Evaluating Virtual Router Performance for a Pluralist Future Internet

Diogo M. F. Mattos, Lyno Henrique G. Ferraz,
Luís Henrique M. K. Costa, and Otto Carlos M. B. Duarte
Universidade Federal do Rio de Janeiro - GTA/COPPE/UFRJ
Rio de Janeiro, Brazil
Email: {menezes, lyno, luish, otto}@gta.ufrj.br

***Abstract*—Internet Service Providers resist innovating in the network core, fearing that deploying a new protocol or service compromises the network operation and their profit, as a consequence. Therefore, a new Internet model, called Future Internet, which enables core innovation, must accommodate new protocols and services with the current scenario, isolating each protocol stack from others. Virtualization is the key technique that provides concurrent protocol stack capability to the Future Internet elements. In this paper, we evaluate the performance of three widespread virtualization tools, Xen, VMware, and OpenVZ, considering their use for router virtualization. We conduct experiments with benchmarking tools to measure the overhead introduced by virtualization in terms of memory, processor, network, and disk performance of virtual routers running on commodity hardware. We also evaluate the effects of the increasing number of virtual machines on Xen network virtualization mechanism. Our results show that Xen best fits virtual router requirements. Moreover, Xen fairly shares the network access among virtual routers, but needs further enhancement when multiple virtual machines simultaneously forward traffic.**

Keywords-Xen; OpenVZ; VMware; Hypervisor; Virtual Router

I. Introduction

The Internet success is mainly based on two pillars, the end-to-end data transfer service and the TCP/IP stack. The intelligence of the network is placed at the end systems, while the network core is simple and transparent. The TCP/IP model, however, has some structural issues that are difficult to solve, like scalability, mobility, management, and security [1]. Furthermore, the deployment of innovations on the network core is difficult because Internet Service Providers have no practical way to experiment new protocols and services in realistic scenarios without disturbing the running services. Current trends point to a new Internet architecture, which must provide flexibility and support for innovation in the network core [2]. Hence, many proposals for the Future Internet [3], [4] advocate a network model with multiple protocol stacks running simultaneously, called pluralist model. A way to provide a pluralist network is virtualizing router to allow multiple router instances to share the same underlying substrate. Thus, virtualization is a key concept for a pluralist architecture [5].

One of the key advantages of virtualization is isolating logical environments from each other. With virtual routers, multiple network stacks are deployed over the same physical substrate [6]. The virtual router concept adds flexibility to Future Internet architectures and keeps the new model backward compatible, as one of the virtual routers runs the current TCP/IP stack. In this sense, one powerful virtualizing technique is hardware virtualization, since it allows multiple router operating systems run over a hardware abstraction layer, which multiplexes the virtual router accesses to the physical substrate. Nevertheless, hardware virtualization introduces processing overhead to 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. We introduce a methodology to compare the different tools.

In this paper, we study three widespread hardware virtualization tools, VMware ESX [7], Xen [6], and OpenVZ [8]. Each one implements a different virtualization technique. The main difficulty to evaluate a virtualization tool is that common benchmarking tools have their measurements distorted by the time keeping inside the virtual environment [9]. Within a virtualized environment, the guest operation system does not have access to physical time sources or timer interrupts. Virtualized time system is usually implemented through software emulation of the real time devices. Thus, a difference between virtualized time and real world time often exists. Common benchmark tools use the guest OS time and get affected by the virtualized time distortion. Therefore, we propose a methodology to evaluate virtualized systems which is independent of virtual environment time distortion. We use the proposed methodology to evaluate the virtualization tools, and we conclude that Xen is the one that best fits virtual router requirements. Thus, we also analyze the performance of a Xen virtual router. Our results show that Xen virtual routers fairly share the network hardware access, although the aggregated packet forwarding performance is degraded as the number of virtual routers over the same physical substrate increases.

Our proposed evaluation methodology differs from the main proposals of virtualization tools benchmark because it takes external time sources as reference, while the conventional benchmarks take system specific variables as reference. Indeed, two common performance evaluation tools are VMmark [10] and Xenoprof [11]. The former can only be used with VMware platform and consists of measuring the score of processing several workloads simultaneously within different virtual machines. The latter is a profiler for Xen virtualization environment. Xenoprof provides detailed information about each individual process and routine running in the virtual machines or in the Xen hypervisor. Xenoprof estimates the virtualization overhead introduced by Xen. Xenoprof, how-

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

ever, is specific to Xen. Our methodology, on the other hand, is generic.

This paper is organized as follows. Section II analyzes the considered 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. Section VI concludes the paper and introduces our future work.

II. Virtualization Tools

Virtualization is the technology that allows sharing a physical hardware among multiple systems. Virtualized systems are isolated from each other, ignoring the existence of other systems sharing the same hardware. The software layer that isolates the guest systems is called hypervisor. We present three of the most well-known hypervisors available: VMware, Xen, and OpenVZ.

A. 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 is not modified or adapted to run in a virtual environment [7]. 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 accomplished 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 by the hypervisor, or Virtual Machine Monitor (VMM), a software layer that emulates the instruction execution. Therefore, the VMM adds a variable amount of virtualization overhead depending on the instruction type. VMware memory virtualization approach is to create a new level of memory address translation that provides to each guest operating system a virtual page table that is invisible to the memory-management unit (MMU) [6]. Within a VMware virtualization environment, the guest operating system accesses a virtual memory space that is internal to the virtual machine. As for network I/O virtualization, VMware implements the *vmxnet* [7] 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.

B. Xen

Xen is an open-source hypervisor designed to run on commodity hardware platforms [6]. Xen implements the paravirtualization technique, which enhances guest system performance by changing its behavior to call the hypervisor when necessary, avoiding the need of binary translation of system instructions. Xen virtualizes the processor by assigning virtual CPUs (vCPUs) to virtual machines. Xen hypervisor implements a CPU scheduler that dynamically maps a physical CPU to each vCPU during a certain period, based on a scheduling algorithm [12]. Memory virtualization in Xen is static and the RAM is divided among virtual machines. Each machine accesses a fixed amount of memory space, specified at the time of its creation. In addition, device drivers are kept in a special virtual machine, called the driver domain. The driver domain tasks are often executed by Domain 0, a privileged virtual machine that also executes hypervisor managing tasks. The driver domain access I/O devices directly by using its native device drivers and also performs I/O operations on behalf of virtual machines. In their turn, a virtual machine, also called unprivileged domain or Domain U, employs virtual I/O devices controlled by virtual drivers to request Domain 0 for device access [13], [14].

C. OpenVZ

OpenVZ is an open-source operating system-level virtualization tool [8]. OpenVZ allows 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 that 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 in 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 in each time slice, taking into account the standard process priority parameters. OpenVZ allows VPSs to directly access the memory. The memory amount dedicated for each VPS can be dynamically changed by modifying the virtual memory space of each VPS. OpenVZ kernel manages VPSes memory space in order to keep in physical memory the block of the virtual memory corresponding to the VPS that is currently running. 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 simplifies network virtualization because allows VPS to receive and to send packets using the host system routing module, but introduces an additional hop in the route packets follow.

III. Evaluation of Hypervisor

The main task of a router is to forward packets. The basic operation is to receive a packet, check the forwarding table for the packet destination, and forward the packet to the outgoing

network interface. Therefore, the main resources of interest for a router are networking, memory, and processing. Therefore, we evaluate networking throughput, memory access, and processing capacity, in order to assess which hypervisor has the most satisfactory performance for router virtualization. Although hard disk access performance is not critical for router virtualization, we also consider the hard disk virtualization performance to provide a more complete analysis. To provide a fair comparison, we run different tests to evaluate CPU, RAM, storage, and networking performance of the virtualized environments. Our main performance metric is the period of time required for each tool to perform a specific task. We use the following versions of the three hypervisors: Xen 3.2-1, VMware ESX 3.5, and Debian Linux Kernel 2.6.26-amd64 with OpenVZ support. All tools are considered with no modifications, tunings, or enhancements. We also present the results for a native Linux environment as a reference value. We expect that the results for the virtualized scenarios will be equal or worse than the reference values, as a consequence of the overhead introduced by virtualization.

A. Evaluation Methodology

Evaluating the virtualization overhead is not a trivial task. Whitepapers from VMware [15] and XenSource [16] compare the performance of the two hypervisors producing different conclusions. When virtualization is implemented, several issues arise from the fact that hardware is being shared by virtual machines. One of the problems specifically related to performance measuring is how the hypervisor provides the timekeeping mechanisms to the virtual machine [9]. There are several ways to keep track of time in a personal computer, like reading BIOS clock, using CPU registers like the Time Stamp Counter (TSC), or requesting system time from the OS. For the system timekeeping mechanism, Xen constantly synchronizes the virtual machine clock with the clock of Domain 0 by sending correct time information through shared memory between Domain U and the hypervisor. With that solution, system timekeeping is correct for most of the applications but, for applications that sample time more frequently than the virtual machine clock is being synchronized, a cumulative time measurement error may occur. To avoid such error, we do not consider timekeeping from the virtual machine in our tests, but instead we use an alternative time measurement procedure.

We use a different virtualization benchmarking methodology based on an external time measuring mechanism to evaluate the performance of the hypervisors. 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 needed to networking I/O. For the networking tests, the metric of interest is the throughput achieved by the virtual routers both when receiving and sending. For this kind of tests, it is necessary to have at least two computers, one for generating and another one for receiving traffic. Only one computer needs to be virtualized. The remaining non-virtualized system measures the amount of bits and the time difference, which are therefore not affected by virtualization.

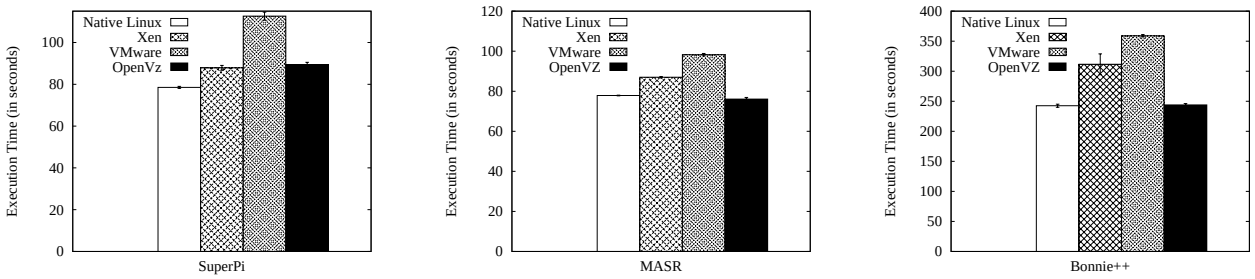
IV. Experimental Results

Our experimental setup is composed of a physical server, a traffic generator machine and a time measuring machine. The physical server that hosts the virtualized environments is an HP Proliant DL380 Generation 5 server equipped with two Quad Core Intel Xeon processors (2.83 GHz each), 10 GB of RAM, and an integrated 2-port Broadcom Nextreme II gigabit network interface. The traffic generator machine sends packets to the system under test or receives in the reverse-traffic test. The traffic generator machine is a desktop computer with an Intel motherboard, an Intel 2.4 GHz Core 2 Quad processor, 4 GB of RAM, and an on-board Intel gigabit network interface. The role of the Time Measuring Machine is to measure the period of time required by a system under test to perform each benchmark task. For that role we use a desktop computer equipped with an Intel motherboard, an Intel 2.66 GHz Core 2 Duo processor, 2 GB of system memory, and an on-board Intel gigabit network interface.

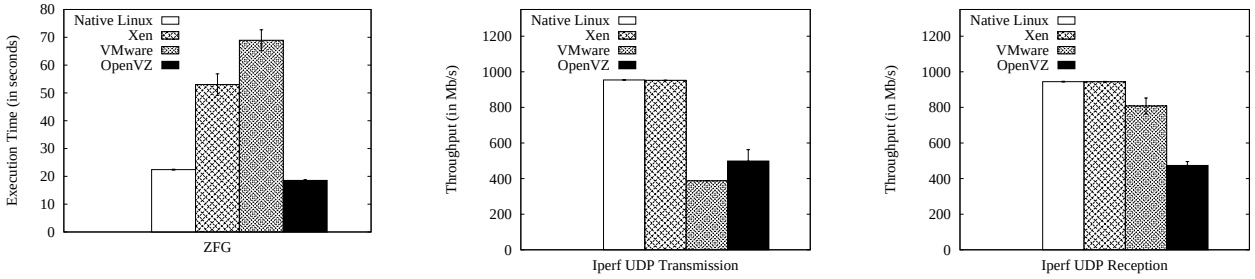
A. Processor Performance

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. We execute the Super Pi benchmark to compute the Pi value with 2^{22} digits.

The mean execution time for a round of the Super-Pi test is shown in Fig. 1(a), smaller values are faster and, consequently, better. The virtualization tools and native Linux are spread along the horizontal axis. Fig. 1(a) indicates that all virtualization tools introduce an overhead in processor usage. The smallest overhead is introduced by the Xen Hypervisor, which implements the paravirtualization technique. VMware introduces the bigger overhead in processor usage, because it uses full virtualization, where each instruction that generates a fault is trapped by the hypervisor. The hypervisor simulates its execution and then returns the result to the application in the virtual machine. This overhead is larger than using paravirtualization, where an instruction that would generate a fault is modified to execute directly over the hypervisor. OpenVZ has an overhead slightly higher than Xen, due to the scheduling mechanism to share CPU among the containers. Xen CPU scheduling involves vCPU context switching, whereas OpenVZ schedules VPSes and, inside a VPS, process over a single OS kernel context.



(a) Mean execution time for Super Pi benchmark. (b) Mean execution time for MASR benchmark. (c) Mean execution time for Bonie++ benchmark.



(d) Mean execution time for ZFG benchmark. (e) Traffic transmission throughput experiment. (f) Traffic reception throughput experiment.

Figure 1. CPU, memory, disk, and network benchmark results. All results are shown with a 95% of confidence interval.

B. Memory Performance

Memory access performance has significant influence in the overall router performance [6]. We developed a benchmarking tool, called *MASR* (Memory Allocation, Set and Read), to evaluate the overhead caused by the virtualization layer on virtual router memory access. MASR benchmarks memory performance by allocating 2 GB of memory, setting sequential memory positions to a fixed value and afterwards reading each one. MASR was developed for benchmarking memory with a deterministic number of operations, no matter the performance of the computer. The main focus of common memory benchmarks is to determine writing or reading data rates on memory. On the other hand, MASR focus is to execute deterministic memory writings and readings, independently of the virtualization tool, to allow comparing the time taken to execute the same set of memory operations between each virtual machines tools.

Fig. 1(b) shows the results for the MASR benchmark. The mean execution time for a round of the test is shown in the vertical axis. OpenVZ is the virtualization tool that introduces less overhead in memory access. OpenVZ directly accesses the memory, then a 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. A Xen virtual machine accesses a portion of the total system memory. As Xen employs paravirtualization, the virtual machine is aware of its address space and directly handles the physical address, improving the memory access performance. VMware has worse memory access performance than the others. VMware implements memory translation. As it is fully virtualized, a virtual machine is unaware that it runs over a hypervisor, and when it tries to access memory, the memory instructions are trapped by the hypervisor. The hypervisor translates and executes the memory access instruction, and then gives the result to the virtual machine. Hence, it introduces a larger overhead than the other hypervisors.

C. Hard Disk Performance

For sake of completeness, we also analyze the hard disk access performance even though this resource is the least important for a virtual router. 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 small files. Bonnie++ 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 the disk. If that 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. Common disk benchmark tools do not guarantee these properties.

Fig. 1(c) and Fig. 1(d) present the hard disk access performance comparison. The mean execution time for a round of the

test is shown in the vertical axis, smaller values are better. The virtualization tools and native Linux are along the horizontal axis. 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 close to native Linux, as shown in Fig. 1(c) and Fig. 1(d). On the other hand, Xen and VMware perform poorly, especially with ZFG. As Xen and VMware 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 causing additional processing overhead and memory page copies.

D. Networking Performance

A virtual router must be able to efficiently receive packets and send them to the 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 throughput from an external host to the system under test, and vice-versa. We use an UDP flow with data packets of 1472 bytes, which represents the maximum size for a packet without fragmenting Ethernet frames.

The networking evaluation results are shown in Fig. 2. First of all, it is important to highlight that, for both traffic transmission (Fig. 1(e)) and reception (Fig. 1(f)), native Linux system achieves a throughput near the nominal gigabit Ethernet transmission bit rate. Xen virtual machine also achieves the same transmission and reception rates of native Linux. This shows that Xen networking virtualization mechanism is fast enough in sending/receiving packets to/from virtual machines and was not the bottleneck in the evaluated scenario. On the other hand, VMware and OpenVZ present a poor performance for network virtualization.

VMware has a lower network performance than Xen, but it is better than OpenVZ. VMware performs worse than Xen, because it emulates a network device to the virtual machine. Then, the virtual machine calls a generic network device interface, which is mapped to hypervisors functions, and after, mapped in devices functions. It also implies on copying the packet twice to the memory: one from the network interface to the hypervisor memory, and other from hypervisor memory to the virtual machine memory. Xen copies the packet to memory only once, after that, it handles the descriptor of the memory address. OpenVZ has the worst network performance because it implements the network virtualization on higher level. OpenVZ virtualization environment access protocols on network device at IP layer, but does not access the Ethernet layer. Hence, to send a packet to the virtual environment and from the virtual environment to the network, the OpenVZ hypervisor needs to route the packet between host and virtual hosts. The additional hop introduced by OpenVZ plus the scheduling time needed to run the host and the virtual environment justifies its bad network virtualization performance.

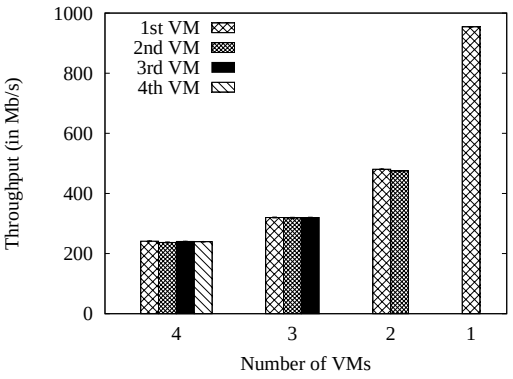
V. XEN VIRTUAL ROUTER EVALUATION

Xen performs better than VMware and OpenVZ for network, therefore, we further investigate Xen as a possible platform for router virtualization and evaluate its scalability. We analyze the packet forwarding capability of a Xen virtual router for increasing routing table and increasing number of virtual machines over the same hypervisor.

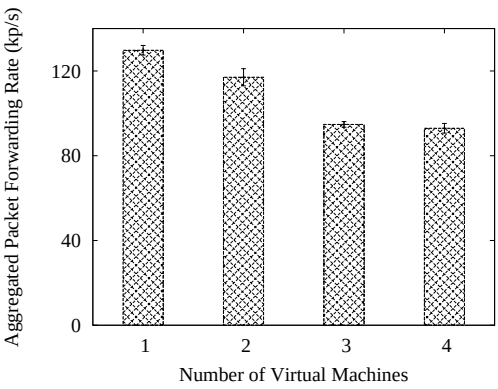
The routing table size influences the packet forwarding rate, because a large table increases the per-packet delay on routing table lookup. The Linux kernel, however, implements the routing table as a hash table. As a consequence, there is no linear search over the routing table and the next hop in packet route is discovered in a constant time, regardless of the routing table size. To confirm this fact, we conduct experiments in which a 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 typical size for the routing table of a BGP border router [17]. 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 also perform an evaluation of the fairness in Xen network virtualization mechanism, whereas multiple virtual machines share a single network interface card. In a first experiment, whose results are shown in Fig. 2(a), we instantiate from one up 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, 1472 B. 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. Therefore, Xen network virtualization fairly scales to multiple virtual machines.

We also evaluate the degradation of the packet forwarding rate as the number of virtual routers increases. In this experiment, the sending packet rate is 130 kp/s and we measure the packet forwarding rate as the rate of packets that reach the next-hop destination machine after being forwarded by the virtual router. Fig. 2(b) shows that, as the number of virtual routers running in parallel increases, the traffic forwarding performance degrades. It is a consequence of the processing capacity starvation [12]. The CPU scheduler, which is responsible for sharing the processor among all virtual routers, has to switch context between a running virtual router and a blocked one. The time consumed with context switching comes at the expense of time for serving router vCPUs. In this way, 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 virtual routers drop packets, reducing the aggregated packet forwarding rate. Our results show a 30% reduction on the overall packet forwarding rate, when four virtual routers are running over the same hypervisor.



(a) Network traffic transmission test.



(b) Network traffic forwarding test.

Figure 2. Xen scalability tests with multiple virtual machines sending/forwarding network traffic to the external computer.

VI. Conclusion

In this paper, we have investigated the virtualization tool that best fits the requirements of a virtual router for a pluralist Future Internet architecture. We have proposed a new methodology for evaluating the overhead introduced by the virtualization layer, considering more precise time accounting. We performed CPU, memory, disk, and networking experiments. Our results show that OpenVZ is the virtualization tool that introduces less overhead over CPU, disk, and memory usage performing almost as well as native Linux. On the other hand, OpenVZ requires the same operating system (OS) and degrades for networking, which is crucial for virtual routing. Xen presents the best performance for networking and has a small overhead on processor and memory usage. VMware, which provides a fully virtualized environment, is flexible, but introduces bigger overhead over all resources usage.

Xen provides multiple virtual environments, each one with its complete OS environment, independent of other virtual machines and of host OS. Xen virtual environment performance is compatible with the virtual router requirements. Hence, Xen is the virtualization tool that presents the best trade-off between performance and flexibility. Therefore, it fits well for router virtualization requirements. Hence, we have also analyzed 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 and the transmitted throughput is maintained, the throughput of each router is equally reduced. Nevertheless, the aggregated throughput is maintained. In the second scenario, we increase the routing table size and the packet forwarded rate remains the same.

Xen Hypervisor, however, must be enhanced to support a virtual router within a production network. Our results have shown that, when submitted to a high packet transmission rate, the packet forwarding rate is reduced by 30% when running multiple virtual machines over the same hypervisor. Therefore, our future work focuses on developing Xen as virtual router and evaluating new virtualization techniques for improving virtual routers performance.

Acknowledgements

We would like to thank Carlo Fragni and Marcelo D. D. Moreira for their help with the benchmarking tools and measurements.

References

[1] N. Fernandes, M. Moreira, I. Moraes, L. Ferraz, R. Couto, H. Carvalho, M. Campista, L. Costa, and O. Duarte, "Virtual networks: isolation, performance, and trends," *Annals of Telecommunications*, vol. 66, pp. 339–355, 2011.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S., and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[3] N. Feamster, L. Gao, and J. Rexford, "How to Lease the Internet in your Spare Time," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 61–64, Jan. 2007.

[4] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy, "Network Virtualization Architecture: Proposal and Initial Prototype," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, ser. VISA '09. New York, NY, USA: ACM, 2009, pp. 63–72.

[5] M. Casado, T. Koponen, R. Ramanathan, and S. Shenker, "Virtualizing the network forwarding plane," ser. PRESTO '10, 2010, pp. 8:1–8:6.

[6] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley, "Evaluating xen for router virtualization," in *International Workshop on Performance Modeling and Evaluation (PMECT)*, August 2007.

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

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

[9] S. Chen, M. Zhu, and L. Xiao, "Implementation of virtual time system for the distributed virtual machine monitor," *IEEE/ISECS International Colloquium on Computing, Communication, Control, and Management*, August 2009.

[10] V. Makhija, B. Herndon, P. Smith, L. Roderick, E. Zamost, and J. Anderson, "Vmmark: A scalable benchmark for virtualized systems," *VMware Inc, CA, Tech. Rep. VMware-TR-2006-002, September*, 2006.

[11] D. Gupta, R. Gardner, and L. Cherkasova, "Xenmon: Qos monitoring and performance profiling tool," *HP Labs, Tech. Rep.*, 2005.

[12] R. S. Couto, M. E. M. Campista, and L. H. M. K. Costa, "XTC: a throughput control mechanism for Xen-based virtualized software routers," in *IEEE Global Communications Conference (GLOBECOM'2011)*, Houston, Texas, USA, Dec. 2011.

[13] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in Xen," in *USENIX Annual Technical Conference*, May 2006, pp. 15–28.

[14] K. Ibrahim, S. Hofmeyr, and C. Iancu, "Characterizing the performance of parallel applications on multi-socket virtual machines," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*. IEEE, 2011, pp. 1–12.

[15] *A Performance Comparison of Hypervisors*, VMWare Inc, 2007. [Online]. Available: www.vmware.com/pdf/hypervisor_performance.pdf

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

[17] "BGP Reports," http://bgp.potaroo.net/ipv4-stats/prefixes_adv_pool.txt, Jan. 2012.