A comprehensive latency profiling study of the Tofino P4 programmable ASIC-based hardware

Franco, David; Ollora Zaballa, Eder; Zang, Mingyuan; Atutxa, Asier; Sasiain, Jorge; Pruski, Aleksander; Rojas, Elisa; Higuero, Marivi; Jacob, Eduardo

Published in:
Computer Communications

Link to article, DOI:
10.1016/j.comcom.2024.01.010

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

Computer Communications 218 (2024) 14–30

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A comprehensive latency profiling study of the Tofino P4 programmable ASIC-based hardware

David Franco [a,1], Eder Ollora Zaballa [b,c,1,*], Mingyuan Zang [b], Asier Atutxa [a], Jorge Sasiain [a], Aleksander Pruski [b], Elisa Rojas [d], Marivi Higuero [a], Eduardo Jacob [a]

[a] Dept. of Communications Engineering, University of the Basque Country (UPV/EHU), Bilbao, Spain
[b] Department of Electrical and Photonics Engineering, Technical University of Denmark, Lyngby, Denmark
[c] 3 (Hi3G Denmark Aps), Fadet 4, Copenhagen V, Denmark
[d] Universidad de Alcalá, Departamento de Automática, Alcalá de Henares, Spain

ARTICLE INFO

Keywords:
Software-defined networking
Network programmability
P4
Data plane latency
Machine learning

ABSTRACT

Network softwarization has significantly evolved since programmable data planes became topical in academia and industry. Programming Protocol-Independent Packet Processors (P4) is a language to define packet forwarding behavior. Forwarding devices that are programmed with the P4 language support a flexible way to define headers, parse graphs, and data plane logic. However, extending the data plane with additional functionalities has an impact on packet data plane latency. For this reason, this paper analyzes the key factors that affect data pane latency to packets processed by the Tofino-based target (Tofino Native Architecture (TNA)), which can be considered the *de facto* production-ready and P4-programmable Application-Specific Integrated Circuit (ASIC). Our work first provides an extensive set of latency measurements and, afterwards, it includes a set of data plane latency predictions using the model derived from the latency results and machine learning (ML) algorithms. We demonstrate that the PCA-lasso polynomial (PLP) obtains the best results among the algorithms tested. The best-case results show that PLP obtained an accuracy of 98.22% prediction accuracy when considering the parser, deparser, and the control block for traffic running at 10 G/s (SFP+) and 100 G/s (QSFP28). To the best of our knowledge, this is the first work that provides such a comprehensive profiling, including a method to predict data plane latency in production-grade Tofino ASIC-based switching hardware, which could be leveraged to yield accurate latency values prior to investment and deployment.

1. Introduction

Society is experiencing an enormous, technology-driven step forward in services and applications. Communication systems thus face a great challenge, as they must keep pace with the requirements that services impose, such as real-time transmission or massive data rates. Furthermore, the diversity of new services has caused an unavoidable need for flexible networks.

In this scenario, Software-Defined Networking (SDN) [1] flourished as an alternative to traditional network systems, facilitating network evolution. Additionally, new perspectives and possibilities have arisen with the emergence of Data Plane Programming (DPP) [2,3] and the Programming Protocol-Independent Packet Processors (P4) language [4]. While, SDN promotes independence from vendor-specific technologies and interfaces by implementing a logically centralized control plane, DPP complements SDN as it provides additional flexibility at data plane level by defining the packet processing behavior. In fact, the P4 language is used to define the packet processing pipeline and create custom algorithms and functions to handle traffic, resulting in both tailored control and data planes.

While flexibility is essential to modern networks, performance must not be overlooked. In fact, the requirements of currently deployed applications are becoming more stringent and focused on specific performance indicators. For example, in the industrial sector, response time may be crucial, mainly due to the need for manufacturing processes to send status messages and receive action messages in near-real-time. Processing systems designed to support DPP and, more specifically, the P4 language, can be implemented in multiple targets, such as (i) software switches (Behavioral Model version 2 (bmv2) Simple Switch target [5], T4P4S and DPDK), (ii) SmartNICs (Netronome Agilio [6], Pensando Distributed Services Card (DSC) [7]), (iii) field-programmable gate array (FPGA) boards (NetFPGA [8]), or (iv) Application-Specific

* Corresponding author at: Department of Electrical and Photonics Engineering, Technical University of Denmark, Lyngby, Denmark.
E-mail address: eoza@dtu.dk (E. Ollora Zaballa).
[1] David Franco and Eder Ollora Zaballa are co-first authors.

https://doi.org/10.1016/j.comcom.2024.01.010
Received 6 October 2023; Received in revised form 22 December 2023; Accepted 14 January 2024
Available online 9 February 2024

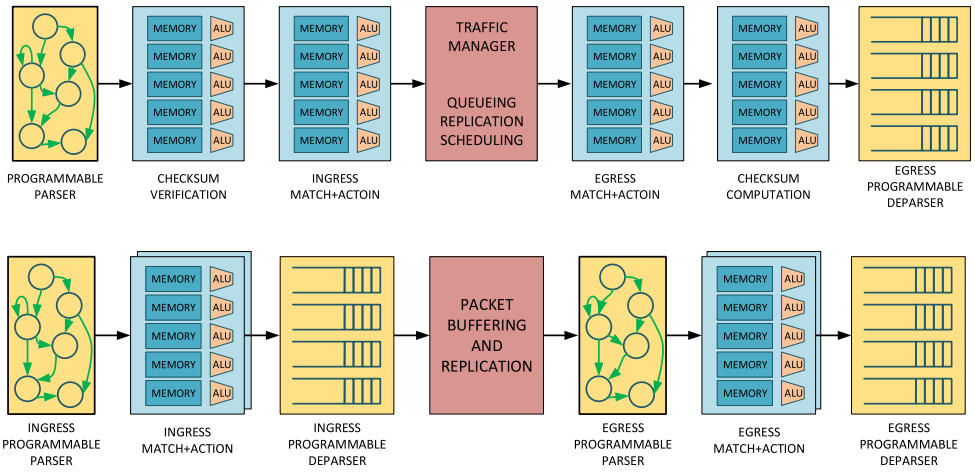


Fig. 1. V1Model [14] (top) and PSA [15] (bottom) architectures.

Integrated Circuits (ASICs) (Tofino [9] and Tofino 2 [10]). The programmable software switches, even if they are low-cost and particularly flexible, might present limitations in high-performance prototypes. Therefore, delay, processing capabilities, and load optimization should be carefully studied and optimized to fulfill performance requirements in designing packet processing programs. Nevertheless, few works provide a thorough study of production-grade programmable switch pipelines and the effect of key parameters in data plane latency such as packet format, program composition, parsing and deparsing blocks, etc. The effect of these parameters could become pivotal in pipelines that rely on data plane modularity [11,12]. Since modules include a considerably large set of P4 constructs, it is vital to predict the modules' influence on data plane latency to fulfill the requirements of each use case.

Accordingly, the purpose of the paper is threefold. First, (i) it analyzes the data plane latency behavior of Tofino-based P4 switches (APS BF2556X-1T [13]), as they represent the hardware target that might potentially obtain better performance results for production networks. Once the analysis is presented, (ii) the paper provides a comparison of data plane latency for the traffic running via interfaces configured as 10 Gb (SFP+) compared to 100 Gb (QSFP28). In addition, the paper creates a model and compares it with several machine learning (ML) algorithms to predict its data plane latency. Finally, (iii) the paper lists important use cases that can benefit from deploying a system that predicts the expected data plane latency. To the extent of our knowledge, it is the first work that tries to provide such a comprehensive analysis. Our main objective is that our study serves as a reference profile of the P4 Tofino-based switch.

The paper is organized as follows. Section 2 describes the background of SDN, data plane programming, and the relevance of P4-related concepts in this paper. Section 3 reviews the research context that motivate this work and that establishes its present relevance. Section 4 explains the methodology used, and it reviews the theory behind the data plane measurements performed in our evaluation; the test results for the parser/deparser and the control block are presented in Section 5. Section 6 develops, from these results the relevant regression models to be analyzed. Section 6.3 presents the different machine learning algorithms used to predict data plane latency; subsequently, these predictions are compared in Section 7. To end, Sections 8 and 9 discuss and conclude the paper, respectively.

2. Background

2.1. Software-Defined Networking (SDN) and P4 language

In 2008, McKeown et al. [16] published a Data Control Plane Interface (DCPI) protocol (also referred to as southbound interface protocol) named OpenFlow. The publication of this work triggered a revolution in networking, in which SDN became a key topic of research. OpenFlow and the SDN architecture [17] defined a logically centralized control plane that communicates with the data plane (OpenFlow switch) using the OpenFlow protocol to install flow rules and collect network statistics, among other control actions. With the logical centralization of the control plane, new path calculation algorithms and network deployments arose, changing the *de facto* distributed networking design. As the OpenFlow specifications advanced [18], some missing features became evident [19]. The limitations fostered the creation of a data plane programming language named P4 [4]. The advent of P4 triggered what could be called the second generation of SDN, in which data plane programming fostered a new design based on centralized and local programmable control [20] as a key component in network softwarization.

Currently, P4 [4] is the most popular language to program the data plane. Since its first version ($P4_{14}$ [21]), the most extended P4 version is $P4_{16}$ [22]. The language comprises key terms within the core library and additional constructs, known as externs, defined by architecture-dependent libraries. The externs are used to describe library elements like counters, meters, registers, or checksum computation functions (to name a few).

2.2. Data plane target architectures

The device used in this paper is programmed using the Tofino Native Architecture (TNA) [23] (more in Fig. 2). Architectures represent a design feature that creates an abstraction of the pipeline, associating the P4 language with a P4 target. Architectures also define the programmable blocks of the pipeline and the interfaces that represent a contract between the developers and the devices. Individual manufacturers can model their target's programmability, exposing only the programmable parts of the pipeline. The V1Model [14], used in the bmv2 Simple Switch target [5], and the Portable Switch Architecture (PSA) [15] are two well-known architectures in the P4 developer community.

In Fig. 1, both architectures share blocks that perform analogous functionalities. The similarities and differences between models depend, generally, on the programmable blocks and the extern definitions. Still, both architectures in Fig. 1 share elements like the parser, match-action units (also named control block in this paper), and deparser blocks. While a few differences exist, the block organization is primarily similar in the V1Model and PSA.

The parser defined in the architecture is responsible for establishing the state machine in control of header extraction. In other words, it matches packet bits into typed representations [24]. The match-action

Table 1
Summary and comparison of the relevant publications in Related Work.

Publication	Features						Latency modeling		
	Target	Architectural blocks	Precision	Testing methodology	Testing tools	Tested speeds	Fitting algorithm	Prediction	Test case
Dang et al. [26]	bmv2	Parser control block (action table)	millisecond	NIC timestamps (MoonGen)	MoonGen [28]	10 Gbps	N/A	N/A	N/A
	PISCES		microsecond						
	P4FGPA + Xilinx SDNet		nanosecond		FGPA-based custom packet generator	Hardware simulation			
Harkous et al. [29]	Netronome Agilo CX SmartNIC [6]	Parser/deparser control (headers tables)	microsecond	NIC timestamps (MoonGen)	MoonGen [28]	10 Gbps	Data interpolation + target profile vector	Accuracy: 94%	L3_fwd, L3_fwd+firewall, VxLAN_Decap
	NetFPGA-SUME (Xilinx Virtex XC7V690T) [8]								
	T4P4S + DPDK								
Scholz et al. [31]	T4P4S + DPDK (Intel Xeon E5-2640)	Baseline parser/deparser Control (multiple table properties)	microsecond	NIC Timestamps (MoonGen)	MoonGen [28]	10 Gbps (+ packet duplication)	CPU performance model (linear regression, multiple fitting algorithms)	Model is stated but not tested	N/A
	Tofino 1 (Delta ET-X064FFRRB)	Parsing/deparsing resource cost for table entries and key width.	nanosecond				ASIC resource usage model	N/A	N/A
Franco et al.	Tofino hardware switch [9]	Parser/deparser, Ingress control	nanosecond	Data plane timestamps	Anritsu MT1000A Network Master Pro [30]	10 100 Gbps	Both linear and nonlinear fitting: SLP, PLP, kNN, RF	Accuracy: 96%	Enterprise, data center, edge, service provider, INT, Big Union

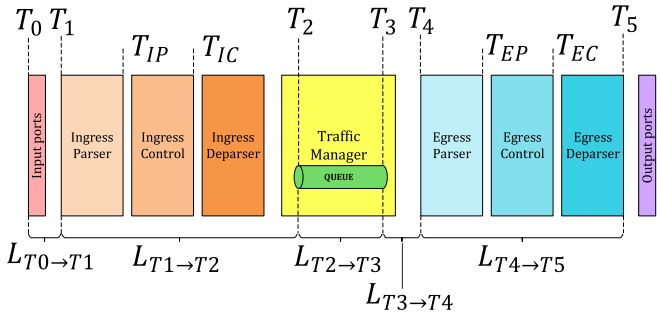


Fig. 2. Architecture [23] and timestamps available as metadata (T_0 to T_5) [26,27] and timestamps used only for clarification purposes (T_{IP}, T_{IC}, T_{EP}, T_{EC}) in the switch architecture.

units or control blocks specify the packet processing logic using tables and actions (among other constructs). Finally, the deparser assembles the packet structure [25], using a developer-defined header order.

2.3. Data plane language compilers

In addition, the P4 compiler is a key element when programming a P4 target. The official compiler for the P4 programming language is *p4c*. It provides a standard frontend and midend that can be used with a target-dependent backend. For instance, the bmv2 Simple Switch uses the V1Model. *p4c-bm2-ss* is the backend included in the *p4c* compiler to be used with the bmv2 Simple Switch target. Similarly, targets based on the NetFPGA, Pensando DSC, or the Tofino ASIC feature their own P4 compilers. Manufacturers distribute these compilers along with the necessary architecture-specific libraries (see Table 1).

3. Related work

Packet parsers usually represent a bottleneck in high-speed networks due to their complexity, as argued by Gibb et al. [28]. The authors introduce the design principles for packet parsers, pointing out the difference between fixed and programmable parsers, and providing several tips for an optimal design of programmable parsers. The literature shows a range of approaches to measure the performance of P4 programmable hardware and software devices.

For instance, Dang et al. [29] presents a benchmark designed to identify the key features and metrics of P4 compilers, regardless of the P4 target. Platform-agnostic and platform-specific benchmarks and artificial programs and workloads are presented to evaluate the essential P4 operations in several P4 targets. This method does not consider the whole target but treats the P4 operations in isolation, which may not indicate the actual performance if different devices are analyzed.

In an experimental analysis, Harkous et al. [30] measure the latency of packets through the P4 pipeline using a basic set of P4 constructs. They carry out four experiments that include multiple scenarios, considering modifications in header fields, binary or arithmetic operations, the number of parsed headers, and the addition of tables. They also propose a packet delay estimation method, which is then validated with realistic network functions. They perform their tests using a Netronome Agilio SmartNIC [6] as a P4 target and packet flows of up to 10 Gb/s using the MoonGen [31] traffic generator. They conclude that changing one or more header fields does not increase latency, as the headers are written entirely when a single field is modified.

In a following paper, Harkous et al. [32] continue the P4 pipeline latency estimation study previously presented. This paper analyzes additional parameters that might increase latency, using 75 different pipelines and implementing them in 3 different targets: NetFPGA-SUME card, Netronome SmartNIC, and T4P4S DPDK-based software switch. In addition to repeating their previous experiments, they also study header addition copying and removal. Results show essential differences among the targets, as in the case of parsed headers. Parsing more headers increases the latency in devices such as Netronome SmartNIC and the NetFPGA, whereas in T4P4S, this process does not introduce additional latency. This is because all parsed headers are copied to the memory in a single operation, and the headers' size does not impact latency. Authors conclude that different targets respond differently to P4 constructs, depending on the parameters added, removed, or modified.

In contrast to Harkous et al. [30] [32], we study and compare the packet processing latency of P4 pipelines at speeds of 10 Gb/s and 100 Gb/s, using a purpose-specific traffic generator named Anritsu

MT1000A Network Master Pro [33]. Moreover, we consider a hardware switch that includes the Tofino ASIC, covering the most widespread P4-programmable ASIC in the market.

Scholz et al. [34] focus on performance modeling of P4-programmable devices, aiming at the impact of match-action tables on the latency, throughput, and resource consumption of P4 programs. The authors cover two different target platforms — a general-purpose CPU-based target (extensively) and an ASIC target (minimally) — and elaborate performance and resource models. Their method consists of building a baseline model by executing a minimal P4 program and later measuring the differences due to adding match-action tables regarding the number of tables, size, and match type. The core of the work is focused on the highly dynamic nature of CPU targets, leading to throughput and latency models in different scenarios. The paper identifies the main limiting factors when using P4 targets, such as L3 cache misses and the lack of specialized hardware like Ternary Content-Addressable Memory (TCAM). For the ASIC target analysis, the authors emphasize the minimal impact that P4 program complexity has on latency (in the nanosecond range) and, hence, focus their work on elaborating a resource consumption model for the utilization of static random-access memory (SRAM) and TCAM resources. However, authors only aim at match-action tables, and parser or deparser blocks are not considered. In addition, the paper mainly focuses on the T4P4S [35] software target, and the latency impact at the nanosecond range is marginal, which we thoroughly analyze in this paper.

Having the latency measurements, latency modeling has also been studied in some publications in order to analyze the latency data. Among the literature, linear regressions are a common-used method to fit the latency data and analyze the data features. For instance, Scholz et al. [34] apply the least squares curve fitting for the packet rate later to derive the cycles per packet for exact match entries. Harkous et al. [30] also leverage a modeling method to predict the latency in different network test cases. In particular, they design a target-profile vector to structure the blocks and make the linear fitting based on it. However, most of these authors use software switches, which are affected by vastly different factors compared to hardware switches. For instance, it is common to use timestamps collected at the NIC to measure the software switch data plane latency. Besides, the software switches rely on a general-purpose CPU, which does not guarantee the same performance and consistency as ASICs do [32]. Measuring the processing latency by a CPU might, in some cases, be affected by additional CPU tasks, I/O, and delays unrelated to packet processing. As long as data plane timestamps are available, profiling a hardware switch guarantees consistency across architecture blocks with a lower standard deviation. Nonetheless, latency measurement presents a non-linear tendency in some of these results, and the presented regression might not be enough to fit such a performance in order to make accurate predictions. In this case, ML algorithms can be beneficial.

Regarding this last aspect, a few researchers have investigated the use of ML for latency analysis in other scenarios. For example, Qianet al. [36,37] introduce the support vector regression method to study the latency performance of Network-on-Chip systems. In addition, ML (specifically linear regression and random forest models) is used by Wang et al. [38] to predict computation performance variables, such as latency, on a fog computing manufacturing scenario. Yet, ML algorithms have not been applied for efficient data plane latency analysis. In order to fill this gap, we bolster our analysis with ML algorithms to model and predict the data plane latency based on the features described in the following sections.

3.1. Summary of contributions

The main contributions of this article can be summarized into four pillars:

- We provide an extensive analysis of the Tofino ASIC, not yet investigated in any related work.
- We study packet processing latency of P4 pipelines at faster speeds than literature (10 Gb/s and 100 Gb/s).
- Additionally, our latency modeling not only involves the match-action tables, but also the parser, control and deparser blocks of the Ingress pipeline.
- Finally, our study comprises an ML-based prediction algorithm to extend our results to further scenarios.

4. Data plane measurements

This section describes the measurement methodology followed to test the latency in the data plane for packets processed by a Tofino P4 programmable switch. Our method focuses on calculating the latency experienced by packets at various blocks in the TNA architecture model [23], namely: the parser, the control block (match-action units), and the deparser. This process is compatible with the resources on the match-action unit (MAU). However, might not trust the latency values as they change often, that is why we need further testing. Even though the current paper only focuses on the TNA architecture, some observations also apply to similar architectures like the PSA. However, the paper is based on Tofino-specific building blocks, including time measurements, metadata, or externs.

This paper defines *data plane latency* as the end-to-end delay experienced by a packet along the pipeline, measured at ingress port (T_0) and egress deparser (T_5). More precisely, it is the time difference between T_5 and T_0 ($T_5 - T_0$) as defined in Fig. 2. Understanding how the possible packet features and P4 statements affect each architecture model block if the time is strictly calculated end-to-end is not trivial. Besides, calculating data plane latency using the last and first timestamps complicates data plane predictability. Therefore, taking the entire set of available data plane timestamps is complicated, but the current section offers a thorough explanation of how each block eventually affects end-to-end pipeline latency.

Following the widespread methodology to measure data plane latency mentioned in Section 3, our paper relies on using data plane timestamps instead of timestamps measured at the NIC. These timestamps are the means for a model that, afterwards, will help predict the latency (Section 6). The TNA switches leveraged in this paper offer timestamping information available to the P4 programming language (as metadata fields). As depicted in Fig. 2, the pipeline exposes a set of timestamps (T_0, T_1, T_2, T_3, T_4, T_5) that can help determining the latency experienced at each block. The rest of the timestamps (T_{IP}, T_{IC}, T_{EP}, T_{EC}) were added for clarification purposes in the figure. For instance, if the goal is to create a model that tries to predict the control block latency, then that is equivalent to predicting ($T_{IC} - T_{IP}$). However, (T_{IC} and T_{IP}) are not available to the programmer, so the process of predicting the latency of the control block will be based on calculations made with the available timestamps (T_2 and T_1), which is done keeping certain code parts static across tests.

As indicated in Fig. 2, the data plane latency can be calculated as the difference between the first and last pipeline timestamps ($L = T_5 - T_0$): It can be further described as the sum of all the data plane block latencies, as follows:

$$T_5 - T_0 = \sum_{i=0}^{4} L_{T_i \to T_{i+1}} = L_{T_0 \to T_1} + L_{T_1 \to T_2} + L_{T_2 \to T_3} + L_{T_3 \to T_4} + L_{T_4 \to T_5} \quad (1)$$

Among the terms listed in Eq. (1), $L_{T_1 \to T_2}$ and $L_{T_4 \to T_5}$ refer to programmable blocks in the pipeline, named ingress and egress pipelines, respectively. Since no timestamp is exposed by the target's architecture within ingress and egress pipelines, Fig. 2 includes a few additional labels to separate each of the three blocks (T_{IP}, T_{IC}, T_{EP}, T_{EC}). Thus, $L_{T_1 \to T_2}$ and $L_{T_4 \to T_5}$ are defined as follows:

$$L_{T_1 \to T_2} = (T_{IP} - T_1) + (T_{IC} - T_{IP}) + (T_2 - T_{IC}) \quad (2)$$

$$L_{T_4 \to T_5} = (T_{EP} - T_4) + (T_{EC} - T_{EP}) + (T_5 - T_{EC}) \quad (3)$$

As depicted in Fig. 2 and Eq. (2), T_{IP} and T_{IC} define a timestamp at the end of the ingress parser and the ingress control block, respectively. Similarly, T_{EP} and T_{EC} define the end of the egress parser and the egress control block, respectively. These timestamps are not available to the programmer, but they help to define the start and end of each block from the ingress and egress pipeline.

Actually, this paper focuses on calculating the parser and deparser latency (together as one value) and the control block separately. This allows parser and deparser block programming and latency calculation, independent of how tables and actions are handled. This method also calculates control block latency, irrespective of how the packet is parsed.

According to Eq. (2), the ingress parser and deparser latency are defined and calculated as $(T_{IP} - T_1) + (T_2 - T_{IC})$. The ingress control block latency at the ingress pipeline refers to the $(T_{IC} - T_{IP})$ expression.

In order to build consistent experiments, when testing an architectural block, we assume that additional static code is necessary for other architectural blocks (the ones that are not under test).

4.1. Parser and deparser

When this paper refers to parser and deparser latency, it does not refer to the specific hardware entity in the ASIC responsible for header identification or header extraction/assembly. Instead, this paper refers to the architecture blocks that abstract the group of elements in charge of parsing and deparsing headers, which could encompass several hardware entities such as header identification, header extraction, buffering, and memory I/O as stated by Gibb et al. [28]. More precisely, the paper refers to the parser and deparser as the blocks defined as such in Fig. 2.

To find a correlation across different tests, the logic of the control block is kept static. In other words, the code does not change across tests that show the influence of the parser and deparser in data plane latency.

The ingress/egress controls remain statically programmed across all tests related to the parser and deparser latency. Therefore, the latency difference across tests can directly be associated with the additional P4 code added to the parser and deparser. The latency is calculated as $T_2 - T_1$ and $T_5 - T_4$ in the ingress and egress pipelines, respectively. The control block's only purpose is to collect timestamps, which are, across all tests, deparsed as a header.

4.2. Control block

As in the parser and deparser case, the latency in the control block (ingress or egress) is calculated by keeping the parser and deparser code static across the tests. When the control block latency is calculated, the same headers are parsed and deparsed, so no changes are made to the parser or deparser from the baseline test to the last one. The data plane code that changes across tests employs different types of table keys, number of keys, number of entries in the table, and actions. The tests that try to define the features in the control block latency will always have a baseline test. The baseline is established by assigning a static egress port (no table is used); this is the action that shows the lowest latency impact on packets that can be associated with the control block.

As in Section 4.1, the latency regarding the control blockis calculated using the same timestamps: $T_2 - T_1$ and $T_5 - T_4$ for both ingress and egress blocks, respectively. Since the parser and deparser remain static across tests, the latency time differences are only associated with the control block.

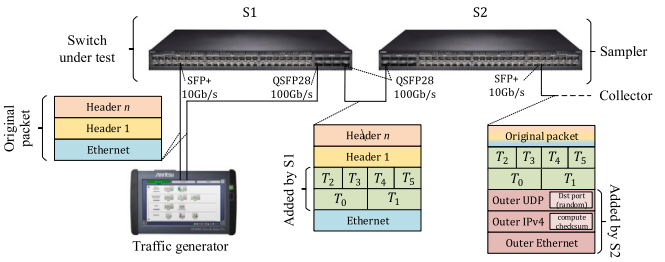


Fig. 3. Testbed setup and traffic treatment. The network tester generates packet streams and sends them to the switch under test (S1). This switch processes the packet and sends them to the sampling switch (S2). Finally, the collector captures and processes the timestamps of the packet as the last switch encapsulates the sampled packets.

4.3. Packet features and model sections

While other research papers [30,32] have followed a performance estimation for SmartNICs, software switches, and FPGAs, in this paper, we focus on providing a similar model to predict data plane latency in a switch based on a programmable ASIC (Tofino). Our tests are designed to measure the latency performance of a programmable pipeline under different load conditions and considering packets with different features (e.g., packet length, number of headers, etc.), which will help in understanding and predicting the performance of the programmable data plane.

4.4. Testbed and equipment

The testbed used for this study consists of four entities, as seen in Fig. 3. First, the traffic is generated by the Anritsu Network Master Pro MT1000A [33], which supports several interfaces in order to generate traffic for the switch under test. Notably, the tester generates up to 10 Gbit/s and 100 Gbit/s traffic using the SFP+ and QSFP28 interfaces from the traffic generator. The tester is directly connected to the first P4-programmable switch (APS BF2556X-1T [13]) that will collect timestamps in every test. This switch is responsible for processing packets and collecting in-band data plane timestamps. Once the packet is sent to the egress port, it holds the original headers and payload, and all the timestamps collected en route through the pipeline, regardless of whether the measurements relate to the ingress or egress part. This way, timestamp collection remains consistent across all tests in this paper. The reason for adding a second switch (also an APS BF2556X-1T) is to ensure that packet sampling and encapsulation are executed only in S2 without affecting the performance and latency of S1 (switch under test). Since packet sampling and encapsulation require additional processing, this process shall not affect the first switch. Accordingly, it does not affect the timing results in any way. Lastly, a UDP collector receives the sampled packets and exports relevant statistics based on the in-band timestamps.

The traffic from the test results and predictions in Sections 6 and 7 follows the treatment depicted in Fig. 3. The traffic received at the switch under test (via SFP+ or QSFP28 interfaces) is processed, and timestamps are appended. The second switch samples the traffic, sending 10 million packets to the collector in each test, regardless of the port configuration. It applies a 10% and 1% traffic sampling to the input traffic running via 10G (SFP+) and 100G (QSFP28) port configurations, respectively.

5. Identifying key packet and data plane features

While data plane latency can be measured and tested for the ingress and egress pipeline, this paper focuses only on the ingress pipeline, which is conceptually similar to the egress one. Since both are architecturally equivalent (i.e., comprising a parser, a control block,

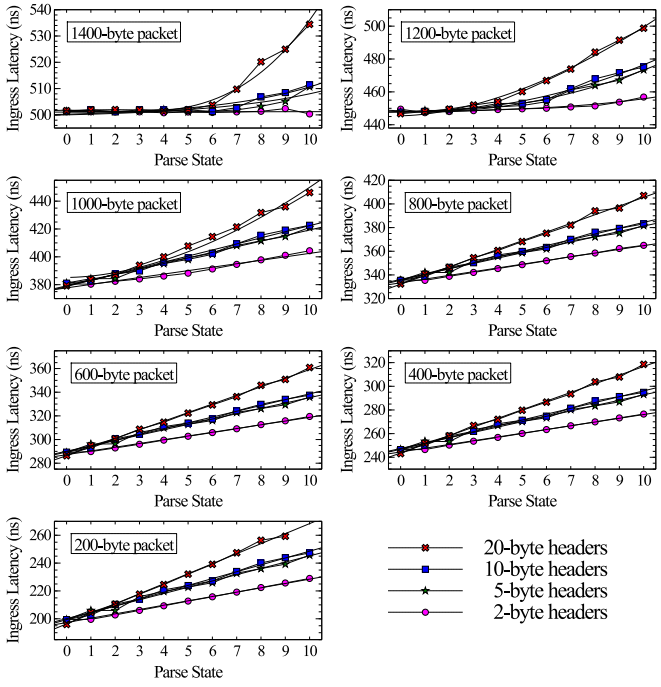


Fig. 4. Each figure shows a different per-packet-size latency trend when extracting headers of 2, 5, 10, and 20 bytes per state (Ethernet + 10 states). Ports are configured as 10G.

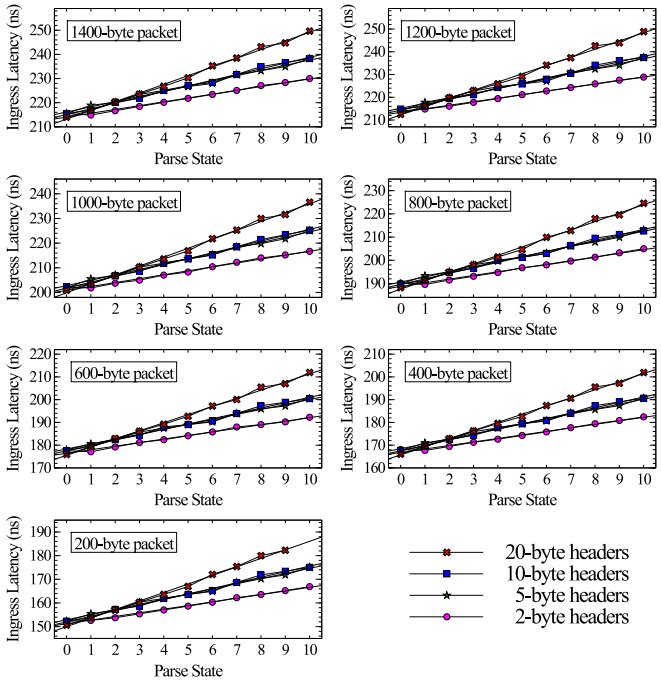


Fig. 5. Each figure shows a different per-packet-size latency trend when extracting headers of 2, 5, 10, and 20 bytes per state (Ethernet + 10 states). Ports are configured as 100G.

and a deparser), we do not consider egress pipeline latency ($L_{T_4 \to T_5}$) in our calculations. The depicted timestamps and latency values are always expressed as nanoseconds unless stated otherwise. Similarly, the latency measured in the ingress pipeline and plotted in the following sections refers to the average one calculated from each test so the models will also predict the average latency (arithmetic mean).

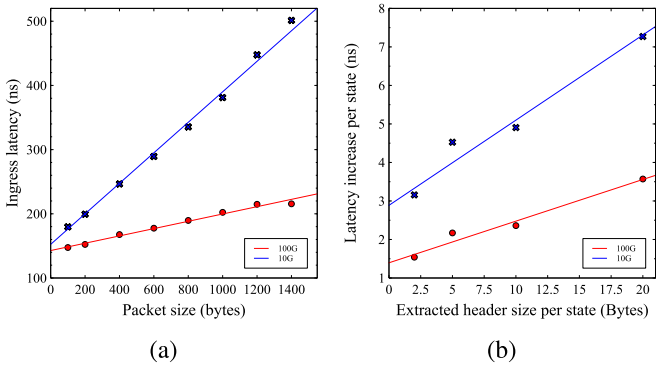


Fig. 6. Ingress latency in regard to (a) packet size (100 to 1400 bytes) and (b) header size extracted for traffic size running at 10G via SFP+ (blue) and 100G via QSFP28 (red).

The data plane latency experienced by packets traversing different parser graph paths, or matching against different tables, is variable. For instance, not all packets with the same headers experience the same latency. Considering this variability, the timestamps of 10 million packets are processed on each test to ensure the statistical validity of the results. From a superficial perspective, a packet can be modeled as a set of features like header types or total packet size. Therefore, the latency results from a function of those features, which is later analyzed in Section 6. Second, additional factors like data plane programming constructs can also contribute to the data plane latency, such as the parse graph structure, number of parse states, etc. Third and finally, configuration parameters of forwarding devices can also be determining factors. The following sections further explain the effects of the features mentioned in this paragraph.

5.1. Packet size

This paper shows that packet size is a determining feature. When measuring ingress pipeline latency ($L_{T_1 \to T_2}$), the time experienced by packets varies when packet size changes, as shown in Figs. 4 and 5. From the two figures mentioned before, Fig. 6(a) decouples the packet size influence from header parsing and deparsing. This latency increases linearly, which is a key factor in Section 6.

5.2. Parse states and extracted header size

As shown in Figs. 4 and 5, parse states do not seem to show a different latency increment in most cases. However, when ports are configured as 10G, packets larger than 1000 bytes show non-linear results. When parsing in this configuration, packets are parsed, and a few regressions show a non-linear latency increment is observed when more than five parse states are used. Therefore, the number of parse states and the extracted header size must be considered.

5.3. Tables and keys

The test conducted show that the increment of the number of matched tables has a linear impact on the latency in both *EXACT* and *LPM* match types (see Figs. 7(a) and 11(a)). In the *EXACT* match, the increment of latency is between one and three nanoseconds when we increment the key length or the number of entries (Figs. 8(a) and 10(a)). The *LPM* match shows a similar increment of latency when increasing the key length (see Fig. 11(a)).

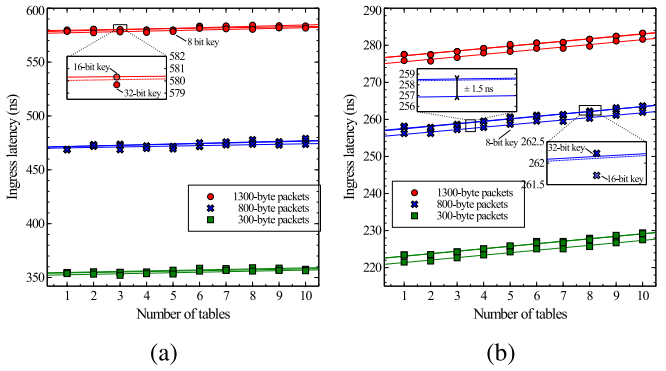


Fig. 7. Ingress latency using 8, 16, and 32-bit exact key match tables using 300, 800, and 1300-byte packets running at 10G (a) and 100G (b).

5.4. Actions

The performed tests only consider one type of action (set output port), which could make the model less accurate to predict data plane latency when other types of actions are used. However, the validation data set includes additional actions that have not been trained, such as modifying a header, writing registers, or arithmetic operations, to check the accuracy of the model in such cases.

5.5. Port configuration

As mentioned in the previous section, our preliminary tests show a higher latency for packets traversing the data plane at ports configured as 10G compared to 100G. Besides, Fig. 4 shows a non-linear latency increment, which demonstrates that the behavior is different in a few cases. It is essential to mention that when a figure or section states that a test was performed with ports configured at 10G, it means that the traffic generator worked at a rate nearly equal to 10 Gb/s (specifically, 9.5 Gb/s). The same applies to 100G (95 Gb/s). This difference accommodates new headers being appended to packets from the switch under test (S1) to the sampling switch (S2). And because the test design accommodates new headers added (timestamps), it is necessary to adjust the initial rate used for testing to prevent packets from being dropped. Similarly, we will accommodate that the timestamps are being added to the switch under test (S1). We will always understand that the extra time for timestamp addition exists but will always be accounted for, no matter which test.

6. Modeling data plane latency

The traffic features analyzed in the previous section allow the authors to examine and evaluate which models can be used to predict data plane latency. The preliminary results show linear latency progressions and a few non-linear results in parser and deparser tests at 10G ports. In order to achieve high accuracy, our research has considered several methods that can fit and predict the latency, such as linear regression (LR) and PCA-lasso-polynomial (PLP), which are later compared to the time measured in this section. In particular, we present a linear regression model derived from the results of the parser and deparser tests (Section 6.1) and also control block tests (Section 6.2). The additional ML algorithm (PLP) used for comparison are described in Section 6.3. All the test performed can be reproduced with the shared from the official Git repository [39].

6.1. Ingress parser and deparser

In the tests, each parse state considers only one header extraction. Most ingress latency results in 10G and 100G port configurations present linear increments with the number of parser states. Therefore, we present a linear regression-based prediction model for the parser and deparser block, and the control block, which are presented in different sections. We then compare the linear regression model prediction (combining the parser, deparser, and control blocks) to three additional ML algorithms in Section 7.

As seen in Fig. 2, the latency regarding the ingress pipeline includes three main parts, which refer to the parser, control block, and deparser. In this section, we focus only on the ingress pipeline because, as mentioned earlier, the egress pipeline is functionally equal to the ingress one, as both blocks share the same type and number of architectural blocks.

First, the ingress parser and deparser are modeled in one function, as observed in Eq. (4) ($L_{PD} = L_P + L_D$). As observed in Figs. 4 and 5, the baseline latency (Ethernet header parsing, $x = 0$) starts at a different latency unit (y-axis value). It means that the influence of the packet size is a factor on its own (introduced as L_B below) and can be decoupled from the latency influenced by the parser and deparser, and also the control block. As seen in Figs. 4 and 5, in most tests, the increment of latency per header extraction is linear. Besides, the influence of extracting the Ethernet header is considered to be our baseline. Therefore, the equation used in the current section (Section 6.1) and Section 6.2 to model the ingress pipeline latency is:

$$L_{T_1 \to T_2} = L_B + L_{PD} + L_C \tag{4}$$

where:

L_B is the baseline latency influenced by the packet size when Ethernet is extracted.

L_{PD} is the latency influenced by the ingress parser and deparser (i.e., headers extracted and deparsed).

L_C is the ingress control block latency.

The previous equation joins the parser and deparser latency as L_{PD} and considers L_C as the control block latency. When modeling, the L_{PD} latency, L_C is constant across all tests, and vice versa when modeling L_C. It is necessary to mention that L_B (latency influenced by packet size) affects the total pipeline latency results collected in Sections 6.1 and 6.2.

The parser and deparser tests comprise a total of 616 tests using 77 different pipelines, shown in Fig. 4/ Fig. 5 to Fig. 11(a)/ Fig. 11(b). The test results show, primarily, that traffic running at 10G and 100G port configurations experience a linear incremental trend as packet size and header size are increased. We separated their calculations to analyze the influence of the packet size and the header size (along with parse states) as independent terms.

Fig. 6(a) (10G, blue) and Fig. 6(a) (100G, red) express the linear regression of the ingress pipeline latency experienced by packets as a function of packet size. In this figure, the packet size feature is decoupled from any latency influenced by the parser and deparser.

Analyzing the results from Fig. 6(a) (10G, blue), the regression line of the latency as a function of the packet size (L_B), results in the model shown in Table 2 Eq. (1).

Additionally, the regression line in Fig. 6(b) (10G, blue) regarding the extracted header size per parse state is expressed as shown in Table 2 Eq. (2). When it comes to the latency in 100G, based on Fig. 6(a) (100G, red), the linear regression of the latency as a function of the packet size (L_B), results in the model shown in Table 2 Eq. (3). The regression line in Fig. 6(b) (100G, red) regarding the extracted header size per parse state is express as shown in Table 2 Eq. (4).

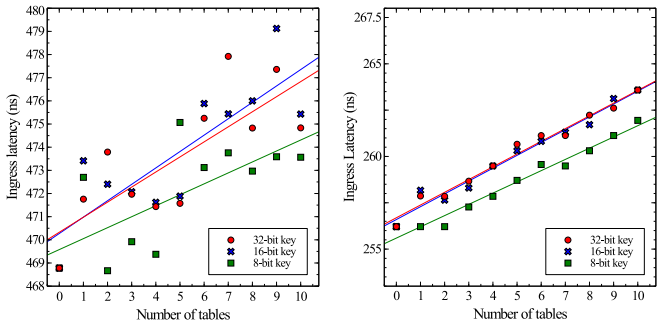


Fig. 8. Ingress latency: 8, 16 and 32-bit keys, 1300-byte packets and traffic running at 10G (a) and 100G (b).

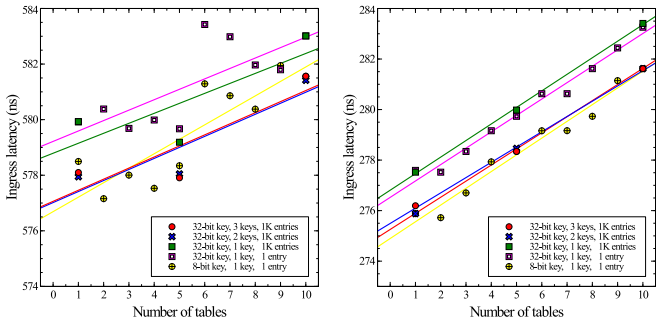


Fig. 9. Comparing multiple 32-bit exact match keys based on Figs. 8(a) and 10(a) running at 10G (a) and 100G (b).

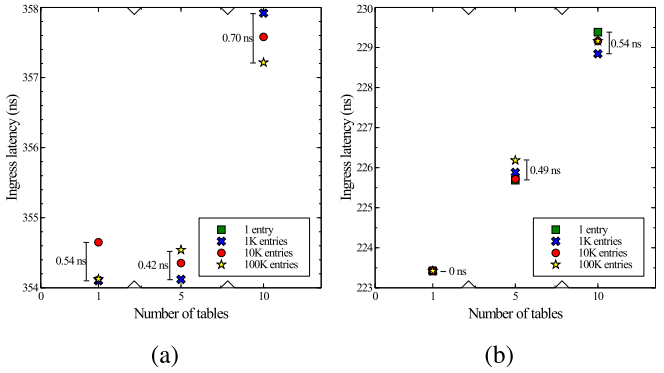


Fig. 10. Latency differences with 1, 1 K, 10 K, and 100 K, single 32-bit entry tables running at 10G (a) and 100G (b).

6.2. Ingress control block

In terms of the control block, the key features that we consider in our tests and that could affect the data plane latency are the same in both ingress and egress (e.g., tables, actions, etc.). For these reasons, we aim to measure the latency at the ingress control block, focusing on the number of tables, table key size, number of table keys, or table key type, to name a few. This section only considers one type of action in our test measurements. However, the model prediction validates its performance, in Section 7, using different types of actions associated with each test case (e.g., header modifications, register interactions, etc.) as described in Table 5.

The complete test set described and plotted in the following sections comprises 628 tests and 130 different pipelines. The same number of headers with a consistent size are extracted in all the tests. We first extract Ethernet, IPv4, and UDP headers, followed by four custom headers that include fields of size 2^n.

We consider several table features in the control block: *{number of tables across tests, table key size, table keys, number of entries and table key match types (exact and longest prefix match (LPM))}*. The action remains consistent across tests, and it is executed when the table is matched. The action assigns an output port for the packet.

In the tests depicted above in Fig. 7(a) and Fig. 7(b), we compare three different packet sizes (300, 800, and 1300 bytes) with an 8, 16, and 32-bit single exact matching key, matching against 1 to 10 tables. Fig. 7(a) and Fig. 7(b) show a comparison of the results for 10G and 100G port configurations, respectively. The regression lines in Fig. 7(a) show different coefficients. 8-bit table key results experience a lower latency compared to 16 or 32-bit table key results (which are similar). The results of the 100G (Fig. 7(b)) case are more consistent, and the regression lines still show a similar difference. In all our tests, we consider significant differences when they exceed one nanosecond. In this case, matching approximately two to three tables show a difference larger than 1 ns in both 10G and 100G cases. This means that the key size must be considered as a relevant feature when modeling the function to estimate the latency. Still, the regression lines are very similar across results that used different sizes of packets. At this point, we could estimate that packet size does not influence the control block. As mentioned in the previous section, it is a key feature that influences data plane latency and can be decoupled from parsing, deparsing, and control block-specific functions.

Fig. 8(a) (10G) and 8(b) (100G) show a specific case extracted from the previous figures, using 1300-byte packets. These figures show the ingress latency from 0 to 10 tables for 10G and 100G port configurations. The results show that 32-bit and 16-bit key tables experience around half a nanosecond higher latency per table than 8-bit key tables. Fig. 8(a) shows scattered and inconsistent results (compared to the 100G figure) across table increments without a clear per-table increment of latency. However, an increment exists when a regression line is calculated since the results show an increment when matching ten tables compared to none.

As seen in Fig. 9(a) and Fig. 9(b), when multiple exact keys are used, the regression results are similar to those of 8-bit single key tests. Surprisingly, 2 or 3 exact match keys experience slightly lower latency than the results of 32-bit single-key tests. More tests might be needed to confirm or deny these results.

Additional tests with EXACT rules showed that the number of entries declared and inserted for tables could affect the final latency to a certain extent. Fig. 10(a) and Fig. 10(b) compare the latency experienced by packets that matched 32-bit single key tables with 1, 1 K, 10 K, and 100 K entries. The difference between most of the measurements in both figures is equal to or lower than 0.7 ns. The declared tables with the highest number of entries do not necessarily experience higher latency measurements than tables with fewer entries. This demonstrates that packets do not experience a higher latency when more rules are inserted.

Drawing on all the tests in the current section, the prediction model shows the latency increment per table for different exact matching key sizes. As explained in the previous section, L_B was already modeled. Although L_B influences the data plane latency, it is possible to solely model L_C. Therefore, L_C can be modeled in this first equation with 10G and 100G configured ports as shown in Table 2 Eqs. (5) and (6).

When multiple keys are tested in our evaluation (see Section 7), the model for the 8-bit single key ($k = 8$) has to be used (since the results show a similar latency evolution). That is $0.492 \cdot t$ and $0.607 \cdot t$ for 10 and 100G port configurations, respectively. When keys larger or equal to 16 bits are evaluated, the model calculated for 16 or 32-bit keys ($k = 16, 32$) is used.

Table 2
Summary of the equation for the different modeled blocks using the linear regressions where p is the packet size, PS_n is the nth parse state, h_n is the size of the extracted nth header at the nth parse state, t is the number of tables and k is the matching key size ($k \in [8, 16, 32]$). A Subsection in the Appendix includes a generalized equation.

Port (GB)	Base latency(L_B)	Ingress parserand deparser latency (L_{PD})	Key type	Ingress control latency (L_C)
10	$0.237 \cdot p + 152.42$ (1)	$\sum_1^n PS_n = \sum_1^n 0.221 \cdot hs_n + 2.886$ (2)	EXACT	$f(t,k) = \begin{cases} 0.492 \cdot t, & k = 8 \\ 0.601 \cdot t, & k = 16, 32 \end{cases}$ (5)
			LPM	$f(t,k) = \begin{cases} 0.692 \cdot t, & k = 8 \\ 0.887 \cdot t, & k = 16, 32 \end{cases}$ (7)
100	$0.056 \cdot p + 143.441$ (3)	$\sum_1^n PS_n = \sum_1^n 0.105 \cdot hs_n + 1.431$ (4)	EXACT	$f(t,k) = \begin{cases} 0.607 \cdot t, & k = 8 \\ 0.685 \cdot t, & k = 16, 32 \end{cases}$ (6)
			LPM	$f(t,k) = \begin{cases} 0.749 \cdot t, & k = 8 \\ 0.896 \cdot t, & k = 16, 32 \end{cases}$ (8)

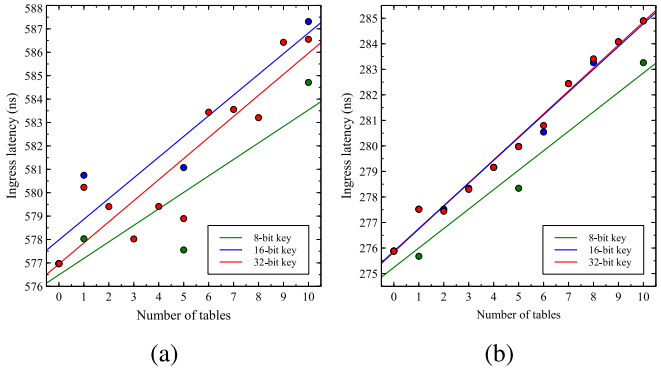


Fig. 11. Ingress latency for 8 (green), 16 (blue), and 32-bit (red) LPM keys, matched up to 10 tables. The results are divided into traffic running at 10G (a) and 100G (b).

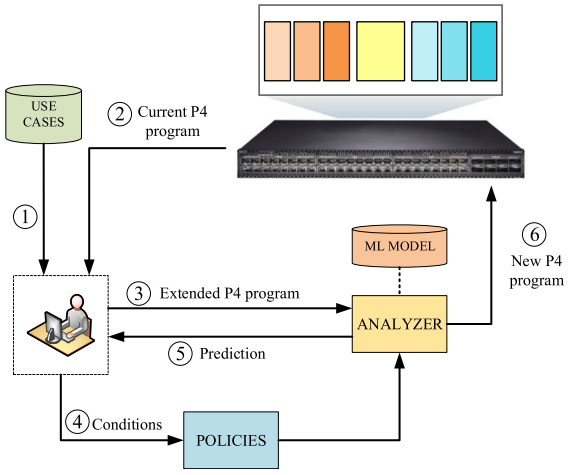


Fig. 12. Application scenario of prediction model.

In addition to exact match results, we have also tested LPM key-type tables. As the previous tests suggested, Fig. 11(a) (10G) and Fig. 11(b) (100G) show similar results to those found in Fig. 7(a) and Fig. 7(b). The difference between the 8-bit key size and the 16 and 32-bit key size is similar in both figures. Besides, Fig. 11(b) (100G) shows stable results with a clear increment in latency as more tables are matched. In fact, the latency appears to be equally scattered for 10G tests in exact and LPM-match key types.

Observing the results in the tests at Fig. 11(a) (10G) and Fig. 11(b) (100G), the following model has been created to predict latency results involving tables with LPM keys. Table 2 equations (7) and (8) refer to the 10G and 100G cases respectively.

Table 2 summarizes the models for every block that is used to predict the average data plane packet latency in Section 7.

6.3. ML prediction algorithms and use case definition

The results allow building a linear regression (LR) model as listed in Table 6, but the preliminary validation of the model shows less accuracy when more states with larger header sizes are parsed per state (e.g., the curve presents a non-linear increment when the packet size is longer than 800 bytes with the parser of 10G in Fig. 4). ML regression model is trained and tuned to predict the non-linear relationship between the packet features and latency, to be compared with the LR model proposed in the previous section.

PCA-lasso-polynomial (PLP) regression: Observing the experimental data, we realized that a few regressions seen in Fig. 4 show a non-linear relationship between the extracted header size, parser states, and packet size. For instance, there is a non-linear regression when: packet size is larger than 800 bytes (see Table 2 Eq. (1), $p > 800$), parse states are larger than 5 (see Table 2 Eq. (2), $n > 5$), and when header size is larger than 5 bytes (see Table 2 Eq. (2), $h_n > 5$). To fit these cases, a quadratic polynomial (polynomial of degree 2) regression is applied to fit the non-linear data.

Besides, our preliminary tests show that not all features have the same importance for latency prediction, and the potential correlation among the features can affect the prediction results. Therefore, principal component analysis (PCA) and Lasso regularization are applied to reduce the feature dimensionality and collinearity issues to avoid overfitting [40].

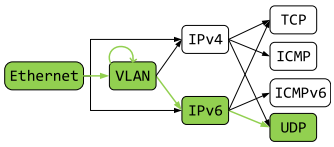
Among the model parameters, one of the key parameters that affects the PCA performance is the number of feature dimensions to keep *n_components* after the dimensionality reduction. One of the key parameters that affect the lasso performance is the coefficient *alpha* on the L1 regularization.

By introducing the PCA and Lasso, the polynomial regression can be more robust when polynomial features are correlated.

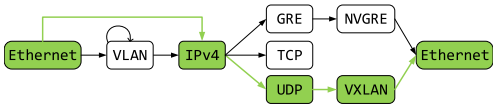
The PLP model is trained similarly to the LR algorithm. The training process is done separately to model the latency in parser and deparser and the control block based on datasets. A dataset is collected for model training and evaluation with the features summarized in Tables 7 and 8 in Appendix. This dataset is collected and sampled from 10 million packets at the second switch in the testbed. The Python library *sklearn* [41] is used for the ML models. The trained model is saved in the file system and can later be easily loaded for latency prediction. The key parameters of the well-trained PLP model are listed in Table 3.

6.4. Model deployment and use case

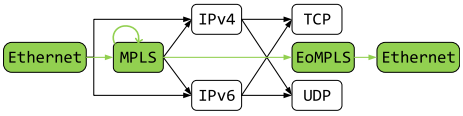
The LR and PLP model can be applied for latency prediction as demonstrated in Fig. 12. In this scenario, we consider the latency-critical applications as the use case. The programmable ASICs can be applied for use cases like latency-critical applications to optimize the



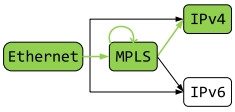
(a) Test case 1 (TC1): Enterprise use case



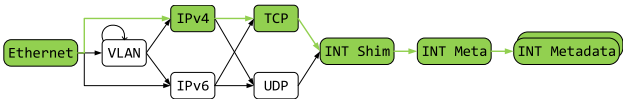
(b) Test case 2 (TC2): Data center use case



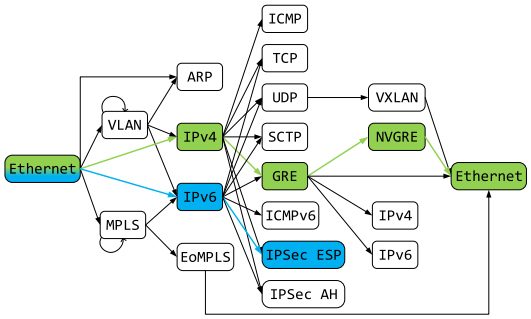
(c) Test Case 3 (TC3): Edge use case



(d) Test case 4 (TC4): Service provider use case



(e) Test case 5 (TC5): In-band Network Telemetry use case



(f) Test case 6 (TC6A, TCB6): Big Union use case

Fig. 13. 7 different cases tested (TC1, TC2, TC3, TC4, TC5, TC6A, TC6B). The green-colored parse states represent the path followed when extracting the parser's headers in each use case. The last graph includes two colored parse graph paths, representing two different tests for the same parser. The test cases are based on the ones proposed by Gibb et al. [28].

Table 3
Key Parameters in each ML Model.

Model	Parameter	Parser and deparser	Ingress control block
PLP	PCA n_components	10	20
	lasso alpha	650	0.0005
	polynomial degree	2	2

network performance and meet the stringent latency requirement [42]. For example, if a latency-critical application requires a specific latency metric (step ①), the developer has to optimize the current P4 code (step ②) to achieve the performance. Typically, the developer can only know the performance after compiling the code on the hardware switch, but further tests would require verification. Such a procedure entails the developer testing the code on the hardware switch frequently. Given that the developer has an extended P4 program as an extra design added to the current P4 program for optimization and the load conditions as pipeline policies and conditions, with the prediction model introduced into this process, the developer can send the extended P4 program (step ③) and the corresponding pipeline policies and conditions (step ④) to the predictor with the well-trained regression model. The predictor will send back an estimated latency performance to the developer (step ⑤), and the developer can improve the P4 code based on that. When the P4 code can achieve a satisfying latency performance, it is then ready to be run on the hardware switch (step ⑥).

Based on this application scenario, the prediction method can bring convenience to the hardware development process. It reduces the frequency of hardware verification and spares the hardware access limitation. It is beneficial, especially for those still at the prototyping stage and unwilling to purchase high-performance and expensive equipment.

7. Model validation

7.1. Parser and deparser predictions

After creating a model to determine parser and deparser latency (Section 6), it is necessary to test it with actual scenarios. Gibb et al. [28] present an in-depth parser design study that identifies parser design principles and discusses its trade-offs. We define the parse graphs in Fig. 13 on top of the ones found at Gibb's publication. Our parse graph includes an additional test case that parses the In-band Network Telemetry (INT) headers. To describe a few examples, TC2 includes virtualization headers like VXLAN or GRE that are typically used in data centers. Test cases like TC3 or TC4 are considered to be cases that label the traffic with the MPLS headers, such as in service provider networks or routing devices at the network edge. TC5 includes INT shim, meta and metadata headers, typically used when processing data plane telemetry in INT-MD mode. The actual parse graph traversed and measured in each test case is colored in green and has been randomly picked out of all the end-to-end possibilities. Fig. 13(f) has an additional parse graph traversed in blue. The reason for testing, first, parser-only cases is that we can measure whether the selected features are enough to predict the latency that affects the parser and deparser latency alone. Later, the control block is added to the prediction validation in order to validate the pipeline entirely. In our tests, $L_{T_0 \to T_1}$, $L_{T_2 \to T_3}$ and $L_{T_3 \to T_4}$ remain consistent. However, the queuing latency ($L_{T_2 \to T_3}$) could be further studied, since it could be modeled. However, queuing latency is consistent across all tests in this paper, so it imposes no influence when modeling ingress latency.

As seen in Table 4, we combine 10G and 100G tests, with a total of 7 cases (Case 1 to Case 6A/B). The tests were designed with random packet sizes, and as mentioned, parse graphs were also chosen randomly. The cases presented vary from 2 to 7 parse states (without considering the first Ethernet parse state). The majority of the packet sizes (300, 700, 900, 1100, and 1300 bytes—excludes 200 and 600) and header sizes (4, 8, 12, 14, and 40 bytes—excludes 20 bytes) are different from the ones trained in Section 6.

Table 4
Parser and deparser features for each test case.

	PARSER FEATURES								
	PACKET SIZE	STATES	STATE 1	STATE 2	STATE 3	STATE 4	STATE 5	STATE 6	STATE 7
	Packet size (bytes)	Number	Ext. header (bytes)	Ext. header (bytes)	Ext. header (bytes)	Ext. header (bytes)	Ext. header (bytes)	Ext. header (bytes)	Ext. header (bytes)
Test case 1	600	4	Metadata + 14	4	4	40	8		
Test case 2	900	4	Metadata + 14	20	8	8	14		
Test case 3	700	5	Metadata + 14	4	4	4	4	14	
Test case 4	1300	6	Metadata + 14	4	4	4	4	4	20
Test case 5	1100	7	Metadata + 14	20	20	4	12	8	8
Test case 6A	200	4	Metadata + 14	20	4	4	14		
Test case 6B	300	2	Metadata + 14	40	8				

7.2. Control block predictions

In addition to the parser and deparser tests, the control block latency is the missing part that defines ingress pipeline latency and the end-to-end latency (since we do not consider egress pipeline latency in our tests). In this test, we use the linear regressions (noted as LR) described in Section 6.2, comparing the results to the PLP. Testing the ingress control block aims to propose cases that use tables and keys from the parsed headers. TC1 and TC6A have followed a similar table key and actions to the ones tested in Section 6.2. Other test cases (TC2, TC3, TC4, TC5, TC6B) have incorporated additional statements (header modification, register writing, etc.) that were not measured and are used as training features. The goal is to assess how much the untrained additional statements affect the latency prediction. This could be used as a future step in further testing to achieve a more precise latency prediction.

Table 6 shows the results for ingress pipeline latency, which is characterized by a dynamic packet size and the parser, control and deparser. Considering the 7 test cases and merging 10G and 100G cases, the PLP algorithm predicts the ingress latency with the highest certainty (98.22%). The linear regressions detailed in Section 6 achieves the second-best prediction results (certainty of 97.49%).

8. Discussion

8.1. Results and comparison

In this paper, we perform a latency profiling study of the Tofino P4 programmable ASIC-based hardware, and to further extend this analysis, a regression model is proposed based on the test results from the parser, deparser and control blocks. We have selected several relevant features for the relevant architecture blocks and obtained the results of the tests and trained several ML algorithms to compare their predictions against the LR model. This could serve as a reference for predicting the performance of latency-constrained application based on this type of common P4 target. To this purpose, our comparison in Table 6 shows that the PLP model obtains the best results overall, when predicting the parser, control and deparser latency. PLP is also the algorithm that obtains results with the slightest discrepancy. PLP achieves several results within the 1 ns error we defined in the beginning. And still achieve the 1% error accuracy in most of the results. We believe this happens because of the combination of algorithms (PCA-lasso-polynomial) that creates a more complex model that better accommodates the training data and adapts to verify untrained tests for the 10G results. We also believe that the 100G results are almost the same for every algorithm, having improvements in the decimals.

We observe in Table 6 that most predictions are worse when non-tested P4 statements in control blocks are validated (like header modifications or registers). It means that further tests are needed to understand other P4 statements, especially for 10G results. Still, our PLP predictions results show an accuracy rate of 98.22% across multiple use cases, demonstrating that, even when new features are added to test cases, our models achieve accurate latency predictions. In addition, we achieve higher accuracy compared to state-of-the-art estimation methods, such as Harkous et al. [32], whose authors can estimate the packet forwarding latency with an accuracy of around 94% for T4P4S, Netronome SmartNIC, and NetFPGA-SUME. Moreover, our methodology for measuring the packet processing latency is more accurate as it uses the internal timestamps inside the ASIC instead of measuring timestamps at the traffic generator using RTT-based estimations as shown Harkous et al. [32] or Scholz et al. [34]. v

It is necessary to mention that the LR model performs second-best in the results. The proposed model is derived directly from the figures shown in Section 6. It demonstrates that the behavior, in terms of latency prediction, can be expressed as a linear model and achieve accurate predictions. The linear behavior of the data plane latency and consistency of the traffic running at QSFP28 ports shows that the performance of the ASIC is suitable for use cases in production environments. Using the SFP+ interfaces will achieve, in a few cases, non-linear latency results that deteriorate the performance of traffic forwarding in production environments.

On the other hand, regarding the results observed in Fig. 4, our a analysis suggests that the non-linear results are not a consequence of the P4 parser or deparser, nor is it affected by the ASIC behavior. Instead, the non-linear results seem to be caused by the gearbox of the switch when the port configuration is set to 10G (using SFP+ interfaces). The non-optimal behavior when configuring ports at 10G is confirmed in Fig. 7(a) and Fig. 9(a). It is possible to observe that the latency results point to a non-linear regression both in parsing and deparsing cases but also in a few control block tests (see Fig. 8(a), Fig. 10(a) and Fig. 11(a)). The 100G results as observed in Fig. 9(b) point to a clear linear incremental trend of the experienced latency.

8.2. Use cases and applications

Finally, this paper shows a nanosecond-scale prediction accuracy. While it might seems excessive for certain applications, some other ones like Precision Time Protocol (PTP) standard [43], Time Sensitive Networking (TSN) [44], or CPRI and evolved CPRI (eCPRI) [45].

PTP is used to provide sub-microsecond time synchronization, a requirement for enabling deterministic communications [46]. It standardizes several types of clocks, including the Transparent Clock (TC), which corresponds to a device that neither provides a clock source nor updates their own, but is still capable of propagating PTP messages between end devices while adjusting the messages for its residence time. Hence, nanosecond-accurate latency estimation facilitates the implementation of a TC in the Tofino-based P4 switch without the need for timestamping.

Time-aware traffic shaping is a cornerstone feature of TSN and it involves the computation of a gate-based time schedule and its coordination across the network in order to enable bounded frame delivery times. Verticals such as industrial and automotive often have tight latency requirements in the order of few microseconds [47]. To accurately synchronize the time slots for the different types of traffic

Table 5
Control block features per test case. NM states for *no match*. NA states for *no action*. OP states for only *output port*. OP+MF states for *output port + modify field*. OP+AO+2 ·MF states for *output port + arithmetic operation + 2 modify field operations*. MF states for only *modify field*. REG states for *register* (write).

	CONTROL BLOCK FEATURES												
	TABLE 1		TABLE 2		TABLE 3		TABLE - ACTION TYPES (Num. tables as *T* - Num. actions as *A*)						
	Key size (bits) / Type	Table size (entries ins.)	Key size (bits) / Type	Table size (entries ins.)	Key size (bits) / Type	Table size (entries ins.)	NM	NA	OP	OP+MF	OP+AO +2MF	MF	REG
Test case 1	48 / exact 16 / exact	64	128 / exact 128 / exact	10 K	128 / LPM 16 / exact	1 K	1T - 1A	1T - 1A		1T - 1A			
Test case 2	32 / exact 32 / exact 16 / exact 16 / exact	5 K							1T - 1				
Test case 3	20 / exact	20 K								1T - 1A			
Test case 4	20 / exact	1 K	128 / LPM	1 K					1T - 1A		1T - 1A		
Test case 5	32 / exact 32 / exact	50	4 / exact	1 K					1T - 1A			1T - 3A	
Test case 6A	48 / exact 48 / exact	5 K	32 / exact 24 / exact	1 K				1T - 1A	1T - 1A				
Test case 6B	128 / LPM	30 K	32 / exact	1.5 K					1T - 1A				1T - 1A

Table 6
This table shows parser+control+deparser results (Ingress block). The actual and real latency is expressed as "Measured (ns)" (first column) and it is defined in nanoseconds, within the "Real time" column. One can also observe the "1 ns limit (%)" and "1% limit (%)" columns, next to the "Measured (ns)" one. Both columns express the 1 ns error and also the 1% error. The rest of the columns (LR, PLP) are predictions. The ingress pipeline latency results shown by the Linear Regressions (LR) are explained in Section 6. PLP states for PCA-lasso-polynomial. One can also observe that adding the value in the column "Precision (%)" and the one in the "Deviation (%)" (in all predictions), we get a 100. The results that are lower (or equal) than 1% are expressed in yellow ( ). Those that are both lower (or equal) than 1% and also 1 ns, are expressed in light blue ( ). Finally, we express the combined results (ALL CASES) in green color ( ).

		Parser+Control+Deparser test - Ingress Latency								
		Real time			Linear Regression (LR)			PCA-lasso-polynomial (PLP)		
	Port speed (G)	Measured (ns)	1 ns limit (%)	1% limit (%)	Prediction (ns)	Precision (%)	Deviation (%)	Prediction (ns)	Precision (%)	Deviation (%)
Test case 1 (TC1)	10	313.926	0.318	3.139	318.926	98.408	1.592	312.486	99.542	0.458
Test case 1 (TC1)	100	189.082	0.528	1.890	188.751	99.825	0.175	183.847	97.231	2.769
Test case 2 (TC2)	10	376.01	0.265	3.760	388.886	96.576	3.424	383.294	98.063	1.937
Test case 2 (TC2)	100	199.254	0.501	1.992	204.945	97.143	2.857	199.074	99.909	0.091
Test case 3 (TC3)	10	326.292	0.306	3.262	339.825	95.853	4.147	334.191	97.579	2.421
Test case 3 (TC3)	100	187.358	0.533	1.873	193.041	96.967	3.033	195.574	95.614	4.386
Test case 4 (TC4)	10	472.113	0.212	4.721	487.497	96.742	3.258	470.235	99.603	0.397
Test case 4 (TC4)	100	224.233	0.445	2.242	229.189	97.789	2.211	226.932	98.796	1.204
Test case 5 (TC5)	10	425.988	0.235	4.259	447.049	95.055	4.944	431.486	98.709	1.291
Test case 5 (TC5)	100	215.226	0.465	2.152	221.355	97.151	2.848	208.468	96.860	3.140
Test case 6A (TC6A)	10	220.597	0.453	2.205	220.781	99.916	0.083	219.937	99.702	0.298
Test case 6A (TC6A)	100	161.946	0.617	1.619	164.830	98.218	1.781	163.071	99.305	0.695
Test case 6B (TC6B)	10	244.301	0.409	2.443	240.09	98.279	1.721	250.212	97.580	2.420
Test case 6B (TC6B)	100	173.441	0.576	1.734	168.212	96.985	3.014	167.527	96.591	3.409
10G						97.262	2.738		98.682	1.318
100G						97.726	2.274		97.758	2.242
ALL CASES						97.494	2.506		98.221	1.779

Table 7
Summary of dataset details: parser block.

Feature	Data type	Range	Description
Port config (Gb)	Integer	[10, 100]	Port rate
Extracted header size (Byte)	Integer	[2, 5, 10, 20]	Size of the header to be extracted in the parser
Extracted header times	Integer	0–10	Number of times when header is extracted
Packet size (Byte)	Integer	[100, 200, 400, 600, 800, 1000, 1200, 1400]	Total size of packet
Latency (ns)	Float	140-540	Latency obtained from timestamps

across a TSN network, being able to estimate the delay introduced in the device with sub-microsecond accuracy is critical for implementing synchronous traffic shaping. This work could constitute an starting point for its implementation in Tofino-based P4 targets.

The CPRI and eCPRI specifications for transport networks including 5G impose some tight requirements on latency to the most critical traffic, which is 100 microseconds [45]. One the one hand, TSN can be used in Ethernet-based fronthaul networks [48] as well as to enable a converged Xhaul network [49]. On the other hand, some works have explored P4-based transport networks [50]. Thus, nanosecond-accurate latency estimation in Tofino-based P4 switches could have applicability in this line.

9. Conclusion

The current paper describes a detailed study on how data plane latency would vary with different features programmed in a programmable Tofino-based switch. This paper introduces a reproducible and extensible methodology to process the data plane latency that packets experience and verify the paper's results. Compared to the related work, our P4 programs collect in-band timestamps for later latency measurements. Traffic running on SFP+ (~10Gb/s) and QSFP28 (~100 Gb/s) interfaces are separately tested. The paper studies how different port speeds might affect latency performance. A dataset is collected with different block features configured in parser, deparser and control block.

To model the latency performance, we present and discuss linear regressions and ML algorithm PLP to perform data plane latency predictions. The regressions calculated from test results serve as the baseline to illustrate how latency would vary with different functions programmed in the pipeline. ML algorithm is used to accommodate the non-linear regressions observed in some measurements. Six common-used test cases are presented to test the model prediction results in order to evaluate the model's performance. The results show that linear regression-based calculations and PLP achieves an 98.22% of accuracy. It shows that the PLP algorithm can predict, more accurately, new control blocks.

With the prediction method proposed in this work, the researchers can evaluate their design without the ASIC-based hardware. This work highlights how latency prediction can be applied in designing and verifying time-critical applications. The test results open future research possibilities, suggesting that a new test design should accommodate new features included in the validation of the model but not considered in the training process (like multiple field modifications or register interactions). In addition, researchers could extend the work by modeling egress pipeline latency. While it is conceptually the same block as the ingress pipeline, the models that define the egress pipeline might differ from those presented in this paper.

CRediT authorship contribution statement

David Franco: Conceptualization, Formal analysis, Methodology, Validation, Writing – original draft, Writing – review & editing. **Eder Ollora Zaballa:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing. **Mingyuan Zang:** Conceptualization, Data curation, Formal analysis, Software, Visualization, Writing – original draft, Writing – review & editing. **Asier Atutxa:** Conceptualization, Writing – original draft. **Jorge Sasiain:** Conceptualization, Writing – original draft. **Aleksander Pruski:** Funding acquisition, Supervision, Validation. **Elisa Rojas:** Conceptualization, Investigation, Methodology, Project administration, Validation, Writing – original draft, Writing – review & editing. **Marivi Higuero:** Investigation, Supervision, Validation. **Eduardo Jacob:** Funding acquisition, Supervision, Validation.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Elisa Rojas reports financial support was provided by Spanish Ministry of Science and Innovation. David Franco reports financial support was provided by Spanish Ministry of Science and Innovation. David Franco reports financial support was provided by Basque Government. Elisa Rojas reports financial support was provided by Community of Madrid.

Acknowledgments

Manuscript created in August, 2023. Thanks to Vladimir Gurevich, a reference in all our questions. This work was developed by DTU Electro from the Technical University of Denmark (DTU), the University of the Basque Country (EHU) and the University of Alcalá (UAH). This research has been supported by the Spanish Ministry of Science and Innovation under the "Towards zero touch network and services for beyond 5G (TRUE5G)" PID2019-108713RB-C54, and project "ONENESS" (PID2020-116361RA-I00). It has also been supported by the Basque Government through the "B-INDUSTRY5G" ELKARTEK 21/14 KK-2021/00026, and by grants from Comunidad de Madrid through project MistLETOE-CM (CM/JIN/2021-006).

Appendix

Generalized formula

Eqs. (5) and (6) show the generalized formula for 10G and 100G respectively.

For 10G:

$$152.42 + 0.237 \cdot p + \sum_{1}^{n} 0.221 \cdot hs_n + 2.886 + 0.492 \cdot t_e + 0.601 \cdot t_{e'} + 0.692 \cdot t_l + 0.887 \cdot t_{l'} \quad (5)$$

For 100G:

$$143.441 + 0.056 \cdot p + \sum_{1}^{n} 0.105 \cdot hs_n + 1.431 + 0.607 \cdot t_e + 0.685 \cdot t_{e'} + 0.749 \cdot t_l + 0.896 \cdot t_{l'} \quad (6)$$

where:

t_e is the number of exact tables with $k \in [8]$

$t_{e'}$ is the number of exact tables with $k \in [16, 32]$.

t_l is the number of LPM tables with $k \in [8]$

$t_{l'}$ is the number of LPM tables with $k \in [16, 32]$.

Table 8
Summary of dataset details: control block.

Feature	Data type	Range	Description
Number of tables	Integer	1–10	Number of M+A tables in control block
Match type	Integer	[0, 1]	Type of match: Exact/LPM
Key size	Integer	[8, 16, 32]	Size of the key in M+A tables
Number of keys	Integer	[1, 2, 3]	Number of keys in M+A tables
Number of entries	Integer	[1, 1000, 10000, 100000]	Number of entries in M+A tables
Port config (Gb)	Integer	[10, 100]	Port rate
Latency (ns)	Float	0–10	Latency obtained from the timestamps

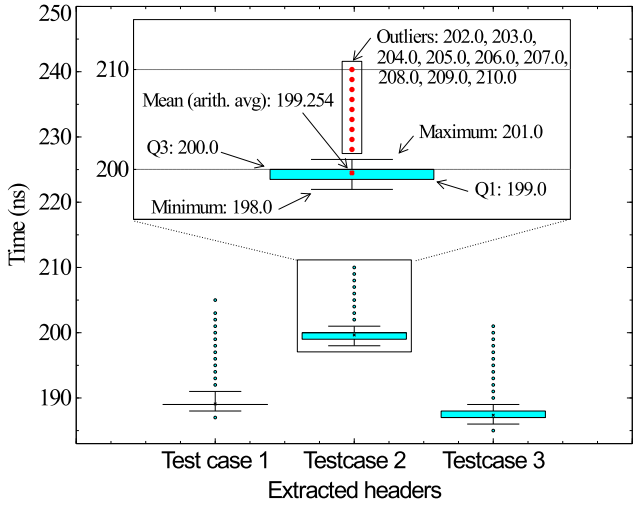


Fig. 14. Explanation of the box plot that is part of one of the 100G parser (Test case 2).

Dataset

Tables 7 and 8 show the summary of dataset details for parser and control blocks respectively.

Ingress block latency in 10G and 100G traffic

See Figs. 14–16.

Notation list

Table 9 depicts the notation list for this paper.

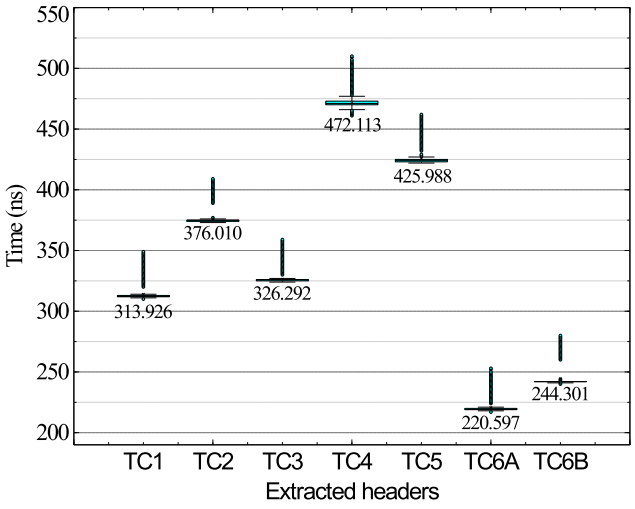


Fig. 15. Explanation of the 10G Ingress pipeline.

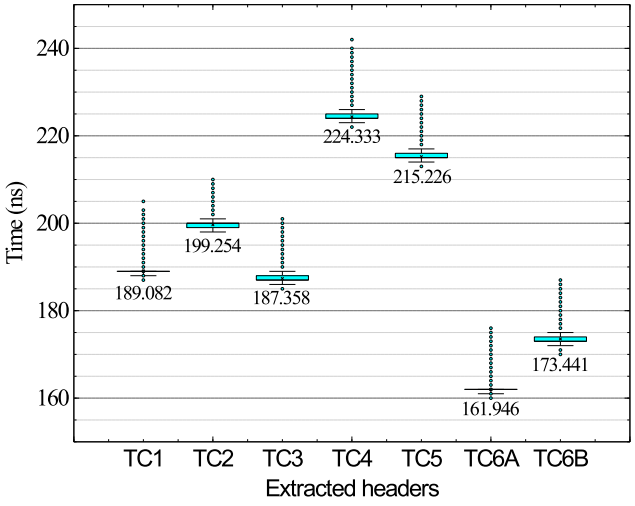


Fig. 16. Explanation of the 100G Ingress pipeline.

Table 9
Notation of the architecture.

Notation	Definition
T_i	The ith timestamp exposed by the target
L	Latency
T_{IP}	Timestamp at the end of ingress parser
T_{IC}	End of ingress control block
T_{EP}	End of egress parser
T_{EC}	End of egress control block
L_P	Parser latency
L_C	Control block latency
L_D	Deparser latency
B_L	Base latency associated with the packet size
L_{PD}	Latency comprising the parser and deparser
p	Packet size
$p_{\min}$	Supported minimum packet size
$p_{\max}$	Supported maximum packet size
PS_n	The nth parse state latency
h_n	The nth header size extracted at n parse state
$h_{\min}$	Supported minimum header size to extract
$h_{\max}$	Supported maximum header size to extract
k	Matching key size
t	Number of tables

References

[1] ONF, Software-Defined Networking (SDN) Definition, 2021, URL: https://opennetworking.org/sdn-definition/.

[2] O. Michel, R. Bifulco, G. Rétvári, S. Schmid, The programmable data plane: Abstractions, architectures, algorithms, and applications, ACM Comput. Surv. 54 (4) (2021) http://dx.doi.org/10.1145/3447868.

[3] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, M. Menth, A survey on data plane programming with P4: Fundamentals, advances, and applied research, J. Netw. Comput. Appl. 212 (2023) 103561, http://dx.doi.org/10.1016/j.jnca.2022.103561, URL: https://www.sciencedirect.com/science/article/pii/S1084804522002028.

[4] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, D. Walker, P4: Programming protocol-independent packet processors, SIGCOMM Comput. Commun. Rev. 44 (3) (2014) 87–95.

[5] P4 community, Behavioral Model version 2 (bmv2). The reference P4 software switch, 2021, GitHub repository. https://github.com/p4lang/behavioral-model.

[6] Netronome, Agilio CX SmartNICs, 2021, URL: https://www.netronome.com/products/agilio-cx/.

[7] Pensando, Distributed Services Card (DSC), 2021, URL: https://pensando.io/new-product-dsc/.

[8] NetFPGA, NetFPGa SUME, 2021, URL: https://netfpga.org/NetFPGA-SUME.html.

[9] Intel, Intel Tofino. P4-programmable Ethernet switch ASIC that delivers better performance at lower power , 2021, URL: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

[10] Intel, Intel Tofino 2. Second-generation P4-programmable Ethernet switch ASIC that continues to deliver programmability without compromise, 2021, URL: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html.

[11] E. Ollora Zaballa, D. Franco, E. Jacob, M. Higuero, M.S. Berger, Automation of modular and programmable control and data plane sdn networks, in: 2021 17th International Conference on Network and Service Management, CNSM, 2021, pp. 1–5.

[12] H. Soni, M. Rifai, P. Kumar, R. Doenges, N. Foster, Composing dataplane programs with μ P4, in: Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication, SIGCOMM '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 329–343, http://dx.doi.org/10.1145/3387514.3405872.

[13] Advanced Programmable Switches (APS), BF2556X-1T Advanced Programmable Switch, APS Networks, URL: https://www.aps-networks.com/products/bf2556x-1t/.

[14] P4 community, P4-16 declaration of the P4 v1.0 switch model from the reference compiler for the $P4_{16}$ programming language, 2013, GitHub repository. https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4.

[15] The P4.org Architecture Working Group, $P4_{16}$ Portable Switch Architecture (PSA), P4.org, URL: https://p4.org/p4-spec/docs/PSA.html.

[16] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, Openflow: Enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev. 38 (2) (2008) 69–74, http://dx.doi.org/10.1145/1355734.1355746, https://doi.org/10.1145/1355734.1355746.

[17] ONF TR 521 Issue 1.1 - SDN Architecture 1.1, SDN Architecture 1.1, Open Networking Foundation, Palo Alto, CA, USA, 2014, https://opennetworking.org/wp-content/uploads/2014/10/TR-521_SDN_Architecture_issue_1.1.pdf.

[18] ONF, OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06), Technical Report, Open Networking Foundation, 2015, URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[19] ONF, ONF SDN Evolution, Technical Report, Open Networking Foundation, 2016, URL: https://opennetworking.org/wp-content/uploads/2013/05/TR-535_ONF_SDN_Evolution.pdf.

[20] ONF/P4.org, P4Runtime Specification, Technical Report, The P4.org API Working Group, 2020, URL: https://p4.org/p4-spec/p4runtime/v1.3.0/P4Runtime-Spec.html.

[21] ONF/P4.org, The P4 Language Specification Version 1.0.5, Technical Report, The P4 Language Consortium, 2018, URL: https://p4.org/p4-spec/p4-14/v1.0.5/tex/p4.pdf.

[22] ONF/P4.org, $P4_{16}$ Language Specification version 1.2.2, Technical Report, The P4 Language Consortium, 2021, URL: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html.

[23] INTEL, $P4_{16}$ Intel® Tofino™ Native Architecture – Public Version, 631348–0001, INTEL, 2021, Application Note – Public Version 2. Available at https://github.com/barefootnetworks/Open-Tofino.

[24] Nate Foster (Cornell), P4: Types and parsers, 2019, URL: https://cornell-pl.github.io/cs6114/lecture05.html.

[25] S. Ibanez, C. Kim, CS344 stanford: Build an internet router (Lecture 2 P4 language overview), 2020, URL: https://cs344-stanford.github.io/lectures/Lecture-2-P4-tutorial.pdf.

[26] Barefoot - INTEL, Tofino 1 Architecture Base in P4, github.com, URL: https://github.com/barefootnetworks/Open-Tofino/blob/5a805ae468444b561ea8f66f0f46ebbe5d3dce41/share/p4c/p4include/tofino1_base.p4.

[27] INTEL, $P4_{16}$ Programming for Intel®Tofino™using Intel P4 Studio™, opennetworking.org, URL: https://opennetworking.org/wp-content/uploads/2021/05/2021-P4-WS-Vladimir-Gurevich-Slides.pdf.

[28] G. Gibb, G. Varghese, M. Horowitz, N. McKeown, Design principles for packet parsers, in: Architectures for Networking and Communications Systems, 2013, pp. 13–24, http://dx.doi.org/10.1109/ANCS.2013.6665172.

[29] H.T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, H. Weatherspoon, Whippersnapper: A p4 language benchmark suite, in: Proceedings of the Symposium on SDN Research, 2017, pp. 95–101.

[30] H. Harkous, M. Jarschel, M. He, R. Priest, 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, http://dx.doi.org/10.1109/ANCS.2019.8901881.

[31] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, G. Carle, MoonGen: A scriptable high-speed packet generator, in: Proceedings of the 2015 Internet Measurement Conference, IMC '15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 275–287, http://dx.doi.org/10.1145/2815675.2815692, URL: https://doi-org.proxy.findit.dtu.dk/10.1145/2815675.2815692.

[32] H. Harkous, M. Jarschel, M. He, R. Pries, W. Kellerer, P8: P4 with predictable packet processing performance, IEEE Trans. Netw. Serv. Manag. (2020) 1, http://dx.doi.org/10.1109/TNSM.2020.3030102.

[33] Anritsu, Network Master Pro (Ethernet/CPRI/OTDR Test Equipment), Anritsu, URL: https://www.anritsu.com/en-us/test-measurement/products/mt1000a.

[34] D. Scholz, H. Stubbe, S. Gallenmüller, G. Carle, Key properties of programmable data plane targets, in: 2020 32nd International Teletraffic Congress (ITC 32), 2020, pp. 114–122.

[35] Eötvös Loránd University, T_4P_4S, a multitarget P416 compiler. Retargetable compiler for the P4 language, 2021, GitHub repository.

[36] Z. Qian, D. Juan, P. Bogdan, C. Tsui, D. Marculescu, R. Marculescu, A support vector regression (SVR)-based latency model for Network-on-Chip (NoC) architectures, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 35 (3) (2016) 471–484, http://dx.doi.org/10.1109/TCAD.2015.2474393.

[37] Z. Qian, D.C. Juan, P. Bogdan, C.Y. Tsui, D. Marculescu, R. Marculescu, SVR-noc: A performance analysis tool for network-on-chips using learning-based support vector regression model, in: 2013 Design, Automation Test in Europe Conference Exhibition, DATE, 2013, pp. 354–357, http://dx.doi.org/10.7873/DATE.2013.083.

[38] L. Wang, Y. Zhang, X. Chen, R. Jin, Online computation performance analysis for distributed machine learning pipelines in fog manufacturing, in: 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE), 2020, pp. 1628–1633, http://dx.doi.org/10.1109/CASE48305.2020.9216979.

[39] E. Ollora Zaballa, D. Franco, M. Zang, Official P4 latency prediction test and dataset repository, 2022, Gitfront repository. https://gitfront.io/r/user-9379215/MesTSFgstja6/TNSM_Latency_Prediction/.

[40] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer New York, 2009, pp. 605–615.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[42] F. Paolucci, F. Cugini, P. Castoldi, T. Osiński, Enhancing 5G SDN/NFV edge with P4 data plane programmability, IEEE Netw. 35 (3) (2021) 154–160, http://dx.doi.org/10.1109/MNET.021.1900599.

[43] IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008), 2020, pp. 1–499, http://dx.doi.org/10.1109/IEEESTD.2020.9120376.

[44] IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic, IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015), 2016, pp. 1–57, http://dx.doi.org/10.1109/IEEESTD.2016.8613095.

[45] CPRI Cooperation, Common public radio interface: eCPRI interface specification, 2019, eCPRI specification V2.0. http://www.cpri.info/downloads/eCPRI_v_2.0_2019_05_10c.pdf.

[46] P. Pandey, B. Pratap, R.S. Pandey, Analysis and design of precision time protocol system based on IEEE1588 standards, in: 2019 International Conference on Communication and Electronics Systems, ICCES, IEEE, 2019, pp. 1963–1967.

[47] Use Cases IEC/IEEE 60802 V1.3, IEC/IEEE, URL: https://www.ieee802.org/1/files/public/docs2018/60802-industrial-use-cases-0918-v13.pdf.

[48] IEEE Standard for Local and Metropolitan Area Networks – Time-Sensitive Networking for Fronthaul, IEEE Std 802.1CM-2018, 2018, pp. 1–62, http://dx.doi.org/10.1109/IEEESTD.2018.8376066.
[49] R.M. Rao, M. Fontaine, R. Veisllari, A reconfigurable architecture for packet based 5G transport networks, in: 2018 IEEE 5G World Forum, 5GWF, IEEE, 2018, pp. 474–477.
[50] Y.B. Lin, C.C. Tseng, M.H. Wang, Effects of transport network slicing on 5G applications, Future Internet 13 (3) (2021) 69.

David Franco received his M.Sc. degree in Telecommunication Engineering from the University of the Basque Country (UPV/EHU) in 2018. He joined the Communications Engineering Department of the UPV/EHU as a researcher in the I2T (Engineering and Research on Telematics) research lab in 2016. His research is focused on software-defined networking and network function virtualization applied to distributed and secure communication systems. He is a current Ph.D. student in Mobile Network Information and Communication Technologies at the UPV/EHU.

Eder Ollora Zaballa obtained his B.Sc. at the University of the Basque Country (EHU) and the M.Sc. at the Technical University of Denmark (DTU). He also received his Ph.D. degree in the Network Technologies and Service Platforms group at DTU. He has previously been involved in the European project NGPaaS, Bandwidth-on-Demand (GEANT) and X2Rail. His research interest focuses on control and data plane programming (ONOS, Openflow, P4, and P4Runtime), distributed control planes, slicing, telemetry and network security in SDN networks. As a Postdoc at the same university (DTU), he is currently involved with P4 programming at a later project with GEANT.

Mingyuan Zang is a Ph.D. student at the Department of Photonics Engineering, Technical University of Denmark. She received her Master's Degree in Telecommunication at Technical University of Denmark and Bachelor's Degree in Electronic Information Engineering at Beijing University of Technology. Her work is focused on reliable and trustworthy IoT networks. Her research interests are: software-defined networking, programmable networks, network security and the Internet of Things.

Asier Atutxa received his B.Sc. degree in Telecommunication Engineering in 2018 and his MSc degree in Telecommunication Engineering in 2020 from the University of the Basque Country (UPV/EHU). He joined the Communications Engineering Department of the UPV/EHU as a researcher in the I2T Research Lab (Engineering and Research on Telematics) in 2018. His research interests include software-defined networking, data plane programming and virtualization. Currently, he is a Ph.D. student in Mobile Network Information and Communication Technologies at the UPV/EHU.

Jorge Sasiain received his M.Sc. degree in Telecommunication Engineering from the University of the Basque Country (UPV/EHU) in 2020. He joined the Communications Engineering Department of the UPV/EHU as a researcher in the I2T (Engineering and Research on Telematics) research lab in 2018. His research interests include network functions virtualization (NFV), software-defined networking (SDN), mobile edge computing (MEC) and their applications in 5G networks and industrial environments.

Aleksander Pruski is an industrial Ph.D. Researcher at Comcores ApS and Technical University of Denmark (DTU). He received the BEng EE and CS degree from the Technical University of Lodz in 2015, and then the M.Sc. degree in Telecommunication from DTU in 2017. Since then, he has been working at Comcores as Systems/Digital Design Engineer with TSN Ethernet and switching. He enrolled in the Ph.D. programme in 2018, focusing on experimental evaluation of TSN features and TSN switch design.

Elisa Rojas (PhD'13) received her Ph.D. degree in Information and Communication Technologies engineering from the University of Alcalá, Spain, in 2013. As a postdoc, she worked in IMDEA Networks and, later on, as CTO of Telcaria Ideas S.L. She has participated in diverse projects funded by the EC, such as FP7-NetIDE or H2020-SUPERFLUIDITY. She currently works as an Assistant Professor in the University of Alcaá, where her current research interests encompass SDN, NFV, IoT routing, and high-performance Ethernet and data center networks.

Marivi Higuero received the B.S. and M.S. degrees in electrical engineering, and the Ph.D. degree from the University of the Basque Country (UPV/EHU), in 1992 and 2005, respectively. She was with Sarenet, an Internet Service Provider, as a member of the technical department. She is currently an Assistant Professor with the Communications Engineering Department, UPV/EHU, where she is also a member of the I2T Research Laboratory. Her research interests include computer networks and services, sensor environments, mobility, and security.

Eduardo Jacob received the B.Sc. degree in industrial engineering and the M.Sc. degree in industrial communications and electronics in 1987 and 1991, respectively, and the Ph.D. degree in communications engineering from the University of the Basque Country (UPV/EHU) in 2001. He worked two years in a public research and development telecommunications enterprise (currently Tecnalia). He had a management role for several years as IT Director in the private sector. Since 1994 he started to work full time at the Faculty of Engineering in Bilbao, University of the Basque Country, where he was the first Head of the Department of Communications Engineering from 2012 to 2016. He is currently a Full Professor at the same university. He also leads the I2T (Engineering and Research on Telematics) Research Laboratory. He also directed several Ph.D. theses and managed several research projects at local, national, and European levels. His research interests include applying software-defined networks to industrial communications, cybersecurity in distributed systems, software-defined wireless sensor networks, and in-network processing.