Towards including batch services in models for DPDK-based virtual switches

Zidong SU* Thomas BEGIN* Bruno BAYNAT†
zidong.su@ens-lyon.fr thomas.begin@ens-lyon.fr bruno.baynat@lip6.fr

*Université Lyon 1, ENS Lyon, Inria, CNRS, UMR 5668 - Lyon, France
†Sorbonne Université UPMC Univ Paris 06, CNRS, LIP6 UMR 7606 - Paris, France

***Abstract*—With the development of NFV (Network Function Virtualization) paradigm, networking functions would gradually move from specialized and proprietary hardware to open-source software run over a virtual machine (VM) deployed on commodity hardware. Open vSwitch (OVS) is the most prominent open-source solution implementing a virtual switch, while DPDK (Data Plane Development Kit) is a set of specialized libraries to enhance the performance. In particular, DPDK allows packets to be processed by batches. In this paper, we address the issue of modeling the behavior of a DPDK-based virtual switch. First, we investigate the influence of batch services on the overall performance of a virtual switch. Second, we extend two former analytical models to include the processing of packets by batches. We propose a simple means to do it, and we evaluate the accuracy of our solution on two different scenarios. Numerical results show that, despite its simplicity, our approach provides fairly good results when compared to simulation.**

***Keywords*—NFV; virtual switch; DPDK; performance modeling ; batch services; approximation.**

I. Introduction and related work

Since the early 2000s (with the release of the first x86 virtualization product by VMware in 1999), an ever-growing fraction of application servers are getting virtualized. In a nutshell, with the help of a hypervisor, a single physical server is divided into multiple isolated VMs (Virtual Machines), each behaving similarly to a regular server from an end-user point of view. Virtualizing servers enables both a better use (in terms of energy and costs) and an easier management of the physical resources.

More recently, the networking community is pushing for virtualizing the network itself. In this regard was proposed the NFV (Network Function Virtualization) paradigm [1] wherein networking functions would gradually move from specialized and proprietary hardware to open-source software run over a VM deployed on commodity hardware. These functions are then referred to as VNFs (Virtualized Network Functions). Examples of VNFs include a simple forwarding, complex routing as well as more advanced operations such as encrypting, NAT (Network Address Translation), etc. Analogously to virtualization of servers, potential benefits of NFV include cost-saving (by hosting several VNFs on a single physical machine) and ease of management (with a better flexibility in the resource provisioning).

Aside from its advantages, the NFV paradigm raises some concerns, two of them being the security and the performance. In this paper, we deal with the latter in the case of the arguably foremost VNF, namely the packet switching between ports of a VM. Such a VNF is typically referred to as a virtual switch (or bridge) and aims at reproducing the behavior of a "classical" switch or a router. As for its implementation, Open vSwitch (OVS) [2], [3] is the most prominent open-source solution implementing a virtual switch while proprietary solutions such as Cisco Nexus 1000V and VMWARE vSphere Distributed Switch have also been released.

In the aim of accelerating the performance of a virtual switch, a couple of solutions (e.g., Netmap [4], [5], OpenOnloadn [6], PacketShader [7], DPDK [8], [9]) has been proposed. In particular, DPDK (Data Plane Development Kit) is a set of specialized libraries and drivers primarily developed by Intel and 6WIND. DPDK involves four main strategies to speed up virtual switches. First, the packet processing is moved outside the OS kernel and into the user space. By bypassing the kernel, the datapath is no more interrupted by system calls. Second, DPDK avoids packets copying, and instead, it only switches memory pointers referring to packets content. Third, DPDK dispatches efficiently the load of incoming packets among its CPU cores. Fourth, packets can be processed by batches. We discuss in more details the last two points in the next section. While the first three strategies have been taken into account in existing performance models [10]–[12], to the best of our knowledge, the fourth point has not been tackled so far.

The contributions of this paper are twofold: (i) we investigate the influence of batch services on the overall performance of a virtual switch and (ii) we propose an easy yet accurate way of taking into account batches in the packet processing in two existing models [11] and [12]. Note that in [11] we came up with a first modeling framework that does not account for packet batches, nor switch-over times between CPUs. In our subsequent work [12], we extended the framework to handle the case of non-negligible switch-over times, but the infeasibility of packet batches remains.

The remainder of this paper is organized as follows. Section II describes the internal architecture of a DPDK-based virtual switch. In Section III we review the principles of two existing analytical models that assume packet processing without batch

service. We describe how we extend these models to take into account batch services in Section IV. Numerical results that show the accuracy of the approach are given in Section V. Section VI concludes this paper.

II. System architecture and notation

A. DPDK operations

This sub-section provides a description of the general architecture of a virtual switch using the DPDK library set in its "standard configuration". Basically, a VM running a virtual switch implementation can access a set of physical resources consisting of several CPU cores, I/O ports, and RAM. Upon their arrival at a port, incoming packets are immediately dispatched in separate queues, called RX queues. This first step is typically carried out through the application of a hash function on the packet headers and aims at ensuring an even balance of the incoming packets among the RX queues. Each CPU core is assigned to a single RX queue of each port so that any core handles just as many RX queues as the total number of ports in the virtual switch. This mapping is illustrated by Figure 1. CPU cores process their associated RX queues in a cyclic manner. Each round starts with a core polling an RX queue. It processes the M-first-in-line packets, if any, before moving to the following RX queue and so on. Note that a core may not immediately switch from the current RX queue to the other. We refer to this (possibly non-negligible) delay as a switch-over time. Note also that a core only processes packets that are already waiting in the RX queue at the time it starts serving the queue. Subsequent arrivals must wait until the next core visit. This policy is usually known as "gated M-limited" policy [13]. To complete the processing of a packet, a core needs at least to read (and possibly edit) its headers, extract the destination address, and find out to which output port the packet must be forwarded, as well as possibly additional operations such as deep packet inspection, encryption, and QoS monitoring. Note that the length of this step (processing a packet) varies significantly depending on the state of the cache of the corresponding CPU core. Indeed, if the instructions needed to process a packet are found in the cache (cache hit), the processing of a packet becomes much faster, say around an order of magnitude or so, (compared to the case of a cache miss). Afterward, the packet is (logically) forwarded from its RX queue to the TX queue associated to the appropriate output port. At that stage, the packet only has to wait for its transmission on the next link.

B. Notation

We now introduce the 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

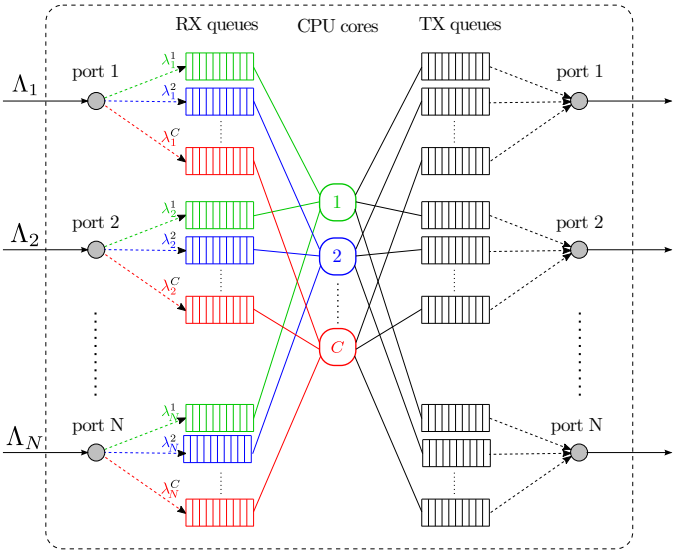


Fig. 1. Internal architecture of a virtual switch with N ports and C cores.

queued in it. We use M to refer to the (maximal) size of a batch. We denote by T_H and T_M the time needed by a CPU core to process a packet when the set of instructions is found (cache hit), respectively not found (cache miss), in the cache. T_R indicates the time needed by the CPU core to forward a packet from an RX queue to an TX queue over a PCI bus (once the CPU core processing has ended). As a result, the total time needed by a core to serve a given packet is either $T_M + T_R$ (in case of a cache miss) or $T_H + T_R$ (in case of a cache hit). Broadly speaking, the first case is more likely to occur if the considered packet is among the first packets of a batch (the cache is "cold") whereas the second case has more chances to happen for the last packets of a batch benefiting from the cache information. Finally, we denote by T_S the mean switch-over time taken by a core to switch from its current RX queue to the next one.

III. Models without batch service

We start this section by reviewing two of our previous models, [11] and [12] that work without batch service. As a result they correspond to a value of $M = 1$. We show that these two models can be presented in a very general framework that aims at decomposing the polling system into independent queueing models with server vacations.

In these two modeling papers, a CPU core is supposed to process only one packet at each round for each of its associated RX queues. The system therefore does not really benefit from the cache. As a result the mean processing time of the j-th CPU core when serving a packet from its i-th RX queue can be considered as a constant given by:

$$\frac{1}{\mu_i^j} = T_M + T_R. \tag{1}$$

The switch-over rate β is simply defined in [12] as the inverse of the mean switch-over time:

$$\beta = \frac{1}{T_S}. \tag{2}$$

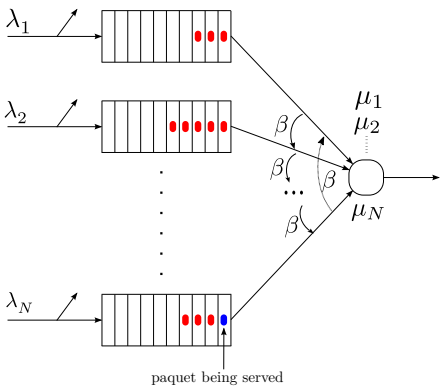


Fig. 2. Subsystem involving a single CPU core polling several RX-queues.

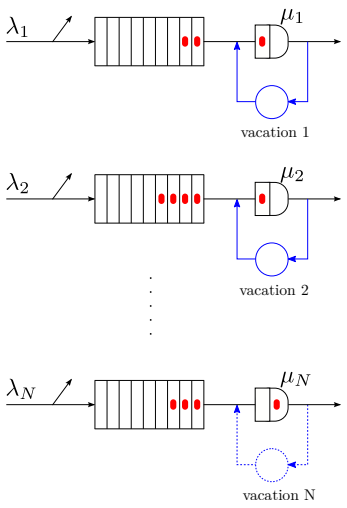


Fig. 3. Decomposition of a sub-system into N separate queues.

A. General modeling framework

1) Decomposition principle: The first step of these two models 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 models 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 general idea of the two models is to subsequently replace each polling system with a set of N decoupled 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 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.

2) Modeling assumptions: In order to derive tractable models, the following Markovian assumptions are made. First, authors assume that the arrival of packets at the entrance of queue i follows a Poisson process of rate λ_i. Then, they assume that the processing time of one packet from queue i is exponentially distributed with rate μ_i. They also assume that the switching time is exponentially distributed with rate β. Finally, as developed in the following sections, the subsequent models will represent the vacation times by a succession of different phases with exponential distributions.

B. Model for a negligible switch-over time: Model 1

In [11] it is assumed that the switch-over time is negligible, which corresponds to a null value of T_S in our general framework.

1) Vacation representation: Because the switch-over time is zero, when, e.g., queue 1 ends its processing, and if, at that very instant, all other RX queues are empty, it skips vacation and immediately starts the processing of the next in-line packet (if any). This possibility is illustrated in Figure 4 that represents the vacation of server 1, by a direct transition looping on the processing stage with some probability f_1. With a probability $1 - f_1$, when queue 1 ends its processing, it thus goes in vacation and stays unavailable for a time that is supposed to be exponentially distributed with rate α_1 (corresponding to the red "V" labeled circle of Figure 4).

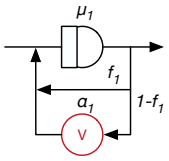


Fig. 4. Vacation representation of Model 1.

2) Markov chain model associated with each RX queue: Under the markovian assumptions given in Section III-A2, we can associate with each queue i of the decomposed model, the continuous-time Markov chain depicted in Figure 5 [11]. A state (k, P) of this chain, $k = 1..., K$, 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

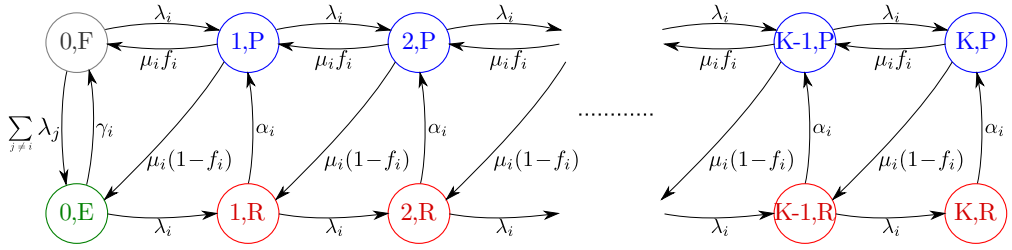


Fig. 5. Model 1: Continuous-Time Markov Chain associated with queue i.

i. A state (k, R) of this chain, $k = 1..., K$, also corresponds to queue i with k packets, but with the first-in-line packet that is not currently in process and that is ready (R) to be processed, i.e., the CPU core is assigned to another RX queue. State $(0, E)$ corresponds to an empty (E) queue i but not a full empty system, i.e., the CPU core is currently assigned to another non empty RX queue. Finally state $(0, F)$ corresponds to a full (F) empty system, meaning a system where all queues are empty and the CPU core is idle.

Three parameters are left to be estimated in order to fully characterize the Markov chain associated with queue i, namely the probability f_i and rates α_i and γ_i. Because these parameters depend at the same time on the stationary solution of the other Markov chains and on the performance parameters given in Section III-B3, and conversely, the performance parameters depend on the stationary solution of all Markov chains, the model relies on a fixed-point iterative technique, as developed in [11].

3) Performance parameters: Once all Markov chains (associated with all queues $i = 1, ..., N$) have been solved in stationary regime, we can derive the steady-state performance parameters of the system as follows. First, we can easily obtain the average number of packets in queue i from the stationary probabilities of the chain:

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

as well as the loss rate at the entrance of queue i:

$$b_i = \pi_i(K, P) + \pi_i(K, R). \tag{4}$$

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

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

C. Model taking into account the switch-over time: Model 2

The model presented in [12] is an extension of the one in [11] that takes into account the switch-over times, and thus considers non-negligible values of $T_S = 1/\beta$.

1) Vacation representation: Unlike the model developed in Section III-B, this new model includes the possibility of a switch-over time. As a result, the vacation time of a given queue can never be zero, as there are at least N switch-over times in-between two successive packet processing of a same queue. Instead of developing the vacation time as the whole succession of switch-over and processing times (as in [10]), [12] keeps in the vacation time the first switch-over time and aggregate all the remaining phases. The idea is illustrated in Figure 6 that represents the vacation of server 1. 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 exponential phase with a given rate α_1 (i.e., with a given mean duration $1/\alpha_1$), that has to be accurately estimated.

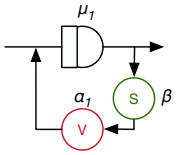


Fig. 6. Vacation representation of Model 2.

2) Markov chain model associated with each RX queue: Under the markovian assumptions given in Section III-A2, we can associate with each queue i of the decomposed model, the continuous-time Markov chain depicted in Figure 7. 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.

In order to solve this chain, only one parameter remains to be estimated, namely α_i. Not surprisingly, parameter α_i of a given Markov chain depends on the stationary solution of the other Markov chains. As a result, the resolution of the model relies on a fixed-point iterative technique (see [12] for details).

3) Performance parameters: From the stationary probabilities of the Markov chains we can derive the average number of packets in queue i:

$$\bar{q}_i = \sum_{k=1}^{K} k \times (\pi_i(k, P) + \pi_i(k, S) + \pi_i(k, V)). \tag{6}$$

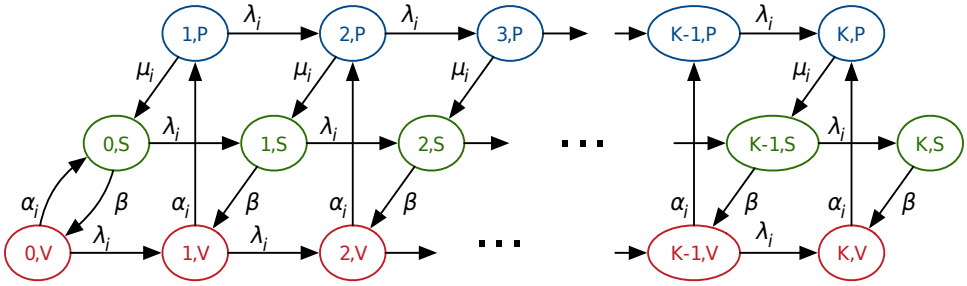


Fig. 7. Model 2: Continuous-Time Markov Chain associated with queue i.

as well as the loss rate at the entrance of queue i:

$$b_i = \pi_i(K, P) + \pi_i(K, S) + \pi_i(K, V). \quad (7)$$

The average sojourn time of an accepted packet in queue i is then obtained using relation 5.

IV. Taking into account batch service

A. General insights

The main objective of this paper is to develop new models that take into account the influence of batch size on the performance of a virtual switch. As stated before, the fact that packets are processed by batch has two consequences. (i) First, the structural behavior of the system is peculiar. Indeed, when compared to a system where each round of a CPU core consists in serving (at most) one packet from each of its RX queues, a system in which the CPU core processes batches of packets implies more variability in the time between the processing of two consecutive packets of the same RX queue. This time is very short if the two packets belong to the same batch and is much longer if they belong to two different batches separated by a round. This increased variability necessarily has a consequence on the performance of the system. (ii) Second, when the batch size increases, the influence of the cache also increases. Indeed, as stated in Section II-A, the larger the batch size, the more likely the CPU instructions needed for a given packet of a batch can be found in the cache, and the lower the average processing time of a packet. The batch size has thus also a consequence on the parametrization of the models, and more specifically on the value of the mean processing rates μ_i.

B. Influence of batch service

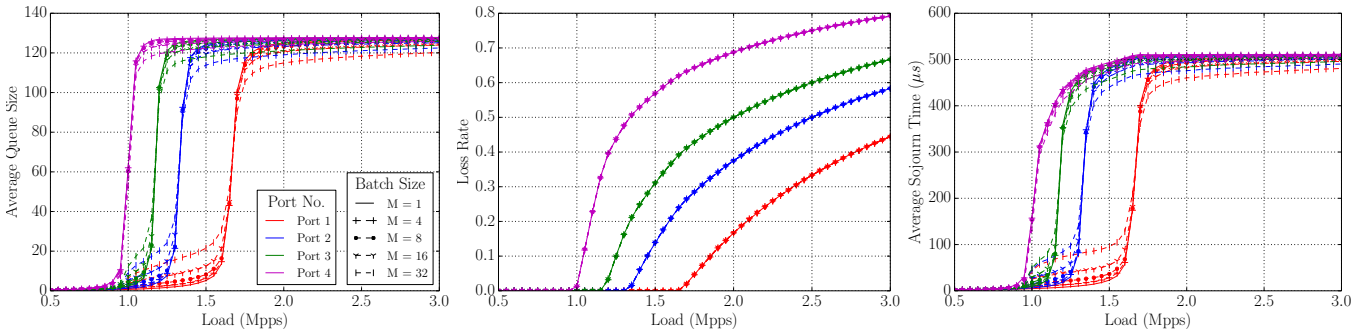
In order to evaluate the influence of both phenomenons separately, we first consider a system where the cache has no effect and we concentrate on the influence of batch service. The easiest way to realize such a system it is to take $T_M = T_H$. As a consequence the average processing time is the same for all packets and corresponds to the value given by relation 1 in the case without batches ($M = 1$), i.e., $T_M + T_R$.

Let us consider an example (extracted from [11]) of a virtual switch comprising $N = 4$ input ports served by a set of homogeneous CPU cores whose service rates are constant and set to $\mu_i = 1$ Mpps (million packets per second) each. We assume that the dispatching function is well-behaved so that every CPU core undergoes the same performance. Hence, we can restrict our analysis to only one of them. The global input rate of packets bounded to the selected CPU core, $\lambda_{in} = \sum_{i=1}^{4} \lambda_i$, is supposed to be unevenly spread among the 4 ports as follows: 15% on port 1, 20% on port 2, 25% on port 3 and 40% on port 4. We let λ_{in} vary from 0.5 Mpps to 3 Mpps in order to explore the whole spectrum of possible loads. For each level of the load λ_{in}, we evaluate by simulation the average queue size $\bar{q}_i$ (Figure 8(a)), the loss rate b_i (Figure 8(b)), and the mean sojourn time of an accepted packet $\bar{r}_i$ (Figure 8(c)), associated with each individual port $i = 1, \ldots, 4$, and for different values of the batch size $M = 1, 4, 8, 16$ and 32. The magenta curves pertain to the performance of port 4, which is the first to saturate since it captures the largest fraction of incoming packets. On the other hand, the red curves are associated to port 1 which is the last to saturate.

Figure 8(b) clearly shows that the loss rate of packets at the entrance of each RX queue is totally insensitive to the batch size. Indeed, for each port, all curves corresponding to the different values of M are superimposed. This is a quite surprising and important result. Indeed, the loss rate is arguably one of the most, if not the most important performance parameter for networking software engineers. If it is the only parameter of interest, there is seemingly no need for developing models that structurally take into account batches. On the other hand, if we look at the average queue size shown in Figure 8(a) or equivalently at the average sojourn time of a packet in Figure 8(c), we can see that the batch size has a non-negligible effect on performance. However, this effect is much more important on red curves corresponding to port 1 that saturates last, than on purple curves corresponding to port 4 that saturates first, meaning in zones of loads that do not correspond to normal operational mode (where the system is almost totally saturated).

As a conclusion of this example (verified by numerous other examples), we believe it is sound to rely on models that do not structurally take into account the batch size, in order to provide pretty accurate performance parameters, at least in load zones corresponding to standard operational modes.

This is exactly what we propose to do in this paper: using models [11] and [12] (presented in Section III) that do not take into account the batch size, but adjusting carefully their input parameters in order to reflect the influence of the cache (that would not exist in the case of no batch service).



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

Fig. 8. Simulated performance for different batch sizes.

C. Influence of cache

In order to understand the influence of the cache on the mean processing times, let us take an example of a virtual switch having $N = 4$ ports and processing packets by batch of size $M = 16$. Let us consider that the processor at a particular round has to process a batch of 16 packets from RX queue 1 that is made of 4 packets destined to port 2, 7 packets destined to port 3, and 5 packets destined to port 4. We can reasonably assume that for the first packet destined to a given port, the information necessary for its processing (typically contained in the forwarding table) has to be fetched from the RAM, whereas for the subsequent packets destined to the same port, the information is present in the cache. If we make this assumption, the total time necessary to process all packets of the considered batch is thus $3T_M + 13T_H + 16T_R$, and the average processing time by packet is $\frac{3T_M+13T_H+16T_R}{16}$.

By generalizing this result, we propose to estimate the average processing time by the following simple relation:

$$\frac{1}{\mu_i} = \begin{cases} \frac{N-1}{M}T_M + \frac{M-(N-1)}{M}T_H + T_R & \text{if } M \geq N-1 \\ T_M + T_R & \text{if } M < N-1 \end{cases} \quad (8)$$

This estimation is realistic when $M >> N$ and when the system is heavily loaded. Indeed in this case there is a high chance that the CPU core processes full batches, i.e., batches made of M packets, and that in each batch there is at least one packet destined to each of the possible $N-1$ output ports. In this case, the processing time of the whole batch is $(N-1)T_M + (M-(N-1))T_H + MT_R$ and the formula becomes exact. When M is small or when the load is low, relation 8 is not accurate anymore, but we will see in the next section that this inaccuracy has a negligible impact on the overall performance of the virtual switch.

Finally, when considering a virtual switch with non negligible values of the switch-over time T_S, we have to adapt the value of β since the model that does not structurally take batch services into account. The easiest way is to reduce the mean switch-over time by a factor M, leading to replace relation 2 by:

$$\beta = \frac{M}{T_S}. \quad (9)$$

V. Numerical Results

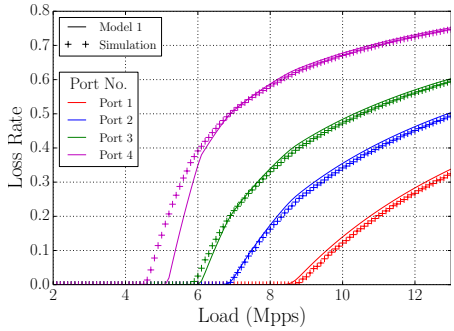
We evaluate the accuracy of our solution for including batch services in existing models by comparing their outcome with that from a home-made discrete-event simulator that fully complies with the behavior of DPDK (i.e., packets are processed by batches of size M according to a gated M-limited policy as described in Section II-A). Each simulation run lasts for 5 seconds corresponding to the completion of several millions of packets.

We now state assumptions made for both subsequent scenarios. For the sake of simplicity we assume that the hash function used upon the packet arrivals to select an RX queue and hence a CPU core is evenly-balanced. We also suppose that incoming packets are equally likely to be destined to any of the $N-1$ output ports (we dismiss the port on which the packet arrived). Furthermore, as discussed in Section IV-C, we assume that, within any batch, the first packet being destined to a given port experiences a cache miss and thus a delay of T_M while subsequent packets destined to the same port benefit from the cache and hence undergo a delay of T_H. Either way, there is the additional delay T_R to forward the packet from an RX queue to the chosen TX queue.

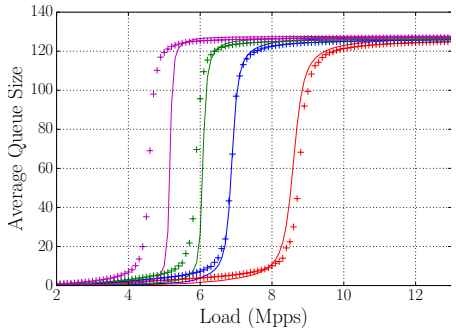
A. Scenario A

In our first scenario, we consider a virtual switch with $N = 4$ ports served by a set of homogeneous CPU cores. Because we assume that the dispatching function is well-behaved, every CPU core undergoes the same performance and we can restrict our analysis to only one of them. Nonetheless, for each CPU core, the input rate of packets on each of its RX queues (associated with ports), λ_{in}, is unbalanced and spread as follows: 15% on RX queue 1, 20% on RX queue 2, 25% on RX queue 3 and 40% on RX queue 4. The capacities of every RX queues is set to $K = 128$ packets. Packets are processed by batches whose maximum size is set to $M = 8$. Each packet is processed by a CPU core in $T_H = 30$ ns in case of a cache hit, and in $T_M = 300$ ns otherwise (cache miss) before a delivery delay to the corresponding TX-queue of $T_R = 62.2$ ns. Finally we assume that there is no switch-over time so that $T_S = 0$.

Because there is no additional delay for a CPU core to switch from one RX queue to another, we rely on Model 1 (see Section III-B) to analytically evaluate the performance of such a system. Figure 9 shows the corresponding results found

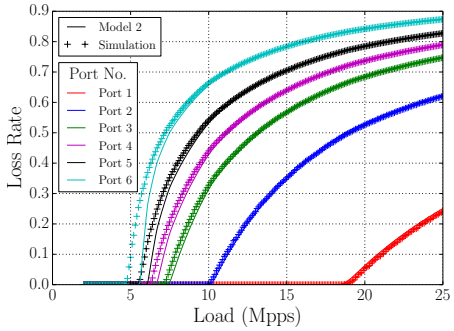


(a) Loss rate.

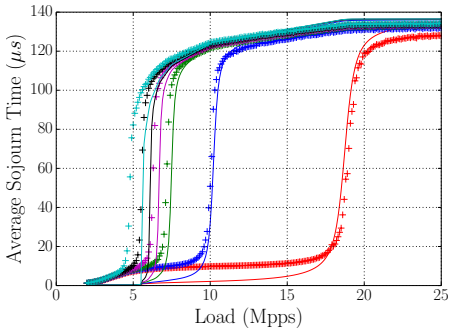


(b) Average queue size.

Fig. 9. Accuracy of our proposed modeling of batch services in case of no switch-over time - Scenario A.



(a) Loss rate.



(b) Average sojourn time.

Fig. 10. Accuracy of our proposed modeling of batch services in case of a switch-over time - Scenario B.

for two key performance parameters as a function of the global input rate λ_{in} ranging from a low 0.5 Mpps to a high 3.0 Mpps. In Figure 9(a), we observe that the loss rate obtained by our model closely matches that delivered by the simulation on each of the four ports except for port 4 for a moderate load around 5 Mpps. We believe that this deviation occurs because relation 8 was essentially thought for high levels of load (resulting in full batches) and might lead to inaccuracies at moderate levels of load. Figure 9(b) illustrates the accuracy of our model when dealing with the average number of packets waiting to be processed in each RX queue. Despite a small decay of precision at RX queue 4 for a load near 5 Mpps, our model captures the general behavior as given by the simulation.

B. Scenario B

In our second example, we extend the number of ports to $N = 6$, we enlarge the batch size to $M = 16$, and we introduce a switch-over time equal to $T_S = 20$ ns (representing nearly 7% of the packet processing delay in case of a cache miss). Note that the input rates on RX queues are set to (in ascending order) 5%, 10%, 15%, 18%, 22% and 30%. All remaining parameters keep the same values as in Scenario A.

The non-negligible value of T_S implies the use of our Model 2 (see Section III-C). The corresponding numerical results are given by Figure 10. First, we compare the loss rate obtained by our model with that delivered by the simulation. Figure 10(a) shows that our model is able to accurately forecast the level of losses experienced by every RX queues of the switch, again with a small discrepancy occurring on the most loaded RX queue around 5 Mpps. In Figure 10(b) we focus on the average sojourn time. We observe a good match between the values found by our model and those given by the simulation.

Overall, these experiments as well as many others (not shown in this paper) show that, despite its simplicity, our proposed modeling approach to take into account batches in the packet processing of a DPDK-based virtual switch provides a relatively good solution.

VI. Conclusions

In this paper, we address the issue of modeling the behavior of a DPDK-based virtual switch. To this end, we extend two former analytical models to include the processing of packets by batches, which is an important feature of the DPDK library. We propose a simple means to do it, and we evaluate the accuracy of our solution on two different scenarios. Numerical results show that, despite its simplicity, our approach provides fairly good results when compared to simulation. Future works aim at improving our solution, as well as developing more realistic scenarios wherein the model may be used, e.g., to discover "optimal" values of the batch size for a given configuration of the virtual switch (i.e., physical resources and load).

References

[1] M. Chiosi, D. Clarke, P. Willis, A. Reid, J. Feger, M. Bugenhagen, W. Khan, M. Fargano, C. Cui, H. Deng *et al.*, "Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action," in *SDN and OpenFlow World Congress*, 2012, pp. 22–24.

[2] Open vSwitch - An Open Virtual Switch, http://www.openvswitch.org, 2016.

[3] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The Design and Implementation of Open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, 2015.

[4] Netmap, http://info.iet.unipi.it/ luigi/netmap, 2016.
[5] L. Rizzo, "netmap: A novel framework for fast packet I/O," in *2012 USENIX Annual Technical Conference, June 13-15, 2012*, 2012, pp. 101–112.
[6] OpenOnload, http://www.openload.org, 2016, solarflare.
[7] PacketShader, http://shader.kaist.edu/packetshader/, February 2011.
[8] Data Plane Development Kit (DPDK), http://dpdk.org, 2016, Intel, 6WIND, etc.
[9] G. Pongrácz, L. Molnár, and Z. L. Kis, "Removing roadblocks from SDN: openflow software switch performance on intel DPDK," in *2nd European Workshop on Software Defined Networks, EWSDN*, 2013.
[10] A. Sohail, "Performance evaluation of a non-exhaustive polling system with asymmetrical finite queues," in *14th International Conference on Computer Modelling and Simulation, UKSim*, 2012.
[11] G. A. Gallardo, B. Baynat, and T. Begin, "Performance modeling of virtual switching systems," in *IEEE International Conference on Modeling, Analysis, and Simulation on Computer and Telecommunication Systems, MASCOTS*, 2016.
[12] Z. Su, B. Baynat, and T. Begin, "A new model for dpdk-based virtual switches," in *IEEE Conference on Network Softwarization (IEEE NetSoft 2017)*, 2017.
[13] H. Levy and M. Sidi, "Polling systems: applications, modeling, and optimization," *IEEE Transactions on communications*, vol. 38, no. 10, 1990.

ACKOWLEDGMENT

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