Towards integrating hardware Data Plane acceleration in Network Functions Virtualization

David Franco¹, Asier Atutxa¹, Jorge Sasiain¹, Eder Ollora², Marivi Higuero¹, Jasone Astorga¹, Eduardo Jacob¹

¹Department of Communications Engineering, University of the Basque Country (UPV/EHU). 48013 Bilbao, Spain. {david.franco, asier.atutxa, jorge.sasiain, marivi.higuero, jasone.astorga, eduardo.jacob}@ehu.eus
²DTU Fotonik, Technical University of Denmark. Kongens Lyngby, Denmark. eoza@fotonik.dtu.dk

Abstract—This paper proposes a framework for integrating data plane (DP) acceleration within the Network Functions Virtualization (NFV) architecture. Data plane programming (DPP) proves to be beneficial for NFV environments, as it provides full packet forwarding flexibility through the use of self-designed algorithms. Additionally, DPP provides high-performance networking, as the DP can be configured to execute specific functions on dedicated hardware. We present an integration of the DP acceleration within the ETSI NFV architecture that leverages custom DP functions implemented in hardware switches using P4 language. Besides, OpenStack and Kubernetes are used as Virtualized Infrastructure Managers (VIMs) and Open Source MANO (OSM) as the Management and Orchestration (MANO) element.

Keywords—P4, NFV, data plane acceleration

I. INTRODUCTION

In the last years, Network Functions Virtualization (NFV) and Software-Defined Networking (SDN) have changed the framework for the deployment of services. On the one hand, NFV allows Network Functions (NFs) to be deployed as Virtual Network Functions (VNFs) over a commercial off-the-shelf hardware infrastructure. On the other hand, SDN solves the problem of having a vendor-specific control plane (CP) in network devices, allowing the definition of custom CPs designed by the network operator. However, full packet forwarding flexibility is given by data plane programming (DPP), which can forward packets following self-designed algorithms and perform custom actions to packets with user-defined formats. Programming Protocol-Independent Packet Processors (P4) is a DPP language that provides a high abstraction level to define packet processing pipelines. DPP also provides high performance networking, as the data plane of network devices can be configured to execute specific functions on dedicated hardware. The addition of DPP to NFV through hardware network devices enhances the performance in the communication among VNFs thanks to line-rate packet processing capability and the ability to offload certain compute-intensive functions.

This paper proposes an integration of the DP acceleration within the standard ETSI NFV architecture, using OpenStack and Kubernetes to manage the NFV Infrastructure (NFVI), and Open Source MANO (OSM) as the top-level NFV orchestrator. The framework integrates the lifecycle of P4-enabled hardware switches in the NFV architecture to enable the automatic offloading of NFs to the DP.

II. RELATED WORK

Different works have been presented in the literature regarding the data plane (DP) function offloading in NFV. For instance, [1], [2] introduce two frameworks to offload VNF processing to P4-based Network Interface Cards (NICs) and software switches respectively. The former decomposes VNFs into small embedded NFs that are offloaded to the DP, and the latter runs NFs as P4 programs on Docker containers that implement P4 targets.

Authors in [3] propose an ETSI NFV-based architecture that allows the instantiation of P4-based NFs. They explain the necessity of reconfiguring the DP when multiple users are sharing the same physical P4 target and they test their architecture over P4 software switches. In this sense, [4], [5], [6] provide different approaches to achieve DP modularity and allow a transparent reconfiguration of the DP. The majority of them require modifications of the P4 compiler, or even in the architecture of the P4 target. For instance, [5], [6] present abstraction layers and mechanisms to create modular P4 programs that can then be merged.

Authors in [7], [8] describe how to integrate P4 within OpenStack, by modifying the Neutron module to allow the offloading of some VNF functions to the DP of P4 software switches. They classify their VNF offloading according to the acceleration techniques defined by the ETSI NFV [9]. However, none of these approaches considers the use of hardware-based targets.

This paper focuses on the advantages of integrating P4-enabled hardware switches in NFV to automate the offloading and acceleration of custom NFs to the DP.

This work is licensed under a Creative Commons 4.0 International License (CC BY-NC-ND 4.0)
III. PROPOSED FRAMEWORK FOR DP ACCELERATION IN NFV

This section describes the proposed integration of the DP acceleration within the ETSI NFV architecture. The proposed framework allows the users to create network services (NSs) that leverage custom DP functions implemented in P4 language. For this purpose, physical P4-enabled switches are integrated into the NFVI.

A. Functional description

In this subsection we present an overview of the proposed technologies and approaches to enable the deployment of NSs combining traditional NFs —VNFs and Container Network Functions (CNFs)— with our novel proposed concept of P4 Network Functions (P4NFs). We propose the deployment of such P4NFs over P4 switches.

As part of the NFV architecture, OpenStack and Kubernetes are used as Virtualized Infrastructure Managers (VIMs) and Open Source MANO (OSM) is used as the Management and Orchestration (MANO) element. The NFVI is composed of several compute nodes that form the OpenStack cloud and the Kubernetes cluster, as well as of the switching infrastructure to interconnect those nodes, which includes both regular and P4 switches. OSM can deploy VNFs and CNFs on top of OpenStack and Kubernetes respectively, whereas P4NFs are employed to configure the P4 switches. These switches are registered in OSM as Physical Deployment Units. Connectivity between VNFs and CNFs can be provided through the regular top of the rack switches and/or through the P4 switches. To make this possible, the compute nodes have network interfaces connected to both types of switches, enabling the VNFs/CNFs to connect to any network. The VNF/CNF interfaces towards the P4 switches are SR-IOV Virtual Functions (VFs).

The deployment of these P4NFs leverages OSM’s support of Juju charms, which encapsulate a set of Python scripts that can be executed towards the target of the P4NF, i.e. a P4 switch. Apart from the P4NF, the user encapsulates the desired VNFs/CNFs into an NS. The P4NF designer provides the network administrator with the P4NF metadata file plus the VNFs/CNFs, so that the latter builds the complete package and instantiates it. Furthermore, Juju charms can be used to implement day-2 operations that allow the P4NF designer to directly modify the CP rules at any time once the NS is created.

An example of the onboarding, instantiation, and lifecycle management processes of a composite NS is illustrated in Fig. 2.

B. Low-level design

This subsection presents a low-level description of the proposed deployment methodology for DP acceleration...
Towards integrating hardware Data Plane acceleration in Network Functions Virtualization

in NFV. Specifically, a workflow for the lifecycle management of the envisioned NSs is analyzed, detailing the procedures involved in each task.

The framework hereby presented provides three main features that enable the description, instantiation, and management of the P4-enabled NSs. It provides (a) multi-tenancy, so the connectivity between NFs is restricted to NSs connected to the same OpenStack VLAN network, and, thus, to NSs belonging to the same tenant. It also offers (b) DP composition to describe P4 functions at NS level and then combine them to create a single P4 program that comprises the P4 functions of all running NSs—even those from different tenants. In this way, the DP must be recomposed each time an NS is instantiated or terminated. The DP composing is implemented according to [5], [6]. Lastly, it supports (c) DP Slicing to implement different and isolated packet processing pipelines for each NS that is configured when composing the DP.

A standard NS is formed by a combination of VNFs/CNFs while a P4-based NS includes a P4NF, which describes the configuration of the custom DP dedicated to that NS. Therefore, a new workflow must be defined to accommodate the management of the lifecycle of these P4NFs. In this case, Juju is in charge of the lifecycle management of the P4NFs, being responsible for their instantiation and termination. Thus, a P4NF is a PNF that points to the physical P4 switch and implements the required Juju charms for its configuration.

To achieve these features, maintaining different types of state information is crucial. This leads to the following requirements:

- Minimal Layer 2 (L2) forwarding requires the P4 switch to know the MAC addresses connected to each physical port. Traffic from different VNFs/CNFs could come into the same switch port as they could be hosted by the same compute node. A static match-action table (MAT), forward_l2, is used for this purpose. Each compute node SR-IOV VF is preconfigured with a static MAC address, and the entries with each VNF/CNF MAC address to switch port mapping are preloaded into the aforementioned table.
- Multi-tenancy and DP slicing support require maintaining a record in an external database of which DP functions are in use by each NS. This information is stored in the \textit{ns_functions} table, which is consulted whenever the DP needs to be recomposed.
- Upon the instantiation or termination of a P4NF that requires recompiling the DP, all CP information is lost as the DP tables are destroyed. For this reason, an image of the current CP state must be kept up to date in an external database. The \textit{cp_rules} table is introduced.
- The P4 code used by any P4NF needs to be stored in a public repository. This includes the code that would be required by a new P4NF, in order to pull said code during the instantiation of the P4NF. This repository, named \textit{P4 function repository}, allows any user to upload their custom DP functions so they are beforehand checked.

When an NS is instantiated, VNFs and CNFs are configured following the standard lifecycle operations defined in OSM. For the P4NF, the code defined in the Juju charms performs the following lifecycle operations to instantiate the P4NF:

1) **Check required P4 functions**: The P4NF metadata file, which is part of the Juju charms, includes a list of the P4 functions that the DP must implement to support the P4NF.
2) **Update ns_functions table**: As the P4 switch cannot run more than one single DP P4 program at once, this database table is required in order to obtain the complete list of the running P4 functions and recreate the program in the following step. Likewise, the table must be updated to reflect the DP pipeline state after the introduction of the new P4NF.
3) **Compose P4 program**: This step consists in combining different P4 functions to create a single P4 program that configures the DP pipeline. Even if the new NS only requires P4 functions that already exist in the DP, a new DP needs to be composed, as the tables must be unique to restrict access to each DP table from the CP. This process is challenging and requires specific P4 coding rules to achieve composable P4 functions. The literature provides different approaches to accomplish this task [5], [6]. To use custom P4 functions, they must be uploaded to the \textit{P4 function repository}, which assures their "composability".
4) **Compile P4 program**: The Juju charms establish an active connection to the P4 switch to load, compile, and run the P4 program. During this step, the P4 switch is not operational until the new program is running.
5) **Configure switch ports**: in case the DP was recomplied, it is necessary to bring up all the ports that are in use by the switch. This is a fixed task, as all the ports with a physical connection to a compute node need to be enabled.
6) **Update CP**: Finally, the Juju charms perform a basic CP configuration to provide L2 connectivity between the VNFs/CNFs specified in the NS. This is achieved by having the VNFs/CNFs send their MAC addresses to the P4NF through Juju. These parameters are then used as input parameters for the CP to configure the \textit{forward_l2} MAT. On the other hand, the \textit{cp_rules} database table needs to be consulted to restore the most recent state of the CP.

Additionally, CP operations that involve filling owned DP tables are allowed through the NF day-2 lifecycle operations supported by OSM and Juju. This includes both setting up the initial CP rules after the P4NF instantiation, and inserting new rules on demand. The exception to this is the \textit{forward_l2} MAT, which, despite containing entries for all users, is managed by the network administrator only. Each day-2 operation is registered in the \textit{cp_rules} database to keep it up to date.

This work is licensed under a [Creative Commons 4.0 International License](CC BY-NC-ND 4.0)
The termination of the NS must be considered as part of the lifecycle management. Therefore, the code defined in the charms performs the following operations to remove the P4NF:

1) **Update ns_functions table:** All active P4 functions are retrieved from the `ns_functions` table. Those functions that are no longer needed after the termination of the P4NF are removed from the table.

2) **Compose P4 program:** Similarly to the NS instantiation, the P4 program is reconfigured. The no longer needed P4 functions after the termination of the P4NF are excluded to build the new P4 program. This includes the removal of unused MATs. This process is done to achieve the simplest possible DP pipeline.

3) **Compile P4 program:** The same as in the NS instantiation workflow is applied.

4) **Configure switch ports:** The same as in the NS instantiation workflow is applied.

5) **Update CP:** The entries in the `forward_l2` MAT for the terminated P4NF are discarded. In addition, CP changes triggered through day-2 operations need to be reverted during the reconstruction of the CP. This is done by looking up the `cp_rules` table.

IV. CONCLUSIONS AND FUTURE WORK

DP acceleration in NFV is a promising idea, as the dynamic reconfiguration of the DP in an NFV-based environment grants network service providers with powerful possibilities. However, there are still some challenges that must be overcome in order to optimize the system for production environments. We hereby highlight and discuss the following three challenges.

**The first challenge** is sharing P4 MATs among tenants. This might seem a good option, as it results in better usage of resources and optimization of the P4 program. However, if a table is shared between two tenants, one could change the entries that correspond to the other tenant and illegitimately modify its behavior. In fact, unauthorized changes in the communication among VNFs could be a potential attack vector inside the system. The proposed solution has been to duplicate the tables for each tenant or NS, so that each of them only has access to its own tables and cannot modify other equivalent tables.

**The second challenge** is concerned with the duplication of parser code in P4 programs. The P4 program in the switch implements P4 functions in a modular way, only composing and compiling P4 functions required by instantiated NSs. However, multiple NSs may have partially concurrent needs, or, in other words, share one P4 function but require others that are different among them. This leads to different parsing requirements of those NSs. To put this in context, while two NSs might use the same static L2 forwarding table, one NS may just need to parse the Ethernet header, and the other one may need to parse even the application layer—in order to execute other actions in a later phase. Therefore, a solution to this could be the duplication of parsers—one for each NS—or having a “super-parser” that is able to fulfill any parsing requirement of all instantiated NSs, using only the necessary parts for each case.

**The third challenge** is related to the service downtime that arises due to the recompilation of the P4 program during the instantiation and deletion periods of NSs. Each time one of these processes is executed, the DP must be recomposed and recompiled, interrupting the service in the compilation part. This means that all deployed NSs lose connectivity in that period, which entails an availability issue. Approaches similar to [5], [7] could be studied to address this issue, in which the switch enters in a “slow-mode” state while the new DP is being recompiled, but does not stop serving with the current DP. Even though experiencing service downtime might not be reasonable for critical services, it could be for experimental environments that are not significantly affected by it.

Therefore, our research shows that there are several methods under investigation that can provide solutions to the aforementioned challenges, and advance towards a fully operational system. There is still more work to do, mainly focusing on the challenges analyzed in this section, but we think that the overall approach is promising and may change the current service deployment system in NFV environments.

ACKNOWLEDGEMENTS

This work was supported in part by the Spanish Ministry of Science and Innovation through the national project (PID2019-108713RB-C54) titled “Towards zero touch nEvolution and services for beyond 5G” (TRUE-5G), and in part by the “Smart Factories of the Future” (5G-Factories) (COLAB19/06) project.

REFERENCES

[1] Mafioletti, Diego Rossi, et al. “PlaFFE: A Place-as-you-go In-network Framework for Flexible Embedding of VNFs.” ICC 2020-2020 IEEE International Conference on Communications (ICC). IEEE, 2020.

[2] Bononfio, Gaetano, et al. “Kathari: A container-based framework for implementing network function virtualization and software defined networks.” NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, 2018.

[3] He, Mu, et al. “P4nfv: An nfv architecture with flexible data plane reconfiguration.” 2018 14th International Conference on Network and Service Management (CNSM). IEEE, 2018.

[4] Stoyanov, Radostin, and Noa Zilberman. “MTPSA: Multi-Tenant Programmable Switches.” Proceedings of the 3rd P4 Workshop in Europe. 2020.

[5] Soni, Hardik, et al. “Composing dataplane programs with μP4.” Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication. 2020.

[6] Zaballa, Eider Ollora, et al. “A perspective on P4-based data and control plane modularity for network automation.” Proceedings of the 3rd P4 Workshop in Europe. 2020.

[7] Osęski, Tomasz, et al. “DPPx: A P4-based Data Plane Programmability and Exposure framework to enhance NFV services.” 2019 IEEE Conference on Network Softwarization (NetSoft). IEEE, 2019.

[8] Osęski, Tomasz, et al. “Offloading data plane functions to the multi-tenant Cloud Infrastructure using P4.” 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). IEEE, 2019.

[9] ETSI ISG NFV, “ETSI GS NFV-IFA 001 V1.1.1: Network Functions Virtualisation (NFV); Acceleration Technologies; Report on Acceleration Technologies Use Cases.” 2015.