Demonstrating FOP4: A Flexible Platform to Prototype NFV Offloading Scenarios

Daniele Moro*, Manuel Peuster†, Holger Karl†, and Antonio Capone*
*Politecnico di Milano: {daniele.moro, antonio.capone}@polimi.it
†Paderborn University: {manuel.peuster, holger.karl}@upb.de

***Abstract*—Emulation platforms supporting Virtual Network Functions (VNFs) allow developers to rapidly prototype network services. None of the available platforms, however, supports experimenting with programmable data planes to enable VNF offloading. In this demonstration, we show *FOP4*, a flexible platform that provides support for Docker-based VNFs, and VNF offloading, by means of P4-enabled switches. The platform provides interfaces to program the P4 devices and to deploy network functions. We demonstrate *FOP4* with two complex example scenarios, demonstrating how developers can exploit data plane programmability to implement network functions.**

I. Introduction

Network Function Virtualization (NFV) aims to improve the flexibility and manageability of networks. It transforms legacy (hardware-based) network functions into software components that can be executed on low-cost off-the-shelf servers using virtualization technologies. Those Virtualized Network Functions (VNFs) can be developed much faster than hardware appliances and be rapidly prototyped in emulation-based platforms, like Containernet [1]. However, the packet processing performance of virtual machines (VMs) or containers is often limited and an interesting option is that of accelerating it by offloading packet processing tasks to the data plane. Programmable data planes [2], [3] allow to exploit network devices to implement tasks different from the mere packet forwarding. VNFs can be partially or even fully *offloaded* directly to the network data plane. P4 [4] is the reference programming language for network data planes and several use cases of VNF offloading, exploiting P4, have been proposed [5], [6]. However, the availability of prototyping platforms to ease the realization of such scenarios is still limited and existing prototyping platform lack support for offloading technologies [1], [7], [8].

In this demonstration, we show *FOP4* [9], a prototyping platform that supports the execution of hybrid P4 and container-based VNFs. The proposed platform provides support for P4 software switches and containerized VNFs (Docker) combined with Mininet functionalities, such as simulated link delays or loss. With this platform, VNF and service developers can implement and locally test hybrid, chained network services consisting of both container-only VNFs as well as VNFs that offload tasks to the P4-programmable data plane.

During our demonstration, two scenarios of hybrid/accelerated VNF deployments are shown. The first one is the *hybrid Intrusion Detection System (IDS)* scenario that exploits the programmable data plane to implement the load balancing function, while the IDS is left as a Docker container. The second one is an *extended In-band Network Telemetry (INT)* scenario [10] in which we show how we can use both P4 switches as well as container-based VNFs to collect in-network statistics.

II. The FOP4 Prototyping Platform

The objective of our platform is to provide a rapid prototyping environment to implement VNF offloading use cases without needing to deploy a real system. We decided to start from a previous work of some of the authors, Containernet [8] (a fork of the *Mininet* emulation platform), because it already supports (Docker) container-based VNFs. FOP4 extends this platform to support the execution of BMv2 (Behavioral Model version 2) software switches together with Open vSwitch (OVS) or other user-space switches. BMv2[1] is the reference P4 software switch developed by the *P4.org* consortium.

FOP4 integrates the BMv2 P4 switch exploiting two different interfaces: A P4Runtime-based[2] and a Thrift-based interface. P4Runtime allows controlling the P4 switch using the ONOS controller while the Thrift-based interface allows to program and modify the switch configuration with a Command Line Interface (CLI). Both interface options are supported by FOP4 through a simple Python API which allows to add P4 switches to the emulated scenarios using a single line of Python code. Examples for this are, together with the implementation of the two demonstration scenarios, publicly available on GitHub[3].

To be fully compliant with Containernet we also support adding new interfaces at runtime to the P4 switch through Containernet's Python APIs. In this way VNFs can be added even after the emulated network has been started (e.g., to scale up a service). To simplify the development of complex VNF offloading use cases, we also extend the *MiniEdit* GUI adding support for the P4 BMv2 switch as we will show in the demo.

III. Demonstration

Our demo shows how developers can use FOP4 to quickly prototype and run container-based VNFs, offloading some tasks to the P4 data plane. The entire platform is installed on a single computer on which the live demo will be executed.

[1]BMv2 repository: https://github.com/p4lang/behavioral-model
[2]P4Runtime repository: https://github.com/p4lang/PI
[3]FOP4 repository: https://github.com/ANTLab-polimi/FOP4

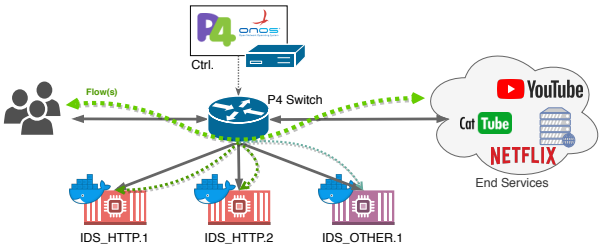

Fig. 1: Example architecture of scenario 1: Hybrid IDS implemented as a combination of P4 switches and differently configured IDS containers

A. Demonstration Scenarios

The demo uses two different real-world scenarios. The first scenario is called *hybrid IDS* and is shown in Fig. 1. It demonstrates how an IDS can use offloading technologies and be partially implemented in the data plane using P4 while keeping some parts of its functionality as normal software components executed in containers. The main idea of this scenario is to use the data plane as a smart load balancing function that distributes traffic to a couple of container-based IDS VNFs that do the actual rule matching. In contrast to naive load balancing approaches, which equally distribute the traffic among similar IDS instances, we use the data plane to pre-classify the traffic and send flows to a particular IDS instance. This allows us to optimize the rules in each of the IDS instances for a certain kind of traffic. For example, some IDS instance might be optimized to analyze HTTP traffic, others to analyze incoming SSH connections. In this way, not every IDS instances need to match the complete rule set and the overall system load can be reduced.

The second scenario shows an *extended In-band Network Telemetry* use case. It implements in-band network telemetry scenario in which network telemetry information is not only captured on data plane elements, i.e. the P4 switches, but also from container-based legacy VNF implementations. This allows to inject telemetry statistics coming from VNFs directly into the stream of packets flowing in the network. The statistics can be CPU and memory usage, timestamps etc. and they can be leveraged, for example, to decide when to scale up or down a Network Function. In this use case, P4 is used to implement INT into the P4 switches, while the VNFs are implemented as docker containers running a Python script to inject statistics into the packets. This use case is presented using a deployment of four P4 switches and two container-based VNFs.

B. Demonstration Steps

The following steps will be performed during the live demo:

Step 1 (development): Show and describe the used VNFs as well as the used P4 programs.

Step 2 (composition): Use Containernet's GUI and Python APIs to compose the scenarios, e.g., shown in Fig. 1.

Step 3 (instantiation): Instantiate the scenarios on top of FOP4, including the automatic deployment and configuration of container-based VNFs and P4 programs.

Step 4 (interaction): Use a web browser to access a video streaming service through the deployed network service.

Step 5 (stressing): Use traffic generators to stress the scenario and to validate them.

IV. Conclusion

We demonstrated FOP4 a new prototyping platform that allows VNF developers to implement offloading use cases exploiting programmable data planes via P4 enabled devices. We demonstrate two examples of offloading use cases which can be implemented in our framework exploiting both programmable network devices and Docker-based VNFs. FOP4 can be used as a prototyping platform by NFV researchers and VNF developers that want to implement network functions that exploit the programmability of the data plane without deploying a costly physical testbed in the initial phase of experimentation. FOP4 is released as open source software and available on GitHub.

Acknowledgments

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

References

[1] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid Prototyping of Production-ready Network Services in Multi-PoP Environments," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 148–153.

[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 99–110, 2013.

[3] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda *et al.*, "Flowblaze: Stateful packet processing in hardware," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19). USENIX Association*, 2019.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[5] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 15–28.

[6] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 121–136.

[7] 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.

[8] M. Peuster, J. Kampmeyer, and H. Karl, "Containernet 2.0: A rapid prototyping platform for hybrid service function chains," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, June 2018, pp. 335–337.

[9] D. Moro, M. Peuster, H. Karl, and A. Capone, "FOP4: Function Offloading Prototyping in Heterogeneous and Programmable Network Scenarios," in *2019 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2019.

[10] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, 2015.