Taxonomy of the Performance of P4 Targets

Irina Tsareva, Dominik Scholz*, Sebastian Gallenmüller*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: irina.tsareva@tum.de, {scholz,gallenmu}@net.in.tum.de

***Abstract*—The P4 language enables abstraction and flexible programming of the data plane for various hardware- or software-based targets, like field programmable gate arrays (FPGAs) or software switches for general-purpose CPUs. Each of these targets has its own limitations in available parallelization (e.g., multi-core), memory resources (e.g., RAM), or number of high bandwidth ports. Thus, one P4 program may have different latency and throughput characteristics, depending on the target machine. Analyzing the performance differences for P4 targets is crucial to identify bottlenecks and predict the performance of any P4 program. In this paper, we provide a comparison of performance measurements of P4 targets and show the impact of different P4 constructs. Hardware-based solutions have the lowest latency and highest throughput. Furthermore, we describe two model approaches to predict the performance of P4 targets: models based on benchmarking and on stochastics. While benchmarking-based models allow a straight performance comparison, they are highly dependent on target-specific information, and thus, may not be applicable to every target. Whereas the probabilistic model is implemented to be more general and can be further refined with information about the target.**

***Index Terms*—programmable data plane, P4, performance, performance model**

1. Introduction

Software-defined networking (SDN) [1,2] enables faster deployment, centralized management, and scalability through network control. This architecture decouples the control and data plane physically, such that the software-based control plane controls the data plane (e.g., switches, routers) over an open protocol like OpenFlow. Consequently, to support new header fields the OpenFlow API specification has to be extended and hardware switches redesigned or reprogrammed (e.g., application-specific integrated circuits (ASICs)), leading to an increased complexity of the API, additional deployment cycles, and a need for domain experts. The programming protocol-independent packet processors (P4) language aims to solve these issues by providing an abstraction layer between an API such as OpenFlow and the data plane.

P4 [2] is a protocol-independent, target-independent, and field reconfigurable language that can express how the data plane has to process packets. Protocol-independent means that there is no set of supported protocols (e.g., IPv4, Ethernet), and thus, this set has to be defined as desired. With P4, different supported platforms (*P4 targets*) can be programmed, like field programmable gate arrays (FPGAs), ASICs, or network processing units (NPUs). Field reconfigurable describes the property that P4 targets can be reconfigured after they were shipped.

Various implementations of P4 targets exist; either software- or hardware-based. Although the hardware-based solutions provide high bandwidth ports and specialized many-core architectures, they differ highly in their price, e.g., about 7,000$ for a NetFPGA SUME board [3], and maintenance costs. Moreover, each implementation uses different libraries, compilers, and optimization techniques to enable different features (e.g, stateful operations), and thus, are not suitable for every network setup. Yet, the implementation also influences the latency or throughput of P4 programs. It is crucial to understand and estimate the performance of P4 programs on a P4 platform correctly to have predictable execution and to identify optimization possibilities of a given setup.

At present, two approaches for performance estimation of arbitrary P4 programs exist: models based on benchmarking and on stochastics. The goal of this paper is to provide an overview of both approaches and their implementations. To this end, we focus on a selection of P4 targets (Section 2) and describe necessary performance measurements for the benchmarking-based model (Section 3). We present implementations of the benchmarking-based approach and one probabilistic model and discuss their advantages and disadvantages (Section 4).

2. Background

A device must explicitly support P4 programmability, otherwise it is not possible to program the device via P4. To understand the architecture of a *P4 target*, knowledge about the P4 language is needed.

2.1. P4 Programming Language

A P4 program consists of three stages: the parser, pipeline, and deparser stage. The parser is a finite state machine that takes the arriving packets (*ingress queue*) and extracts its headers into a stack. The pipeline includes multiple match-action units that process the parsed headers by performing user-defined actions such as modifications upon them. The match-type can be either exact, ternary, or lpm. Finally, the deparser reconstructs the packets with the modified, added, or removed headers and sends them to the outgoing packet buffer (*egress*

doi: 10.2313/NET-2022-01-1_06

queue) or drops them. In this context, we call the parsers, actions, and tables *P4 constructs*. They are the basic blocks out of which arbitrary P4 programs can be build. A P4 programs defines network functions (NFs), like NATs or firewalls. [4]

The *P4 compiler* consists of two parts: A generic open-source front-end and a target-specific back-end compiler (provided by the vendor). The front-end compiler transforms the P4 program into an intermediate representation (IR), which is then mapped into target-specific code (e.g., in C or Verilog) by the back-end compiler. [4]

P4 targets implement a target-specific *P4 architecture model* which describes the interface as well as fixed-function and programmable blocks. This interface enables the P4 constructs to be mapped onto the target. [5]

2.2. P4 Targets

We present a selection of P4 targets ordered by their degree of flexibility (descending), specialization (ascending), and price (ascending). Although GPU-based implementations exist [6], we do not discuss them here, since not enough performance measurement results exist.

2.2.1. Software-based. Software-based implementations can be run on a general-purpose CPU. They can make use of, e.g., the heap memory and available cores. They are more flexible to implement, since there are hardly any hardware constraints, but their behavior might be non-deterministic due to scheduling or interrupts during runtime [7].

Behavioral Model version 2 (bmv2) – bmv2 [8] is the reference software switch implementation of `p4.org`. This switch has only developing, testing, and debugging purposes due to its high latency and low throughput (up to 1 Gbit/s).

T4P4S DPDK-based – The data plane development kit (DPDK) is a collection of user-space libraries to accelerate packet processing. It includes multiple features to accelerate software-switches. The P4 program is compiled through T4P4S into C code that can be run on top of DPDK. [4]

PISCES – PISCES [9] is a software switch that is based on the virtual Open vSwitch (OVS) and uses the DPDK fast path instead of the less performant kernel modules. Additionally, the authors added new primitives to support encapsulation.

BPF-based – P4rt-OVS [10] extends the OVS-DPDK by the user-space berkeley packet filter (uBPF). These libraries add support for runtime extensibility and stateful operations. This implementation introduces a new front-end P4-to-uBPF compiler.

2.2.2. NPU-based. NPUs consist of "tens of multi-threaded purpose-built" cores that are optimized for network data packet processing, and thus, they are more specialized than CPUs. One example is the **Netronome SmartNIC**. To program this NPU, the IR is compiled into C code, and then, into firmware which is loaded onto the P4 target. [4]

2.2.3. FPGA-based. Hardware design can be described using a hardware descriptive language (HDL). This design is then synthesized, placed, and routed onto a specific FPGA. While developing the design the developer already has to know the target FPGA so she can meet platform-specific constraints, such as timing or available resources. If the constraints are not met, placing and routing will fail. In case of success, a bitstream can be generated and flashed onto the FPGA. The advantage is that the hardware can be optimized for an application.

P4→NetFPGA – P4→NetFPGA [11] is a workflow on top of the Xilinx P4-SDNet compiler and NetFPGA SUME to compile P4 code into Verilog code. The target FPGA is the NetFPGA SUME board.

P4-to-VHDL – P4-to-VHDL [12] compiles a P4 program into VHDL code without having an IR.

P4FPGA – P4FPGA [13] extends the `p4.org` front-end compiler by a back-end that first generates Bluespec System Verilog code. Then, this code is converted into Verilog code that can be synthesized to either Xilinx or Altera FPGAs. The generated code contains a P4 programmable packet-processing pipeline and a fixed-function pipeline.

2.2.4. ASIC-based. Hardware switches that are hard-wired for a specific application are ASIC-based. Their function cannot be reprogrammed making them less flexible, but more specialized. **Intel Barefoot Tofino 1** [14] is such an Ethernet switch. It uses static RAM (SRAM) and ternary content-addressable memory (TCAM) for P4 tables, depending on the match types [15].

3. Performance of P4 Targets

In this section we compare the performance of a selection of P4 targets on basic P4 constructs. Performance metrices are latency and throughput. Typically, the latency and throughput are best for an FPGA and ASIC, since they have application-specific, optimized hardware. However, hardware constraints limit their expressibility.

3.1. Latency

The overall latency depends on the amount of occurrences of basic P4 constructs in a P4 program. Table 1 depicts the impact of eight P4 constructs on the bmv2, T4P4S switch, PISCES, Netronome SmartNIC, and NetFPGA SUME (compiled via P4→NetFPGA).

For instance, modifying header fields has a negligible impact on the T4P4S switch, Netronome SmartNIC, and NetFPGA SUME board since they write the complete header, even if a single field is changed [4]. Since whole headers are emitted in the P4 deparser syntax [4] we expect to see a similar behavior for bmv2 and PISCES.

Comparing the targets listed in Table 1, the NetFPGA SUME board has the lowest overall latency for each P4 construct, while bmv2 has the highest. The T4P4S switch has comparable latency to the Netronome SmartNIC with packet sizes of 256 Bytes (B); for packets larger than 1000B or 1500B the Netronome SmartNIC has a better latency by 3ns-7ns (depending on the amount of the occurrence of P4 constructs). However, the Netronome SmartNIC scales the worst with increasing pipeline and action complexity. [4]

doi: 10.2313/NET-2022-01-1_06

P4 constructs	Impact on Targets								
	bmv2 [7]		T4P4S Switch [4]	PISCES [7]		Netronome SmartNIC [4]		NetFPGA SUME (P4→NetFPGA) [4]	
Parsing Headers	- -	$O(n^2)$	+	-	$O(n)$	-		- -	$O(n)$
Modifying Header Fields	n.a.		+	n.a.		+		+	
Operation Executions	- -	$O(n)$	n.a.	- -	$O(n)$	+		n.a.	
Modifying Headers	n.a.		+	n.a.		- -	$O(n)$	+	
Copying Headers	n.a.		+	n.a.		-		+	
Removing Headers	n.a.		+ +	n.a.		- -	$O(n)$	+ +	
Adding Headers	n.a.		-	n.a.		- -	$O(n^2)$	-	
Adding Tables	- -	$O(n^2)$	+	+		- -	$O(n)$	-	$O(n)$

TABLE 1: Latency impact of P4 constructs on P4 targets (median values). *(- - significant degradation (2.5μs – >30.3μs), - degradation (1.5μs–2.5μs), + no or a slight impact (0μs–1.5μs), ++ improvement (0.1μs–2μs), n.a. no value available)*

Osiński *et al.* [10] show that the latency of P4rt-OVS increases lineary for increasing number of match-action tables and is constant for varying number of table entries.

Vörös *et al.* [16] demonstrate that their implementation of T4P4S has comparable latency as PISCES. However, they do not compare it to isolated P4 constructs.

On the other hand, compiler and hardware design optimizations further decrease the overall latency. For example, P4rt-OVS [10], PISCES [9] and P4FPGA [13] implement post-pipeline editing, which postpones the modification of packets to the deparser. This reduces the performance at the deparser stage. PISCES merges multiple match-action pipelines into one, which leads to non-changing latency for increasing number of tables. P4-to-VHDL [12] introduces offset width and multiplexer optimizations. However, the authors do not provide performance comparisons with other P4 targets which is why we cannot classify its performance relative to the listed P4 targets. Zhou *et al.* [17] reduce the latency of Netronome SmartNIC and bmv2, by chaining NFs, reducing redundant functions, and bypassing undesired functions.

3.2. Throughput

Parallelism, like introduced by FPGAs, may significantly increase the throughput. P4FPGA outperforms PISCES also in terms of throughput. Moreover, throughput may depend on the complexity of the P4 program: If the number of headers increases, the throughput of P4FPGA (on NetFPGA SUME) and PISCES (on a CPU) decreases. [13]

PISCES [9] (optimized) has a smaller throughput by 2% compared to OVS. If new protocols are added, the throughput decreases, e.g., about 35% from 51.1Gbps to 33.2 Gbps if post-pipeline editing is activated.

Osiński *et al.* [10] show for three network functions, that the throughput of P4rt-OVS is comparable to PISCES and OVS for packet sizes of 128B and 256B; For 512B the throughput is larger than for PISCES.

P4-to-VHDL [12] can parse traffic with a complex protocol structure with 100 Gbps. The optimized Intel Barefoot Tofino 1 [14] can achieve a bandwidth of 100 Gigabit Ethernet per port.

4. Performance Models

Evaluating each possible P4 application and combination of P4 constructs for each P4 target is neither feasible nor desirable. A performance model should describe the estimated performance of an arbitrary complex P4 program for every or a specific P4 target. Next, we describe two approaches to model estimation and discuss their advantages and disadvantages.

4.1. Models Based on Benchmarking

This approach combines performance measurements of isolated P4 constructs obtained through benchmarking with the occurrences of P4 constructs obtained through P4 program analysis. Since the benchmarks do not test for real-world workloads but for core features of P4, they are called *synthetic benchmarks*. Each feature is tested in isolation with varying parameters, e.g., incrementally increasing number of packet headers and fields in the parser.

WhipperSnapper [7] was the first benchmarking suite for the P4 language. It contains latency, throughput, and memory usage measurements against five features: Parsing, processing, state accesses, packet modification, and action complexity. Additionally, WhipperSnapper includes target-dependent benchmarks, that test specifically for the hardware support and usage of ASICs and FPGAs as well as the latency and throughput of CPUs and NPUs with reduced scheduling and locking impact.

Harkous *et al.* [4,5] continue this idea and focus on eight features related to parsing, processing, packet modification, and action complexity (see **P4 constructs** in Table 1). Yet, their benchmarks focus only on latency. They explain and validate their model examplary for the Netronome SmartNIC, NetFPGA SUME board (compiled via P4→NetFPGA), and T4P4S switch. Each of these has target-specific latency measurement results. The slopes of the measurement results for each feature can be (piecewise) interpolated and stored in a target-profile-vector. This vector is calculated only once for each target since it does not change. Next, the P4 program has to be analyzed and the occurence of P4 constructs determined. The estimated average latency of a network function is then the sum of the prior measured latencies of the (isolated) P4 constructs with respect to their amount of occurrences.

Scholz *et al.* [15] suggest to derive models based on the weaknesses of a target. In the case of software switches, the weakness is their significantly varying latency due to different implementations of P4 elements (e.g., the match types) and the underlying memory access pattern. Whereas the resource utilization is a weakness of

ASICs since their available memory limit the complexity of P4 programs. The authors derive cost functions for varying properties of the match-action pipeline (e.g., table entry size and varying table key width lengths) for the T4P4S switch and Intel Barefoot Tofino 1.

All described models require target specific information (e.g., memory resources, software implementation) to accurately estimate the performance. For instance, the model of Harkous *et al.* [4] has an accuracy of more than 94% and is especially accurate for the NetFPGA SUME. It is less accurate for the Netronome SmartNIC due to its high dependency on the P4 pipeline. Moreover, latency measurements of isolated P4 constructs are not always additive; the measured latency of combined P4 constructs may be smaller due to e.g., reduced memory access [5]. Therefore, summing the latency of isolated P4 constructs may reduce the accuracy, too.

Due to the highly target-dependent nature of these models, the performance measurements for isolated P4 constructs could be made publicly available by researches, and hence, facilitate and accelerate performance prediction. The P4 program analysis can be done by the compiler. [4]

4.2. Models Based on Stochastics

While the previously described approach requires benchmarking to derive target-specific parameter values for cost functions, this approach is based on a more generic probabilistic model (cf. Bayesian network). The match-action tables of a P4 program are converted into a control flow graph (CFG). The CFG depicts all possible execution paths of the program: a node depicts the line number the program counter points to, while an edge is a possible transition to the next event; each node is associated with an execution cost. Thus, the expected execution cost of a P4 program is the sum of the conditional expected costs of an execution path. For instance, if a path is not executed, it has cost of 0. [18]

The advantage of this framework is that it is generic enough to be used for every target. To get more accurate results, additional information about the target (e.g., hardware configuration, runtime environment) can be added in a modular way. [18]

5. Conclusion and Future Work

The P4 language allows abstraction and programmability of the data plane. Due to its target-independence, multiple targets can be programmed using the same set of P4 constructs. However, the performance differ significantly for each target due to hardware constraints and software implementations.

In this paper we summarized studies comparing and analyzing the impact of different P4 constructs on the latency, throughput, and memory usage. These benchmarks in addition with information about the target implementation and the control flow can be used to derive performance models. Another approach is to use a probabilistic model based on expected execution costs. These performance models can estimate the performance for arbitrary complex P4 programs, and thus, help create predictable networks.

As future work, it would be of interest to analyze other P4 targets (e.g., P4rt-OVS) with respect to P4 constructs and compare these results with the existing ones. Furthermore, more exhaustive measurements could be done to also investigate the impact of not yet included performance aspects, such as jitter or power draw.

References

[1] IBM, "What is Software-Defined Networking (SDN)?" https://www.ibm.com/services/network/sdn-versus-traditional-networking, [Online; accessed 22-March-2021].

[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. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[3] Digilent, "NetFPGA-SUME Virtex-7 FPGA Development Board," https://store.digilentinc.com/netfpga-sume-virtex-7-fpga-development-board/, [Online; accessed 08-May-2021].

[4] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with Predictable Packet Processing Performance," *IEEE Transactions on Network and Service Management*, pp. 1–14, 2020.

[5] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards Understanding the Performance of P4 Programmable Hardware," in *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2019, pp. 1–6.

[6] P. Li and Y. Luo, "P4GPU: Accelerate packet processing of a P4 program with a CPU-GPU heterogeneous architecture," in *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2016, pp. 125–126.

[7] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A P4 Language Benchmark Suite," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. Association for Computing Machinery, 2017, pp. 95–101. [Online]. Available: https://doi.org/10.1145/3050220.3050231

[8] p4.org, "BEHAVIORAL MODEL (bmv2)," https://github.com/p4lang/behavioral-model, [Online; accessed 22-March-2021].

[9] 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. Association for Computing Machinery, 2016, pp. 525–538. [Online]. Available: https://doi.org/10.1145/2934872.2934886

[10] T. Osiński, H. Tarasiuk, P. Chaignon, and M. Kossakowski, "P4rt-OVS: Programming Protocol-Independent, Runtime Extensions for Open vSwitch with P4," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 413–421.

[11] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4→NetFPGA Workflow for Line-Rate Packet Processing," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. Association for Computing Machinery, 2019, pp. 1–9. [Online]. Available: https://doi.org/10.1145/3289602.3293924

[12] P. Benácek, V. Puš, and H. Kubátová, "P4-to-VHDL: Automatic Generation of 100 Gbps Packet Parsers," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 148–155.

[13] H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4FPGA: A Rapid Prototyping Framework for P4," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. Association for Computing Machinery, 2017, pp. 122–135. [Online]. Available: https://doi.org/10.1145/3050220.3050234

[14] "Intel® Tofino™Series," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch.html#tofino, [Online; accessed 22-March-2021].

[15] D. Scholz, H. Stubbe, S. Gallenmüller, and G. Carle, "Key Properties of Programmable Data Plane Targets," in *Teletraffic Congress (ITC32), 32nd International*, 2020, pp. 1–9.

[16] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, "T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–8.

[17] 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.

[18] D. Lukács, G. Pongrácz, and M. Tejfel, "Performance guarantees for P4 through cost analysis," in *2019 IEEE 15th International Scientific Conference on Informatics*, 2019, pp. 305–310.

doi: 10.2313/NET-2022-01-1_06