P4-MTAGG - a Framework for Multi-Tenant P4 Network Devices

Fabian Brisch*, Andreas Kassler*†, Sándor Laki‡ and Peter Hudoba‡
*Institute of Applied Computer Science, Deggendorf Institute of Technology, Deggendorf, Germany
†Computer Science Department, Karlstads Universitet, Karlstad, Sweden
‡Faculty of Informatics, ELTE Eötvös Loránd University, Budapest, Hungary

Email: *fabian.brisch@th-deg.de,andreas.kassler@th-deg.de †andreas.kassler@kau.se, ‡lakis@inf.elte.hu,
‡peter.hudoba@inf.elte.hu,

***Abstract*—The current P4 programmability model assumes that a P4 programmable device is owned and controlled by a single tenant. However, in typical NFV scenarios, support for multiple tenants is desirable. When each tenant may want to deploy their own P4 pipeline offering different network functions (NF), supporting multiple co-existing tenant pipelines on a single platform is difficult because it requires pipeline merging, control plane support, and resource management of the platform. In this paper, we present P4-MTAGG, a novel framework for flexibly deploying multiple P4 programmable NFs on a programmable match-action pipeline while supporting multiple tenants. P4-MTAGG consists of i) novel compiler-add-ons for automatic merging multiple P4-pipelines, ii) p4runtime-proxy to allow for control plane access of the aggregated pipelines together with policy-based resource management for the P4 target, and iii) orchestrator to automate the provisioning of a network node utilizing aggregation either in a simulated or real hardware environment. In this demo, we show how P4-MTAGG aggregates multiple NFs of varying complexity in Mininet. The user can orchestrate the aggregation process through a GUI. The per-tenant traffic is routed through the set of NFs using segment routing. Through the GUI, the user can instruct the p4runtime-proxy to enforce per-tenant bandwidth limits, which configure the per-tenant available resources in the data plane.**

I. Introduction

Software Defined Networking (SDN) and Network Function Virtualization (NFV) have emerged as an important paradigm for making networks more flexible and programmable. However, deploying network functions (NF) on commodity infrastructure limits the performance of the data plane. Recently, P4 has emerged as an important concept that enables data plane programmability supporting both software and hardware targets. Indeed, P4 has the promise to provide fast packet processing performance when deployed on hardware targets such as switching ASICs (e.g., Tofino) or FPGA-based packet processors while at the same time providing flexibility of software-based definition of networking functions. P4-based NFs can be implemented with less code compared to when implemented in a traditional programming language and can be deployed on multiple targets having different hardware capabilities (e.g. on a smartNIC, FPGA or on a CPU-based host).

However, the current P4 programmability model assumes that the hardware target is under the control of a single entity. Although some targets provide support of several parallel pipelines, the current approach is to deploy these pipelines all under the same administrative control. While each pipeline may implement its own NF, still such approach supports only a single tenant. However, in cloud-based scenarios, the support of multiple tenants on a single common infrastructure is imperative. Consequently, multiple tenants can only be served with the same functionality [4]However, different tenants may have the need to deploy different NFs as they want to implement different networking features or protocol versions.

Supporting multiple tenants in a single programmable target poses multiple challenges, including i) how to merge the individual tenants' NFs so that they can be deployed on the target (as the target typically has less pipelines than number of NFs of all tenants that should be deployed), ii) how to provide control plane access to individual tenants, as the merged NFs typically only expose a single control plane interface that all tenants must share and iii) how to manage the target resources between tenants, such as memory (e.g., TCAM entries, SRAM) processing power (e.g., processed packets per second) or control plane load (e.g., number of installed/updated rules per second).

Some recent works tackled similar issues. For example, $\mu P4$ [5] proposes a novel programming language/framework based on the P4-standard that focuses on modularity and interoperability of different functions. However, support of different tenants is not provided.Placement of NFs and its impact on performance and resource consumption has been discussed in [3]. While it assumes that a host is capable of deploying multiple NFs, the process to support this has not been solved. Supporting multiple tenants on a single platform with implications on the control interface has been discussed in [1], but is limited to fixed-function multi-tenancy. Lemur [6] allows to merge multiple NFs but does not consider multiple tenants and resource sharing policies.

We propose P4-MTAGG, which is a framework to dynamically aggregate/merge and deploy different P4 programmable tenant pipelines composed of multiple NFs. P4-MTAGG consists of i) a Meta-Compile stage (MTAGG-CP) in order to merge different tenant NFs specified in P4 language to synthesize a single aggregated pipeline that supports the functionality

978-3-903176-66-9

of the different tenant NFs, ii) a control-plane proxy (MTAGG-Proxy) which translates P4-Runtime requests for aggregated pipelines and enforces per-tenant resource allocation policies and iii) a central orchestrator automating the aggregation and deployment functionality. During the aggregation stage, P4-MTAGG merges individual NFs parsers, tables and processing logic and wires the individual NFs together using segment routing. It adds additional resource management functionality in the data plane by adding per tenant meters that are exposed to the control plane proxy to enable rate-limiting per tenant packet processing according to the resource sharing policies. Our demonstration illustrates the capabilities of our framework to aggregate NFs from different tenants implemented in P4 and deploy them on P4-capable targets with a single click on a GUI. In the demo, the users can interact with the aggregated NFs from individual tenants through the control-plane proxy. Finally, we demonstrate the capabilities of our framework to enforce per tenant resource allocation policies by limiting the allocated rate for each NF while the GUI will show the per tenant packet throughput.

II. P4-MTAGG Framework

The proposed framework consists of three key elements, the Meta-Compiler (MTAGG-CP), control plane proxy (MTAGG-PROXY) and the orchestrator (MTAGG-ORCH) (see Fig. 1). Running the framework requires a configuration describing tenants, NFs and resources available for NF deployment. The configuration consists of a set of P4 pipelines of all NFs with given SRv6-segment IDs [2] to be used for packet identification. The configuration further provides the MTAGG-ORCH with information about available targets to run the NFs.

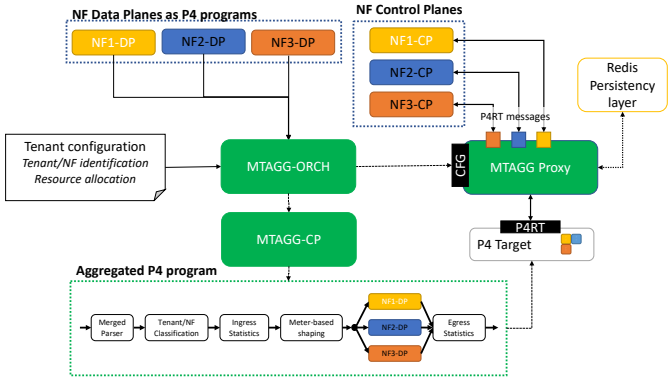

Fig. 1. P4-MTAGG framework components

A. MTAGG-CP

The Meta-Compiler combines multiple P4-pipelines while preserving individual functionality and injecting additional functionality into the aggregated pipeline for resource management and monitoring. The Meta-Compiler i) merges individual parsers into a single one capable to parse packets as defined in all source pipelines, ii) combines the match-action stages from all pipelines, dealing with possible naming collisions and iii) adds additional tables and parser states to identify incoming packets based on SRv6-header and segment IDs provided in the configuration and apply the correct NF, collect statistics about per tenant NF packet processing rates and resource (e.g., bandwidth, memory) usage and apply rate-limiting to each NF as configured by the proxy.

The P4-MTAGG compiler extension is based on the T4P4S compiler, utilizing HLIR-16 for internal P4-representation and manipulation. The compiler extension takes multiple .p4 files as input and outputs, in conjunction with the t4p4s compiler output, an additional .p4 file of the aggregated pipeline as well as p4runtime-information for use in the proxy/other compilers for different hardware targets.

B. MTAGG-PROXY

The communication between the P4 target running the NF data plane and the corresponding control plane occurs through a gRPC-based P4Runtime interface. This way, the NF's control plane can access the data plane objects (e.g., tables, counters, registers). After aggregation, the resulting P4 code contains all the data plane objects of the aggregated NFs. As a result, the P4 target running it opens a single P4Runtime interface through which all the data plane objects of the aggregated NFs would be available. MTAGG-PROXY is introduced to isolate the access of various control planes to the data plane objects. The proxy launches independent P4Runtime servers for each tenant and NF to control and isolate the control plane access to the data plane objects. Through the proxy, the control plane of each NF and tenant can manage the NF data plane as if it were running on the P4 target alone without aggregation. It receives gRPC-based P4Runtime messages from the individual control planes, translates the data plane object identifiers of the individual P4 program to the identifiers in the aggregated program, and sends the translated P4Runtime message to the P4 target running the aggregated data plane via gRPC. The reply messages are translated back similarly. The configuration needed for mapping the identifiers is created in aggregation time by MTAGG-CP.

The proxy has a configuration interface that allows the setting of resource handling policies for both control (rule insert/update operations per second, table entries, registers, meters, and counters - per tenant) and data planes (allowed packets and bytes per tenant processing rate). These policies are used to configure the per-tenant rate limiters (i.e., meters) in the synthesized aggregated data plane code and regulate the control plane message rates for each tenant and NF.

C. MTAGG-ORCH

Meta-Compiler and Proxy are controlled by the Orchestrator. It reads the framework config, executes the aggregation process and handles deployment of network functions. The Orchestrator also configures the proxy with addresses and ports of the aggregated program. The demo controller is implemented in python and can be configured with a set of network-functions as well as P4 targets. It displays the state of the configured functions and P4 targets in a web-based GUI. We currently support the following P4 targets: BMv2

	NF1	NF2	NF3	Aggregation
Nr. of parsed headers (ex. eth,ipv6)	0	5	10	$\sum \text{NF}_{i,j}+2$
Nr. of tables	1	6	11	$\sum \text{NF}_{i,j}+1$
Hash-calculations	0	1	5	$\sum \text{NF}_{i,j}+0$

TABLE I
COMPLEXITY OF NETWORK FUNCTIONS

switches in Mininet, Intel Tofino, Intel DPDK (through t4p4s) and Netronome Agilo SmartNIC.

III. DEMONSTRATION

A. Demonstration Setup

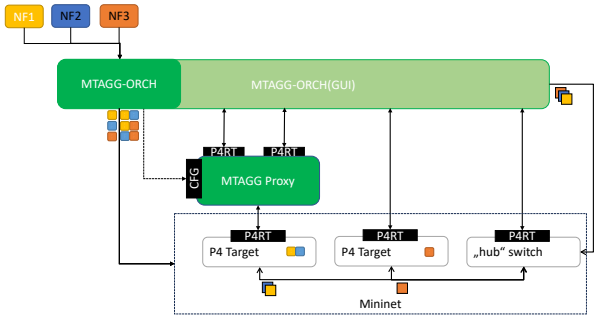

Fig. 2. Figure of setup

In this demo, we demonstrate the proposed framework in a Mininet-based emulated network. Figure 2 depicts the demo setup, which consists of a central programmable switch handling traffic distribution and metering as well as several P4 programmable targets for running aggregated data plane programs. The central switch is responsible for steering the generated traffic to the target where the NF of the given tenant is deployed. In the demo, we will showcase the aggregation functionality using three NFs designed to have significantly different resource demands. They vary in the number of headers parsed, number of match-action tables and hashing functions applied (see Table I).

B. Demo scenarios

We use a web interface and a GUI to show the current state of the targets (see Figure 3). On the top left, current state of P4 targets and their deployed NFs can be seen, while table entries for each NF as well as their current processing latency can be seen on the bottom. The network topology and throughput of all connections are visualized on the right.

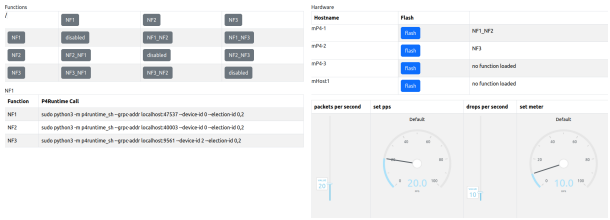

Fig. 3. Demonstrator User Interface

a) Aggregation and Provisioning: In this scenario, we demonstrate the orchestrator's capability to dynamically deploy pipelines in an aggregated (as well as non-aggregated) state. Conference Attendees can inspect and change the current deployment via a web-interface, which will start a reconfiguration process of the target, running the aggregation framework, deploying the new pipeline and populating table entries, if necessary.

b) Aggregated Operation: In this scenario, we demonstrate traffic steering within an aggregation-capable P4 target. Traffic is steered through the central switch, which also monitors target-state and allows packets to pass through various network functions in sequence. Conference Attendees can view real-time packet flow statistics and observe adjustments made in the central switch when (aggregated) NFs are deployed onto targets.

c) Fairness: In this scenario, we demonstrate per-tenant resource allocation and the effects on throughput and latency. Conference Attendees can change the traffic volume sent to the different network functions and observe the impact on packet loss. Conference Attendees can change per-NF meters to control packet flow.

ACKNOWLEDGEMENTS

Parts of this work has been supported by the Bavarian State Ministry of Education and Culture, Science and Art through the High-Tech Agenda (HTA) and the Knowledge Foundation of Sweden (KKS) through project DRIVE. The research leading to these results has also received funding from the European Commission thought the HORIZON 6G SNS JU DESIRE6G (G.A. 101096466). S. Laki also thanks the support of National Research, Development and Innovation Office - NKFIH, FK_21 138949.

REFERENCES

[1] Buck Chung, Chien-Chao Tseng, Jim Hao Chen, and Joe Mambretti. P4MT: Multi-Tenant Support Prototype for International P4 Testbed. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 1–2, Cambridge, United Kingdom, September 2019. IEEE.

[2] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li. Segment Routing over IPv6 (SRv6) Network Programming. Technical Report RFC8986, RFC Editor, February 2021.

[3] Hasanin Harkous, Bassel Aboul Hosn, Mu He, Michael Jarschel, Rastin Pries, and Wolfgang Kellerer. Performance-Aware Orchestration of P4-Based Heterogeneous Cloud Environments. *IEEE Transactions on Network and Service Management*, 20(4):4765–4778, December 2023.

[4] Mu He, Arsany Basta, Andreas Blenk, Nemanja Deric, and Wolfgang Kellerer. P4NFV: An NFV Architecture with Flexible Data Plane Reconfiguration. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 90–98, November 2018. ISSN: 2165-963X.

[5] Hardik Soni, Myriana Rifai, Praveen Kumar, Ryan Doenges, and Nate Foster. Composing Dataplane Programs with P4. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 329–343, Virtual Event USA, July 2020. ACM.

[6] Jane Yen, Jianfeng Wang, Sucha Supittayapornpong, Marcos A. M. Vieira, Ramesh Govindan, and Barath Raghavan. Meeting SLOs in cross-platform NFV. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 509–523, Barcelona Spain, November 2020. ACM.