Evaluating Virtual Router Performance for the Future Internet

Diogo Menezes Ferrazani Mattos, Carlo Fragni,
Marcelo Duffles Donato Moreira, Lyno Henrique Gonçalves Ferraz,
Luís Henrique Maciel Kosmalski Costa, Otto Carlos Muniz Bandeira Duarte
Grupo de Teleinformática e Automação - PEE/COPPE - DEL/POLI
Universidade Federal do Rio de Janeiro - Rio de Janeiro, Brazil
Email:{menezes,carlo,marcelo,lyno,luish,otto}@gta.ufrj.br

Abstract

The Internet is ossified. It is difficult to create new services or innovate in the core of the network. In order to address this issue, we need a new Internet model to council the existing protocols with new ones. Virtualization is a technique that provides this capability to the Future Internet elements. In this paper, we evaluate the performance of three representative virtualization tools, Xen, VMware, and OpenVZ, for router virtualization. We conduct experiments with benchmarking tools to measure the overhead introduced by virtualization in memory, processor, network, and disk performance of virtual routers running on commodity hardware. Our results show that Xen is the tool that best fits a virtual router needs, because it introduces less overhead on network performance. We also evaluate effect of the increasing number of virtual machines on Xen network virtualization mechanism, which is proved to be fair, but needs further enhancement when multiple virtual machines forward traffic simultaneously.

1 Introduction

The Internet has currently more than 1 billion users. The Internet success is mainly based on two pillars, the end-to-end data transfer service and the TCP/IP stack [9]. The intelligence of the network is placed at the end systems, while the network core is simple and transparent. Even if those are reasons for the Internet being a huge success, they also ossify the current Internet. The TCP/IP model has some structural issues, like scalability, mobility, management, and security [4], that are difficult to solve. Today, there is a thought that a new Internet architecture must provide the flexibility and the support for innovation in the network core. The Future Internet [2, 4, 9] models are divided into two types of approach, monist and pluralist. The monist approach proposes a monolithic network capable of dealing with the requirements for providing a flexible network. On the other hand, the pluralist approach models the Internet as a network that has multiple stacks running simultaneously. A way to provide a pluralist network is virtualizing the network resources, like routers. Hence, router virtualization is a key concept for the pluralist Future Internet architecture.

A virtual router is a logical router that shares its underlying substrate with other routers. One of the advantages of virtualizing is the isolation of virtual environments. When we virtualize routers, we can have multiple network stacks over the same physical router [5]. The virtual router concept brings the flexibility needed by the new Future Internet architecture and keeps the new model compatible with the old one, as one of the virtual routers can run the TCP/IP stack while the other virtual routers run other layer 3 protocols. One powerful virtualizing technique is hardware virtualization, because it allows multiple router operating systems over the same hardware. Nevertheless, it introduces processing overhead because it must control the access of the different OSes to the hardware. In this paper, we devise the overhead and isolation properties of different hardware virtualization techniques. For that matter, we introduce a methodology to compare the different tools.

The main difficult to evaluate a virtualization tool is that common benchmarking tools are distorted by the time relation in the virtual environment [3]. Within a virtualized environment, guest operation system does not have access to physical time sources and timer interrupts. Virtualized time system is commonly implemented through software emulation of the real time devices. Thus, a difference between virtualized time and real world time often exists. Common benchmark tools believe on guest OS time and get affected by the virtualized time distortion.

In this paper, we study three representative virtualization tools, VMware ESX [13], Xen [1], and OpenVZ [11]. Each one implements a different virtualization technique. We propose a methodology to evaluate virtualized systems that is independent of virtual environment time distortion.

We use the proposed methodology to evaluate some virtualization tools, and we conclude that Xen is the tool that best fits a virtual router needs. Thus, we also analyze the performance of a Xen virtualized router.

This paper is organized as follows. Section II analyzes representative virtualization tools and their main characteristics. Section III presents our proposed methodology and experimental setup. Section IV discusses the virtualization tool evaluation results, and Section V investigates our Xen virtual router evaluation results. We conclude and introduce our future works in Section VI.

2 Virtualization Tools

Virtualization is the technology that allows sharing a physical hardware among multiple systems. Each virtualized system is isolated from others, and is not aware of the existence of other systems sharing the same hardware. In order to achieve it, it is necessary to introduce a software layer called hypervisor. We present some of the most important hypervisors available today: VMware, Xen, and OpenVZ.

2.1 VMware

Hereafter, we consider the VMware ESX Server product, which is a datacenter virtualization platform that implements the full virtualization technique, i.e., the guest operating system does not need to have anything changed to run in a virtual environment [13]. VMware guarantees virtual machine isolation and resource sharing fairness based on resource-allocation policies set by the system administrator. Resources are allocated and re-allocated to virtual machines on demand. CPU virtualization is done by setting a virtual CPU for each virtual machine. The virtual machine does not realize that it is running over a virtual CPU, because virtual CPUs provide their own registers and control structures. VMware combines two CPU virtualization modes: direct execution and CPU emulation. In the direct execution mode, instructions from the user-space of virtual machine are executed directly on the physical CPU. On the other hand, guest operating system instructions, like system calls, traps, interrupts, and other events, are trapped to VMM that emulates the instruction execution, adding a varying amount of virtualization costs. VMware memory virtualization approach is to create a new level of memory address translation. It is done by providing each guest operating system a virtual page table that is not visible to the memory-management unit (MMU) [1]. Within a VMware virtualization environment, the guest operating system accesses a virtual memory space that is internal of the virtual machine. In order to accomplish network I/O virtualization, VMware implements the *vmxnet* [13] device driver, which is an abstraction of the underlying physical device. When an application wants to send data over the network, the *vmxnet* device driver is called and the I/O request is intercepted by the VMM, which calls the specific device driver on the physical machine.

2.2 Xen

Xen is an open-source hypervisor proposed to run on commodity hardware platforms [1]. Xen implements the paravirtualization technique, which enhances guest system performance by changing its behavior to call the hypervisor when necessary, obviating the need of binary translation of system instructions. Xen virtualizes the processor by assigning virtual CPUs (VCPUs) to the virtual machines. Xen hypervisor implements a CPU scheduler that dynamically maps a physical CPU to each VCPU under a certain period, based on a scheduling algorithm. Memory virtualization in Xen is currently done statically. The RAM memory is divided among virtual machines, with each machine receiving a fixed amount of memory space, specified at the time of its creation. In addition, device drivers are kept in an isolated virtual machine, called Domain 0, which access I/O devices directly by using its native device drivers and also performs I/O operations on behalf of virtual machines. By their turn, virtual machines employ virtual I/O devices controlled by virtual drivers to request Domain 0 for device access [10].

2.3 OpenVZ

OpenVZ is an open-source operating system-level virtualization tool [8]. OpenVZ allows to have multiple isolated execution environments over a single operating system kernel. Each execution environment is called a *Virtual Private Server* (VPS). A VPS looks like a physical server; it has its own processes, users, files, IP addresses, system configuration, and provides full root shell access. OpenVZ claims to be the virtualization tool which introduces less overhead, because each VPS shares the same operating system kernel. On the other hand, OpenVZ is less flexible than other virtualization tools, like VMware or Xen, because OpenVZ execution environments have to be a Linux distribution, based on the same operating system kernel of the physical server. For processor virtualization, OpenVZ implements a two-level CPU scheduler. On the first level, the OpenVZ virtualization layer decides which VPS will execute for each time slice, taking into account the VPS CPU priority. On the level-2 scheduler, which runs inside the VPS, the standard Linux scheduler defines which process will execute for each time slice, taking into account the standard process priority parameters. OpenVZ allows VPSs to directly access the memory and the memory amount dedicated for each VPS can be dynamically changed by modifying the virtual

memory space of each VPS. OpenVZ kernel manages VPSs memory space to keep in physical memory a block of the virtual memory corresponding to the running VPS. OpenVZ default network virtualization mechanism creates a virtual network interface for a VPS and assigns an IP address to it in the host system. When a packet arrives to the host system with an IP address assigned to a VPS, the host system routes the packet to the corresponding VPS. This approach of network virtualization allows VPS packets to be received and sent using the host system routing module. This simplifies network virtualization, but introduces a new hop in packets path.

3 Hypervisors Evaluation

A router purpose is to forward packetsIt receives a packet, check checks its forwarding table for the packet's destination, and forwards it to the correct network interface. Routers main resources are networking, memory, and processing. A router must be able to quickly access its networking interface, get its packets, and store them in memory. After that, the router has to access memory, get the packets, look for their destinations on a routing table, and then send the packets to the correct networking interface. Taking these actions into account, this paper evaluates some well known hypervisors. It evaluates networking, memory access, and processing capabilities, in order to find which hypervisor has the most satisfactory performance for router virtualization. Although hard disk access performance is not critical for router virtualization, we also analyze the hard disk virtualization performance. We analyze three hypervisors: Xen, OpenVZ, and a proprietary virtualization solution (PS) [1]. In an effort to show a reference value, we present the tests results for a native Linux environment. Hence, the virtualized scenarios results should be equal or worse than the reference values.

3.1 Evaluation Methodology

Evaluating the virtualization overhead is not a trivial task. Whitepapers from VMware [12] and XenSource [14] compare these two hypervisors performance reaching different conclusions. Their approach is to run standard benchmarking tools within a virtual environment and compare the results with those obtained in running the benchmarking tool in the native operating system. One disadvantage of this approach is that it relies on the time measurement taken inside the virtual environment. The accuracy of the virtualized operating system time is not assured by the hypervisors [3].

[1]We do not mention the proprietary virtualization tool name because of its License Agreement Limitations. From now, we refer it as Proprietary Solution or PS.

In this work, we use a different virtualization benchmarking methodology based on an external time measuring mechanism, in order to evaluate the hypervisors performance. The methodology consists of running standard benchmarking tools within the virtualized system, but instead of relying on benchmark time information, the time information is provided by an external mechanism. This mechanism is implemented over a point-to-point network link between the target machine and the external time-measuring machine. Over this link, the target machine sends an ICMP Echo request to the time-measuring machine. At this moment, it starts a timer. After the target machine completes its job, another ICMP Echo request packet is sent, and the time-measuring machine stops the timer. As a result, the responsibility of reporting the time difference relies on a non-virtualized system.

Based on the proposed methodology, we evaluate the overhead of the different hypervisors in terms of processor usage, disk access performance and memory access performance. Nevertheless, this methodology is not applicable to identify the overhead on networking I/O, an important router resource. Networking I/O performance measures are not just based on time. A traffic generator sends packets from the tested system to a real machine. In the real machine, we measure the bit rate achieved by the tested system. In this manner, as the amount of bits and the time difference are measured by the real machine, there is no time deviation caused by virtualization.

3.2 Scenario

A physical server, a traffic generator machine and a time measuring machine form our testbed. The physical server supports the virtualized environments. It is a HP Proliant DL380 Generation 5 server equipped with two quad core Intel Xeon processors (2.83 GHz each), 10 GB system memory, and integrated 2-port Broadcom Nextreme II Gigabit network interface. The traffic generator machine is responsible for sending packets to the system under test. The traffic generator machine is a desktop equipped with an Intel motherboard, Intel 2.4GHz Core 2 Quad processor, 4GB of system memory, and an integrated Intel PCI-Express gigabit network interface. The role of the Time Measuring Machine is to measure the time taken by a system under test to each benchmark. It is a desktop computer equipped with an Intel motherboard, an Intel 2.66GHz Core 2 Duo processor, 2 GB of system memory, and an integrated Intel PCI-Express gigabit network interface.

3.3 Evaluation

3.3.1 Processor Performance

In order to evaluate CPU virtualization overhead, we perform CPU-intensive workloads using the *Super Pi* benchmark. It is based on the Gauss-Legendre algorithm to compute the Pi value. The Gauss-Legendre algorithm is iterative and does many arithmetic operations. The arithmetic operations are mainly sum, division, square root, potentiating, subtraction and multiplication. We execute the Super Pi benchmark to compute the Pi value with 2^{22} digits.

Fig. 1(a) indicates that all virtualization tool introduce an overhead in processor usage. The smallest overhead is introduced by Xen Hypervisor, which implements the paravirtualization technique. The proprietary solution (PS) is the virtualization tool that introduces the bigger overhead in processor usage. It is because PS implements full virtualization technique, where each instruction that generates a fault is redirected to the hypervisor. The hypervisor simulates its execution and returns the result to the application on the virtual machine. This process causes a bigger overhead than paravirtualization, where an instruction that would generate a fault is modified to execute over the hypervisor. OpenVZ performs similar to Xen. A process of a virtual machine, however, is subject to a two level scheduler before being executed. Because of the two level scheduler, in OpenVZ, a process of a virtual machine takes more time to be executed than in native Linux.

3.3.2 Memory Performance

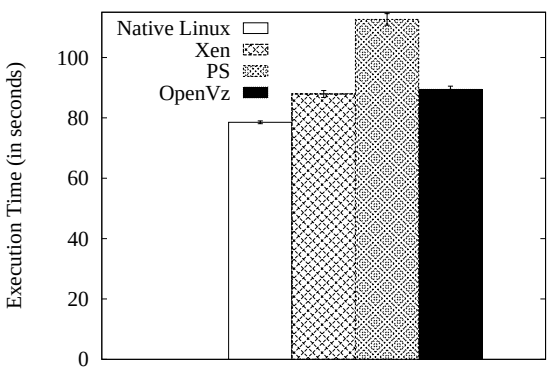
Memory access performance has significant influence in the whole router performance [6]. In order to evaluate the overhead caused by the virtualization layer on virtual router memory access, we developed a benchmarking tool, called *MASR* (Memory Allocation, Set and Read). MASR benchmarks memory by allocating 2GB of memory, setting sequentially memory positions to a fixed value and after reading each one. MASR was developed for benchmarking memory with a deterministic number of operations, independent of the performance of the computer.

Fig. 1(b) shows that OpenVZ is the virtualization layer that introduces less overhead in memory access. OpenVZ accesses directly the memory. The OpenVZ virtual environment accesses memory as an application accesses virtual memory pages on a native operating system. As a consequence, OpenVZ performs similar to native Linux. On the other hand, Xen Hypervisor statically allocates memory areas to each virtual environment. The Xen virtual machine access its memory space as if it was the machines total memory, however, it is just a portion of it. Nevertheless, the virtual machine directly accesses memory, because, as it paravirtualized, it is aware of its address space and it handles the physical address of its portion of the memory. The Proprietary Solution has worse memory access performance than the others. PS implements the memory translation, and it is also full virtualized. As it is full virtualized, a virtual machine is unaware that it is over a hypervisor, and when it tries to access memory, this set of instructions is trapped by the hypervisor. The hypervisor executes the memory access instruction, and then, gives the result to the virtual machine. It introduces a larger overhead than the other hypervisors do.

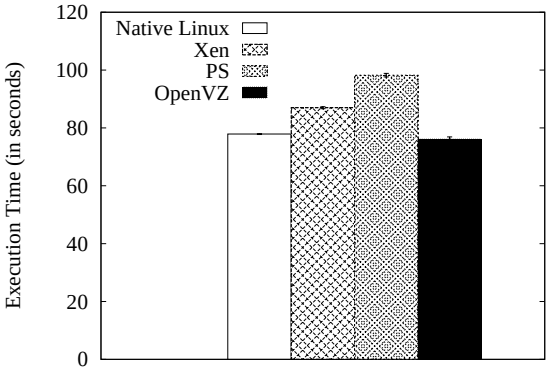
3.3.3 Hard Disk Performance

We also analyze the hard disk access performance, expanding our hypervisors evaluation to a wider virtualization scenario than only router virtualization. We aim at measuring the virtualization overhead of disk writing and reading tasks, and compare the overhead obtained by each hypervisor. We use two different benchmarking tools. The first one is Bonnie++, an open-source disk benchmarking tool that simulates some file operations, such as creating, reading and deleting of small files, and also tests the performance of accessing different regions of the hard disk, reading sectors at the beginning, middle, and end of the hard disk. The second tool, developed by the authors, is ZFG (Zero File Generator), which was designed to run within virtualized systems. ZFG benchmarks the hard disk continuous writing speed by writing ten times a 2 GB binary file filled with zeros. The main feature of this tool is that it writes some dump information on disk during the time between two rounds. This is important to guarantee that the time elapsed in each round is actually the time spent by the virtual machine to write the file on disk. If this is not guaranteed, the measured time can be the time that a virtual machine writes the file on a buffer, whose content will be later written on physical disk by the hypervisor.

OpenVZ implements hard disk access using the quota method provided by native Linux. The difference between OpenVZ and native Linux hard disk access is just that OpenVZ introduces a scheduler to decide which virtual environment should access the hard disk in each turn. As a consequence, OpenVZ performs near native Linux, as shown in Fig. 2. On the other hand, Xen and the PS perform poorly, especially in ZFG tests. Because Xen and PS implement the hard disk resource as a virtual abstraction of the physical hard disk, virtual machines access an interface that is believed to be the real hard disk. Writing and reading requests are trapped by the hypervisors and properly handled. Therefore, this procedure introduces a delay on disk access caused by additional processing overhead and memory page copies.

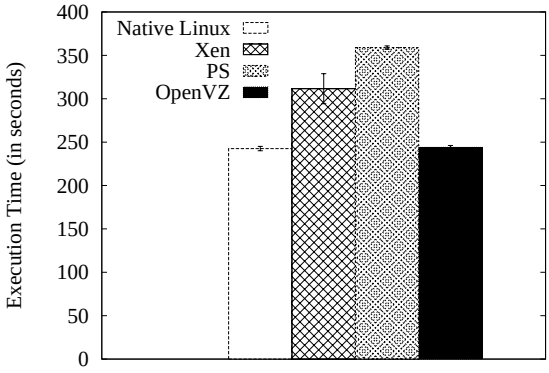


(a) Mean execution time for Super Pi test.

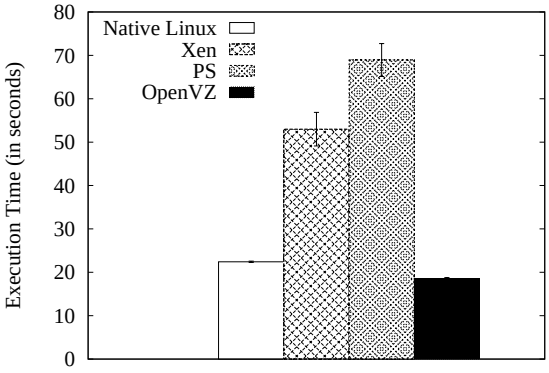


(b) Mean execution time for MASR.

Figure 1. Mean execution time for CPU and memory access benchmarks.



(a) Bonnie++ test.



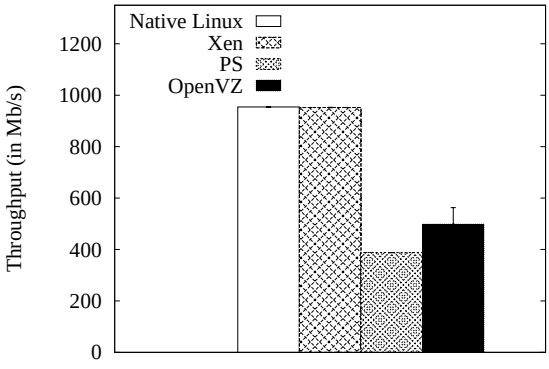
(b) ZFG test.

Figure 2. Hard disk performance evaluation results using Bonnie++ and ZFG benchmarking tools. Smaller values are better.

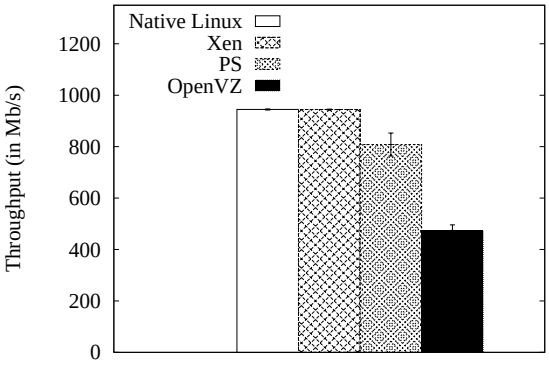
3.3.4 Networking Performance

A virtual router must be able to efficiently receive packets and sent them to the correct output interface. Hence, the virtualization layer cannot degrade networking performance. In order to evaluate the virtualization overhead on networking performance, we adopted the Iperf tool for measuring the available bandwidth from an external host to the system under test, and vice-versa. Iperf is an open-source networking benchmarking tool that uses both unidirectional and bidirectional data flows over TCP or UDP. Iperf has several parameters that can be configured, such as packet size, intended bandwidth, and test duration. In our experiments, the network traffic is an UDP flow with data packets of 1472 bytes, which represents the maximum size for a packet payload to avoid fragmentation over an Ethernet domain, whose most common MTU is 1500 bytes.

The networking evaluation results are shown in Fig. 4. First of all, it is important to highlight that, for both network traffic transmission (Fig. 3(a)) and reception (Fig. 3(b)) scenarios the native Linux system achieves a network performance near the nominal gigabit Ethernet transmission bit rate. An also important result is that Xen virtual machine achieves the same transmission and reception rate of the native Linux. This shows that Xen networking virtualization mechanism is fast enough in sending/receiving packets to/from virtual machines to not be a bottleneck in the evaluated scenario. On the other hand, the other virtualization tools have not succeeded on virtualizing the network device. There are some implementation issues that may be a bottle-



(a) Network traffic transmission test.



(b) Network traffic reception test.

Figure 3. Average throughput achieved by the evaluated systems while transmitting/receiving unidirectional UDP flows generated by the used network benchmarking tool (Iperf).

neck to the rate that the network virtualization mechanism can handle network I/O operations, such as multiple memory page copies, the time wasted to schedule the virtual environments and the host system, or small input/output buffer sizes.

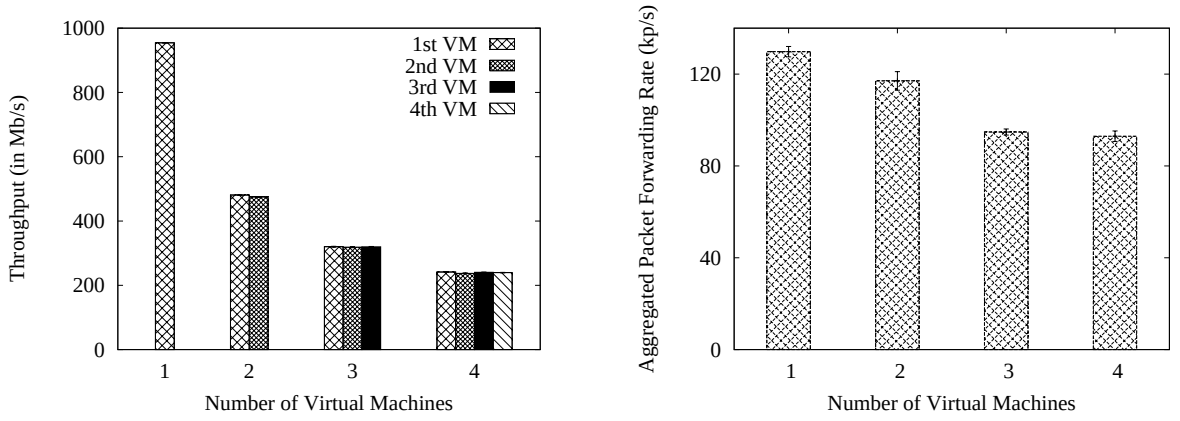
4 Xen Virtual Router Evaluation

In the following experiments we aim at evaluating the scalability of the Xen hypervisor. First, we evaluate the impact of increasing the routing table in a virtual router. After that, we measure the degradation of the packet forwarding rate as we increase the number of virtual routers running over the same hypervisor.

The routing table size may influence the packet forwarding rate, because a large table can increase the per-packet delay on routing table lookup. The Linux kernel, however, implements the routing table as a hash table. As a consequence, there is no linear search over the routing table and the next hop in packet route is discovered in a constant time, regardless of the routing table size. To confirm this fact, we conduct experiments in which the virtual router is configured with routing tables with 1,000, 10,000, and 100,000 routes. The maximum routing table size is fixed in 100,000 routes because this is a common size of a routing table of BGP border router [7]. In all scenarios, the packet forwarding rate does not change due to the addition of new entries in the routing table. Hence, Xen virtual router scales to the routing table size.

We perform an evaluation of the fairness in Xen network virtualization mechanism, in order to evaluate the Xen scalability for multiple virtual machines sharing a single network interface card. In a first experiment, whose results are shown in Fig. 4(a), we instantiate from one to four virtual machines over the Xen hypervisor and set them to generate a network traffic to an external computer. The generated traffic is an UDP flow, with maximum Ethernet frame size. In this scenario, using just one virtual machine, we achieve the theoretical maximum Gigabit Ethernet bandwidth, which is 969 Mb/s. The results show that, as the number of virtual machines increases, each virtual machine achieves a throughput that is inversely proportional to the number of virtual machines. It is also important to observe that the aggregated throughput is equally shared by all virtual machines running on the same physical substrate. The conclusion of this first scalability test is that Xen network virtualization fairly scales to multiple virtual machines.

The second scalability experiment measures the degradation of the packet forwarding rate with the number of virtual routers. In this experiment, the sent packet rate is 130 kp/s and we measure the packet forwarding rate as the rate of packets that reach the external machine after being forwarded by the virtual router. Fig. 4(b) shows that, as the number of virtual routers running at the same time increases, the traffic forwarding performance degrades. It is a consequence of the lack of CPU. The CPU scheduler, which is responsible for sharing the processor among all virtual routers, has to quickly switch context between a running virtual router and a blocked one. As this context switching becomes more frequent, there is a processing time lost for doing this task, instead of serving virtual router processors. Hence, each virtual router is requesting to forward its own traffic, but the available resources are being used by the other virtual routers and for context switching. Hence, the



(a) Network traffic transmission test.

(b) Network traffic forwarding test.

Figure 4. Xen scalability tests with multiple virtual machines sending/forwarding network traffic to the external computer.

virtual routers drop packets, reducing the aggregated packet forwarding rate. Our results show a 30 percent reduction on packet forwarding rate, when four virtual routers are running over the same hypervisor.

5 Conclusion

In this paper, we have investigated what is the virtualization tool that best fits the requirements of a virtual router for the Future Internet pluralist approach. In order to do this, we propose a new methodology for evaluating the overhead introduced by the virtualization layer. We performed disk, memory, and networking tests. Our results show that OpenVZ is the virtualization tool that introduces less overhead over processor, disk and memory usage. OpenVZ performs almost as well as native Linux. On the other hand, Xen performs better for networking and has a small overhead on processor and memory usage. Finally, the Proprietary Solution, which provides a full virtualized environment, is more flexible, but introduces bigger overheads over all resources usage.

Xen provides multiple virtual environments, which are complete OS environments, independent of each one and of the guest OS. Xen virtual environment performance is compatible with the virtual router requirements. Hence, Xen is the virtualization tool that best fits for the requirements for router virtualization in a Future Internet architecture.

We also analyze the performance and the scalability of router virtualization over Xen. It scales well for multiple virtual routers, running simultaneously, and for increasing routing table size. In the first scenario, as the number of virtual routers over a single physical router increases, the throughput of each router is equally reduced. Nevertheless, the aggregated throughput is maintained and is near the theoretical Ethernet maximum throughput. In the second scenario, we increase the routing table size and the packet forward rate remains the same. It shows that the processor and memory access overhead, introduced by Xen, a virtual router.

After all, we conclude that Xen is the virtualization tool that best fits the router virtualization requirements. Xen Hypervisor, however, must be enhanced to support a virtual router within a production network, as we observed that the packet forwarding rate is 30 percent reduced when running multiple virtual machines over the same hypervisor. Therefore, our future work focus on developing Xen virtual router enhancement and evaluate new virtualization technologies for improving virtual routers performance.

Acknowledgment

This work was supported by CNPq, CAPES, FINEP, FUNTTEL, and FAPERJ.

References

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles - SOSP03*, October 2003.

[2] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the Internet: the end-to-end argu-

ments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.

[3] Siming Chen, Mingfa Zhu, and Limin Xiao. Implementation of virtual time system for the distributed virtual machine monitor. *IEEE/ISECS International Colloquium on Computing, Communication, Control, and Management*, August 2009.

[4] Karen Sollins David Clark, Robert Braden. New Arch: Future generation internet architecture. Technical report, MIT Computer Science and AI Lab, August 2004.

[5] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Laurent Mathy, and Tim Schooley. Evaluating xen for router virtualization. In *International Workshop on Performance Modeling and Evaluation (PMECT)*, August 2007.

[6] Norbert Egi, Mickael Hoerdt, Adam Greenhalgh, Mark Handley, Felipe Huici, and Laurent Mathy. Towards high performance virtual routers on commodity hardware. In *In Proceedings of ACM CoNEXT 2008*, 2008.

[7] Geoff Huston. *BGP Reports*, may 2010. http://bgp.potaroo.net/ipv4-stats/prefixes_adv_pool.txt.

[8] Kirill Kolyshkin. *Virtualization in Linux*, 2006. http://download.openvz.org/doc/openvz-intro.pdf.

[9] Luís Henrique M. K. Costa e Otto Carlos M. B. Duarte Marcelo D. D. Moreira, Natalia C. Fernandes. Internet do Futuro: Um Novo Horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores - SBRC'2009*, pages 1–59, 2009.

[10] Aravind Menon, Alan L. Cox, and Willy Zwaenepoel. Optimizing network virtualization in Xen. In *USENIX Annual Technical Conference*, pages 15–28, May 2006.

[11] *OpenVZ User's Guide*. SWsoft Inc, 2005.

[12] *A Performance Comparison of Hypervisors*. VMWare Inc, 2007. www.vmware.com/pdf/hypervisor_performance.pdf.

[13] *VMware ESX Server 2 Architecture and Performance Implications*. VMWare Inc, 2005.

[14] *A Performance Comparison of Commercial Hypervisors*. XenSource, Inc., 2007.