MPVisor: A Modular Programmable Data Plane Hypervisor

Cheng Zhang, Jun Bi, Yu Zhou, Abdul Basit Dogar, Jianping Wu
Tsinghua University
zhang-cheng13@mails.tsinghua.edu.cn, junbi@tsinghua.edu.cn, {y-zhou16, bas15}@mails.tsinghua.edu.cn, jianpingwu@tsinghua.edu.cn

ABSTRACT

P4 is a domain-specific language designed to define the behavior of a data plane. It facilitates offloading simple Network Functions (NFs) onto a programmable data plane in order to reduce latency and improve device efficiency. However, these NFs enforced by different operators may be contradicted. Besides, dynamic management of NFs without interrupting the operation of a data plane is almost impossible. And operation expenditure (Opex) inevitably increases due to the need of expressing richer polices and resolving policy inconsistencies. Therefore, in this poster we propose MPVisor, a high performance hypervisor of P4 specific data plane with modular programmability, to enable isolation and dynamic management of NF instances without data plane interruption. We implement MPVisor and evaluate several benchmarks by comparing with equivalent baseline applications implemented in Hyper4 and native P4 switch.

CCS Concepts

•**Networks** → Programmable networks; •**Software and its engineering** → System modeling languages;

Keywords

Modular programming model; P4; data plane hypervisor

1. INTRODUCTION

Recent researches of offloading simple network functions (NFs) onto the programmable data plane have shown an advantage in saving more CPU power for user applications while benefiting from high performance of the hardware network devices especially in Network Function Virtualization environment. However, as more and more NFs could be shifted to programmable data planes on demand, due to the self-constraint of programmable devices such as FPGA, it is hardly possible to manage NFs dynamically without interrupting the data plane (often takes minutes to configure a new logic). Furthermore, NFs enforced by different operators need isolated environments in case of conflicts. And operation expenditure (Opex), e.g. deploying or debugging, will inevitably increase in more complex scenarios.

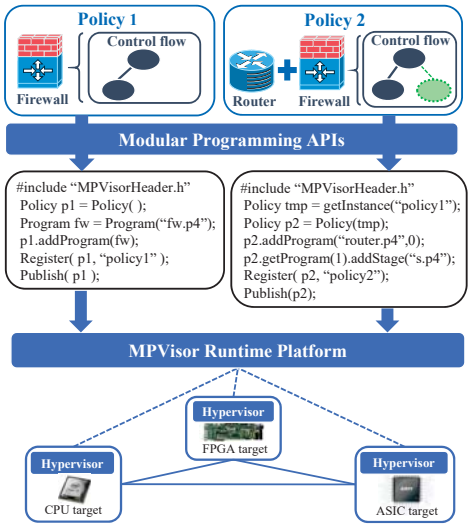

Figure 1: Architecture of MPVisor.

Prior work such as ClickNP [1] presents a modular programmable data plane with high performance. However, due to a lack of design for virtualization, it cannot enforce isolation and uninterrupted management of NFs. Hyper4 [2] proposes a data plane hypervisor with partial virtualization. This hypervisor is based on P4 language [3]. However, it suffers from a severe loss of performance, and a lack of programming models for developing and debugging of NFs.

Inspired by aforementioned works, we take a step forward into designing MPVisor as shown in Figure 1. MPVisor is composed of three components: (i) a high performance hypervisor of P4 specific data plane providing virtualization and isolation; (ii) a runtime platform serving to compile and manage NF instances without data plane interruption; and (iii) a suite of Modular Programming API (MP-API) enabling flexible deriving

This research is supported by the National Natural Science Foundation of China (No.61472213 and No.61502267). Jun Bi is the corresponding author.

SOSR '17, April 3–4, 2017, Santa Clara, CA, USA
© 2017 ACM. ISBN 978-1-4503-4947-5/17/04…$15.00
DOI: http://dx.doi.org/10.1145/3050220.3060600

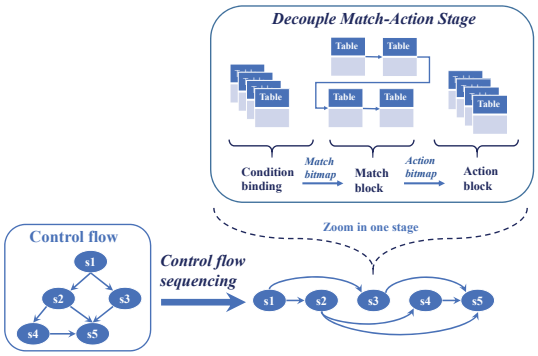

Figure 2: Control flow sequencing and decoupling.

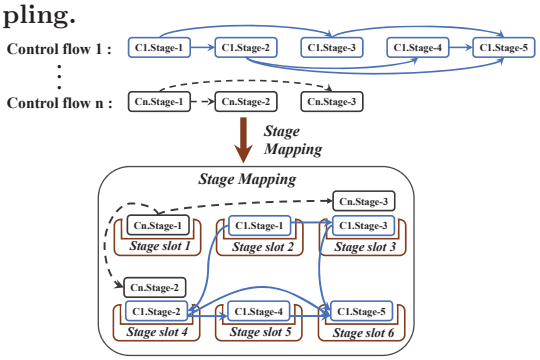

Figure 3: Dynamic stage mapping.

of published NFs. We will briefly describe these components in the following.

2. DESIGN OF MPVISOR

Data plane hypervisor involves several novel techniques to achieve high performance and full virtualization of P4 language elements. And as shown in Figure 2, we use the topology ordering algorithm to uniformly sequence a control flow in a P4 program. Then we decouple one match-action stage (we will use stage for short) into a unified processing pattern to interpret complex language elements such as *if-else*. After that, inspired by virtual memory management in the operating system, we abstract a concept of stage slot as shown in Figure 3, which can hold an infinite number of stages. So that any sequence of stages can be dynamically mapped into a limited number of slots. Meanwhile, stage branching is achieved by setting the address of the slot where the next stage resides in.

Runtime platform compiles code and dynamically manages NFs by maintaining status, including stage mapping, policy id, program id and etc. So that operators can easily enforce isolation between different policy instances, as well as instantiate a NF from published NFs. Additionally, through the runtime platform, stage slots can be pooled and managed as a unified resource which is as same as CPU in a virtualized data center.

MP-API provides operators with three abstracted models to compose NFs. One is a stage, which functionally equals to a match-action stage; another is a program, which denotes an individual NF, such as a *firewall* or a *l2-switch*; last is a policy, which denotes a chain of programs, as *policy2* is shown in Figure 1. All these models can be flexibly derived, modified and published as the code snippet shown in Figure 1. Our experience also confirms that MP-API greatly improves the code reuse and simplifies the composing of new NFs.

Program	Native P4	Hyper4	MPVisor
L2 switch	2	13	5
Firewall	3	22	8
Router	4	28	16
Arp Proxy	4	48	10

Table 1: Table usage in runtime.

Program	Latency ratio		Bandwidth ratio	
	Hyper4	MPVisor	Hyper4	MPVisor
L2 switch	3.41 : 1	1.71 : 1	0.16 : 1	0.43 : 1
Firewall	4.71 : 1	2.21 : 1	0.11 : 1	0.49 : 1

Table 2: The respective ratio of Hyper4 and MPVisor to native P4 switch in terms of latency and bandwidth.

3. EVALUATION AND DISCUSSION

We implement a prototype of MPVisor and evaluate it through comparative tests with native P4 switch and Hyper4 in terms of 4 benchmarks. Our tests are executed on the x86 server with 8 E5-2637 3.50Ghz cores and 64GB memory.

Line of code: Through MPVisor, operators can maximally benefit from code reuse to instantiate a NF from published NFs usually in less than 10 LoC, while native P4 switch and Hyper4 need the complete code of P4 programs, which is much more than MPVisor.

Table declaration: In order to support programs with no more than 5 stages, Hyper4 has to declare more than 400 tables. On average, MPVisor saves 5x to 8x table declaration comparing with Hyper4.

Table in runtime: Due to a well-designed structure, MPVisor, as shown in Table 1, saves 2x to 4x of tables comparing with Hyper4, while averagely takes 3x of native P4 switch.

Performance impact: As shown in Table 2, MPVisor generally has a 2x to 4x performance advance comparing with Hyper4. We attribute this superiority to a much less use of resources, including tables and control variables, in runtime and a much more simple inner control logic.

4. REFERENCES

[1] Bojie Li, Kun Tan, and et al. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *SIGCOMM '16*, pages 1–14, 2016.

[2] David Hancock and Jacobus van der Merwe. Hyper4: Using p4 to virtualize the programmable data plane. In *CoNEXT '16*, pages 35–49, 2016.

[3] Pat Bosshart, Dan Daly, and et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.