Towards Understanding the Performance of P4 Programmable Hardware

Hasanin Harkous*†, Michael Jarschel†, Mu He*, Rastin Pries†, Wolfgang Kellerer*
**Technical University of Munich*, firstname.lastname@tum.de
†*Nokia Bell Labs*, firstname.lastname@nokia.com
Munich, Germany

***Abstract*—P4 programmable data planes are becoming more popular due to the flexibility they provide in describing the packet processing pipeline. P4 successfully abstracts the processing pipeline of data planes using a limited set of constructs. The performance variation as a function of the configured P4 pipeline is an important aspect that should be studied. Analyzing the impact of different P4 constructs on packet latency helps in understanding the overall performance of P4 programmable devices.**

In this paper, we analyze the impact of a basic set of P4 constructs on packet processing latency to derive the influential parameters. We use the derived results to propose a method for estimating the packet latency of P4-based network functions implemented using the surveyed P4 constructs. Finally, we validate the accuracy of the proposed method by applying it to realistic network functions.

***Index Terms*—Programmable Data Plane, Performance Evaluation, P4**

I. Introduction

Traffic demands of computer networks are in constant flux: the increasing number of devices and emerging new applications call for flexible network designs that can adapt to changing traffic patterns [1]. In this regard, the emergence of Software-Defined Networking (SDN) and Programming Protocol-Independent Packet Processors (P4) [2] follows the paradigm shift toward network programmability, which enables new perspectives to build adaptable networks.

Since its proposal, the data plane programming language P4 has captured the attention from both academia and industry. In a nutshell, P4 describes the processing pipeline of packets, including parsing, match-action, and deparsing, all of which can be customized. Furthermore, new protocols (e.g., HULA [3]) and applications (e.g., Inband-Network Telemetry (INT) [4]) can be efficiently prototyped with P4.

As another advantage, P4 is a high-level domain-specific language which abstracts details of devices (i.e., targets), and only keeps the logical steps of packet processing. Therefore, the same P4 design can be executed on various targets, including BMv2 software switch [5], Netronome SmartNIC [6], TOFINO ASIC Switch [7] and NetFPGA [8]. To leverage different targets for different use cases, we need to fully understand their performance characteristics. We try to answer the question: *what is the relation between packet processing latency and a certain pipeline structure?* which is not yet fully explained in the literature. In this paper, we fill this research gap starting with the widely used Netronome SmartNIC [6].

Performance analysis of P4 targets is challenging because each stage of a P4 pipeline can have different features and building blocks which makes it practically impossible to evaluate all combinations. In this regard, we start our evaluation by measuring the simplest cases and gradually increase the complexity of the pipeline (including parser, control blocks, and deparser) to make the evaluation procedure more meaningful. The measurement results enable us to present a generalized estimation method for predicting packet latency by analyzing P4 programs. With this method, we can cover a wide set of possible network functions rather than measuring particular ones. Overall, we make the following main contributions:

- We analyze the impact of P4's various basic elements on packet latency and identify the influential variables.
- We propose and validate a method to estimate the total packet latency of a P4 pipeline implemented on SmartNIC.

This paper is organized as follows. The related work is listed in Section II. Section III provides some background information about P4 and Netronome SmartNIC. Section IV introduces our measurement testbed and methodology. We discuss the measurement results and provide a latency estimation method in Section V. Section VI concludes the paper.

II. Related Work

In the following, we review the previous work targeting the performance evaluation of programmable networks. Begin et al. [9] presented an analytical queueing model to evaluate the performance of a DPDK-based software switch which is abstracted as a polling system. A model based black-box systematic testing method [10] is introduced for the multiple-level pipeline of an SDN data plane. Jarschel et al. [11] developed a mathematical model for the forwarding speed and blocking probability of an OpenFlow switch. With extensive measurements, He et al. [12] explored potential latencies involved in OpenFlow hardware switches that motivate a rethinking of the switch design. Jose et al. [13] explored the design of compilers for programmable ASIC switches that map logical lookup tables to physical tables, while meeting data and control dependencies in the program.

Regarding programmable networks implemented with P4, WhipperSnapper [14] was the first benchmarker that can generate various test cases which cover both target-agnostic properties, e.g., the latency of parsing and state access, and target-dependent properties, e.g., occupied hardware resources. Performance evaluation of various P4 targets can be found in the literature, e.g., P4FPGA [15] and P4-to-VHDL [16] for FPGA, and PISCES [17] for Open vSwitch. In a nutshell, they evaluate the impact of the number of headers, tables, and actions on the processing latency or throughput. Furthermore, P4NFV [18] considers the performance degradation during changing the packet processing pipeline at runtime.

In this paper, we focus on a SmartNIC, whose performance has not been studied in previous research, and generate representative test cases that give us the most insights and enable estimation of its packet processing latency.

III. Background

In this section, we briefly overview the basic elements of a P4 program and the architecture of Netronome SmartNICs under evaluation.

P4 Programming Language: P4 is a domain-specific language for describing the pipeline of packet processors. The language can describe arbitrary data plane protocols and can run on different platforms. In the latest P4$_{16}$ version [19], the language is separated from the target's architecture so that each can evolve independently. While the target's architecture reveals the view of the pipeline, defining the programmable and non-programmable blocks, the P4 code describes the functionality that should run on the programmable devices. A typical packet processing pipeline, programmed via P4, is abstracted into the following stages:

Parser: The parser is the block where bits are extracted from packets and saved into header structures. Parsers are programmed as a Finite State Machine.

Control Blocks: The extracted headers are manipulated and transformed at this stage. The body of a control block resembles a traditional imperative program. Match-action units (i.e., tables) can be applied from within the control blocks. The match-action units are made up of keys, with three matching types, and a set of actions that can be applied upon matching. The actions are defined separately, and they can be called explicitly in the control block, or implicitly by the match-action units. The actions consist of a basic set of operations, such as modifying header fields, modifying metadata, evaluating arithmetic and binary expressions.

Deparser: The modified packet is reconstructed at this stage where headers are appended to the packet before leaving the packet processor pipeline.

Netronome SmartNIC: The Netronome SmartNIC is a Network Processor Unit (NPU) that can be programmed via P4. It has a highly parallel architecture with tens of multi-threaded purpose-built cores. In addition, the architecture utilizes hierarchical transactional memory and built-in accelerators to enhance its processing performance. A P4 program is compiled using the open-source front-end compiler from P4.org [19], and a back-end compiler from Netronome to generate target-specific C implementation of the datapath. The C files are then used to generate the firmware to be downloaded on the SmartNIC. Further details about the SmartNIC architecture and its programmability via P4 can be found in [20]. A schematic showing Netronome SmartNIC's pipeline is presented in Fig. 1.

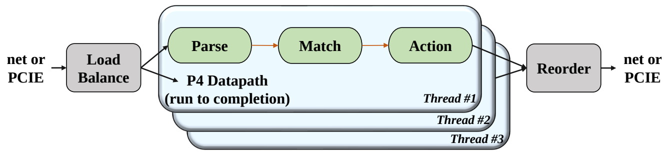


Fig. 1: P4 pipeline with multi-threaded architecture in Netronome SmartNICs [20].

IV. Testbed Setup and Methodology

In this section, we describe the testbed setup and the experiments designed to understand the latency cost of different P4 constructs.

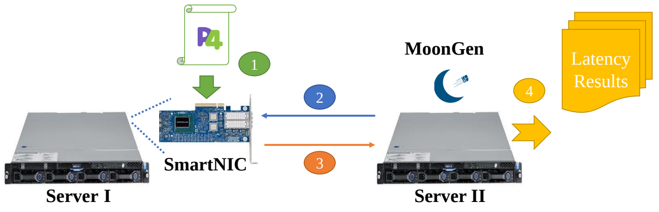


Fig. 2: Testbed setup consisting of P4 programmable SmartNIC and packet generator.

Fig. 2 shows the built setup. We perform the experiments using two Nokia NDCS16 AirFrame Compute Nodes with 16 cores (dual-socket Intel Xeon CPU E5-2630 v3 @ 2.40GHz) and 64GB of 2133 MHz DDR4 memory. An Agilio CX 2x10GbE SmartNIC from Netronome [6] is plugged into Server I through the PCI bus. The two physical ports of the SmartNIC are connected to two interfaces of Server II running MoonGen packet generator [21]. For every measurement case, a different P4 program is loaded into the SmartNIC. More than 100 thousand packets of length 1000 Bytes are generated by MoonGen and sent over one link to the SmartNIC at 10 Gbps rate. After being processed in the SmartNIC, packets are sent back to MoonGen for measuring and reporting packet latency results.

Four experiments are designed to study the effect of the basic elements of a P4 pipeline on packet latency. The examined P4 constructs cover header parsing, actions applied on headers, and match-action tables. The results of these experiments are used in Subsection V-E to derive a method for estimating the packet latency of network functions, which is made up of the examined constructs, by analyzing its P4 program. In the following, we describe the objective of the performed experiments and the P4 pipelines used for every test case. Note that all tested P4 pipelines contain a single table responsible for modifying the egress port while matching on the ingress port based on a fixed rule. Additionally, the applied actions are defined as explicit actions, in the control block of the ingress stage, unless it is specified that they are implicitly invoked due to table matching.

A. Modifying Header Fields

The objective of this experiment is to study the effect of modifying a different number of fields of the same header. This answers the question: *How will the processing latency of a P4 pipeline vary if a single IP header field, such as source IP address, was modified compared to the case when multiple IP header fields, such as source and destination IP addresses, were modified?* For this purpose, we consider two sets of P4 programs that differ in the following: In the first set, a single header field of three different headers is modified, while in the second set all the fields of the headers are modified.

B. Executing Operations

In this experiment, we investigate the latency cost of the application of arithmetic and binary operations on header fields defined in P4 actions. We aim to study the impact of each type of operation on the processing latency separately. For this purpose, we consider two sets of P4 programs where we vary the number of applied binary operations in the first set, while we vary the number of applied arithmetic operations in the second set. Note that the compiler ignores the expressions if the result is not used in the P4 pipeline. For this purpose the header fields are modified in all the P4 pipelines considered in this experiment.

C. Parsing and Modifying Headers

TABLE I: Parsing and Modifying Headers.

Case ID	**Parsed Headers**			**Modified Headers**		
	Eth	IPv4	UDP	Eth	IPv4	UDP
A0	+	-	-	-	-	-
A1	+	-	-	+	-	-
B0	+	+	-	-	-	-
B1	+	+	-	+	-	-
B2	+	+	-	-	+	-
B3	+	+	-	+	+	-
C0	+	+	+	-	-	-
C1	+	+	+	+	-	-
C2	+	+	+	-	+	-
C3	+	+	+	-	-	+
C4	+	+	+	+	+	-
C5	+	+	+	+	-	+
C6	+	+	+	-	+	+
C7	+	+	+	+	+	+

The objective of this experiment is to investigate the impact of header parsing and header modification on the processing latency. We look into the output of varying these two parameters in the same experiment to analyze the relationship between these P4 stages, besides analyzing its scaling behavior. The summary of the tested P4 pipelines in this experiment is shown in Table I. First, we increase the number of parsed headers from one to three, which correspond to Case IDs A*, B*, and C* shown in Table I. Afterward, we consider all possible combinations of modifying the parsed headers. For example, while A0, B0, and C0 correspond to the cases where only parsing of headers takes place, A1, B3, and C7 correspond to the cases where all parsed headers are modified. Furthermore, we measure the latencies of all the cases listed in Table I when header modification takes place using explicit actions, from within P4 control blocks, as well as when it takes place using implicit actions, as a result of table matching, to analyze the difference. In total, the latencies of 28 different P4 pipelines were measured in this experiment. Note that the analysis in this experiment is limited to three headers since the used packet generator can only measure the latency of Ethernet and UDP packets [21].

D. Adding Tables

The objective of this experiment is to quantify the latency cost of adding tables into a P4 pipeline. We start by a pipeline that contains a single table responsible for modifying the egress port while matching on the ingress port based on a single installed rule. Then, we increase the number of defined tables in the ingress stage of the pipeline up to 10 tables, while measuring the latency cost of added tables. All added tables match on a single field and perform no actions.

V. Results & Estimation Method

In this section, the results of the experiments described in Subsections IV-A, IV-B, IV-C, and IV-D are presented and interpreted in Subsections V-A, V-B, V-C, and V-D respectively. Finally, we make use of the analyzed measurement results to propose a method for estimating the latency of P4 programs in Subsection V-E. Note that all conclusions derived in this section are valid for Netronome SmartNICs.

A. Impact of Header Fields Modification

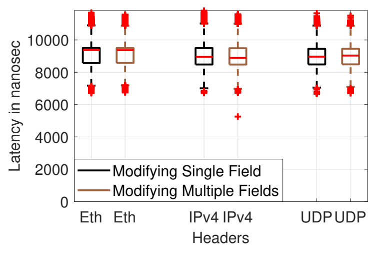
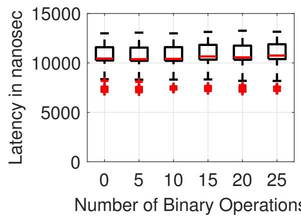


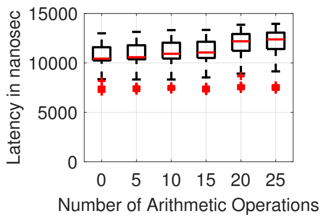
Fig. 3: Forwarding latency given number of modified fields.

Fig. 3 shows the box plots of the measured packet latency, in nanoseconds (ns), when a single field of Ethernet, IPv4 and UDP headers is modified versus the case when multiple fields are modified. For all the considered headers, we can observe that the latency results when modifying a single header field are very similar to the results when multiple header fields are modified. This is justified as the SmartNIC writes the complete new header when a single field is modified. This is also consistent with the P4 language deparsing syntax, where complete headers are emitted in case they are modified. As a result, we infer that the packet latency of a P4 pipeline depends on the number of modified headers disregarding the number of fields modified within a header.

B. *Impact of Operations Execution*



(a) Binary Operations.



(b) Arithmetic Operations.

Fig. 4: Forwarding latency given number of executed operations.

Fig. 4 shows the box plots of packet latency, in ns, as a function of the number of operations applied on header fields. From Fig.s 4a and 4b, we can observe that the increase in packet latency is negligible when more binary operations are applied, while it reaches approximately 1.9μseconds (μs) when the number of arithmetic operations increases to 25. The SmartNIC efficiently performs operations in on-chip memory [20] which lead to negligible latency when processing binary operations and minimal latency when processing arithmetic operations. As a result, the latency cost of binary operations can always be neglected when estimating the packet latency of a P4 pipeline, while arithmetic operations should be included only if a significant number of operations is applied.

C. *Impact of Headers Parsing and Modification*

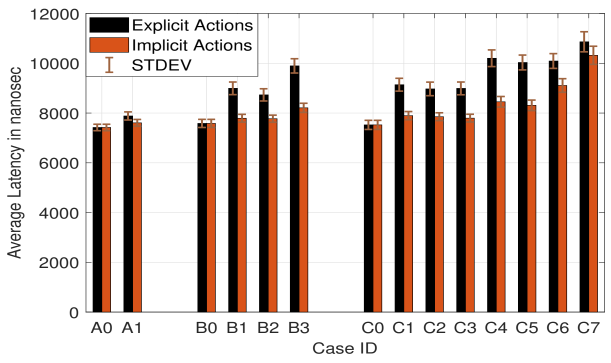


Fig. 5: Forwarding latency given number of parsed and modified headers.

Fig. 5 shows the average and standard deviation of packet latency when P4 pipelines defined in Table I are loaded into the SmartNIC. The cases that have the same parsing stage are grouped together, referenced as A*, B*, and C* respectively. The latency results when headers are modified using explicit actions are shown in black, while the latency results when actions are applied implicitly are shown in brown. Recalling that the number of fields modified within a header does not impact the packet latency, as discussed in Subsection V-A, the analysis in this section focuses on the impact of parsing and modifying different headers disregarding which fields are modified. From Fig. 5, we can derive the following observations:

Headers Parsing: From cases A0, B0 and C0, we can observe that the impact of parsing additional headers, without header modification, on packet latency is negligible. Note that this observation is valid while parsing up to three headers, and further scaling is needed to generalize this statement.

Headers Modification: The latency cost of modifying additional headers is clearly observable. For example, in the case of UDP parsing, when the number of modified headers varied from 0 to 3 as in cases C0, C1, C4, C7, the total additional latency reached 3.3 μs in the explicit case and 2.8 μs in the implicit case.
Additionally, we can observe that the extra latency of modifying headers is not additive. For example, adding the extra latency of modifying Ethernet and IPv4 headers separately, as in B1 and B2, sums up more than the case B3 where both headers are modified within the same pipeline. This is interpreted as a result of single memory access for writing the headers in B3 compared to adding the latencies in B1 and B2 where memory is accessed twice even though less data is written in each case.

Relation between Parsing and Header Modification: We can observe that modifying the same header while having different parsing stages, leads to different latency costs. For example, modifying the Ethernet header while parsing a different number of headers, leads to different latency results as can be observed from cases A1, B1, and C1. The latter observation shows that the latency cost of varying the parsing stage and varying the number of modified headers is not independent. Further implementation details from the target vendor are needed to understand the concurrency model between the stages of the P4 program.

Moreover, we can observe that with identical parsing blocks, the latency varies according to the number of modified headers with less effect, 0.2 μs difference, when looking to which headers are being modified. For example, we can observe that the latency is almost the same in cases C1, C2, and C3 where only a single header is modified, i.e. Ethernet, or IPv4 or UDP header. The same holds in cases C4, C5, and C6 when two headers are modified. This is an expected result keeping in mind that P4 is designed to be protocol independent where headers are defined and treated as a string of bits. As an exception, the result of the implicit case in C6 shows a slight increase in latency compared to C4 and C5 although the number of modified headers is the same in these three cases; a behavior which needs further investigation.

Implicit versus Explicit Actions: In all cases when header modification is applied, we can observe that explicit actions lead to more latency compared to implicit actions. This is interpreted as an effect of target-specific optimizations, such as the flow tracker described in [20].

Based on the observations presented in this subsection, we can infer that the packet latency of a P4 pipeline depends on both the number of parsed headers and the number of modified headers. Table II summarizes the results of the average measured latency, for both explicit and implicit actions, as a function of the number of parsed and modified headers.

TABLE II: Average measured latency as a function of the number of parsed and modified headers.

Headers Parsed	1	2		3		
Headers Modified	1	1	2	1	2	3
$L_{Explicit}$(in μs)	7.9	8.9	9.9	9.0	10.1	10.9
$L_{Implicit}$(in μs)	7.6	7.8	8.2	7.8	8.6	10.3

D. Impact of Tables Scaling

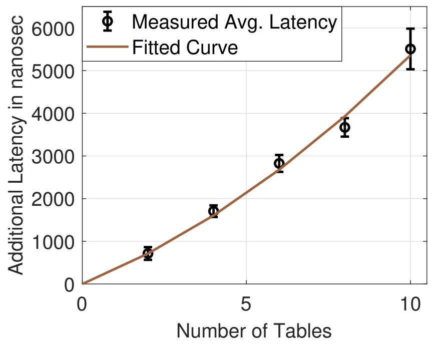


Fig. 6: Additional latency due to tables.

Fig. 6 shows the average and standard deviation of the measured extra latency when additional tables are added into the P4 pipeline. Adding tables into the P4 pipeline increases the latency as more matching takes place before a packet can leave the pipeline. The latency cost of adding 10 tables reaches more than 5μs. Additionally, the standard deviation of measured packet latency also increases as a function of the added tables, which can be a result of packets contention on accessing the SmartNIC's memory where tables' data are stored. We fit a curve for the measured latency cost to capture the increasing trend. The curve can be formulated as a second-order polynomial with the following equation:

$$f(\gamma) = 22.44 \cdot \gamma^2 + 311.6 \cdot \gamma \quad \text{for} \quad \gamma = 0, ..., 10 \qquad (1)$$

where $f(.)$ is the additional latency in ns, which varies as a function of $\boldsymbol{\gamma}$, the number of added tables.

E. Estimation Method and Validation

In this subsection, we propose a method for estimating the packet latency of P4 pipelines that are built using the previously investigated P4 constructs. Then, we validate the accuracy of the proposed estimation method using two realistic network functions.

According to the previously presented results in Subsections V-C and V-D, the estimation method depends on the following parameters: *(1)* The number of parsed headers, α, *(2)* The number of modified headers, β, and *(3)* Number of tables minus one, γ. Note that γ counts the number of added tables to the basic P4 pipeline which already contains a single forwarding table. The number of fields modified within a header and the number of arithmetic and binary operations can be ignored based on the results presented in Subsections V-A and V-B.

Given a P4 program, after extracting the parameters α, β, and γ, the estimated average packet latency, denoted as $\hat{L}$, can be computed as the summation of two components:

1) The latency component that depends on parsing and header modification which can be read from Table II based on the values of α and β.
2) The latency component that depends on the number of tables in the P4 pipeline which can be evaluated using Eq. 1 based on the value of γ.

Validation: For validation, we use the proposed method to evaluate the performance of two realistic network functions: *(i) L3_forwarding*, and *(ii) L3_forwarding + Firewall*. Then, we compare the estimated latency to the real measured one to evaluate the accuracy of the proposed method.

L3_forwarding: The P4 program of this network function includes: *(1.)* Parsing Ethernet and IPv4 headers, *(2.)* Matching on IPv4 destination address in a single table, and *(3.)* Modifying Ethernet header (MAC source and destination addresses) and IPv4 header (time to live) in case of matching. Extracting the relevant parameters of this P4 program gives: $\alpha = 2$, $\beta = 2$, and $\gamma = 0$, where actions are applied implicitly. Accordingly, the average latency is estimated as follows: $\hat{L} = 8200 + f(0) = 8200$ ns.

L3_forwarding + Firewall: In this case, we add a UDP-based Firewall to the previously described P4 pipeline. In addition to the previously described P4 elements, the P4 pipeline will also include parsing of UDP header, and one additional table matching on a UDP port. No extra additional actions are applied in this case. Extracting the relevant parameters of this P4 program gives: $\alpha = 3$, $\beta = 2$, and $\gamma = 1$, where actions are applied implicitly. Accordingly, the average latency is estimated as follows: $\hat{L} = 8600 + f(1) = 8957$ ns.

The measured and estimated average latency of the evaluated network functions are reported in Table III. The small error in the latency estimation of two examples reflects the precision and validity of our proposed method.

TABLE III: Validation results of the proposed method.

Network Function	**Meas. Avg.** L	**Est. Avg.** L	ΔL
L3_Fwd	8387 ns	8200 ns	187 ns
L3_Fwd+Firewall	9022 ns	8957 ns	65 ns

VI. Conclusion & Future Work

As P4 programming language is becoming more mature, it is important to understand the impact of different P4 constructs on the packet processing performance. We start by looking into the impact of a basic set of P4 constructs on packet latency when running on Netronome SmartNICs. After identifying the number of parsed headers, the number of modified headers, and the number of tables to be the influential parameters, we propose a method for estimating the latency of network functions implemented using the surveyed P4 constructs. We also validate the accuracy of the proposed method and show that it can estimate the packet latency of realistic network functions with high precision. This work is considered a first step towards modeling the latency of P4 programs. As a continuation of the current work, we plan to extend the evaluation to cover the following: *(i)* Studying the impact of other P4 constructs such as adding/removing headers, registers, checksum update, etc. *(ii)* Further investigating the concurrency model of different P4 stages. *(iii)* Performing measurements on other P4 targets such as software switch, NetFPGA, and TOFINO switch.

References

[1] M. He, A. M. Alba, A. Basta, A. Blenk, and W. Kellerer, "Flexibility in softwarized networks: Classifications and research challenges," *IEEE Communications Surveys & Tutorials*, 2019.

[2] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 44, no. 3, pp. 87–95, 2014.

[3] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research (SOSR)*. ACM, 2016, p. 10.

[4] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018, pp. 357–371.

[5] "Behavior model software switch," https://github.com/p4lang/behavioral-model, accessed: 2019-07-04.

[6] "Netronome smartnic," https://www.netronome.com/products/smartnic/overview/, accessed: 2019-07-04.

[7] "Barefoot tofino," https://www.barefootnetworks.com/technology, accessed: 2019-07-04.

[8] 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, 2014.

[9] T. Begin, B. Baynat, G. A. Gallardo, and V. Jardin, "An accurate and efficient modeling framework for the performance evaluation of dpdk-based virtual switches," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 15, no. 4, pp. 1407–1421, 2018.

[10] J. Yao, Z. Wang, X. Yin, X. Shiyz, and J. Wu, "Formal modeling and systematic black-box testing of sdn data plane," in *Proceedings of the International Conference on Network Protocols (ICNP)*. IEEE, 2014, pp. 179–190.

[11] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and performance evaluation of an openflow architecture," in *Proceedings of the 23rd International Teletraffic Congress (ITC)*, 2011, pp. 1–7.

[12] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in sdn-enabled switches," in *Proceedings of the Symposium on SDN Research (SOSR)*. ACM, 2015, p. 25.

[13] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling packet programs to reconfigurable switches," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, 2015, pp. 103–115.

[14] 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 (SOSR)*. ACM, 2017, pp. 95–101.

[15] 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 (SOSR)*. ACM, 2017, pp. 122–135.

[16] P. Benácek, V. Pu, and H. Kubátová, "P4-to-vhdl: Automatic generation of 100 gbps packet parsers," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 148–155.

[17] 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 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016, pp. 525–538.

[18] M. He, A. Basta, A. Blenk, N. Deric, and W. Kellerer, "P4NFV: An NFV architecture with flexible data plane Reconfiguration," in *Proceedings of the International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 90–98.

[19] "P4 language consortium," https://p4.org, accessed: 2019-07-04.

[20] "Mapping p4 to smartnics," https://p4.org/assets/p4_d2_2017_nfp_architecture.pdf, accessed: 2019-07-04.

[21] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *Proceedings of the Internet Measurement Conference (IMC)*. ACM, 2015, pp. 275–287.