Modeling and Prediction of Software-Defined Networks Performance using Queueing Petri Nets

Piotr Rygielski
Institute of Computer Science
University of Würzburg, Germany
piotr.rygielski@uni-wuerzburg.de

Marian Seliuchenko
Dept. of Telecommunication
Lviv Polytechnic National University, Ukraine
m.seliuchenko@gmail.com

Samuel Kounev
Institute of Computer Science
University of Würzburg, Germany
samuel.kounev@uni-wuerzburg.de

ABSTRACT
Using various modeling and simulation approaches for predicting network performance requires extensive experience and involves a number of time consuming manual steps regarding each of the modeling formalisms. Descartes Network Infrastructure (DNI) is a data center network performance modeling approach that addresses this challenge by offering multiple performance models but requiring to use only a single modeling language. In this paper, we thoroughly extend DNI to support new networking paradigms like, among others, Software-Defined Networking (SDN) and Network-Function Virtualization (NFV). Additionally, we demonstrate how SDN-based networks can be modeled using DNI and how are they transformed later into Queueing Petri Nets (QPN) using a model-to-model transformation. In the analysis of the performance prediction accuracy, we show that automatically generated QPN models represent the performance of heterogeneous SDN hardware with maximal prediction accuracy error of 12%.

CCS Concepts
• Networks → Network performance modeling; Network simulations; • Hardware → Networking hardware;

Keywords
performance modeling, software-defined networking, data center networks, meta-modeling

1. INTRODUCTION
Performance modeling and prediction approaches help system operators to analyze data center performance at the system’s design-time and during operation. Nowadays, data centers are becoming increasingly big and dynamic due to the common adoption of virtualization technologies. Virtual machines, data, and services can be migrated on demand between physical hosts to optimize resource utilization while enforcing service-level agreements. This high level of complexity makes an accurate and timely performance analysis a challenging problem [6]. Furthermore, the network infrastructures shift towards virtualization with emergence of such paradigms as Software-Defined Networking (SDN) and Network Functions Virtualization (NFV).

In our research, we focus on the modern network infrastructures of virtualized data centers that leverage SDN or NFV technologies. The network infrastructures in such environments introduce several new challenges for performance analysis. Some examples of such challenges include the growing density of modern virtualized data centers (increasing amount of network end-points), the high volume of intra-data-center traffic (having its source and destination within the same data center), or the new traffic sources introduced in the management layer of virtualized environments (e.g., migration of virtual machines). For SDN and NFV paradigms, new performance-related challenges emerge as the classical hardware-based networking is now tightly bound to software running on commodity servers (e.g., applications in the SDN Controller in SDN-based networks, or virtualized network functions in NFV setups).

Data center networks can be represented in multiple performance modeling formalisms, for example, domain-specific simulation models, stochastic Petri nets, queueing networks, and stochastic process algebras. The modeling with a given performance model requires understanding of its formalism and the usual modeling steps. Thus, specific knowledge and experience with multiple modeling formalisms are required in order to benefit from the variety of their characteristics. Usually, such knowledge and experience is missing or it is limited to a single modeling formalism.

In [16], we proposed an initial modeling approach that requires to model the network using a high-level descriptive language named DNI (Descartes Network Infrastructures). DNI is a generic network modeling formalism and contains elements familiar to any network operator. That approach allows to use multiple various modeling and analysis approaches without requiring in depth expertise in the respective modeling formalisms. Using the descriptive DNI model as a basis, we provide model-to-model transformations to automatically generate predictive performance models without requiring the operator to have expertise in any of them. Network models that are built using DNI modeling language can be automatically transformed to various predictive models. In this paper we present new, substantial changes to the
architecture of the DNI Meta-model and its model-to-model transformations.

1.1 Motivation

Software-Defined Networking and Network Functions Virtualization slowly become ubiquitous in nowadays networks. Most of the networking hardware vendors offer SDN-enabled devices and software solutions to bring more programmability into networks and move the network design from the hardware layer to the software tier. However, new networking paradigms impose a new way of the network workload processing and thus enable new performance bottlenecks that may originate from the software in contrast to the purely hardware-related sources of the performance degradation in the classical networks. Moreover, to enable support for SDN, the hardware vendors need to redesign their equipment by adding new processing workflows. This may lead to the emergence of new performance-influencing factors that were not present in the classical network architectures.

Many authors support the need to investigate the performance of SDN setups in a more throughout manner including the hardware and the software aspect of a network. In [8], the authors investigate various performance-related parameters of an SDN network. The authors note that “performance bottleneck may be located in the existing switches, and the flow table entry installation delay is a pressing issue.” This statement is confirmed by the authors of [11] who investigated control planes of three hardware SDN switches and observed that hardware is still not mature enough for the performance to be repeatedly predicted, because “each switch under test has many quirks which result in unexplained performance changes.” They conclude that “the switch performance is difficult to predict—a single rule can degrade the update rate of a switch by an order of magnitude”. The authors stress high diversity of the performance of the switches. The statements are amplified by the authors of [12], who stress the diversity of switch capabilities and behaviors what makes network harder to understand and control. They observe that rules may be shifted between flow tables in the switch. In some switches rules may be rejected whereas on other, rules may be installed into the software or the hardware flow table. In contrast to this, the authors of [5] find out that building an SDN switch emulator is possible as “an appropriately calibrated emulation infrastructure can approximate the behavior of the switches.”

Many other authors (e.g., [7, 4]) focus on the data-plane of an SDN setup and identify possible bottlenecks in the SDN controllers. In this paper, we do not focus on the SDN controller aspect, although the SDN controllers and the modeling of their performance is supported in DNI. In evaluation of this paper, we analyze proactive networks scenarios, i.e., such a state of the network, where all SDN flow rules are already installed in the switch.

Many SDN-related performance factors cannot be ignored and the performance modeling approaches should take them into consideration despite of the challenges stated in related work. This constitutes the main incentive for the substantial extension of the DNI modeling approach.

1.2 Contributions

The main contributions of this paper are the following.

(i) We present new, substantial changes to the architecture of the DNI meta-model. The newly introduced modeling entities enable modeling of modern network infrastructures (such as SDN, NFV) and load balancing scenarios (load balancing in the sense of both: using multiple network paths and multiple software entities as destinations) while still supporting the classical data center network architectures and virtualization technologies (e.g., VLAN, tunneling) at a high level of abstraction. (ii) Furthermore, we show how the newly introduced modeling entities are transformed from the descriptive DNI model into the predictive QPN models. (iii) We evaluate the automatically generated predictive models with respect to their performance prediction accuracy in three scenarios where we compare the QPN performance predictions against performance measurements of three heterogeneous SDN-enabled HP switches. We show how different switch characteristics can be modeled in DNI and how well the QPN simulation can represent the throughput of the switches. Finally, we discuss the results and describe technical challenges.

1.3 Novelty

The novelty of our approach can be characterized by the following aspects. First, we propose an original descriptive performance modeling language—DNI—capable of modeling modern network virtualization infrastructures based on SDN and NFV paradigm. Additionally, we offer automatic transformation to performance predictive models for performance prediction so the DNI models can be transformed and solved with a single mouse click. Second, thanks to the technology-independence of the DNI modeling language, our approach can be used for modeling novel networking technologies and custom protocols without limiting its scope to a single technology. Third, DNI is the first descriptive model to support all switching modes of an SDN-enabled switch: native, software SDN switching mode, hardware SDN switching mode, and the reactive SDN scenarios with the packet-in-flow_mod message exchange between the switch and the SDN controller. Moreover, the generic character of DNI, the modeling of the virtualization, and the support for the modeling of the software layers allows to model NFV setups where parts of the network functions are offered by commodity servers. Finally, we have characterize the most relevant performance aspect of four heterogeneous SDN-enabled switches that served as a validation case study for DNI and QPN simulation.

1.4 Organization

The rest of this paper is organized as follows. In Section 2, we introduce the foundations of SDN, briefly reviewing the related work on SDN performance prediction approaches. In Section 3, we introduce our approach to performance modeling and prediction. In Section 4, we present the DNI model and its novel features. In Section 5, we present the DNI-to-QPN model transformation, whereas in Section 6, we evaluate the automatically generated simulation models using four heterogeneous switches in three scenarios. We present future work directions and conclude in Section 7.

2. FOUNDATIONS AND RELATED WORK

In this section, we briefly introduce the most relevant aspects of the SDN-based networks, including, so called, the software- and the hardware SDN switching mode. Moreover, we briefly review the related work on performance modeling of SDN and NFV-based networks.
2.1 SDN Performance Foundations

Software-defined networking (SDN) assumes separation of the data plane and the control plane. In the data plane, a switch forwards the packets, whereas in the control plane, algorithms make decisions where the packets should be forwarded to. The control plane is implemented using an SDN controller that is a special software running in a commodity server. The controller makes the forwarding decisions which are later stored in switch forwarding tables.

The forwarding tables contain forwarding rules that define behavior of the switch. The packets arriving to the switch are matched against the table entries and once a match is found, the programmed action is executed (e.g., forward, modify, drop packet). If a matching rule cannot be found, a packet_in message is sent to the controller so that it can react and decide what to do with the packet—we will refer to this as the reactive SDN switching. The controller calculates the decision and sends a flow_mod message to the switch to modify the table. If the controller preconfigures the switches and installs the forwarding rules beforehand, we speak of the proactive SDN switching. The flows installed by the controller remain in the table for the time defined in the timeout parameter (where 0 means no timeout). The communication between the switch and the controller is usually realized with the OpenFlow protocol.

The rules saved in the switch can be exact-match (all match fields are specified explicitly) or wildcard-match (some fields are wildcarded). Exact match rules are stored in the BCAM memory (binary content-addressable memory), whereas the wildcard rules are saved in the TCAM (ternary content-addressable memory) that allows each cell to have three states: 1, 0, and * (wildcard match). TCAM is usually power hungry and expensive [11]. The rules that do not fit to the hardware flow tables (i.e., implemented using hardware memory chips, either in BCAM or TCAM) are placed in the switch SDRAM—so called software flow table. Placement of the rules in the software or hardware flow tables influence the switching performance significantly (as we show in Section 6). If a packet is matched against a rule from the hardware switching table, we observe the hardware SDN switching mode, whereas if the matched rule is placed in the switch SDRAM, we refer to the software SDN switching mode, which is usually slower than the hardware one. Moreover, a rule placed in a given flow table can be later moved (promoted) to another, which may be implement using a faster memory chip. Finally, various switch models provide various performance characteristics of the flow table implementations. More information about SDN and OpenFlow foundations can be found in [14].

2.2 Related Work on Performance Models of SDN-based Networks

There is a large body of existing work on performance modeling of communication networks. However, only several cover SDN and NFV-based networks and even less work put focus on a whole network (i.e., including server virtualization and software), as opposed to, for example, [7], where only parts of a network are modeled.

The topic of performance of SDN-based networks attracts many researchers and some aspects were already investigated in the literature. However, various authors usually focus on selected parts of SDN networks so they miss the overall picture of the system (e.g., they focus on the control path, data path, or a controller [11, 5, 2], or model an SDN network too coarsely [7]. The relatively young concept of SDN resulted in multiple heterogeneous hardware products that offer the support for the OpenFlow protocol. This variety resulted in different implementations and thus the offered performance may differ among the vendors, switch models, or even among the versions of the same switch model.

Jarschel et al. [7] model an SDN switch using two queues (M/M/1-S and M/GI/1-S) and thus specify two processing paths (with and without the controller) with different performance. The selection of the controller and non-controller path is modeled in a probabilistic manner. Unfortunately, the software SDN switching mode, which involves CPU and SDRAM processing on a switch, is not modeled making the results applicable only to a switch with unlimited BCAM/TCAM capacity. Similar work was conducted by Azodolmolky et al. [1], where the authors propose an analytical performance model similar to the one proposed by Jarschel et al. The authors validate the model against results that were obtained in the literature and do not investigate any real hardware on their own.

In the NFV-related literature, the authors of [19] claim that many performance problems in the modern NFV testbeds are located in the software of the virtualized network function. This implies that the analysis of the software performance needs to be taken into account to accurately predict the performance of an NFV-based network. Influence of the server virtualization on the ClickOS router was the main focus in [13]. The authors analyzed software-emulated networking hardware and found out that a network function that is virtualized may increase switching latencies and decrease switching capacity. This has a visible influence on the performance of an NFV-based network.

In the contrast to the related work, we do not limit our focus to selected parts of the system but treat the data center as a whole (including the software architecture using the Descartes Modeling Language (DML) [10]). By widening the modeling scope, we represent the system at a higher level of details as we aim to provide a good-enough performance prediction in a timely manner for run-time capacity planning purposes. Moreover, the DNI meta-model presented in Section 4 is the first descriptive performance model capable of modeling SDN and NFV-based network infrastructures.

3. APPROACH TO MODELING AND PERFORMANCE PREDICTION

The wide variety of performance models makes it challenging to select the proper models and learn them extensively to model the performance accurately. Model-based approaches assume that there exists a single descriptive model and all predictive models are derived automatically using model transformations. The general performance prediction process based on model-to-model transformations is presented in Figure 1a. A model of a real network is built and stored in a descriptive form with performance annotations, whereas the transformations to predictive models are automated and can deliver as many different predictive performance models as many model transformations are available. The prediction results can be used to further refine the descriptive models (the dashed line in Fig. 1a).

We divide the area of performance models as shown in Figure 1b. We intentionally exclude the simulation models
4. MODELING NETWORK INFRASTRUCTURES USING DNI

In this Section, we present the new DNI meta-model designed to model data center SDN- and NFV-based networks for performance prediction purposes. Later, in Section 5, we demonstrate how do we transform the DNI models to QPNs.

Since its last version (DNIv2) [16], the DNI meta-model was redesigned and extended to support SDN, NFV, and load-balancing scenarios (e.g., Equal-Cost Multi Path Routing (ECMP)). We use the DNI meta-model to describe a network infrastructure. The DNI meta-model—initially presented in the work-in-progress paper [17] (e1) and later extended in [16] (v2)—is intended to describe the common network components in an abstract manner. In this section, we present the new aspects of the DNI model (DNIv3).

The DNI meta-model covers three main parts of every data center network infrastructure: network structure, network traffic and network configuration. The network structure is intended to model the structure (topology) of the network. The meta-model for network structure contains entities such as nodes and links connected through network interfaces. Nodes can be nested to represent server virtualization. We describe the performance-relevant parameters of every performance-relevant element in the model. The DNI network structure meta-model is presented in Figure 2.

We characterize the network nodes as end (e.g., virtual machine, server) and intermediate (e.g., switch, router), because their roles (and thus the way they influence the performance) are different. An intermediate node represents a node that only forwards the traffic between its ports; an end node usually represents a server or a virtual machine that is a traffic generator, traffic sink, or both. Moreover, a node can be both end and intermediate simultaneously that enables modeling NFV scenarios where commodity servers serve as network devices. A node that is neither End nor Intermediate is assumed to have no influence on the performance and is usually used in the model to reflect the topology of the network.

Nodes can host other nodes (e.g., VM). In the transformations, we currently support only one level of virtualization (i.e., no VMs in a VM), but the modeling formalism does not prohibit multi-level virtualization. Nodes are connected using Links and Network Interfaces that have their respective performance descriptions. Any DNI entity missing a performance description is assumed to offer infinite performance—the element in the model has purely descriptive role.

Additionally, a Node can be either SDN or Common (IType). Common nodes are described by their Performance (end or intermediate respectively), whereas the SDN nodes with the PerformanceSdnNode that exclusively defines their performance in SDN modes: software- or hardware SdnSwitchingPerformance as some devices offer only one mode while other support modes simultaneously. The decision how a flow is processed (using nonSDN, software, or hardware SDN switching mode) depends on the SdbFlowRules. An SdnFlowRule defines probabilities (see Fig. 4) for a given Flow on a given Node to be processed using the SDN controller (reactive scenarios), software SDN, or the hardware SDN switching mode.

The SdnController is a special type of a CommunicationApplication that represents software deployed on an End...
node. The SdnController hosts SdnControllerApplications that are responsible for processing the traffic forwarded to the controller. An SdnControllerApplication applies delay to the switching process and installs the rules in the switch. Unfortunately, installing the rule cannot be modeled directly in DNI (the model cannot be changed during solving) and the rule installation needs to be expressed using the probabilities located in the SdnFlowRule entities.

In the DNI meta-model, network traffic is generated by TrafficSources that are deployed on end nodes. Each traffic source generates traffic Flows that have exactly one source and possibly multiple destinations. Flows can be composed in a Workload that defines how each flow is generated (e.g., with sequences, loops, or branches). Flows are described using the amount of transferred data in the GenericFlowTraffic entity. The traffic description proposed in this paper covers all possible open workloads including traffic sources, sinks, and traffic profile characteristics. The meta-model and its transformations can be systematically extended to support other flow descriptions, e.g., [3]. The traffic metamodel is presented graphically in Figure 3.

Figure 3: Traffic representation of the DNI meta-model. Dotted boxes represent other DNI entities.

The NetworkConfiguration contains information about SDN configuration, routes (paths), protocols and protocols stacks. We use this information to calculate the paths in the topology graph and to coarsely estimate the overheads introduced by the protocols. In DNI, we describe a snapshot of the currently used routes in the system, disregarding if the system uses static or dynamic routing. The protocols are described by a set of generic parameters such as, MTU (maximal transfer unit) and overheads introduced by the data unit headers. We depict the configuration meta-model in Figure 4.

Figure 4: Configuration representation of the DNI meta-model. Dotted boxes represent other DNI entities.

In the meta-model, a path between two nodes is defined as a set of Directions. Each Direction defines via which network interface on a given node, the given Flow should be directed. Additionally, a probability of taking this path is given, so that load-balancing can be modeled. The modeling approach allows to define a network-path load-balancing (node A and B are connected using multiple network paths) and destination application load-balancing (for flows having multiple destinations the ratio of traffic division is defined using the probability parameter).

5. TRANSFORMATION DNI TO QPN

An instance of the DNI meta model is a descriptive model of a network. To conduct performance analysis, the instance (the DNI model) must be transformed into a predictive model. In this section, we describe the transformation that transforms a DNI model into a QPN model, which can be simulated using the SimQPN simulator. The SimQPN...
Depository in delay = latency + mod in (Fig. 7) is identical and implements the node forwarding per-switching abilities are mapped on to the firing weights represented SDN switching mode. Figure 7. Two separate subnets (hw- and software SDN switching mode) respectively. The SdnFlowRule probabilities are mapped on to the firing weights represented by parameters $A$, $B$, and $C$ in the Figure. The internal structure of the switching (in Fig. 6a), hw- and swSdnSwitching (Fig. 7) is identical and implements the node forwarding performance as given by equation forwarding delay = latency + max(capacity, bandwidth).

The in the third processing mode all packets are passed to the controller. This is represented using the buffer and toController places. The traffic tokens (color $t$) directed to the SDN controller are forwarded to the toController place. Next, the controller-tr transition issues a new token with packet_in message (color $p$) and forwards it via output place to the node where the controller is deployed. At the same time, the traffic token (color $t$) is deposited in the buffer until the controller responds. When the controller responds with a flow_mod token (color $f$), the traffic token is released form the buffer and switched using hwSdnSwitching.

The SdnController subnet—depicted in Figure 8a—is responsible for receiving the packet_in tokens and issuing a flow_mod reply. The SdnController subnet is a special type of a traffic source located in a node. The tokens arriving to the controller are delayed twice. First, the delay of the controller itself and the delay of the respective controllerApp that represent internal controller components programmed by SDN developers.

For scenarios with load-balancing the routing-transition is organized differently. The transition (as depicted in Fig. 6a) is replaced by multiple transitions, each with a single mode but different firing weight. The firing weight represents relative firing frequency of the transition, so that it can properly represent load-balancing ratios. An example of SDN switching and “60/40” load-balancing is depicted in Figure 8b. In this example, the tokens consumed from the sdnSwitching place are interchangeably deposited to port port-1-tx and port-2-tx with probabilities 0.6 and 0.4 respectively.

More information about the transformation can be obtained by looking at the examples and its source code online. The transformation with example models are available publicly in our git repository\textsuperscript{1}. The validation of the automatically-generated QPN simulation model is presented in Section 6.

6. VALIDATION

In this section, we validate the approach by comparing the performance predicted by the generated QPN models against their respective physical hardware setups. We analyze the prediction accuracy and focus mainly on the new SDN capabilities of DNI.

Although the modeling capabilities of DNI and QPN are large, in this Section, we focus on simple topologies and relatively simple workloads but use heterogeneous SDN hardware that differs in almost every SDN performance aspect. We conduct three experiments to investigate the factors that influence the switching performance the most: switching latency and switching capacity. We limit the validation to proactive SDN scenarios (no traffic is forwarded to the SDN controller), as the controller has no influence on the performance heterogeneity of the switches.

6.1 Testbed

Our testbed consists of two servers connected with a switch. The servers are exchanging data over the network organized in a dumbbell topology. In the experiments, we replace the switch with other models to investigate their heterogeneity and its influence on the performance characteristics. We present the set of available switches and their brief performance characteristics in Table 1. All switches used in the experiment support OpenFlow protocol in version at least 1.0. All connections are 1Gbps copper cables.

\textsuperscript{1}DNI resources: http://descartes.tools/dni
Switch models named 5130 and 5700 (Comware product line) run under control of a different operating system than the two 2920 and 3500 (ProVision product line). The Comware switches support only hardware SDN switching whereas ProVision support both, hardware and software modes. According to the data sheets, the switches are heterogeneous not only to their data plane performance but also built-in control plane hardware. The 2920 for example runs a Tri-Core ARM1176 processor at 625 MHz and uses 512 MB SDRAM memory; the 3500 is an older model and is equipped with a Freescale PowerPC 8540 processor at 666 MHz and 256 MB DDR SDRAM. The vendor specify the Comware switches as follows: 1000MHz CPU and 2GB SDRAM memory for the 5700 and 1GB for the 5130.

### 6.2 Experiment Setup

The two commodity servers handle the traffic using **iperf** tool. A separate server runs the HP VAN SDN Controller but this has no influence on the measurements as we do not investigate reactive rule-insertion scenarios. The testbed is controlled from an experiment controller that is connected over a separate, isolated network to not interfere with the experiment traffic.

Every experiment starts with a configuration of the SDN flow tables. The rules forcing a specific behavior are pushed to the switches over the SDN Controller. We use wildcard matching rules with defined source and destination IP addresses, as the ProVision switches do not support SDN switching using MAC addresses. As default action for a matching rule, we set “output to port”. The rule timeout is set to 0, so the rules are never removed by the switch. For ProVision switches, matching against the software flow rules is achieved with filling the hardware flow table with dummy rules (rules that can never be matched) as long as the hardware flow table capacity is reached, so the next rule is automatically installed into slower software flow table.

### 6.3 Heterogeneity in SDN Support

As shown in [11], the capacity of hardware flow tables varies from 750 to 2000 rules so the probability of filling the hardware table completely is non-negligible and should be considered in the analysis. Our observations complement the results of [11]; we observe the following maximal capacities of the hardware flow tables: 460 for the 2920, 381 – 1526 for the 3500, 384 – 512 for the 5130, and 512 – 640 for the 5700. The Comware switches offer an additional memory allowing for matching exclusive against the MAC and IP addresses with capacities 16000 and 65535 rules for 5130 and 5700 respectively. The rest of the observations about switches heterogeneity are abstracted in this paper as they have no or little influence on the data-plane performance.

### 6.4 Experiment 1: Maximal Throughput

In the first experiment, we measure the maximal throughput achieved for each switch in the available SDN modes;
the Comware switches in hardware and the ProVision in hardware- and software SDN switching modes. We measured the maximal throughput of the using the values delivered by `iperf` benchmark tool using TCP as the transport protocol.

We simulate the scenario by building a DNI model using the parameters from Table 1. The parameters for the software SDN switching mode (switching latency) were estimated as the official data sheets do not specify them. The maximum switch capacity was acquired from the operating system of the switches, whereas the switching latency was determined empirically. The constructed DNI model was later transformed using the model transformation presented in Section 4 and the resulting QPN model was solved using `SimQPN`. The measured (real) and predicted (simulated) results are presented in Table 2. Additionally, we measure and predict the native (non-SDN) switching throughput. The values were identical to the SDN switching in the hardware SDN switching mode.

The predicted results truly depict the measured performance. The variations of the predicted maximal throughput hold within 1% relative difference for the hardware SDN switching mode (and the not shown native switching). The throughput reported for software SDN switching mode of the 2920 and 3500 switches was bound by two parameters: maximum switching capacity expressed in packets per second (pps) and the switching latency that aggregates multiple internal latencies in the switch that we abstract in DNI (e.g., flow table lookup).

Taking the 3500 switch as an example, we can calculate the upper bound of the switching throughput (assuming switching latency= 0) using the default IP packet size (MTU=1500B) multiplied by maximal switching capacity in packets per second (pps) and the switching latency that aggregates multiple internal latencies in the switch that we abstract in DNI (e.g., flow table lookup).

The results of the SimQPN simulation of software SDN switching mode include additional data that presents the predicted behavior of the switch (and also the sensitivity of the DNI model) for other switching latencies. We observe that the simulation properly represents the measured values. Only for low MTU values the performance for hardware SDN switching reported by simulation was underestimated. This may be caused by the fact, that DNI represented the full protocol stack (TCP/IP/Ethernet) and the additional overhead of the Ethernet frame was also the part of the switching in the simulation, whereas the real switch has decapsulated the IP packet and conducted L3-switching (what is usually not expected in the switching). In the end, the switch has processed about 14 bytes less per packet (11% less for MTU=128) than in the simulation.

In the software SDN switching mode the throughput was underestimated by maximally 12% for 900B and latency 90µs. Additionally, we show how similar switching latencies influence the reported throughput as the precise measurement of the real switching latency was impossible. The value used in other experiments (90µs) was marked in bold.

### 6.6 Experiment 3: Switching Capacity

In the third experiment, we focus on the switching capacity and its influence on the performance of the ProVision...
switches in the software SDN switching mode. Using the operating system of the switch, we lowered the switching capacity stepwise and analyzed the offered performance. The operating system of the Comware switches does not support neither limiting of the switching capacity nor the software SDN switching mode.

6.6.1 Switch 2920

The 2920 switch in the software SDN switching mode offers maximal switching capacity of 2000 packets per second. We varied the value of the maximal switching capacity parameter to investigate the throughput curve of the switch and the curve predicted by the QPN simulation. Additionally, we investigated selected values of switching latencies to present the modeling alternatives. The results are depicted in Figure 9.

Figure 9: Throughput of the 2920 switch in the software SDN switching mode for variable switching capacity.

The throughput curves presented in Figure 9 represent the measured performance less accurately than in the previous experiments—the prediction errors reach even $\approx 100\%$ relatively (for 340µs and 500pps). The switching latency in the DNI model was calibrated to 340µs to properly represent the throughput at the maximal switching capacity of 2000pps. The worse fit of the model for other capacities can be explained as follows. First, the switch in the software SDN switching mode demands more CPU (up to 100%) and the CPU is not modeled in DNI. Second, the switch’s CPU conducted also other operations during the experiment (logging, SDN counters update) that influenced its load but were not possible to control or disable. Finally, the software SDN switching mode received only 20% of the resources available to the 3500 switch so offering maximal throughput of 14Mbps must impose that the switch is incapable of using this mode efficiently. The DNI model could include more performance influencing factors to capture such situations better, but we intentionally abstract them out to keep the generic character of the model while accepting possible inaccuracies to some degree.

6.6.2 Switch 3500

The 3500 switch in the software SDN switching mode offers maximal capacity of 10000 packets per second. Similarly to the switch 2920, we varied the value of the maximal switching capacity and switching latency. The results are depicted in Figure 10. In this experiment, the QPN model represented the switch performance better than for the 2920 switch. The throughput curve grows nearly linearly and the incline is properly represented using the switching latency between 80 and 90µs. Based on the measurements, we observe, that the CPU of the 3500 switch has more processing power, although it has similar frequency and half of the memory size compared to the 2920.

6.6.3 Conclusions: Software SDN Switching

The experiments with the performance in the software SDN switching mode lead to the following conclusions. First, the throughput in the software SDN switching mode is low—about 10 to 70 times lower than the throughput in the hardware SDN switching mode. Second, the behavior of the switch in the software mode is challenging to predict as it depends on the switch’s CPU that is difficult to observe and model accurately. Third, the capacity of hardware flow table in the investigated switches is limited and can be easily reached for complex SDN scenarios (e.g., SDN QoS provisioning based on customer class). The probability of placing a rule in the software flow table cannot be neglected as the performance penalty is high. Moreover, some switches (e.g., 5130, 5700) do not offer software SDN switching mode and report an error if the hardware flow table is full. Finally, DNI is the first model to support the modeling of the SDN performance in the software SDN switching mode, although it does not support modeling of the switch internal architecture (CPU) and the prediction accuracy for the 2920 switch could be further optimized. All in all, the software SDN switching mode should be avoided in general as the performance penalty is high.

7. CONCLUSION

In this paper, we presented a redesigned generic model-based approach to network performance prediction. We introduced the modeling entities allowing to represent SDN, NFV and load-balancing scenarios without limiting the DNI’s ability to represent end-to-end data center network scenarios including server virtualization and even software architectures (when used with DML). Furthermore, we presented the DNI-to-QPN model transformation, so that the descriptive DNI models can be automatically transformed into QPN simulations and solved.

We characterized four models of SDN-enabled switches and validated the prediction capabilities of DNI in three challenging scenarios. We showed that the transformation works correctly and the prediction accuracy is good for runtime prediction purposes. We stress that the presented pre-
dictions were obtained through an automatically generated simulation model from a high-level descriptive model where most low-level details were abstracted, so the prediction errors are acceptable to some degree.

The previous version of DNI (v2) could be transformed into five predictive models (two additional are currently under development). As part of our future work, we will update the remaining model transformations to support the DNIv3, i.e., SDN, NFV and load-balancing scenarios. Finally, we aim to provide more transformations to models with different granularities to enable flexibility in performance prediction of virtualized networks, so that simulation at different level of details can be used depending on the required accuracy and time constraints.

8. REFERENCES


