Hard Virtualization of P4-based switches with VirtP4

Mateus Saquetti, Guilherme Bueno, Weverton Cordeiro, José Rodrigo Azambuja
Federal University of Rio Grande do Sul (UFRGS) - PGMICRO - PPGC
Porto Alegre, RS, Brazil
first.last@inf.ufrgs.br

ABSTRACT

We present VirtP4, a conceptual architecture for hard (type-1) virtualization of P4-based switch programs in a NetFPGA SUME board. In contrast to existing (type-2) solutions like Hyper4, HyperVDP, and P4Visor, our design does not require access to switch source code (either for merging or custom compilation). We thus preserve the intellectual property of switch programs, while providing near line-rate performance for virtual switch instances.

CCS CONCEPTS

• **Networks** → **Programmable networks**; • **Hardware** → **Reconfigurable logic and FPGAs**.

KEYWORDS

Programmable Forwarding Planes; P4; Virtualization; NetFPGA-SUME; Simple Sume Switch

ACM Reference Format:

Mateus Saquetti, Guilherme Bueno, Weverton Cordeiro, José Rodrigo Azambuja. 2019. Hard Virtualization of P4-based switches with VirtP4. In *SIGCOMM '19: ACM SIGCOMM 2019 Conference, August 19–23, 2019, Beijing, China*. ACM, New York, NY, USA, 2 pages. <https://doi.org/10.1145/3342280.3342314>

1 INTRODUCTION

In this paper, we make a case for hard (type-1) virtualization of P4-based switches. Recent investigations like HyPer4 [2], HyperV [5], and P4Visor [6] brought soft (type-2) virtualization to the data plane, by enabling emulation of several network contexts and/or composing several functions through a single running P4 [1] program. In contrast to previous work, we posit that deploying virtual switch instances should not require access to any source code (either for merging [2, 6] or custom compilation [5]). Instead, virtualization solutions for the data plane should (i) enable users to deploy virtual instances that operate independently (just like in any COTS virtualization solution), and (i) enable vendors (including novel players) to distribute compiled switch byte-code to users while protecting their intellectual property.

Delivering such a solution is challenging. In addition to addressing OS virtualization aspects like resource mapping and isolation, enabling hard virtualization of P4 switches requires addressing

the following issues: (i) How to decouple virtual switch instances from the virtualization box? (ii) How to steer frames from/to virtual switch instances? In our previous work [4], we described a conceptual architecture for virtualization of P4 switches. In this paper, we move a step further, by addressing the challenges enumerated above and reporting our experience with the development of VirtP4 as a fork of the canonical NetFPGA reference design [3]. We also discuss our deployment of VirtP4 in a NetFPGA-SUME board. Unlike related work [2, 5, 6], we approach these issues with VirtP4 while preserving the intellectual property of switches.

2 VIRT P4

Our goal is to think virtualization of P4-based switches using commodity hardware conceptually and identify the main technical challenges towards an actual deployment using programmable switch hardware. To achieve it, we built a proof-of-concept prototype for a NetFPGA-SUME board.

To foster acceptance, one main requirement is that the same switch byte-code that runs on top of commodity hardware must also run on top of the virtualization box. To decouple virtual switch instances from our virtualization box, we built VirtP4 following the canonical NetFPGA reference design [3], refactoring its architecture to support parallel execution of P4-based switch instances. In Figure 1 we present an overview of VirtP4 and the canonical model.

In summary, our fork replaces the *Simple Sume Switch* block with two VirtP4 modules: Input P4 Interface (IPI) and Output P4 Interface (OPI), which are responsible for steering frames from/to virtual switch instances. The former is responsible for dispatching incoming data frames (received from the input arbiter) to the correct virtual switch instance. In our current stage of development, VirtP4 uses 32 bit VLAN tagging (802.1Q), and a match+action table, hosted in the controller, to determine which switch instance must receive a given data frame. In case some data frame contains a tag not found in the table, that means the switch instance responsible for that tag is not deployed, and the frame is silently discarded. The latter block (OPI) is responsible for delivering data frames to the correct physical (output) port. This module does not discard any frames.

In VirtP4, one may write and compile switch code independently (which means one may retain ownership over one's code). To operate on top of VirtP4, the switch must be capable of parsing the VLAN tagging that the IPI module uses to decide frame dispatching. Once in operation within VirtP4, each deployed virtual switch instance runs within its context, i.e., their control flow (along with metadata and variable scope) operate without sharing information or resources with other instances running in parallel. Note that, similarly to [3], VirtP4 switch instances must also be implemented following the SimpleSumeSwitch P4 architecture.

A network operator can interact with deployed virtual switch instances using the same configuration script generated in the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGCOMM '19, August 19–23, 2019, Beijing, China

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6886-5/19/08...\$15.00

<https://doi.org/10.1145/3342280.3342314>

Figure 1: A comparison between VirtP4 and the canonical NetFPGA design.

Table 1: Area occupation after synthesis

Module	LUT		FF	
	#	%	#	%
4x10Gb Ports and DMA	24066	5.56%	31946	3.69%
Input Arbiter	1720	0.40%	3765	0.43%
Input P4 Interface	684	0.16%	1017	0.12%
Output P4 Interface	2953	0.68%	5404	0.62%
Output Queues	2234	0.52%	4857	0.56%
Controller	13626	3.15%	21935	2.53%
L2 switch (instance 0)	25124	5.80%	37357	4.31%
Router (instance 1)	45530	10.51%	77953	9.00%

compilation process. If necessary, switch instances may exchange packets between each other using the DMA port to this end; whenever some switch instance outputs a packet that must be sent to another instance, the packet is sent to the DMA, which in this case behaves as a loopback, sending the frame back to the input arbiter.

3 PRELIMINARY EVALUATION

Our VirtP4 prototype is a fork of the P4-NetFPGA code, with a total diff of around 3,000 lines of code¹. For experimentation, we used a NetFPGA-SUME board, equipped with a Virtex-7 690T FPGA, four SFP+ 10Gb/s ports and five independent high-speed memory banks.

The generation of switch instances from P4 code follows the same process described by Ibanez *et al.* [3]. In our experiment, we considered a simple L2 switch and router as virtual instances. We depict in Table 1 area occupation for each instance and the other modules deployed in the FPGA, including those specific to VirtP4.

From a controlled experiment, we assessed that the IPI was able to route incoming data frames to their corresponding virtual switch instances, and that the OPI was able to direct the output frame to the respective physical port. We intend to work with NETRONOME Dual-port 10GbE PCI-e programmable network cards (ISA-4000-10-2-2) to assess the performance of deployed instances at the combined 40GbE NetFPGA-SUME line rate.

¹BitBucket repo: <https://bitbucket.org/p4virtualization/virtp4/>

4 SUMMARY AND FUTURE WORK

VirtP4 is a conceptual solution and proof-of-concept implementation for hard virtualization of P4-based switches. One current limitation of VirtP4 is that the match+action table queried by the IPI/OPI modules are hardcoded. It means that changes to the VLAN tagging scheme used (or the tagging configuration populated in the table) require rebuilding the entire project (and reconfiguring the FPGA board from scratch). We are working on a partial reconfiguration of the FPGA board, to enable reconfiguration of the VLAN tagging and hot-swapping of virtual switch instances, without requiring the synthesis of the entire project.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their valuable feedback. This work has been supported in part by funding from National Science Foundation (NSF), Brazilian Education and Research Network (RNP) (P4Sec 8278), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001, CNPq, and FAPERGS.

REFERENCES

- [1] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-independent Packet Processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95.
- [2] David Hancock and Jacobus Van Der Merwe. 2016. Hyper4: Using p4 to virtualize the programmable data plane. In *12th Int'l on Conference on emerging Networking EXperiments and Technologies*. ACM, 35–49.
- [3] Stephen Ibanez, Gordon Brebner, Nick McKeown, and Noa Zilberman. 2019. The P4-NetFPGA Workflow for Line-Rate Packet Processing. In *ACM/SIGDA Int'l Symposium on Field-Programmable Gate Arrays (FPGA '19)*. ACM, New York, NY, USA, 1–9.
- [4] Mateus Saquetti, Guilherme Bueno, Weverton Cordeiro, and José Rodrigo Azambuja. 2019. VirtP4: An Architecture for P4 Virtualization. In *2019 RAW workshop - Short Paper Introductions (RAW '19)*. IEEE, 1–4.
- [5] Cheng Zhang, Jun Bi, Yu Zhou, Abdul Basit Dogar, and Jianping Wu. 2017. Hyperv: A high performance hypervisor for virtualization of the programmable data plane. In *Computer Communication and Networks (ICCCN), 2017 26th Int'l Conference on*. IEEE, 1–9.
- [6] Peng Zheng, Theophilus Benson, and Chengchen Hu. 2018. P4Visor: Lightweight Virtualization and Composition Primitives for Building and Testing Modular Programs. In *14th Int'l Conference on Emerging Networking EXperiments and Technologies (CoNEXT '18)*. ACM, New York, NY, USA, 98–111.