

A Virtual Switch Architecture for Hosting Virtual Networks on the Internet

Fabienne Anhalt, Dinil Mon Divakaran, and Pascale Vicat-Blanc Primet

INRIA – Université de Lyon

{fabienne.anhalt, dinil.mon.divakaran, pascale.primet}@ens-lyon.fr

Abstract—The future Internet is envisioned to host a large number of virtual networks managed by different operators sharing the same physical infrastructure. In such a scenario, an operator may not even own physical resources as such, but lease virtual resources to have their own virtual networks. While control-plane virtualization with several routing instances becomes common in equipments, the data plane is generally shared relying on logical isolation or resource segmentation, lacking in efficient sharing with performance guarantees. This makes it necessary to look into layer 2, and virtualize the switching fabric to have control over the sharing of the most critical resources for packet switching. In this article, we come up with a flexible architectural design for virtualizing a switching fabric. This new architecture enables a multitude of choices to customize virtual switches as needed. Our simulations show the relative loss in performance brought by virtualization, and demonstrate how the offered flexibility can help in exploiting the resources in a new way, satisfying independent virtual switch requirements.

I. INTRODUCTION

Network virtualization is expected to be a potential solution to *de-ossify* the Internet. It gives operators a powerful substrate for managing their ‘virtual’ networks, independent of other co-existing virtual networks. In the light of network virtualization, the Internet can be seen as a huge physical infrastructure hosting multiple virtual networks owned by different network operators. Network virtualization can open a new market of selling virtual-network equipments on demand. These are pre-installed, connected, and deployed for a given period of time, and can be dynamically adjusted in switching resources and set to a geographic location.

While cloud computing and virtual infrastructures expose computing resources to users as a service in the production network [1], [2], virtual networks remain part of experimental facilities. They are used for proof-of-concept and functionality. The next goal is to expose them, too, as a service on the Internet. Different operators might manage different virtual networks, all hosted on the Internet, but sharing the same physical infrastructure [3]. Besides bringing savings in equipment costs, splitting the network into isolated virtual networks introduces flexibility in hosting easily configurable virtual networks on a common infrastructure that can be optimized independently by operators to maximize network utility. From another perspective, users can choose to connect to one or more virtual networks depending on which utility they would like to maximize. The network itself would become a service. However, to make this possible, virtual networks need to be

implemented and managed on the Internet. Facilities to manage the coordinated hosting of virtual networks by different providers, such as HIPerNet [4], are needed at a software level. At the hardware level, the physical infrastructure has to be upgraded in order to enable efficient virtualization and isolation in all the participating nodes. Each virtual network performs individual routing inside ‘virtual routers’ while they share a common physical infrastructure. Independently of each other, virtual routers need to behave like physical routers, having the same functionality and giving deterministic link capacities and performance and need strong isolation in the data plane.

This brings focus to layer 2, with the necessity of virtualized switching in addition to virtualized routing, as one of the building blocks of the virtual networks, towards realizing a fully virtualized Internet. Today, the control plane of physical routers can be virtualized [5], but the data plane is either logically isolated, giving no control over the physical-resource partitioning, or segmented, limiting the multiplexing property [6]. Software router virtualization offers more flexibility, but the lack of performance [7] makes it inappropriate for the production Internet.

This paper proposes a new switch architecture designed to be used as a virtualized data plane in virtual routers or switches. Compared to previously cited solution, it virtualizes the important resources of the switching plane — buffer and capacity. Moreover, this architecture allows customized packet scheduling per virtual switch for flexible and individual configuration.

The rest of this paper is organized as follows. Section II describes the requirements and architecture of a virtual switch (VS), while section III describes resource-sharing strategies. A simulation-based evaluation of this architecture is presented in section IV. Section V describes the related work, before concluding with future perspectives in section VI.

II. VIRTUAL SWITCH DESIGN

This section describes the design of a physical switch, virtualized to host a number of customizable virtual switches. It describes the architecture and the associated resource-sharing strategies to isolate the hosted virtual switches from each other, in performance and configuration.

A. Requirements of a virtual switch

To fit the requirements of the Internet, a virtual switch (VS) has to be functionally identical to a physical switch and give

deterministic performance, appearing as a dedicated physical switch to the operator. This means that it should be given a guaranteed share of the physical resources. Therefore, the resource virtualization of layer 2 is required. The operators should also be given maximum flexibility in setting the specifications of virtual switches.

In general, an operator can request a virtual switch k with specification $S^k = \{N^k, \mathbf{B}^k, \mathbf{C}^k, SP^k\}$, where:

- N^k is the number of ports of the VS,
- $\mathbf{B}^k = [b_{i,j}^k], i = 1 \dots N^k, j = 1 \dots N^k$ with $b_{i,j}^k$ specifying the buffer size on virtual forwarding path $i - j$,
- $\mathbf{C}^k = [c_i^k], i = 1 \dots N^k$ where c_i^k is the capacity of the i^{th} virtual link of VS^k , the k^{th} VS.
- SP^k specifies the scheduling policy inside VS^k

Further details such as, e.g., burst rate, average rate, priority, latency, can be added to the specification, but are omitted here in the first design. In addition to these VS-specific parameters, the physical switch has policy parameters on the resource sharing between VSes, such as the inter-VS scheduling policy and the inter-VS buffer-sharing policy. There are many combinations of VS specifications that are possible.

While combining different VSes on the same physical switch, the allocation of any VS^k has to satisfy the following conditions:

- $c_i^k \leq c_i - \sum_{x=1, x \neq k}^V c_i^x, \forall i \in [1, N]$
- $b_{i,j}^k \leq b_{i,j} - \sum_{x=1, x \neq k}^V b_{i,j}^x, \forall i, \forall j \in [1, N]$

These constraints guarantee that each VS sharing the physical router obtains the required amount of resources and predictable performance.

The design of the virtualized switch is the result of the implementation of different techniques to satisfy a given set of specifications on a physical-switch architecture, with the help of resource partitioning and scheduling principles. The first step here is to choose an appropriate switch architecture.

B. Architecture choice

Two commonly-used switch architectures are shared-memory and crossbar. The shared-memory is known to have scalability issues challenged by the need to access the memory at a speed equivalent to the product of the line rate and the number of ports [8]. Overcoming these inhibitions, the crossbar architecture permits N pairs of I/O ports to communicate, with memories running at speeds independent of the number of ports.

The traditional crossbar switches deploy a centralized, complex matching algorithm to decide which among the contesting ports should communicate. All the selected ports transfer fixed-size packets at the same time, in a synchronous manner. This also requires segmentation of variable-size packets into fixed-size ‘cells’ at the input, and complementary reassembly at the output. These constraints are removed with the introduction of buffers at the crosspoints (CP), and leave the choice to perform cell- or packet-scheduling. The schedulers are now

distributed at each output point, and can operate independently to switch packets of variable sizes asynchronously.

Having distributed schedulers is an important reason for choosing this architecture for virtualization. The complexity of scheduling algorithms in centralized crossbar architecture (with no buffering at crosspoints) is usually greater than $O(N^2)$ [9]. Virtualization of such an architecture brings up scalability issues as the number of virtual ports increases. On the other hand, the distributed schedulers in the Crosspoint-Queued (CQ) switch architecture pose no such problem.

Exploiting this technology, a CQ switch [10] has queues neither at the inputs nor at the outputs, but only at the crosspoints. Besides, the simple architecture removes the need to virtualize input/output queues. For more details on CQ switches, we refer the readers to [10]. Fig. 1(a) shows a CQ switch. Each crosspoint of the switching fabric implements a queue. Incoming packets are sent to the crosspoints according to their destination output port. A scheduler runs at each output port, selecting packets from one of the different crosspoints connected to that output. Once a packet is scheduled, it is sent out through the output line. In the whole process of crossing the switch, the packets go through a single buffer only. Here, we do not take into account the fragmentation/reassembly buffer at the line-cards (that is necessary in case of fixed-size cell scheduling), as the further virtualization architecture is independent on it.

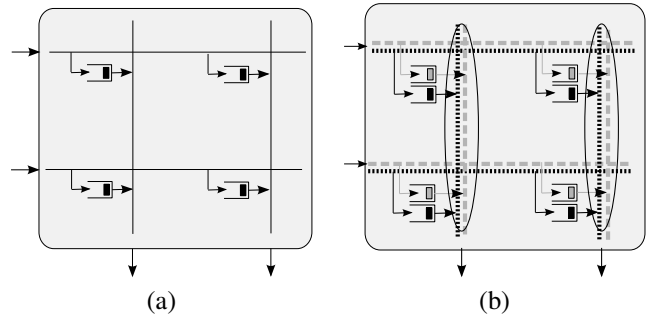


Figure 1. (a) a Crosspoint-Queued switch and (b) a Virtualized CQ-switching fabric.

C. Virtualizing the CQ architecture

The resources to be shared in a switch are link capacity (in time) and buffer size (in space). As said before, each crosspoint of a CQ switch implements a queue. Virtualizing this architecture, each crosspoint buffer is divided into V virtual buffers, each one implementing a virtual queue. If $b_{i,j}$ is the physical queue size at the crosspoint (i, j) , it is shared among the VSes, with each $VS^k, k \in [1, V]$ owning $b_{i,j}^k$ of the queue such that $\sum_{k=1}^V b_{i,j}^k \leq b_{i,j}$ at each crosspoint. In this paper, we refer to the virtual buffers as ‘virtual crosspoint buffers’ (VXBs). Table I lists some of the commonly used notations in this paper.

The sharing of input/output ports is similar. That is, the link capacity of a physical port is shared among different VSes. Virtualizing ports gives rise to ‘virtual ports’, each receiving a part of the capacity of the physical port. In short, if there are V VSes all requiring a share of port i , then each virtual port of VS^k has a capacity c_i^k such that $\sum_{k=1}^V c_i^k \leq c_i$. This sharing is

N	Number of physical ports
$\mathbf{C} = [c_i], i = 1 \dots N$	Capacity vector for physical switch
$\mathbf{B} = [b_{i,j}]_{N \times N}$	Matrix of CP buffer sizes for physical switch
V	Number of VSes
$1 \leq N^k \leq N$	Number of ports of VS^k
$\mathbf{C}^k = [c_i^k], i = 1 \dots N^k$	Capacity vector for VS^k
$\mathbf{B}^k = [b_{i,j}^k]_{N^k \times N^k}$	Matrix of CP buffer sizes of VS^k

Table I
TABLE OF NOTATIONS

performed by a scheduler whose service discipline is adjusted to satisfy the criteria of all VSes owners while optimizing the productivity of the physical host-switch. Details on scheduling are given in Section III-A.

Fig. 1(b) depicts a virtualized switch based on the CQ architecture. In this simplified example, the switch hosts two virtual switches, each one using all the physical ports. Incoming packets are demultiplexed to the corresponding VS after a lookup action. This classification can depend on header fields, for example, the IP source/destination addresses, VLAN tag, ToS, or an additional shim header like in MPLS. One could also imagine using a controller such as OpenFlow [11] to dynamically change and identify virtual networks. A classified packet is sent to the corresponding VXB if it has enough space left. Scheduling takes place at each output port. A scheduler per physical output port chooses among the virtual switches at each time-slot, then another scheduler, inside the chosen VS and the output port, selects a VXB to be dequeued.

The mechanism of classification and queuing to different virtual buffers is derived from QoS management in routers, where flows are grouped into service classes and memory is partitioned into logical queues. While, in classical QoS implementations, shaping and policing are usually performed on an output port with a set of virtual buffers, the virtualized crosspoint-queue switch implements shaping in a decentralized manner scheduling several crosspoint queues per output port.

III. RESOURCE-SHARING STRATEGIES

The resources of the virtualized switch are shared in time and space. Sharing in time is done by scheduling algorithms determining at which moment each virtual buffer is dequeued. Sharing in space takes place in the buffers which are split into multiple virtual queues. Each crosspoint buffer is shared into VXBs, one per VS using this crosspoint. Fig. 2(a) shows an example of two crosspoints used by three VSes. Incoming packets at each port are classified (at \mathcal{C} on the figure) to the corresponding VXB. For a given output port, a scheduler decides from which crosspoint and from which VXB the next packet will be dequeued.

A. Scheduling

A scheduler per output port schedules the VSes as well as the VXBs inside each VS. For isolation and customization of VS scheduling, there are two levels of scheduling. An *Inter-VS* scheduler selects among the VSes, which one to select. Each VS has an *Intra-VS* scheduler, which selects from which VXB to dequeue. Both the inter- and intra-VS schedulers can operate using different policies. The choice of the policy will depend on the type of each hosted VS, and on the switch

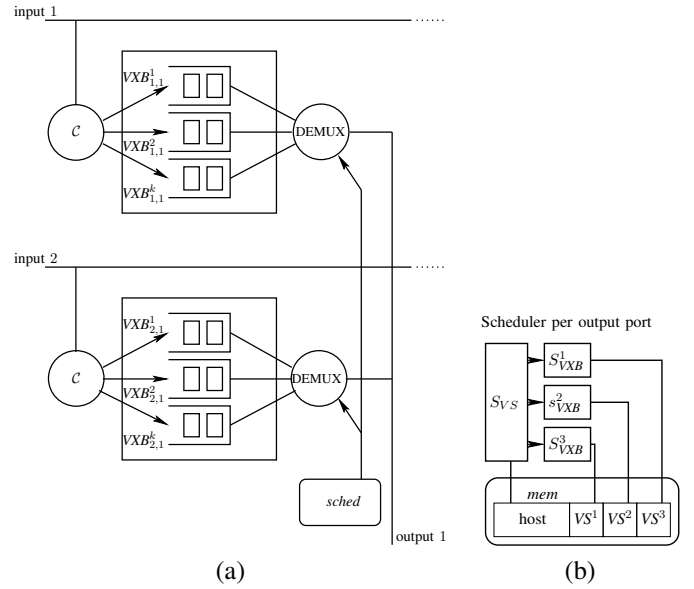


Figure 2. Crossbar architecture with (a) virtualized crosspoints and (b) a VS scheduler per output port.

maintainer's interest in maximizing utilization. As an example, a switch maintainer hosting several VSes, each one switching bulk traffic that needs high throughput, and having no strict constraints on delay and burstiness, can use a variable sharing policy. This means, it allows VSes to temporarily exceed their bandwidth, and risk in turn to get smaller bandwidth at another moment while other VSes exceed their bandwidth. For the switch maintainers, this brings the advantage of maximizing their equipment use, and for the VS operators, of not paying for peak rate guarantees if it is not strictly required for them.

In another scenario, one of the hosted VSes might have delay-sensitive traffic requiring strict minimum bandwidth guarantees. Hosting such a VS would imply a hosting switch to use another scheduling model in order to satisfy the VS's delay constraints and avoid impact from other VSes.

Inside a VS, an operator can also chose different strategies, e.g., Round-Robin (RR), Longest Queue First (LQF), or First Come, First Served (FCFS), depending on the requirements in terms of delay, jitter, buffer-size, or scheduling complexity. In an operational network, the choice of the scheduling and sharing strategies comes with a business and pricing model which are not developed here. It focuses primarily on the architecture allowing this flexibility in scheduling. Note that inter-VS scheduling policies are identical on the different output ports and so are intra-VS scheduling policies.

An architecture of this type of two-level scheduling module within the crossbar is given on Fig. 2(b). The inter-VS scheduler S_{VS} shares capacity between the different VSes. At each scheduling decision, it selects which VS is scheduled and for how long. Each VS has an internal VXB scheduling module S_{VXB}^k responsible for this, which is called by the inter-VS scheduler. Inter-VS and intra-VS schedulers logically share the scheduler's memory to register state information. This requires N times more memory for the scheduler, when the switch hosts N VSes. When a VS is scheduled, it acts like a physical switch, but choosing only from its own VXBs, and dequeuing only during the time period which it has been attributed. Each

VS having at maximum N VXBs, each intra-VS scheduler has a complexity of $O(N)$. As only one intra-VS scheduler operates at a time, the overall scheduling complexity is also of $O(N)$.

B. Buffer-sharing policies

While scheduling aims at guaranteeing a specific rate for the ‘virtual link’, the ‘virtual buffer’ size is defined by the different possible buffer-sharing policies. The physical crosspoint queue can be shared among different VSes in a strict way or in a flexible way. If the sharing is strict, a VS can enqueue packets only at the queues it owns. Such a setting may lead to an inefficient use of buffers, as packets belonging to a VS are dropped when the corresponding VXB is full, even though there is space for it in the physical crosspoint queue.

This is where another mode of buffer sharing is helpful. In the flexible buffer sharing policy, a VS can queue its packets at VXBs belonging to other VSes provided there is no space in its VXB. Such a dynamic buffer sharing reduces the number of losses (compared to the strict buffer-sharing policy). This allows a number of dynamic buffer-sharing policies — every VS^k can decide to set aside a fraction of its queue size, say $F_{i,j}^k$, that can be used by other VSes that share the crosspoint (i,j) . The order in which packets are dequeued can also be an interesting decision criteria. A VS operator would like to give priority to its packets before dequeuing $F_{i,j}^k$.

IV. SIMULATION

To analyze the performance of the described virtualized switch architecture and its scheduling paradigms, we developed a simulator that runs in parallel multiple VSes, simulating the sharing of common physical resources. Exploring the benefits of crosspoint buffers, it operates asynchronously on variable-size packets. Different VS-scheduling algorithms are applied to explore different resource-sharing strategies.

A. Objectives and performance metric

The first objective of this study is to observe the loss of (physical) switch performance due to the resource sharing that is required among different VSes. Another objective is to evaluate according to VS efficiency. The different resource-sharing policies that we discussed earlier will lead to different performance levels. With an example of strict sharing, compared to priority scheduling, we intend to quantify these differences here.

The metrics we consider are: throughput, loss, and delay. The throughput of a switch is defined as the percentage of arrived packets that departed the switch. In a virtualized switch, to calculate the total throughput, we use the sum of the arrived packets which were admitted in any of the VSes. The loss rate is complementary to the throughput, being the percentage of arrived packets which are dropped due to queue overflow. The delay is the time spent by a packet in the switch. The scalability of the virtualized switch is analyzed by varying the number of VSes it hosts for different buffer sizes.

The generated input traffic emulates a bursty network traffic. A Markov Modulated Poisson Process (MMPP, refer Fig. 3) is

used for the arrival process. The arrivals are in packets, with sizes taken from bimodal uniform distribution, where the first mode varied from 64 to 100 bytes and the second mode varied from 1350 to 1500 bytes.

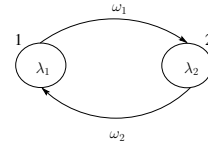


Figure 3. MMPP model: λ_1 and λ_2 are the arrival rates; ω_1 and ω_2 are the transition rates

B. Experiments

The simulations were executed using the described input traffic scenario. Each configuration was run 10 times. The average values are presented. The physical crosspoint buffer (CB) sizes vary as multiples of 1500 bytes, since it is the minimum size necessary to hold one maximum-size packet. The simulated physical switch has $N = 32$ ports. The input traffic is generated at each physical input port with a load of $\rho = 1/N$, in order to charge the output port by a load of 1. It is then split into the different VSes according to weights. Similarly, we assume that the traffic at each input port is uniformly distributed to the output ports. Thus, for simplicity, we explain the details with respect to a single output port.

1) *Virtualized switch performance:* The first experiment evaluates a virtualized switch hosting homogeneous VSes (same VXB size and same virtual port capacities). Input traffic for each VS arrives with a load of $1/(N * V)$ on each port, to have a total load of 1 on the output port. VSes are served using a deficit round-robin (DRR) [12] scheduler to ensure the fair sharing of output link rate, serving each VS during at least $1/V$ of a scheduling round, as long as it has packets to dequeue. All the VSes have the same quantum $q = 1500$. This means that when a VS is scheduled, it can dequeue at maximum 1500 bytes. This small quantum size minimizes rate variation and delay in the VSes and maximizes isolation between them. Inside each VS, a separate instance of DRR is used to dequeue packets in a fair way from the different VXBs. The quantum of each VXB is also chosen equal to 1500 bytes. Note that, independently of the host switch’s VS scheduling, a VS could also use another VXB scheduling algorithm like FCFS, LQF, etc. in a heterogeneous virtualized switch.

In the different runs, the number of VSes as well as the size of the CBs are varied to evaluate the scalability of the switch, and the impact of buffer size on the performance. Fig. 4 shows the throughput of the virtualized switch hosting 2 to 32 VSes and having crosspoint buffers sizes between 3 and 192 KB. The result numbered ‘1’ on the x axis corresponds to the performance of the physical switch. Per VS throughput is not represented here, as it is equivalent to the total throughput of all VSes. Mean deviation is of negligible significance in these results.

In this configuration, with up to 32 VSes, the total throughput of the physical switch is always above 90%, given a VXB size (CB/V) of at least 1500 bytes. The throughput is

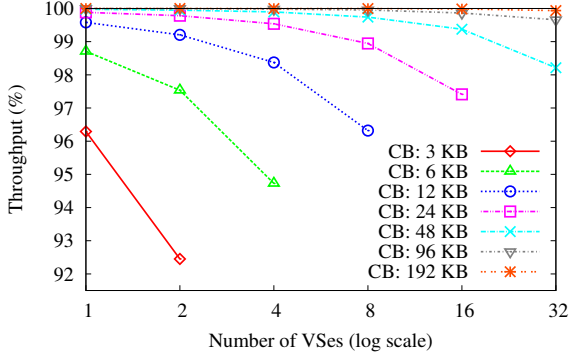


Figure 4. Throughput of a virtualized switch with different CB sizes, hosting different numbers of VSes.

highly dependent on the CB size. By segmenting the buffers into fixed-size VXBs, the usable buffer capacity shrinks by a fraction, and this inefficiency increases with the number of VXBs. Thus, enabling the hosting of several VSes with isolated VXBs requires the host switch to scale in buffer size, at least to give each VXB the size of one maximum size packet. A CB size of 96 KB allows over 99% of throughput for up to 32 VSes. For a crossbar with 32 1 Gbps-port, this would require a total of about 100 MB of buffer.

Inefficiency for any buffer size can be decreased using variable buffer sharing, and also by more traffic-aware scheduling. In this example of homogeneous sharing, VSes are served in a fixed order. Hence, a VS might be scheduled at a moment where it has few packets to dequeue, thus blocking other virtual switches with full queues. This might be particularly penalizing for 'small' VSes using only a small part of the resources and being scheduled after very big intervals, thus suffering from high delays in round-robin.

2) *Heterogeneous switch*: In this experiment, the performance of two heterogeneous VSes sharing the same physical switch, is analyzed. VS^1 requires little bandwidth—10% of the available capacity—on each port, and minimal delay. VS^2 needs a high bandwidth—90% of the available capacity—on each port, and has no constraints on delay. In this scenario, DRR scheduling like before would cause high delays to VS^1 , giving it only one quantum in 10, to dequeue a packet. As an alternative, strict-priority (SP) scheduling is implemented in the physical switch, always giving VS^1 priority over VS^2 . Hence, VS^1 can dequeue packets when they arrive, thus momentarily exceeding the 10% of reserved average rate. The resulting variable-sharing scheduling policy is compared to the previous fair sharing with DRR. Inside each VS, DRR is used. In both cases, the throughput and delay are evaluated as a function of different allocations of the 15-KB CBs among the two VXBs. Fig. 5 and 6 show, respectively, the average throughput and the delay obtained on each VS, in the 10 runs.

Giving priority to VS^1 increases the loss on VS^2 by less than 1%, compared to DRR, while throughput of VS^1 is highly increased. It reaches 100% for a VXB size starting from 3 KB, while with DRR, VS^1 obtains 99% of throughput starting only from a buffer size of 6 KB, due to increased latency, as VS^1

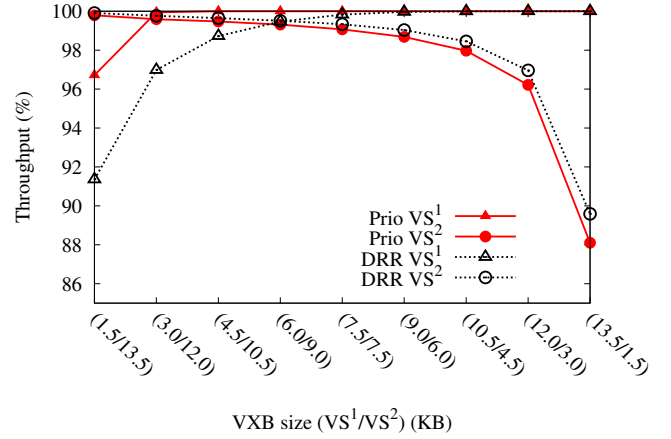


Figure 5. Throughput of a virtualized switch hosting two VSes, using respectively 10% and 90% of the capacity.

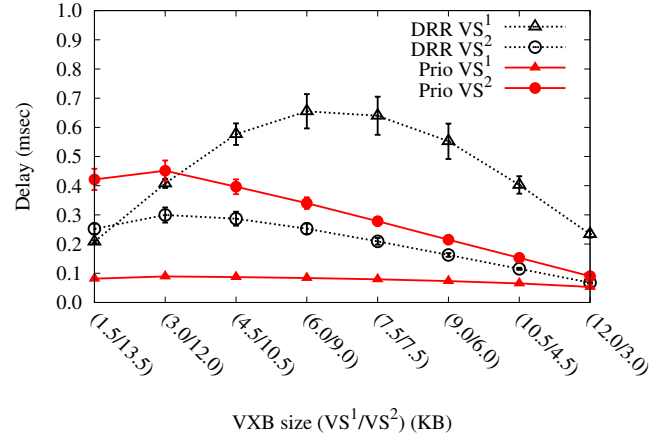


Figure 6. Delay of two VSes hosted on a physical switch and using respectively 10% and 90% of the capacity.

is scheduled only once in 10 quanta.

Regarding throughput, giving priority to a low-bandwidth VS improves, on the average, the individual throughput of both VSes, even if the total physical switch throughput decreases very slightly (by less than 1%). Regarding the delay, the improvement using SP is even more important. Fig. 6 shows that VS^1 , using SP scheduling, decreases its delay by 0.1 to nearly 0.6 ms. Meanwhile, the delay of VS^2 is increased by less than 0.2 ms in the worst case of having only a 1500-bytes VXB. Also, standard deviation is depicted in both scheduling scenarios. Giving VS^1 priority over VS^2 allows it to achieve an almost constant delay with very slight variation, while with DRR, it varies by up to 0.15 ms around the average. The delay deviation of VS^2 is negligible and similar in both scenarios, varying around less than 0.05 ms. For VS-operators, in this configuration, priority scheduling sounds to be the better solution, at the cost of a very small decrease in the overall throughput.

These experiments give an insight on the variation of VS performance, depending on the type of resource sharing employed. Further investigation on this virtualized architecture is

necessary to choose the best sharing strategies.

V. RELATED WORK

Different directions have been taken to build virtual routers. Vendors propose control-plane virtualization [5], [13], to have several routing instances running on the same forwarding engines. This allows customized routing strategies and redundancy in the router configuration, which has also been proposed in software [14]. However, the absence of control in the resource sharing does not allow this approach to share the hardware fairly between concurrent users, nor to provide them with deterministic performance guarantees. Data-plane virtualization offering isolation and control has been proposed [6] to spread the devices of a same physical switch between several virtual switches. This means that physical ports are exclusively attributed to virtual switches. A similar approach consists in scaling the routing engine and assigning different forwarding engines to each routing engine [5]. But in both cases, a single physical port can not be shared by different virtual switches. By segmenting the resources in this way and giving dedicated ports to virtual routers, isolation and deterministic performance are guaranteed, but the approach is less scalable and flexible and the resources risk to be over-provisioned.

Virtualization at layer 2 and port sharing is implemented in virtual LANs (VLAN) [15]. While multiple VLANs can use the same physical ports, sharing is controlled through differentiated queuing on the ports in QoS-enabled equipments. Software data-plane virtualization [16] has been used to build virtual routers on commodity servers [7], [17]. This approach is interesting for proof of concept and evaluation, but gives limited performance. Aggregating several servers to build a single virtual software router has been proposed as a solution to this performance issue [18].

NetFPGA hardware allows to have much more efficient performance, forwarding directly in the hardware, and causing no virtualization overhead [19]. Such a virtualized router does not currently offer custom link-capacity allocation nor full isolation. Rate-control features have been proposed in the Crossbow Virtual NICs [20], which is a network-stack implementation in the OpenSolaris OS. To efficiently control the resource sharing, it is combined with virtualization support in the physical NICs. Here, software virtualization overhead is removed. But this kind of router might not be adapted to the Internet in terms of packet-forwarding rates. Therefore, it is necessary to stay as close as possible to the physical-router architecture while designing a virtualized router.

An alternative solution to switch virtualization is OpenFlow [11], which, in combination with FlowVisor [21], allows to slice the network and to assign each slice to a different priority queue on the switch ports. FlowVisor offers basic QoS and is limited to eight different priority queues with respect to actual switch features, while the virtual-switch architecture proposed here overcomes these limitations and allows precise per VS packet scheduling strategies.

VI. CONCLUSIONS

This paper presented a design for a virtualized switch that is able to host several customizable virtual switches. Such a

design gives operators a high flexibility to choose independent configurations for their virtual switches. Different service disciplines can be chosen to schedule VSes. Simulation results showed that the performance overhead in virtualizing a switch is acceptable, and depends on the resource dimensioning. Also, different scheduling strategies help to individually satisfy different performance requirements of different virtual switches. Further investigation is planned on other scheduling strategies to cover different combinations of VSes sharing the same resources. Moreover, we plan to develop the virtual switch model with programmable hardware. Finally, management paradigms need to be defined with the integration of virtual switches/routers into virtual networks, such as allocation, mobility and pricing strategies.

REFERENCES

- [1] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *Proc. SIGCOMM '06*, pp. 3–14, ACM, 2006.
- [2] "Geni design principles," *Computer*, vol. 39, no. 9, pp. 102–105, 2006.
- [3] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *SIGCOMM CCR*, vol. 37, no. 1, pp. 61–64, 2007.
- [4] F. Anhalt, G. Koslovski, and P. Vicat-Blanc Primet, "Specifying and provisioning Virtual Infrastructures with HIPerNET," *ACM International Journal of Network Management (IJNM)*, 2010.
- [5] "Control plane scaling and router virtualization," Tech. Rep. 2000261-001-EN, Juniper Networks, Feb. 2009.
- [6] "Technical overview of virtual device contexts," tech. rep., Cisco Systems, 2008.
- [7] N. Egi, A. Greenhalgh, M. Handley, M. Hoerd, F. Huici, and L. Mathy, "Fairness issues in software virtual routers," in *Proc. ACM PRESTO '08 workshop*, pp. 33–38, ACM, 2008.
- [8] S. Iyer and N. McKeown, "Techniques for fast shared memory switches."
- [9] R. E. Tarjan, *Data structures and network algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1983.
- [10] Y. Kanizo, D. Hay, and I. Keslassy, "The crosspoint-queued switch," in *IEEE INFOCOM 2009*, 2009.
- [11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, 2008.
- [12] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *SIGCOMM CCR*, vol. 25, no. 4, pp. 231–242, 1995.
- [13] "Router virtualization in service providers," tech. rep., Cisco Systems.
- [14] "Virtually Eliminating Router Bugs," in *CONEXT '09: Proc. of the 2009 ACM CoNEXT Conference*, ACM, 2009.
- [15] "Virtual bridged local area networks, ieee std 802.1q-2005." IEEE Computer Society, 2005.
- [16] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. SOSP '03*, pp. 164–177, ACM, 2003.
- [17] N. Egi, A. Greenhalgh, M. Hoerd, F. Huici, P. Papadimitriou, M. Handley, and L. Mathy, "A Platform for High Performance and Flexible Virtual Routers on Commodity Hardware." *SIGCOMM 2009 poster session*, Aug. 2009.
- [18] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy, "Routebricks: exploiting parallelism to scale software routers," in *Proc. ACM SOSP '09*, pp. 15–28, 2009.
- [19] M. B. Anwer and N. Feamster, "Building a fast, virtualized data plane with programmable hardware," in *Proc. ACM VISA '09 workshop*, pp. 1–8, ACM, 2009.
- [20] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaid, "Crossbow: from hardware virtualized nics to virtualized networks," in *Proc. ACM VISA '09 workshop*, pp. 53–62, ACM, 2009.
- [21] Rob Sherwood and Glen Gibb and Kok-Kiong Yap and Guido Appenzeller and Martin Casado and Nick McKeown and Guru Parulkar, "FlowVisor: A Network Virtualization Layer," Tech. Rep. OPENFLOW-TR-2009-1, OpenFlowSwitch.org, oct 2009.