Towards a Pluralist Internet Using a Virtual Machine Server for Network Customization

Rafael dos Santos Alves*, Miguel Elias M. Campista*,◇,
Luís Henrique M. K. Costa*, and Otto Carlos M. B. Duarte*

* Universidade Federal do Rio de Janeiro
GTA/COPPE-PEE/Del-Poli
Rio de Janeiro, Brasil
{santos,miguel,luish, otto}@gta.ufrj.br

◇ UPMC Sorbonne Universités
NPA/LIP6
Paris, France
Miguel.Campista@lip6.fr

ABSTRACT

The Internet success is frequently credited to its basic pillars, the end-to-end argument and the TCP/IP protocol stack. Nevertheless, this simple architecture does not facilitate the addition of new services such as mobility, security, and quality of service support. As a consequence, virtual networks are often being used as a tool to experiment with new protocol architectures. In this work, we propose a Virtual Machine Server (VMS) to manage virtual networks that are customized upon user needs. The proposed VMS uses the idea of machine virtualization within the networking context. Instead of managing virtual machines, the VMS manages virtual routers and uses them to build virtual networks. The result of the proposed approach is the possibility to experiment with pluralist architectures for the future Internet, which are neither as radical as the "clean-slate" approach nor as conservative as the evolutionary approach. In addition, the proposed VMS is flexible enough to allow the interaction of users, or intelligent software agents, with the network resources. The server is implemented using web services and a prototype with Xen stations is currently operational. Our experimental results show that the operation of the VMS is simple and motivates larger implementations.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Computer Communication Networks—*Network Architecture and Design*; C.5.5 [**Computer Systems Organization**]: Computer System Implementation—*Servers*

AINTEC'12, November 14–16, 2012, Bangkok, Thailand.

General Terms

Design, Experimentation.

Keywords

Virtual Machine Server, Virtualization, Future Internet.

1. INTRODUCTION

Many future Internet challenges are related to emerging requirements not originally foreseen, such as security, mobility, and quality of service. As of today, the networking community has been trying to find solutions to those requirements but, many in this community agree that the Internet architecture itself should be revisited and projects on this theme are flourishing. A consensus has not yet been achieved but two opposite views can be identified [1]: the "clean-slate" [2] and the evolutionary approaches. The former proposes the Internet recreation considering all the experiences to date, whereas the latter argues that a drastic change is not economically viable and the Internet must keep continuously evolving. Both approaches involve huge challenges. The clean-slate approach aims at finding a totally new architecture that could support as many service requirements as possible, whereas the evolutionary approach has to come up with a proposal that also considers conflicting requirements, but builds on top of the current TCP/IP stack.

In this work, we do not engage into the debate of which one, evolutionary or clean-slate, is the most suitable architecture for the future Internet. Instead, we aim at using virtualization as the basis for a pluralist architecture, where multiple virtual networks coexist [3] and can be experimented with, no matter if they are clean-slate or evolutionary. By virtualizing the physical substrate, the Internet becomes one of the virtual networks and wide-scale experiments can be conducted without affecting production traffic. This characteristic renders the pluralist architecture backwards compati-

bility, which cannot be achieved with the clean-slate approach alone.

Enabling the pluralist architecture, however, requires a practical tool to manage the virtual networks. In this work, we propose and implement a Virtual Machine Server (VMS)[1] to support the creation of customized virtual networks upon users' requests. These virtual networks run in parallel using the same principles of machine virtualization in the networking case. The VMS simplifies virtual networking management tasks such as creation, deletion, or initialization of a virtual network, and provides an interface to different sorts of clients. These clients can be accessed via a web interface used by users, ranging from skilled personal to home users. The VMS provides support for different virtualization platforms. Moreover, it can also interact with users to build virtual networks according to their requirements. We analyze different possibilities for virtual network creation, which introduces a tradeoff between flexibility and performance as shown in our experiments.

Our main contributions are (*i*) the proposal and implementation of a virtual machine server to manage virtual networks; (*ii*) the utilization of open programming interfaces, general libraries, and standardized protocols to manage virtual networks over different physical substrates; and (*iii*) the reinforcement of the pluralist architecture as an alternative for the future Internet. In this work, however, we do not aim at proposing a new Internet. Instead, we look forward moving one step ahead, showing that the pluralist architecture must be considered as, at least, an intermediate approach. We conduct experiments in a testbed to evaluate the time to create a virtual router and the processing capacity required from clients, considering possible virtual router image transfers and different physical machines.

This paper is organized as follows. In Section 2, we overview machine virtualization. Section 2.1 introduces the network model as well as describes the proposed VMS architecture and implementation. Section 3 shows our results and Section 4 presents related work. Finally, Section 5 concludes this work and identifies future directions.

2. VIRTUALIZATION OVERVIEW

Machine virtualization decouples from the above layers the underlying physical machine resources (CPU, memory, and I/O devices), inserting a virtualization layer in-between. This layer manages the access from the different virtual machines to the physical resources, hiding from them the underlying resource sharing. Machine virtualization is highly flexible allowing different operating systems to run in parallel. The tradeoff, however, is the software overhead needed to arbitrate above-layer calls for the shared resources. New technologies to assist virtualization at the hardware level, such as direct I/O, are being developed to improve performance.

[1]Available at `http://www.gta.ufrj.br/~santos/vms`.

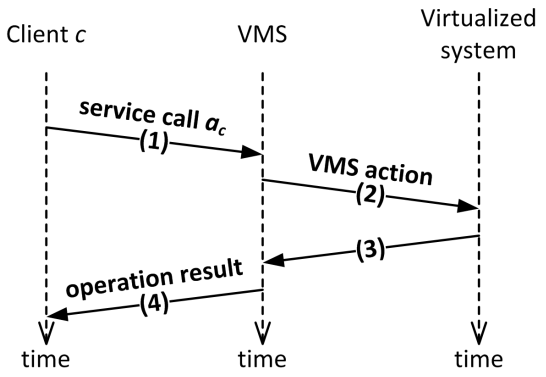

Figure 1: Evolution in time of the system operation.

We define a virtual network as a set of virtual machines playing the role of routers and the virtual links connecting them. Virtualization, then, must be able to correctly identify packets to and from each virtual network. An example of utilization is the possibility to setup on demand a customized wireless network by virtualizing access points.

2.1 Virtual Machine Server

Figure 1 shows the system evolution over time. The first step, represented in the figure by (1), consists of a client request for a service to the Virtual Machine Server (VMS). Client c sends a service call a_c to the VMS. To this end, it creates a message containing the service required and sends it by using a communication protocol encapsulating an XML (eXtensible Markup Language) file, as represented in the figure by (2). Upon receiving the message, the VMS identifies the virtualized system, as indicated in the received message, and sends a request for an action to the corresponding system. The VMS uses the management library provided by the virtualized system to interact with it. The command issued by the VMS triggers a sequence of actions which will lead the virtualized system to a state according to the request of client c. After step (2), in step (3), the virtualized system sends a response to the VMS indicating the final operation result. The VMS finally encapsulates the response in a meta-data format and sends it to the client, as represented by step (4).

2.2 Implementation overview

The main task of the Virtual Machine Server (VMS) is the management of virtual networks upon users' requests using the available physical infrastructure. The VMS must also yield virtual link configuration to guarantee network connectivity. Using the VMS, users can have virtual networks customized according to their ap-

plication requirements. The virtual network is then created from pre-built virtual router disk images, which combine the best possible settings and protocols to fit the needs of the user. The level of freedom users can deal with is controlled by the VMS programming interface, which is available for users through VMS clients. The VMS operation on the physical substrate is transparent for users, who can only have access to the information provided by VMS clients.

The VMS has two main modules: the database and the virtual machine image repository. The database stores physical and virtual network status, such as the available physical resources and traces that allow measuring the performance of virtual networks. The VM image repository maintains pre-built virtual router images to be used according to users' requirements. For example, a user may request an image stored in the repository and have it transferred to chosen physical routers in order to create a virtual network. Note that the repository can be moved to each physical router or to a location where they can directly access through the local network. This alternative may impose the existence of multiple copies of the same images throughout the network and limits the virtual network to a local area (LAN), revealing a tradeoff between flexibility and performance. Keeping the repository in the VMS simplifies new virtual router image creation and does not limit virtual networking to a local scope. Nevertheless, it has to cope with transferring virtual router images to physical routers before creating the virtual network. An intermediate solution could cache in physical routers the most used images, computed according to statistics on the VMS database. Creating a virtual network also involves with physical to virtual network interface mapping and network address configuration.

2.3 Service primitives

In addition to the basic services, the proposed VMS also implements services to control the amount of physical resources (memory and CPU) allocated to a virtual router, if it becomes a bottleneck. Table 1 summarizes the services provided by the VMS.

The service `createVirtualMachine` creates virtual machines. It uses the parameters passed as input by the service requisition to create a virtual node on a specific physical network node. This service has two variations. The VMS can send the file used as virtual disk to the physical machine which will host the new virtual machine. Alternatively, the virtual disk can be accessible to the physical machine which will host the new virtual machine. The service `createVirtualNetwork` creates a set of virtual nodes in network physical machines. Additionally, to guarantee that the virtual machines created compose a network, the mapping between the indicated physical interface and the new virtual interface is done, besides the configuration of the network addresses.

When a virtual machine creation is requested with the variation for virtual disk transfer, the details concerning the operating system to be used must be defined. Some of the VMS services provide the clients with information on the virtual routers available. Service `getAvailableOSes` presents the available operating systems (Linux, Windows, MacOs, etc.), service `getAvailableArch` presents the available architectures, and service `getAvailableKernelVersions` presents the kernel versions available.

The service `destroyVirtualMachine` can be used to destroy a virtual machine, for instance, when the network in which it was created to operate is no longer needed. In some cases, the virtual machine will have to be turned off to be rebooted afterwards. The service `shutdownVirtualMachine` can be used with this goal. The services `getPhysicalServerStatus` as well as `getVirtualMachineStatus` are used to obtain general information, such as memory and number of processors, regarding a physical or a virtual machine, respectively. This information can be used by a knowledge plane during a decision making process. The service for migration (`migrateVirtualMachine`) can be used to move a virtual machine from a physical node to another, when a given physical node is overcharged, for instance.

In multitask operating systems, CPU schedulers are used to divide processing resources among processes. Similarly, Xen uses CPU schedulers to share resources among virtual machines. Xen originally implemented three different schedulers, the Credit Scheduler [4], BVT (Borrowed Virtual Time) [5], and SEDF (Simple Earliest Deadline First) [6]. Currently, the default scheduler of Xen is the Credit Scheduler. The scheduler used by the virtual machine can be get by the `getVirtualMachineSchedulerType` service. In addition, the scheduler parameters of a virtual machine can be obtained via the service `getVirtualMachineSchedulerParameters` and can be modified via the service `setVirtualMachineSchedulerParameters` of VMS.

Initially, the virtual machine server is totally unaware of the physical machines which it has access to. To guarantee that the virtual machine server has knowledge about these nodes, the service `registerNodes` allows that a given node is registered in the server for future administration, i.e., the name, the public key, and the IP addresses are sent to the server which stores this information locally. The registered nodes can be accessed by using the service `getRegisteredNodes`.

2.4 VMS client

In this work, we also developed a client class, called `HorizonXenClient`, to facilitate the interaction with the VMS. All the services offered by the VMS are called

Table 1: Services provided by the Virtual Machine Server.

Service	Description
`createVirtualMachine`	Service for virtual machine creation
`createVirtualNetwork`	Service for virtual network creation
`destroyVirtualMachine`	Service for virtual machine destruction
`getAvailableArch`	Service for obtaining available virtual machine architectures
`getAvailableOSes`	Service for obtaining available operating systems for virtual machines
`getAvailableKernelVersions`	Service for obtaining available kernel version for virtual machines
`getPhysicalServerStatus`	Service for obtaining information of a physical machine
`getRegisteredNodes`	Service for obtaining the list of registered nodes
`getVirtualMachineSchedulerParameters`	Service for obtaining virtual machine scheduling parameters
`getVirtualMachineSchedulerType`	Service for obtaining the virtual machine scheduling type
`getVirtualMachineStatus`	Service for obtaining a given virtual domain
`migrateVirtualMachine`	Service for virtual machine migration
`registerNodes`	Service for node registration
`sanityTest`	Service for server sanity check
`setVirtualMachineSchedulerParameters`	Service to set virtual machine scheduling parameters
`shutdownVirtualMachine`	Service to shutdown a virtual machine

by the client using specific methods. These methods are provided as the client class API, which can be used to build software systems capable to interact with the VMS. These software systems can be as simple as a web page with a floating menu for human interaction or even more sophisticated systems such as intelligent agents for autonomous operation. Listing 1 summarizes the API provided by the developed client class. Note that all methods return an object of the class `OMElement`, which are used as the content of SOAP (Simple Object Access Protocol) messages. The `OMElement` object returned also reports execution failures to clients.

The `createVirtualMachine` method has as input parameters the virtual router name (`vmName`), the name of the physical router or its IP address (`phyServer`), the IP address of the virtual router (`vmIP`), and the RAM size (`vmRAM`).

Concerning the parameters employed by the methods described in Listing 1, the `vmName` refers to the virtual machine name. This name will be the name accessible via Xen system administration resources and shall be used for future interactions with the VMS. The parameter `phyServer` refers to the name externally accessible via DNS of the physical node or its IP address. The parameters `vmIP` and `vmRAM` are related, respectively, to the IP address and to the RAM size destined to the new virtual domain. Parameters `Weight` and `Cap` are used to configure the Credit Scheduler [7].

There are also specific parameters for virtual machine migration: `sourcePhyServer` and `destPhyServer` define, respectively, the source and destination physical nodes of the virtual machine to be migrated; the parameter `live`, which can receive boolean values, defines if the migration should be conducted live, i.e. without the interruption of the virtual machine operation.

Some services can operate upon a set of physical and virtual machines. In these cases, the parameters expected are vectors and the semantics are similar to those already pointed out. The VMS also implements a sanity check method which has a `testString` as parameter. This parameter defines the string which will be part of the test message body and will be sent back by the server, in case it is not working properly.

It is worth mentioning that the utilization of the `HorizonXenClient` class is not mandatory. Instead, the client can be build without using this library and even without limiting the programming language. Because it is based on a web service, it allows such flexibility. The only requirement is the support for SOAP and the use of a valid XML container for correct parsing. This provides enough flexibility to the development of different clients, which could replace the utilization of the `HorizonXenClient`.

2.5 Implementation details

The virtualized systems managed by the VMS have as their only requirement to run a virtualization platform, which must provide public programming interfaces and support open-source libraries for management. We use the `Libvirt` library v.0.7.5, which supports different programming languages and virtualization platforms, e.g. Xen, VMware, OpenVZ, QEMU, since they all provide the same set of basic management tasks. Thus, the VMS supports different virtualization platforms. An important remark has to be done on the location of the virtual router hard disk image. This file must contain the operating system and all functions required by users to provide their service. For example, if the virtual

Listing 1: API provided by the `HorizonXenClient` class.

```
public OMElement
    createVirtualMachinePayload(String
    phyServer, String vmName, String vmIP,
    String vmRAM);
public OMElement
    createVirtualNetworkPayload(Vector<
    String> phyServers, Vector<String>
    VMNames, Vector<String> IPs, Vector<
    String> RAMs, Vector<String>
    netInterface);
public OMElement
    destroyVirtualMachinePayload(String
    phyServer, String vmName);
public OMElement
    getPhysicalServerStatusPayload(String
    phyServer);
public OMElement getRegisteredNodesPayload
    ();
public OMElement
    getVirtualMachineSchedulerParameters-
    Payload (String phyServer, String
    VMName);
public OMElement
    getVirtualMachineStatusPayload (String
    phyServer, String vmName);
public OMElement
    migrateVirtualMachinePayload(String
    sourcePhyServer, String destPhyServer,
    String vmName, String live);
public OMElement registerNodesPayload(
    Vector<PhysicalServer> phyServers);
public OMElement sanityTestPayload(String
    testString);
public OMElement
    setVirtualMachineSchedulerParameters-
    Payload (String phyServer, String
    VMName, String Weight, String Cap);
public OMElement
    shutdownVirtualMachinePayload(String
    phyServer, String vmName);
```

router will run IPv4 and RIP (Routing Information Protocol), the image must be built from an operating system supporting these protocols. We use Xen [8] as our virtualization platform since it is open-source, provides public APIs (`XenAPI`), and has many adepts worldwide.

The VMS is written in Java and uses the `Tomcat` v.6 as the web server. Both VMS and an implemented client use the open-source library `Axis2` v.1.5.1 to communicate. Users interact with the client, which invokes methods to create XML containers with their requests. `Axis2` implements the Simple Object Access Protocol (SOAP) to encapsulate XML containers in SOAP messages, which will be sent from the client to the VMS and vice-versa. All services provided by the VMS are publicly available for clients and are accessible via an object defined in `Axis2`, the Object Model Element (`OMElement`). The `OMElement` is instantiated in the client, inserted in the XML container with the respective user request, and sent to the VMS using SOAP.

If a new service not yet implemented is required, adding it is simple. Each service is implemented as a method of the main class `VirtualMachineServer` of the VMS. Every service must be implemented as a public method, which receives an object of the class `OMElement` and returns another object of the same `OMElement` class. The `OMElement` class is offered by the Axis2 library and has as its main characteristic the capacity to encapsulate XML messages, i.e., after converted to a string, an object `OMElement` becomes a tag of a XML message. In this case, the element received as a parameter is the content of a SOAP message and the `OMElement` received back is also the content of a SOAP message sent as a response by the Virtual Machine Server.

3. EXPERIMENTS

Our testbed is composed of four PCs connected via a switch in a star topology, as depicted in Figure 2. One of them is the VMS (`vms`) whereas the others (`xen1`, `xen2`, and `xen3`) are physical routers with the hardware listed in Table 2. Another PC is used as a client.

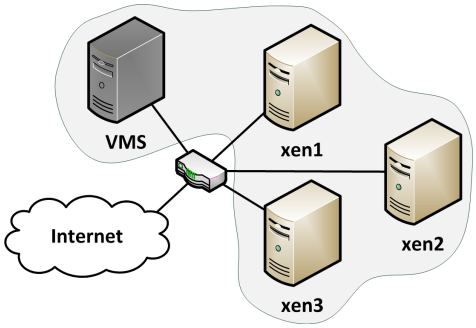

Figure 2: Testbed topology.

3.1 Results

We create a JAR (Java ARchive) and a JSP (Java-Server Pages) client to permit service requests from command lines and web pages, respectively. We show the average result and the standard deviation of each experiment.

Table 3 presents the time needed to create a virtual router on physical routers, considering the possibility of transferring or not the disk image. Although not transferring the disk image is more efficient, it limits the virtual network to a local scope. Results show that the total creation time is less than one minute with image transfer and less than $10\,s$, otherwise. Both results are encouraging since they can still be improved using specific hardware. Considering the whole process with image transfer comprised of the time for client-VMS

Table 2: Configuration of the testbed machines.

PC	Architecture	Processor	RAM	Kernel
vms	i386	Core2 2.13 GHz	2 GB	2.6.30-2
xen1	amd64	Core2 Duo 2.53 GHz	4 GB	2.6.32-5-xen
xen2	i386	Celeron 2.8 GHz	3 GB	2.6.32-5-xen
xen3	i386	Pentium4 HT 3.4 GHz	2 GB	2.6.32-5-xen

Table 3: Virtual router creation time in seconds.

PC	With image transfer	Without image transfer
xen1	42.20±1.62	5.72±0.37
xen2	43.77±1.35	6.86±0.25
xen3	46.62±5.80	7.05±1.74

communication, disk image transfer, and virtual router initialization, we have separately measured the time for image transfer. Figure 4 shows the time elapsed for image transfer, which represents a large amount of the total time. Comparing the performance of the different hardware, we note that the best result is always achieved with the best configuration (`xen1`).

Table 5 presents the processing time consumed by the client when creating a virtual router on each machine. We observe that the amount of CPU resources required is similar and low, independent of the hardware used. As a consequence, devices with limited processing power can also be used, including smartphones and intelligent agents.

3.2 Discussion

Considering the number of virtual routers per virtual network equal to the number of virtual networks (n), a virtual network creation is finished after creating all its virtual routers in sequence or in parallel. In the first case, virtual router creation starts only after the previous one, leading to a $\Theta(n)$ problem; whereas in the second case, the virtual router creation starts simultaneously, leading to a $\Theta(1)$ problem. Combining the Internet hierarchical organization with the parallel approach, the virtual networking creation problem scales in a log-based fashion. In the first step, the VMS creates in parallel the virtual routers within its local network, including border routers. The border routers repeat the same procedure within their networks on behalf of the VMS, as a second step. Thus, the procedure is recursively repeated until the virtual network is complete. We assume that the VMS is aware of the available physical resources, otherwise we should also consider the virtual router allocation problem [9, 10].

4. RELATED WORK

New future Internet architectures, following either evolutionary or clean-slate approaches, have been proposed. On the one hand, proposals based on the evolutionary approach are less radical and, therefore, have been used in the Internet so far, e.g. DNS (Domain Name System) [11], NAT (Network Address Translation) [12], IPv6 [13]. The clean-slate approach [2], on the other hand, requires deeper modifications into the current Internet. The role-based architecture is an example which does not rely on the widely known Internet multi-layered architecture. The goal is to allow improvements without requiring cross-layer violations. Hence, in the role-based architecture, modules are employed to replace layers to permit modular protocol design. The major difference compared with the multi-layered counterpart is the existence of hierarchical levels among roles. In addition, it is worth mentioning that the roles are built based on blocks of standardized structures to allow the creation of well-defined services. The Content-based Networking (CBN) [14] is another architecture based on the clean-slate approach, which assumes that Internet users are not interested from whom or from where they can obtain a given content. Instead, they are only interested in timely response for content requests. Following the CBN idea, the Internet would change from station centered architecture, to data centered. DONA (Data-Oriented Network Architecture) [15] is a content-based proposal which does not use DNS requisitions for destination name resolution into IP address. In DONA, anycast primitives based on names are inserted above the network layer for content acquisition and, thus, IP addresses are not used. The VMS can store evolutionary- and clean-slate-based proposals for experiments using the pluralist network.

Feamster *et al.* propose CABO (Concurrent Architectures are Better than One) [16] which already used virtual machines to build pluralist architectures. Hence, multiple virtual networks share the same physical substrate and each one can use a different configuration or even a distinct protocol stack. The main motivation behind CABO is to permit ISPs (Internet Ser-

Table 4: Image transfer time in seconds for virtual machine creation with disk transfer.

PC	Image transfer
xen1	35.21±0.13
xen2	35.47±0.41
xen3	36.52±4.99

Table 5: Processing time consumed by the client in seconds.

PC	With image transfer	Without image transfer
xen1	0.79±0.01	0.79±0.01
xen2	0.82±0.03	0.79±0.01
xen3	0.81±0.02	0.79±0.01

vice Providers) to offer differentiated services to their costumers, which is not possible today. Because ISPs do not have end-to-end routers between their Internet users, service and infrastructure providers are separated. In this case, infrastructure providers offer computational and network resources to service providers through commercial agreements. A service provider can use resources from different infrastructure providers and, therefore, can build an end-to-end path with accessible routers. This allows the provision of differentiated services to final users. The VMS allows users to build different networks based on a tool for creation, control, management, and deletion.

5. CONCLUSIONS AND FUTURE WORK

In this work, we proposed a Virtual Machine Server (VMS) for virtual networking management. Our results are promising and revealed a tradeoff between flexibility and performance, i.e. transferring virtual router images before creating a virtual network provides more flexibility but incurs in higher delay. As a future work, we plan to use real data for performance analysis and to extend our testbed.

6. ACKNOWLEDGMENTS

The authors would like to thank the Brazilian agencies CAPES, CNPq, Faperj, FINEP, and FUNTTEL and the French agency ANR for their support. In addition, Miguel Elias M. Campista would like to thank NPA (Network and Performance Analysis) Laboratory in UPMC Sorbonne Universits and LINCS (Laboratory of Information, Network and Communication Sciences) for hosting him.

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] J. Rexford and C. Dovrolis, "Future Internet architecture: clean-slate versus evolutionary research," *Communications of the ACM*, vol. 53, no. 9, pp. 36–40, Sep. 2010.

[2] D. Clark, R. Braden, K. Sollins, J. Wroclawski, D. Katabi, J. Kulik, X. Yang, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa, "New Arch: Future generation Internet architecture," USC Information Sciences Institute Computer Networks Division, MIT Laboratory for Computer Science and International Computer Science Institute (ICSI), Tech. Rep., Aug. 2004.

[3] N. M. M. K. Chowdhury and R. Boutaba, "Network virtualization: state of the art and research challenges," *IEEE Communications Magazine*, vol. 47, no. 7, pp. 20–26, Jul. 2009.

[4] Citrix, "Credit-based CPU scheduler," http://wiki.xensource.com/xenwiki/Credit-Scheduler, 2007, acessado em abril de 2011.

[5] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler," in *Proceedings of the ACM symposium on Operating systems principles (SOSP)*, 1999, pp. 261–276.

[6] I. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1280–1297, Sep. 1996.

[7] L. Cherkasova, D. Gupta, and A. Vahdat, "When virtual is harder than real: Resource allocation challenges in virtual machine based IT environments," HP Laboratories Report No. HPL-2007-25, Tech. Rep., 2007.

[8] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, L. Mathy, and T. Schooley, "Evaluating Xen for router virtualization," in *ICCCN*, Aug. 2007, pp.

1256–1261.

[9] G. Alkmin, D. M. Batista, and N. L. S. da Fonseca, "Optimal mapping of virtual networks," in *IEEE Globecom*, Dec. 2011, pp. 1–6.

[10] F. Gu, C. Xie, M. Peng, iek avdar, S. Khan, and N. Ghani, "Virtual overlay network scheduling," *IEEE Communications Letters*, vol. 15, no. 8, pp. 893–895, Aug. 2011.

[11] P. Mockapetris, "Domain names - implementation and specification," RFC 1035, Nov. 1987.

[12] K. Egevang and P. Francis, "The ip network address translator (NAT)," RFC 1631, May 1994.

[13] R. Hinden and S. Deering, "Internet protocol version 6 (IPv6) addressing architecture," RFC 3513, Apr. 2003.

[14] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," in *Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, Dec. 2009, pp. 1–14.

[15] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, Aug. 2007.

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