Analysis and Experimental Evaluation of Network Data-Plane Virtualization Mechanisms

Fabienne Anhalt and Pascale Vicat-Blanc Primet
INRIA, LIP - ENS Lyon
{fabienne.anhalt, pascale.primet}@ens-lyon.fr

Abstract

Combining end-host, server and router virtualization could offer isolated and malleable virtual networks of different types, owners and protocols, all sharing one physical infrastructure. However, the virtualization of the data plane may lead to performance degradation and unpredictability. These arise not only due to additional processing, but also from the sharing of physical resources like memory, CPU and network devices. This article analyses network virtualization from the data plane's perspective. We explore the resulting network performance in terms of throughput, packet loss and latency between virtual machines, as well as the induced CPU cost. The virtual machines act as senders or receivers, or as software routers forwarding traffic between two interfaces in the context of Xen and KVM. Our results show that the impact of virtualization with Xen has become smaller with its successive versions. Moreover, compared to KVM, Xen gives better network performance with less processing overhead. Therefore, Xen currently seems to be one of the most interesting software data-plane virtualization technologies.

***Keywords*-Network virtualization; data plane; performance; Xen; KVM**

I. Introduction

System-virtualization technology has been recently redefined in the context of distributed systems and enterprise servers. By running several virtual servers on one physical machine, organizations and companies can better exploit the physical resources of one machine in terms of CPU and energy consumption: often, a server uses only a small part of the available hardware resources. Moreover, server virtualization allows increasing security, due to the isolation factor. Also, migration and mobility enhance reliability.
To introduce these benefits into the network itself, an emerging idea is to implement virtualization not only on the end-host servers or at link level, but also on network devices, like routers. This technology could make coexist networks or autonomous systems of different types, owners, and protocols over one physical network. Such virtual IP networks are an emerging approach to provide virtual network infrastructures as a service, allowing users to create customized networks with personalized IP routing paradigms over a shared physical infrastructure. They can be used for research (implementation and test "in the wild" of new protocols) and further to decouple the service from the infrastructure in the real Internet, rethinking the backbone architecture.
However, if virtualization could potentially solve the main issues of the current Internet (security, mobility, reliability, reconfigurability), it would nevertheless introduce an overhead due to the additional layers inserted between the virtual machines and the hardware. In particular, if the data plane is virtualized, which allows the greatest customization, all network traffic has to cross these virtualization layers, which impacts performance. Furthermore, in addition to the routing and bandwidth-allocation problems of traditional routers, we have to take into account the problem of local-resource sharing by different virtual networks. Considering this sharing of resources—e.g., the network interfaces, the processors, the memory(buffer space), the switching fabric—it is a challenge to get a predictable, stable and optimal performance.
To have a better insight into these issues, this article analyzes the current network data-plane virtualization solutions. We evaluate their properties and potential in terms of network performance like throughput, packet loss, latency and the induced CPU cost. End-host network performance is first analyzed to evaluate virtualization in a simple sending or receiving scenario with a single network interface, using different system configurations. We propose our design of a virtual router prototype with a virtualized plane, and then evaluate virtual forwarding.
This article is an extension to our previous work on Xen's network performance [1]. It details the formerly announced

results and proposes a comparison to the recent KVM virtualization technology.
The rest of this article is organized as follows. The next section describes the context and background of network virtualization. Section III analyses the technology of virtualizing the data plane. Section IV discusses experimental results on network performance. Finally, section V compares our results to related work, and section VI concludes this article.

II. Background

A. Towards network virtualization

Virtualization came up to cross the barrier of physical hardware, and to share a resource between several users, giving each one the illusion that the resource belongs entirely to him. It was introduced by IBM in 1973 [2] and became very popular with the arrival of solutions like Xen [3] and VMware [4]. This approach is very useful to enhance isolation, mobility, dynamic reconfiguration and fault tolerance. It is now widely used by companies and institutions using only a few physical servers to host a multitude of virtual servers. Multiplying their servers this way allows them to isolate services, facilitate reconfiguration and maximize fault-tolerance and flexibility. They can easily duplicate servers, and thus deal with hardware problems, as virtual servers can be migrated from one physical machine to another if necessary. Thus, virtualization makes institutions less dependent on the hardware while optimizing their utilization. The emerging cloud concept currently pushes the level of abstraction from the hardware even a step further. Commercial cloud providers (e.g., [5][6][7]) propose to host the company's data-centers and services inside virtual machines on the servers they manage. Hence companies do not need to maintain hardware servers anymore. By externalizing their data and services, they can considerably reduce costs, and enjoy more flexibility, as they can rent the necessary amount of server resources on demand.
While contents and services moved to a virtual environment, the network has to take the same direction, to get the same benefits. By virtualizing not only end-host servers, but also network nodes and links, the physical network infrastructure can be shared by several virtual networks, thus optimizing resource utilization. Users of virtual networks have the illusion to manipulate a dedicated network, and virtual-node and link-migration simplicity can improve fault tolerance and optimize resource allocation. Hence, the network becomes a flexible resource which can be booked on demand, for example by providers, just like companies rent clouds today. This gives virtual-network providers the flexibility to allocate resources dynamically over the physical network and have end-to-end control on their virtual paths [8]. In this scope, regarding the utilization of virtual networks on the Internet, isolation is very important to guarantee confined virtual environments with performance guarantees to each virtual network operator who wants to offer a continuous service to its clients.
In this context, research was conducted on infrastructure virtualization, including virtual routers as a new resource. As an example, VINI [9] allows several virtual networks to share a single physical infrastructure. Researchers can run experiments in isolated virtual network slices with real routing software they can personalize. Trellis [10] is a network-hosting platform deployed on the VINI facility. It is a virtual-network substrate that can run on commodity hardware. It is built using VServer [11] container-based virtualization. In this implementation, only the control-plane is virtualized which enables only limited isolation between virtual networks. However, these solutions show interesting new functionalities, hosting several virtual networks with customizable routing paradigms on a single physical infrastructure. Therefore, data-plane virtualization is required in addition to control-plane virtualization to add the right level of confinement and enable fairness.

B. Data-plane virtualization techniques

Virtualizing the data plane means having a separate and independent data plane and virtual hardware in each virtual machine. To manage the virtual hardware, each virtual machine runs a separate OS and the virtualized hardware is shared among the virtual machines. Basically, this sharing can be performed by software emulation of the hardware, exposing emulated hardware to each virtual machine [12] or using hardware with virtualization extension [13][14]. As emulation requires the translation of all the instructions, it has a very important performance overhead. Using hardware virtualization, virtual machines instructions are executed directly on the hardware which offers accelerated performance. Nonetheless, generally not all hardware supports virtualization. If modern CPUs are provided with hardware virtualization support, most of the time I/O devices like network interfaces need to be emulated in software to be used in fully virtualized systems.
An alternative to emulation is *paravirtualization*, where each virtual machine uses special virtual drivers to access the hardware devices. This requires patching the virtual machine's OS to include the virtual drivers, but device access is more efficient because the virtual driver's design is adapted to the virtualization mechanism. For this reason, paravirtualization seems to be the most promising solution for network-intensive virtual machines—assuming the virtual machine kernels can be patched, which is now possible for the major OSes. Figure 1 illustrates the difference

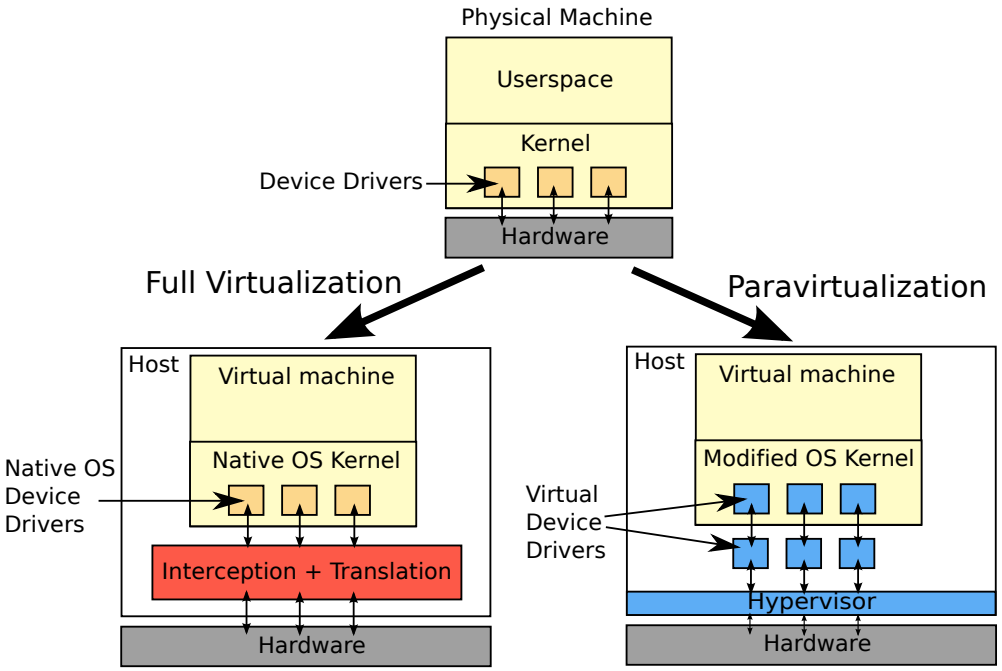

Figure 1. Full virtualization and paravirtualization.

between full- and paravirtualization.

The most commonly used paravirtualization solution is presently Xen [3]. A more recent virtualization technology is KVM [15], which initially supported only full virtualization with hardware-assisted CPU virtualization and emulated I/O device drivers. When the virtual machines are network-intensive, an important performance overhead is expected, as emulation is a very costly procedure. With the appearance of the *virtio* [16] tools proposing a set of virtual I/O drivers, KVM became able to perform paravirtualization as well, to leverage the I/O performance. In the following, to shed light on these technologies, Xen's and KVM's network mechanisms are described in detail, and their performance is evaluated.

III. Network Virtualization

This section details the mechanisms used by paravirtualization techniques to virtualize the network devices. In Xen [3], each virtual machine resides in a so-called guest domain. Among the guest domains, only a driver domain, domain 0 (dom0) by default, has direct access to the hardware. All the other domains, called domain U (domU) for "Unprivileged", need to use virtual interfaces. A virtual-machine monitor, called hypervisor, manages all the virtual machines and the access to the hardware devices. In KVM, the virtual machines are created as device nodes, and the host system operates as the hypervisor. It runs two KVM kernel modules, a modified QEMU [12] module performing hardware-device emulation, and a processor-dependent module to manage hardware virtualization.

A. Data path in Xen

The virtual machines in Xen access the network hardware through the virtualization layer. Each domU has a virtual interface for each physical network interface of the physical machine. This virtual interface is accessed via a *split-device driver* composed of two parts, the front-end driver in domU and the back-end driver in dom0 [17]. Figure 2 illustrates the path followed by the network packets emitted on a virtual machine residing inside a domU. The memory page where a packet resides in the domU kernel is either mapped to dom0 or the packet is copied to a segment of shared memory by the Xen hypervisor from where it is transmitted to dom0. Inside dom0, packets are bridged (path 1) or routed (path 2) between the virtual interfaces and the physical ones. The reception of packets on a domU is similar. To receive a packet, domU gives a grant to dom0 so that dom0 can access the grant page and copy the packet to domU's kernel space [18].

The additional path a packet has to go through due to virtualization is marked by the dashed line. A significant processing and latency overhead can be expected due to the additional copy to the shared memory between domU and dom0. Moreover, the multiplexing and demultiplexing

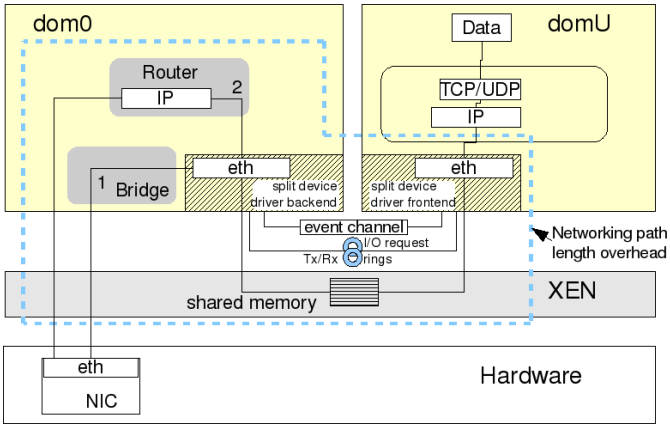

Figure 2. Path of a network packet with Xen, from a domU to the NIC.

of the packets in dom0 can be expensive.

B. Data path in KVM

Similar to Xen, KVM uses a virtual driver in paravirtualization mode. This virtual driver is part of the virtio [16] I/O drivers used within the Linux kernel. Virtual machine kernels also use a front-end driver with particular code to communicate with a back-end driver which interfaces with the KVM module inside the Linux kernel [19]. This architecture is represented on Figure 3. To communicate

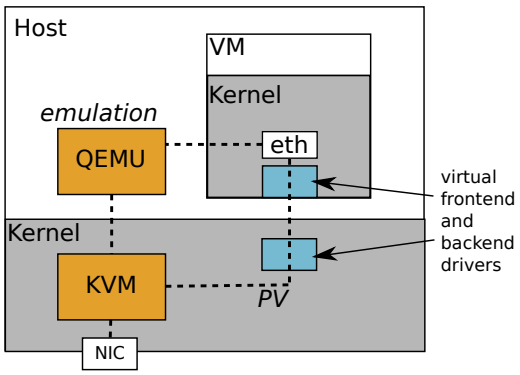

Figure 3. Path of a network packet in KVM using paravirtualization (PV) or full virtualization (emulation).

between front-end and back-end, ring-buffers are used for the implementation of net channels, like in the Xen virtual driver.
In its original full-virtualization mode, KVM emulates I/O drivers using a QEMU [12] module in the userspace.

C. Routers data plane virtualization

The described virtualization techniques can be used for fully (i.e., control-plane and data plane) virtualized software routers, to implement virtual network infrastructures as described before. In particular, we used Xen, to implement such a virtual router, due to its more promising performance it showed in our end-host experiments, compared to KVM. Figure 4 shows an example of such an architecture with software routers uploaded (control- and data-path) into virtual machines to create virtual routers. In this example, two virtual routers share the resources (NICs, CPU, memory) of a single physical server. The governing principle inside such virtual routers

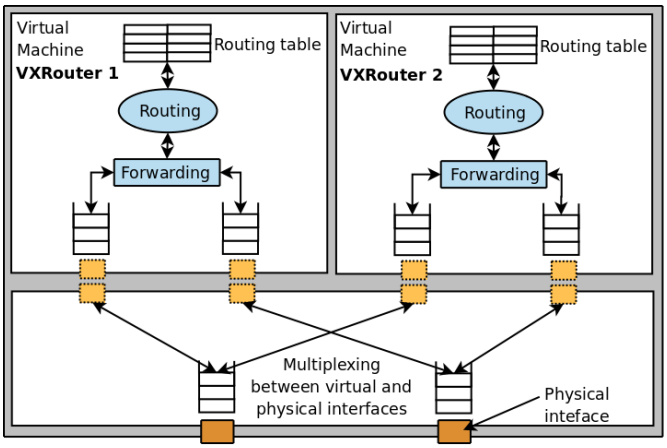

Figure 4. Machine with 2 virtual routers sharing the 2 NICs.

is the same as inside a standard software router, except that the virtual machines do not have direct access to the physical hardware interfaces. The packets are forwarded between the virtual or emulated interfaces and the corresponding physical interfaces thanks to a multiplexing and demultiplexing mechanism. This is implemented in an intermediate layer located between the hardware and the virtual machines, which corresponds to the hypervisor and the host's operating-system kernel in the virtualization techniques described before. For example in Xen, this layer corresponds to the hypervisor and the driver domain or dom0. As a consequence, there is additional computing in this model whose impact on the network performance needs to be analyzed.

D. Performance problem statement

An efficient usage of virtual machines for networking requires a certain number of non-functional properties like *efficiency*, *fairness* in resource sharing and *predictability*

of performance. We define and evaluate these properties using the following metrics. Given $N \in \mathbb{N}$ the number of virtual machines inside the considered physical machine, the defined metrics are listed in Table I.

R_i	Throughput of virtual machine i, $i \in [1, N]$
$R_{aggregate}$	$\sum_{i=1}^{N} R_i$: aggregate throughput
$R_{aggregate}/N$	Effective fair share of the rate
C_i	CPU cost on virtual machine i, $i \in [1, N]$
$C_{aggregate}$	$\sum_{i=1}^{N} C_i$: total CPU cost
L_i	Latency on virtual machine i, $i \in [1, N]$
$R_{classical(T/R)}$ $R_{classical(F)}$	Throughputs for sending/receiving and forwarding on classical Linux without virtualization
$C_{classical(T)}$ $C_{classical(R)}$ $C_{classical(F)}$	CPU costs of sending, receiving and forwarding on a classical software router
$L_{classical(F)}$	Latency on a classical Linux router

Table I. Metrics.

For our proposal, we define the efficiency in terms of throughput by (1)

$$E_{throughput} = \frac{R_{aggregate}}{R_{classical}} \quad (1).$$

The fairness of the inter-virtual machine resource sharing is derived from the classical Jain index[20], defined by (2).

$$Fairness(x) = \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n \times \sum_{i=1}^{n} x_i^2} \quad (2)$$

where n is the number of virtual machines sharing the physical resources and x_i the metric achieved by each virtual machine i, for example the throughput or the CPU percentage.
Predictability and scalability are evaluated analyzing the performance according to the number of virtual machines.

IV. Experiments and Analysis

In this section, we examine the impact of the virtualization layer on the networking performance. In this context, at first, virtual end-host performance is evaluated with Xen. Then it is compared to results obtained on KVM. Finally, a virtual router is implemented with Xen, which showed better performance in virtual networking than KVM. This virtual router's forwarding performance is then evaluated.

A. Experimental setup

The experiments were all executed on machines reserved on the fully controlled, and reconfigurable French national testbed Grid'5000 [21]. The machines used for the initial experiments on Xen end-hosts were IBM eServers 325, with 2 AMD Opteron 246 CPUs (2.0 GHz/1 MB) with one core each, 2 GB of memory and a 1 Gb/s NIC. Virtual routers were built with Xen on IBM eServers 326m, with 2 AMD Opteron 246 CPUs (2.0GHz/1MB), with one core each, 2 GB of memory and two 1 Gb/s NICs. The latest experiments on KVM were executed on more recent machines provided with hardware virtualization enabled processors. These were Dell PowerEdges 1950 with two dual-core Intel Xeon 5148 LV processors (2.33 GHz) and 8 GB of memory. We tested with Xen that performance was equivalent in these two different hardware configurations. In each experiment, all the machines were located inside one LAN interconnected by a switch.
The software configurations used were Xen 3.1.0 and 3.2.1 with respectively the modified 2.6.18-3 and 2.6.18.8 Linux kernels. Comparative experiments were performed on KVM 84 with the Linux 2.6.29 kernel in the host system as well as in the virtual machine. Measurement tools were *iperf* [22] for the TCP throughput, *netperf* [23] for the UDP rate, *xentop* and *sar* for the CPU utilization, and the classical *ping* utility for latency.

B. Evaluation of virtual end-hosts

In the following experiments, network performance on virtual end-hosts implemented with Xen 3.1 and Xen 3.2 was evaluated. As some results with Xen 3.1 were not satisfying, dom0 being the bottleneck, a second run of the experiments on Xen 3.1 was performed, allocating more CPU time to dom0 (up to 32 times the part attributed to a domU). This choice was made as dom0 is in charge of forwarding all the network traffic between the physical and the virtual interfaces. This setup will be called *Xen 3.1a*. In addition, the obtained results were compared to KVM virtual end-hosts. KVM was used either in its native full hardware virtualization setup or using lightweight paravirtualization.

1) Sending performance: In this first experiment, the TCP sending throughput on 1, 2, 4 and 8 virtual hosts inside one physical machine, as well as the corresponding CPU overhead, were evaluated. The throughput per virtual machine and the aggregate throughput with Xen are represented on Figure 5. In both Xen configurations,

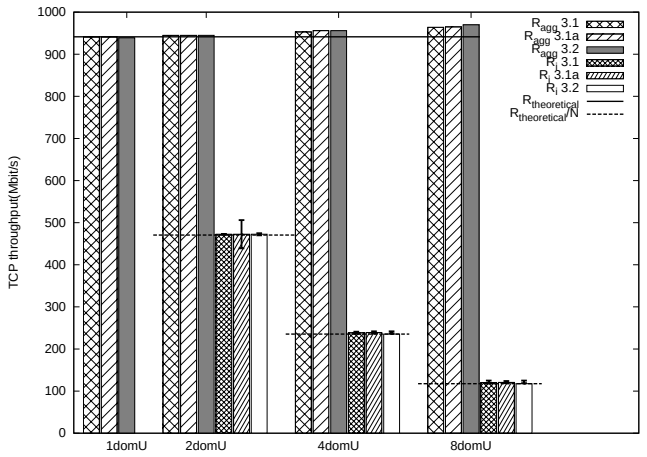

Figure 5. TCP Sending throughput on respectively 1, 2, 4 or 8 VMs with Xen versions 3.1 and 3.2.

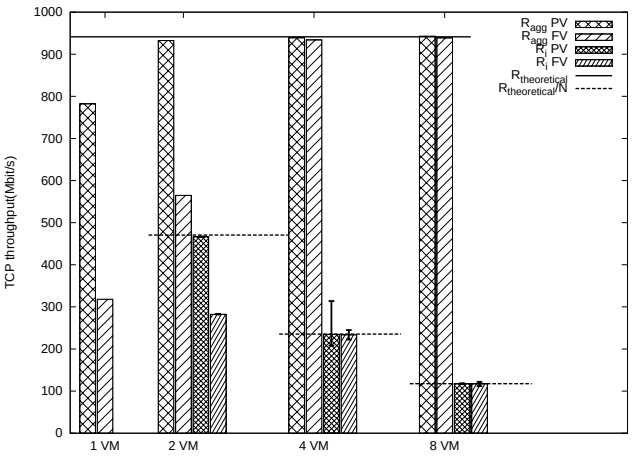

Figure 7. TCP Sending throughput on 1, 2, 4 or 8 VMs with KVM 84 using paravirtualization (PV) or full virtualization (FV).

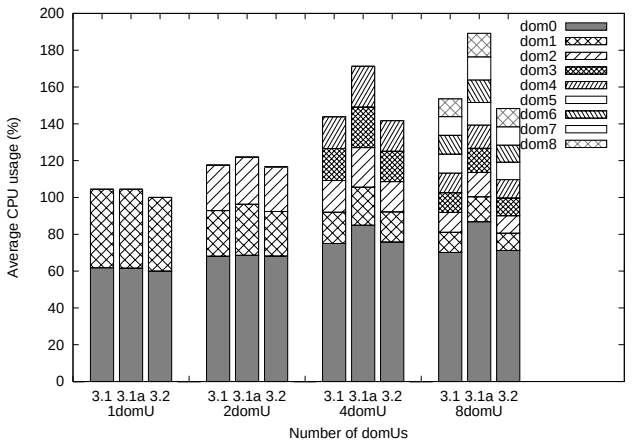

Figure 6. Average utilization of the two CPUs during TCP sending on 1, 2, 4 or 8 VMs with Xen.

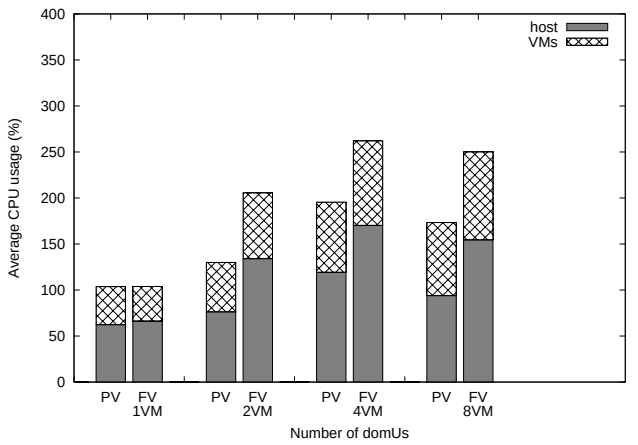

Figure 8. Average utilization of the four CPU cores during TCP sending on 1, 2, 4 or 8 VMs with KVM.

3.1 and 3.2, performance was close to classical Linux throughput $R_{classical(T/R)} = 938\ Mb/s$. In the 3.1a and 3.2 setups, the aggregated throughput obtained by all the virtual machines barely reached more throughput than on 3.1. We conclude that in the three cases (3.1, 3.1a, 3.2), the system is efficient and predictable, in terms of sending throughput. Indeed, the throughput per virtual machine corresponds to the fair share of the available bandwidth of the link ($R_{theoretical}/N$).

The associated average CPU utilization for each Xen guest domain is represented on Figure 6. For a single domU, around half the processing power of the two CPUs was used in the three setups (Xen 3.1, 3.1a and 3.2), whereas on a native Linux system without virtualization, we measured that only $C_{classical(E)} = 64\%$ of the two CPUs, out of 200%, was in use running the same network benchmark. In the experiment with 8 domUs, the CPUs were used at over 140%. The overall CPU overhead did not differ much between 3.1 and 3.2 setups. However, by increasing dom0's CPU weight (setup 3.1a), the overall CPU cost also increased while leveraging the throughput insignificantly. We notice that even though virtualization introduced a processing overhead, two processors like the ones used in these experiments achieved a throughput equivalent to the maximum theoretical throughput on 8 concurrent virtual machines, sending TCP flows of default maximum-sized packets on a 1 Gb/s link. Here,

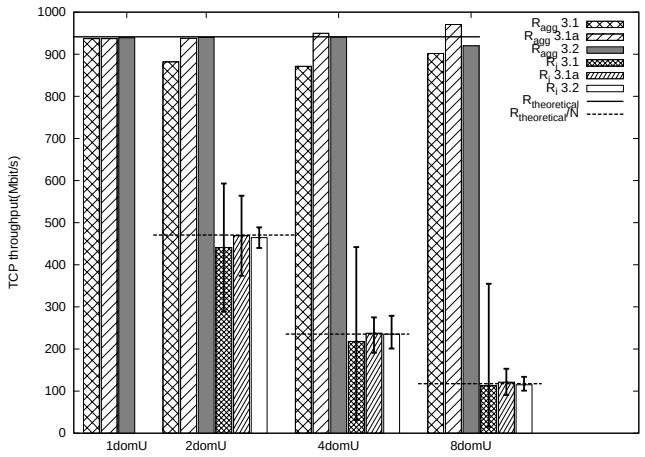

Figure 9. TCP Receiving throughput on respectively 1, 2, 4 or 8 VMs with Xen versions 3.1 and 3.2.

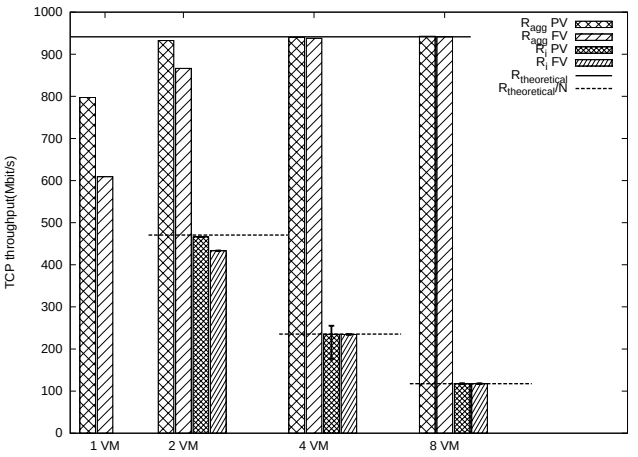

Figure 11. TCP Receiving throughput on 1, 2, 4 or 8 VMs with KVM 84 using paravirtualization (PV) and full virtualization (FV).

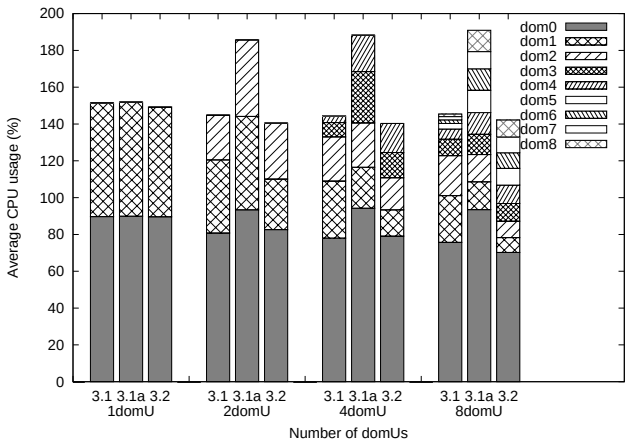

Figure 10. Average utilization of the two CPUs during TCP receiving on 1, 2, 4 or 8 VMs.

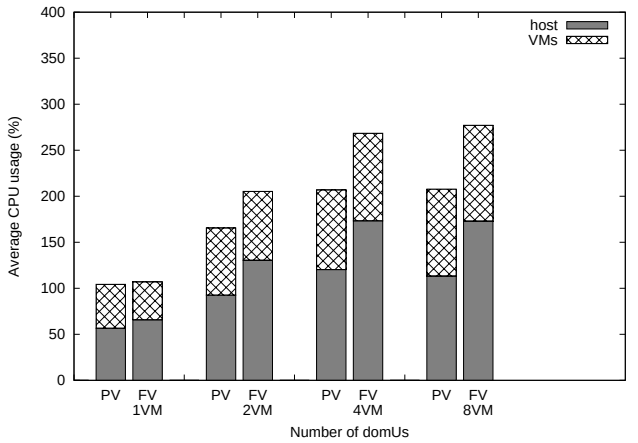

Figure 12. Average utilization of the four CPU cores during TCP receiving on 1, 2, 4 or 8 VMs with KVM.

the fairness index was close to 1, bandwidth and CPU time were fairly shared between the different domUs.

The same experiment was executed on KVM virtual machines. KVM was used in two different configurations: paravirtualization (PV) using virtual drivers from *virtio*, and native full hardware virtualization (FV) where network drivers are emulated. The results in terms of throughput are represented on Figure 7. The first point to notice is that paravirtualization clearly outperformed full hardware virtualization with network driver emulation. For a single virtual machine, KVM with full virtualization reached only around 30% of the native Linux throughput. In this case, the bottleneck was clearly the CPU utilization. Figure 8 shows that the entire CPU core assigned to the evaluated virtual machine was used. In the case of several virtual machines, where each one was assigned a different CPU core, the throughput increased, as more CPU cores were involved.

2) Receiving performance: In this experiment, the TCP receiving throughput on 1, 2, 4 and 8 concurrent virtual machines and the corresponding processing overhead were evaluated. Figure 9 represents the results of this experiment in terms of TCP throughput per domU and aggregate throughput with Xen. We notice that the aggregate throughput decreased slightly according to the number of virtual machines on Xen 3.1. It reached only 882 Mb/s on a single domU and only 900 Mb/s on a set

of 8 concurrent domUs, which corresponds to about 96% of throughput $R_{classical(T/R)} = 938\ Mb/s$ on a classical Linux system. Efficiency $E_{throughput}$ varied between 0.96 for 8 domUs and 0.94 for a single domU. By changing the scheduler parameters (Xen 3.1a), we managed to improve the aggregate throughput to reach about 970 Mb/s on 8 virtual machines. Also, setup 3.1a improved fairness in Xen 3.1, but increased CPU consumption, leading to a trade-off between determinism and scalability. Xen 3.2 showed similar trends, leveraging throughput and removing unfairness (improving the event channel mechanism [24]). Moreover, CPU utilization decreased slightly in Xen 3.2, while being more efficient than Xen 3.1, achieving even better throughput. We conclude that important improvements have been implemented in Xen 3.2 to decrease the excessive dom0 CPU overhead.
[24]. This problem was fixed in Xen 3.2. Providing dom0 with more CPU time simply, as it was done in the 3.1a setup, allowed also to improve fairness in Xen 3.1 by giving dom0 enough time to treat all the events before the scheduler ran out of credits and started switching unnecessarily between dom0 and domUs. The resulting fair resource sharing made performance much more predictable. The measured aggregate receiving throughput in Xen 3.2 was more similar to the Xen 3.1a results with the modified scheduler parameters. The throughput increased by about 6% compared to the default 3.1 version. Figure 10 gives the CPU time distribution among the guest domains. The total CPU cost of the system varied between 140% and 150% in the default Xen 3.1 and 3.2 versions, which represents an important overhead compared to a Linux system without virtualization, where a network reception takes $C_{classical(R)} = 48\%$ with the same benchmark. We notice that on the default Xen 3.1, the efficiency in terms of throughput decreased to around $E_{throughput} = 0.91$, while the available CPU time was not entirely consumed. Also, the distribution of the CPU time consumption among the domUs followed the same unfairness pattern than for the throughput. This shows that the virtual machine scheduler on the CPU looses efficiency when stressed with networking. The fairness index decreased until only 0.46 on 8 concurrent domUs on Xen 3.1 because of the described scheduling problem.

In comparison, KVM using the virtualized network driver achieved very similar results to sending: Using a single CPU core, as in the case of one virtual machine, is not enough to achieve maximal Linux throughput as shows Figure 11. Figure 12 shows that the CPU overhead for receiving is slightly more important than for sending using virtio paravirtualization driver. This is similar to the results obtained with Xen 3.2, which nevertheless used between 10% and 30% less processing to achieve the same throughput. In the case of full virtualization, KVM's receiving mechanism is more efficient than its sending mechanism, but it still does not reach Xen's performance, needing three of the available CPU cores to achieve maximum Linux throughput.

C. Evaluation of virtual routers

To figure out the forwarding performance of virtual routers with 2 NICs, the UDP receiving throughput over virtual machines sending maximum-sized packets at maximum link speed over the virtual routers, and the TCP throughput was measured. Further the latency over virtual routers was measured. For this experiment, only Xen 3.2 was used, which was the best performing technology in the previous experiments. The results were obtained on Xen 3.2 in its default configuration and with increased weight parameter for dom0 in CPU scheduling (32 times the part attributed to a domU). We call this setup *Xen 3.2a* in the following.

1) Forwarding performance: To determine the performance of virtual routers, UDP traffic was generated with either maximum- (1500 bytes) or minimum- (64 bytes) sized packets over one or several virtual routers (from 1 to 8) sharing a single physical machine. All the flows were sent at the maximum rate from distinct physical machines to avoid bias. Then, end-to-end TCP throughput was also evaluated.
Figures 13 show the obtained UDP bit rate with maximum-sized packets and the TCP throughput. The corresponding CPU cost is represented in Figure 14. Table II details the UDP packet rates and the loss rates per domU with maximum- and minimum-sized packets. With UDP, the

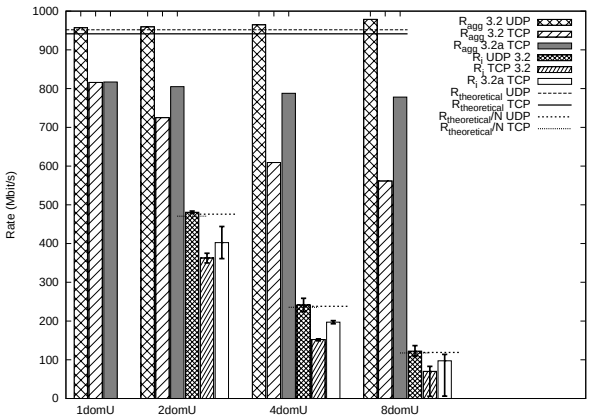

Figure 13. Receiver side throughput over 1, 2, 4 or 8 virtual routers with Xen 3.2 forwarding 1500-byte packets.

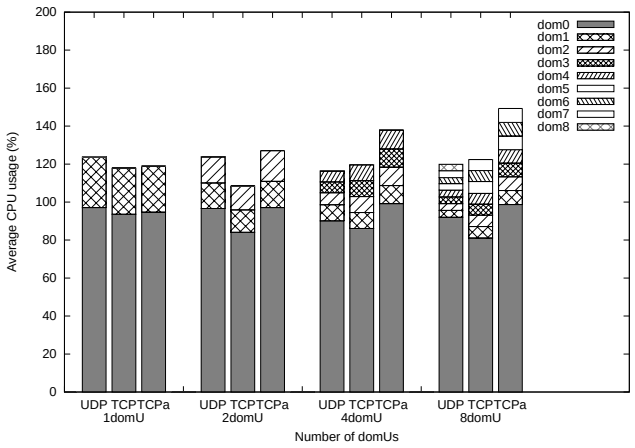

Figure 14. CPU cost of 1, 2, 4 or 8 virtual routers with Xen 3.2 forwarding 1500-byte packets.

packet loss rate with maximum-sized packets on each virtual machine corresponded to $1 - R_{theoretical}/(N \times R_{theoretical})$.
The bandwidth was fairly shared between the virtual routers. The results showed efficiency, the throughput corresponded to the value obtained on a classical Linux router $R_{classical(F)} = 957\ Mb/s$. The aggregate UDP throughput was in some cases a bit higher than the theoretical value due to little variation in the start times of the different flows. Resource sharing was fair: in this case, performance is predictable. With maximum-sized packets, dom0 used an entire CPU, forwarding at maximum rate with UDP and only 80% of the maximum throughput with TCP, having an efficiency of $E_{throughput} = 0.80$. With minimum-sized packets on 4 or 8 virtual routers, dom0 became overloaded, not being able to forward on all virtual routers anymore.
Regarding the processing of the router, dom0 used much more CPU resources than the domUs, compared to the simple sending or receiving scenario. This is due to the fact that it has to forward the traffic of twice as many virtual interfaces than before (16 in the case of 8 virtual routers). In the case of UDP and TCP with the modified scheduling parameters (named TCPa), the usage of domU's CPU was pushed to its maximum. It used an entire CPU. Hence, the TCP throughput was obviously limited by dom0's processing limitation. In the case of TCP forwarding with the default CPU scheduling parameters, dom0 did not get enough processing resources, especially when the number of virtual routers increased, in turn increasing the number of virtual interfaces to treat, which caused the throughput to decrease. With the important overload with 8 virtual routers, the last domU was not even able to forward packets anymore, and thus used no CPU.

	1500 byte packets		64 byte packets	
	pps/VM	loss/VM	pps/VM	loss/VM
1 VM	81284	0 %	109685	60 %
2 VM	40841	50 %	12052	96 %
4 VM	20486	75 %	/	/
8 VM	10393	87 %	/	/
Linux	81277	0.00 %	356494	0.06 %

Table II. Average UDP packet-forwarding rate and loss rate per domU.

With TCP, the throughput throttled down, especially with an increasing number of virtual routers. This might be related to an increasing latency, discussed in the next paragraph.

2) Virtual router latency: In this experiment, the latency on one virtual router (VR) was measured, while concurrent virtual routers (1, 3 or 7) sharing the same physical machine were either idle or stressed forwarding maximum-rate TCP flows. Table III represents the results in both cases. The latency over a virtual router sharing

Latency(ms)	Linux	1 VR	2 VR	4 VR	8 VR
idle	0.084	0.147	0.150	0.147	0.154
stressed			0.888	1.376	3.8515

Table III. Latency over one VR among 1, 2, 4 or 8 VRs idle or stressed with TCP forwarding.

the physical machine with other idle virtual routers was about 0.150 ms, no matter the number of virtual routers, which was almost the double of the latency on a classical Linux router $L_{classical(F)} = 0.084ms$. In the case of a stressed system, the latency on the considered virtual router increased with the number of concurrent virtual routers forwarding maximum throughput TCP flows. The average latency reached nearly 4 ms on a virtual router sharing the physical machines with 7 virtual routers forwarding TCP flows. The more virtual machines asking for the scheduler, the more the latency on the virtual router increased. For TCP, this can lead to timeouts. In this case, frequent retransmissions throttle the throughput down.

V. Related work

The performance of virtual packet transmission in Xen is a crucial subject and was treated in several papers. Table IV lists the main network issues of the successive Xen versions, discussed in this section. On Xen 2.0, Menon et al. [25] measured default transmit and receive TCP throughput on a domU below 1000 Mb/s using four 1-Gigabit NICs, which is much less than the throughput we measured on Xen 3.2 (940 Mb/s on a single NIC). The

	Key problem	Key improvement
Xen 2.0	End-host throughput (≅25% of CL)[25]	VNIC offloading I/O channel modif.
Xen 3.0	Forwarding rate <25% of CL, CPU bottleneck[26]	SMP load balancing CPU scheduler
Xen 3.1	Unfairness	Event channel modif.[24]
Xen 3.2	Small packets I/O (memory bottleneck)[27]	
KVM	CPU overhead	virtio virtual driver

Table IV. Comparison of Xen versions and KVM.

authors improved Xen's networking performance by modifying the virtual network interfaces to include hardware NIC features and optimize the I/O channel between dom0 and domU. After optimization, they obtained results for transmissions similar to what we obtained on Xen 3.2: 3310 Mb/s on four NICs which corresponds to around 830 Mb/s per NIC, but still less for receptions (only 970 Mb/s on four NICs) which corresponds to only 26% of Xen 3.2's receiving throughput.

A study about scheduling on Xen 3 unstable (changeset 15080) [24] confirms our result, with Xen 3.1, of the unfair bandwidth distribution among the domUs in the default configuration, with the credit scheduler [28]. The authors measured an aggregate throughput of less then 800 Mb/s on 7 domUs, varying from less than 25 Mb/s to around 195 Mb/s per domU. They proposed event-channel improvements which enhanced the fairness in the sharing of the bandwidth between the virtual machines, to vary only about ± 25 Mb/s on the different domUs. We noticed this improved fairness in the new Xen 3.2 version.

McIlroy and Sventek in [29] proposed the virtual router concept to separate traffic into virtual forwarding planes. In this case, data-plane virtualization offers QoS with individual packet forwarding. An evaluation of Xen 3.0 for routers [30] shows that the aggregate forwarding throughput on two to six domUs reaches less than 25% of what is achievable on classical Linux for 64-byte frames.

In a recent paper [27], the authors showed that memory access time is the main system bottleneck with Xen. They finally proposed a system to map forwarding paths to CPU cores so that as many packets as possible can be kept in the closest CPU cache to limit costly memory accesses [31]. This system is an interesting step to improve the performance of forwarding in software, but there is no real data-plane virtualization as packet data remains in dom0 and only routing is performed in domU. It is an interesting trade-off between performance and the level of virtualization in software virtual routers.

In parallel to these evaluations and optimizations of Xen, KVM appeared as a new full virtualization solution. To leverage it's low I/O performance, due to the emulation of the drivers, paravirtualization was also included within the *virtio* drivers. Nevertheless, as our results showed, Xen outperforms KVM with virtio when it comes to network virtualization. While virtio is a sort of trade-off between genericity and performance, Xen's virtual drivers are tailored to the Xen technology and, as described before, have gone through several optimizations. This can explain its better performance to date.

Another very recent software network virtualization solution is Crossbow [32], which appeared on OpenSolaris. Crossbow allows to build virtual NICs using so-called virtualization lanes. It works with hardware-virtualization-enabled NICs, and packets are directly classified on layer 2 into the right virtualization lane. In the absence of hardware virtualization of the NICs, virtual lanes are implemented entirely in software, but giving no guarantees on the fairness in the resources sharing. In both cases, Crossbow is a solution to virtualize the data plane and is an interesting technology to study as future work.

VI. Conclusion and perspectives

In this article, we evaluated the network performance of virtual end-hosts and a virtual router we designed and implemented with Xen. The results were also compared to KVM virtualization technology. Virtualization mechanisms like additional copy and I/O scheduling of virtual machines sharing the physical devices are costly. Nevertheless, our results show that virtualizing the data plane by forwarding packets inside the virtual machines becomes a more and more promising approach with the successive versions of Xen, improving those mechanisms. We show that end-host throughput improved in Xen 3.2 compared to 3.1. Also, previous fairness issues have been corrected. Xen, with its virtual network drivers, outperforms KVM, offering higher network transmission rates, as it requires significantly less processing power. Virtual routers built with Xen act similarly to classical Linux routers, while forwarding big packets. The evaluation of successive Xen versions shows that the technology is constantly improving and gives a promising perspective to network virtualization. However, our results showed an important overhead, due to additional processing needed, since packets travel through virtual drivers. For this reason, the best solution to leverage performance would probably be to use virtualization-enabled network interfaces. Our next goal is to evaluate the impact of executing large-scale applications on concurrent virtual network topologies with virtual routers performing network control in terms of routing and bandwidth sharing.

Acknowledgement

This work has been funded by the French ministry of Education and Research and the ANR, INRIA, and CNRS, via ACI GRID's Grid'5000 project, ANR HIPCAL grant and INRIA Aladdin ADT. We thank Olivier Mornard for his assistance in setting up the testbed.

References

[1] F. Anhalt and P. Vicat-Blanc Primet, "Analysis and experimental evaluation of data plane virtualization with Xen," in *ICNS 09 : International Conference on Networking and Services*, (Valencia, Spain), Apr. 2009.

[2] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," in *SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles*, (New York, NY, USA), p. 121, ACM, 1973.

[3] 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 *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, (New York, NY, USA), pp. 164–177, ACM, 2003.

[4] J. Sugerman, G. Venkitachalam, and B.-H. Lim, "Virtualizing I/O devices on VMware workstation's hosted virtual machine monitor," in *Proc. 2001 Usenix Annual Technical Conference*, pp. 1–14, Usenix Assoc., 2001.

[5] J. Varia, "Cloud architectures," in *NSF Data-Intensive Scalable Computing Workshop*, 2008.

[6] http://www.3tera.com/.

[7] http://www.microsoft.com/azure.

[8] N. Feamster, L. Gao, and J. Rexford, "How to lease the internet in your spare time," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 1, pp. 61–64, 2007.

[9] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In vini veritas: realistic and controlled network experimentation," in *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 3–14, ACM, 2006.

[10] S. Bhatia, M. Motiwala, W. Muhlbauer, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford, "Hosting Virtual Networks on Commodity Hardware," tech. rep., Georgia Tech Computer Science Technical Report GT-CS-07-10, Jan 2008.

[11] "Linux VServers Project." http://linux-vserver.org.

[12] F. Bellard, "Qemu, a fast and portable dynamic translator," in *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, (Berkeley, CA, USA), pp. 41–41, USENIX Association, 2005.

[13] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel virtualization technology: Hardware support for efficient processor virtualization," tech. rep., Intel Technology Journal, 2006.

[14] "AMD-V Nested Paging," tech. rep., Advance Micro Devices, Inc., July 2008.

[15] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "kvm: the Linux Virtual Machine Monitor," in *Linux Symposium*, 2007.

[16] R. Russell, "virtio: towards a de-facto standard for virtual i/o devices," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 95–103, 2008.

[17] D. Chisnall, *The Definitive Guide to the Xen Hypervisor*. Prentice Hall, 2007.

[18] J. R. Santos, G. J. Janakiraman, Y. Turner, and I. Pratt, "Netchannel 2: Optimizing Network Performance," in *Xen summit*, 2007.

[19] D. Laor, "KVM Para-Virtualized Guest Drivers." KVM Forum 2007, Aug. 2007.

[20] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," *ACM Transaction on Computer Systems*, Sept. 1984.

[21] F. Cappello, P. Primet et al., "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pp. 99–106, IEEE Computer Society, 2005.

[22] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "iperf : testing the limits of your network," *http://dast.nlanr.net/Projects/Iperf*.

[23] http://www.netperf.org/netperf/.

[24] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling i/o in virtual machine monitors," in *VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pp. 1–10, ACM, 2008.

[25] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in xen," in *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pp. 2–2, USENIX Association, 2006.

[26] F. Anhalt and P. Vicat-Blanc Primet, "Analysis and evaluation of a XEN based virtual router," Tech. Rep. 6658, INRIA Rhône Alpes, Sep 2008.

[27] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Fairness issues in software virtual routers," in *PRESTO '08*, pp. 33–38, ACM, 2008.

[28] http://wiki.xensource.com/xenwiki/ CreditScheduler.

[29] R. McIlroy and J. Sventek, "Resource virtualisation of network routers," in *HPSR 06: 2006 Workshop on High Performance Switching and Routing*, June 2006.

[30] N. Egi et al., “Evaluating Xen for Router Virtualization,” in *ICCCN*, pp. 1256–1261, 2007.

[31] N. Egi, A. Greenhalgh, M. Hoerdt, 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.

[32] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaied, “Crossbow: from hardware virtualized nics to virtualized networks,” in *VISA ’09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, (New York, NY, USA), pp. 53–62, ACM, 2009.