A new model for DPDK-based virtual switches

Zidong Su, Bruno Baynat, Thomas Begin

A new model for DPDK-based virtual switches

Zidong SU*† Bruno BAYNAT* Thomas BEGIN†
*Sorbonne Université UPMC Univ Paris 06, CNRS, LIP6 UMR 7606 - Paris, France
†Université Lyon 1, ENS Lyon, Inria, CNRS, UMR 5668 - Lyon, France

***Abstract*—In an SDN/NFV-enabled network, the behavior of virtual switches is a major concern in determining the overall network performance. The prominent open-source solution for virtual switching is Open vSwitch while the DPDK library has been developed to accelerate the packet processing. In this paper, we develop a general framework for the modeling and the analysis of DPDK-based virtual switches, taking into account the switch-over times (amount of time needed for a CPU core to switch from one input queue to another). Our model delivers performance metrics such as the buffer occupancy, the loss rate and the sojourn time of a packet in RX queues. We compare our new model with two existing models. Numerical results show that our model combines the accuracy of one model and the efficiency of the other.**

***Keywords*—NFV; virtual switch; DPDK; modeling; performance evaluation; polling system; switch-over time.**

I. Introduction

The softwarization of networking is profoundly reshaping the landscape of telecom and computer networks. ICT industries envision costs optimization through a better management of their resources and a faster deployment of their services. Two technologies are key enablers to this end. First, Software-defined networking (SDN) outsources all the decision-making networking functions into a (set of) controller(s) in charge of determining how to handle the incoming traffic. Hence, networking devices, such as routers, load balancers, and firewalls are replaced by appliances receiving their instructions directly from the controller(s) (using a standard interface like OpenFlow [1]). These appliances can thus be reprogrammed at will by the controller(s). Second, Network function virtualization (NFV) refers to the gradual move of network functions out of dedicated hardware devices and into software. Routers, firewalls, load balancers and other networking devices are then running virtualized on commodity hardware, like a standard x86 server. Therefore, in an SDN/NFV-enabled network, the nodes forwarding, filtering or performing other advanced operations on the traffic are typically referred to as *virtual switches* because they are software-implemented and not deciding the rules.

The network programmability combined with the softwarization of its functions allows a greater extent of flexibility in the way network operators handle their resources. A good case in point is the dimensioning of a network. Instead of the usual overprovisioning strategy, an operator will scale up and down its resources as the demand varies. For instance, when more bandwidth is required on a virtual switch, additional physical resources, e.g., CPU cores, can be provisioned to take part of the load.

In a 2016 paper, Artero et al. [2] describe an analytical model for virtual switches based on the decomposition of the switch into several poling systems, each one being decomposed into simple Markov chains. Presented as a first step towards a more general model, this work assumes a negligible switch-over time (time spent by a CPU core to switch from one input queue to the next one). In [3], Sohail presents another possible model that takes into account the switch-over time, but that relies on the analysis of multi-dimensional Markov chains, resulting in a time-consuming algorithm. The objective of the current paper is to combine the simplicity of the first model and the ability of the second one to take into account switch-over times. To this end, we develop a general framework that first decomposes the virtual switch into several poling systems and then subsequently decompose each polling system into several queueing systems with server vacation. The two pre-cited papers, as well as the current paper, fall within the scope of this framework, and differ by the way the vacation is represented and the corresponding queueing system is analyzed.

The remainder of the paper is as follows. In Section II, we describe the internal architecture of a virtual switch. Section III describes our new proposed model. Section IV covers the numerical results that validate our model accuracy. Section V concludes this paper.

II. System description

A. Virtual switches solutions

In SDN/NFV-enabled networks, virtual switches are in charge of the data plane. They are software implemented and designed to run on commodity hardware, either directly on the appliance or, more commonly, in a virtual environment. Therefore they may have to share physical resources with other virtual machines (VMs) running on the same appliance. Following rules received from the SDN controller(s), virtual switches basically commute incoming packets between their (physical or logical) ports. They may also perform other operations like filtering, headers editing, and encrypting. A couple of proprietary virtual switch solutions have been released; e.g. by Cisco (Nexus 1000V) and VMware (vSphere Distributed Switch). In the open-source domain, the most prominent solution is Open vSwitch (OvS) [4]. Although OvS works in Linux hypervisors such as Xen and KVM, and is integrated into OpenStack [5], virtual switches performance are usually seen a bit low for commuting very high rates of packets. Hence several techniques, e.g., Netmap [6], OpenOnload [7], PacketShader [8] and DPDK [9], have been developed to provide a faster packet processing. In this paper, we focus more specifically on the DPDK library [10].

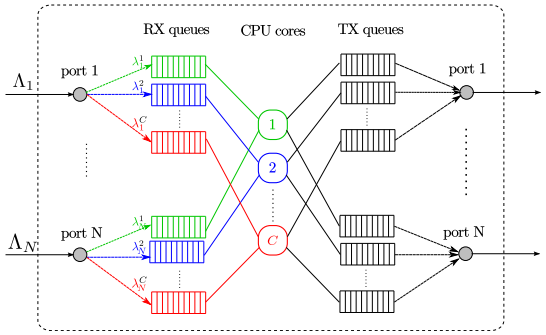


Fig. 1. Internal architecture of a virtual switch with N ports and C cores.

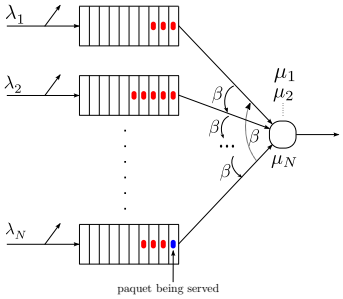


Fig. 2. Subsystem involving a single CPU core polling several RX-queues.

B. Internal architecture

Hardware-wise, a virtual switch includes RAM, several I/O ports, and a set of (physical or logical) CPU cores. If DPDK is enabled, each core polls cyclically all the ports, as illustrated by Figure 1.

Upon arrival on a port, packets are immediately dispatched in separate queues, called RX queues. This dispatch aims at spreading evenly the load among the RX queues. This first step is typically carried out using a hash function on the packet headers (e.g., Receive Side Scaling). On the other hand, each core is assigned to a single RX queue of each port (so that a core handles as many RX queues as the number of ports in the virtual switch). It follows that the total number of RX queues is thus equal to the product of the number of cores by the number of ports. When a core starts polling an RX queue, it processes the first-in-line packet, if any, (or the M-first if the batch mode is enabled) before moving to the following RX queue. We denote the time taken by a core to switch from its current RX queue to the next one as the the switch-over time.

From the standpoint of cores, a core processes packets from various RX queues in a polling fashion, as depicted by Figure 1. To complete the processing of a packet, a core needs at least to read (and edit) its headers, extract the destination address, and find out to which output port the packet must be forwarded. The processing task can include additional steps like flow-specific operations (e.g., deep packet inspection, encryption, QoS monitoring). Afterward, the packet is (logically) forwarded from its RX queue to the TX queue associated to the appropriate output port. From this time on, the packet is simply waiting for its transmission on the next link.

Given the current transfer rates of SDRAM and the data rates of communication links, the bottleneck of a virtual switch, if any, lies in the packet processing steps. Hence, we concentrate our modeling efforts on the interactions between the CPU cores and the RX queues.

C. System notation

We conclude this section by introducing notation used throughout this paper. We denote by C the total number of allocated (physical or logical) CPU cores. We let N represent the number of ports attached to the virtual switch. We use Λ_i to denote the packet arrival rate on each port i $(i = 1, \ldots, N)$ while λ_i^j refers to the rate of packets dispatched to the j-th RX queue of port i, and hence handled by the j-th core $(j = 1, \ldots, C)$. It follows that $\Lambda_i = \sum_{j=1}^{C} \lambda_i^j$. Each RX queue has a finite capacity limiting to K the maximum number of packets being simultaneously queued in it. We use μ_i^j to denote the processing rate of the j-th core when it is serving the i-th RX queue. In other words, $1/\mu_i^j$ represents the average time the j-th core needs to process a packet from the i-th RX queue. Finally, we denote by $1/\beta$ the switch-over time taken by a core to switch from its current RX queue to the next one (β is thus the switch-over rate). Note that these latter quantities are directly measurable on a virtual switch.

III. Model

A. Decomposition principle

The first step of the model is to break down the general switch architecture into C independent subsystems, each of them consisting of one CPU core that polls N independent RX queues. Every subsystem is identified with a distinct color in Figure 1 and is simply referred to as a *polling system*. In the rest of the paper we only consider the model associated with a given CPU core j and its N related RX queues. Therefore, for the sake of clarity, we drop superscript j in all subsequent notations and equations. Figure 2 represents the polling system associated with the considered CPU core having a service rate μ_i when serving its i-th RX queue, and a switch-over rate β.

The idea is to subsequently replace each polling system with a set of N independent queueing models with server vacations. This decomposition step is illustrated in Figure 3. The buffer of queue i in the decomposed model represents the i-th RX queue associated with the considered CPU core. The server of the i-th queue in the decomposed model aims at reproducing the way packets of the i-th RX queue are processed by the CPU core. Because the core polls all its RX queues in-between the processing of two successive packets of a given queue i, there is an in-between time that corresponds to the processing of one packet at all the other non-empty queues and N switch-over times. In the model, this total time will be referred to as a *vacation time*. As an illustration, in Figure 3, the server of queue N is in process, meaning that the CPU core is currently processing a packet in RX queue N, and all other queues are in vacation. In this particular example, when queue N ends its processing, it goes in vacation, the first in-line packet of queue N is put on a hold, and at the same time the switch-over time

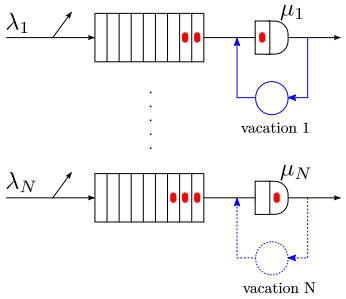


Fig. 3. Decomposition of a sub-system into N separate queues.

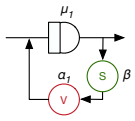


Fig. 4. Vacation representation.

between RX queue N and RX queue 1 starts. It is only after the completion of this switch-over time that queue 1 ends its vacation and starts the processing of its first in-line packet.

Instead of developing, as in Sohail model [3], the vacation time as the whole succession of switch-over and processing times, we keep in the vacation time the first switch-over time and aggregate all the remaining phases. The idea is illustrated in Figure 4. Following the processing stage of the server, the vacation starts by a switch-over time between RX queue 1 and RX queue 2. The remaining of the vacation time is then aggregated into a single phase with a given rate α_1 (i.e., with a given mean duration $1/\alpha_1$), that has to be accurately estimated.

B. Markov chain

In order to derive a tractable model, we make the following Markovian assumptions. First, we assume that the arrival of packets at the entrance of queue i follows a Poisson process of rate λ_i. Then, we assume that the processing time of one packet from queue i is exponentially distributed with rate μ_i. Finally, we assume that both phases of the vacation time of queue i are exponentially distributed with rates β and α_i, respectively.

Under these assumptions we can associate with each queue i of the decomposed model, the continuous-time Markov chain depicted in Figure 5. A state (k, P) of this chain, $k = 1, \ldots, K$, corresponds to queue i with currently k packets and the first-in-line packet being processed (P), i.e., the CPU core is assigned to RX queue i. A state (k, S) of this chain, $k = 0..., K$, corresponds to queue i with k packets, in which the CPU core is not anymore processing a packet but is switching (S) between RX queue i and RX queue $i+1$. Finally, a state (k, V) of this chain, $k = 0..., K$, also corresponds to queue i with k packets, but now the CPU core is either processing another RX queue or switching between the other RX queues.

From any state of this chain (except for the right ones corresponding to a full buffer) we can reach the state immediately on the right with some rate λ_i corresponding to the arrival of a new packet in queue i. We can exit a state (k, P), $k = 1, \ldots, K$, after a processing time of rate μ_i taking the chain to state $(k-1, S)$, and we can exit a state (k, S), $k = 0..., K$, after a switch-over time of rate β taking the chain to state (k, V).

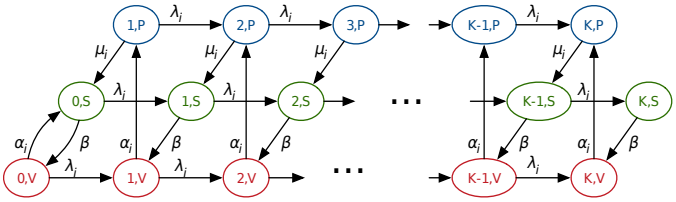


Fig. 5. Continuous-Time Markov Chain associated with queue i.

Now, as defined above, the exiting rate of any state (k, V) by the end of a vacation is α_i. Exiting a state (k, V) with $k > 0$ corresponds to the CPU core returning to RX queue i and then processing the next-in-line packet, i.e., coming back to state (k, P). Now, exiting state $(0, V)$ corresponds to skipping a turn for RX queue i. Indeed, although the core becomes available for RX queue i, this latter has no packet to be processed. This corresponds to the transition from $(0, V)$ to $(0, S)$.

In order to solve this chain corresponding to a particular queue i, only one parameter remains to be estimated, namely α_i (the other parameters λ_i, μ_i, β, and K are supposed to be known from measurements on the system). However, assuming that α_i is known, we can easily check that this Markov chain can be solved without resorting to any numerical technique. As a result, solving a Markov chain of our model is as easy (in terms of computational complexity) as solving a Markov chain of Artero et. al model [2].

C. Estimation of the chain parameters

Instead of considering α_i, we estimate $1/\alpha_i$, corresponding to the mean time between the end of switching from i-th to $(i+1)$-th RX queue (marking the time when the core is leaving queue i) and the end of switching from $(i-1)$-th to i-th RX queue (marking the time when the core is returning to queue i). Therefore, this time includes $N-1$ switch-over times, but also includes the processing of one packet for all non-empty RX queue j different from i. It follows that:

$$\frac{1}{\alpha_i} = (N-1) \times \frac{1}{\beta} + \sum_{j \neq i} (1 - \mathcal{E}_j) \times \frac{1}{\mu_j} \qquad (1)$$

In this expression, $\mathcal{E}_j$ represents the probability that RX queue j is empty when the core is returning to it, i.e., at the particular instant when the switch-over time from $(j-1)$-th to j-th RX queue ends. This parameters can be extracted from the Markov chain associated with RX queue j (equivalent to the one represented in Figure 5 but where i is replaced by j). Indeed, $\mathcal{E}_j$ can be expressed as the ratio between the number of transitions from state $(0, V)$ to state $(0, S)$ by unit of time, and the total number of transitions from red states by unit of time, each of them correspondonding to the end of a vacation for RX queue j. Therefore, we have:

$$\mathcal{E}_j = \frac{\pi_j(0, V)\alpha_j}{\sum_{k=0}^{K} \pi_j(k, V)\alpha_j} = \frac{\pi_j(0, V)}{\sum_{k=0}^{K} \pi_j(k, V)} \qquad (2)$$

where the π_j are the stationary probabilities of the j-th Markov chain.

Not surprisingly, the parameters of a Markov chain associated with a given queue i depend on the stationary solution of

the other Markov chains (through Eq. 1 and 2). As a result, the resolution of the model relies on a fixed-point iterative technique that is summarized by Algorithm 1. The main loop of the algorithm is repeated until a given convergence criterion is reached, e.g., the maximum relative difference of varying parameters between two successive iterations is very small.

Algorithm 1: Fixed-point iterative technique

```
Input    : System parameters K, μ_i, λ_i, β for each queue i
Initialize π_i, E_i for each queue i;
while convergence criterion not satisfied do
    foreach queue i ∈ [[1, N]] do
        Compute α_i using Eq. 1;
        Solve the Markov chain associated with queue i and
        compute the stationary probabilities π_i;
        Compute E_i using Eq. 2;
    end
end
Compute all performance metrics of interest;
```

D. Performance parameters

After convergence of our algorithm, we can derive the system performance parameters from the stationary probabilities of the Markov chains as follows. The average number of packets in queue i is given by:

$$\bar{q}_i = \sum_{k=1}^{K} k \times (\pi_i(k,P) + \pi_i(k,S) + \pi_i(k,V)), \qquad (3)$$

As for the loss rate at the entrance of queue i, we obtain:

$$b_i = \pi_i(K,P) + \pi_i(K,S) + \pi_i(K,V). \qquad (4)$$

The average sojourn time of an accepted packet in queue i is then obtained using Little's law [11]:

$$\bar{r}_i = \frac{\bar{q}_i}{\lambda_i(1-b_i)}. \qquad (5)$$

IV. Numerical Results

Throughout this section, we validate the analytical results with a home-made discrete-event simulator. Each simulation is run for 50 seconds of simulated time, which corresponds to the completion of millions of packets. Note that the corresponding confidence intervals are typically very small and hence not displayed in the subsequent figures. In addition to the simulator, we also include, where relevant, results from two existing models: Artero et al. [2] and Sohail [3].

For the sake of clarity, throughout this section we express the switch-over time as a fraction of the time needed by a CPU core to process a packet. For instance, for a core processing packets at rate $\mu = 1$ Mpps (corresponding to a mean processing time $1/\mu = 1$ µs), and a switch-over time $1/\beta = 0.1$ µs, the overhead of the switch-over time amounts to 10%.

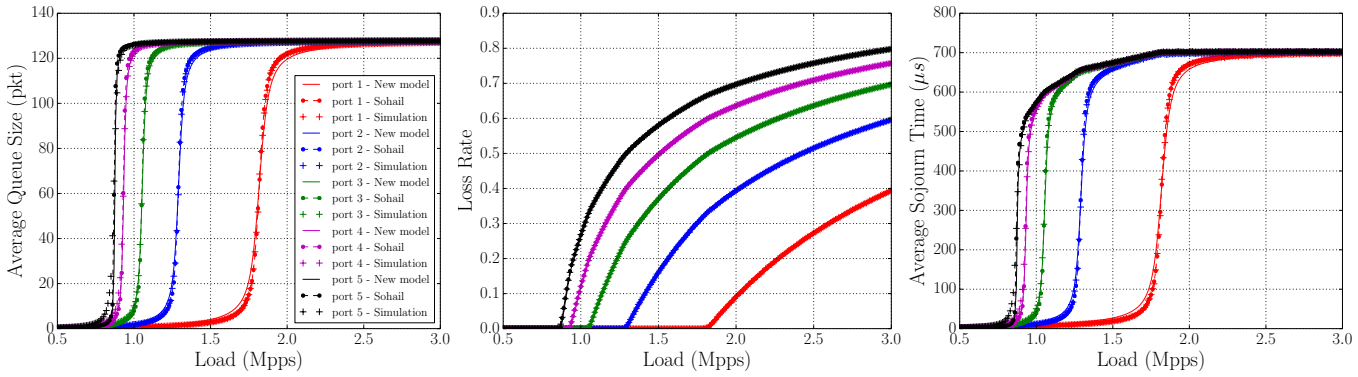
A. Model accuracy

This section opens with an assessment of accuracy for our new model under various settings of load and switch-over time. We consider a virtual switch with homogeneous CPU cores and a total of $N = 5$ ports. As discussed in Section III-A, we only focus on a given subsystem involving a single CPU core polling several RX-queues. The processing rate of the core when poling RX queue i is set to $\mu_i = 1$ Mpps with $i = 1, \ldots, 5$, while the overhead due to the switch-over time amounts to 10%. Each RX queue has a finite capacity of $K = 128$ packets. As for the load, we let the total packet arrival rate ($\Lambda = \sum_{i=1}^{N} \lambda_i$) vary from a low value of 0.5 Mpps to a high value of 3 Mpps. However, ports are unevenly loaded. Specifically, ports 1, 2, 3, 4 and 5 capture 10%, 15%, 20%, 25% and 30%, respectively, of the total load (i.e., $\lambda_1 = 0.1\Lambda$, $\lambda_2 = 0.15\Lambda$, $\lambda_3 = 0.2\Lambda$, $\lambda_4 = 0.25\Lambda$, and $\lambda_5 = 0.3\Lambda$).

Figure 6 shows the performance parameters associated with each port of the virtual switch as found by our new model, the simulation and Sohail model. Figure 6(a) depicts the average queue size (i.e., the number of packets being buffered in RX queues) as a function of the load. As expected, possible values range from 0 to 128, and the 5-th port (depicted in black) receiving the greater fraction of the load is the first to saturate when load increases. It is worth noting that curves are significantly steep revealing a high sensibility to the load. In Figure 6(b) we show the loss rate (i.e., the blocking probability) undergone by each port for increasing levels of load. We observe that losses start to appear for a total load near 0.9 Mpps (while the core is able to process packets at a speed of 1.0 Mpps). This gap is of course due to the non-zero value of the switch-over time. Finally, Figure 6(c) reports the evolution of the average sojourn time spent by a packet in its RX queue. Note that, under a heavy load, the average sojourn times of each port converge to a limiting value given by $((K-1)N+1)(1/\mu+1/\beta) = 699.6$ 1 µs. Overall, Figure 6 shows the good accuracy of both models (ours and Sohail) as both closely match values found by the simulation.

We now study more specifically the accuracy of our new model for various levels of the switch-over time. To this end, we focus on the behavior of a single port (the 3rd) from the previous example. We consider switch-over times ranging from a very low overhead representing 0.1% of the average packet processing time, to a heavy overhead of 10%.

The results are reported in Figure 7, which represents the average size of the corresponding RX queue as a function of the load for several switch-over time overheads. First, we observe that the switch-over time can have a huge impact on the queue behavior: its saturation occurring 10% sooner when the switch-over time is high. Second, when the switch-over time is as small as 0.1% of the processing time, its footprint is virtually null. Interestingly, under these circumstances, the results found by our new model precisely match those brought by Artero et al. model (which was designed to handle only examples with no switch-over time). Last but not least, Figure 7 also illustrates the good ability of our model to evaluate the average size of RX queues over a large and realistic spectrum of switch-over times as its values and those delivered by the simulator almost coincide. Note that we obtained similar accuracy for other performance metrics, not shown here for the sake of brevity.



(a) Average queue size. (b) Loss rate. (c) Average sojourn time.

Fig. 6. Accuracy on the behavior of RX queues for a virtual switch with $N = 5$ ports loaded respectively with 10%, 15%, 20%, 25% and 30% of the traffic.

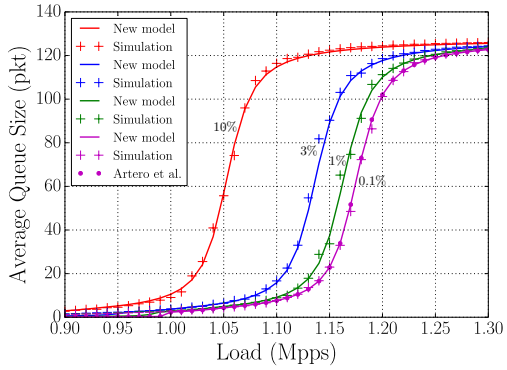


Fig. 7. Accuracy on the behavior of an RX queue for different overhead values of switch-over times, $1/\beta$.

B. Model complexity

We now examine the complexity of our new model. Its memory requirement is very small as the number of states in the Markov chains grows in $\mathcal{O}(KN)$ like in Artero et al. model but unlike Sohail model where it goes in $\mathcal{O}(KN^2)$.

To describe its computational complexity, we proceed in two steps: (i) we evaluate the number of iterations, and (ii) we evaluate the cost of each iteration. Unlike Sohail model, each iteration of our model involves a finite and known number of simple mathematical operations, which is in $\mathcal{O}(KN)$. However, as often when dealing with a fixed-point solution, the number of iterations cannot be analytically determined. Therefore, we report in Table I the number of iterations to convergence for a growing number of ports, N (using the example described in Section IV-A with a convergence criterion equal to 10^{-9}). This table shows that the number of iterations typically lies in the several tens. Furthermore, it grows only slightly with the number of ports and tends to stabilize around 50 when the number of ports exceeds 8.

Thus, in practice, our model is much faster to run than Sohail model. The difference is particularly marked when the number of ports is large.This is confirmed by Figure I which represents the execution time for both model as a function of the number of ports. Unlike Sohail model whose execution time can exceed several tens of seconds, our model, like Artero et al. model, is solved at a click-speed, regardless of the number of ports.

TABLE I
NUMERICAL BEHAVIOR OF OUR MODEL.

Number of of ports, N	2	4	6	8	12	16
Number of iterations of our model	30	43	48	50	52	53
Execution time (ms) of our model	26	88	165	230	342	467
Execution time (ms) of Sohail model	317	934	1,857	3,077	6,740	12,014

V. CONCLUSIONS

This paper deals with the modeling of a virtual switch architecture taking into account the switch-over times (amount of time needed for a CPU core to switch from one queue to another). We not only develop a new model, but we also propose a general framework that can be applied to estimate the performance of very general virtual switches. The contributions of the current paper are the following. First, we link together previous papers by showing that they fall within the scope of this general framework. Second, we use this framework to enhance the accuracy of previous models. Third, we propose a new model that accounts for switch-over times but that remains both computationally efficient and accurate. Future works aim at extending our model to the processing of batches of packets.

REFERENCES

[1] N. McKeown et al., "OpenFlow: enabling innovation in campus networks," *Computer Communication Review*, vol. 38, no. 2, 2008.
[2] G. A. Gallardo, B. Baynat, and T. Begin, "Performance modeling of virtual switching systems," in *IEEE MASCOTS*, 2016.
[3] A. Sohail, "Performance evaluation of a non-exhaustive polling system with asymmetrical finite queues," in *UKSim*, 2012.
[4] Open vSwitch - An Open Virtual Switch, http://www.openvswitch.org.
[5] OpenStack, https://www.openstack.org/, October 2016.
[6] Netmap, http://info.iet.unipi.it/ luigi/netmap, 2016.
[7] OpenOnload, http://www.openload.org, 2016, solarflare.
[8] PacketShader, http://shader.kaist.edu/packetshader/, February 2011.
[9] Data Plane Development Kit (DPDK), http://dpdk.org, 2016, Intel, 6WIND, etc.
[10] G. Pongrácz, L. Molnár, and Z. L. Kis, "Removing roadblocks from SDN: openflow software switch performance on intel DPDK," in *EWSDN*, 2013.
[11] A. O. Allen, *Probablity, Statistics and Queueing Theory with Computer Science Applications, Second Edition*. Elsevier, 1990.

ACKOWLEDGMENT

This work is partly funded by the French ANR REFLEXION under the "ANR-14-CE28-0019" project.