Automation of Modular and Programmable Control and Data Plane SDN Networks

Eder Ollora Zaballa
DTU Fotonik
Technical University of Denmark
Kongens Lyngby, Denmark
eoza@fotonik.dtu.dk

David Franco
Department of Communications Engineering
University of the Basque Country
Bilbao, Spain
david.franco@ehu.eus

Eduardo Jacob
Department of Communications Engineering
University of the Basque Country
Bilbao, Spain
eduardo.jacob@ehu.eus

Marivi Higuero
Department of Communications Engineering
University of the Basque Country
Bilbao, Spain
marivi.higuero@ehu.eus

Michael Stübert Berger
DTU Fotonik
Technical University of Denmark
Kongens Lyngby, Denmark
msbe@fotonik.dtu.dk

***Abstract*—In the last years, Software-Defined Networking (SDN) has provided a new approach to network programmability, first regarding the control plane and later the data plane. With the popularity of the data plane programming languages like P4, SDN network automation has extended from developing control plane applications and deploying controllers to integrating custom packet processing pipelines in this process. However, developing control and data plane applications can become burdensome since expertise in both fields is scarce. The process of automating SDN networks requires (among many tasks) inter-plane correlated application self-collection and assembly. As a result, the orchestrator presented in this paper, named *P4click*, provides high-level interfaces in order to transparently deploy modular control and data plane applications for SDN networks. This paper describes the architecture design of the orchestrator, outlines the deployment structure, and provides a general view of control plane application deployment and data plane pipeline assembly. Besides, P4click requires no previous knowledge of data plane programming and provides a simple interface for network operators that have to deploy new network functionalities. The results in this paper show which tasks in the network automation are most influential (timewise) in bringing a network up and running from the ground up.**

***Index Terms*—SDN, P4, control plane, data plane, automation, deployment**

I. Introduction

Research on Software-Defined Networking (SDN) has gained popularity since the OpenFlow protocol was first published [1]. However, control and data plane decoupling were already a research topic years before the OpenFlow publication. For instance, Devolved Control of ATM Networks (DCAN) or Open Signaling happened in the mid-1990s. Still, OpenFlow is considered to be the *de facto* protocol for the first generation SDN. A considerable number of vendors support it in their hardware and software switching gear too. As the OpenFlow versions evolved, more headers and actions were being supported by the switches.

The changes in OpenFlow to adopt more features and headers implied having to extend the standard continuously. The ever-increasing use cases and new headers that have become popular requested new approaches for programmable networks. Research and industry have focused on programmable data planes (e.g., Application-Specific Integrated Circuits (ASICs)) when the P4 language was first published by Bosshart *et al.* [2]. With the advent of OpenFlow and P4, becoming proficient in control and data plane programming is complicated. It implicates proficiency in algorithms, expertise in different languages, and infrastructure management. Developers and industry miss the connection of network programming, deployment, and automation. The management of networks with many flexible and variable components requires an all-in-one approach to cover the necessary aspects.

This paper presents an extended work on P4click [3], an orchestration tool that automates programmable network deployment and management using P4-based modules and control applications (from the Open Network Operating System (ONOS) controller). Section II will introduce other research works related to the developments in P4click. Sections III and IV serve as a general view of the platform and how it manages to merge modules and deploy them. The use cases will list the applications on which P4click could be useful. The Proof-of-Concept (PoC) results (Section V) show the time aspects of a full ground-up network deployment, and thus, the time necessary to wait until P4click brings all necessary network components up. The tests also show how much two tenants within a network need to wait until the whole process is finished and connectivity exists between them. Finally, the paper lists the major challenges (Section VI) of the current platform, future work and conclusion (Sections VII and VIII).

II. Related Work

Previous research has already focused on SDN/NFV automation and deployment. Gharbaoui *et al.* [4] describe the automation and deployment of a tenant-based SDN network. The use case shown is built with the Open Source

978-3-903176-36-2 ©2021 IFIP

MANO, configuring Virtual Network Functions (VNFs) as SDN switches and controllers. Their results point out that the deployment time is dependent on the number of VNFs being used. Riftadi *et al.* [5] created an Intent Definition Language (IDL) that uses templates that are later merged to generate P4 code. The presented framework by Riftadi can install and remove intents in the field using their Proof-of-Concept (PoC) implementation. A later work by Riftadi [6] proposed a new framework with a genetic programming approach to generate new P4 programs.

The work presented in the current paper (P4click) uses preconfigured and simplified static modules to create a merged pipeline. However, other researchers have published about modular data plane programming, covering an elaborate and extensive work. One early paper that merges P4 and modular programming introduced *ClickP4* [7] (the similarity with this name is coincidental). Zhou and Bi present a programming architecture that decomposes monolithic programs and creates reusable modules. ClickP4 uses a token-based design to decide which features process the packet in the pipeline. A later paper by Zhou, presented FlexMesh [8], a further iteration to the one presented before [7]. In this later work, Zhou *et al.* present an elaborate data plane model runtime-configurable with user-defined data plane function chains.

Additionally, Soni *et al.* [9] proposed a system that deploys independently created modules and merges them (parsers, deparser, control logic, etc.). The system uses a *Linker* component that merges modules and refactors, decomposes, and schedules tables. A later research work by Soni *et al.* [10] presented a new modular programming framework named μP4. The authors define the framework as modular, composable, and portable. This second research work from Soni shows a high-level abstraction that can build pipelines with target-agnostic modules. Finally, Hancock *et al.* [11] and Zhang *et al.* [12] propose several solutions for data plane virtualization.

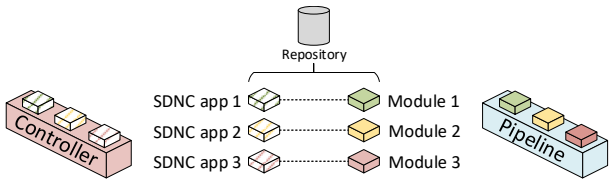

Fig. 1. Data plane modules have a matching control plane application deployed at the same time.

III. Design

P4click is a platform that orchestrates data plane modules and control plane applications into SDN network automation and deployment. As seen in Figure 1, P4click relies on the simple principle of matching data plane modules with SDN controller (SDNC) applications. One data plane module matches an SDN control plane application, and both are pulled from the repository simultaneously.

Overall, the system will let network managers decide which control plane infrastructure to use, that is, the type and number of controllers, the switching gear to use, and how the data and control plane look like as seen in Figure 2. The general overview in Figure 2 is further detailed in Figure 3.

Currently, P4click accepts CLI input and blueprints or descriptors to build modular SDN deployments. The platform can download data plane modules from a central repository and merge them. It also deploys the corresponding control plane applications in a centralized control plane. When the platform builds the merged pipeline, it is responsible for supplying the resulting *p4info.txt*, and *bmv2.json* files to one of the control plane applications.

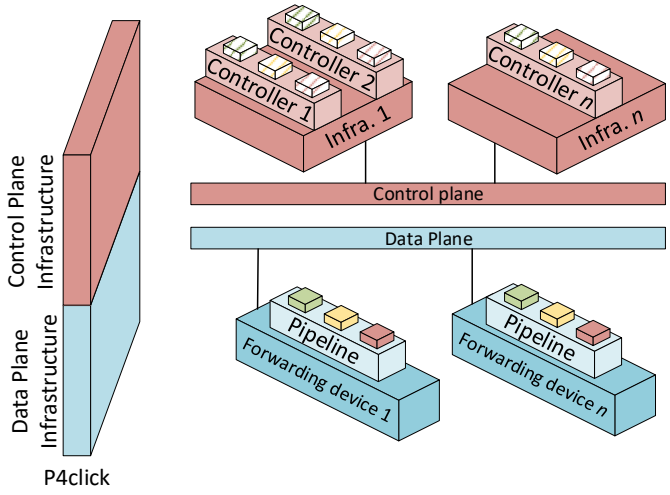

Fig. 2. An overview of the P4click orchestrator as per the whole deployment architecture.

These files are necessary for the *pipeconf*, which relies upon one of the various control plane applications. In the current PoC implementation, the Access-control list (ACL) SDNC application is responsible for pipeconf configuration in order to keep the ACL and the rest of the modules independent. In the tests shown in the Section V, the ACL SDNC application handles *packet_in* messages for LLDP, ARP, or IPv4 packets. The rest of the SDNC applications are responsible for L2 forwarding and L3 forwarding.

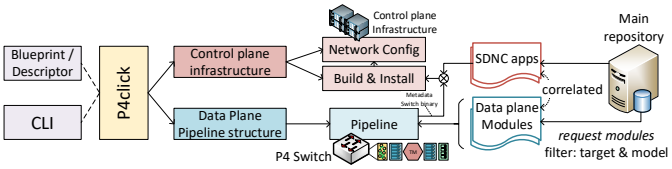

Fig. 3. Procedure for generating a data plane pipeline and the control plane applications.

Composing the pipeline: A straightforward approach

In order to provide a simple procedure to generate data plane pipelines, modules are packaged as compressed files (with a *.p4z* extension) that P4click retrieves from a central repository. The control plane applications are also retrieved

and deployed later on. When P4click has decompressed the data plane modules, it checks the configuration files (from each module) to process the headers, parser, deparser, etc. It analyses all modules' parsers and merges them: the platform detects the same parse states, the same forward transitions, the same backward states, builds merged conditional transitions, and so on. It finds the proper deparsing structure by identifying the parsing stage of each header. For instance, Figure 4 shows IPv4 and IPv6 headers being parsed after Ethernet, following the same order in the deparser.

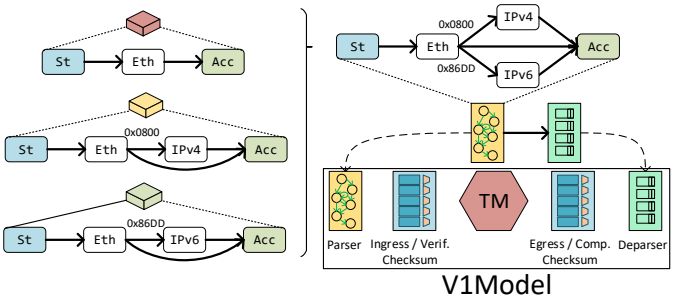

Fig. 4. Merging 3 different module parsers into the same union parser and deriving the deparser structure.

The Ingress and Egress match-action pipelines consist of sequentially applying control blocks from each of the modules (Figure 5). This approach for integrating the P4 logic into the merged pipeline expects that control blocks can run independently. However, modules can detect that if a particular functionality was executed. For instance, in Section V's test case, the ACL module checks if any module has set the egress port for the current packet being processed (to trigger *PACKET_IN* messages). In terms of checksum verification and computation, the functionality is partially supported too.

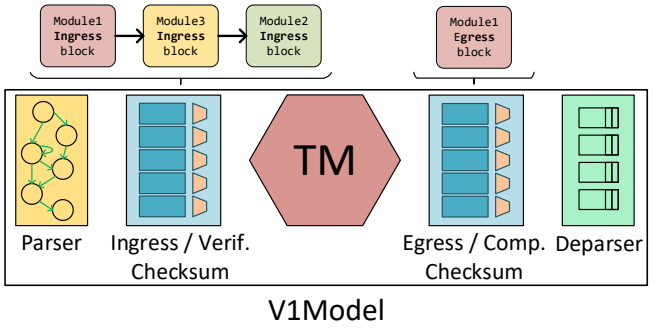

Fig. 5. Integrating control blocks from modules to the Ingress and Egress match-action pipelines.

IV. Use cases

The design of P4click to automate control and data plane deployment and configuration envisions several use cases. This section depicts three main use cases:

- Gradual module aggregation: Involves adding new functionalities to one or more existing switches in the network. Since P4click requires no P4 or control plane programming knowledge, a non-expert network operator can transparently add modules to the existing switches.
- Sliced pipeline: Secondly, P4click can be used to assign modules or features to different tenants. The switch would identify tenants by ingress port, VLAN IDs, IP subnet, etc. Then control blocks from modules can be linked to one of the existing slices and their tenants (works similar to [11]).
- NFV MANO component: Third, due to the open interfaces of P4click, third-party network configuration and management entities can deploy services and network control applications with a blueprint/descriptor.

V. Results

In the following tests, P4click [13] is running on a virtual machine (VM) with 32 GB of RAM and 3 cores from an Intel Xeon E5-1680. To emulate the network, Mininet is used with the Stratum OS and BMv2 software switches, all running on a container. The controller is the Open Network Operating System (ONOS) version 2.2. As mentioned in the previous sections, P4click handles a complete SDN network management: from controller deployment, through control and data plane app retrieval and building, to providing tenant connectivity. Therefore, our tests have measured the necessary time to build a ground-up modular SDN network. The test case includes one tenant (tenant A) constantly sending ICMP requests to another tenant (tenant B). The test shows the time that each relevant task takes in the network deployment until connectivity between tenants exists (i.e., ICMP responses arrive at tenant A). With this information, each task can be compared with others and contextualize the results.

P4click first receives a blueprint or processes CLI commands to define pipeline modules, features, infrastructure details, etc. P4click is responsible for first building the pipeline that includes different modules. Left boxplot at Figure 6 shows that P4click needed an average of *220 ms* to collect compressed modules and control plane apps from the central repository and decompress them. The central repository runs an HTTPS server (see Figure 1). The outlying results in module retrieval are due to the inconsistencies of the library when uncompressing modules. To build the merged pipeline (right boxplot at Figure 6), the orchestrator spent 16 ms on average. This task involves assembling the parser, ingress and egress blocks, and the deparser.

Once the P4 pipeline is built, P4click is responsible for deploying the ONOS controller(s). ONOS is deployed locally in a container, but P4click supports deploying the SDN controller in any machine as long as the addressing and authentication information is provided. Deploying ONOS is the longest task, which takes 59.5 seconds on average (left boxplot at Figure 7). Since no hardware switches were used, P4click deployed Mininet and the Stratum+BMv2 switches in a container (with a predefined topology and scenario). This task lasts 2.2 seconds on average (middle boxplot at Figure 7). Once the controller and the switches are deployed, P4click provides a network configuration to the controller used for various

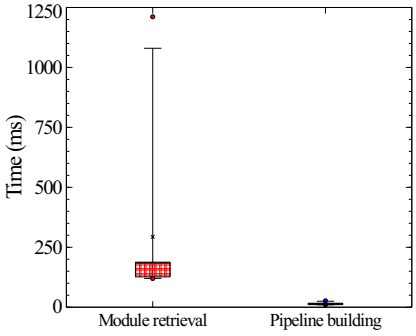

Fig. 6. P4click's module retrieval process (left) and assembling a union pipeline (right).

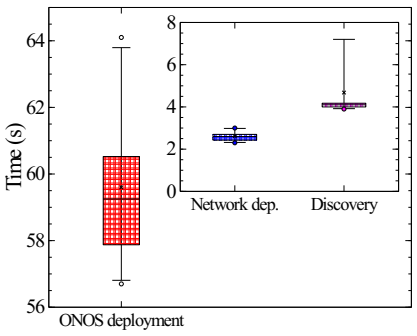

Fig. 7. ONOS deployment (left), data plane infrastructure deployment using Mininet and Stratum+BMv2 (center) and link discovery time (right).

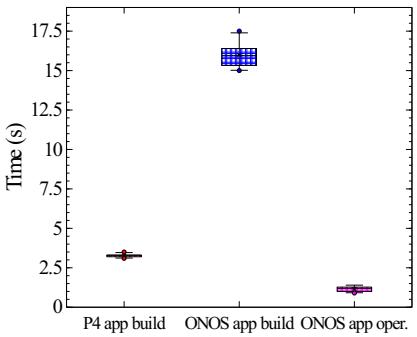

Fig. 8. P4 application build time (left), ONOS application build time (center) and time to install the ONOS application and become operational (right).

tasks, mainly, to define the network switches configuration and management information. The network configuration results in network discovery (i.e., ONOS detecting the switches and links). Events like controller-switch connections and networks discovery (among others) last 4.1 seconds on average in total (right boxplot at Figure 7).

Once the pipeline is assembled and the infrastructure deployed, P4click compiles the pipeline and distributes the *p4info.txt* and the BMv2 configuration files to the ACL SDNC application, since that is the one carrying the *pipeconf* for the SDN controller. The average time to build the P4 app is under 3 seconds (left boxplot at Figure 8). When these tasks are finished, P4click compiles the ACL, L2 forwarding, and L3 forwarding applications (used in this example) if necessary. Generally, the repository distributes control plane applications as *oar* packages. Building control plane apps lasts 16 seconds on average (middle boxplot at Figure 8). Installing ONOS apps on the controller until they become operational slightly varies from app to app, but average results show that after 1.5 seconds, the apps are ready to function (right boxplot at Figure 8).

Considering the results of all tasks, the overall tests can show how much time it takes to build a modular SDN network from scratch. In other words, the time it takes until connectivity exists between tenant A and tenant B. From the moment that P4click starts processing the last submitted command or deployment descriptor until the first ICMP response arrives at tenant A: 70.53 seconds passed (worst case scenario). If the controller is already running, the wait time is reduced to 11.03 seconds. Consider that these results did not include the time taken to build the control plane applications but included the rest of the network events.

VI. Challenges

Merging data plane modules and having different control plane applications running in the same controller implies great complexity. Configuring the control and the data plane requires independent code to run seamlessly. We list some challenges detect while designing the system depicted in this paper.

Inter-module communication

In our approach, modules are expected to run independently without passing any information among modules. According to the current design, the module execution logic entails calling different control blocks per module. This simplifies the procedure of integrating different modules' logic. However, some cases might need an efficient communication interface that can indicate if a certain action happened. For instance, if a module performed a forwarding decision (that might not be overridden), other modules might need to check if this happened. To exemplify this case, in our PoC test, L2 and L3 forwarding modules change the *fwd_done* user metadata field to *true* when the egress port is set to a value. This metadata indicates to other modules that forwarding decisions were made. A similar case happens when a firewall decides to drop a packet that could also be forwarded to a port by another module integrated into the pipeline. While this is partially solved by selecting the order of module execution, standardized user or local metadata fields (like *fwd_done*) become a valuable communication interface for modules.

Control plane dependencies

The first conceptual design of P4click intended to have independent control plane applications. This can be achieved since several controllers support archive-based applications that can be imported at runtime. However, incorporating applications that support flexible pipelines implies translating high-layer forwarding abstractions for applications using abstractions for OpenFlow-based pipelines. For instance, link discovery applications that process LLDP packets request that any LLDP packet must be sent to the controller. This is directly mapped to a flow rule with a particular Ethernet type value as key and an output action to be sent to the controller. This further implies that mapping high-level forwarding abstractions from pipeline-agnostic applications can only be done by the pipeline interpreter (typically by the pipeconf in the ONOS controller) associated with the merged pipeline. This condition requires that specific mappings and forwarding objective translations must be integrated too in the control plane or delegate to only

one of the control plane applications (as done in our PoC implementation).

The cost an inter-layer integration

In the design depicted in this paper, data plane modularity follows a rigid but straightforward process. The logic of a module is contained within one or more P4 control blocks, and the rest of the pipeline information is outlined as YAML configuration files (although future developments could parse P4 syntax directly, without needing YAML configuration files). This type of module brings simplicity to users as no knowledge of data plane programming is required to build a pipeline, which differentiates P4click from other projects referenced in Section II. Unfortunately, this design implicates a very tight architecture and module composition since control plane applications are bound to the module as written when packaged as a p4z module. Further pipeline changes when integrating several modules (e.g., mixing control blocks, changing tables, action names, headers/fields, etc.) would also involve changes in the control plane.

VII. Future Work

This paper presents an all-in-one orchestrator to deploy modular data and control plane applications in SDN networks. However, covering a wide variety of aspects in network automation and deployment implies simplifying several aspects too. Having pre-built control plane applications implies that no data plane header, fields, tables, control block statements, etc., can change. There is very little flexibility in this sense as the current PoC implementation does not consider different header names, header field sizes, or names if modules are extended. Besides, the modular approach is simple and straightforward compared to the ones described [8] [10] [14] in Section II, which offer other value-added features. P4click does not bound to any module structure, so integrating with other projects like μP4 [10] is a plausible extension in order to integrate more robust modularity. Still, further integration with other modular structures requires updating compatibility with control plane applications and how those applications are managed.

VIII. Conclusion

This paper describes the structure, functionality, and working procedure of the P4click orchestrator. To our knowledge, this is one of the first works that try to encompass a SDN cross-layer modular network automation. The results show that that pipeline assembly and compilation are not the bottleneck in the network but starting the SDN controller is. Having an already active controller lowers down tenant connectivity time by almost 60 seconds. If the network deployment only involves control and data plane application retrieval, assembly, and deployment, (no control/data plane infrastructure deployment) the total wait time lowers to 4.7 seconds.

Acknowledgment

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 nEtwork and services for beyond 5G" (TRUE-5G), by the "Smart Factories of the Future" (5G-Factories) (COLAB19/06) project and by the Basque Country Department of Economic Development, Sustainability and Environment through the project (KK2021-00026) titled "Federated Infrastructure for Experimentation in Industry 4.0 applications" (B-Ind5G).

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, p. 69–74, Mar. 2008. [Online]. Available: https://doi.org/10.1145/1355734.1355746

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[3] E. Ollora Zaballa, D. Franco, M. S. Berger, and M. Higuero, "A perspective on p4-based data and control plane modularity for network automation," in *Proceedings of the 3rd P4 Workshop in Europe*, ser. EuroP4'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 59–61. [Online]. Available: https://doi-org.proxy.findit.dtu.dk/10.1145/3426744.3431330

[4] M. Gharbaoui, B. Martini, and P. Castoldi, "Programmable and automated deployment of tenant-managed sdn network slices," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–6.

[5] M. Riftadi and F. Kuipers, "P4i/o: Intent-based networking with p4," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 438–443.

[6] M. Riftadi, J. Oostenbrink, and F. A. Kuipers, "GP4P4: enabling self-programming networks," *CoRR*, vol. abs/1910.00967, 2019. [Online]. Available: http://arxiv.org/abs/1910.00967

[7] Y. Zhou and J. Bi, "Clickp4: Towards modular programming of p4," in *Proceedings of the SIGCOMM Posters and Demos*, ser. SIGCOMM Posters and Demos '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 100–102. [Online]. Available: https://doi-org.proxy.findit.dtu.dk/10.1145/3123878.3132000

[8] Y. Zhou, J. Bi, C. Zhang, M. Xu, and J. Wu, "Flexmesh: Flexibly chaining network functions on programmable data planes at runtime," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 73–81.

[9] H. Soni, T. Turletti, and W. Dabbous, "P4Bricks: Enabling multiprocessing using Linker-based network data plane architecture," Feb. 2018, working paper or preprint. [Online]. Available: https://hal.inria.fr/hal-01632431

[10] H. Soni, M. Rifai, P. Kumar, R. Doenges, and N. Foster, "Composing dataplane programs with μp4," in *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*, H. Schulzrinne and V. Misra, Eds. ACM, 2020, pp. 329–343. [Online]. Available: https://doi.org/10.1145/3387514.3405872

[11] D. Hancock and J. van der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 35–49. [Online]. Available: https://doi.org/10.1145/2999572.2999607

[12] C. Zhang, J. Bi, Y. Zhou, A. B. Dogar, and J. Wu, "Hyperv: A high performance hypervisor for virtualization of the programmable data plane," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, 2017, pp. 1–9.

[13] E. Ollora Zaballa and D. Franco, "P4click," https://github.com/ederollora/CNSM_2021_P4click, 2021.

[14] R. Stoyanov and N. Zilberman, "Mtpsa: Multi-tenant programmable switches," in *Proceedings of the 3rd P4 Workshop in Europe*, ser. EuroP4'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 43–48. [Online]. Available: https://doi.org/10.1145/3426744.3431329