Key Properties of Programmable Data Plane Targets

Dominik Scholz, Henning Stubbe, Sebastian Gallenmüller, Georg Carle
Chair of Network Architectures and Services, *Technical University of Munich*, Munich, Germany
{scholz,stubbe,gallenmu,carle}@net.in.tum.de

***Abstract*—We currently see a shift from fixed-function network devices with limited configurability towards network devices with a fully programmable processing pipeline. A prominent example of this development is P4 that provides a language and reference architecture model to design and program network devices. The core element of this reference model is the programmable match-action table that defines the processing steps for the network packets. In this paper, we demonstrate that these tables, which we use to create our own modeling framework, are the key driver of device performance.**

P4-programmable devices come in a wide variety regarding their underlying hardware architecture, such as CPU-based systems or ASICs, as representatives of both ends of the spectrum. CPU-based P4 target platforms offer limited performance but are easily extensible. ASIC P4 targets have dedicated P4 processing pipelines with limited programmability but offer highly optimized performance. To reflect these fundamental differences, our modeling framework incorporates different approaches to accurately model and predict the performance of P4-enabled devices.

***Index Terms*—Data Plane Programming, P4, Key Performance Indicator, Model, Device Benchmarking**

I. Introduction

In 2014, Bosshart et al. [1] introduced P4, a domain-specific language (DSL) for software-programmable networking equipment. Subsequently, a variety of hard- and software devices supporting P4 emerged. Given the appropriate P4 program, such a device can assume almost any packet processing task, only limited by its specific hardware capabilities. This paper analyzes the performance, i.e., latency, jitter, throughput, and resource consumption, of such highly flexible and extensible packet processing systems.

Dang et al. [2] and Rotsos et al. [3] use benchmarks to determine the performance of network devices. Their results show that network devices can be classified according to their degree of specialization, for instance, highly specialized ASICs built for a specific purpose or flexible software-based devices relying on general-purpose CPUs.

Our paper investigates the performance of two distinct representatives of both device groups. As *hardware target* we analyze a purpose-built Intel Tofino switching ASIC. This device stands out performance-wise due to a high degree of parallelism and the absence of caches. On account of its fixed number of programmable pipeline stages, low latency with low jitter, properties typical for hardware targets, is achieved. Resource limiting factors are the complexity of required functionality and fitting the program to the target resources. Our *software target* uses the t4p4s P4 compiler, which produces DPDK-compatible [4] code that runs on commercial off-the-shelf (COTS) CPU-based systems. As is typical for CPU-based systems, t4p4s behavior is influenced by many factors, including interrupts, memory hierarchies, and cache sizes [5]. While achieving lower throughput and higher latency, the advantage is the virtual non-existence of limits regarding the P4 program complexity and custom functionality.

In P4, packet processing tasks are expressed as a series of matches and actions on packet or metadata. Being at the center of every P4 program, the match-action performance is crucial to the understanding of the performance of the entire packet processing pipeline. Therefore, we analyze this component with a special focus on the differences between the different target platforms. Based on our measurements, we derive performance models for the two investigated target platforms reflecting the individual hardware restrictions of each platform.

Contributions of this paper can be summarized as follows: Providing a fine-grained classification of P4 targets with regards to properties of interest for end-users. An in-depth analysis of the match-action tables of P4 programs, and finally the introduction of a novel modeling technique to predict the performance of P4 devices based on its core component—the match-action tables.

The remainder of this paper is structured as follows. Section II discusses related work. In Section III, background regarding P4, key performance indicators (KPIs), and particularities of target devices are discussed. Our methodology and measurement setup are presented in Section IV and Section V, respectively. Subsequently, Section VI elaborates on our CPU performance model, while our ASIC resource model is introduced in Section VII. Section VIII concludes this paper.

II. Related work

Since the concept of benchmarking as a means to understand performance characteristics of a networking device is well explored, a variety of related discussions exist.

Rotsos et al. [3] introduce OFLOPS, a benchmarking suite for OpenFlow switches. Their suite benchmarks the switch behavior with different OpenFlow rules in place. They demonstrate that switch performance differs significantly between different hardware and software implementations of OpenFlow switches. Dang et al. [2] present a similar idea of a benchmark suite, but specialize in performance evaluation for P4 capable devices, such as CPU or FPGA. Comparing a wide range of P4 implementations, they remain abstract in their performance investigations but focus on the components of P4 programs.

Adding to that, we provide insights specific to P4 tables, focusing on the recent version of P4, that is, $P4_{16}$.

An in-depth discussion of the performance of different implementations of an extern P4 function, namely cryptographic hashing, is given by Scholz et al. [6]. They add external hashing capabilities for different P4 targets, namely a CPU, a Network Processing Unit (NPU), and an FPGA. The performance impact of this extension is investigated through thorough benchmarks, revealing large performance differences on a per-platform basis. Further, they show that this functionality only requires up to 2 % of the target's total resources. In their performance investigation of a Netronome SmartNIC's P4 implementation, Harkous et al. [7] explore the impact of header parsing and modification as well as influence of match-action table applications. They present a model describing observable latencies in high throughput scenarios. We improve on the provided insights by increasing granularity of investigated parameters. Also, a comparison between different targets puts individual measurements into perspective. Geyer et al. [8] investigate P4 in the context of avionic applications. They benchmark P4 implementations of avionics full-duplex switched Ethernet (AFDX) on different targets. Their investigation shows that latency differs between target platforms, but the latency is comparable to existing dedicated AFDX hardware.

An attractive property of P4, not only to avionics, is its design that fosters the simplified verification of program behavior, e.g., by the absence of loops in P4. Liu et al. [9] and Neves et al. [10] demonstrate the verification of P4 programs using asserts to identify bugs in applications. A different approach towards code correctness is taken by Nötzli et al. [11]. Based on the P4 program, they automatically create test cases to check correct program compilation. Our model of P4 devices extends the prediction beyond correctness to performance.

The previous examples show that performance of P4 programs is highly target-specific. Thus, our paper measures these properties in a target-specific manner. Our modeling approach also reflects target specificity, presenting models based on the fundamentally different hardware architectures of the investigated platforms, enabling more accurate predictions.

III. Programmable Network Devices

This section provides background information on P4 as a language, KPIs of networking hardware, and a selection of P4 targets of importance for this paper.

A. P4 Programming Language

P4 provides a novel approach to SDN allowing network operators to program custom network device behavior. It is a packet-centric language, focusing on applying match-action tables. The abstract architectural model defines a sequence of control blocks. Framed by a parser—constructing packets from bit sequences—and its deparser counterpart, these control blocks perform header-dependent actions and packet modifications. Consequently, parser, match-action control blocks, and deparser describe basic components of a P4 architecture. Details, e.g., the number of available control blocks, additional meta information, or availability of additional functionality, vary between different networking devices. Hence, P4 introduces abstract representations of hardware, summarizing its design and functionality.

P4's centerpiece are match-action tables. They are designed to allow combining sets of keys, such as, particular header fields, to determine actions. The table entries are provided during runtime by the control plane. A sequence of different match-action tables may be applied to a packet to realize packet processing tasks. [1]

B. Key Performance Indicator

KPIs outline the properties of interest of networking hardware. For this discussion, we differentiate two groups of KPIs:

P4 target selection properties are relevant criteria to determine P4 targets meeting an expected level of service quality. Among these properties are *resource consumption*, *functionality*, and *setup time*. Depending on the program complexity, *resource consumption* may limit the potential devices capable of running the program. In contrast to that, *functionality* refers to the device's support for standardized P4 functionality as well as additional features—provided via so-called P4 *extern*s. The ability to define custom functionality may provide an alternative if required *extern*s are unavailable. Lastly, *setup time* is the time needed to deploy and start up a P4 program and device. Assuming most P4 devices are provisioned seldom, we disregard this non-functional property—including compile time—for this discussion.

Runtime properties are defined mainly through *throughput*, *latency*, and *jitter*. Typically, one aims for the highest available throughput, which still allows the device to operate without impairment by packet loss. In contrast to that, latency should remain as low as possible, while variations of packet latencies, the observed jitter, should be as minimal as possible. We consider the latency between ingress and egress port of a device and jitter as the fluctuations of this latency.

C. Classes of Programmable Devices

P4 is available for a variety of programmable devices. Table I provides an overview using the KPIs defined in Section III-B. Based on a general-purpose COTS **CPU**, these systems use software implementations to provide arbitrary functionality. While this, in general, comes at the cost of performance, it provides flexibility in regards to functionality and complexity. The CPU's processing power limits the throughput on these systems, interrupts and cache effects may impact latency and jitter.

The **Network processing unit (NPU)** is a many-core architecture consisting of cores optimized for packet processing. Programmable NICs can be equipped with such an NPU. Its optimized architecture provides high throughput performance and consistently low latency. However, as NPUs are specialized hardware, available in fewer shapes than fixed-function NICs, these benefits come at the cost of reduced flexibility.

	CPU	NPU	FPGA	ASIC
Throughput	+	++	+++	++++
Latency	$> 10\,\mu s$	$5\,\mu s$ to $10\,\mu s$	$< 2\,\mu s$	$< 2\,\mu s$
Jitter	−−−−	−−−	−−	−
Resources	++++	+++	++	+
Flexibility	++++	+++	++	+
Example	t4p4s DPDK	NFP-4000 SmartNIC	NetFPGA SUME	Barefoot Tofino

Table I: Comparison of different P4 target architectures. Performance categorizations are estimates for available products based on own measurements and related work [2], [6], [7].

Programmable via hardware description languages (HDLs), **FPGAs** are programmable to provide almost arbitrary functionality. Limited only by basic hardware constraints such as memory resources and timing limitations, FPGAs often surpass the previous target architectures in regards to throughput, jitter, and latency. While, in theory, being a highly flexible architecture, programming FPGAs requires hardware-specific knowledge and implementing network algorithms in an HDL becomes a time-consuming task.

In contrast to all previous platforms, **application-specific integrated circuits (ASICs)** have a purpose-built, but limited instruction set. Through optimizations, e.g., a high degree of parallelism, they excel in regards to processing, i.e., throughput, latency, and jitter. However, limited expressiveness of an ASIC's instruction set also imposes boundaries on their flexibility regarding feature implementations.

While other architectures exist, the ones above are the most common targets in literature. In addition, they score in number of commercial products either in development or already available. Among the discussed targets, a trade-off between runtime properties and cost on the one hand, and flexibility and resource restrictions on the other becomes apparent.

IV. Methodology

We transition from a complete P4 program towards independent components in a top-down approach. Our work focuses on the performance of P4's main feature: match-action tables.

A. P4 Program Components

One of the core principles of P4 is segmentation of packet processing into disjunct programmable blocks [2], for instance, parser, match-action pipelines, and deparser. This is reflected in the language, as well as architecture models, defining the stages for a concrete target. Following this strict modularity, we investigate each P4 stage separately. We argue, that models of individual P4 stages can later be combined to represent complete P4 programs. Performance predictions of combined models can be used to validate individual P4 stage models.

Our performance analysis considers multiple P4 programs with increasing complexity. Initially, a baseline P4 program is derived which includes as little functionality as possible while still allowing performance measurements. Subsequently, the baseline program is extended by a match-action table, whereby

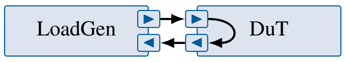


Figure 1: Setup

in successive tests, different properties of the table are scaled and benchmarked. Each of these programs differs from the common baseline program modifying a single property, e.g., the number of match keys or table entries. From this, we derive a performance model for the match-action table component. Ultimately, combining models of individual components facilitates modeling arbitrary programs by describing the program's behavior as a sum of the applied components. A consequence of this modeling approach is that with increasing granularity of the described component properties, precision of the resulting model is expected to increase.

B. P4 Match-Action Table Properties

Applying the presented approach of investigating component properties for performance modeling, the following parameters for match-action tables were identified: *(1)* the table's match type—exact, longest prefix match (lpm), or ternary—which determines the mode of comparison between a packet header or metadata field value and available table entries; *(2)* size of individual table entries, defined by the size and number of keys, number of actions, and the action data; *(3)* number of entries in the match-action table; *(4)* the total number of match-action tables in a P4 program.

An experimental evaluation of the combination for all parameters, e.g., testing different key widths, is not feasible. Therefore, we identify the parameters with the largest impact and focus on them for modeling.

V. Measurement Setup

Measurements were conducted in an automated and reproducible fashion [12]. Figure 1 shows the setup consisting of two nodes, load generator and Device-under-Test (DuT), which are directly connected. We use MoonGen [13] as load generator for throughput and precise latency measurements. Configuration of the DuT is given in the respective sections.

VI. Performance Model

We use the DPDK-based t4p4s P4 switch as basis for our performance model [14]. Running on a COTS CPU system it has highly dynamic throughput and latency characteristics compared to other available P4 target architectures and therefore requires an in-depth analysis of these KPIs. The t4p4s P4 compiler generates target-independent C code, which can then be linked with libraries providing target-specific code for different platforms. One such library utilizes DPDK, which is covered by the following discussion. We use the upstream t4p4s version (commit 919c521 [15]) with small changes due to performance or functionality reasons. The DuT running the switch is equipped with an Intel Xeon CPU E5-2640 v2 clocked at 2.0 GHz and an Intel X540-AT2 NIC. For all measurements turboboost and hyperthreading were disabled to reduce performance jitter. We discuss CPU cyles $\widetilde{C}$, end-to-end latency $\widetilde{L}$ and resources $\widetilde{R}$.

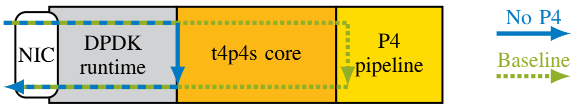


Figure 2: t4p4s traffic flow

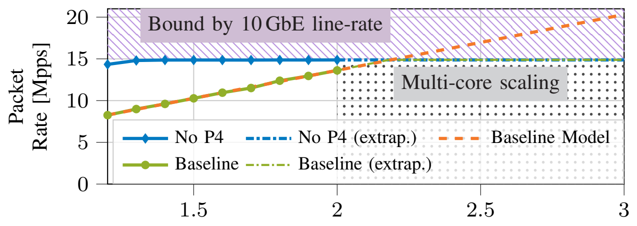


Figure 3: Baseline packet rate

A. *Resource Utilization*

Memory consumption is typically no issue for P4 programs on a CPU-based system, as modern servers can provide RAM up to the TB range. However, the actual usage of memory has an impact on the performance when different levels of caches are involved. This may happen for tables with a large number of entries, i.e., large BGP routing tables. Therefore, we use white-box profiling measurements to analyze the impact of memory consumption and present the results as part of our performance model.

B. *Baseline Model*

We use the two-fold approach shown in Figure 2 to determine the baseline model for the CPU target. The first step (solid blue arrow, referred to as *No P4*) does not include any P4 related code, focusing solely on receiving and sending packets using DPDK. This is to understand the performance overhead of the underlying DPDK runtime. The second step (dotted green arrow, referred to as *Baseline*) adds a minimal P4 pipeline that each packet traverses, consisting only of a minimal parser and deparser, setting the egress port statically without using match-action tables. As before, the goal is to understand the processing overhead generated through P4 boilerplate code even when no actual processing by the P4 program takes place. We use this *Baseline* model for all further measurements of table components.

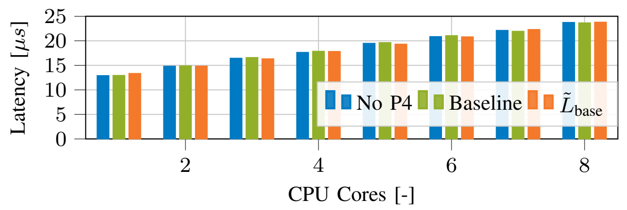
Figure 3 shows the scaling of processed packet rate with CPU core frequency. At 1.3 GHz the *No P4* program is bound by the 10 GbE line-rate. Simply adding the boilerplate code generated by the minimal P4 program reduces the packet rate by approx. 6 Mpps. Increasing the CPU frequency on one core or processing with multiple cores results in linear scaling, bound by line-rate. A higher frequency equals more CPU cycles per second, allowing more packet processing operations in the same interval.

For the remainder of the paper we use the measured maximum packet $\widetilde{P}$ rate to calculate the cycles per packet $\widetilde{C}$ based on the CPU frequency F:

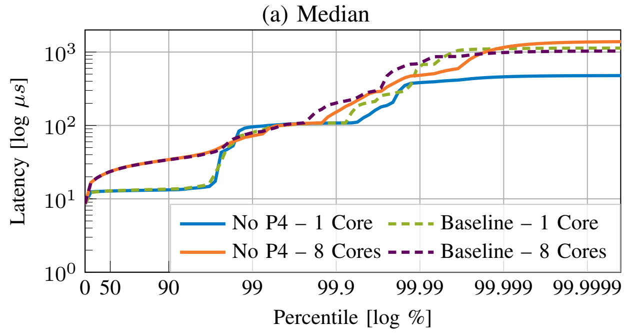
$$\widetilde{C} = \frac{F}{\widetilde{P}} \tag{1}$$

Overall processing of one packet requires approx. 84 and 146 cycles for *No P4* and *Baseline* scenario, respectively, which is in the expected range for DPDK packet processing [5]. For all further experiments the CPU is clocked at $F = 2\,\text{GHz}$ and we assume Equation (2) as baseline CPU cycles consumption:

$$\widetilde{C}_{\text{base}} = 146 \tag{2}$$



(a) Median



(b) Distribution

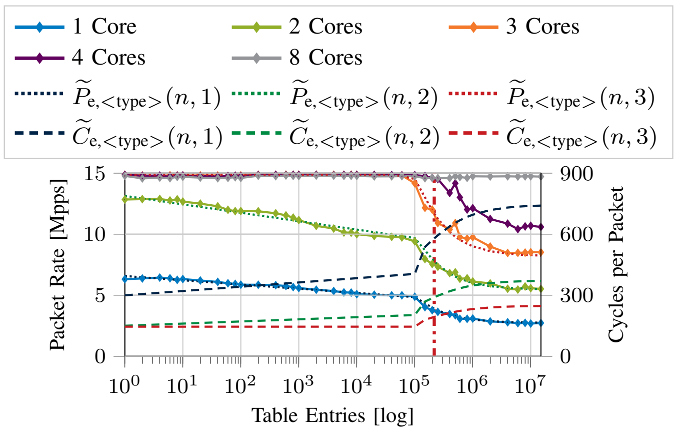
Figure 4: Baseline latency

Median latency, shown in Figure 4, increases linearly with the number of CPU cores. This is due to batch processing of DPDK. While the batch size remains the same, increasing the number of cores also increases the number of batches processed. Maintaining a constant packet rate results in each batch filling slower with an increasing number of cores, resulting in longer delays. The difference in processing for the two scenarios of 62 cycles per packet results in a latency difference of approx. 31 ns at 2 GHz. The High Dynamic Range histogram in Figure 4b shows a typical latency distribution for a CPU-based system. Batches are processed every 100 µs in the worst case [15], resulting in the plateau for the 99th percentile, system interrupts and other side effects cause the long-tail [16], which persists throughout all following measurements.

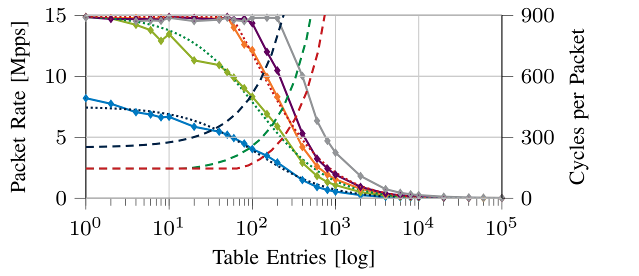
Based on Figure 4a we use linear regression to model the baseline median latency $\widetilde{L}_{\text{base}}$ in relation to the number of CPU cores c clocked at 2 GHz:

$$\widetilde{L}_{\text{base}}(c) = 1.5 \cdot c + 11.73 \tag{3}$$

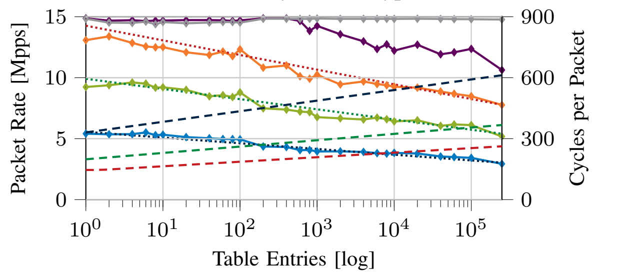
This baseline is used to evaluate and compare the influence of adding P4 instructions utilizing match-action tables in the P4 program.



(a) Exact Match Type



(b) Ternary Match Type



(c) lpm Match Type

Figure 5: Maximum packet rate, $\widetilde{P}$ and $\widetilde{C}$ for increasing number of table entries

C. Number of Table Entries

The number of table entries is a key factor for many P4 applications. As the CPU target has plenty of memory compared to hardware targets, we can push the limits of possible number of table entries. One real-world example are BGP IPv4 routing tables, which have a steadily increasing count of unique routable prefixes, reaching 800 000 IPv4 entries at the beginning of 2020 [17].

P4 supports 3 different match types: exact, ternary, and lpm. In hardware, match types are realized using specialized hardware, e.g., ternary content-addressable memory (TCAM) for ternary and lpm matches. In software, different algorithms with varying properties in regards to limitations and expected performance are used. The second important aspect of software is the memory required to store and access table entries, i.e., the influence of the memory hierarchy on the performance.

Figure 5 depicts the maximum processed packet rate for different match types using up to 8 CPU cores. Like the *Baseline* program, performance scales linearly with the number of used CPU cores. Table entries consist of a $4 \times 4\,\text{B}$ match key with the exception of lpm matches where a $1 \times 4\,\text{B}$ key is used. Caches can accelerate memory access by saving entries that are queried with a high probability. We aim for a challenging scenario, therefore, our traffic is generated such that every packet hits another table entry.

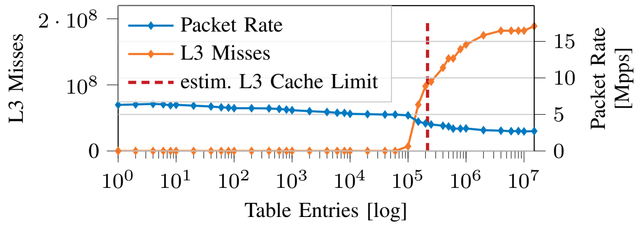


Figure 6: L3 cache misses for exact match

a) Exact Match Type: Figure 5a shows a gradual performance decrease up to 10^5 entries. Adding more entries drastically reduces the performance, resulting in halving the processed packet rate. This is due to the memory hierarchy, in particular, L3 cache misses shown in Figure 6, requiring access to slow main memory. Inspecting the implementation of exact matches results in Equation (4) to model the required resources $\widetilde{R}_{\text{exact}}$ (in B) for an exact match table based on table entries n, key size k (in B) and size of the action a (in B):

$$\begin{aligned}\widetilde{R}_{\text{exact}}(n,k,a) &= 2\cdot 64\,\text{B} + \underbrace{(k\cdot n)}_{\text{Hash table}} + \underbrace{(8\,\text{B}\cdot n)}_{\text{Entries}} + \underbrace{(a\cdot n)}_{\text{Actions}} \\ &= 128\,\text{B} + n\cdot\underbrace{(k+a+8\,\text{B})}_{\text{Table entry size}}\end{aligned} \tag{4}$$

The size of the involved table structs amounts to $2 \times 64\,\text{B}$ due to cache-line alignment. Solving Equation (4) for n using $k = 16\,\text{B}$, $a = 64\,\text{B}$ (cache-line aligned), and $\widetilde{R} = 20\,\text{MB} = \widetilde{R}_{L3}$ (L3 cache size of the used processor) results in 2.2×10^5 entries to fill the L3 cache (see mark in Figure 6). This is an overestimation as the cache is not exclusively used for table entries but also includes packet data. Similarly, the performance drops around 3×10^1 and 5×10^2, a number of entries that roughly correlates with L1 and L2 cache sizes. However, due to access time differences between these two caches, which is below 5 ns, the performance loss is not as noticeable as when exceeding the L3 cache size [18].

While we generate traffic such that every packet hits another entry, it is not the worst case for hash table lookups. Theoretical search complexity in the hash table is constant on average ($\mathcal{O}(1)$), wherefore the main performance influencing factor is memory access, in particular when exceeding the L3 cache size. Using the data of one CPU core we first derive a model for the packet rate (shown in Figure 5 as $\widetilde{P}_{\text{e},<\text{type}>}$) using least squares curve fitting. From this we derive the cycles per packet for exact match entries $\widetilde{C}_{\text{e,exact}}$ (shown in Figure 5 as $\widetilde{C}_{\text{e},<\text{type}>}$), in relation to the number of $4 \times 4\,\text{B}$ table entries

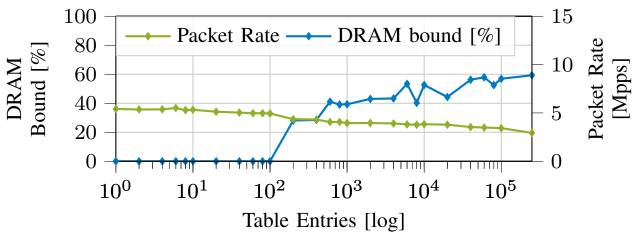


Figure 7: lpm DRAM-bound table accesses

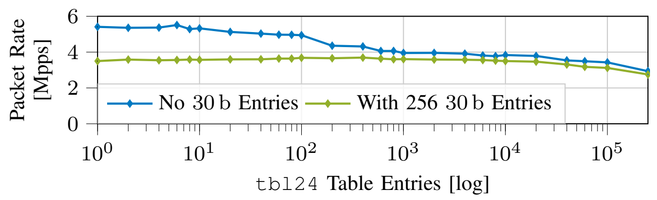


Figure 8: Different prefix lengths for lpm match

n and CPU cores c:

$$\widetilde{C}_{\text{e,exact}}(n,c) = \frac{1}{c} \cdot \begin{cases} p \cdot \ln(q \cdot n) + r, & \widetilde{R}(n) < \widetilde{R}_{L3} \\ \frac{s}{t \cdot n + u} + v, & \text{otherwise} \end{cases} \tag{5}$$

The concrete values for the parameters $p, \ldots, v$ are listed in Table II and are not related for the different match types. Figure 5a shows that the model remains accurate when scaling CPU cores.

<type>	exact	ternary	lpm
p	9.13	4080.98	22.64
q	90.19	-0.00062	201.78
r	258.16	-3831.64	210.57
s	538706.18	-	-
t	-0.01	-	-
u	-1101.14	-	-
v	743.90	-	-

Table II: Derived parameters for an increasing number of table entries using least squares curve fitting. "-" indicates that this parameter is not applicable for the respective model.

b) Ternary Match Type: Due to the lack of specialized hardware like TCAM, implementing ternary matches in software is difficult. The current implementation of t4p4s simply iterates through the list of table entries until a matching entry is found, resulting in exponential search complexity. Due to the lower number of table entries ($< 10^3$) in this scenario, there is no visible impact of the memory accesses on the performance. This can be seen in Figure 5b and our derived model Equation (6).

$$\widetilde{C}_{\text{e,ternary}}(n,c) = \frac{1}{c} \cdot (p \cdot e^{q \cdot n} + r) \tag{6}$$

c) lpm Match Type: t4p4s uses a DIR-24-8 datastructure [19] for 32 bit key sizes (IPv4 lpm). While a different data structure is used to allow 128 bit keys (IPv6 lpm), we focus on the former. DIR-24-8 uses two different sorts of tables, `tbl24` is a single table to store the most significant 24 bit of the prefix in up to 2^{24} entries. The second sort of tables (`tbl8`) are, by default, 2^8 tables to store the remaining 8 bit. Prefix lengths of $\leq$ 24 bit can be resolved with one lookup in the first table, longer prefixes require a second lookup in the respective `tbl8`. DIR-24-8 assumes that routes with a prefix length of greater than 24 bit are rare, optimizing lookups for smaller prefix lengths, while limiting the number of greater than 24 bit prefixes that can be stored [19].

Figure 5c shows the results when using only 24 bit prefixes. Inserting more than 250 000 entries takes considerable amount of time wherefore those measurements are excluded. The measurements show logarithmic instead of the theoretic constant scaling. As with the exact match, cache sizes, in particular L3 cache, are the limiting factor. Already with 200 table entries 30% of lookups are DRAM-bound, requiring data fetched from main memory (see Figure 7). This is due to the increased size of the DIR-24-8 structure, as already the `tbl24` requires 64 MB [19], [20]. As a consequence, the shared L3 cache is filled with per-core DIR-24-8 structures. These restrictions imposed by the hardware architecture result in sub-linear scaling with the number of used CPU cores, resulting in Equation (7) for the basic case.

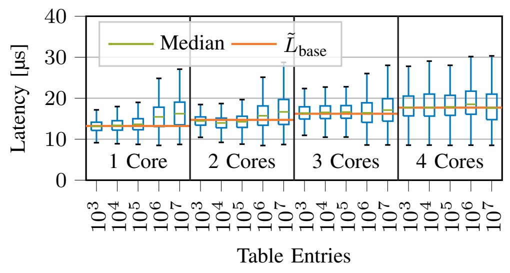
$$\widetilde{C}_{\text{e,lpm}}(n,c) = \frac{1}{c+0.5} \cdot (p \cdot \ln(q \cdot n) + r) \tag{7}$$

The influence of prefix length is shown in Figure 8. We added a constant 256 30 bit prefixes to `tbl8` such that every table match now consists of two lookups. For less than 200 entries a performance increase is noticeable. However, this increase is due to the static increase of 256 table entries. When increasing the number of table entries, the cost for the additional 256 entries amortizes, leaving the cost for the additional lookup in `tbl8`. For simplicity, we do not include the additional cost in our model (approx. 6 % loss in accuracy for $>$ 1000 entries).

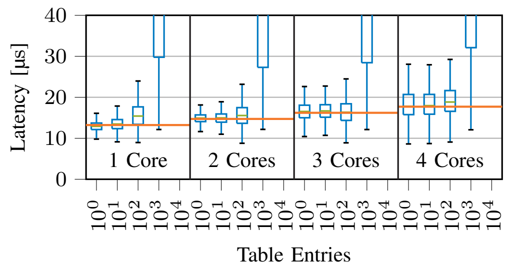
d) Latency: Latency measurements for an increasing number of table entries are shown in Figure 9. The behavior is comparable to the performance results of the previous section. Deviations from the baseline model $\widetilde{L}_{\text{base}}$ are only noticeable when the maximum processed packet rate has large drops due to memory or algorithm restrictions. This is the case for exact and lpm matches with 3 or fewer cores. Furthermore, the median latency for ternary matches with 10^3 and 10^4 entries raises to 50 µs and above 80 µs, respectively, for a single core. While the lower whiskers (1.5 interquartile range) remain constant throughout all measurements with the exception of ternary matches, upper whiskers increase with the number of table entries. This can be explained with the increase in L3 cache misses, resulting in a higher chance that a packet is stalled before the correct memory is accessed.

D. Increasing Table Entry Size

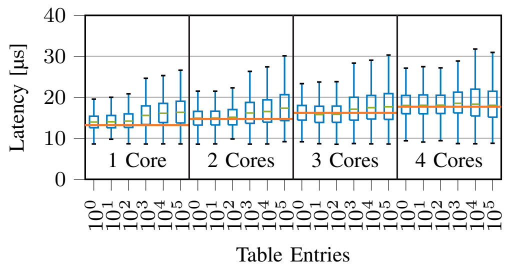
The size of a table entry can be increased either by increasing the size or number of match keys or by increasing the



(a) Exact Match



(b) Ternary Match



(c) lpm Match

Figure 9: Latency for selected number of table entries and increasing amount of CPU cores. Boxplot shows median latency, whiskers display 1.5 interquartile range.

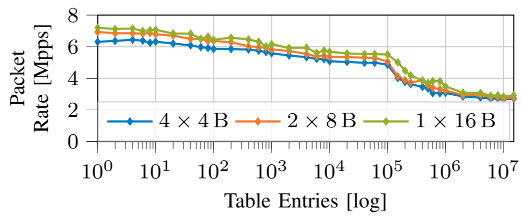


Figure 10: Data type influence (exact match)

stored action data. In both cases, this changes the parameters k and a for $\widetilde{R}$ and, therefore, when the L3 cache limit is reached. Our experiments have shown that this is indeed the case. However, for ternary and lpm tables the already discussed limitations of the respective table implementations outweigh this effect.

E. *Table Key Segmentation*

On CPU-based systems, data used as match key can be represented using different data types. Figure 10 shows the packet rate when using a $2 \times 8\,\text{B}$ or $1 \times 16\,\text{B}$ data structure instead of the $4 \times 4\,\text{B}$ data structure used to represent the

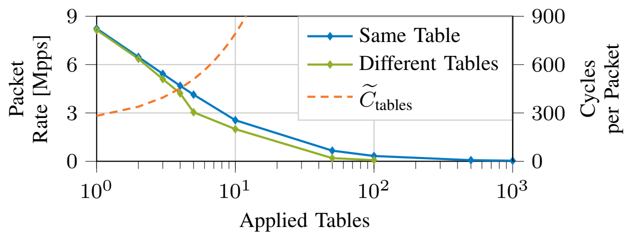


Figure 11: Packet rate for multiple applications of exact match table

match key. As the total key size is the same we can rule out memory accesses as cause for the performance differences. In fact, profiling reveals that the reduced performance for $4 \times 4\,\text{B}$ key segmentation is architecture specific. For this case, *Store Forwarding*, a performance enhancing feature that allows previous memory writes to be forwarded to a subsequent memory read without having to write the value to main memory [21], failed in 100 % of the cases due to incorrectly passing the data to the hash function. For our CPU architecture, this results in a 12 clock cycle performance penalty [21]. Optimizing the code generated by t4p4s would allow Store Forwarding to succeed, yielding better performance. While this effect has a performance impact, we do not include it in our model for the sake of simplicity. Instead, we use the worst-case (default) key segmentation for all our measurements.

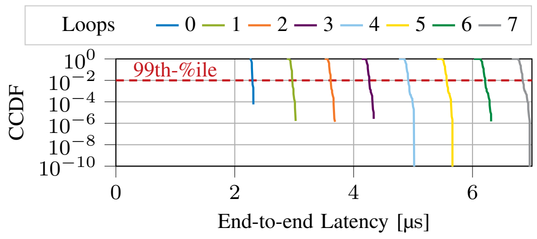
F. *Number of Table Applications*

The last property we analyze is the number of applied tables in the P4 program. While hardware P4 targets typically do not allow the application of the same table multiple times per packet, this restriction does not exist in t4p4s. This flexibility allows a closer estimation of the cost of the actual table application (i.e., hash calculation), without fetching data for keys and entries. The data fetched is the same for successive table applications, getting cached and amortizing the cycles required. We, therefore, analyze both scenarios, applying the same table t times and applying t different tables. Each table is filled with one entry using a $4 \times 4\,\text{B}$ match key.

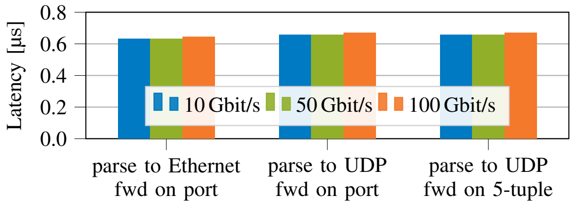
As shown in Figure 11, the packet rate decreases linearly with the number of tables applied. Increasing the number of entries per applied table (not shown) causes the effect discussed in Figure 5a, where memory accesses increase the cycles per packet. The difference between applying the same or different tables is small, wherefore we focus on applying the same table. The cycle model for table applications $\widetilde{C}_{\text{tables}}$ reveals that every table application adds 57 cycles per packet:

$$\widetilde{C}_{\text{tables}}(t) = 56.67 \cdot t + 225.35 \tag{8}$$

The primary factor is the cost of calculating the hash used to access the table as the cost for loading the input data and result are amortized through caching.

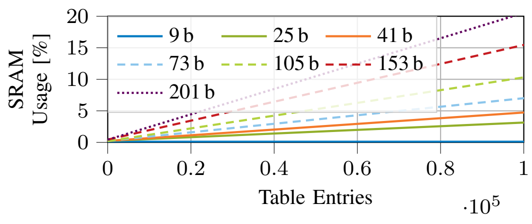


(a) CCDF when looping through additional ports

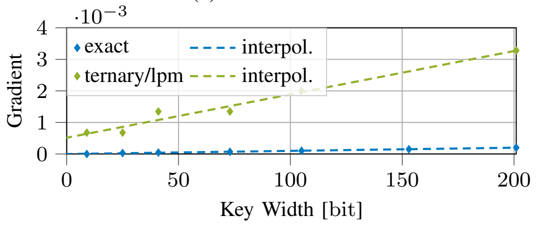


(b) Average latency when increasing complexity of parser and forwarding decision

Figure 12: Latency of ASIC



(a) Exact match



(b) Interpolation of gradient of increasing key width

Figure 13: Key width influence

VII. ASIC Resource Model

For the resource model we used a commercial P4-programmable Intel Tofino 1 ASIC Delta ET-X064FFRB. It is equipped with and capable of switching 65 100 GbE ports.

A. Performance

We use a 10 GbE NIC and the packet duplication feature of the Intel Tofino to generate 100 Gbit/s of traffic. Baseline end-to-end latency in this setup is approx. 2 µs. Latency increases linearly when looping 100 Gbit/s of traffic through increasing amounts of 100 GbE ports via loopback cables (see Figure 12a). Effectively, this increases the load on the ASIC with every additional loop, however, latency remains stable, showing no long-tail behavior under higher load.

As shown in Figure 12b, increasing the complexity of the P4 program yields latency changes in the nanosecond range. This behavior persists even in more complex scenarios, wherefore we focus on the resource consumption for the ASIC target.

B. Available Resources

ASICs typically provide a large amount of affordable static RAM (SRAM). For specialized purposes, a smaller amount of more expensive TCAM is used. These resources are used to create P4 tables, depending on the match type. Exact matches are realized on SRAM, while ternary or lpm matches work best in TCAM. The amount of SRAM and TCAM is fixed, limiting not only the P4 program complexity, but also the size of tables, in particular, such that use ternary or lpm matches.

C. Table Resource Model

As in Section VI we have identified three primary influencing factors regarding the resource consumption $\overline{R}$ of P4 tables on an ASIC. A single entry is defined by the cost $\overline{R}_{\text{width}}$ caused by the key width k (for a certain match type) and total action data a (the result of the lookup). This is then scaled with the number of total table entries n depending on the match type:

$$\overline{R}_{\text{<type>}}(n,k,a) = n \cdot (\overline{R}_{\text{width, <type>}}(k) + a) \tag{9}$$

The action data a can be any data structure defined in the P4 program and is therefore highly variable and difficult to measure the resource cost of. As such, we leave it as unknown variable with the assumption of linear scaling with action data size. However, we have a detailed look at the impact of different key widths to verify our assumption of a linear dependency. As tables can be applied only once per packet due to hardware restrictions, the required resources for all tables of the P4 pipeline are a sum of individual table cost.

D. Key Width Resource Cost

We have tested P4 programs containing a single table with varying key widths for all match types on our DuT. Resource consumption is a discrete staircase function when increasing the number of table entries, which we approximate with a linear function (see Figure 13a for exact match). Key width increases the incline of this approximated function for resource usage. From this, we interpolate the gradient of increasing key width for every match type, shown in Figure 13b.

For simplicity, we assume that a table only uses one match key. However, the model can be extended by summing the cost of every match key before multiplying it with the number of table entries.

Instead of key segmentation and number of table keys, we only depend on the total key width being used. As a result, we can represent resource usage of different key widths using Equation (10):

$$\overline{R}_{\text{width, <type>}}(k) = p_{\text{<type>}} \cdot k + q_{\text{<type>}} \tag{10}$$

where p is the gradient and q the offset of the interpolation function depending on the match type. The concrete values for our tested device are listed in Table III. Note that the usage for ternary and lpm match were identical in our test. The explanation is that lpm is a special case of ternary match, wherefore the hardware implementation is likely identical.

<type>	p	q	Resource
exact	$1.00 \cdot 10^{-6}$	$-2.09 \cdot 10^{-6}$	SRAM
ternary/lpm	$1.44 \cdot 10^{-5}$	$4.81 \cdot 10^{-4}$	TCAM

Table III: Interpolated parameters of resource model

VIII. Conclusion

Network performance is the main concern of network infrastructure providers. While changes in network infrastructure management, such as SDN or P4, open up new possibilities, the performance remains relevant. Our contribution is a set of performance models for hard- and software-based P4 targets. Depending on the type of target, different aspects of the target architecture gain importance. We investigated software-based systems, where the underlying hardware resources like cache sizes impact the performance, whereas memory resources are plenty and hardly impact performance. The analyzed ASIC-based device provides high throughput with low delay and jitter, even when increasing program complexity. However, available memory resources are a limiting factor.

Both, performance and resource model suggest, that each P4 target provides unique properties making it preferable depending on the task in question. The general guideline is that program complexity through available resource benefits appertain to software targets. Throughput and latency, on the other hand, is generally a benefit particular to hardware targets. Furthermore, the presented models enable determining feasibility of a given P4 implementation. While presented models remain target-dependent, that is, parameterization of other P4 targets may differ, model outputs provide hints regarding possibilities and limits.

It is generally known and accepted that ASIC targets excel in providing stable, scaleable performance, while software targets provide a platform with virtually unlimited resources. We argue that modeling of device characteristics should be focused on their respective weaknesses instead. This leads to a focus on resource usage modeling as sensible metric on ASIC targets. The Achilles' heel of software targets, in contrast to that, is performance behavior.

For future work on this matter, we plan to investigate model parameters for other available software targets, for example, P4 transpilers for eBPF or XDP [22].

Acknowledgment

This work was supported by the German Research Foundation (project ModANet under grant no. CA595/11-1) and the German-French Academy for the Industry of the Future. The authors thank Maximilian Endrass and Alexander Frank for their contributions to the evaluation, and the anonymous reviewers for their valuable feedback.

References

[1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[2] H. T. Dang, H. Wang, T. Jepsen, G. Brebner, C. Kim, J. Rexford, R. Soulé, and H. Weatherspoon, "Whippersnapper: A P4 Language Benchmark Suite," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: ACM, 2017, pp. 95–101.

[3] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation," in *Passive and Active Measurement*, N. Taft and F. Ricciato, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 85–95.

[4] DPDK Project, "DPDK Homepage," 2019, https://www.dpdk.org/.

[5] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of Frameworks for High-Performance Packet IO," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2015)*, Oakland, CA, USA, May 2015.

[6] D. Scholz, A. Oeldemann, F. Geyer, S. Gallenmüller, H. Stubbe, T. Wild, A. Herkersdorf, and G. Carle, "Cryptographic Hashing in P4 Data Planes," in *2nd P4 Workshop in Europe*, Cambridge, UK, Sep. 2019.

[7] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "Towards Understanding the Performance of P4 Programmable Hardware," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems - 2nd EuroP4 Workshop*, Cambridge, UK, Sep 2019.

[8] F. Geyer and M. Winkel, "Towards Embedded Packet Processing Devices for Rapid Prototyping of Avionic Applications," in *9th European Congress on Embedded Real Time Software and Systems*, Jan. 2018.

[9] J. Liu, W. T. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Cascaval, N. McKeown, and N. Foster, "p4v: practical verification for programmable data planes," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2018, Budapest, Hungary, August 20-25, 2018*, S. Gorinsky and J. Tapolcai, Eds. ACM, 2018, pp. 490–503.

[10] M. C. Neves, L. Freire, A. E. S. Filho, and M. P. Barcellos, "Verification of P4 programs in feasible time using assertions," in *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, CoNEXT 2018, Heraklion, Greece, December 04-07, 2018*, X. A. Dimitropoulos, A. Dainotti, L. Vanbever, and T. Benson, Eds. ACM, 2018, pp. 73–85.

[11] A. Nötzli, J. Khan, A. Fingerhut, C. W. Barrett, and P. Athanas, "p4pktgen: Automated test case generation for P4 programs," in *Proceedings of the Symposium on SDN Research, SOSR 2018, Los Angeles, CA, USA, March 28-29, 2018*. ACM, 2018, pp. 5:1–5:7.

[12] S. Gallenmüller, D. Scholz, F. Wohlfart, Q. Scheitle, P. Emmerich, and G. Carle, "High-Performance Packet Processing and Measurements (Invited Paper)," in *10th International Conference on Communication Systems & Networks (COMSNETS 2018)*, Bangalore, India, Jan. 2018.

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

[14] P. Vörös, D. Horpácsi, R. Kitlei, D. Leskó, M. Tejfel, and S. Laki, "„T4P4S: A Target-independent Compiler for Protocolindependent Packet Processors"," in *IEEE HPSR*, 2018, pp. 17–20.

[15] P4ELTE, "T4P4S, a multitarget P4$_{16}$ compiler," 2020. [Online]. Available: https://github.com/P4ELTE/t4p4s

[16] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5G QoS: Impact of Security Functions on Latency," in *2020 IEEE/IFIP Network Operations and Management Symposium (NOMS 2020)*, Budapest, Hungary.

[17] Huston, Geoff, "BGP in 2019 – The BGP Table," 2020. [Online]. Available: https://blog.apnic.net/2020/01/14/bgp-in-2019-the-bgp-table/

[18] *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Intel.

[19] P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds," in *Proceedings. IEEE INFOCOM '98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No.98)*, vol. 3, pp. 1240–1247 vol.3.

[20] DPDK Project, "`rte_lpm.h` File Reference," 2019, https://doc.dpdk.org/api-19.02/rte__lpm_8h_source.html; last accessed on 2020-05-29.

[21] A. Fog, "The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers," *Copenhagen University College of Engineering*, p. 134, 2012.

[22] W. Tu, F. Ruffy, and M. Budiu, "P4C-XDP: Programming the linux kernel forwarding plane using P4," in *Linux Plumber's Conference*, Vancouver, Canada, November 13-15 2018. [Online]. Available: http://budiu.info/work/p4c-xdp-lpc18-paper.pdf