Application of Network Calculus Models on Programmable Device Behavior

Max Helm*, Henning Stubbe*, Dominik Scholz*, Benedikt Jaeger*, Sebastian Gallenmüller*, Nemanja Deric†,
Endri Goshi†, Hasanin Harkous†, Zikai Zhou†, Wolfgang Kellerer†, and Georg Carle*
*Chair of Network Architectures and Services, †Chair of Communication Networks,
Technical University of Munich
{helm|stubbe|scholz|jaeger|gallenmu|carle}@net.in.tum.de
{nemanja.deric|endri.goshi|hasanin.harkous|zikai.zhou|wolfgang.kellerer}@tum.de

***Abstract*—Critical applications, such as industrial control systems or remote medical applications, require highly reliable networks. A key enabler of such applications are networks that deliver the required strict performance guarantees. A prominent tool for deriving such guarantees for networks and the involved components is network calculus (NC). Device specifics may have a stark influence on model characteristics, making modeling in heterogeneous environments work-intensive. OpenFlow and P4 are two approaches that emerged from the Software-Defined Networking (SDN) community making networks more flexible and, consequentially, even harder to model.**

In this work, we demonstrate a novel approach that uses NC to model such SDN-based devices despite their increased complexity. Abstracting away from overall device behavior, we initially model only the fundamental building blocks of SDN devices that define network device behavior. NC provides a framework to compose different NC models into a single model, which we use to combine the building blocks into a model that describes a network device program built from these building blocks. This approach allows for modeling a maximal number of devices with a minimal amount of measurements. We apply our approach to two different SDN devices, the Zodiac FX and the NetFPGA SUME. A comparison between the prediction of our composed models and real measurements reveals a prediction error below 1 %, thereby proving the validity of our approach.

***Index Terms*—Network Calculus, SDN, OpenFlow, P4, Modeling, Performance measurements**

I. Introduction

Industrial control applications, robots for medical applications, or self-driving cars are just a few applications that require communication with real-time characteristics, such as a 99.999 % delivery probability with sub-millisecond latency [1]. In recent years, novel approaches emerged targeting flexible, reconfigurable, and programmable networks. These technologies allow tailoring networks to the specific needs of the ultra-reliable, low-latency communication. A significant step towards programmable networks was the introduction of Software-Defined Networking (SDN) [2]. SDN proposes the separation of control and data plane, thus enabling operators to *control and program* networks. P4 advances this concept, introducing a fine-grained platform-independent *data plane* programmability of networking devices [3].

Both approaches increase the complexity of data plane devices, making the provision of performance guarantees more challenging. A common approach to describe deterministic performance guarantees is Network Calculus (NC) [4], [5]. NC requires an appropriate performance description, i.e., an NC service curve, of each deployed networking device. Determining such service curves was comparatively simple for traditional, non-programmable switches that rely on fixed-function processing with constant or negligible processing costs. However, finding an adequate model becomes a complex problem for programmable network devices with highly flexible processing pipelines. To apply NC concepts in programmable networks, authors resorted to measuring the performance of programmable devices while considering only a specific set of functions and configurations [6]. The corresponding service curves are then generated based on the exhibited performance. Focusing only on a selected number of functions and configurations limits the predictive capabilities of the derived models to the previously investigated scenarios.

Motivated by the previous observation, we present a novel modeling methodology based on NC to enable deriving performance guarantees. Instead of modeling a device as a whole, we advocate modeling internal functionalities of a device separately. Our methodology constructs multiple curves, where each curve represents the performance of a specific device function. Utilizing the NC framework, we derive a single service curve of a device by combining the internal service curves. We aim for the following goals: *(I)* We study the isolated performance of specific device functions, simplifying device configuration and accelerating model derivation. *(II)* We combine the derived models for specific device functions to describe device behavior. These specific functions are combined to perform complex packet processing algorithms that create the device behavior. *(III)* We perform measurements to check the validity of our approach on different platforms, a P4 device realized on an FPGA-based platform — the NetFPGA SUME — and an SDN-enabled device implemented in software on the Zodiac FX board.

The remainder of the paper is structured as follows: Section II provides an overview of NC and programmable devices. Section III describes our proposed methodology. Section IV evaluates the application of our methodology to the two aforementioned devices. Section V introduces related work. Section VI concludes this work.

ISBN 978-3-903176-43-0 © 2021 IFIP

II. Background

The following introduces background on NC and two classes of programmable network devices: customizable OpenFlow switches and fully programmable P4 switches following the Portable Switch Architecture (PSA).

A. Network Calculus

NC is a framework utilizing min-plus and max-plus algebra to calculate upper bounds for delay and backlog in communication networks consisting of one or multiple nodes. Such a node is characterized by a service curve, a function of time that describes the resources available at this node.

a) Minimum Service Curve: A minimum service curve β is offered by a node with input function A and output function D iff β is wide-sense increasing, $\beta(0) = 0$, and $D \geq A \otimes \beta$, where $\otimes$ is the min-plus convolution [4]. It describes the minimal amount of resources available at a node.

b) Maximum Service Curve: A maximum service curve β is offered by a node with input function A and output function D iff β is wide-sense increasing, and $D \leq A \otimes \beta$ [4]. It describes the maximal amount of resources available at a node.

c) Service Curve Shapes: While service curves can be arbitrarily complex, two shapes relevant for this paper are the burst-delay- and rate-latency service curves as shown in Equation (1) and Equation (2) respectively. In both examples R describes the rate and T the latency. For the minimum service curve they are the minimal rate and maximal latency, for the maximum service curve they are the maximal rate and minimal latency.

$$\delta_T = \begin{cases} +\infty & \text{if } t > T \\ 0 & \text{else} \end{cases} \tag{1}$$

$$\beta_{R,T} = \begin{cases} R \cdot (t - T) & \text{if } t > T \\ 0 & \text{else} \end{cases} \tag{2}$$

d) Convolution of Rate-Latency Service Curves: The min-plus convolution of n rate-latency service curves can be described as shown in Equation (3), taking the minimum of the rates and the sum of the latencies of all service curves.

$$\beta^{e2e}_{R,T} = \beta_{min(\{R_i|0\leq i<n\}),\sum_{i=0}^{n-1} T_i} \tag{3}$$

B. Programmable Networking Devices

Different approaches are trying to introduce a common standard for programmability to computer networks. In this work, we consider two of those approaches: OpenFlow and P4. In the following, both of them are briefly introduced.

1) OpenFlow Switch: OpenFlow was one of the first SDN standards to manifest the benefits of control and data plane decoupling. For this paper, we focus on the capabilities and performance of the data plane implemented in OpenFlow switches.

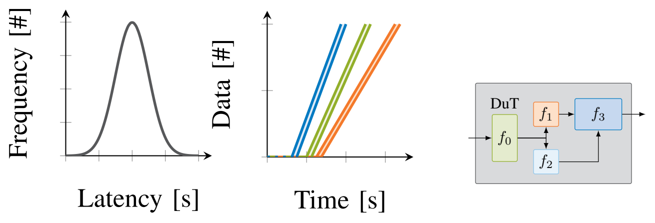
OpenFlow switches process packets according to the entries in dedicated tables: their flow table. This kind of table specifies patterns and actions. OpenFlow switches match newly arriving packets against the patterns contained in the flow table, which defines parameters such as the protocol, header entries, or values. If a packet matches successfully against such an entry in the flow table, the specified action of the associated table entry is performed. Actions can be functions such as forwarding decisions or header manipulations on the packet. The different combinations of table entries are used to realize packet processing algorithms in OpenFlow. Non-matching packets, on the other hand, may be either dropped, forwarded to the control plane, or compared against a different table, depending on the device configuration.

The OpenFlow standard defines the supported protocols, match types, and actions of OpenFlow switches, setting the limits which algorithms can be realized on such switches. Controlling other components of the device's pipeline, e.g., packet parsing and deparsing, requires a higher degree of programmability and is outside the scope of OpenFlow.

2) P4: P4 follows the same fundamental approach as OpenFlow. In other words, all algorithms in P4 are also realized using the previously introduced match-action principle. However, P4 offers a higher degree of freedom for their programmers, as it is not limited to a specified set of protocols and header fields but allows the definition of arbitrary protocols and header fields. This freedom requires additional functional entities compared to OpenFlow, such as parsers and deparsers, before and after the match-action processing units.

In addition, P4 is a device-independent programming language, i.e., it tries to avoid requiring specific functions. Current P4 programs enable writing programs for different network devices, also referred to as hardware targets in context of P4. Regardless of how different these P4 hardware targets, and their P4 language support, may be. As a consequence of this heterogeneity, portability of programs across hardware targets suffers. To mitigate this conflict, PSA was introduced. PSA defines a P4 functionality baseline for targets to support. If a hardware target supports PSA, P4 programmers can assume the availability and structure of certain features. P4, in the form of the PSA, allows to model switch internals as a pipeline of separate basic building blocks. Basic building blocks include, e.g., parsing or modifying a header. Each of those basic building blocks may have more than one characteristic, such as the number of fields in a header or the size of those fields.

3) Comparison: Even though both approaches, OpenFlow and P4, target a similar domain, they differ in fundamental properties. OpenFlow offers a match-action pipeline with a fixed set of supported protocols and allowed operations. P4, on the other hand, offers a more flexible pipeline supported by additional entities such as parsers and deparsers that allow arbitrary protocols. However, parallels between both approaches exist. At their core, both use the match-action principle. The table-based match-action principle defines the basic building blocks used in P4 and OpenFlow to realize more complex algorithms. In the following sections, we investigate if this similarity enables a common modeling approach.



(a) Measurement (b) NC Model (c) Device Model

Figure 1: Measurement results enable NC modeling with different service curve levels. Moreover, devices may be modeled as series of functions.

III. Methodology

Our methodology consists of multiple steps: (1) a definition of service curve levels, (2) measurements, (3) service curve derivations, (4) basic function delta derivation, (5) combination of functions to devices, and (6) verification by measurements. The main advantage of this methodology is that it allows modeling a wide range of different device functionalities with a minimal number of measurements. This is achieved by first measuring and later combining single components. Figure 1 shows the transition from measurement over service curve description to basic function combination.

A. Service Curve Levels

Traditionally, service curves describe the minimum and maximum service available at a node. The service curve parameters of nodes have to be estimated, for example, by measurements. For minimum and maximum rate R_{min} and R_{max}, as well as minimum and maximum latency T_{min} and T_{max}, respectively, these service curves can be defined as:

$$\beta^{min}(t) = \begin{cases} R^{\text{min}} \cdot (t - T^{\text{max}}) & \text{if } t > T^{\text{max}} \\ 0 & \text{else} \end{cases}$$
$$\beta^{max}(t) = \begin{cases} R^{\text{max}} \cdot (t - T^{\text{min}}) & \text{if } t > T^{\text{min}} \\ 0 & \text{else} \end{cases} \tag{4}$$

The service curves shown in Equation (4) are derived from extrema in the measurement data. Since it is hard to deterministically find those parameters using measurements due to the possibility of unrelated measurement artifacts, e.g., equipment faults, we aim to provide different levels of minimum and maximum service curves. Those different levels are based on the type of data from our measurements used to derive them. For example, the n^{th}-percentile η_n of measurements can be used instead of the extrema — with appropriate justifications based on domain knowledge — in order to have a more realistic service curve derivation. Additionally, we can use mean or median values to extend the modeling approach to other frameworks such as queuing theory.

B. Measurements

Our measurement methodology targets the logical function level inside the device and not the physical device level. At the logical level, the device is said to contain a certain number of *basic functions* $\mathbb{F} = \{f_0, f_1, \ldots\}$. One of these basic functions is the *baseline function* f_0 that is required for the device to operate. We measure f_0 as well as each basic function $f_i \in \mathbb{F}' = \mathbb{F} \setminus \{f_0\}$ in combination with f_0. For each f_i, a measurement $\mathbb{M}_i$ is a set of pairs consisting of a measurement packet $p^j \in \mathbb{P}$ and the measured delay for that packet d^j as shown in Equation (5). While $\mathbb{M}_0$ measures f_0, $\mathbb{M}_i$ measures the combination of f_0 with f_i, $\forall i \in \mathbb{N} \wedge f_i \in \mathbb{F}'$.

$$\mathbb{M}_i = \left\{ (p^j, d^j) \mid \forall i \forall j \in [0, |\mathbb{P}| - 1] \subset \mathbb{N}_0 \right\} \tag{5}$$

We also define the set containing all sets of measurement results as $\mathbb{M} = \{\mathbb{M}_i | \forall i \in \mathbb{N}_0 \wedge f_i \in \mathbb{F}\}$.

C. Service Curve Derivation

A service curve of any given level can be derived from a measurement $\mathbb{M}_i$ extracting the associated percentiles, e.g., the extrema or $\eta_{99.999}$, from the distribution of delay values d^j. For example, the latency parameter of the minimum service curve β_i^{min} — i.e., β^{min} of $\mathbb{M}_i$ — can be derived as shown in Equation (6).

$$T^i_{\text{max}} = \max \left(\left\{ d^j \mid \forall (p^j, d^j) \in \mathbb{M}_i \right\} \right) \tag{6}$$

The derived latency value is an upper estimation. Methods for deriving the exact service curve parameters from a set of arrival and departure times, considering cross-traffic and measurement traffic, are surveyed in [7].

D. Basic Function Delta Derivation

Measurements contain delays for f_0 as well as for each $f_i \in \mathbb{F}'$ in combination with f_0. To obtain service curve parameters for each basic function f_i in isolation, we calculate the delta between the service curve function parameters determined from $\mathbb{M}_0$ and $\mathbb{M}_i$. An example of the latency parameter of the minimum service curve of basic function f_1 is shown in Equation (7).

$$T^{1,0}_{\text{max}} = T^1_{\text{max}} - T^0_{\text{max}} \tag{7}$$

The same approach can be used for all elements in the set $\mathbb{M}$, resulting in service curve descriptions of each basic function of the device.

E. Combination of Basic Functions to Device Models

Building on previous results, the different basic functions can now, together with the baseline function, be combined into devices of different functionalities. This approach allows us to model a wide range of possible devices using a small number of measurements because we only need to measure each basic function in isolation and then create a model for any feed-forward chaining of these basic functions representing a specific device with a certain functionality. A device consisting of a baseline function f_0 and an execution path of basic functions $f_1, \ldots, f_n$ can be modeled by a convoluted service curve as shown in Equation (3) where each function corresponds to a single service curve. A depiction of an example convolution of two rate-latency service curves is shown in Figure 2.

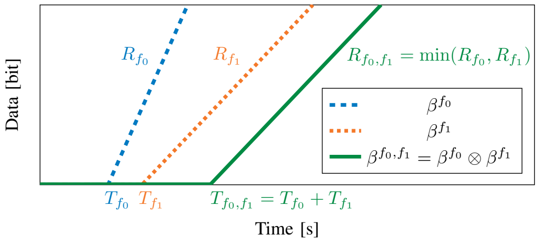


Figure 2: Convolution of two rate-latency service curves β^{f_0} and β^{f_1}.

F. Verification Measurements

To quantify the accuracy of our methodology, we use derived models to make behavior predictions and perform verifying measurements. We predict delay values for device functionalities that consist of multiple basic functions and are not contained in any $\mathbb{M}_i \in \mathbb{M}$. The absolute relative error e_{rel} as shown in Equation (8) gives the accuracy of our methodology for a device configuration consisting of any number of basic functions $f_i \in \mathbb{F}'$.

$$e_{\text{rel}} = \frac{|d_{\text{predicted}} - d_{\text{measured}}|}{d_{\text{measured}}} \tag{8}$$

G. Deriving Rate

The previously presented methodology focused on decomposing and deriving latency parameters of service curves. This can be used to model a burst-delay service curve. To derive a more complex service curve, such as a rate-latency service curve, we also need the offered rate R. We need to consider two different strategies to derive minimum and maximum rate.

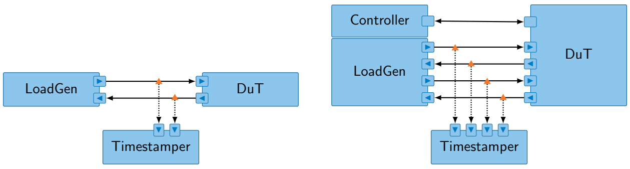
1) Maximum Rate: Most of the networking devices usually offer line rate throughput, or at least a value which is close to it [8]. Therefore, to derive the maximum rate, comprehensive measurements are not needed, as we can assume that each decomposed maximum service curve also has the rate, which corresponds to the line rate of the corresponding device.

2) Minimum Rate: The minimum guaranteed rate, which is needed for the derivation of a minimum rate-latency service curve can be approximated by detecting dropped frames. Measuring with multiple increasing rates r_i, the minimum guaranteed rate is the last rate before we encounter a rate where we observe dropped frames. This means we select r_{i-1} as the minimal guaranteed rate if we observe frame drops at rate r_i. Following this definition, we treat the investigated system as lossless.

3) Additional Considerations: In contrast to high-cost network devices, low-cost network devices often achieve throughput values significantly lower than the line rate. The offered rate can also vary depending on the considered scenario, e.g., the number of rules in the table. Additionally, based on the processing time of each function, in some cases and for some devices, it is possible to derive more precise rate estimates. However, in this paper, we focus on deriving and evaluating the processing time aspect of burst-delay service curves; thus, we consider this as out of our scope.

IV. Evaluation

In this section, we empirically verify the applicability and achievable precision of our proposed methodology to arbitrary devices. To this end, we deploy our methodology on two very different devices. We use one SDN switch (Zodiac FX [9]) and one P4 device (NetFPGA SUME [8]).



(a) NetFPGA [10]

(b) Zodiac FX

Figure 3: Measurement setups employed for the evaluation.

Figure 3 shows the setup for both platforms, which require different measurement setups due to their different requirements. The P4 platform uses 10G fiber optics and does not require a separate controller; the OpenFlow setup uses twisted-pair cables and requires an OpenFlow controller to manage the investigated switch.

The setup for the NetFPGA uses three nodes (Figure 3a): A load generator (LoadGen) running MoonGen [11], the Device under Test (DuT), which is equipped with the NetFPGA SUME, and a third node. This third node uses optical taps to hardware-timestamp the packets between DuT and LoadGen. All three nodes use an Intel quad-core SoC (Xeon-D 1518, 2.2 GHz) running Debian Buster. The LoadGen and timestamper use the integrated Intel 10G X552 dual port NIC.

The setup for Zodiac FX extends the previous setup (cf. Figure 3b). This extended setup uses an additional *Ryu*-based controller [12] which manages Zodiac FX through an OpenFlow 1.0 connection. The data plane traffic is generated by a *scapy*-based [13] packet generator. Furthermore, we use two networking taps to mirror the traffic towards the switch and a measurement card (Endace DAG7.5G4 [14]). The measured processing time is obtained from the measurement card. This processing time also includes the processing time of an integrated switch and internal transmission times.

A. Proof of Concept — SDN Switch

1) Zodiac FX Architecture: The Zodiac FX is a hybrid low-cost SDN switch supporting OpenFlow versions 1.0 and 1.3 [15]. It has in total four physical 100 Mbit/s Ethernet ports, which are internally connected to an integrated L2 switch. The fifth port of the integrated switch is connected to an ARM Cortex-M4 single-core 120 MHz micro-controller (CPU), thus, providing the connection between the CPU and the physical ports. This kind of architecture is common in low-cost SDN devices [6].

When the Zodiac FX is working in SDN enabled mode, all traffic, i.e., data and control plane traffic, received on the physical ports is forwarded to the CPU. The CPU runs a single-threaded infinite software loop, which implements both, OpenFlow control plane agent and OpenFlow data plane pipeline. In other words, all the packet processing is done in software.

2) Motivation & Scenario: As all the packet processing is done in software, this suggests that processing time differs significantly based on the considered scenario. For instance, in OpenFlow 1.0, it is possible to match on 14 different packet headers with varying lengths. The processing time increases with the amount of data to be processed by the switch. Longer headers typically increase the amount of data for processing, hence, indicating that the processing time is highly correlated with header lengths. [16], [17]

We consider the different OpenFlow actions and matches as a basic building block for our modeling approach. In order to evaluate the methodology's applicability, we use all possible combinations of parameters listed in Table I. Apart from considering *standard* OpenFlow match types, like the transport protocol destination port referred to as *tp-dst*, we also consider two additional matching combinations: *five-tuple*, combining all of *ip-src*, *ip-dst*, *tp-src*, *tp-dst*, and *nw-proto*; *all* which conjoins *five-tuple*, *in-port*, *nw-tos*, *dl-src*, and *dl-dst*.

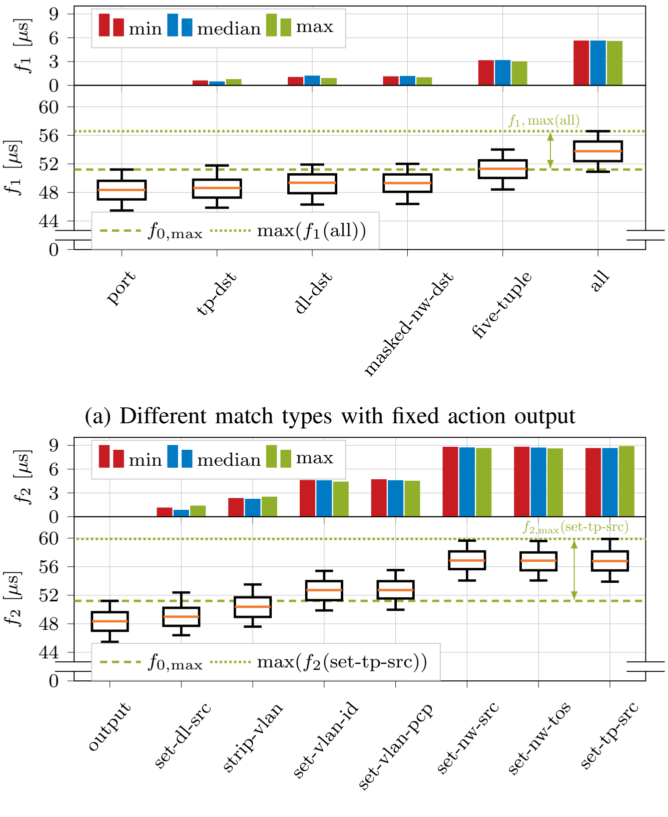
3) Offered Rate: For the aforementioned parameters, Zodiac FX exhibits a worst-case throughput of around 50 Mbit/s. We use this value for modeling the rate part of each decomposed service curve, as discussed in Section III-G.

4) Measurement Data: We divide the entire set of measurements $\mathbb{M}$ into two sets, training set ($\mathbb{M}_{\text{training}}$) and testing set ($\mathbb{M}_{\text{testing}}$). $\mathbb{M}_{\text{training}}$ contains the measurements of the basic building blocks that we use to generate our models. We compose these models to predict the performance of programs built from the basic building blocks. Table I lists the investigated building blocks. $\mathbb{M}_{\text{testing}}$ contains the measurements that we use to validate our composed models. The results of our validation are shown in Figure 5.

Figure 4 shows the measurement results of $\mathbb{M}_{\text{training}}$ for matches and actions separately. Figure 4a demonstrates the measured data for the matches. We use the match on a specific switch port as a baseline measurement. Compared to the baseline measurement, the single-field matches, such as *tp-dst*, *dl-dst*, and *masked-nw-dst*, increase matching time by approximately 2 µs. More complex matches, such as *five-tuple* and *all*, took approximately 3 µs and 6 µs. These increasing numbers indicate that the cost of the matches increase with the amount of data to be matched. Figure 4b measures the latency of the different actions. The baseline measurement, in this case, outputs received packets on a specified switch port. Setting the destination MAC address increases latency by approximately 2 µs. Manipulating the VLAN of a received packet increases latency by approximately 4 to 5 µs. We attribute this increase to the increased complexity of parsing the additional VLAN header fields. Costs rise by approximately 9 µs if network or transport protocol headers are manipulated.

Parameter	**Values**
num. rules	*1*
packet size	*64 B*
match types	*port, tp-dst, dl-dst, masked-nw-dst, five-tuple, all*
action types	*output, set-dl-src, strip-vlan, set-vlan-id, set-nw-src, set-nw-tos, set-tp-src*

Table I: SDN Switch Evaluation Parameters.



(a) Different match types with fixed action output

(b) Different action types with fixed match port

Figure 4: Impact of match and action types on the processing time of Zodiac FX.

We observed a difference between minimum and maximum latencies of roughly 6 µs with the median being almost perfectly centered. This distance was almost the same for the investigated matches and actions alike.

5) Modeling: Even though the *worst-case* processing latency is crucial and the most important metric in NC, as discussed in Section III-A, other metrics are just as important. We use Equation (9) to generate three different processing time estimation models, i.e., models based on *minimum*, *median*, and *maximum* service curve.

$$f(m, a) = f_0 + f_m + f_a \tag{9}$$

f_0 in Equation (9) is the baseline function. This function represents the most basic program executable on the platform.

In the case of an OpenFlow switch, this is a program that performs the action *output* and the match *port*. Thus, sending out packets on a specified port that were received on a certain switch port. The baseline program is the minimal OpenFlow program for which we can determine the forwarding latency described by f_0. Here, m and a are variables representing different match and action types, respectively. $f_m, f_a \in \mathbb{F}'$ denote basic functions, as introduced in Section III. These basic functions incorporate additional impact on processing latency caused by different match and action types.

To derive one estimation model, e.g., *worst-case*-based, we initially determine the value of a constant term f_0. We consider that the constant term f_0 corresponds to the processing time — *minimum*, *median*, or *maximum*. In case of the *worst-case* model $f_0 = 51.058\,\mu\text{s}$. After determining this constant term f_0, we can determine f_m and f_a by subtracting the processing times observed in the corresponding training set, see Figure 4, with this constant.

6) Evaluation: Figure 5 demonstrates the absolute relative error exhibited when applying our model to the training and testing set. Overall, the absolute relative error is always below 1 %, indicating that our methodology is applicable to Zodiac FX, while also expressing high prediction accuracy. Furthermore, we can conclude that our model performed equally well in most cases. In other words, the observed errors do not exhibit high correlation with different match or action types.

B. Proof of Concept — P4 Switch/Device

1) NetFPGA SUME Architecture: For our proof of concept, we use the NetFPGA SUME [8] running a P4 implementation. This FPGA's P4 programmability is achieved by transpiling P4 to VHDL as implemented by the P4→NetFPGA project. Obtained translation results are then integrated into the NetFPGA architecture [18], [19]. As an FPGA-based platform, the NetFPGA SUME employs a Xilinx Virtex-7 690T and supports up to $4 \times 10\,\text{Gbit/s}$. To communicate with its host system, a PCIe interface is used. When programmed with P4, this interface is used to populate tables with match-action pairs during runtime. Furthermore, NetFPGA SUME may forward packets to the host's CPU with this interface.

2) Motivation & Scenario: A translation from high- to low-level languages, such as from P4 to VHDL, is challenging. Even though the process of transpiling between these two languages amounts to a translation between fundamentally different programming paradigms, the thesis, which motivates this research, is that a correlation between the structure of a P4 program and the observable performance of the programmed FPGA exists. An advantage of the FPGA-based platform compared to the CPU-based platform of the Zodiac FX is the creation of a purpose-built packet processing pipeline. The used P4 transpiler creates a processing pipeline that is specific to the executed P4 program. This pipeline can be optimized for a specific processing task in a way that is impossible on CPUs. Also, the processing power is not shared between different processes as it is the case for the switching software on the Zodiac FX. These differences in the architecture lead to higher performance and a highly stable processing latency for NetFPGA-based packet processing tasks.

Based on measurements conducted for this work, we know that the NetFPGA SUME is capable of handling up to and including 9 Gbit/s constant bitrate input traffic consisting of minimum-sized packets. Using this knowledge and applying the approach presented earlier, a model of the board's behavior can be derived. While the methodology is independent of such details, domain knowledge may simplify its application. In this case, understanding of P4.

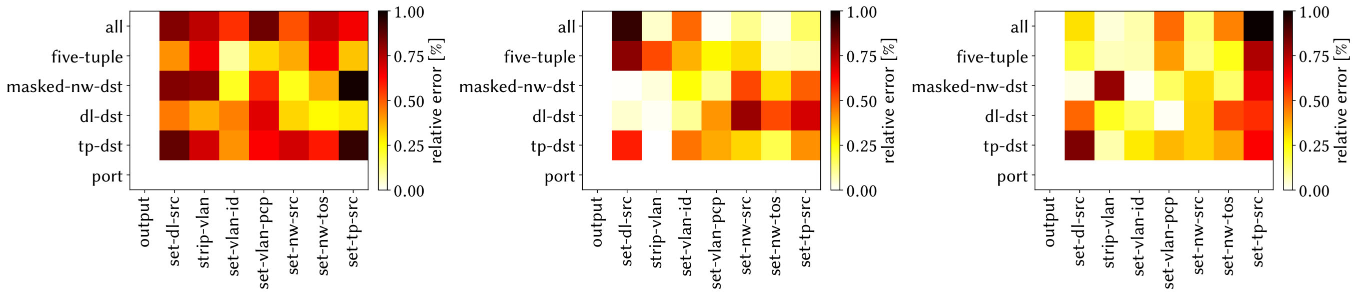
One of the selling points of P4 lies in its flexibility when compared to OpenFlow. OpenFlow is, by its specification, restricted to a fixed set of functionality. Hence, an investigation of an OpenFlow device is limited to supported functions. Contrary to that, P4 devices allow combining more fundamental building blocks almost arbitrarily. These building blocks of P4 offer a more fine-granular packet manipulation than OpenFlow functions. To replicate OpenFlow functionality within P4, several of these building blocks need to be combined. Modeling the performance of a P4-based device only requires the models of these fundamental building blocks that can be composed to predict the performance of the device running a P4 program.

Following this behavioral abstraction, measurements targeting the individual building blocks were conducted. Still, the focus on OpenFlow comparability was decisive. As a result, we effectively conducted the same set of experiments as with the Zodiac FX. One could object that, some measurements, e.g., the action *set-vlan-id* and *set-vlan-pcp*, target the same P4 building block. However, as mentioned, we argue comparability with OpenFlow results outweighs the induced overhead without influence on the necessary measurements.

3) Offered Rate: For all investigated parameter combinations, NetFPGA SUME has a worst-case throughput equal to the applied measurement rate, i.e., 6.6 Gbit/s. However, additional experiments suggest, that this target is able to offer even higher worst-case rates.

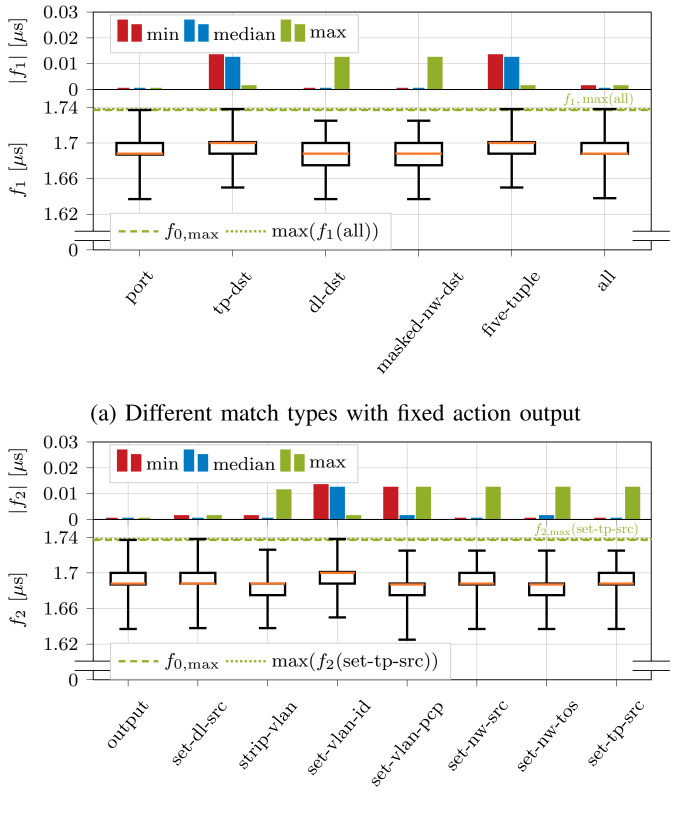
4) Measurement Data: Similar to the OpenFlow investigations, data gained through measurements is divided into two sets — training $\mathbb{M}_{\text{training}}$ and testing $\mathbb{M}_{\text{testing}}$ — whose elements remain as described. Results of our measurements are summarized by Figure 6. Measurements were conducted at two-thirds of NetFPGA SUME's throughput capacity, i.e., 6.6 Gbit/s with a packet size of 68 B, to avoid overloading the DuT during a measurement.

Figure 6 shows the results of the NetFPGA measurements, Figure 6a shows the results of the match measurements, Figure 6b the results of the action measurements. We use the same baseline scenarios as the Zodiac FX, i.e., a match on the ingress port and forwarding to a specified switch port as an action. The difference between the baseline and the other scenarios is below 0.01 µs. It should be noted that the employed measurement equipment has a timer resolution of 0.0125 µs. Considering the resolution of our measurement equipment, the observed measurements were stable across all investigated scenarios. The difference between the respective minimum and maximum values lies within 0.025 µs, i.e., double the



(a) Minimum (b) Median (c) Maximum

Figure 5: Absolute relative error illustrating the achieved precision of the three models (i.e., minimum, median, and maximum) on the Zodiac FX. White parts of the heatmaps correspond to the training set; thus, the error is always 0 %.



(a) Different match types with fixed action output

(b) Different action types with fixed match port

Figure 6: Impact of match and action types on the processing time of NetFPGA SUME.

measurement resolution. This indicates that the measurement scenarios are also highly stable within themselves.

5) Modeling: To derive a model from the presented data, the approach presented in Sections III and IV-A is applied. The baseline measurement for the P4 model consists of a port-based match and the output action. From the measurements follows that $f_0 = 1.867\,\mu\text{s}$. Costs of actions $f_a \in \mathbb{F}'$ lie within the interval of $0\,\mu\text{s}$ to $0.025\,\mu\text{s}$ additional latency. Matching $f_m \in \mathbb{F}'$ induces costs in the same order of magnitude. As a result, that observed latency difference may amount to measurement inaccuracies. However, even the combined differences between the extrema indicate a worst-case deviation below $0.05\,\mu\text{s}$. Therefore, by applying Equation (10), processing time estimation models can be deduced reliably with information gathered from the training set, as depicted in Figure 6.

$$f(m,a) = f_0 + f_a + f_m \tag{10}$$

Considering the note on timer resolution, the observation-based model Equation (10) suggests that neither the investigated matches nor actions had a measurable influence on the observed latency. Given the approximately constant impact of the NetFPGA on overall latency (cf. Figure 6) across all tested match-action types, the model can be simplified further — in this scenario — to a constant offset.

6) Evaluation: Figure 7 shows the absolute relative error we observed, when relying on the derived model to predict performance for the measurements in our test set.

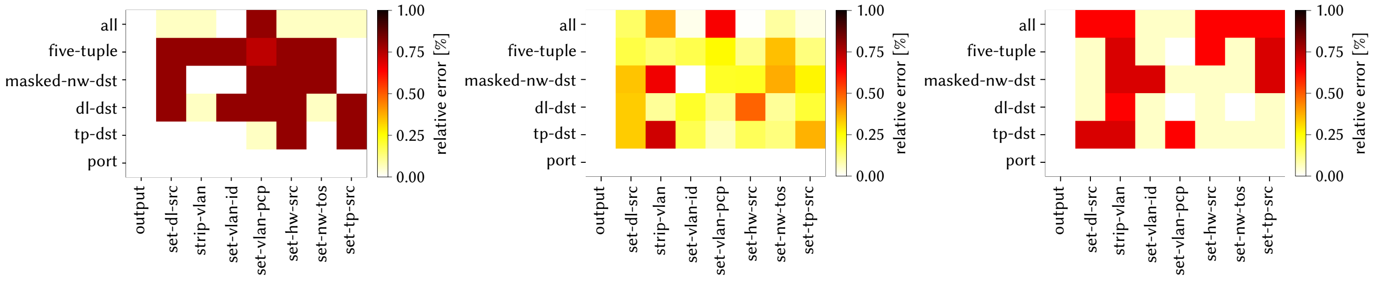
The measurement results suggest that the performance of NetFPGA SUME is not notably impaired by any of the investigated match-action-combination when using our P4 implementation of the respective OpenFlow functionality. While this result could be described as surprising, the rather low prediction relative errors of ±1 % suggest a high prediction accuracy of our derived model. As for the Zodiac FX, also for the NetFPGA SUME, no correlation between match or action types is apparent.

V. Related Work

The approach presented in this paper relates to different areas, network modeling in general but also the benchmarking of network devices.

a) Modeling: Performance modeling of networking devices is widely investigated. While stochastic models based on queuing theory provide information about the mean of different network metrics [20]–[22], network calculus based models are concerned with the worst-case analysis [23].

These different models are applicable for single devices, such as general software switches [24], software DPDK switches [25], OpenFlow switches [20], [26], and virtual network functions (VNF) [27]; and for networks of switches [21],



(a) Minimum

(b) Median

(c) Maximum

Figure 7: Absolute relative error illustrating the achieved precision of the three models (i.e., minimum, median, and maximum) on the NetFPGA SUME. White parts of the heatmaps correspond to the training set; thus, the error is always 0 %.

[28] and VNF chains [29]. Bredel et al. [30] derive network calculus representations using extensive measurements. They inspect both software and hardware routers, using guaranteed rate and packet-scale rate guarantee models. Different error terms based on the measurements are used to refine the derived models. Runge et al. [31] propose a model for the resource management of resource-constrained devices such as software routers. Extending the intra-node model from simulation to real systems, they are successful at predicting performance. Hu et al. [32] take a modular approach using NC to model end-to-end QoS in video transmissions. They derive parameter values for NC curves of specific software and hardware components on the end-to-end path. Fidler [7] provides an extensive survey of service curve derivation methods. Different methods are based on analyzing busy periods, the inversion of the min-plus convolution, or Gaussian cross-traffic models.

Iyer et al. [33] demonstrate an approach that predicts NF performance utilizing symbolic execution. Manousis et al. [34] analyze and predict network performance based on x86-CPU performance counters. Both models provide accurate predictions, however, they require extensive knowledge about the code of the NF itself or the underlying hardware platform.

Unlike previously mentioned papers, this work provides a measurement methodology to derive best-, median-, and worst-case values which can be used to parameterize NC models as well as other approaches. We demonstrated that it is applicable to different SDN-capable switches, such as OpenFlow and P4 programmable devices. In addition, our solution relies on simple black-box measurements and can be applied independent of the underlying hardware platform. Furthermore, we use a methodology that allows us to measure internal functions in isolation and combine them into different devices with a small relative error. This allows for modeling a large range of devices off a comparatively small number of measurements.

b) Benchmarking: Rotsos et al. [16] provide a framework for evaluating different OpenFlow switch implementations based on the NetFPGA platform. They demonstrate an automated framework to benchmark the capabilities of OpenFlow switches. Dang et al. [17] present a benchmarking suite to analyze the performance of P4-enhanced devices. Their framework supports different P4 target platforms, such as FPGA or ASIC, by splitting their suite into target-specific benchmarks, which are executed only for a specific platform, and target-independent benchmarks, which are used across all the available platforms. Further, their benchmarking approach investigates the performance of specific P4 functions. Harkous et al. [35], [36] propose a method for estimating the packet processing latency as a function of the configured P4 program when running on different P4 targets. First, they measure the latency cost of basic P4 constructs when running on three different P4 targets. Then, the measurement results are used to estimate the processing latency of realistic P4 programs, composed of the surveyed P4 constructs. Scholz et al. [37] focus on modeling the critical aspects of devices. For example, ASICs have stable performance, but resources are limited, while for software targets, it is the opposite. They propose performance models, focusing on throughput and latency for the software target and resources for the ASIC. Furthermore, the model focuses on the match-action table component of P4 programs, deepening the understanding of influencing factors and features.

Our methodology tries to bridge the gap between OpenFlow and P4-related benchmarking. Like the papers mentioned before, we investigate specific components of SDN devices. However, we focused on the common functionality between OpenFlow and P4 to be applicable to a wider range of devices.

VI. Conclusion

This paper investigated a methodology that can model modern SDN-capable devices despite their increased flexibility that makes them hard to predict. We cover two different techniques for realizing SDNs: OpenFlow and P4. OpenFlow is a technique that relies on powerful, complex functions that perform the packet manipulation, while P4 relies on a smaller set of less powerful operations that can be combined to describe complex packet processing tasks. Further, we choose devices that cover a wide range of bandwidth demands, an embedded OpenFlow switch platform called Zodiac FX offering 100 Mbit/s ports and the NetFPGA platform that supports P4 with bandwidths of 10 Gbit/s and more.

The advantages of our methodology are twofold. First, we demonstrate the broad applicability. The investigated devices may differ in their capabilities. However, our investigations show that the discussed methodology is not impacted by this. Second, we introduce a methodology consisting of composable models. Both investigated approaches for SDN consist of basic building blocks. By measuring and modeling these building blocks separately, we can derive simple models for the individual building blocks. These simple models can be combined to describe the performance of a more complex program consisting of the original building blocks. We demonstrate that these composed models predict the performance of the real program with an error rate below 1 % for all investigated devices and scenarios.

Given that precision of conducted measurements benefits from expertise in this area, a trait not everybody attributes to themselves, future work should focus on applying the presented methodology to other devices and discussing results in extensive case studies. An increasing number of modeled devices would not only remove obstacles stopping network operators from adopting this approach. Moreover, the NC community would likely benefit from models of devices deployed in real-world scenarios. Additionally, methodologies from related work, such as inversion of the min-plus convolution, can be applied to obtain more precise service curves.

Acknowledgment

The German Research Foundation partially funded this project (Modanet, grant no. CA595/11-1 and KE1863/8-1). Moreover, this work has received funding by the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project 6G Future Lab Bavaria.

References

[1] 3GPP, "22.104 Service requirements for cyber-physical control applications in vertical domains V17.3.0," http://www.3gpp.org/ftp//Specs/archive/22_series/22.104/22104-h30.zip, accessed: 2021-04-30.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, 2008.

[3] 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, 2014.

[4] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*, 2001, vol. 2050.

[5] Y. Jiang and Y. Liu, *Stochastic network calculus*, 2008, vol. 1.

[6] A. V. Bemten, N. Deric, J. Zerwas, A. Blenk, S. Schmid, and W. Kellerer, "Loko: predictable latency in small networks," in *CoNEXT*, 2019.

[7] M. Fidler, "Survey of Deterministic and Stochastic Service Curve Models in the Network Calculus," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 1, pp. 59–86, 2010.

[8] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 5, 2014.

[9] N. Networks. [Online]. Available: https://northboundnetworks.com/

[10] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5G QoS: Impact of Security Functions on Latency," in *NOMS*. IEEE, 2020.

[11] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *IMC*, K. Cho, K. Fukuda, V. S. Pai, and N. Spring, Eds., 2015.

[12] S. Ryu, "Framework community: Ryu SDN framework," *http://osrg.github. io/ryu*, 2015.

[13] P. BIONDI, "Packet generation and network based attacks with scapy," *CanSecWest/core05*, 2005.

[14] D. Endace, "7.5 G2 datasheet," 2012.

[15] NorthboundNetworks, "Northboundnetworks/zodiacfx." [Online]. Available: https://github.com/NorthboundNetworks/ZodiacFX

[16] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: An Open Framework for OpenFlow Switch Evaluation." in *PAM*, vol. 7192, 2012.

[17] 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*. ACM, 2017.

[18] N. Zilberman, Y. Audzevich, G. Kalogeridou, N. Manihatty-Bojan, J. Zhang, and A. Moore, "NetFPGA: Rapid Prototyping of Networking Devices in Open Source," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 2015.

[19] S. Ibanez, G. Brebner, N. McKeown, and N. Zilberman, "The P4-> NetFPGA Workflow for Line-Rate Packet Processing," in *SIGDA*. ACM, 2019.

[20] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia, "Modeling and Performance Evaluation of an OpenFlow Architecture," in *ITC*. IEEE, 2011.

[21] B. Xiong, K. Yang, J. Zhao, W. Li, and K. Li, "Performance evaluation of openflow-based software-defined networks based on queueing model," *Computer Networks*, vol. 102, 2016.

[22] G. Shen, Q. Li, S. Ai, Y. Jiang, M. Xu, and X. Jia, "How Powerful Switches Should be Deployed: A Precise Estimation Based on Queuing Theory," in *INFOCOM*. IEEE, 2019.

[23] A. K. Koohanestani, A. G. Osgouei, H. Saidi, and A. Fanian, "An analytical model for delay bound of openflow based sdn using network calculus," *Journal of Network and Computer Applications*, vol. 96, 2017.

[24] K. Suksomboon, N. Matsumoto, S. Okamoto, M. Hayashi, and Y. Ji, "Configuring a software router by the erlang-k-based packet latency prediction," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, 2018.

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

[26] Y. Goto, B. Ng, W. K. Seah, and Y. Takahashi, "Queueing analysis of software defined network with realistic openflow–based switch model," *Computer Networks*, vol. 164, 2019.

[27] J. Prados-Garzon, P. Ameigeiras, J. J. Ramos-Munoz, P. Andres-Maldonado, and J. M. Lopez-Soler, "Analytical modeling for virtualized network functions," in *ICC Workshops*. IEEE, 2017.

[28] K. Mahmood, A. Chilwan, O. Østerbø, and M. Jarschel, "Modelling of openflow-based software-defined networks: the multiple node case," *IET Networks*, vol. 4, no. 5, 2015.

[29] Q. Ye, W. Zhuang, X. Li, and J. Rao, "End-to-end delay modeling for embedded vnf chains in 5g core networks," *IEEE Internet of Things Journal*, vol. 6, no. 1, 2018.

[30] M. Bredel, Z. Bozakov, and Y. Jiang, "Analyzing router performance using network calculus with external measurements." in *IWQoS*, 2010.

[31] T. M. Runge, B. E. Wolfinger, S. Heckmüller, and A. Abdollahpouri, "A modeling approach for resource management in resource-constrained nodes," *Journal of Networks*, vol. 10, no. 1, 2015.

[32] X. Hu and Z. Lu, "End-to-End System QoS Modeling Based on Network Calculus: A Multi-Media Case Study," in *ICICSE*, 2020.

[33] R. R. Iyer, L. Pedrosa, A. Zaostrovnykh, S. Pirelli, K. J. Argyraki, and G. Candea, "Performance contracts for software network functions," in *NSDI*, J. R. Lorch and M. Yu, Eds., 2019.

[34] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry, "Contention-aware performance prediction for virtualized network functions," in *SIGCOMM*, H. Schulzrinne and V. Misra, Eds., 2020.

[35] H. Harkous, M. Jarschel, M. He, R. Priest, and W. Kellerer, "Towards understanding the performance of p4 programmable hardware," in *ANCS*. IEEE, 2019.

[36] H. Harkous, M. Jarschel, M. He, R. Pries, and W. Kellerer, "P8: P4 with predictable packet processing performance," *IEEE Transactions on Network and Service Management*, 2020.

[37] D. Scholz, H. Stubbe, S. Gallenmüller, and G. Carle, "Key Properties of Programmable Data Plane Targets," in *ITC*, Osaka, Japan, 2020.