Virtual Network Performance Evaluation for Future Internet Architectures

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.**

Index Terms— **Xen, OpenVZ, VMware, Hypervisor, Virtual Router, Pluralism, Future Internet.**

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]–[3]. 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 [4]. Hence, many proposals for the Future Internet [5], [6] 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. In this sence, the current Internet architecture may run in parallel with other virtual networks, e.g., a secure network [7] or a intelligence-oriented network [8], sharing the same physical substrate. Thus, virtualization is a key concept for a pluralist architecture [9].

One of the key advantages of virtualization is isolating logical environments from each other [10]. With virtual routers, multiple network stacks are deployed over the same physical substrate [11]. 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 extend a previous work [1] in which we study three widespread hardware virtualization tools, VMware ESX [12], Xen [11], and OpenVZ [13]. 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 [14]. 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

This paper is based on "Evaluating Virtual Router Performance for a Pluralist Future Internet," by D. M. F. Mattos, L. H. G. Ferraz, L. H. M. K. Costa, and O. C. M. B. Duarte, which appeared in the Proceedings of the 3th International Conference on Information and Communication Systems (ICICS 2012), Irbid, Jordan, April 2012.

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

doi:10.4304/jetwi.4.4.304-314

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. We also evaluate how new hardware-assisted I/O virtualization technologies affects Xen network virtualization performance.

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 [15] and Xenoprof [16]. 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, however, 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 evaluates hardware assited Xen network virtualization performance. Section VII concludes the paper and introduces our future work.

II. Virtualization Tools

Virtualization is a technology that allows sharing a physical substrate 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 shared guest systems is called Virtual Machine Monitor (VMM) or hypervisor. We present three of the most well-known available hypervisors: VMware, Xen, and OpenVZ. VMware and Xen are hardware virtualization systems that virtualize hardware resources such as CPU, memory, and I/O guaranteeing high flexibility and isolation between the virtualized systems [2]. OpenVZ, however, claims to be a lightweight virtualization platform because it virtualizes the OS kernel, creating multiple isolated user spaces [13].

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 [12]. It is mainly used for server consolidation and it is one of most used enterprise virtualization software. VMware ESX Server aims at guaranteeing virtual machine isolation and resource-sharing fairness based on resource-allocation policies set by the system administrator. Resource sharing is dynamic, because resources can be allocated and re-allocated to virtual machines on demand [12].

VMware architecture, as shown in Figure 1, is composed of the Hardware Interface Components, the Virtual Machine Monitor, the VMkernel, the Resource Manager, and the Service Console. The Hardware Interface Components are responsible for implementing hardware-specific functions and create a hardware abstraction that is provided to virtual machines. It makes Virtual Machines hardware independent. The Virtual Machine Monitor (VMM) is responsible for CPU virtualization, providing a virtual CPU to each virtual machine. The VMkernel controls and manages the hardware substrate. VMM and VMkernel together implement the Virtualization Layer. The Resource Manager is implemented by VMkernel. It partitions the underlying physical resources among the virtual machines, allocating the resources for each one. VMkernel also implements the hardware interface components. The Service Console implements a variety of services such as bootstrapping, initiating execution of virtualization layer and resource manager, and runs applications that implements supporting, managing, and administrative functions.

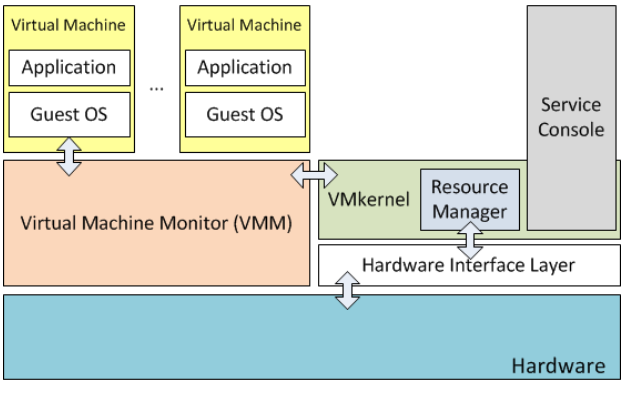


Figure 1. VMware architecture.

VMware ESX Server, as others virtualization tools, virtualizes four main resources: CPU, memory, disk, and network device. We will further detail bellow how VMware virtualizes each resource.

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 seem to have their own registers and control structures [12]. A virtual machine can have one or two virtual CPUs. When it has more than one CPU, it is called a Symmetric Multi-Processing (SMP) virtual machine. The virtual machine monitor is responsible for CPU virtualization, by setting system states and executing instructions issued by the virtual machine.

In a virtualized environment, the guest operating system runs in a lower privilege level than it was designed to run. A classical approach to virtualize CPU resources

is trap-and-emulate, which is a technique in which the virtual machine tries executing an instruction, and, if it cannot be executed in a lower privilege level, the CPU generates a trap that is treated by the VMM, which emulates the instruction execution to the guest operating system. Nevertheless, this technique does not work for the x86 architecture, since it has instructions that are sensitive to privilege level and would execute in a different way than meant by the OS, without generating a trap. In order to solve this issue and keep a satisfactory performance, VMware combines two CPU virtualization techniques: direct execution and CPU emulation. The instructions from the user-space of a virtual machine are executed directly on the physical CPU, a technique known as direct execution. Instructions that are sensitive to privilege level are trapped by the VMM, which emulates the instruction execution, adding performance overhead. Combining both techniques allows CPU intensive user-space applications to have near-native performance. The performance loss depends on the number of sensitive instructions that had to be replaced.

CPU scheduling is made by the Resource Manager. CPU scheduling is based on `shares`, which are units used to measure how much time is given to each virtual machine. CPU scheduling is proportional-share, meaning that CPU time given to each virtual machine is proportional to the amount of `shares` it has in comparison with the total amount of `shares` in the system. In a Symmetric Multi-Processing (SMP) virtual machine, CPU allocation is different. Resource Manager schedules the virtual CPUs one-to-one onto physical CPUs, and tries executing them at the same time. VMware CPU scheduling tries to keep fairness between virtual machines CPU allocation. When a virtual machine is halted, or idle, Resource Manager schedules its CPU time to other virtual machines that are running.

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) [17]. Within a VMware virtualization environment, the guest operating system accesses a virtual memory space provided to the virtual machine. The guest operating system page table maintains the consistency between guest virtual pages and guest virtual "physical" pages. Guest virtual pages are virtual memory pages within a virtual machine, as in a native operating system virtual memory. However, guest virtual paging mechanism cannot access directly the physical memory, it accesses guest virtual "physical" memory. Guest virtual "physical" memory is an abstraction of the physical memory. When a guest operating system tries to execute an instruction to access physical memory, this instruction is trapped by the VMM and its address is translated to the real physical address. Guest virtual "physical" memory is always contiguous, but can be mapped into non-contiguous real physical memory. VMware memory sharing obeys administration policies, which for example defines a minimum and a maximum amount of physical memory to be accessed by a virtual machine. It is also possible to have virtual machines consuming more than the total amount of physical memory available on the physical machine. This is possible because the host system can also do swap as in a traditional virtual memory mechanism used in modern OSes. The memory sharing scheduler works like the CPU scheduler, but takes into account memory shares instead of CPU shares.

VMware I/O virtualization approach is to emulate performance-critical devices, such as disk and network interface cards. Device accesses are emulated by the VMkernel. The VMkernel calls the hardware interface layer, which is responsible for accessing the device driver and executing the operation on the physical hardware device. For storage virtualization, a SCSI driver is presented to the virtual machine. Virtual machines access this driver, the VMkernel traps driver access instructions and implements virtual machine disks as files in the host file system.

Concerning network I/O virtualization, VMware implements the *vmxnet* [12] device driver, which is an abstraction of the underlying physical device. When an application wants to send data through the network, the guest operating system processes the request and calls the *vmxnet* device driver. The I/O request is intercepted by the VMM and control is transferred to VMkernel. VMkernel is independent of the physical device. It processes the request, manages the various virtual machine requests, and calls the hardware interface layer, which implements the specific device driver. When data arrives to the physical interface, the mechanism for sending it to the specific virtual machine is the same, but in reverse order. The main overhead introduced by this mechanism is the context switching between the virtual machine and the VMkernel. In order to decrease the overhead caused by context switching, VMware ESX Server collects cluster of sending or receiving network packets before doing a context transition. This mechanism is only used when packet rate is high enough, to avoid increasing packet delays.

Summing up, VMware ESX Server is a full virtualization tool which provides a number of management and administrative tools. VMware ESX Server is focused on datacenter virtualization. It provides a flexible and high-performance CPU and memory virtualization. However, I/O virtualization is still an issue, since it is done by emulating the physical devices and involves context changes.

B. Xen

Xen is an open-source hypervisor designed to run on commodity hardware platforms [17]. Xen allows to simultaneously run multiple virtual machines on a single physical machine. 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 architecture is composed of

one hypervisor located above the physical hardware and several virtual machines over the hypervisor, as shown in Figure 2. Each virtual machine has its own operating system and applications. The hypervisor controls the access to the hardware and also manages the available resources shared by virtual machines. In addition, device drivers are kept in an isolated virtual machine, called *Domain 0* (dom0), in order to provide reliable and efficient hardware support [11]. Because dom0 has total access to the hardware of the physical machine, it has special privileges compared with other virtual machines, referred to as user domains (domUs). On the other hand, user domains have virtual drivers, called front-end drivers (fe), which communicate with the back-end drivers (be) located in dom0 to access the physical hardware. Next, we briefly explain how Xen virtualizes each machine resource of interest to a virtual router: processor, memory, and I/O devices.

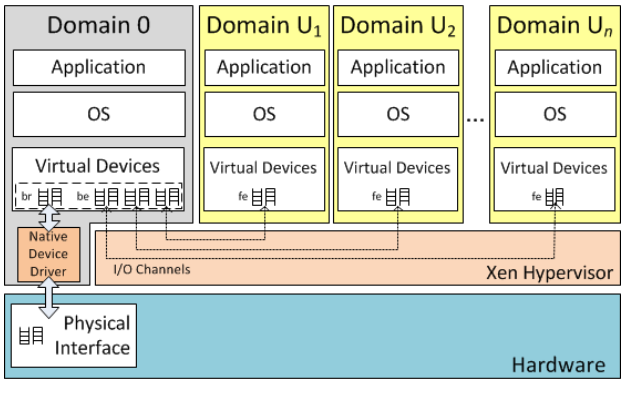


Figure 2. The Xen architecture.

Xen virtualizes the processor by assigning virtual CPUs (vCPUs) to virtual machines. Virtual CPUs are the CPUs that the running processes within each virtual machine can see. The hypervisor maps vCPUs to physical CPUs. Xen hypervisor implements a CPU scheduler that dynamically maps a physical CPU to each vCPU during a certain period. The default Xen scheduler is the Credit Scheduler [18], which makes a proportional CPU share. The Credit scheduler allocates CPU resources to each virtual machine (or, more specifically, to each vCPU) according to weights assigned to virtual machines. The Credit scheduler can also be work conserving on Symmetric Multi-Processing (SMP) hosts. This means that the scheduler permits the physical CPUs to run at 100% if any virtual machine has work to do. In a work-conserving scheduler there is no limit on the amount of CPU resources that a virtual machine can use [19].

Memory allocation in Xen is currently done statically. Each virtual machine receives a fixed amount of memory space which is specified at the time of its creation. In addition, to require a minimal involvement from the hypervisor, virtual machines are responsible for allocating and managing the corresponding portion of the hardware page tables. Each time a virtual machine requires a new page table, it allocates and initializes a page from its own memory space and registers it with the Xen hypervisor, which is responsible to ensure isolation.

In Xen, data from I/O devices is transferred to and from each virtual machine using shared-memory asynchronous buffer descriptor rings. The task of Xen hypervisor is to perform validation checks, e.g., checking that buffers are contained within a virtual machine memory space. Dom0 access I/O devices directly by using its native device drivers and also performs I/O operations on behalf of user domains (domUs). On the other hand, user domains employ their back-end drivers to request device access from dom0 [20]. A special case of I/O virtualization is network virtualization, which is responsible for demultiplexing incoming packets from physical interfaces to virtual machines and also for multiplexing outgoing packets generated by virtual machines. For each domU, Xen creates the virtual network interfaces required by this domU. These interfaces are called front-end interfaces (fe) and are used by domUs for all of its network communications. Furthermore, back-end interfaces (be)are created in dom0, corresponding to each front-end interface in a user domain. In order to exchange data between back-end and front-end interfaces, Xen uses an I/O channel, which employs a zero-copy mechanism. This mechanism remaps the physical page containing the data into the target domain [20]. Back-end interfaces act as the proxy for the virtual interfaces in dom0. Front-end and back-end interfaces are connected to each other through the I/O channel. In Figure 2, back-end interfaces in dom0 are connected to the physical interfaces and also to each other through a virtual network bridge. This default architecture used by Xen is called bridged mode. Thus, both the I/O channel and the network bridge establish a communication path between the virtual interfaces created in user domains and the physical interface.

C. OpenVZ

OpenVZ is an open-source operating system-level virtualization tool. OpenVZ allows multiple isolated execution environments over a single operating system kernel. Each isolated execution environment is called a *Virtual Private Server* (VPS). A VPS looks like a physical server, having its own processes, users, files, IP addresses, system configuration, and providing full root shell access. OpenVZ claims to be the virtualization tool which introduces less overhead, because each VPS shares the same operating system kernel, providing a high-level virtualization abstraction. The main usages for this virtualization technology are in web hosting, providing every customer a complete Linux environment, and in information technology (IT) education institutions, providing every student a Linux server that can be monitored and managed remotely [13]. Despite the small overhead introduced by OpenVZ, it 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.

OpenVZ architecture, as shown in Figure 3, is composed of a modified Linux kernel that runs above the hardware. The OpenVZ modified kernel implements virtualization and isolation of several subsystems, resource management and checkpoints [13]. In addition, I/O virtualization mechanisms are provided by the OpenVZ modified kernel, which has a device driver for each I/O device. This modified kernel implements also a two-level process scheduler which is responsible of, in the first level, defining which VPS will run and, in the second level, on deciding which VPS process will run. The two-level scheduler and some features that provide isolation between VPSs form the OpenVZ Virtualization Layer. VPSs run above the OpenVZ Virtualization Layer. Each VPS has its own set of applications and packages, which are segmentations of certain Linux Distributions that contains applications or services. Therefore a VPS can have its own services and applications independent of each other.

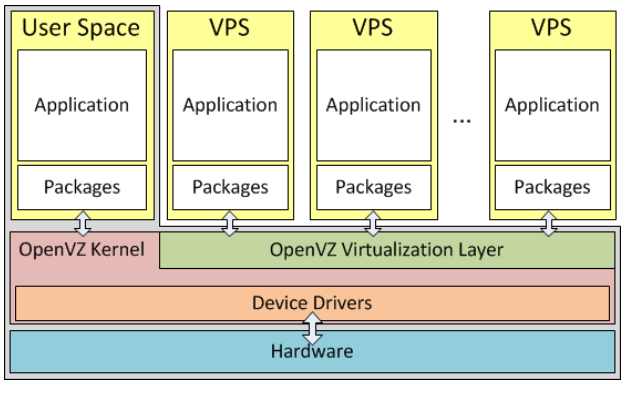


Figure 3. The OpenVZ architecture.

Resource virtualization in OpenVZ is done by allowing or prohibiting a VPS to access a resource on the physical server. In general, resources in OpenVZ are not emulated, they are shared among VPSs. In order to define the amount of each resource that is guaranteed for each VPS, a number of counters (about 20) are defined in VPS configuration file. Next, we further detail how OpenVZ virtualizes the processor, memory, and disk and network devices.

For processor virtualization, OpenVZ implements a two-level CPU scheduler [13]. In the first level, the virtualization layer decides which VPS will execute for each time slice, taking into account the VPS CPU priority, measured in *cpuunits*. 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. Moreover, it is more flexible than other virtualization technologies, such as Xen. During VPS execution, the memory amount dedicated to one VPS can be dynamically changed by the host administrator. OpenVZ kernel manages VPSs memory space to keep in physical memory a block of the virtual memory corresponding to the running VPS.

OpenVZ virtual disk is a partition of the host file system. Similarly to CPU scheduling, OpenVZ disk usage is determined by a two-level disk quota. On the first level, OpenVZ virtualization layer defines a disk quota for each VPS, for example by limiting the maximum size of folder in the host file system. On the second level, it is possible to define disk quotas for users and groups in a VPS, using standard Linux quota mechanisms.

Network virtualization layer isolates VPSs from each other and from the physical network [13]. The default network virtualization mechanism of OpenVZ 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 the destination IP address of 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.

Summing up, OpenVZ provides a high-level virtualization abstraction and, thus, it claims to introduce less overhead than other virtualization tools. On the other hand, it is more restrictive in terms of the virtual environments that have to share the physical host system kernel.

III. Evaluation of the Hypervisors

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. Methodology

Evaluating the virtualization overhead is not a trivial task. Whitepapers from VMware [21] and XenSource [22]

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 [14]. 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, as shown in Figure 4. 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. All results are shown with a 95% confidence interval.

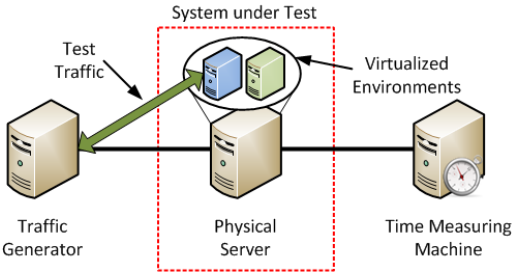
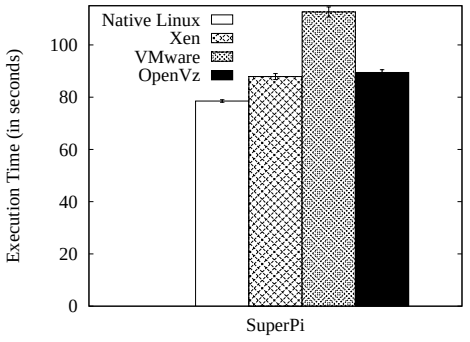


Figure 4. Experimental setup.

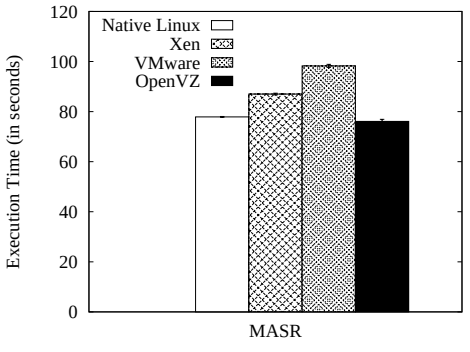
A. Processor Performance

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 π 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. 5(a), smaller values are faster and, consequently, better. The virtualization tools and native Linux are spread along the horizontal axis. Fig. 5(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.

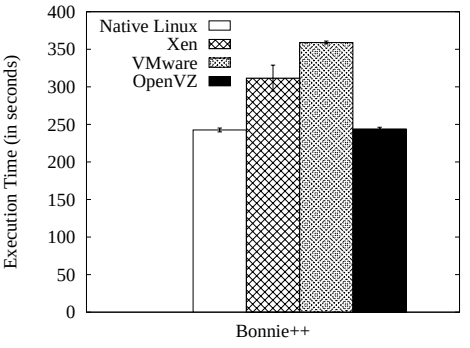


(a) Mean execution time for Super Pi benchmark.

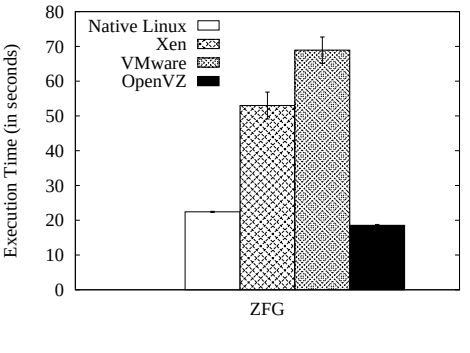


(b) Mean execution time for MASR benchmark.

Figure 5. CPU and memory benchmark results.



(a) Mean execution time for Bonie++ benchmark.



(b) Mean execution time for ZFG benchmark.

Figure 6. Disk benchmark results.

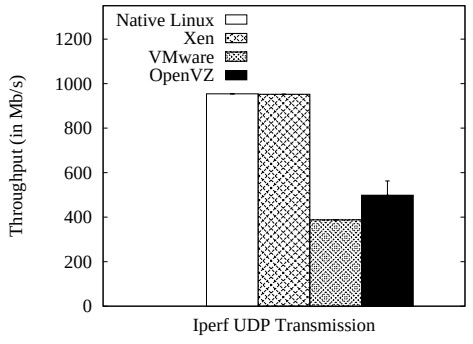
B. Memory Performance

Memory access performance has significant influence in the overall router performance [11]. 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, independently of 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.

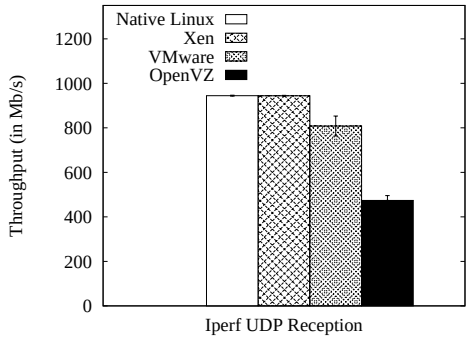
Fig. 5(b) shows the results for the MASR benchmark. The mean execution time for a round of the test is shown in the vertical axis. OpenVZ is the virtualization tool that introduces less overhead in memory access. OpenVZ directly accesses the memory, then a virtual environment accesses memory as an application 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 [23]. 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



(a) Traffic transmission throughput experiment.



(b) Traffic reception throughput experiment.

Figure 7. Network benchmark results.

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. 6(a) and Fig. 6(b) 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. 6(a) and Fig. 6(b). 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 [24] 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. 8. First of all, it is important to highlight that, for both traffic transmission (Fig. 7(a)) and reception (Fig. 7(b)), 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.

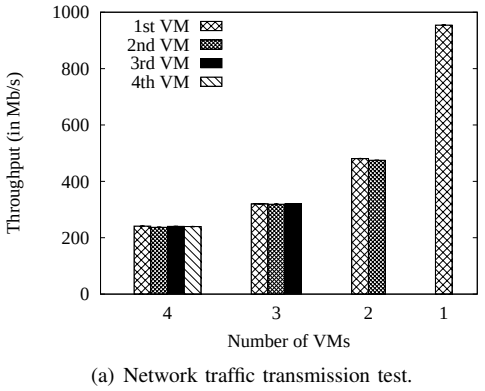
V. Xen Virtual Router Evaluation

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.

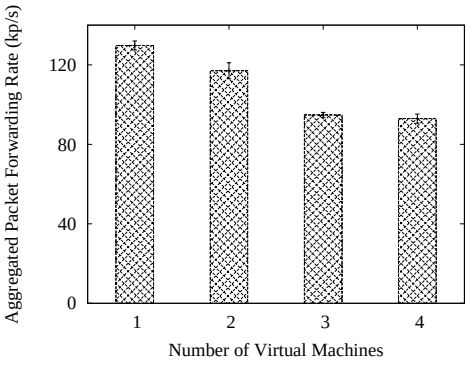
The routing table size influences the packet forwarding rate, because a large table increases the per-packet delay on routing table lookup. The Linux kernel, however, implements the routing table as a hash table. As a

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 [25]. 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. 8(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.



(a) Network traffic transmission test.



(b) Network traffic forwarding test.

Figure 8. Xen scalability tests with multiple virtual machines sending/forwarding network traffic to the external computer.

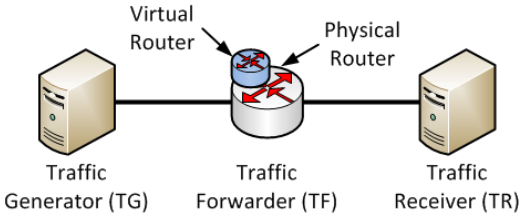
We also evaluate the degradation of the packet forwarding rate as the number of virtual routers increases. In this experiment, the sending packet rate is 130 kp/s and we measure the packet forwarding rate as the rate of packets that reach the next-hop destination machine after being forwarded by the virtual router. Fig. 8(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 [19]. 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.

VI. Hardware-Assisted Network Virtualization

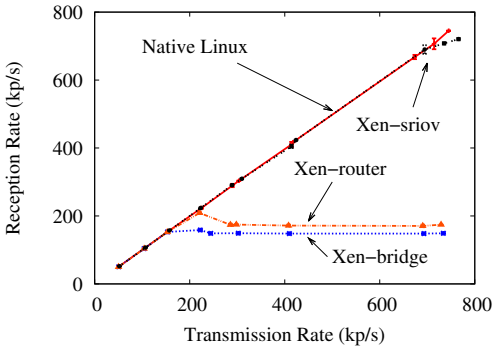
Xen hypervisor multiplexes virtual machine access to the physical network interface cards using either bridge or router modes. In bridge mode, packet forwarding among Xen virtual interfaces and physical network interface cards is performed as layer-2 forwarding. Thus, as a packet arrives at a physical interface in Domain 0, the bridge forwards packets to the correct virtual machine according to their MAC addresses. The bridge mode implements a software switch. On the other hand, the router mode performs like an IP router. Therefore, when a packet arrives at the network physical interface card in Domain 0, it is forwarded to the correct virtual machine according to its IP address. As the Domain 0 uses IP addresses to forward packets to virtual machines, it has valid routes to each virtual machine.

Besides bridge and router modes, another possible way of implementing packet forwarding among virtual routers and physical interface cards is using hardware-assisted virtualization techniques. In this mode, virtual routers directly access network interface cards, without Xen hypervisor interference, such as the PCI-SIG Single Root I/O Virtualization (SR-IOV) [26]. Hence, hardware-assisted virtualization techniques improve I/O virtualization performance, as they reduce hypervisor participation into I/O operations. A single PCI Express (PCIe) device with SR-IOV technology presents itself as multiple *virtual functions*, which act as dedicated devices to different virtual routers. Each virtual function is dedicated to a virtual router that has direct access to the hardware instance as a virtual device. Thus, it avoids context switching, control overhead, and traffic classification tasks. In addition, it also avoids extra memory copies when compared with software switches for multiplexing I/O operations to access physical devices.

We perform an evaluation of the forwarding packet rate comparing Xen default network virtualization modes with hardware assisted network I/O virtualization. Three computers compose the experiment scenario. The computers act as traffic generator (TG), as traffic receiver (TR) and as traffic forwarder (TF), as shown in Figure 9(a).[1]



(a) Experiment topology.



(b) Packet forwarding rate for 64 Bytes packets.

Figure 9. Packet forwarding rate for a Xen virtual router with hardware-assisted I/O virtualization.

Our experiment compares the routing of I/O hardware-assisted virtualization with native Linux, Xen bridge mode, and Xen router mode. The traffic generator (TG) generates a flow of 64 bytes packets to the traffic receiver (TR) through the traffic forwarder (TF). The main idea of this experiment is measuring the maximum rate of packet forwarding in the four scenarios. The native Linux scenario is the theoretical maximum forwarding rate that the physical router can achieve. In bridge mode, the packets are switched between the physical and virtual interfaces. In router mode, routing tables are created to forward packets among physical and virtual interfaces. Finally, when virtualizing I/O with hardware-assisted techniques, each virtual router directly accesses a virtual network function, which is created by physical network devices according to the PCI-SIG SR-IOV specification. Figure 9(b) shows that native Linux forwarding scheme achieves packet forwarding rates of up to 750 kp/s. Xen bridge and router network virtualization modes perform worse and do not achieve rates above 150 kp/s and 200 kp/s, respectively. In Xen bridge and router modes, packets are first received on a physical interface card by Domain 0, which forwards packets to the virtual routers. Then, virtual routers forward packets back to Domain 0, which sends them to their next destination. This process overloads the virtual router network access and, consequently, degrades the overall network performance. In the scenario of I/O virtualization with SR-IOV, the hardware device itself classifies packets addressed to the virtual routers, and virtual routers directly access the physical device dedicated to each of them. Hence, hardware-assisted I/O virtualization achieves a packet forwarding rate similar to the one of native Linux.

[1]The generator and receiver are servers equipped with two Intel Xeon Quad-Core 2.93 GHz processors and 48 GB of RAM. The traffic forwarder is a desktop computer equipped with an Intel i7 Quad-Core 3.06 GHz and an Intel E1G44ET network card, supporting SR-IOV I/O. The kernel version of the physical machines is 2.6.32-5, and virtual router kernel is the 2.6.32-5-xen.

VII. 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. We also evaluate a new network virtualization technique, in which each virtual router directly accesses the physical device. Our results show that applying this technique, a virtual router forwards the same packet rate as a native Linux system. Therefore, our future work focuses on developing Xen as virtual router and deploying a virtual network testbed based on Xen.

ACKNOWLEDGMENT

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

REFERENCES

[1] D. M. F. Mattos, L. H. G. Ferraz, L. H. M. K. Costa, and O. C. M. B. Duarte, "Evaluating virtual router performance for a pluralist future Internet," in *Proceedings of the 3rd International Conference on Information and Communication Systems*, ser. ICICS '12. Irbid, Jordan: ACM, Apr. 2012.

[2] 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, Oct. 2011.

[3] P. Pisa, R. Couto, H. Carvalho, D. Neto, N. Fernandes, M. Campista, L. Costa, O. Duarte, and G. Pujolle, "Vnext: Virtual network management for Xen-based testbeds," in *Network of the Future (NOF), 2011 International Conference on the*, Nov. 2011, pp. 41–45.

[4] 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.

[5] 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.

[6] 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. Barcelona, Spain: ACM, Aug. 2009, pp. 63–72.

[7] S. Panda and V. Mangla, "Protecting data from the cyber theft-a virulent disease," *Journal of Emerging Technologies in Web Intelligence*, vol. 2, no. 2, pp. 152–155, 2010.

[8] I. Lykourentzou, D. Vergados, E. Kapetanios, and V. Loumos, "Collective intelligence systems: Classification and modeling," *Journal of Emerging Technologies in Web Intelligence*, vol. 3, no. 3, pp. 217–226, 2011.

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

[10] N. Fernandes and O. Duarte, "XNetMon: A network monitor for securing virtual networks," in *Communications (ICC), 2011 IEEE International Conference on*, June 2011, pp. 1–5.

[11] 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)*, Aug. 2007.

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

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

[14] 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*, Aug. 2009.

[15] 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*, Sept. 2006.

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

[17] 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 *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles - SOSP03*, Oct. 2003.

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

[19] 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.

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

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

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

[23] R. Coker. (2012, June) Bonnie++ file-system benchmark. [Online]. Available: http://www.coker.com.au/bonnie++/

[24] NLANR/DAST. (2012, June) Iperf. [Online]. Available: http://iperf.sourceforge.net/

[25] T. Bates, P. Smith, and G. Huston. (2012, Jan.) BGP Reports. [Online]. Available: http://bgp.potaroo.net/ipv4-stats/prefixes_adv_pool.txt

[26] I. L. A. Division, "PCI-SIG SR-IOV primer, an introduction to SR-IOV technology," Intel, Tech. Rep., Jan. 2011.

Diogo M. F. Mattos is currently a M.Sc. candidate at Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil. He received the Computer Engineer degree from Universidade Federal do Rio de Janeiro, Rio de Janeiro, in 2011. His research interests include virtualization, software-defined networks, Future Internet and network security.

Lyno Henrique G. Ferraz is currently pursuing his Ph.D. degree in the Electrical Engineering Program at Federal University of Rio de Janeiro (Rio de Janeiro, RJ, Brazil). He received his B.Sc. and M.Sc. degrees in Electronical Engineering from the Federal University of Rio de (Rio de Janeiro, RJ, Brazil) in 2010 and 2011 respectively. His current research interests include network virtualization, big data and cloud computing.

Luís Henrique M. K. Costa received the Electronics Engineer degree and the M.Sc. in electrical engineering from the Federal University of Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, in 1997 and 1998, respectively, and the Dr. degree from the University Pierre et Marie Curie (Paris 6), Paris, France, in 2001. Luis spent 2002 as a Post-Doctoral Researcher with Laboratoire d'Informatique de Paris 6, Paris. Then, Luís was awarded a research grant from CAPES (an agency of the Brazilian Ministry of Education), and joined COPPE/UFRJ. Since August 2004, he has been an Associate Professor with UFRJ. His major research interests are in the area of routing, especially on wireless networks, group communication, quality of service, multicast, and large-scale routing.

Otto Carlos M. B. Duarte received the Electronics Engineer degree and the M.Sc. degree in electrical engineering from Universidade Federal do Rio de Janeiro, Brazil, in 1976 and 1981, respectively, and the Dr. Ing. degree from ENST/Paris, France, in 1985. Since 1978, he has been a Professor with UFRJ. His major research interests are in QoS guarantees, security and big data.