MTPSA: Multi-Tenant Programmable Switches

Radostin Stoyanov
University of Cambridge
radostin@stoyanov.io

Noa Zilberman
University of Oxford
noa.zilberman@eng.ox.ac.uk

ABSTRACT

Virtualized multi-tenant programmable switches enable on-demand support of different users' protocols and programs. However, supporting multiple tenants on a virtualized switch raises concerns such as resource isolation and security. Truly isolating users is mandatory for virtualized programmable switches to be deployed in production networks. In this paper we propose MTPSA, a Multi Tenant Portable Switch Architecture. MTPSA offers performance, resource and security isolation. It further introduces *roles* and *privileges* within programmable switches. MTPSA is an open-source contribution, implemented over PSA and NetFPGA. Our evaluation shows that it adds minimal overheads, supports line-rate throughput, and scales with the number of users, while providing an isolation of users.

1 INTRODUCTION

Over the last decade, programmable network devices have emerged as a promising alternative to traditional fixed function switches and network interface cards (NICs). Programmable data planes (PDP) allow network operators to define custom packet forwarding policies, typically using a high-level, domain-specific programming language, such as P4 [2].

PDPs not only enable network operators to use protocols of choice, but also to offload functions that traditionally run on servers to the network [15]. Offloading functions to the network saves CPU cycles for user applications and benefits from the high performance of network devices. However, supporting more than one user function per PDP requires appropriate virtualization mechanisms [6].

Virtualization is widely adopted in cloud environments, increasing the utilization of resources and reducing operating costs. In the context of programmable network devices, PDP virtualization allows a single physical data plane to logically support multiple networking contexts, where the packet processing of each context can be defined independently by a (P4) program.

Multiple previous works have considered the virtualization of PDP [6], focusing on the challenge of merging P4 programs into a single PDP configuration [3, 13, 16, 21], and on designing a special-purpose P4 program (hypervisor) to serve as platform that supports multiple P4 programs [7, 20]. These solutions, however, do not provide sufficient virtualization capabilities. The critical elements in virtualization are i) secure execution of user programs [7], so one user cannot observe data used by another's program; ii) limited temporal interference (performance isolation) among users [6], despite them running on the same physical data plane; iii) resource isolation [16], where each user has allocated set of resources.

Multi-Tenant Portable Switch Architecture (MTPSA) provides a solution to these virtualization concerns, by introducing a multi-tenant architecture that compartmentalizes user programs to provide isolation. By applying operating-system level roles and privileges concepts to the PDP, MTPSA enables secure execution of users programs.

To support the virtualization of PDP, we make the following contributions:

- We propose mechanisms for security, resource, and performance isolation of user P4 programs.
- We introduce a new architecture, MTPSA, supporting multiple isolated tenants over PSA.
- We present an implementation and evaluation of MTPSA on hardware and software targets.

MTPSA is open-source and all the code is available at [14].

2 PDP VIRTUALIZATION

Ordinarily, the PDP of a network device represents a single network context that can be programmed, for example, using P4 [2]. A P4 program specifies a collection of protocol headers to be *parsed*, and match conditions with actions to *process* parsed headers. To maximize resource utilization and minimize total cost of ownership, data centers networks are often shared by many organizations and users, despite having potentially different packet forwarding, bandwidth and security requirements.

A virtualized PDP allows deploying multiple programs on a single network device. Each program operates in an isolated network context. Operations performed in one context are not affected by actions carried out in other contexts. This technique enables simultaneous deployment of multiple forwarding policies, for example, to isolate sets of users or equipment (e.g., public vs. private), and where each networking context can support unique protocol types and functionality. Additionally, PDP virtualization allows quick transitions between network configurations by storing multiple configurations, while a single configuration is active at a time.

3 DATA PLANE ISOLATION

Isolation of user programs in the PDP may refer to different aspects. In this section, we define three different types of data plane isolation, and discuss their usage.

Resource Isolation Resource isolation is a common form of user programs isolation [6, 16]. It means that some resources or state in the PDP, such as tables (or table entries), registers or externs, are dedicated to a specific program. These dedicated resources are not shared with other programs from the control plane perspective. For example, table x is dedicated to program A, while table y is dedicated to program B.

Performance Isolation Performance isolation of user programs means that the execution of one program does not affect the performance of another. For example, if program *A* achieves certain throughput when running alone, it should achieve the same throughput also when program *B* is running, given a similar resource allocation.

Security Isolation Security isolation of user programs refers to aspects of isolation, predominantly access, for security purposes. An example is preventing packets controlled by program *A* from accessing data stored by program *B*. While Resource Isolation refers to the assignment of resources, Security Isolation refers to the access to resources (§4).

3.1 Isolation for Security

Most of PDP virtualization efforts to date have focused on resource [13, 21] or performance [12] isolation. In this work, we focus on isolation mechanisms for security purposes. In particular, how can we prevent malicious actors from accessing or interfering with other users' programs?

Often security isolation in a PDP requires enforcing a set of policies for traffic filtering before and after a packet is provided to a user program. For instance, access control list (ACL) rules can be used to limit the ingress ports a user's program is allowed to receive packets from, and restrict the egress ports it is allowed to send traffic on.

Attacks on PDP can vary significantly [4]. For example, a malicious user can create a Denial-of-Service attack by cloning and recirculating packets infinitely. This attack vector can be mitigated by limiting the number of clone and/or recirculate operations per packet.

Another security concern is preventing users from obtaining information from packets processed by other user programs. For example, the BMv2 framework employs an optimization technique that allows to "recycle" packet header vector (PHV) objects, an approach taken due to the high performance costs of memory allocation and deallocation. Such optimizations, when used with PDP virtualization, should isolate the PHV pool of each user, i.e., similar to the way Linux namespaces are used with containers.

4 ROLES AND PRIVILEGES

To support the isolation of user programs in the pipeline, we introduce concepts from the operating system world to the data plane, and in particular *Roles* and *Privileges*.

In operating systems a Superuser account, often called *root* or *administrator*, is used for system administration, and has full privileges with no access restrictions. Similarly, we define in the PDP a *Superuser* as a user that has access to everything running within the network device, and with permissions to run any available operation. The Superuser will typically be the network administrator. Non-Superusers are referred to as *Users*. With PDP executing programs originating from multitude of sources, it is imperative to distinguish between these two types of users and control the capabilities of Users' programs.

Each User is assigned *permissions* to execute operations within the PDP. These permissions, dictate aspects such as accessibility to tables and externs, which packets can be observed, and which operations are limited or forbidden (e.g., recirculation, mirroring).

In practical scenarios, a multi-tenant networking environment requires isolation mechanisms that provide a holistic security model - *confidentiality* (prevents unauthorized access and misuse of data), *integrity* (protect data from unauthorized alteration) and *availability* (timely and uninterrupted access of authorized users) [11].

Unlike operating systems, where users are directly assigned privileges to execute actions, here the privileges are associated with the packets going through the pipeline. A packet going through the pipeline, and its metadata, will "pass" only through authorized programs, and will be allowed only an assigned set of operations.

Consider an example where programs *E* and *A* are consecutive, and where program *E* is malicious. Program *A* implements a congestion control mechanism by looking at queue occupancy, where queue size is carried over a metadata bus. Malicious actions by program *E* may include, for example, always setting to 0 (or maximum) the queue size on the metadata bus, leading to incorrect congestion control by *A*. This act can be prevented if *E* doesn't have the privilege to change the queue size metadata field. A malicious *E* can also change header information, such as priority (in an attempt to drop a packet), an action that can be prevented by revoking the privilege to change certain header fields.

There are multiple types of entities and actions that require privileges. The first type is headers, where one privilege is required to read a header, and another to modify it. Similarly, metadata fields also require separate read and modify privileges. Memory requires separate read and modify privileges, however where tables are modified through the control plane, a new type of privilege is added - execution. A program may be allowed or forbidden certain execution rights for *actions*, such as on mirroring, recirculation, control-plane notifications and others.

5 ARCHITECTURE OF MTPSA

MTPSA, Multi Tenant Portable Switch Architecture, extends PSA [10] to support multiple programs within the same network device, while providing security, performance and resource isolation, as well as on-the-fly reconfigurability. A key concept of MTPSA is utilizing a *Superuser* P4 program that defines the pre- and post-processing of packets, while supporting multiple *User* P4 programs that specify the core packet processing logic.

Figure 1 illustrates the data plane architecture of MTPSA. Similar to PSA, MTPSA has an ingress pipeline and an egress pipeline. These two pipelines are the *Superuser* program. User programs are represented as user pipelines within the architecture, with all user pipelines sharing the same Superuser ingress and egress pipelines.

5.1 Superuser Pipeline

The Superuser P4 pipeline, programmed and controlled by the device's administrator, acts as a the hypervisor of the network device. The Superuser pipeline has multiple roles, including:

- Packet pre-processing (Ingress).
- Packet post-processing (Egress).
- Associating incoming packets with a user identifier.
- Role and privileges assignment.
- Metadata assignment.

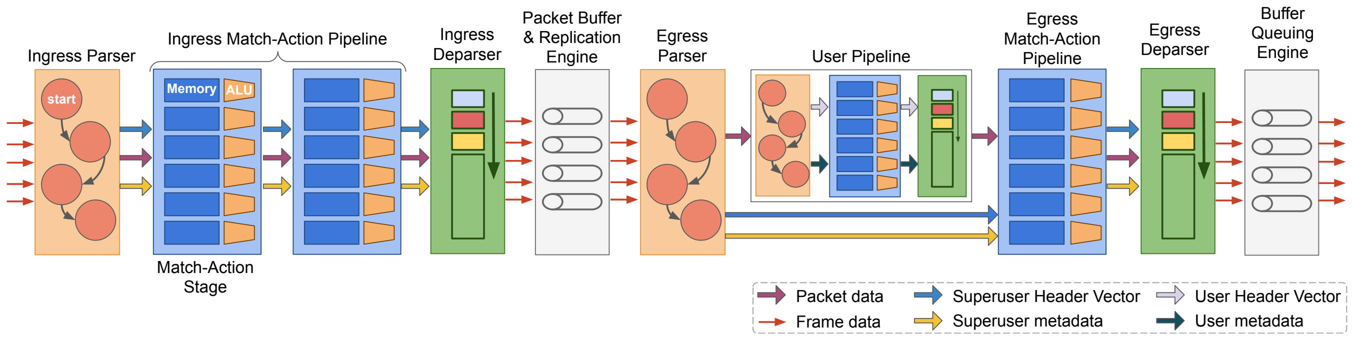

Figure 1: The Architecture of MTPSA

MTPSA supports both ingress and egress Superuser pipelines, similar to PSA's architecture.

Packet pre-processing is used for the assignment of packets to contexts (user programs). Any incoming packet is associated with a user identifier which may vary, e.g., based on ingress port, 5-tuple, etc. The data required for the assignment is extracted as part of the pre-processing, and is included in the output metadata of the ingress Superuser pipeline. In addition, the ingress Superuser pipeline might provide other packet pre-processing services, such as access control lists, or handling encapsulation headers (e.g., outer IP/UDP/VxLAN) in a manner transparent to a user program. This approach enables adding user ID and other information to a packet for multi-stage processing by other devices within the network. The Superuser egress pipeline allows encapsulation headers to be removed before a packet is sent to a destination host (e.g., user's host or VM).

5.2 User Pipeline

User programs are compiled as configuration contexts to user pipelines. These pipelines can be either physical (§ 6) or virtual (§ 8). MTPSA modularizes the PDP pipeline, allowing user programs to be compiled and tested independently. The MTPSA-user pipeline architecture, similar to the P4→NetFPGA SimpleSumeSwitch architecture [8], consists of a parser, a pipeline of match-action tables, and a deparser. In the current implementation (§ 6) the user pipeline is called after the Superuser egress parser. While a user program is basically no different to standard P4 programs, MTPSA guarantees that a user program i) can observe and operate only on its own packets and ii) the resulting actions on packets will match the context's privileges. For example, the privileges can restrict the number of circulations of a packet, or the allowed increase of header size (e.g., turning 64B packets to 256B packets).

5.3 Execution Model

Consider a network operator providing virtualization services to users. The operator uses traffic encapsulation to control the traffic and route packets through the networks, but users can observe only their decapsulated traffic at the edges of the network. Similarly, if users want to run their traffic through specific data-plane programs, these programs should observe only the decapsulated traffic belonging to the specific users. Users should not have visibility to the way the network is managed, meaning encapsulated traffic or other users' traffic.

In MTPSA, the network operator provides processing resources on the switch as a service. Traffic from a user's virtual machine (VM) is encapsulated (using VxLAN headers) and forwarded within the operator's network to the switch. In the switch, the user's program is identified by the VxLAN header, and the packet is decapsulated. The user's programs observes the packet as it was sent from the user's VM, and is processed according to the user's program.

This scenario accounts for some of the design decisions in MTPSA. First, security isolation, as all encapsulated traffic is handled only in the Superuser pipeline, and user programs are exposed only to their own traffic, and not to Superuser or other users' traffic. Second, performance isolation, as the separation of user programs means that packets through one user's program do not interfere with traffic through other users programs. Furthermore, the Superuser pipeline protects against malicious use outside a program (e.g., blocking recirculation). Last, MTPSA provides resource isolation, as users can only access resources, such as tables, assigned to them, and can not take advantage of other resources. Typically, users will be stopped from exceeding allocated resources in the compilation stage, and stopped from accessing other resources during execution.

6 IMPLEMENTATION

We implement MTPSA over two existing targets: PSA [10] over BMv2 as a software target, and P4→NetFPGA [8] as a hardware target. Our implementation is open-source and available at [14]. In this section, we discuss the implementation details of our proof-of-concept, but note that MTPSA can be extended and adapted beyond our local design choices.

Each MTPSA target supports switch configuration contexts that enable multiple P4 user programs to be loaded on a single device, and to assign at run-time packets to user programs. Switch contexts are parallel pipelines that can be configured independently. Thus, a packet will not be sequentially processed by two user programs (performance isolation).

Each user context (shown in Figure 2) has a unique identifier (*user_id*). A *control context* with user_id value of 0 is used to handle packets that are not associated with any user program. For example, when no user programs have been loaded or an unexpected packet has been received. The Superuser program can specify, in the ingress pipeline using the standard metadata bus, a user_id value indicating a user program to process the packet. The user_id metadata field is initialized to the value of 0, therefore, if not explicitly assigned, the packet is handled by the control context.

In addition, each context, in the BMv2 MTPSA target, has a pool of PHV objects. Using this method, the switch target can reuse PHV objects in compliance with the security isolation requirement.

Superuser Program The Superuser P4 program implements both ingress and egress pipelines. The ingress parser extracts header data specified by the network administrator, and assigns the user identifier as well as roles and privileges. MTPSA's Superuser ingress pipeline supports adding packet encapsulation headers (e.g., Geneve, VxLAN, GRE, VLAN, etc.) that include the user ID and additional information, making it available to other devices in the network. This is used to evaluate MTPSA within a virtualized network of multiple devices (§7.2).

There are two major differences between MTPSA Superuser pipelines implemented over PSA and NetFPGA. On PSA, the Superuser egress parser is applied prior to the user pipeline and allows to extract packet headers (e.g., VxLAN) specified by the network administrator, making them inaccessible to user programs. Thus, when using VxLAN instead of VLAN, user programs can process arbitrary packet headers, including Ethernet, while the Superuser pipeline can apply encapsulation the packet in a manner transparent to the user programs.

On the NetFPGA platform, as a property of SDNet, the Superuser pipelines are implemented as two stand-alone P4 programs. In contrast to the implementation over PSA, the Superuser egress parser is applied after the user pipeline, as the pipeline can not be split and the SDNet compiler is closed source (§6.1).

User Programs User P4 programs include an architecture description file (`mtpsa_user.p4`) that defines the data types and constants available to a program in the user pipeline. All user programs are implemented, compiled, and tested independent from one another, and from the Superuser program. The decoupling of user programs enables users to apply changes to their pipeline without affecting other users. This method of abstraction enables network administrators to independently deploy in-network user applications and better control the flow of packets in the network. The software switch implementation supports loading user programs either when the target is initialized, or at run-time. On NetFPGA, user programs can currently be loaded only during initialization.

Runtime Control When an MTPSA switch is up and running, a network administrator is able to control its behavior from a control plane. To this end, the control plane uses an auto-generated Thrift APIs to carry out system and user requests. The MTPSA BMv2 target allows a network operator to select the switch context to which a user program should be loaded, or apply other operations such as updating table entries within specific contexts.

6.1 MTPSA Compiler

$P4C_{16}$ reference compiler does not naturally support MTPSA. We have extended the compiler to support MTPSA by introducing a compiler back-end, *mtpsa_switch*, and two standard files: *mtpsa.p4* - utilized by a Superuser program; and *mtpsa_user.p4* used with User programs.

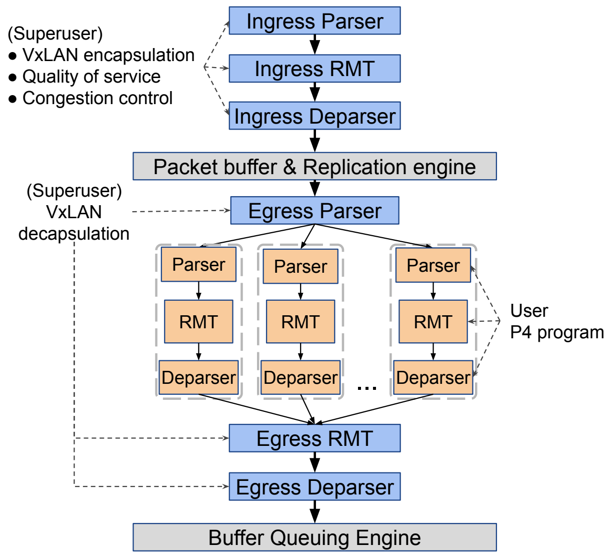

Figure 2: Block diagram of the MTPSA P4 architecture.

The compiler consists of a single executable *p4c-bm2-mtpsa*. It enables independent compiling of P4 programs either to a user pipeline or to the Superuser pipeline, using an additional `--user` command-line option, which enables *user mode*.

Assigning user identifier and permissions to user pipelines do not require changes to the compiler or target. They are embedded in the control metadata bus, and the associated policy (e.g., disable access) is defined with the Superuser P4 program. In particular, the Superuser ingress output metadata includes *user_id* and *user_permissions* fields that instruct the target which user program to apply and its execution permissions.

The NetFPGA implementation does not require changes to the P4SDnet compiler. The Superuser pipeline is defined as two P4 programs – suIngress and suEgress, with 128-bits width TUSER bus, from which the first 40-bits are accessible in user pipelines.

7 EVALUATION

7.1 Experimental Setup

The functional evaluation of MTPSA is conducted both on the software and on the hardware targets. For the software target, MTPSA is compatible with the latest P4C and behavioral model codebase [1]. Mininet 2.3.0d6 is used for emulation, running on 4-core Intel i5-4200U, over Fedora Linux 5.6.11-200.fc31.x86_64. MTPSA is evaluated both as a stand alone switch, and within a network with eight nodes and three MTPSA switches (two leaf switches and one spine switch).

The hardware implementation of MTPSA architecture is evaluated on the NetFPGA SUME platform [22] using Xilinx Vivado 2018.2 and SDNet 2018.2, running on Ubuntu 16.04. OSNT, an open-source network tester, [1] is used for traffic generation and capture. OSNT is connected to the MTPSA device using $4 \times 10Gbps$ links. Latency and throughput are evaluated using a range of packet sizes,

[1]As of September 8th, 2020

	Ref. Switch	$MTPSA_0$	$MTPSA_2$	$MTPSA_3$	$MTPSA_4$	$MTPSA_8$
Logic Util.	29.04%	39.04%	53.89%	60.08%	67.78%	86.51%
Memory Util.	32.69%	40.34%	52.93%	59.22%	65.51%	83.33%
Throughput	40Gbps	40Gbps	40Gbps	40Gbps	40Gbps	40Gbps
Latency (64B)	1.7μs	2.52μs	3.23μs	3.24μs	3.24μs	3.33μs

Table 1: Resource utilization and performance of Reference Learning Switch and MTPSA on P4→NetFPGA. $MTPSA_N$ represents an MTPSA switch running N user programs.

from 64B to 1518B. NetFPGA is a store and forward device, therefore latency increases with packet size.

Several programs are used in the evaluation. Reference Switch is the reference learning switch project of P4→NetFP-GA [8]. $MTPSA_0$ is an MTPSA pipeline implementation with no user programs. This simplified version of MTPSA is similar to the PSA architecture. It consists of ingress and egress pipelines. $MTPSA_{2,3,4}$ are MTPSA implementations with two, three and four user programs, respectively. TCP port number is used for user_id assignment. For resource evaluation, user programs implement layer 2 forwarding, and Superuser ingress and egress pipelines implement the same packet processing. This allows a proper overheads comparison. More programs are discussed as part of the functional evaluation.

7.2 Evaluation Results

Our evaluation spans a functional evaluation, resource consumption and performance. Our goal is to demonstrate that MTPSA is feasible and scalable, while not degrading performance. Our results are summarized in Table 1.

Functionality We implement a range of Superuser and user programs to evaluate MTPSA. The Superuser pipeline typically supports Ethernet, IPv4 and IPv6, TCP, UDP and ICMP headers. In two more examples, exploring a multi-switch topology, the Superuser pipeline encapsulates incoming packets with VxLAN or with VLAN headers that include the user ID. User programs include forwarding with different rules, as used in the performance evaluation, layer 4 load balancing, and others.

We validate our claims for isolation of users, e.g., packets of one user program cannot access other program's resources, and packets with invalid user_id are dropped. We use externs as an example for using privileges, where one user program is allowed to access a counter extern, while a second program has no permission, and verify that this is enforced.

We explore adding 'max_recirculations' and 'max_resubmissions' fields to PSA's standard metadata or as user-defined metadata, which is currently not supported[2]. These metadata fields can be used to prevent denial-of-service attacks by malicious user programs, limiting the amplification effect of each context.

Resource Overhead We focus on the NetFPGA implementation for our resource overhead evaluation. MTPSA has an obvious resource overhead, as due to the nature of the SDNet compiler the Superuser pipelines and user contexts need to be implemented as independent modules. This may not always be the case (§8). We find that each Superuser program and user context take approximately 4% logic resources and 6% memory resources. This indicates the scalability of the solution, and in $MTPSA_8$ ten pipelines are implemented. Complex programs can obviously require more resources, as with any P4 program.

Latency As the results show, while the basic latency of MTPSA is higher than the reference switch, due to the additional *Hypervisor layer* - the Superuser program, as the number of user programs increases from two to eight, the latency almost does not change. Latency that is independent of the number of contexts is one of the benefits of MTPSA compared with some previous works [21].

Throughput MTPSA supports a full line rate across a range of packet sizes, for both TCP and UDP. We evaluate, as measured by sending a billion packets from each port simultaneously, with both equal and different share of traffic to each context. For example, we measure 40Gbps throughput[3] with 10Gbps going through each of $MTPSA_4$ four contexts, as well as with 20Gbps through two contexts and 40Gbps through a single context.

8 DISCUSSION

Feasibility Our implementations over PSA and NetFPGA show that MTPSA is not only feasible, but can also be used within existing environments using common infrastructure, without requiring bespoke new solutions. These implementations are contributed to the P4 community.

Software vs Hardware Targets MTPSA originates from PSA, which currently has a software implementation. As such, some of the design decisions are more appropriate for software than hardware, such as being able to easily increase the number of user pipelines. Our implementation of MTPSA on NetFPGA shows the applicability of the approach to hardware targets. However, different hardware targets will require different design decisions, as was the case with PISA-based [2] implementations.

Applications Our implementation enables any P4 program that runs on a single pipeline to run within MTPSA. This includes, for example, any P4 program prototyped on NetFPGA (e.g., [15, 17, 19]). MTPSA prototypes use buffer queuing engine at the end of the pipeline, allowing to place user programs in the egress pipeline without restricting certain types of packet modifications (e.g., output port change). Programs that benefit from the combination of ingress and egress pipelines, or from a packet buffer and replication engine between pipelines, e.g., for multicast or traffic management (e.g., [9]), are less suitable. Since user packets going through a shared memory create additional security risks, MTPSA's model mitigates such concerns.

Multi-core targets The prototype implementations of MTPSA are based on a single-core (pipe) target model. Some targets, such as

[2]In the current implementation of PSA user-defined metadata is not preserved when recirculating or resubmitting packets, which is an implementation decision not part of the PSA specification.

[3]For TCP traffic of 64B-128B, MTPSA currently achieves 98.46%-99.96% throughput, due to a known bug.

Tofino [5], use multiple parallel cores, with shared memory used as switching fabric. Consequently, such targets may implement user-pipelines in the ingress pipeline in addition or instead of the egress pipeline. Such an implementation would still require pre-processing in the Superuser ingress pipeline before the user pipeline. Supporting user pipelines only at the Egress of multi-core targets means that some routing decisions, such as the assignment to an egress core, need to be taken in the Ingress pipeline by the Superuser program.

Virtual pipelines While MTPSA's user pipelines were implemented using "physical" pipelines, due to the nature of the PSA and SDNet compilers, it is also possible to implement "virtual" pipelines. In devices that support simultaneous lookups in each stage [5], the assigned privileges and user identifier can be used to force packet processing only within a user program's assigned resources. The support of such virtual pipelines is future work.

9 RELATED WORK

The virtualization of PDP has been explored by several previous works. P4Visor [21] and P4Bricks [13] merge multiple P4 programs at source level into a single program, focusing on front-end compiler optimization to maximize the shared resources between different P4 programs, and don't focus on isolation.

HyPer4 [7] and HyperVDP[20] virtualize the data plane by running a P4 program emulating the behavior of multiple P4 programs through packet resubmission and recirculation. However, these approaches require significantly more hardware resources compared to native P4 programs.

Dejavu [18] has a similar programming model that leverages recirculation to connect several network functions in a single switch pipeline. It generates a single multi-pipeline P4 program that can be compiled and loaded onto the physical pipelines. In contrast, MTPSA enables user and Superuser programs to be developed, compiled and configured on a switch independently.

Wang *et al.* [16] focused on resource efficiency and isolation in terms of access control as the main requirements for multi-tenancy. Their solution does not attend to security isolation, and lacks the concept of roles. User programs are merged into a single program, thus exposed to more performance and security vulnerabilities. Unlike MTPSA, in [16] recirculation is completely forbidden, therefore attending to one potential weakness, but also blocking a required functionality.

The most closely related work is P4VBox [12], which explores a conceptual hardware architecture for P4-based switches support virtualization. P4VBox is limited to FPGA hardware and processing packets with VLAN (802.1Q) tags to associate frames with virtual switch instances. Our solution is general, allowing beyond resource and performance isolation also security isolation, through the Superuser P4 program and user privileges. MTPSA enables any policy or rule specified by a network operator, and supports both software and hardware targets.

10 CONCLUSION

In this paper we have introduced MTPSA, a multi-tenant architecture for Portable Switch Architecture. MTPSA supports resource, performance and security isolation of user programs, enabling network administrators better control over user applications. Our implementation of MTPSA over the PSA compiler and NetFPGA shows that the solution is not only feasible, but also scalable with minimal effect on resource and performance.

MTPSA is open-source and available at [14].

REFERENCES

[1] G. Antichi et al. 2014. OSNT: open source network tester. *IEEE Network* 28, 5 (2014), 6–12.

[2] P. Bosshart et al. 2014. P4: Programming protocol-independent packet processors. *ACM SIGCOMM CCR* 44, 3 (2014), 87–95.

[3] X. Chen et al. 2019. P4SC: Towards High-Performance Service Function Chain Implementation on the P4-Capable Device. In *2019 IFIP/IEEE IM.*

[4] M. H. Dumitru et al. 2020. Can we exploit buggy P4 programs?. In *SoSR.* 62–68.

[5] V. Gurevich. 2017. Barefoot Networks, Programmable Data Plane at Terabit Speeds. In *P4 Developer Day.*

[6] S. Han et al. 2020. Virtualization in Programmable Data Plane: A Survey and Open Challenges. *IEEE OJ-COMS* (2020).

[7] D. Hancock et al. 2016. HyPer4: Using P4 to virtualize the programmable data plane. In *ACM CoNEXT.* 35–49.

[8] S. Ibanez et al. 2019. The P4-> NetFPGA workflow for line-rate packet processing. In *ACM/SIGDA FPGA.* 1–9.

[9] R. Kundel et al. 2018. P4-CoDel: Active queue management in programmable data planes. In *NFV-SDN.* 1–4.

[10] P4 Language Consortium. 2018. P4_16 Portable Switch Architecture (PSA). (2018).

[11] R. Ross et al. 2019. *Developing Cyber Resilient Systems: A Systems Security Engineering Approach.* Technical Report. NIST.

[12] M. Saquetti et al. 2019. P4VBox: Enabling P4-based switch virtualization. *IEEE Communications Letters* 24, 1 (2019), 146–149.

[13] H. Soni et al. 2018. P4Bricks: Enabling multiprocessing using Linker-based network data plane architecture. (2018).

[14] R. Stoyanov et al. 2020. Multi-Tenant Portable Switch Architecture, Repository. https://github.com/mtpsa.

[15] Y. Tokusashi et al. 2019. The Case For In-Network Computing On Demand *(EuroSys '19).* Article 21, 16 pages.

[16] T. Wang et al. 2020. Multitenancy for Fast and Programmable Networks in the Cloud. In *USENIX HotCloud.*

[17] J. Woodruff et al. 2019. P4DNS: In-Network DNS. In *EuroP4.* 1–6.

[18] D. Wu et al. 2019. Accelerated Service Chaining on a Single Switch ASIC. In *HotNets.* 141–149.

[19] Z. Xiong et al. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. In *HotNets.* 25–33.

[20] C. Zhang et al. 2019. HyperVDP: High-performance virtualization of the programmable data plane. *IEEE JSAC* 37, 3 (2019), 556–569.

[21] P. Zheng et al. 2020. Building and Testing Modular Programs for Programmable Data Planes. *IEEE JSAC* 38, 7 (2020), 1432–1447.

[22] N. Zilberman et al. 2014. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro* 34, 5 (2014), 32–41.