A Framework for Reproducible Data Plane Performance Modeling

Dominik Scholz[1], Hasanin Harkous[2], Sebastian Gallenmüller[1], Henning Stubbe[1], Max Helm[1], Benedikt Jaeger[1], Nemanja Deric[2], Endri Goshi[2], Zikai Zhou[2], Wolfgang Kellerer[2], Georg Carle[1]
[1]Chair of Network Architectures and Services, [2]Chair of Communication Networks
Technical University of Munich, Munich, Germany
firstname.lastname@tum.de

ABSTRACT

Languages for programming data planes like P4 sparked a plethora of new applications in the data plane. The dynamic, evolving environment makes it challenging to understand what performance can be expected when running a program in a specific data plane target. However, knowing this is crucial for network operators when upgrading their networks.

We present a framework for the reproducible analysis and modeling of P4 program components. By defining and generating precise specifications of the experiments, we separate fully auto-generated components from testbed- or target-specific parts. Measurement results are used to derive performance models automatically. These can then be used to compare the measured with the theoretical performance, or to model the cost of entire paths through the data plane. In two case studies, we use our framework to discover and model selected behavior for a DPDK-based software target and for the NFP-4000 SmartNIC platform.

CCS CONCEPTS

• **Networks → Network performance modeling**.

KEYWORDS

P4, Data Plane, Performance Modeling Framework

ACM Reference Format:
Dominik Scholz, Hasanin Harkous, Sebastian Gallenmüller, Henning Stubbe, Max Helm, Benedikt Jaeger, Nemanja Deric, Endri Goshi, Zikai Zhou, Wolfgang Kellerer, Georg Carle. 2021. A Framework for Reproducible Data Plane Performance Modeling. In *Symposium on Architectures for Networking and Communications Systems (ANCS '21), December 13–16, 2021, Layfette, IN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3493425.3502756

1 INTRODUCTION

High-level domain-specific languages (DSLs) like P4 [3] for data plane programming are the next step towards fully programmable networks. The ability to customize details of switch and router internals without the need for long development cycles attracted researchers and industry alike. Consequently, the P4 landscape has enjoyed steady growth in recent years, resulting in not only a shift of entirely new applications to the data plane, but also new software and hardware targets. Keeping an overview of all applications and targets has become challenging, in particular, regarding performance metrics, i.e., how certain applications behave for specific target platforms. However, answering this question can be crucial, as network operators typically have QoS requirements to fulfill.

Analyzing or modeling the performance of each application for each potential target platform is infeasible due to the sheer complexity of the landscape. However, using an abstract DSL like P4 allows splitting the processing pipeline of a program into individual components, including the parser stage and match-action tables. The complexity of each component can be described using a set of parameters, e.g., the number of parser states or match-action table entries. Each component or feature can then be evaluated and modeled individually by only varying the respective parameter.

We present a framework that analyzes and automatically models the behavior of individual P4 language components. The derived models can then be used to compare their behavior with the expected theoretical behavior. Deviations may point to limitations of the device or not ideally implemented data structures. We identified multiple challenges when implementing the framework: First, like the P4 language itself, the framework needs to be portable. The framework has to support a wide range of software and hardware P4 targets, and vastly different testbed environments. Furthermore, results must be reproducible, so that the resulting models can be extended or verified by others. To achieve this, our framework employs a high degree of automation. Further, we carefully separate our framework into three parts: auto-generated, testbed-, and target-specific. We show the value of our framework by highlighting findings for two different P4 targets: the DPDK-based t4p4s and the NFP-4000 SmartNIC.

The paper is structured as follows: The framework for automated analysis and modeling of P4 components is introduced in Section 2. We present our findings in two case studies that apply our modeling framework to a software-based and a SmartNIC-based P4 target in Section 3. Section 4 discusses related work, before Section 5 concludes our paper.

2 MODELING FRAMEWORK

Just like the language itself, performance evaluation of P4 targets should be protocol-independent, i.e., should not exclusively work for existing protocol headers like IP or UDP. Similarly, applications are manifold, wherefore the evaluation should be independent of existing or future applications.

2.1 Concept

We use short synthetic experiments targeting P4 language constructs, similar to the approach introduced by Dang et al. [5]. We evaluate the basic components of P4 programs, i.e., any feature of the P4 language [24] that can be scaled like the number of parsed headers or table entries. Analyzing small building blocks individually instead of full programs serves multiple purposes. First, we reduce side effects caused by components interacting with each other. Second, these experiments can be used as regression tests by developers of the compilers. Lastly, the measurements allow application developers to gain a fundamental understanding of the cost of P4 language constructs. This is required as, based on the concrete target, the impact of one component on performance and resource metrics can change drastically. A model for the impact of components on a concrete target can be used to estimate the cost of the complete application by extrapolating and adding up the costs of individual components.

Each measurement series only contains a single language component in addition to our baseline program. The goal is to understand and model the impact introduced by this component, i.e., how does the latency change when including this construct n times in a program. As these simple P4 programs are composed of the language primitives investigated in the component measurements, a theoretical performance can be calculated. The comparison between theoretical performance based on the model and measured performance of the composed measurement results in a relative error, which can be used to describe the quality of the model.

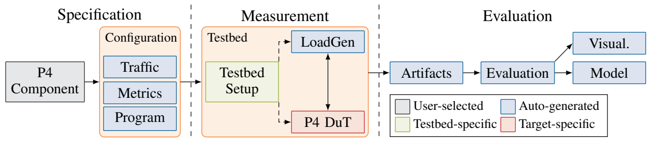

Figure 1: Measurement Framework

Our open-source framework [20] conducts the automated evaluation of a single P4 program component in three phases as shown in Fig. 1: experiment specification, measurement execution, and evaluation of generated artifacts.

Component Experiment Specification. For the sake of reproducibility, each experiment analyzing a single P4 component is defined by a threefold specification. First, it specifies the performance metrics of interest. Throughput, packet rate, and latency metrics are collected on the load generator (LoadGen) per default. Furthermore, internal target-specific parameters can be measured on the device under test (DuT), e.g., CPU cycle usage, cache misses, or the resource consumption of the P4 program complexity. The specification of performance metrics is vital for all other components, including the load generator, P4 program, and the evaluation.

Second, the traffic generated by the load generator and sent to the DuT is specified. This includes targeted throughput, packet size, traffic pattern, and the headers and payload of each packet. In particular, it is defined whether specific bytes of the generated packets have to be changed throughout a measurement series. This is required to generate traffic, e.g., matching different entries of match-action tables.

Lastly, the specification defines the parameters of the P4 program. Based on a baseline program, i.e., the minimal program required to forward packets, program complexity is increased only for the respective component that should be analyzed. The specification contains the number of occurrences for every component and further details, e.g., how many bits each parser state parses. As a result, the components of the whole P4 program are specified such that the program for a specific target can be generated.

Measurement Execution. Running a measurement poses two challenges: First, independent of performing P4 data plane evaluations, each testbed environment is different, be it the hardware of the testbed's management node, the testbed orchestration software used to perform measurements, or the actual testbed nodes. Therefore, the testbed setup requires a testbed-specific implementation. This includes starting and synchronizing individual measurement runs between different nodes, in our case, the load generator and the P4 DuT, as well as gathering all artifacts. We have implemented the setup component for testbeds using the plain orchestrating service (pos) [8, 9] and using a purpose-built approach.

Although P4 programs are intended to be portable, i.e., target-agnostic, in theory, target-specific knowledge is still required. This includes, e.g., the P4 architecture model that the P4 program needs to adhere to, supported extern interfaces, and the control plane interface required for loading the program and inserting match-action table entries. Therefore, the automatically generated P4 program specification needs to be translated to an actual P4 program fitting the P4 target. For this work, we have implemented this target-specific component for the DPDK-based t4p4s target. Further, the testbed-specific component needs to implement means to manage the concrete P4 device.

The MoonGen-based [7] load generator is automatically generated based on the specification. By default, two sets of measurements are performed with constant bitrate (CBR) traffic. First, the maximum throughput and packet rate that the DuT can process without packet loss are determined by subjecting the DuT to traffic at line-rate. We then measure the device's latency at 10 %, 50 %, and 70 % of the maximum packet rate, respectively. Other experiments, e.g., using poisson or bursty traffic, are supported by MoonGen [7].

Artifact Evaluation. As the configuration and execution of the load generator are fully automated, all metrics obtained from this source are evaluated and visualized automatically. Only data obtained from the DuT requires a target-specific processing implementation.

2.2 Automated Model Derivation

We describe our DuT as packet processing system s that, for input values I like the program, packet rate, packet size, or traffic pattern, produces output values O, e.g., the maximum packet rate, latency, CPU cycles per packet, etc. As it is infeasible to determine, measure, and model all inputs and outputs of s, we limit both inputs I^* and outputs O^* of our model m that we want to derive for individual components of s: To derive the model for an individual component, we use the fully automated experiments of the framework to obtain measurement data g that defines the behavior of this component expressed as $g(x) = y$ with $x \in G$ defining the measurement domain. Based on g, we want to derive a model $m(I^*) \mapsto O^*$ represented by a modeling function M with an error metric H that quantizes the quality of M. For each component that we analyze individually we also derive a separate model, e.g., denoted as m_{parser} or m_{MAT}.

Curve Fitting. To derive a model for the measurement data g for the whole or a part of the measurement domain $X \subseteq G$, we use curve fitting applying the non-linear least squares Levenberg-Marquardt algorithm [17]. We use the *curve_fit* algorithm of the python scipy module [25], which works as follows: for a given function prototype τ with free parameters $\vec{p}^*$ the algorithm tries to determine $\vec{p}$ to fit the given measurement data using τ as close as possible. The result is the parameterized free parameter vector $\vec{p}$. An example for such a function prototype with three free parameters is the polynomial of degree two $\tau(x) = p_1^* x^2 + p_2^* x + p_3^*$.

To improve the quality of generated models, we use a set Λ^* of different function templates λ. Each template $\lambda = (\tau, \psi)$ is defined by the function prototype τ for curve fitting and an associated rank ψ that will be explained later. As prototypes we use polynomials of degrees zero to five, exponential and logarithmic functions, and the inverse of all mentioned functions. For every template λ from the set of function templates Λ^*, the curve fitting algorithm is applied to solve for the free parameters $\vec{p}$. The result is the set of possible solutions Λ for the given measurement data $\Lambda := \{\text{curve\_fit}(\lambda) = (\tau, \psi, \vec{p}, \eta) \mid \forall \lambda \in \Lambda^*\}$.

Model Quality. We use an error metric η defined by the experiment specification to quantify the quality of each calculated fitting in Λ. Currently, the framework supports the mean absolute percentage error (MAPE) [4] and the symmetric MAPE (sMAPE) [4] metrics for regular measurement data. However, other metrics can be added as plugins.

MAPE has drawbacks, e.g., it is sensitive to outliers or artifacts in the measurement data [4]. sMAPE improves on the issues of MAPE, wherefore, we use the following variation as default error metric for the remainder of this work:

$$\eta^{\text{sMAPE}} = \sum_{x \in X} \frac{|\tau(x) - g(x)|}{|g(x)| + |\tau(x)|} \tag{1}$$

MAPE, sMAPE, and other metrics are vulnerable to overfitting, resulting in complex functions being preferred. E.g., we assume measurement data with linear dependency. Due to measurement inaccuracies, a high degree polynomial will likely have a lower error and would be preferred according to the sMAPE-based error metric η. Thereby, the polynomial's high degree factors are close to zero, i.e., they are of low relevance for the overall model. While it is mathematically correct to choose the higher degree polynomial, semantically a polynomial of degree one is desired to fit the linear dependency. We counteract this behavior using two independent strategies to improve the calculated error metric η.

First, we forbid small parameters by defining an absolute minimum value γ. All free parameters p returned by the curve fitting algorithm are processed accordingly:

$$p' = \begin{cases} \text{sgn}(p) \cdot \gamma, & \text{if } |p| < \gamma \\ p, & \text{otherwise} \end{cases} \tag{2}$$

Eq. 1 then uses the capped parameters p'. We argue, that limiting the granularity of the free parameters is justified as it reflects the limited measurement accuracy.

The second strategy is based on the Akaike information criterion (AIC) [4]: we assign a rank ψ to every function that is equal to the number of free parameters $\psi = |\vec{p}^*|$. If the difference in error metric for two fittings is between a certain margin ω, we choose the simpler function. ω is based on the minimum of both fitting errors multiplied by a margin factor κ_{rel}. As this would have close to no effect for already small errors, we also define an absolute minimum margin κ. For two fittings for the same domain F_1 and F_2, if $|\eta_1 - \eta_2| \leq \omega$, we choose the fitting with lower rank ψ:

$$\tau_{\text{chosen}} = \begin{cases} \tau_1, & |\eta_1 - \eta_2| \leq \omega, \psi_1 < \psi_2 \\ \tau_2, & |\eta_1 - \eta_2| \leq \omega, \psi_1 \geq \psi_2 \\ \tau_1, & \eta_1 < \eta_2 \\ \tau_2, & \text{otherwise} \end{cases} \quad (3)$$

with $\omega = max(min(\eta_1, \eta_2) \cdot \kappa_{\text{rel}}, \kappa)$. We consider an error metric increased by up to $\kappa = 10\,\%$ acceptable to prioritize simpler model functions. We do not directly use the AIC formula since experiments have shown that this metric is too aggressive in preferring simpler functions in some scenarios, a common point of critique for AIC [4]. The framework is designed to support other metrics like MAPE that can be used as plugins instead. For measurement data that should be modeled using probability distributions, e.g., gaussian or trapezoid, other metrics, like the earth mover's distance [19], can easily be integrated.

Resulting Model. The result of this process is a function prototype τ and the calculated free parameters $\vec{p}$ being the best solution according to the error metric η and function rank ψ out of all calculated fittings Λ. The function models a part of or the full measurement domain, i.e., $X := \{x \in G \mid \alpha \leq x < \beta\}$. We denote this chosen fitting as $F = (\lambda, \vec{p}, \alpha, \beta, \eta)$. For simple systems, F and the adjustable parameters κ, κ_{rel}, and γ represent the complete model $m : (F, \eta; \kappa, \kappa_{\text{rel}}, \gamma)$.

Multiple Partial Fittings. The behavior of complex systems cannot be modeled using only a single function. Events like overloading the system or exceeding the capacity of CPU caches can drastically worsen the performance behavior. Therefore, behavior before and after an event should be modeled independently.

For this, we split the measurement domain G into n separate domains and use the above outlined approach for every individual segment. The combined fitting $\mathcal{F}$ is denoted as:

$$\mathcal{F}(x) = \begin{cases} F_1^n(x), & s_0 \leq x < s_1 \\ F_2^n(x), & s_1 \leq x < s_2 \\ \quad\vdots & \\ F_n^n(x), & s_{n-1} \leq x \leq s_n \end{cases} \quad (4)$$

The n individual fittings $F_i^n = (\lambda_i, \vec{p}_i, s_{i-1}, s_i, \eta_i)$ are delimited by $n+1$ splitting points $\vec{s}$ from the set of possible splitting points S_n. The points s_0 and s_n denote the lower and upper bound of the measurement domain G, respectively. The error θ and rank Φ for the combined fitting is a weighted sum of the individual fitting errors η_i and function ranks.

To determine the best combined fitting, we apply the AIC-based metric of Eq. 3 to always compare two combined fittings $\mathcal{F}_1$ and $\mathcal{F}_2$ using θ and Φ. Eventually, this selects the best modeling function $\mathcal{M}$ consisting of multiple partial fittings for this number of splitting points denoted as $\mathcal{M} = (\mathcal{F}, \theta, \Phi)$. The final model is then defined as $m : (\mathcal{M}, \theta; \kappa, \kappa_{\text{rel}}, \gamma)$.

Determining Splitting Points. For up to three fittings, we use brute force, i.e., we calculate fittings for all possible combinations of splitting points. This results in a total of up to $O(|G|^n)$ fittings that need to be calculated. Calculating these can be parallelized, while a single curve fitting requires less than 0.5 s, depending on the number of data points.

For a higher degree of splitting points, brute force is not feasible due to increased computational time. Instead, we use a heuristic to determine the set S_n. We assume that a performance-altering event is indicated by a drastic change in inclination of the measurement data, i.e., a point after which the slope of the curve alters its direction. Therefore, we calculate the second derivative of the measurement data, approximated by local piecewise derivates using finite differences, and select the x-axis index of local maxima and minima. From these indices we use the l highest absolute extrema as splitting points.

To reduce the impact of measurement artifacts on derivative calculation, we repeat this process while including one or more intermediate steps. E.g., we smooth the measurement data using local linear or polynomial regression. We intentionally perform different rounds of manipulating the measurement data and determining splitting points as each operation, e.g., smoothing measurement data, may improve or worsen the quality of the determined splitting points for the current scenario. We further extend this set S_n by adding the j surrounding data points for each determined splitting point. Extending the set of possible splitting points improves the overall accuracy at the cost of computation time. However, compared to brute force, the number of splitting points can be limited through l and j.

Similar to our error metric, the method to calculate derivatives can be replaced with other approaches [2] as plugins. E.g., different filters to reduce the impact of noise, like the Savitzky-Golay filter [18], can be applied to the measurement data before calculating the derivative function. However, these only provide scenario-specific optimized solutions, wherefore, the outlined approach is used by default.

3 USE CASE EVALUATIONS

Testing every language component for all target platforms might be infeasible. Therefore, we apply the proposed modeling framework to two different P4 platforms in a case study: a software switch and a SmartNIC hardware platform.

Setup. The load generator produces CBR traffic and is directly connected to the DuT. The DuT running the DPDK-based t4p4s [26] P4 switch is equipped with an Intel Xeon CPU E5-2640 v2 clocked at 2.0 GHz and an Intel X540-AT2

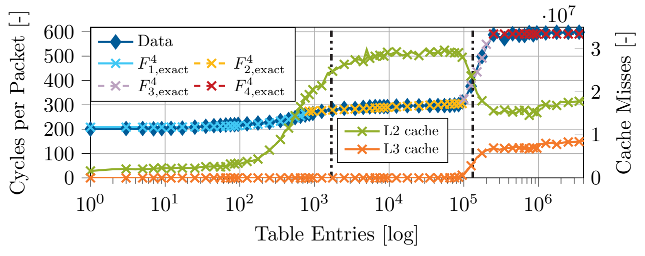

Figure 2: Single core performance on software system for increasing number of match-action table entries

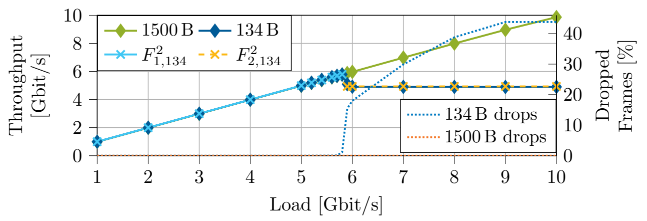

Figure 3: Throughput and percentage of dropped packets for the investigated NFP-4000 platform

NIC. Otherwise, the DuT is equipped with an Agilio CX NFP-4000 SmartNIC from Netronome [1]. Turbo boost and hyperthreading were disabled to reduce performance jitter. The measurements for the t4p4s target were performed in the pos-managed testbed [9], while the NFP-4000 was investigated using the purpose-made testbed. We also extended the framework to gather target-specific metrics on the DuT: CPU cycles and cache misses for t4p4s and frame drops at the ingress buffer for the NFP-4000.

t4p4s Exact Match-Action Table Entries. Fig. 2 shows our reproduction of related work [23] for the t4p4s target when measuring the CPU cycles per packet for an increasing number of exact match entries. The modeling framework has identified four different parts that are modeled individually. These can be explained by cache misses also shown in Fig. 2. Initially, a model with a linear factor of 0.072 is proposed, i.e., the behavior is almost constant for less than 100 table entries. DPDK exact match tables use cuckoo hashing with constant worst-case lookup time [6]. Fetching packet header data used for match keys is also constant as it is independent of the number of table entries. However, for the linearly increasing number of table entries, the memory that has to be loaded is also increasing linearly. Starting with approx. 10^2 table entries, the data no longer fits into the L2 cache, resulting in the increase of L2 misses. At circa 10^3 table entries, the model switches to a logarithmic function. For more than 10^5 table entries, data has to be fetched from main memory as indicated by the increase in L3 cache misses. These fetches take significantly longer than fetches from fast caches, doubling the CPU cycles per packet. This transition period is modeled independently as linear function. After the transition, the load operations from main memory outweigh other effects, resulting in a constant model. The overall model has a combined error of 0.730 %.

To model the memory consumption m^r_e we use a modified version of the model proposed by Scholz et al. [22, Eq. 4]:

$$m^r_e(e, r_k, r_a) = 128\,\text{B} + e \cdot (r_k + r_a + 72\,\text{B}) \tag{5}$$

with the number of table entries e, total key size r_k in bytes, and size of the action data r_a in bytes. We solve Eq. 5 for e, using $r_k = 16\,\text{B}$ and $r_a = 64\,\text{B}$: Setting $m^r_{\text{exact}} = 20\,\text{MB}$, the L3 cache size of the used processor, results in $1.32 \cdot 10^5$ entries to fully fill the L3 cache. This point, marked in Fig. 2, is an overestimation as the cache is not exclusively used for table entries. Similarly, the increase in L2 cache misses can be approximated ($1.68 \cdot 10^3$ table entries for $m^r_{\text{exact}} = 256\,\text{kB}$). While this estimation is close to the point of the detected event, due to access time difference between the L2 and L3 caches of less than 5 ns, the performance loss is not as noticeable as when exceeding the L3 cache [14]. The increase in L1 cache misses is not detected by the automated modeling approach. This is due to the even smaller difference in cache access times from L1 to L2 cache.

NFP-4000 Recirculation. In contrast to the software target, the NFP-4000 is capable of handling a scaled number of match-action table entries without packet loss [10], i.e., processes traffic at line-rate even for a large number of table entries, resulting in a constant model. Instead, we use P4's recirculation feature. Packets are sent once from the egress stage back to the ingress stage that also processes the traffic received from the load generator. For this scenario, MoonGen generates VXLAN packets with a size of 134 B and 1500 B and a load between 1 Gbit/s and 10 Gbit/s.

Fig. 3 shows the received rate and percentage of dropped packets as a function of the send rate for both packet sizes. We observed that the NIC is capable of handling all incoming traffic without packet drops when processing maximum-sized packets. However, when the packet size is set to 134 B, the received rate decreases after 5.7 Gbit/s and then saturates at around 4.9 Gbit/s when further increasing the load. The modeling framework detects this event using a linear and a constant function with a combined error of 0.23 %. The percentage of dropped frames by the NFP-4000 increases proportionally with the observed loss in throughput after 5.7 Gbit/s. This happens because the NFP-4000 fails to process the cumulative throughput of incoming traffic recirculated according to the loaded P4 program. Here, the processing capacity of the NIC is reached when this cumulative throughput approaches the 10 Gbit/s line rate of the NIC. The latter is observed when the packet size is equal to 134 B, i.e., high packet rate, as the ingress buffer of the NFP-4000 is filled faster with original and recirculated packets. This is further confirmed when looking at the percentage of dropped frames collected at the ingress buffer of the NFP-4000. The minimum packet size at which no packet drop is recorded is equal to 275 B for this scenario.

Discussion. While analyzing the impact of the number of table entries is interesting for the software target, it is trivial for the NFP-4000 as it shows a constant model. We chose the presented findings as a case study as they highlight the capabilities of the framework, supporting multiple targets, investigated in different testbeds, using general and target-specific metrics. The generalization of the framework is vital, as it does not only evaluate one component, but all P4 language components that are currently supported. The entire set of microbenchmarks is generated and, depending on the testbed, can be executed and evaluated entirely automatically. Customization is available by selecting metrics of interest or tuning the model's parameters. The resulting reproducible models provide an overview of the target's behavior. Potential outliers can then be further investigated. As future step, we want to use the individual models to deduce a combined model for entire P4 programs. Based on such a model, network operators can decide whether a certain target will suit their requirements, without having to test the program itself.

The framework is limited by its target-dependency. Especially for software targets, a model derived for one platform might not be valid for another. This, however, is counteracted by the huge degree of automation, allowing a new model for a concrete platform to be derived quickly and with low effort. Similar, using individual component models to derive a model for the whole application, e.g., by summing up the models for the components times their occurence in the program, is target specific. Side-effects and interactions, e.g., compiler optimizations, cache effects, etc., will result in inaccuracies, however, the final model will still provide a rough estimate. The model's accuracy is limited by the number of data points used for the curve fitting algorithm. This may become a problem for a language component that permits measurements only with few data points, e.g., adding or removing headers is limited by the maximum packet size.

4 RELATED WORK

As more targets support P4 programmability, evaluating the performance of these targets becomes more crucial to identify the strengths and limitations of these targets for a designated use case scenario. A benchmarking suite for P4 targets was first proposed by Dang et al. [5]. In [11] and [12] benchmarks of different P4 devices identify the influential P4 constructs on packet processing latency. Accordingly, a model that relates the packet processing latency on different P4 targets to the loaded P4 programs is derived. In [21], the key properties of different P4 devices are investigated. For software P4 switches, the packet processing rate is evaluated and modeled, whereby the resource utilization is studied for ASIC-based devices. The study in [10] models the impact of flow scalability on processing latency and the data plane's reaction time to control plane commands. Helm et al. [13] derive analytical models for the performance of P4 and SDN-based switches using network calculus. Lukács et al. [15, 16] propose a probabilistic model of the program execution to calculate the expected cost for a given control flow graph. Thereby, this cost is a result of the sum of the cost of all possible execution paths times their respective execution probability. Through incremental refinement, modeling more parts of the program and providing target-dependent information, Lukács et al. plan to improve the accuracy of the model as future work.

While prior work focuses on benchmarking and modeling the performance of P4 programmable devices, in this work, we focus on providing a methodology and a framework that guarantee a reproducible and automated evaluation of this class of devices.

5 CONCLUSION

We have presented our automated modeling framework that analyzes data plane components in isolation using a model-first approach. The framework focuses on deriving mathematically correct models, which can be used to discover unexpected or wrong performance behavior. Our case study has shown that this is accomplished for different target platforms and metrics. While our modeling approach was developed to analyze data plane components, it can be applied for general network measurement data.

We focus on a high degree of automation to achieve reproducible results. Only a testbed- and a target-specific component have to be contributed for a new testbed or P4 target. These components of the framework have to implement the respective generated specifications.

In the future, we want to combine the models of individual components to predict the performance of entire paths through the data plane. This allows determining average- or worst-case paths, enabling, e.g., worst-case evaluation of the data plane program.

ACKNOWLEDGMENT

The Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) partially funded this project (Modanet, grant no. 397973531, 316878574). We want to thank the anonymous reviewers and our shepherd Antonio Barbalace for their valuable feedback.

REFERENCES

[1] 2016. *NFP-4000 Theory of Operation.* Technical Report. Netronome Systems Inc. https://www.netronome.com/static/app/img/products/silicon-solutions/WP_NFP4000_TOO.pdf Last accessed: 2021-08-23.

[2] Karsten Ahnert and Markus Abel. 2007. Numerical differentiation of experimental data: local versus global methods. *Comput. Phys. Commun.* 177, 10 (2007), 764–774. https://doi.org/10.1016/j.cpc.2007.03.009

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *Computer Communication Review* 44, 3 (2014), 87–95. https://doi.org/10.1145/2656877.2656890

[4] Tiberiu S. Chis. 2016. *Performance Modelling with Adaptive Hidden Markov Models and Discriminatory Processor Sharing Queues.* Ph.D. Dissertation. Imperial College London, UK. http://hdl.handle.net/10044/1/39049

[5] Huynh Tu Dang, Han Wang, Theo Jepsen, Gordon J. Brebner, Changhoon Kim, Jennifer Rexford, Robert Soulé, and Hakim Weatherspoon. 2017. Whippersnapper: A P4 Language Benchmark Suite. In *Proceedings of the Symposium on SDN Research, SOSR 2017, Santa Clara, CA, USA, April 3-4, 2017.* ACM, 95–101. https://doi.org/10.1145/3050220.3050231

[6] DPDK. 2021. DPDK documentation—Hash Library. https://doc.dpdk.org/guides/prog_guide/hash_lib.html Last accessed: 2021-09-14.

[7] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*, Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring (Eds.). ACM, 275–287. https://doi.org/10.1145/2815675.2815692

[8] Sebastian Gallenmüller, Dominik Scholz, Florian Wohlfart, Quirin Scheitle, Paul Emmerich, and Georg Carle. 2018. High-performance Packet Processing and Measurements (Invited Paper). In *10th International Conference on Communication Systems & Networks, COMSNETS 2018, Bengaluru, India, January 3-7, 2018.* IEEE, 1–8. https://doi.org/10.1109/COMSNETS.2018.8328173

[9] Sebastian Gallenmüller, Dominik Scholz, Henning Stubbe, and Georg Carle. 2021. The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments. In *CoNEXT '21: The 17th International Conference on emerging Networking EXperiments and Technologies, Munich, Germany, December, 2021.* ACM. https://doi.org/10.1145/3485983.3494841

[10] Hasanin Harkous, Mu He, Michael Jarschel, Rastin Priest, Ehab Mansour, and Wolfgang Kellerer. 2021. Performance study of P4 programmable devices: Flow scalability and rule update responsiveness. In *IFIP Networking Conference (IFIP Networking).* IEEE.

[11] Hasanin Harkous, Michael Jarschel, Mu He, Rastin Pries, and Wolfgang Kellerer. 2019. Towards Understanding the Performance of P4 Programmable Hardware. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2019, Cambridge, United Kingdom, September 24-25, 2019.* IEEE, 1–6. https://doi.org/10.1109/ANCS.2019.8901881

[12] Hasanin Harkous, Michael Jarschel, Mu He, Rastin Pries, and Wolfgang Kellerer. 2020. P8: P4 with predictable packet processing performance. *IEEE Transactions on Network and Service Management* (2020). https://doi.org/10.1109/TNSM.2020.3030102

[13] Max Helm, Henning Stubbe, Dominik Scholz, Benedikt Jaeger, Sebastian Gallenmüller, Nemanja Deric, Endri Goshi, Hasanin Harkous, Zikai Zhou, Wolfgang Kellerer, and Georg Carle. 2021. Application of Network Calculus Models on Programmable Device Behavior. In *33rd International Teletraffic Congress, ITC 2021, Avignon, France,.*

[14] Intel 2021. *Intel 64 and IA-32 Architectures Optimization Reference Manual.* Intel. https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-optimization-reference-manual.html Last accessed: 2021-08-23.

[15] Dániel Lukács, Gergely Pongrácz, and Máté Tejfel. 2019. Performance guarantees for P4 through cost analysis. In *2019 IEEE 15th International Scientific Conference on Informatics.* IEEE, 000305–000310.

[16] Dániel Lukács, Gergely Pongrácz, and Máté Tejfel. 2021. Control flow based cost analysis for P4. *Open Comput. Sci.* 11, 1 (2021), 70–79. https://doi.org/10.1515/comp-2020-0131

[17] Jorge J Moré. 1978. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis.* Springer, 105–116.

[18] William H Press and Saul A Teukolsky. 1990. Savitzky-Golay smoothing filters. *Computers in Physics* 4, 6 (1990), 669–672.

[19] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 1998. A Metric for Distributions with Applications to Image Databases. In *Proceedings of the Sixth International Conference on Computer Vision (ICCV-98), Bombay, India, January 4-7, 1998.* IEEE Computer Society, 59–66. https://doi.org/10.1109/ICCV.1998.710701

[20] Dominik Scholz. 2021. P4 Component Modeling Framework Repository. https://github.com/p4-modeling Last accessed: 2021-11-26.

[21] Dominik Scholz, Andreas Oeldemann, Fabien Geyer, Sebastian Gallenmüller, Henning Stubbe, Thomas Wild, Andreas Herkersdorf, and Georg Carle. 2019. Cryptographic Hashing in P4 Data Planes. In *2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2019, Cambridge, United Kingdom, September 24-25, 2019.* IEEE, 1–6. https://doi.org/10.1109/ANCS.2019.8901886

[22] Dominik Scholz, Henning Stubbe, Sebastian Gallenmüller, and Georg Carle. 2020. Key Properties of Programmable Data Plane Targets. In *32nd International Teletraffic Congress, ITC 2020, Osaka, Japan, September 22-24, 2020*, Yuming Jiang, Hideyuki Shimonishi, and Kenji Leibnitz (Eds.). IEEE, 114–122. https://doi.org/10.1109/ITC3249928.2020.00022

[23] Dominik Scholz, Henning Stubbe, Sebastian Gallenmüller, and Georg Carle. 2020. Key Properties of Programmable Data Plane Targets. In *ITC.* Osaka, Japan.

[24] The P4 Language Consortium. 2021. The P4 Language Specification, Version 1.2.2. https://p4.org/specs/ Last accessed: 2021-12-03.

[25] The SciPy community. 2021. SciPy documentation—curve_fit. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html Last accessed: 2021-09-14.

[26] Péter Vörös, Dániel Horpácsi, Róbert Kitlei, Dániel Leskó, Máté Tejfel, and Sándor Laki. 2018. T4P4S: A Target-independent Compiler for Protocol-independent Packet Processors. In *IEEE 19th International Conference on High Performance Switching and Routing, HPSR 2018, Bucharest, Romania, June 18-20, 2018.* IEEE, 1–8. https://doi.org/10.1109/HPSR.2018.8850752