A Terabit Hybrid FPGA-ASIC Platform for Switch Virtualization

Mateus Saquetti*, Raphael M. Brum*, Bruno Zatt†, Samuel Pagliarini‡, Weverton Cordeiro*, Jose R. Azambuja*

*Federal University of Rio Grande do Sul (UFRGS) - Institute of Informatics - School of Engineering
{mateus.tirone, brum, weverton.cordeiro, azambuja}@ufrgs.br
†Federal University of Pelotas (UFPEL) - Center for Technological Development
zatt@inf.ufpel.edu.br
‡Tallinn University of Technology (TalTech) - Department of Computer Systems
samuel.pagliarini@taltech.ee

Abstract—The roll-out of technologies like 5G and the need for multi-terabit bandwidth in backbone networks requires networking companies to make significant investments to keep up with growing service demands. For lower capital expenditure and faster time-to-market, companies can resort to anything-as-a-service providers to lease virtual resources. Nevertheless, existing virtualization technologies are still lagging behind next-generation networks' requirements. This paper breaks the terabit barrier by introducing a hybrid FPGA-ASIC architecture to virtualize programmable forwarding planes. In contrast to existing solutions, our architecture involves an ASIC that multiplexes network flows between programmable virtual switches running in an FPGA capable of full and partial reconfiguration, enabling virtual switch hot-swapping. Our evaluation shows the feasibility of a switch virtualization architecture capable of achieving a combined throughput of 3.2 Tbps by having up to 26 virtual switch instances in parallel with low resource occupation overhead.

Index Terms—5G, FPGA-ASIC Platform, Programmable Networks, Switch Virtualization, Terabit Communications

I. INTRODUCTION

The networking community has long taken advantage of virtualization to realize a multitude of services dubbed XaaS (*anything-as-a-service*) [1], [2]. Popular examples include *infrastructure*, *platform*, and *software* as a service (IaaS, PaaS, and SaaS, respectively). In this context, virtualization has been used to leverage abstractions of several networking components (e.g. links [1], [3]), forwarding elements (e.g., switches) [4]–[6], servers [7], management units [8]–[10], etc.

With the emergence of network programmability [11]–[13] and the possibility to redefine the behavior of forwarding elements through home-brewed software [12], [14]–[16], researchers have been investigating how to leverage abstractions of virtual programmable forwarding elements [17]–[22]. Their motivations for programmable virtual forwarding elements are manifold. For example, with the roll-out of 5G and the need for multi-terabit bandwidth in backbone networks [23], [24], telecommunication companies interested in providing 5G coverage to its users may avoid significant capital expenditure with the installation of radio-base stations and equipment purchase by leasing them from an infrastructure provider [20]. In this case, the infrastructure provider may create virtual instances of programmable forwarding elements to its customers. Each customer can then redefine the virtual forwarding element's behavior, using one's set of networking protocols to enable a seamless integration into the customer's network.

To realize the full potential of virtualization for programmable forwarding planes and establish a novel class of XaaS called *programmable switches-as-a-service* [20], the research community has been addressing several challenges in the path of a fully-fledged virtualization solution. In prior investigations, researchers have devised solutions to (i) accommodate various instances of virtual switches within a single physical programmable switch hardware, either using switch emulation or switch program composition [17]–[19], (ii) enable virtual switch isolation and hot-swapping of virtual switch instances [21], and (iii) provide proper abstractions of management channels for tenant-independent switch operation [25].

Despite the progress achieved, existing solutions failed to deliver virtual programmable switches with similar performances to bare metal ones. For example, solutions based on emulation of virtual switches or switch program composition require additional forwarding tables to implement flow steering between virtual switch instances, increasing memory overhead, and imposing severe penalties to packet latency and virtual switch throughput [22]. On the other hand, solutions capable of providing virtual switch isolation have not approached an architectural design capable of achieving line-rate speeds while ensuring virtual switch programmability.

In this paper, we bridge this gap by proposing a hybrid FPGA-ASIC platform for switch virtualization. In contrast to existing work, we take advantage of increased clock frequencies with an ASIC and combine it with the advantages of reconfigurability and flexibility provided by an FPGA – the integrated platform delivers virtual programmable switches

that can work at line rate up breaking the terabit barrier. To the best of our knowledge, no virtualization architecture has proven capable of delivering such throughput while delivering proper abstractions of virtual switches with true tenant isolation, providing virtual switch hot-swapping, and supporting backend scalability [26]–[29].

The remainder of this paper is organized as follows. Section II presents related works. In Section III, we discuss our reference architecture for programmable switch virtualization. In Section IV, we describe our implementation of a hybrid FPGA-ASIC platform and emphasize the core technical aspects that enable it to break the terabit mark for virtual switches. In Section V, we present the results achieved with our prototypical implementation, highlighting area occupation and throughput. Finally, we close the paper in Section VI with concluding remarks and directions for future research.

II. RELATED WORKS

Even though the concept of switch virtualization is well-studied, the recent development of Domain-Specific Languages (DSL) targeting programmable forwarding planes has brought a renewed interest in the area, especially when targeting the data-plane virtualization. The first work in the literature to provide a forwarding plane virtualization platform was P4VBox [22]. The authors realized an open-source platform based on the canonical NetFPGA reference design [30] and adapted it for multiple switch instances. Their approach was able to instantiate up to 13 virtual switches and achieve a 40 Gbps throughput. Following this approach, MTPSA was proposed under the same premises: to provide multi-tenant portable switch architecture over the NetFPGA plataform [31]. Still, MTPSA is restricted to a 40 Gbps throughput due to its FPGA implementation.

Aiming to improve FPGA performance and improve previous works' throughputs, the literature draws the technology enabler to boost the NetFPGA platform to a 100 Gbps throughput with the NetFPGA SUME board based on a Xilinx Virtex-7 690T FPGA device [26]. To do so, the authors propose to use a set of the 30 serial links running at up to 13.1 Gbps. Still, the combined throughput falls short of 0.4 Tbps. Finally, newer FPGA devices such as the Xilinx UltraScale+ could improve performance and boost resource availability, but an improvement in an order of magnitude would be required to break the terabit wall.

When considering ASIC platforms for programmable dataplanes, virtualization can only be achieved by replicating the complete switch hardware. In other words, one must design a DSL-enabled ASIC switch and replicate it for each additional virtualized switch. Thus, for achieving multiple virtual switch instances, one must predefine the maximum number of virtual switches and design the ASIC accordingly. This approach has been adopted by several industry players, such as Broadcom Tomahawk [27], Intel Tofino [28], and Cisco Silicon One [29]. Even though these approaches reach the terabit throughput, their virtualization proposal is limited to a few instances, and
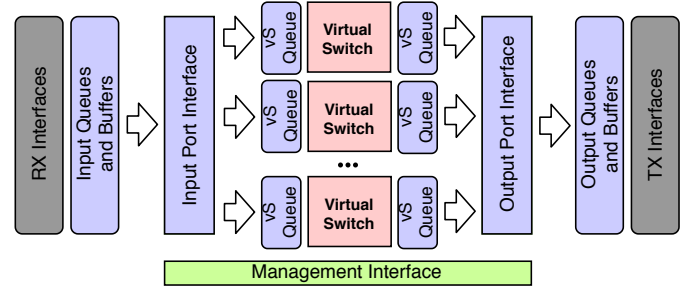

Fig. 1. Conceptual architecture for programmable switch virtualization.

resource waste is high, as one must design the switch hardware according to the worst-case switch instance.

This paper proposes a hybrid FPGA-ASIC platform to combine the best of ASIC and FPGA technologies while minimizing their downsides. To do so, we take advantage of the FPGA flexibility for implementing multiple parallel virtual switches and offload the remaining static hardware blocks to an ASIC platform. Thus, we can forward packets in multiple parallel virtual switches at gigabit throughputs in the FPGA-side and serialize their combined terabit throughput through the ASIC-side. By doing so, we can achieve up to 3.2 Tbps while running up to 26 virtual switches.

III. PROPOSED PLATFORM ARCHITECTURE

Our architecture was built on top of the canonical NetFPGA reference design [30] and adapted for multiple switch instances according to P4VBox [22]. It virtualizes the forwarding plane, providing supporting structures for deployment and parallel execution of virtual switch (vS) instances. The switch code can be written, compiled, and ran independently to provide data isolation between switch instances and protect vendors' intellectual property. Fig. 1 overviews the proposed architecture. It is composed of TX/RX network ports, an Input Port Interface (IPI), an array of vS instances (vS Array), an Output Port Interface (OPI), and a Management Interface (MI).

The **vS Array** is a set of vS placeholders. It contains multiple preallocated circuit areas with a given set of resources, each capable of implementing a single vS instance. The vS instance is the core of our architecture. It actively performs packet switching. To enable vS packet forwarding, the vS placeholders contain three communication channels that receive packets from the IPI, send packets to the OPI, and process control requests from the external MI. They all have individual logic to implement a vS instance, with an individual set of networking protocols, switch metadata, variable scope, and control flow. They also have a private memory space for storing match-action tables and registers, which can only be accessed by the vS instance. The introduction of multiple vS instances creates the need to steer flows form the RX network ports to corresponding vS instances and then back to the TX network ports. The IPI and OPI modules are responsible for these tasks.

The **IPI** buffers and serializes input packets to a parser. The parser searches the packet for a valid network segmentation tag

to decide where to steer it. We currently use 802.1Q VLAN to this end. We then apply the packet to an abstraction of an `Ingress` table, which the manager configures through the control engine. It has two actions: *forward* the packet to a vS instance or *drop* it. The forward action matches the packet VLAN tag and receives a device id as an action field, indicating the identifier of the vS to receive the packet. vS instances may belong to the same VLAN, as long as they do not share the same physical ingress ports. In case the parser cannot find a valid device id, the packet is dropped.

The **OPI** module delivers packets that have been processed by the vS array to a TX network port. It does so by iterating over vS instances' output buffers, forwarding packets as soon as the output TX port is available. It also applies the packet to an abstraction of an `Egress` table. As the IPI's ingress table, it also has two actions: *forward* the packet to a TX port or *drop* it. The forward action matches the packet VLAN and the `device_id` informed by the IPI, to ensure that a vS does not forward packets to unauthorized network segments.

The **MI** module provides a Command-Line Interface (CLI) between all architectural modules and the network administrator. It provides read/write access to all data structures, including tables and registers in the IPI and OPI and match and action tables on deployed vS instances. The control itself is not part of the platform implementation; instead, it lies on the software stack, external to the platform.

IV. HYBRID FPGA-ASIC PLATFORM IMPLEMENTATION

We conceive our hybrid FPGA-ASIC platform as a System-on-Chip (SoC) that targets a 65 nm commercial CMOS process from a partner foundry. Fig. 2 presents an overview of our platform. Red and purple blocks are part of the SoC (FPGA and ASIC logic, respectively). Grey blocks represent the physical layer, implemented using external ICs, providing the system with 32 100 Gbps RX/TX lanes. The green block represents an outer control layer, implemented off-chip, through a software stack running on an external microcontroller. A single input and 26 (one for each vS) output channels of 146 bits provide the middle ground between the MI and the chip.

To support reconfiguration, vS instances were allocated to an FPGA fabric, roughly equivalent to the Xilinx XCVU13P FPGA, while the MI was allocated to an external control layer. At full capacity, this device can support a vS Array with up to 26 virtual instances of the L2-switch, a popular case-study switch in the literature [30]. Partial and full reconfiguration can be used to repurpose the switches' network function, and thus, the function of the system as a whole.

The IPI and OPI modules and their input and output queues and multiplexers are implemented using standard cells and memory IPs in the ASIC portion of the SoC. This approach keeps the implementation of memory-hungry portions out of the FPGA, thus allowing us to pick a reasonable, cost-effective IP. A total of 52 queues are needed to interface IPI and OPI with up to 26 virtual switches. Additional 64 queues are needed to interface the former blocks with the 32 RX/TX lanes. We also implemented one virtual channel

through 2 extra vRX and vTX ports for loopback switching capabilities, resulting in 118 FIFOs being implemented. Our system partitioning approach establishes a trade-off where IPs are moved on the system according to IO count and bandwidth requirements [32].

A. ASIC Logic Implementation

For the ASIC portion of the proposed system, we have developed specific verilog codes that describe, in RTL, the following blocks: IPI, MI, OPI, and the interfaces to drive/read RX and TX ports. We achieved the best performance at power trade-off by combined use of transistors of low, standard, and high threshold voltages (LVT, SVT, and HVT, respectively). Synopsys Design Compiler was our tool of choice for logic synthesis, targeting a 1GHz clock frequency, a demanding frequency for the 65nm node.

A high degree of attention is given to the 118 FIFOs present in the design. Whenever possible, compiled SRAM blocks were used for packet storage in the queues. Our partner foundry also provides a SRAM compiler, describing it as a high-performance low-leakage single port compiler. Unfortunately, compiled SRAM comes in specific sizes/ratios that we have to adhere to. Our FIFOs have odd sizes: 52 x 289 (52 addresses with 289 bits) for the vS Array – IPI/OPI and 66 x 417 for TX/RX – OPI/IPI. The first example was implemented with flip-flops because the ratio is too asymmetrical: the compiler would not generate anything less than 32 addresses for the given word size. The second example is also non-trivial: 417 bits is not a power of two or even a multiple of two, for which we generated three instances of 64 x 144 to minimize the number of unused resources.

A total of 284 memory instances is required to provide the queues enough depth to achieve terabit bandwidth. The total number of SRAM bits is 2.54M (approximately 317K bytes). For all FIFOs but one, the storage is implemented with SRAM. All FIFOs have logic for write/read pointers and full/empty flags, which are implemented with conventional standard cells and operate as a wrapper around the SRAM memory instances. All FIFOs operate in asynchronous mode and are always on: this design choice increases area and power, but it is essential to enable the terabit bandwidth we seek.

B. FPGA Logic Implementation

The FPGA logic implements multiple vS instances in parallel and interface them with the ASIC logic. Fig. 3 shows a vS implementation. A vS is generated through a High-Level Synthesis (HLS) method based on a model similar to the Very Simple Switch (VSS) reference model[1] described in [30]. Each vS contains an ingress parser (Programmable Parser), multiple match-action pipeline stages composed of memory and ALU elements, and an egress parser (Programmable Deparser). Note that this is a suggested vS implementation flow; one could also write its own vS from scratch.

To design and deploy a vS into the FPGA logic, one must describe a vS RTL that is able to communicate through one

---

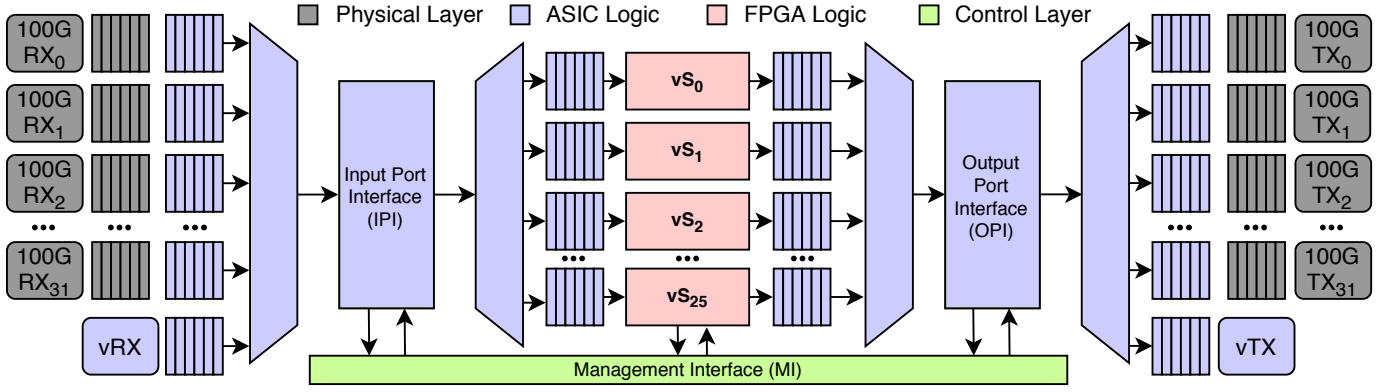[1]https://p4.org/p4-spec/docs/P4-16-v1.1.0-spec.html#sec-vss-arch

Fig. 2. Overview of our hybrid FPGA-ASIC platform conceived as a System-on-Chip.
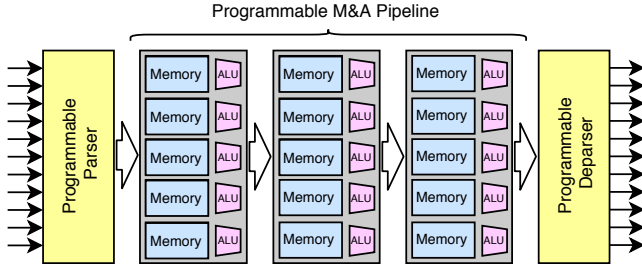

Fig. 3. vS instance implementation

of the available vS queues in the ASIC logic and describe its packet processing pipeline. Thus, as long as designers wrap their packet processing pipelines, they can generate virtual switches ranging from a single wire to a full-stack router.

To speed up the process of generating complex virtual switches, we employ an HLS design flow methodology to synthesize Domain-Specific Languages (DSL) such as P4 [12], Lyra [13], and POF [14]. For generating a switch implementation from a P4 description, we take advantage of the P4-NetFPGA project[2], which uses the Xilinx SDNet tool and the Simple Sume Switch model to automatically generate complex vS HDL. As the HLS tool is only compatible with Xilinx platforms, we emulated our FPGA fabric by targeting the Xilinx XCVU13P device.

The vS implementation flow consists of wrapping a generic HDL-described switch with data and (optionally) control communication channels. To do so, the vS wrapper parses the HDL-described switch, inserts wrapper interfaces to communicate with the ASIC logic, and creates a final HDL file that can then be synthesized/translated into a target board configuration file. Additionally, the implementation flow can generate management drivers to access information about the switch pipeline, including the match-action tables, registers, and methods for accessing these data through the external MI.

It is important that the HLS design takes into account the flexibility brought by network programmability, and make sure that network operators can redefine the forwarding plane behavior at will. Thus, the design must not impose static

switch deployment. Instead, it must ensure network programmability's flexibility while maintaining high throughput. A reconfigurable FPGA is ideal for achieving both constraints, as it provides both flexibility and low latency. FPGA reconfiguration can be achieved by employing full or partial reconfiguration [33].

With *full reconfiguration*, the entire FPGA fabric is reprogrammed using a single bitstream that contains each HDL-described switch that must be deployed. This method reduces the need for resources to place vS instances, as the bitstream synthesis process can be optimized for the specific set of vS instances that must be deployed. It means, for example, better area occupation, because of an optimal placement of each vS in the fabric. As a result, full reconfiguration enables placing more vS instances. It also delivers higher operation frequencies for the entire fabric, meaning that instances will operate with higher throughput and lower latency.

One problem with full reconfiguration is that instances cannot be replaced (i.e., undeploy a and deploy another one) unless the entire board is reconfigured. In this case, any running instances in the board are terminated, and the context and buffers/queues of each are lost. To solve this problem, *partial reconfiguration* can be used. In summary, it involves partitioning the fabric into static and reconfigurable areas. Each reconfigurable area in the design can then be reprogrammed without affecting the operation of other reconfigurable areas. Each reconfigurable area can then be used for deployment of a instance. The partial reconfiguration also enables faster deployment of a single instance, as only a portion of the fabric is reconfigured (in contrast to full reconfiguration).

In spite of the flexibility of independent (un)deployment of instances provided by partial reconfiguration, it requires more resources in the fabric for partitioning, therefore decreasing the number of vS instances that can be placed in the board. As the partitioning is done before vS deployment, each vS is constrained by the size of predefined partitions. More importantly, partial reconfiguration may also degrade the performance of vS instances, as the partitioning increases the distance between modules. It therefore brings critical implications to the deployment of switches that should achieve an aggregated throughput in the terabit mark.

---

[2]https://github.com/NetFPGA/P4-NetFPGA-public

V. RESULTS

In Table I, we show the results for the ASIC logic of our platform. All results are reported for a frequency of 1GHz, nominal VDD of 1.2V, and a TT (typical-typical) process corner. It is clear from the results that the memories dictate the area and power characteristics of the system. The memories represent 93.6%, 90.4%, and 92.8% of the area, dynamic power, and leakage power budgets, respectively. Furthermore, out of the 718K cells (343K flops), approximately 42K are buffers, indicating that our targeted frequency is not trivial to achieve. An analysis of the critical path revealed that SVT cells are employed with modest driving strengths, mostly X1 and X2. Our results also revealed that the frequency could be pushed up to 1.3 GHz for the TT corner, but at a high cost in both dynamic and leakage power since LVT cells (i.e., faster but leaky) would tend to be used more often.

TABLE I
RESULTS FROM ASIC SYNTHESIS ON 65NM TECHNOLOGY.

F=1GHz, VDD=1.2V	Logic	Memory	Total
Area (mm^2)	3.0	44.6	47.6
# cells	718227	284	718511
Dynamic power (W)	2.7	25.6	28.3
Leakage power (μW)	54.5	712.6	767.1

Table II presents data for the FPGA logic synthesis targeting the Xilinx XCVU13P FPGA. It shows the required LUTs, FFs, and BRAMs for each case-study switch implementation as well as achieved throughput for the minimum clock period of 1.392 ns (718.4 MHz clock frequency). The maximum clock frequency is directly related to the vS instances' buffers, implemented with BRAMs, where the critical path resides. Improvements to these structures could increase operating frequency, therefore improving overall throughput.

In terms of area occupation, L2-Switch has minimum requirements, followed by Firewall, Router, and INT, which requires the most resources. Considering a XCVU13P, we could fit 26 Switches-l2, 14 Routers, 17 Firewalls, or 11 INTs in a single board. For these switches, the bottleneck is BRAM availability. They require, on average, double the percentage of available BRAMs as LUTs and FFs. Implementing BRAMs as ASIC could increase design efficiency and throughput.

When considering latency and throughput, latency takes into account the time for a packet to cross the FPGA logic. In contrast, throughput considers the maximum amount of data passing through the FPGA logic. Thus, latency depends on the complexity of the vS instance's parser and deparser and the length of its match and action pipeline. The throughput, on the other hand, depends on vS latency and resource occupation. As an example, the maximum amount of L2-Switch instances running in parallel could achieve a maximum of 26 x 132.63 $Gbps$, or 3.448 $Tbps$. Note that a single L2-Switch instance can only achieve a maximum of 132.63 $Gbps$.

Table III presents the final results for the achieved throughput by our proposed platform. The platform's bottleneck is the

TABLE II
RESULTS FROM FPGA SYNTHESIS ON XCVU13P.

vS	LUTs	FFs	BRAMs	Throughput ($Gbps$)
L2-Switch	27,626	39,520	102	132.63
Firewall	48,979	76,147	153	130.92
Router	49,754	80,915	185	131.45
INT	77,956	155,594	240	129.61

Physical layer, with 32 100 $Gbps$ PHYs adding up to a 3.2 $Tbps$ throughput and, depending on the vS instances' complexity, the FPGA logic, which is constrained by its internal BRAM availability and varies between 1.43 and 3.45 $Tbps$. Note that additional PHYs could be added to the platform, and a different FPGA part could be used to increase maximum throughput. Considering 26 L2-Switch instances running on our proposed platform, we saturated the 32 100 $Gbps$ PHYs at 3.2 $Tbps$ in a single data direction, thus 6.4 $Tbps$ could be achieved in full-duplex mode.

A comparison with related works is difficult because, to the best of our knowledge, this is the first platform in the literature to provide switch virtualization at the terabit mark. While the literature is still focusing on implementing virtualization on FPGA platforms [22], [30], [31] at the gigabit mark, commercial approaches do not provide virtualization [27]–[29]. Therefore, our proposed proof-of-concept platform provides a combined virtualized throughput of up to 3.2 Tbps. This value is an order of magnitude higher than the available FPGA solutions but lower than the highest ranking commercial ASIC solutions at 25.6 Tbps [27]. Still, one could further expand our platform's throughput by increasing the clock frequency on the ASIC-side and the configurable resources on the FPGA-side, at costs in area and power.

TABLE III
ACHIEVED THROUGHPUT METRICS FOR OUR SoC.

Logic	Instance	#	Max. Throughput ($Tbps$)
Physical	100 $Gbps$ PHY	32	3.20
ASIC	AXI Stream	33	3.30
FPGA	vS Instance	11–26	1.43–3.45
Chip	Platform	1	3.20

VI. FINAL CONSIDERATIONS

The dawn of 5G and the emergence of next generation networks will pose a major challenge to virtualization solutions powering public and private clouds. The prospect involves supporting slices of virtual networks capable of handling an aggregated traffic at the terabit scale and whose behavior can be redefined through forwarding plane programmability. However, programmable forwarding plane virtualization is still at is infancy [19], [21], [22], [31], with researchers still leveraging FPGAs to address aspects like resource sharing and isolation between slices of virtual programmable devices.

In this paper, we discussed the design of a hybrid FPGA-ASIC platform for switch virtualization, conceived as a

System-on-Chip. In contrast to existing solutions, we leveraged increased clock frequencies with an ASIC, and combined it with the advantages of FPGA reconfigurability. Our experiments provide evidence of the potentialities of our hybrid platform, which is capable of achieving a combined virtualized throughput of 3.2 Tbps with low resource occupation overhead. Even though a comparison to related works is difficult, our results surpassed FPGA implementations by an order of magnitude while falling short to compete with the best available commercial ASIC solution, that does not provide switch virtualization, in terms of throughput.

In spite of the progresses achieved, much work remains. One direction for further research is ensuring that the pipeline of virtual switches can enable multiple access to counters, registers, and external functions, without degrading virtual switch throughput. We also intend to investigate how to leverage partial reconfiguration to enable the deployment of virtual switches with diverse memory occupation requirements.

References

[1] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 2, pp. 909–928, 2012.

[2] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[3] J. van de Belt, H. Ahmadi, and L. E. Doyle, "Defining and surveying wireless link virtualization and wireless network virtualization," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1603–1627, 2017.

[4] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 117–130.

[5] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown, and J. Rexford, "Pisces: A programmable, protocol-independent software switch," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 525–538.

[6] P. Kumar, N. Dukkipati, N. Lewis, Y. Cui, Y. Wang, C. Li, V. Valancius, J. Adriaens, S. Gribble, N. Foster, and et al., "Picnic: Predictable virtualized nic," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 351–366.

[7] T. Koponen, K. Amidon, P. Balland, M. Casado *et al.*, "Network virtualization in multi-tenant datacenters," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, pp. 203–216.

[8] X. Jin, J. Gossels, J. Rexford, and D. Walker, "Covisor: A compositional hypervisor for software-defined networks," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 87–101.

[9] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 655–685, 2015.

[10] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: A survey on principles, enabling technologies, and solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 3, pp. 2429–2453, 2018.

[11] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 99–110.

[12] 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.

[13] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics," in *ACM SIGCOMM 2020*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 435–450.

[14] H. Song, "Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane," in *ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 127–132.

[15] J. Gao, E. Zhai, H. H. Liu, R. Miao, Y. Zhou, B. Tian, C. Sun, D. Cai, M. Zhang, and M. Yu, "Lyra: A cross-platform language and compiler for data plane programming on heterogeneous asics," ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 435–450.

[16] Broadcom Inc., "Broadcom's new trident 4 and jericho 2 switch devices offer programmability at scale," 2019. [Online]. Available: https://www.broadcom.com/blog/trident4-and-jericho2-offer-programmability-at-scale

[17] D. Hancock and J. Van Der Merwe, "Hyper4: Using p4 to virtualize the programmable data plane," in *CoNEXT'16*. ACM, 2016, pp. 35–49.

[18] C. Zhang, J. Bi, Y. Zhou, and J. Wu, "Hypervdp: High-performance virtualization of the programmable data plane," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 556–569, March 2019.

[19] P. Zheng, T. Benson, and C. Hu, "P4visor: Lightweight virtualization and composition primitives for building and testing modular programs," in *CoNEXT '18*. New York, NY, USA: ACM, 2018, pp. 98–111.

[20] J. Krude, J. Hofmann, M. Eichholz, K. Wehrle, A. Koch, and M. Mezini, "Online reprogrammable multi tenant switches," in *ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*, ser. ENCP '19. New York, NY, USA: ACM, 2019, p. 1–8.

[21] M. Saquetti, G. Bueno, W. Cordeiro, and J. R. Azambuja, "Hard virtualization of p4-based switches with virtp4," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, ser. SIGCOMM Posters and Demos '19. New York, NY, USA: ACM, 2019, pp. 80–81.

[22] M. Saquetti, G. Bueno, W. Cordeiro, and J. R. Azambuja, "P4vbox: Enabling p4-based switch virtualization," *IEEE Communications Letters*, vol. 24, no. 1, pp. 146–149, Jan 2020.

[23] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5g," *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84–91, 2016.

[24] M. Condoluci and T. Mahmoodi, "Softwarization and virtualization in 5g mobile networks: Benefits, trends and challenges," *Computer Networks*, vol. 146, pp. 65–84, 2018.

[25] G. Bueno, M. Saquetti, J. R. Azambuja, and W. Cordeiro, "Defending lightweight virtual switches from cross-app poisoning attacks with vifc," in *Proceedings of the ACM SIGCOMM 2020 Conference Posters and Demos*, ser. SIGCOMM Posters and Demos '20. New York, NY, USA: ACM, 2020.

[26] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "Netfpga sume: Toward 100 gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014.

[27] Broadcom, *Tomahawk 4 switch first to 25.6 Tbps*, 2019. [Online]. Available: https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series

[28] Barefoot Networks, *Barefoot Tofino*, 2020. [Online]. Available: https://www.barefootnetworks.com/products/brief-tofino

[29] Cisco, *ONE Silicon, ONE Experience, MULTIPLE Roles*, 2019. [Online]. Available: https://blogs.cisco.com/sp/one-silicon-one-experience-multiple-roles

[30] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The p4-netfpga workflow for line-rate packet processing," in *ACM/SIGDA Int'l Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: ACM, 2019, pp. 1–9.

[31] 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.

[32] S. Pagliarini, J. Sweeney, K. Mai, S. Blanton, S. Mitra, and L. Pileggi, "Split-chip design to prevent ip reverse engineering," *IEEE Design & Test*, pp. 1–1, 2020.

[33] K. Vipin and S. A. Fahmy, "Fpga dynamic and partial reconfiguration: A survey of architectures, methods, and applications," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018.