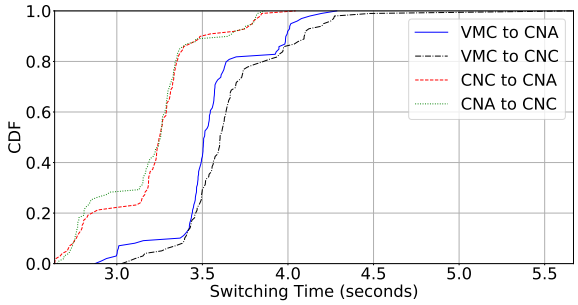

Fig. 7: The time required to switch between CNA to CNA, CNA to CNC, and VMC to CNA versions of the virtual transcoder

Fig. 8 shows the CDF of switching time from CNA to VMC and CNC to VMC versions of vTC. These results show that the switching to VM-based VNFs takes mostly between 64 to 100 seconds, which is much longer than switching to CN-based vTCs. Most of the VM switching time is consumed for the deployment of the VMs in OpenStack which is significantly higher compared to the deployment of CNs in Kubernetes. From Fig. 7 and 8, we observe that the deployment time of CNAs is slightly shorter than CNCs. Breaking down the results, we noticed that the difference comes from the time required by Kubernetes to deploy CNAs and CNCs. Surprisingly, Kubernetes handles the deployment of CNAs faster than CNCs.

Another action involved in version switching is the redirection of the traffic to the new version. To this end, we have measured the time required by Pishahang to generate a new forwarding graph and redirect vTC incoming traffic to the new version. Fig. 9 illustrates the results of this evaluation. From these results, we observe that the packet redirection time also

[8]https://osrg.github.io/ryu/, accessed Jan. 22, 2020

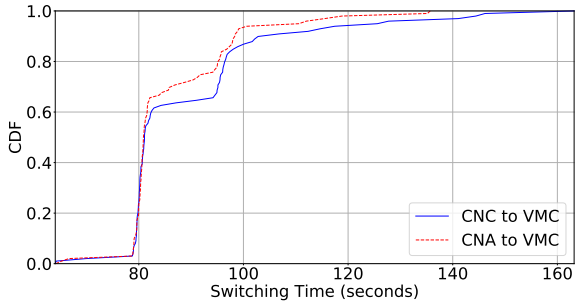

Fig. 8: The time required to switch between CNA to VMC and CNC to VMC versions of the virtual transcoder

plays an important role in the case of switching to CN-based vTCs, which should not be overlooked, as it increases the switching time by up to 0.49%.

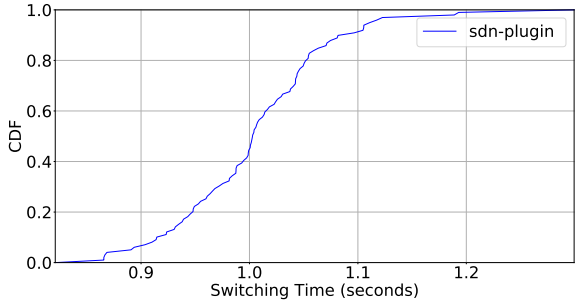

Fig. 9: The time required by the SFC manager to generate a new forwarding graph for redirecting the traffic to the new version

V. Conclusion and Future Work

In this paper, we extended a MANO framework to support the management and orchestration of multi-version services. Multi-version services are a type of NFV services that allow dynamic usage of accelerators according to the service requirements and demands. With the extension provided to the MANO framework, we analysed the management overhead due to such dynamic usage of accelerators.

From the results, we observed that the switching time to CN-based versions is significantly less than the switching time to VM-based versions. Thus, the virtualisation environment type can have a great impact on the management overhead of multi-version services. Comparing the switching time of accelerated versions to COTS versions, we observe that Kubernetes does not require a longer time to schedule GPU resources and that, in the case of GPU as an accelerator, there is no extra management overhead. In the case of switching to VM-based versions, the switching time is significant, therefore, other migration solutions such as warm or live migration [11] should be considered.

As the results show, although the switching time varies between different versions, it is not negligible in any of the evaluated cases. This affects the benefit of dynamic usage of heterogeneous resources by increasing a service's total latency. To reduce this effect, we need to decrease the number of times a service is switched between different versions. In our future work, we will address this issue. Predicting the service load using machine learning techniques is one of the solutions that we consider to optimise the number of times that a service has to be switched.

Acknowledgment

This work has been partially supported by the 5G-PICTURE project, funded by the European Commission under Grant number 762057 through the Horizon 2020 and 5G-PPP programs, the German Research Foundation in the Collaborative Research Centre On-The-Fly Computing (SFB 901), and the Industrial Automation Plattform für Big Data project of the Leading-Edge Cluster it's owl (Intelligente Technische Systeme OstWestfalenLippe).

References

[1] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon ec2 data center," in *2010 Proceedings IEEE INFOCOM*, March 2010, pp. 1–9.

[2] N. Ghrada, M. F. Zhani, and Y. Elkhatib, "Price and performance of cloud-hosted virtual network functions: Analysis and future challenges," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018, pp. 482–487.

[3] L. Nobach and D. Hausheer, "Open, elastic provisioning of hardware acceleration in nfv environments," in *2015 International Conference and Workshops on Networked Systems (NetSys)*, March 2015, pp. 1–5.

[4] I. M. Araújo, C. Natalino, Á. L. Santana, and D. L. Cardoso, "Accelerating vnf-based deep packet inspection with the use of gpus," in *2018 20th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2018, pp. 1–4.

[5] L. Nobach, B. Rudolph, and D. Hausheer, "Benefits of conditional fpga provisioning for virtualized network functions," in *2017 International Conference on Networked Systems (NetSys)*, March 2017, pp. 1–6.

[6] H. R. Kouchaksaraei and H. Karl, "Quantitative analysis of dynamically provisioned heterogeneous network services," in *Proceedings of the 15th International Conference on Network and Service Management*, ser. CNSM '19. IFIP, 2019.

[7] S. Dräxler and H. Karl, "SPRING: scaling, placement, and routing of heterogeneous services with flexible structures," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019.

[8] H. R. Kouchaksaraei and H. Karl, "Service function chaining across openstack and kubernetes domains," in *Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems*, ser. DEBS '19. New York, NY, USA: ACM, 2019, pp. 240–243. [Online]. Available: http://doi.acm.org/10.1145/3328905.3332505

[9] H. R. Kouchaksaraei, T. Dierich, and H. Karl, "Pishahang: Joint orchestration of network function chains and distributed cloud applications," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018, pp. 344–346.

[10] S. Dräxler, H. Karl, M. Peuster, H. R. Kouchaksaraei, M. Bredel, J. Lessmann, T. Soenen, W. Tavernier, S. Mendel-Brin, and G. Xilouris, "Sonata: Service programming and orchestration for virtualized software networks," in *2017 IEEE International Conference on Communications Workshops (ICC Workshops)*, May 2017, pp. 973–978.

[11] S. Thamarai Selvi, C. Valliyammai, G. P. Sindhu, and S. Sameer Basha, "Dynamic resource management in cloud," in *2014 Sixth International Conference on Advanced Computing (ICoAC)*, Dec 2014, pp. 287–291.