Universidade do Estado do Rio de Janeiro

Centro de Tecnologia e Ciência

Faculdade de Engenharia

Leopoldo Alexandre Freitas Mauricio

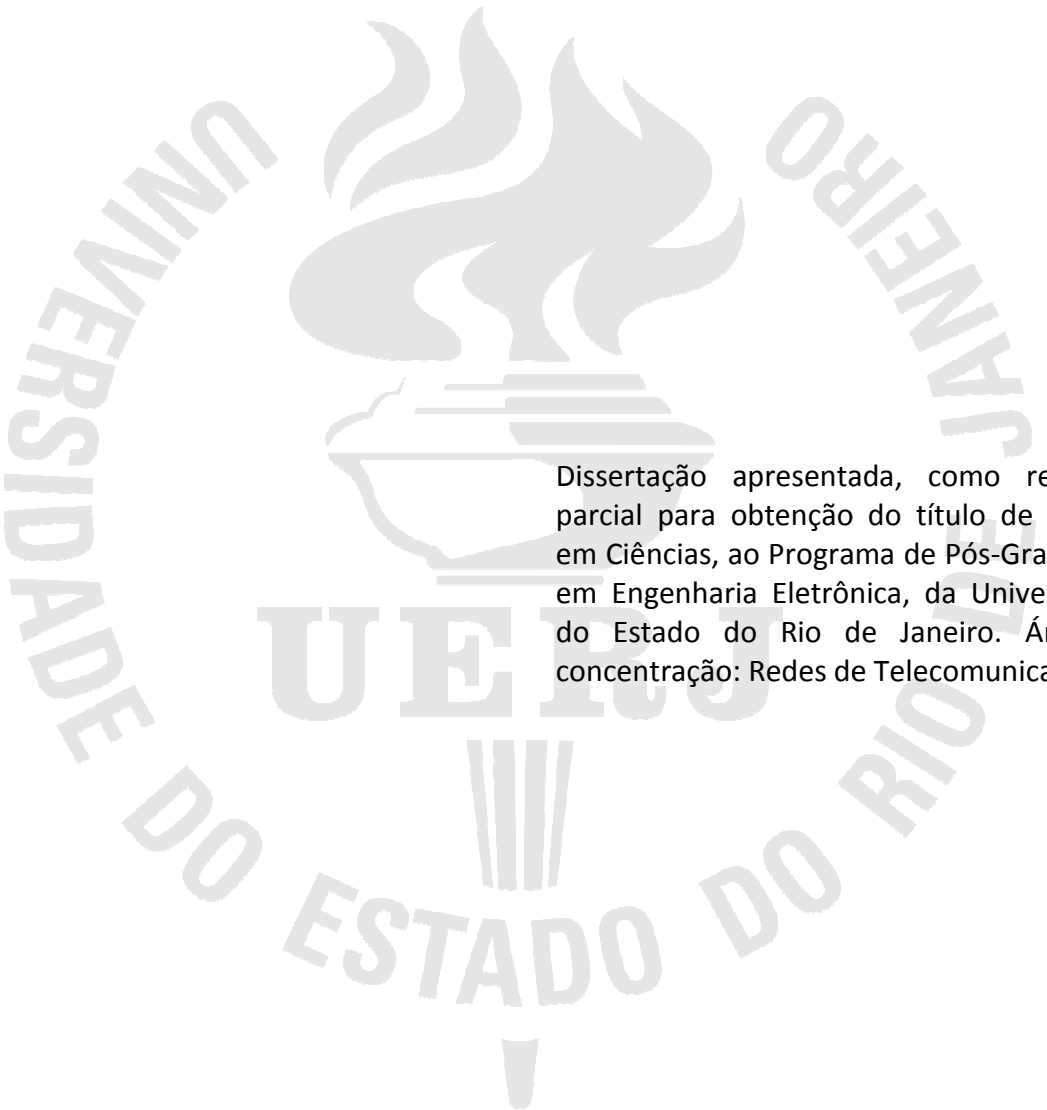
Avaliação de desempenho de plataformas de virtualização de redes

Rio de Janeiro

2013

Leopoldo Alexandre Freitas Mauricio

Avaliação de desempenho de plataformas de virtualização de redes



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações

Orientador: Prof. Dr. Marcelo Gonçalves Rubinstein

Rio de Janeiro

2013

CATALOGAÇÃO NA FONTE

UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

M456 Mauricio, Leopoldo Alexandre Freitas.
Avaliação de desempenho de plataformas de virtualização de redes / Leopoldo Alexandre Freitas Mauricio. – 2013.
78f.

Orientador: Marcelo Gonçalves Rubinstein.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Eletrônica. 2. Roteamento (Adminsitração de redes de computadores) - Dissertações. I. Rubinstein, Marcelo Gonçalves. II. Universidade do Estado do Rio de Janeiro. III. Título.

CDU 004.715

Assinatura

Data

Leopoldo Alexandre Freitas Mauricio

Avaliação de desempenho de plataformas de virtualização de redes

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre em Ciências, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Redes de Telecomunicações.

Aprovado em: 27 de agosto de 2013.

Banca Examinadora:

_____

Prof. Dr. Marcelo Gonçalves Rubinstein (Orientador)

Faculdade de Engenharia - UERJ

_____

Prof. Dr. Alexandre Sztajnberg

Faculdade de Engenharia - UERJ

_____

Prof. Dr. Otto Carlos Muniz Bandeira Duarte

Universidade Federal do Rio de Janeiro – UFRJ

Rio de Janeiro

2013

Dedico esse trabalho primeiro ao meu Deus, razão da minha existência, que adestras as minhas mãos para me fazer vencer as minhas guerras. E à Sophia Beatriz, maior presente de Deus na minha vida. Sophia, você é sempre será o meu amor, minha paixão e minha vida.

AGRADECIMENTOS

Agradeço a minha querida esposa Beatriz Mauricio e a minha filha Sophia Mauricio pela compreensão e ajuda nos diversos momentos em que precisei me ausentar das atividades familiares para dedicar mais tempo a pesquisa e estudo. Tem sido maravilhoso viver cada dia com vocês. Agradeço aos meus pais Luiz Mauricio e Solange Mauricio, que com palavras de sabedoria e orações estão sempre lutando pelo sucesso de seus filhos, fortalecendo nossos objetivos e nos animando. Agradeço ao Programa de Pós-Graduação em Engenharia Eletrônica da UERJ pela oportunidade que me deram de fazer parte de um maravilhoso ambiente de pesquisa; principalmente aos professores Marcelo e Alexandre, pela total disponibilidade e auxílio prestado ao longo de todo o curso. Vocês nunca permitiram a ociosidade e mantiveram um elevado nível de cobrança, que certamente contribui para a elevação da avaliação do nosso Programa de Pós-Graduação. Agradeço ao meu orientador Marcelo G. Rubinstein, pela paciência e maravilhosa orientação. Por estar sempre presente, guiando o trabalho realizado com maestria. Agradeço aos membros da banca pela presteza e atenção que demonstraram ao aceitar participar da avaliação desse trabalho. Enfim, a todos os que de alguma forma me ajudaram a chegar até aqui; muito obrigado!

Deus é a minha fortaleza e a minha força, e ele perfeitamente desembaraça o meu caminho. Faz ele os meus pés como os das cervas, e me põe sobre as minhas alturas. Instrui as minhas mãos para a peleja, de maneira que um arco de cobre se quebra pelos meus braços.

2 Samuel 22:33-35

RESUMO

MAURICIO, Leopoldo Alexandre Freitas. **Avaliação de desempenho de plataformas de virtualização de redes**. 2013. 78 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

O objetivo desta dissertação é avaliar o desempenho de ambientes virtuais de roteamento construídos sobre máquinas x86 e dispositivos de rede existentes na Internet atual. Entre as plataformas de virtualização mais utilizadas, deseja-se identificar quem melhor atende aos requisitos de um ambiente virtual de roteamento para permitir a programação do núcleo de redes de produção. As plataformas de virtualização Xen e KVM foram instaladas em servidores x86 modernos de grande capacidade, e comparadas quanto a eficiência, flexibilidade e capacidade de isolamento entre as redes, que são os requisitos para o bom desempenho de uma rede virtual. Os resultados obtidos nos testes mostram que, apesar de ser uma plataforma de virtualização completa, o KVM possui desempenho melhor que o do Xen no encaminhamento e roteamento de pacotes, quando o VIRTIO é utilizado. Além disso, apenas o Xen apresentou problemas de isolamento entre redes virtuais. Também avaliamos o efeito da arquitetura NUMA, muito comum em servidores x86 modernos, sobre o desempenho das VMs quando muita memória e núcleos de processamento são alocados nelas. A análise dos resultados mostra que o desempenho das operações de Entrada e Saída (E/S) de rede pode ser comprometido, caso as quantidades de memória e CPU virtuais alocadas para a VM não respeitem o tamanho dos nós NUMA existentes no hardware. Por último, estudamos o OpenFlow. Ele permite que redes sejam segmentadas em roteadores, comutadores e em máquinas x86 para que ambientes virtuais de roteamento com lógicas de encaminhamento diferentes possam ser criados. Verificamos que ao ser instalado com o Xen e com o KVM, ele possibilita a migração de redes virtuais entre diferentes nós físicos, sem que ocorram interrupções nos fluxos de dados, além de permitir que o desempenho do encaminhamento de pacotes nas redes virtuais criadas seja aumentado. Assim, foi possível programar o núcleo da rede para implementar alternativas ao protocolo IP.

Palavras-chave: Redes virtuais; OpenFlow; KVM; Xen; Ambientes virtuais de roteamento; Máquinas x86; Plataformas de virtualização; Roteadores virtuais; NUMA.

ABSTRACT

MAURICIO, Leopoldo Alexandre Freitas. **Avaliação de desempenho de plataformas de virtualização de redes**. 2013. 78 f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2013.

The aim of this work is to evaluate the performance of routing virtual environments built on x86 machines and network devices existing on the Internet today. Among the most widely used virtualization platforms, we want to identify which best meets the requirements of a virtual routing to allow programming of the core production networks. Virtualization platforms Xen and KVM were installed on modern large capacity x86 machines, and they were compared for efficiency, flexibility and isolation between networks, which are the requirements for good performance of a virtual network. The tests results show that, despite being a full virtualization platform, KVM has better performance than Xen in forwarding and routing packets when the VIRTIO is used. Furthermore, only Xen had isolation problems between networks. We also evaluate the effect of the NUMA architecture, very common in modern x86 servers, on the performance of VMs when lots of memory and processor cores are allocated to them. The results show that Input and Output (I/O) network performance can be compromised whether the amounts of virtual memory and CPU allocated to VM do not respect the size of the existing hardware NUMA nodes. Finally, we study the OpenFlow. It allows slicing networks into routers, switches and x86 machines to create virtual environments with different routing forwarding rules. We found that, when installed with Xen and KVM, it enables the migration of virtual networks among different physical nodes, without interruptions in the data streams, and allows to increase the performance of packet forwarding in the virtual networks created. Thus, it was possible to program the core network to implement alternatives to IP protocol.

Keywords: Virtual networks; OpenFlow; KVM; Xen; Virtual routing environments; x86 machines; Virtualizations platforms; Virtual routes; NUMA.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACL	*Access List*
API	*Application Programming Interface*
BE	*Back-End*
CPD	Centro de Processamento de Dados
CPU	*Central Processing Unit*
Dom0	*Domain Zero*
DomU	*Domain U*
E/S	Entrada e Saída
FE	*Front-End*
GB	*Gigabyte*
GENI	*Global Environment for Network Innovation*
GTA	Grupo de Teleinformática e Automação
HPCC	*High-Performance Computing Cluster*
IaaS	*Infrastructure as a Service*
IDPS	*Intrusion Detection and Prevention Systems*
IP	*Internet Protocol*
ISPs	*Internet Service Providers*
JVM	*Java Virtual Machine*
LXC	Linux *Container*
MAC	*Medium Access Control*
MB	*Megabyte*
NaaS	*Networking as a Service*
NAT	*Network Address Translation*
NUMA	*Non-Uniform Memory Access*
PaaS	*Platform as a Service*
PC	Plano de Controle
PD	Plano de Dados
QoS	*Quality of Service*
RAM	*Random Access Memory*
SO	Sistema Operacional

TB	*Terabyte*
TCAM	*Ternary Content Addressable Memory*
TCP	*Transmission Control Protocol*
UFRJ	Universidade Federal do Rio de Janeiro
UMA	*Uniform Memory Access*
Una ou U	Unidade de rack
Vlan	*Virtual Lan*
VM	*Virtual Machine*
XCP	*Xen Cloud Platform*

SUMÁRIO

INTRODUÇÃO

Atualmente milhões de pessoas e empresas estão acostumadas a utilizar a Internet como ferramenta de trabalho e entretenimento. De qualquer lugar que ofereça comunicação com a rede, é possível transmitir áudio e vídeo em tempo real, fazer videoconferências a partir de dispositivos móveis, realizar operações financeiras, comprar e vender produtos diversos, interagir com outras pessoas através de redes sociais, entre outras coisas. Estamos cada vez mais acostumados a utilizar as ferramentas e as facilidades providas pela Internet, e a quantidade de novas aplicações não para de aumentar. Entretanto, ela não foi originalmente desenvolvida para suportar todas essas tecnologias. A Internet nasceu nos anos 1970 para interligar usuários confiáveis com tráfego de dados sempre entre nós estacionários. Além disso, o TCP/IP, que é o modelo de referência de arquitetura da Internet, foi desenhado originalmente para ser extremamente simples e evitar complexidades no núcleo da rede [27]. Com um núcleo simples baseado no TCP/IP, a implementação de complexidades, eventualmente necessárias, ficou para as aplicações que rodam nas extremidades da rede. Esse modelo foi responsável pelo gigantesco sucesso da Internet; no entanto, diversos problemas difíceis de serem resolvidos também surgiram porque o núcleo da rede não estava preparado para prover soluções, por exemplo, para a presença de usuários não confiáveis, lidar com usuários móveis, suportar transmissões de áudio e vídeo em tempo real, fornecer qualidade de serviço (QoS), entre outras coisas [31].

Problemas que dificultam a evolução da Internet

As soluções necessárias para que nós móveis consigam operar sem problemas na Internet, por exemplo, é uma tarefa difícil para o protocolo IP devido à sua semântica de localização e identificação [31]. E se uma entidade da Internet desenvolver um mecanismo de entrega de pacotes fim-a-fim mais eficiente que o do TCP/IP, que aumente a segurança, melhore o desempenho no encaminhamento, flexibilize a semântica de localização e identificação do IP, etc., ela dependerá da concordância de todos os provedores de serviço (*Internet Service Providers* – ISPs) para que a nova solução seja implementada [42]. Como essa concordância é extremamente difícil de ser orquestrada, o IPv6 ainda é pouco implantado na rede, apesar de já ter sido desenvolvido desde 1994 para resolver o problema de escassez de endereços na Internet, permitir a autenticação e a privacidade, nativamente, através do IPsec, entre outras melhorias. Isso fica claro quando se observa a quantidade de tráfego IPV6 gerada no AMS-IX, que é um dos maiores pontos de troca de tráfego do mundo de acordo com o Nic.br [28]. Durante um ano, o maior tráfego IPv6 gerado foi inferior a 10 Gigabits por

segundo (Fig. 1). No mesmo dia, o tráfego IPv4 máximo de apenas um provedor de serviços brasileiro (globo.com) chegou a ser 10 vezes maior.

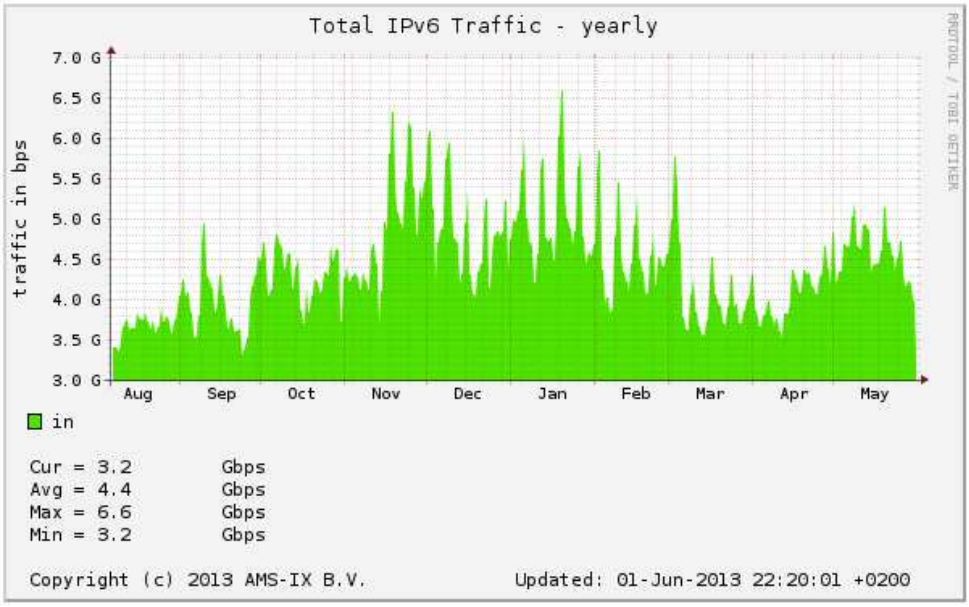


Figura 1 – Tráfego IPV6 total em um ano – junho de 2013, fonte: www.ams-ix.net.

Devido à descentralização e à divisão em múltiplas regiões administrativas autônomas que o protocolo IP implementa na Internet, é necessária uma coordenação global para que uma alteração no núcleo da rede possa ser implantada [42]. E esse é um problema estrutural do IP difícil de ser resolvido porque os ISPs sempre consideram as redes com tráfego de produção partes críticas da infraestrutura [34].

Diversos outros problemas que não são bem resolvidos pelo IP continuam existindo porque estão profundamente enraizados nas primeiras decisões de projeto da Internet. É o caso dos problemas de segurança – spam, ataques de negação de serviço, *phishing* etc. que ocorrem porque a noção de identidade utilizada na Internet é fraca, por permitir a uma entidade da rede realizar fraudes de endereços IP, nomes de domínios, endereços de e-mail e informações de roteamento. Mesmo conhecendo a causa raiz temos sido obrigados a conviver com problemas de segurança na rede, porque eles podem não ser resolvidos sem uma mudança de arquitetura fundamental, uma vez que não é tarefa fácil adaptar noções mais fortes de identidade na Internet de hoje [34].

Embora diversas mudanças incrementais tenham sido implementadas ao longo dos anos para minimizar os efeitos de alguns dos problemas que estão intimamente ligados às primeiras decisões de projeto da Internet, elas são resultado de pesquisas evolucionárias, onde a compatibilidade e a interoperabilidade com as versões anteriores de protocolos e arquitetura

legada são essenciais [42] [34]. Esta restrição faz com que boa parte das melhorias criadas a partir de pesquisas evolucionárias, tais como o NAT (*Network Address Translation*), os sistemas de detecção e prevenção de intrusão (*Intrusion Detection and Prevention Systems* – IDPS), as técnicas de *"caching"* etc., sejam na verdade soluções de contorno incrementais. Em um primeiro momento as técnicas oriundas de soluções de contorno permitiram o sucesso e o rápido crescimento da Internet, porém muitos dos novos requisitos não são atendidos adequadamente pelos atuais protocolos do núcleo da rede, porque a causa raiz dos problemas não foi resolvida. Dessa forma, as soluções de contorno acabam prejudicando a real evolução das tecnologias da Internet e comprometem, a longo prazo, o crescimento da rede e a implantação de novas aplicações.

Por isso, alguns pesquisadores acreditam que a abordagem correta para evitar a "ossificação" da rede, caracterizada pela falta de flexibilidade e habilidade para adaptar-se a novos requisitos, e pela dependência de soluções de contorno, depende do resultado de pesquisas que busquem soluções partindo do zero, conhecidas como pesquisas com abordagem *clean slate* [34]. Dessa maneira os projetistas e operadores da rede poderão rever os princípios que sustentam e causam problemas estruturais na Internet, sem estar ligados às soluções que existem hoje; e poderão definir de forma eficiente a arquitetura da Internet do futuro [5] [24].

Objetivos e motivação

O pluralismo é uma filosofia que defende a existência de diversas arquiteturas de rede sobre o mesmo substrato físico compartilhado como solução para o problema de ossificação da rede [42]. Dessa forma, redes virtuais com pilhas de protocolos diferentes podem ser executadas utilizando a mesma infraestrutura física da Internet, sem que uma interfira no funcionamento da outra. Assim, é possível partir do zero e definir para cada rede, de acordo com a necessidade de cada implementação, um formato de pacotes, um método de endereçamento e mecanismos de encaminhamento e roteamento exclusivos. Portanto, se o núcleo da Internet fosse virtualizado para suportar a ideia defendida pelos pluralistas, os operadores da rede conseguiriam construir redes virtuais exclusivas adequadas ao serviço ou aplicação que irá rodar sobre elas. Uma rede virtual que utiliza roteadores de borda de diferentes ISPs, por exemplo, poderia ser criada para resolver os problemas de QoS de um cliente, caso os ISPs decidam oferecer este tipo de serviço. Nos mesmos roteadores e ISPs poderia coexistir outra rede virtual criada para resolver os problemas de segurança do IP e suportar multicast ou para funcionar com IPv6, entre outras possibilidades [13]. O grande

ganho desse tipo de solução está no fato de que a pilha de protocolos em uso em cada uma dessas redes virtuais poderia ser completamente diferente da existente na rede real, que ainda poderia executar o TCP/IP e os protocolos da Internet que conhecemos. Isto está exemplificado na Figura 2, onde três redes virtuais, cada uma com uma pilha de protocolos diferente, foram configuradas nos roteadores de borda de um ISP. Enquanto a rede virtual 1 implementa o protocolo IP para permitir que o ISP continue suportando o tráfego da Internet atual, as redes 2 e 3 executam lógicas de endereçamento e roteamento completamente diferentes da que é implementada pelo IP.

A virtualização é o tema abordado nesse trabalho porque redes virtuais com necessidades simples podem operar sem serem sobrecarregadas com o custo de suporte a um serviço compartilhado que elas não utilizam [10]; e através de ambientes virtuais é possível realizar testes de novos protocolos e alternativas ao IP, utilizando a infraestrutura das redes reais, que possuem tráfego de produção, permitindo aos ISPs colocar em prática as novas soluções propostas para a rede. Todavia, para que as redes sobrepostas da Figura 2, defendidas pelos pluralistas, possam quebrar as barreiras para a inovação da Internet [13] é imprescindível garantir o isolamento entre elas, para que uma não interfira no funcionamento da outra. Só assim o núcleo da Internet atual poderá suportar com sucesso diversos tipos de redes funcionando em paralelo sobre o mesmo substrato físico.

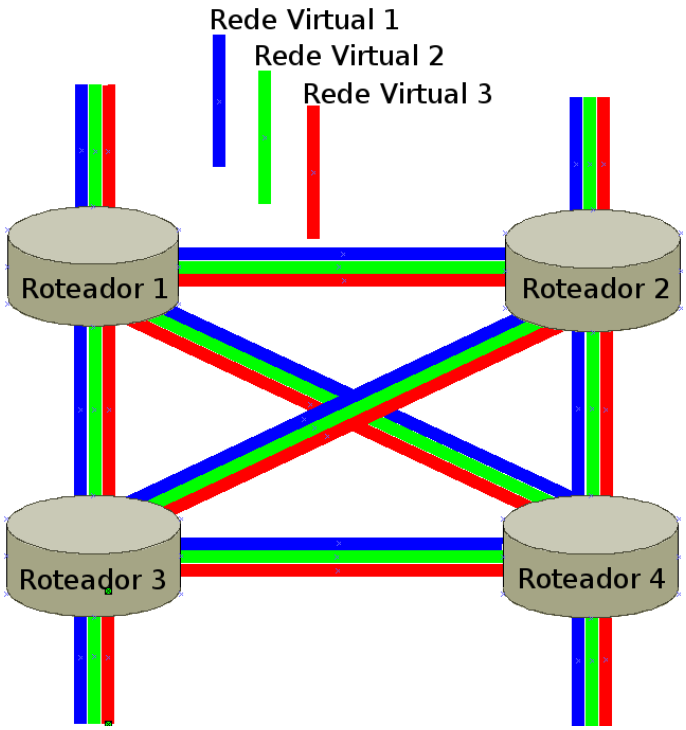


Figura 2 – Exemplo de redes sobrepostas defendidas pelos pluralistas.

Pesquisas anteriores já mostraram que algumas plataformas de virtualização não conseguem garantir que uma rede virtual funcione de forma independente sem sofrer interferência das outras redes criadas sobre o mesmo *hardware* [13] [23]. Além disso, os requisitos necessários para o bom desempenho de um roteador virtual precisam ser atendidos para que uma solução de virtualização de redes seja bem sucedida. Por esses motivos, este trabalho apresenta técnicas para uma arquitetura de virtualização que apresente tanto um bom desempenho quanto um bom isolamento entre as redes virtuais.

Definição das ferramentas e plataformas de experimentação

Antes de considerar a implantação de uma nova arquitetura de virtualização, ferramentas adequadas devem ser utilizadas na experimentação das novas soluções propostas. A simulação e a emulação, por exemplo, são úteis quando se deseja entender um novo projeto, contudo, não substituem a experimentação com tráfego real. Por este motivo, a experimentação de novas arquiteturas através do uso de *testbeds*[1] de produção muitas vezes é utilizada. Um dos exemplos está na arquitetura virtual Internet2, que em 2009 já era um *testbed* com 135 instituições de ensino, visando suportar atividades de pesquisa e o desenvolvimento de novas tecnologias que proporcionem o aumento do desempenho e da segurança na Internet do futuro [27]. Por estar intimamente ligada à arquitetura da Internet atual, *testbeds* de produção, tais como a Internet2, propiciam a obtenção de informações valiosas sobre o comportamento operacional de uma arquitetura. No entanto, por atenderem usuários e tráfego real, esses *testbeds* "físicos" devem ser implementados de forma precisa e conservadora para não apresentarem um padrão de qualidade e navegabilidade pior que o da Internet atual [31]. Por outro lado, *testbeds* de pesquisa, tais como o GENI (*Global Environment for Network Innovation*) ou o Federica, não trabalham com tráfego de usuários reais [34]. Por esse motivo, testes mais aventureiros, tais como colocar apenas a versão 6 do IP em produção numa rede de pesquisa, podem ser realizados nos mesmos. Porém, a falta de tráfego real não garante a viabilidade operacional real das soluções testadas [31]. Portanto, tanto os *testbeds* de produção como os de pesquisa podem tornar o resultado das avaliações pouco convincente. Além disso, a implantação de grandes *testbeds* para permitir a operação de redes em grande escala é substancialmente cara. Por esse motivo uma corrente de pesquisadores defende que tanto os de produção (Internet2) quanto os *testbeds* de pesquisa (GENI) não são a solução mais adequada para o desenvolvimento da Internet do futuro [31].

[1] *Testbed* é o mesmo que um banco de ensaio em tradução livre.

Para identificar a melhor plataforma de virtualização para a criação de redes virtuais com bom desempenho e bom isolamento entre si, as avaliações desse trabalho não fizeram uso de emulação ou simulação. Em lugar disso, as medições e análises foram realizadas em servidores x86 com características de hardware largamente encontradas nas máquinas dos *Datacenters* de provedores de conteúdo e ISPs atuais. O Xen, o KVM e o OpenFlow foram as plataformas de virtualização instaladas e testadas para que a sobrecarga de cada hipervisor sobre o hardware das máquinas pudesse ser medida. Assim, redes virtuais foram criadas com o Xen e com o KVM, em servidores reais, para que seus desempenhos fossem avaliados e comparados em função dos requisitos identificados como essenciais para uma plataforma de roteamento virtual. Verificamos também as vantagens que são oferecidas quando o OpenFlow, que possibilita a implementação de diferentes lógicas de encaminhamento a partir do fatiamento de redes nos dispositivos que oferecem suporte ao protocolo, é combinado com o Xen e com o KVM.

Mais especificamente, avaliamos o impacto que a quantidade de memória e CPUs atribuídas a uma máquina virtual exerce sobre o desempenho do roteamento e do encaminhamento de pacotes; qual plataforma de virtualização atendeu melhor as demandas de um roteador virtual e se alguma delas possui limitações que as impeçam de atender aos requisitos necessários para um encaminhamento e roteamento de pacotes de bom desempenho. Como todos os testes foram realizados em máquinas x86, também foi necessário entender como essa arquitetura funciona e de que forma os diferentes hipervisores operam nela. Dessa forma, é possível avaliar apenas as plataformas de virtualização capazes de atender aos requisitos de um roteador. Portanto, diferentes plataformas de virtualização foram estudadas e testadas em máquinas x86 porque elas são largamente encontradas nos ISPs e *Datacenters* da Internet atual [15]; e através de medições, verificamos quais ferramentas possuem as técnicas mais eficazes para garantir tanto o isolamento de ambientes virtuais, quanto um bom desempenho no encaminhamento e roteamento de pacotes.

Organização do trabalho

A dissertação está organizada da seguinte forma. No Capítulo 1, os conceitos e as diferentes técnicas de virtualização são apresentados. A seguir as plataformas de virtualização Xen e KVM são discutidas em detalhes no Capítulo 2, para serem comparadas em função dos requisitos essenciais de uma plataforma virtual de roteamento no Capítulo 3. Os resultados de diferentes testes realizados, com o objetivo de identificar quem possui melhor desempenho na plataforma x86 quando ela é utilizada para criar roteadores virtuais, estão inseridos nesse

capítulo. Finalmente, no Capítulo 4, apresentamos a plataforma de virtualização OpenFlow. Para melhorar o desempenho do roteamento de pacotes no Xen, apresentamos propostas de integração dele com o OpenFlow. Nesse capítulo, também discutimos as vantagens da integração do OpenFlow com o KVM. Depois disso, a dissertação é concluída e sugestões de trabalhos futuros, relacionados a virtualização de redes, são apresentadas.

1 AMBIENTES VIRTUAIS DE ROTEAMENTO EM MÁQUINAS X86

Roteadores são equipamentos que demandam muito processamento (CPU), grande consumo de memória e muitas operações de Entrada e Saída (E/S) de rede. Para processar um pacote, o roteador retira-o do buffer de entrada, inspeciona o endereço de destino contido no cabeçalho do pacote e verifica se existe rota para a rede identificada em sua tabela de roteamento (repasse). Se uma rota é encontrada, o pacote é inserido no *buffer* da interface indicada na tabela de repasse, para ser encaminhado para o próximo salto conhecido pelo roteador, em caso contrário o pacote é descartado. Portanto, a cada pacote roteado uma operação de E/S é realizada. Além do roteamento de pacotes, roteadores também podem tomar decisões baseadas em listas de acesso (*Access Lists* – ACLs) aplicadas em suas interfaces físicas ou virtuais (*Virtual Lans* - Vlans). Nesse caso, antes de rotear um pacote de uma para outra rede, um roteador verifica primeiro se a política de acesso contida na ACL permite ou nega o encaminhamento. Se existir uma política de permissão na lista de acesso, o roteador segue os passos de roteamento anteriormente descritos [20]. Caso contrário, o pacote é descartado para que a política de negação seja respeitada. As tomadas de decisão de roteamento, as consultas às listas de acesso e as operações de E/S, para serem realizadas a cada pacote encaminhado em alta velocidade, demandam grande consumo de memória e muito processamento em um roteador. Por esse motivo, a maioria dos roteadores modernos toma as decisões de roteamento e encaminhamento de pacotes manipulando tabelas de fluxo em nível de hardware, conhecidas como TCAMs (*Ternary Content Addressable Memory*) [16]. A velocidade da tomada de decisão de encaminhamento e roteamento de pacotes é acelerada em roteadores que utilizam TCAMs porque o processamento das informações contidas nessas tabelas é realizado em nível de hardware, por processadores dedicados para tratar as políticas que definem quais operações de E/S podem ser realizadas. Por este motivo, as consultas às listas de acesso aplicadas nas interfaces do equipamento passam a acontecer na ordem da taxa existente nos circuitos de dados (taxa/velocidade de linha), diminuindo as chances do roteador se tornar o gargalo da comunicação.

1.1 Características de roteadores virtuais executados em uma máquina x86

Roteadores virtuais criados em máquinas x86 não executam o encaminhamento e o roteamento de pacotes entre diferentes redes virtuais fazendo uso de TCAMs, porque a plataforma x86 não foi originalmente desenvolvida para suportar decisões de roteamento complexas (ACLs ou QoS) e operações de E/S de rede em taxa de linha – processar 1 bilhão

de pacotes por segundo (1 Gpps), por exemplo. Em lugar disso, na plataforma x86, as tabelas de fluxo são carregadas e manipuladas na memória RAM das máquinas.

É papel do hipervisor de uma plataforma de virtualização garantir a execução simultânea de múltiplas instâncias de dispositivos virtuais de conectividade [5]. Porém, para garantir o isolamento dos recursos físicos e da pilha de redes dos sistemas que ele supervisiona, o hipervisor de uma plataforma de virtualização insere um *overhead* (sobrecarga) de execução próprio. E essa sobrecarga varia de acordo com as diferentes técnicas e plataformas de virtualização disponíveis. Ou seja, cada hipervisor apresenta um consumo de memória, CPU, e desempenho de E/S diferentes, mesmo quando está submetido a um mesmo *benchmark*, em cenários de execução iguais.

Quanto maior a eficiência de uma plataforma de virtualização, mais rápida será a troca de dados entre os dispositivos de uma rede e menor será a sobrecarga de processamento e memória imposta pelo hipervisor sobre a máquina física. Esse é um requisito importante em redes virtuais porque a convergência dos protocolos de roteamento depende da atualização de informações entre os roteadores e é sensível ao tempo [16]. Portanto, quanto maior a eficiência de um hipervisor, mais rápida será a convergência dos protocolos de roteamento em roteadores virtuais que o utilizam. Por isso, é importante identificar qual é a mais eficiente entre as diferentes técnicas de virtualização e quem garante o fatiamento justo das operações de E/S, entre roteadores virtuais que compartilham um mesmo nó físico.

Roteadores possuem um Plano de Dados (PD) e um Plano de Controle (PC). No plano de controle, os protocolos de rede são implementados e é através dele que as tabelas de fluxo (encaminhamento e roteamento) do equipamento são gerenciadas. No plano de dados de um roteador, ou comutador, acontecem o encaminhamento e o roteamento de pacotes em função das informações que o PC inseriu em sua tabela de fluxo (Fig. 3). Plataformas de virtualização de hardware, tais como o Xen e o KVM, permitem que a filosofia pluralista – diversas arquiteturas de rede sobre o mesmo substrato físico compartilhado – possa ser executada. No entanto, é possível criar um roteador virtual com um plano de controle e um plano de dados, exclusivos, virtualizados (Fig. 4a) ou virtualizar apenas o plano de controle e manter um PD comum para os roteadores virtuais (Fig. 4b).

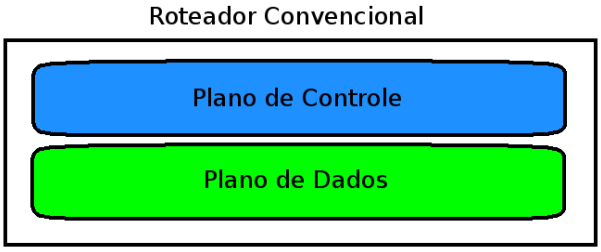


Figura 3 – Arquitetura convencional de um roteador com PD e PC “acoplados”.

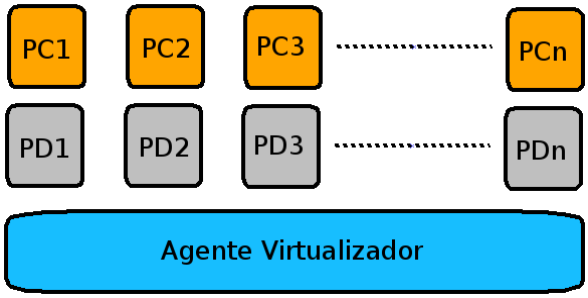


Figura 4a – Roteadores virtuais com PCs e PDs exclusivos.

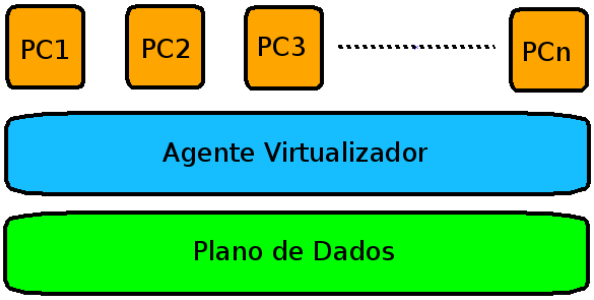


Figura 4b – Roteadores virtuais com PCs exclusivos e PD compartilhado.

Fernandes [11] mostrou que um roteador virtual criado no Xen consegue interferir no funcionamento de vizinhos ativos no mesmo nó físico, quebrando o isolamento entre as redes virtuais. O problema ocorre porque o Xen utiliza um domínio de drivers (dom0), que também é uma máquina virtual (*Virtual Machine* – VM), para compartilhar o acesso de todas as outras VMs aos dispositivos físicos da máquina. Portanto, além de executar o software de gerenciamento, apenas o dom0 acessa o hardware de forma direta porque só ele possui os drivers dos elementos físicos. No entanto, seu hipervisor existe apenas para controlar o acesso do dom0 ao hardware e ninguém gerencia o acesso que as outras máquinas virtuais do Xen, conhecidas como domUs (ou domínios convidados), fazem ao dom0 quando precisam realizar operações de E/S. Ou seja, se um domU conseguir comprometer o consumo de memória e processamento do dom0 ele pode prejudicar as operações de E/S de rede e disco de todos os outros domínios convidados. Por esse motivo, além de boa eficiência, também é importante verificar se uma plataforma de virtualização implantada pode realmente garantir o isolamento

entre os elementos virtuais. No caso do Xen, é importante verificar qual é a alternativa para o problema que foi descrito; e se existem outras plataformas de virtualização com eficiência semelhante, que não possuam o problema de isolamento nele verificado.

A extensibilidade é o requisito que permite a um hipervisor instanciar, apagar, monitorar e definir os parâmetros de alocação de recursos de um elemento virtual, além de conseguir migrá-lo entre diferentes nós físicos. Estas características são fundamentais para a virtualização de redes porque os dados do plano de controle, em arquiteturas virtuais de roteamento que utilizam PC distribuído, tais como o OpenFlow [26], precisam cruzar a estrutura física de forma transparente [35]. Ou seja, o hipervisor tem de viabilizar a extensibilidade do PC [5] nessas arquiteturas. Por último, também é importante verificar se existe flexibilidade para que a quantidade de interfaces de rede nos roteadores virtuais possa ser diferente da quantidade de interfaces de redes disponíveis na máquina física, que é uma característica importante para roteadores virtuais, e se o hardware precisa de características especiais para que a plataforma de virtualização seja executada. Os motivos apresentados mostram que a eficiência, o isolamento, a extensibilidade e a flexibilidade são requisitos essenciais de uma plataforma de virtualização utilizada para criar um ambiente virtual de roteamento [24].

1.2 Operação de um virtualizador de sistemas na plataforma x86∗

Antes de descrever como as diferentes técnicas de virtualização compartilham os dispositivos físicos entre diferentes elementos virtuais, é importante entender como o núcleo (*kernel*) de um Sistema Operacional (SO) administra o acesso ao hardware de uma máquina. O papel do núcleo em um SO é gerenciar os recursos físicos do sistema, controlando a comunicação dos softwares com o hardware. Ele gerencia a comunicação entre processos (aplicativos) e as chamadas de sistema, que são requisições de serviços enviadas para o núcleo do SO pelos aplicativos, todas as vezes que eles precisam acessar recursos de hardware, tais como a memória, a CPU e outros dispositivos [39]. Ou seja, o *kernel* atua como camada de abstração para os recursos físicos da máquina, escalonando o acesso dos processos à CPU, definindo que pedaços de memória cada aplicativo pode utilizar e controlando os pedidos que os aplicativos fazem para realizar operações de E/S em um dispositivo físico, que também pode acionar a CPU para enviar interrupções para o *kernel*, a fim de informar eventos ocorridos. Assim, o núcleo do SO implementa técnicas para evitar que um programa

∗ essa seção é parcialmente baseada em [11] e [39].

prejudique outro que está rodando no mesmo computador e para impedir que um processo acesse informações que ele não tem permissão para acessar.

A plataforma x86 implementa técnicas para que os aplicativos operem com níveis de privilégio menores que os do núcleo do SO, a fim de evitar que as instruções privilegiadas e os objetos protegidos do hardware sejam arbitrariamente acessados pelos processos em execução. Para isto, domínios hierárquicos de proteção são criados a partir dos anéis de privilégio da plataforma x86 (quatro anéis de privilégio, entre zero e três), para que um modo de acesso de usuário (modo de usuário) e um modo de acesso de *kernel* (modo de núcleo, privilegiado ou supervisor) existam. Os aplicativos, por padrão, sempre são executados no modo de usuário podendo chamar apenas com as instruções não privilegiadas de um processador. Além disso, os objetos protegidos ficam em espaços de endereçamento que não podem ser acessados pelos aplicativos. Dessa forma, o núcleo do sistema que está carregado na memória principal, por exemplo, fica protegido pelos modos de acesso, que impedem que um aplicativo mal intencionado sobrescreva arbitrariamente as áreas de memória que estão sendo utilizadas pelo SO, evitando a violação do mesmo. Portanto, quando um aplicativo precisa acessar um objeto protegido ele envia uma chamada de sistema para o núcleo, que acessa o objeto. Ou seja, os modos de acesso permitem que apenas os programas sendo executados no modo de *kernel* consigam acessar as áreas de memória do sistema operacional, objetos protegidos ou uma instrução de acesso a um dado no disco, quando necessário.

Um virtualizador de sistemas (hipervisor) pode operar de diferentes formas para permitir o funcionamento de múltiplas instâncias de dispositivos virtuais. Ele pode atuar no modo de núcleo (anel zero da plataforma x86) para virtualizar completamente ou para-virtualizar um hardware. Na virtualização completa, o hipervisor simula (ou emula) o hardware da máquina para os SOs das VMs (ou SOs visitantes). Ou seja, ele age como uma camada de controle entre os dispositivos físicos e o SO virtualizado para impedir que os SOs das VMs solicitem acesso direto aos dispositivos físicos. Contudo, os SOs visitantes, que são executados em nível de usuário, acreditam que estão se comunicando com os dispositivos físicos de forma direta quando enviam suas chamadas de sistema para o hipervisor, que simula totalmente o hardware e não precisam sofrer nenhum tipo de alteração quando são virtualizados. Ou seja, as instruções enviadas pelos visitantes para o hardware são na verdade interceptadas e tratadas pelo hipervisor antes de serem efetivamente enviadas para os dispositivos físicos [11]. O VMWare, o KVM e o Xen são exemplos de plataformas de

virtualização capazes de realizar a virtualização completa com a técnica de simulação de hardware.

Na para-virtualização o hipervisor também atua no anel zero da plataforma x86. Neste caso, diferente do que acontece na virtualização completa, o hipervisor não simula ou emula os dispositivos físicos. Os SOs das VMs são modificados para serem executados no anel um da plataforma e conhecem todos os endereços de hardware, inclusive os endereços de outros domínios virtuais. Contudo, os drivers de dispositivos do SO virtualizado interagem com o hipervisor que permite o acesso direto, porém controlado, dos SOs convidados aos dispositivos físicos. O hipervisor precisa controlar o acesso aos recursos físicos compartilhados, para impedir que um domínio acesse recursos que foram alocados por outra VM, e vice-versa. Ele atua para garantir que cada VM consiga alocar apenas as quantidades de memória, CPU e disco que foram definidas para si quando foram criadas.

O hipervisor também pode operar em nível de sistema operacional, no modo de usuário da plataforma x86, como um processo normal de um SO nativo. Nesse caso cada hipervisor é capaz de suportar apenas um SO virtual. Portanto, quando é necessário criar uma nova VM um novo processo hipervisor deve ser iniciado no SO hospedeiro (ou nativo). O VirtualBox e o VMWare oferecem esse tipo de solução de virtualização através de softwares que podem ser executados no Windows, MAC OS X e Linux.

Portanto, um hipervisor pode ser executado diretamente sobre o hardware no modo de núcleo, ou ser um processo normal de um SO nativo no modo de usuário [43]. Em todos os casos, até agora discutidos, as máquinas virtuais executam SOs completos, ou seja, são VMs de sistema. Porém, uma VM também pode ser criada em nível de aplicação (VM de aplicação). Neste caso, o virtualizador de sistemas atua abstraindo as camadas de execução para permitir que um software funcione em qualquer SO. Esse é o nível de virtualização utilizado pela *Java Virtual Machine* (JVM) [19].

Existem também as técnicas de virtualização baseadas em contêineres, tais como o OpenVZ, o Linux-VServer e o Linux *Container* (LXC). A técnica de contêiner permite o carregamento de sistemas operacionais segregados através da criação de múltiplos espaços de usuário usando chroots. Um chroot é uma operação que altera o diretório raiz de um processo e de seus filhos para impedi-los de acessar arquivos que não pertencem à árvore de diretórios criada pelo chroot [39]. Através dessa funcionalidade do Linux cópias do sistema operacional nativo podem ser criadas. Estas cópias são consideradas VMs porque são executadas dentro das “jaulas” criadas pelo chroot, que isola o diretório raiz de cada uma das demais [39].

1.3 Técnicas de virtualização na plataforma x86

Um hipervisor pode atuar em diferentes níveis de permissão, no modo de núcleo ou no modo de usuário. Todavia, também podemos comparar as ferramentas de virtualização que operam na plataforma x86 em função das técnicas que utilizam. Vistas dessa forma, elas podem ser plataformas que realizam virtualização completa, plataformas que realizam para-virtualização ou plataformas que utilizam contêineres para criar VMs de sistema. É possível comparar qualitativamente o desempenho do KVM, que utiliza a técnica de virtualização completa, com o desempenho do Xen que pode realizar tanto a essa técnica quanto a para-virtualização. Ou podemos comparar o desempenho destas plataformas de virtualização com o desempenho das que utilizam contêineres. Isto será realizado neste trabalho utilizando sempre o desempenho do Linux nativo como referência.

Por ser uma técnica de virtualização que acontece no Linux nativo, sem o uso de hipervisores para emular os dispositivos físicos, os aplicativos que rodam dentro dos contêineres compartilham e conseguem se comunicar diretamente com o núcleo do SO hospedeiro. Por isso, a sobrecarga de CPU e memória do agente virtualizador das VMs dos contêineres é baixa tornando o desempenho das máquinas virtuais muito semelhante ao encontrado no SO nativo. Contudo, os contêineres apresentam baixa flexibilidade por causa da obrigatoriedade de compartilhamento do mesmo sistema operacional (mesmo *kernel*) [24].

Na virtualização completa os SOs virtuais são executados no modo de usuário da plataforma x86, com permissões semelhantes as de um aplicativo do SO nativo, e não podem executar ações privilegiadas. O hipervisor intercepta as chamadas de sistema (*system call*) realizadas pelas VMs antes de encaminhá-las para a CPU e memória real, e só depois de receber o resultado da chamada encaminha a resposta para cada VM. Por este motivo, a sobrecarga de CPU e memória do agente virtualizador desse tipo de solução é maior que a encontrada nos contêineres [24].

Na para-virtualização os SOs das VMs conseguem realizar ações privilegiadas e o controle realizado pelo hipervisor é uma tarefa mais simples de ser realizada do que a simulação ou emulação completa do hardware. Por isso o desempenho das VMs desse modo de operação é maior que o encontrado na virtualização completa, porque a sobrecarga de memória e CPU do hipervisor é menor [9].

O Gartner é um instituto de pesquisa, execução e consultoria de TI, com pesquisadores em mais de 75 países do mundo, que trabalha para auxiliar os líderes das organizações na escolha da plataforma de virtualização que irão adotar. De acordo com o Gartner a virtualização em plataforma x86 já é responsável por mais de 50% da carga útil dos servidores em operação na Internet, e as máquinas virtuais são utilizadas na maioria dos casos para virtualizar aplicações, servidores web, bancos de dados, servidores de compartilhamento de arquivos, etc. Além disso, anualmente o Gartner divulga um "quadrante mágico" que indica quais plataformas de virtualização apresentam melhor capacidade de execução e abrangência de visão estratégica nas tecnologias que implementam [15].

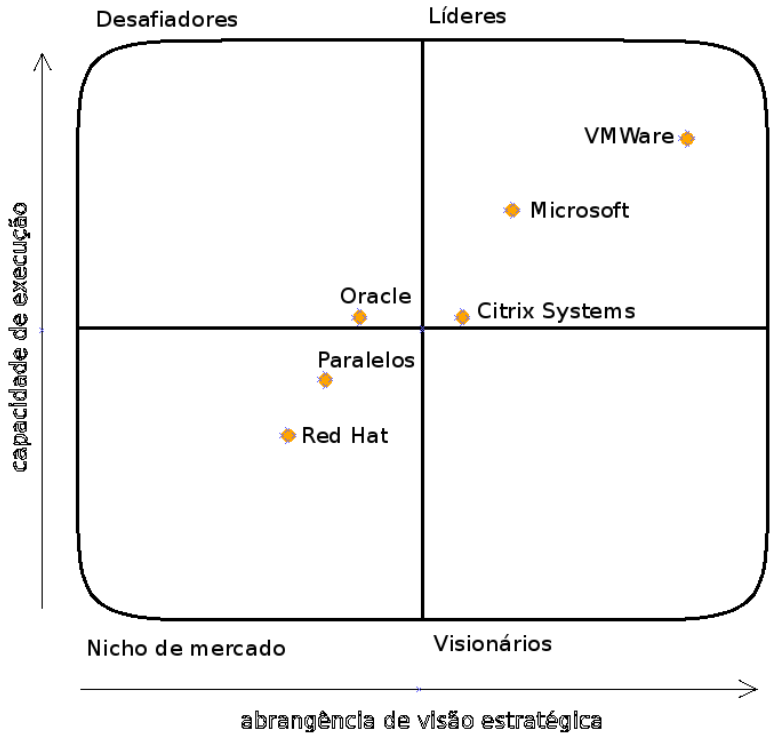


Figura 5 – Quadrante Mágico para infraestrutura de virtualização de servidores x86 – junho de 2012, fonte: Gartner.

A Tabela 1 lista as plataformas de virtualização que apareceram no quadrante mágico que o Gartner montou em 2012. Nele (Fig. 5) o ESX da VMWare, o Hyper-V da Microsoft e o Xen Cloud Platform (XCP) da Citrix foram consideradas as plataformas com maior abrangência de visão estratégica e melhor capacidade de execução. O KVM da RedHat, que pode ser utilizado gratuitamente, não apareceu no quadrante de liderança do Gartner, mas apesar de seu pouco tempo de vida [33] já aparece entre as plataformas de virtualização mais utilizadas, uma vez que a pesquisa considerou também o tamanho do mercado (*market share*) de cada solução. O XCP da Citrix, que utiliza o Xen como plataforma de virtualização, apareceu no quadrante de liderança, mesmo utilizando uma ferramenta de código aberto.

Baseados nesses dados, decidimos realizar todas as medições e análises desse trabalho com o KVM e o Xen, instalados em máquinas x86, porque eles já são plataformas de virtualização de código aberto bastante implementadas nas soluções de virtualização da Internet atual.

Tabela 1 – Infraestrutura de virtualização de cada fabricante.

Empresa	Plataforma de Virtualização
RedHat	KVM
Oracle	Oracle VM
Citrix Systems	Xen
Microsoft	Hiper-V
VMWare	VMWare
Parallels, Inc	Parallels

Portanto, diversas plataformas de virtualização podem operar em máquinas x86, utilizando as técnicas de virtualização completa, para-virtualização ou contêineres e podem atuar em diferentes níveis de permissão. Além disso, de acordo com o Gartner, a maioria dos dispositivos virtualizam servidores de conteúdo e estações de trabalho e é raro encontrar VMs criadas para agirem como roteadores. Contudo, a virtualização só é capaz de possibilitar o crescimento e o desenvolvimento da Internet do futuro se as técnicas e plataformas de virtualização também forem utilizadas na construção de ambientes virtuais de roteamento em ambientes de produção. As características do Xen e do KVM são apresentadas no Capítulo 2 e seus desempenhos são avaliados em seguida. A partir destas avaliações, propostas são apresentadas para que as redes virtuais criadas pelos pesquisadores e projetistas, para permitir a programação do núcleo da rede, possam ser eficientes e adequadas aos serviços ou aplicações que irão rodar sobre elas [42].

2 ARQUITETURA E CARACTERÍSTICAS DO KVM E DO XEN

As plataformas de virtualização escolhidas para serem estudadas nesse trabalho, com o objetivo de verificar quem possui melhor eficiência, isolamento, extensibilidade e flexibilidade foram o Xen e o KVM. Diversos autores já mediram o desempenho do Xen [1] [9] [12] [24] e o compararam com outras plataformas com o objetivo de encontrar a melhor ferramenta para construir ambientes virtuais. Entretanto, ainda não existem muitos textos acadêmicos que confrontem o Xen com o KVM.

2.1 Como o QEMU emula os dispositivos físicos para as VMs do KVM

O QEMU é um virtualizador e um emulador de hardware. Como emulador de máquina ele permite que SOs feitos para rodar em plataforma x86 possam ser executados em um PowerPC, e vice-versa, porque as diferentes CPUs são completamente emuladas para o SO. Como virtualizador ele precisa ser executado junto com o módulo de *kernel* Linux KVM, que é uma plataforma de virtualização. Neste caso, ele realiza virtualização completa abstraindo o conjunto de instruções nativas e criando um mapa entre as instruções do SO visitante e as do SO hospedeiro. As instruções que o visitante envia para o hardware são interceptadas e tratadas pelo hipervisor QEMU em tempo de execução. No entanto, o QEMU pode possibilitar que o desempenho dos SOs visitantes (convidados) seja muito semelhante ao do Linux nativo. Isto porque ele é capaz de aumentar a velocidade de execução na plataforma x86 quando permite que o SO virtualizado, que é executado em modo de usuário, acesse de forma direta a CPU do computador hospedeiro. Neste trabalho, vamos mostrar que isto diminui consideravelmente a perda de desempenho, característica da virtualização completa quando as tarefas do sistema operacional convidado são executadas [3] [21] [32].

O KVM é uma solução de virtualização que foi embutida nas distribuições Linux, mais recentes, que são baseadas no SO da RedHat. O CentOS 6, o Fedora 18 e o próprio RedHat 6, por exemplo, são sistemas operacionais que utilizam o QEMU para virtualizar as operações de E/S dos dispositivos de rede, disco, etc. da máquina física para SOs virtuais. No KVM, o QEMU é um processo em execução dentro do SO virtual que emula o modo de usuário do SO hospedeiro. O QEMU faz isso para gerenciar todas as operações de E/S demandadas pelas VMs, direcionando-as corretamente para o hardware. Ou seja, o *kernel* do Linux age como um hipervisor quando o módulo KVM é instalado no Linux nativo [21]. A partir daí, para virtualizar a memória, por exemplo, o hipervisor KVM exporta uma árvore de dispositivos “/dev/kvm” e atribui espaços de endereços exclusivos para cada SO virtual quando é iniciado.

Em um SO Linux, por padrão, todos os processos de espaço do usuário (aplicativos) compartilham os dispositivos físicos que são mapeados pelo *kernel* no /dev e todos os arquivos contidos nesse diretório são ponteiros para dispositivos de hardware. Por esse motivo, quando um aplicativo precisa acessar o hardware ele "consulta" o /dev para realizar corretamente a chamada de sistema para o dispositivo físico. E para realizar essa chamada, o código da aplicação precisa executar apenas ações de escrita e leitura nos arquivos indicados dentro /dev para que o *kernel* do Linux realize o restante do trabalho. O arquivo /dev/mouse contém as informações enviadas/recebidas pelo mouse e o /dev/sda1 é um arquivo que representa o primeiro disco SCSI de um SO, por exemplo. No SO virtualizado, as aplicações não enxergam os endereços de hardware consultando de forma direta o /dev do Linux que está em execução no hospedeiro. O que as aplicações do SO virtualizado acessam é o /dev/kvm que foi criado para o seu sistema operacional pelo QEMU quando a VM foi criada. Como cada VM possui um espaço de endereços /dev/kvm exclusivo, emulado pelo QEMU, os mesmos ficam segregados do /dev, que é o espaço de endereços do *kernel* do SO hospedeiro, para que o isolamento no acesso ao hardware, nas operações de E/S, seja mantido.

2.2 Modos de acesso da plataforma de virtualização KVM

No KVM, um novo modelo de processamento é criado porque, além do modo de acesso de usuário e do modo de acesso de núcleo (modo de *kernel*) criados a partir dos anéis de privilégio da plataforma x86, um "*modo convidado de núcleo*" também é criado para permitir a execução do SO virtual. Cada VM é vista pelo SO hospedeiro como um único processo em execução e o modo convidado de núcleo permite ao SO da VM acessar, além de seu próprio modo de usuário, quando necessário, o modo de *kernel* do hospedeiro. Para fazer o núcleo do Linux nativo agir como um hipervisor, o código do KVM é integrado ao código do *kernel*. Assim, quando o *kernel* do Linux é carregado o KVM também é executado. Por este motivo, só é possível executá-lo em uma máquina com processadores capazes de realizar a virtualização. Além disso, o KVM depende sempre de um processo QEMU operando como emulador do espaço de usuário para que a virtualização dos dispositivos de E/S seja realizada (Fig. 6).

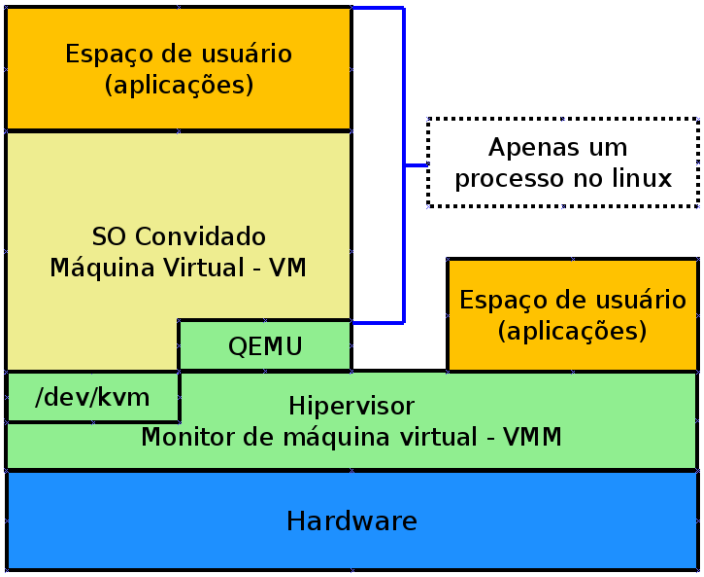


Figura 6 – Arquitetura da plataforma de virtualização KVM.

Vimos anteriormente que em uma solução de virtualização completa padrão, tal como o VMWare, quando um SO virtualizado deseja acessar um dispositivo de hardware (memória, CPU, etc.), cada uma de suas chamadas de sistema é interceptada pelo hipervisor que é quem realmente faz a chamada para o dispositivo real. Ao receber a resposta do hardware, ele encaminha a resposta para a VM, que acredita ter se comunicado de forma direta com os dispositivos físicos, quando na verdade interagiu com um hardware virtual emulado pelo hipervisor. Portanto, na virtualização completa padrão, os SOs das máquinas virtuais não conhecem os endereços dos dispositivos físicos [24].

O KVM é uma solução de virtualização completa porque o SO convidado envia suas chamadas de sistema para o hardware através do QEMU, que virtualiza todas as operações de E/S para os dispositivos físicos que o hipervisor KVM exportou na árvore “/dev/kvm” com espaços de endereços exclusivos para o SO de cada VM. Porém, por ser executado dentro do núcleo do Linux, o desempenho do hipervisor KVM pode ser melhor que o encontrado em uma solução de virtualização completa padrão, caso o SO da máquina virtual seja informado que está sendo executado em um ambiente virtual, para que suas operações de E/S de rede e disco possam ser realizadas no modo de privilegiado. No Linux, isso pode ser feito com a funcionalidade VIRTIO da biblioteca de programação libvirt [22]. Com o VIRTIO habilitado, os drivers do dispositivo de hardware que o SO da VM deseja acessar são informados que estão sendo executados em um ambiente virtual. Assim, eles interagem com o QEMU que lhes permite acessar o modo de *kernel* do hospedeiro e executar ações privilegiadas de E/S de rede, disco, etc.

2.3 Características da plataforma de virtualização Xen

Quando um SO é instalado em uma máquina x86, seus códigos rodam sempre no nível zero de permissão – mais privilegiado, e as aplicações no nível 3 que é o menos privilegiado. No Xen, quando o SO de um domU é carregado, ele roda no nível 1 de permissão da plataforma x86 e as aplicações das VMs são executadas com nível de prioridade 3. Como o hipervisor do Xen precisa controlar o acesso dos domUs ao hardware da máquina, ele precisa ser executado com um nível de prioridade maior que a dos SOs das VMs. Por isso, seu hipervisor é implementado para atuar com nível zero de prioridade. O Xen fatia os elementos físicos para criar VMs e é capaz de suportar tanto a virtualização completa quanto a para-virtualização. Por isso, como acontece no KVM, também permite a execução de diferentes domínios virtuais, com diferentes SOs, sobre a mesma máquina física. Ou seja, em uma VM, o SO em execução pode ser o Windows 7, enquanto outra executa o CentOS 6 e outras máquinas virtuais executam o Mac OS, por exemplo. Entretanto, a arquitetura do Xen (Fig. 7) é diferente da encontrada no KVM (Fig. 6). Seu hipervisor fica acima da camada física, sobre ele há um domínio de drivers privilegiado, conhecido como dom0, e diversos domínios convidados (não privilegiados), conhecidos como domUs, podem ser criados [12].

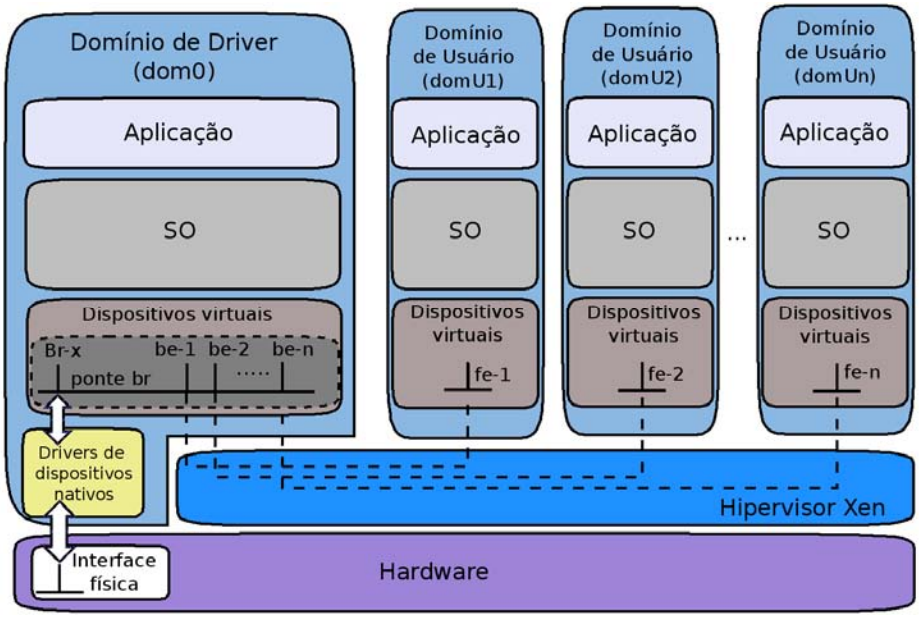


Figura 7 – Arquitetura do Xen.

O dom0 do Xen, além de ser o responsável pelo armazenamento dos drivers, possui privilégios especiais sobre os dispositivos físicos e hospeda o software de gerenciamento através do qual as VMs são criadas e destruídas. Através do software de gerenciamento, a alocação da memória física para cada domínio convidado é realizada e as permissões de

acesso de cada VM aos discos e às interfaces de rede virtuais são criadas [9]. Além disso, ele permite que regras de roteamento, filtros, interfaces de rede virtuais e blocos de disco virtuais possam ser adicionados ou removidos de uma VM [1], e realiza decisões políticas complexas, tais como controle de admissão [9]. Apenas o dom0 enxerga as interfaces de rede da máquina física. Enquanto ele possui acesso total ao hardware, os outros domínios utilizam drivers virtuais para se comunicar com os recursos físicos através do hipervisor [12].

Ao hipervisor do Xen cabe agendar o uso da CPU entre as VMs, filtrar pacotes de rede antes do encaminhamento, aplicar controle de acesso na leitura de blocos de dados etc. Ele realiza esse trabalho controlando o acesso dos domUs aos recursos físicos da máquina, substituindo as chamadas de sistema dos SOs convidados, que não são encaminhadas diretamente para o hardware. As chamadas de sistema dos SOs convidados são substituídas pelas do Xen, que são conhecidas como *hypercalls*. Apenas depois de validar as solicitações feitas pelos SOs convidados, o hipervisor envia as *hypercalls* para os dispositivos físicos. Ele faz isso para tentar garantir o isolamento do hardware e impedir que uma VM consiga afetar o desempenho de outra [9] [12]. Contudo, vimos anteriormente que na arquitetura do Xen ninguém gerencia o acesso dos domUs ao dom0, quando eles precisam acessá-lo para realizar operações de E/S. Por esse motivo, uma VM maliciosa ainda pode quebrar o isolamento quando leva o dom0 a consumir mais CPU do que deveria para manipular uma tabela de encaminhamento ou de roteamento de pacotes, por exemplo.

Apesar de ser implementado para atuar com nível de prioridade zero, o hipervisor do Xen não é responsável por saber como a CPU está sendo compartilhada, porque este papel continua sendo do escalonador do SO hospedeiro. Esse hipervisor foi projetado para realizar operações de controle básicas. Caso políticas de acesso (ACLs) sejam necessárias para controlar o que as máquinas de uma rede podem acessar em máquinas que foram inseridas em outra rede virtual, um software capaz de realizar filtro de pacotes, tal como o Iptables, precisará ser executado dentro do dom0 porque o hipervisor não é capaz de realizar esse tipo de tarefa.

2.3.1 Arquitetura do Xen com pontes

O Xen pode utilizar pontes (*bridges*) para interligar as interfaces de rede das VMs às interfaces físicas ou pode rotear os pacotes que saem das VMs. Por padrão, o Xen utiliza a tecnologia de ponte para interligar as interfaces de rede da máquina física às interfaces virtuais criadas no dom0, que são identificadas com a lógica vifX.Y (Fig. 8), onde X

identifica o domU para quem a interface foi criada, e Y é um identificador único de interface dentro do domU. Assim, interface de rede de um domU é na verdade uma interface virtual criada dentro do dom0 que é conhecida como interface de *Back-End* (BE). No domU são usadas interfaces de *Front-End* (FE) e as interfaces de BE e FE são interligadas através de um canal de E/S que passa pelo hipervisor (Fig. 8). Quando um pacote, destinado à interface virtual ethY de um domU (FE), entra pela interface física da máquina, o dom0 demultiplexa o tráfego para direcioná-lo para o domU correto. Quando o contrário acontece, o dom0 multiplexa o tráfego antes de direcioná-lo para fora da máquina através de sua interface física (Fig. 8) [9].

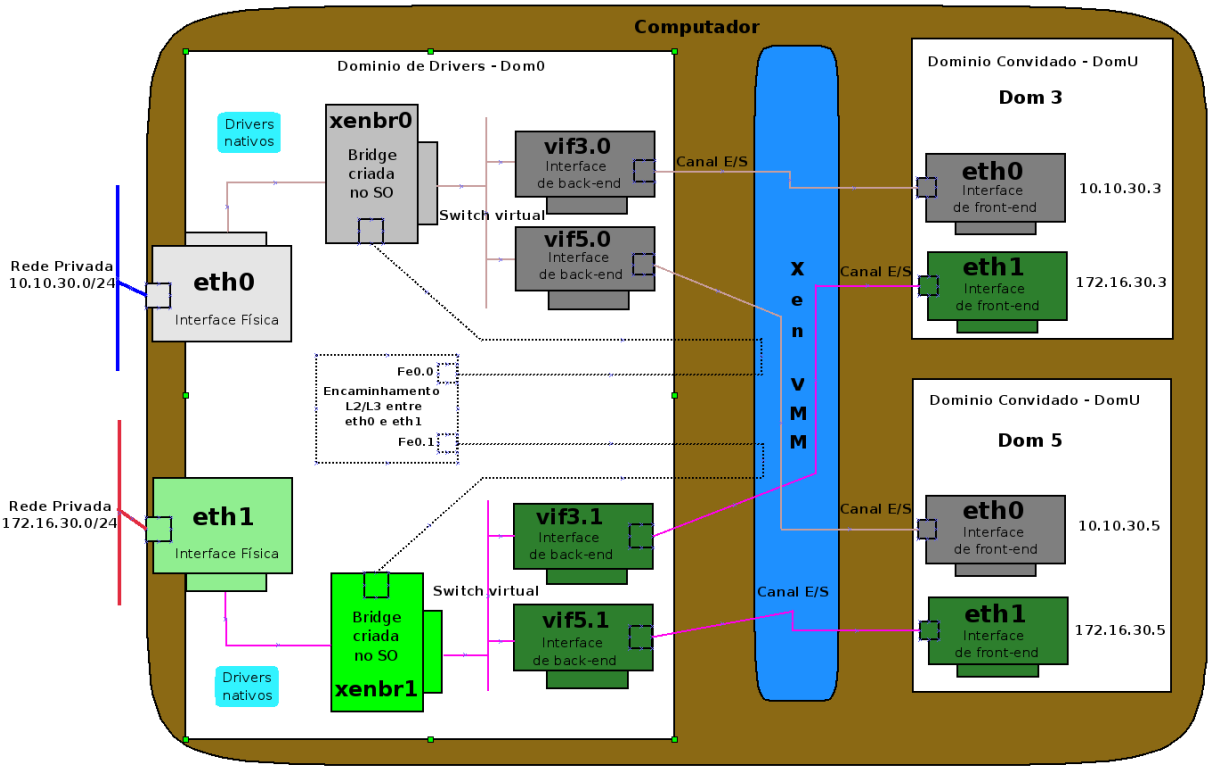


Figura 8 – Modelo de interligação das interfaces de *back-end* e *front-end* do Xen com pontes.

Observe que na tecnologia de ponte (Fig. 8), endereços IP podem ser configurados nas interfaces de FE (ethY) localizadas nos domUs. No entanto, como não existe endereço IP nas interfaces vif3.0 e vif5.0 do dom0, e a ponte xenbr0 criada no dom0 interliga as interfaces de BE dos domínios convidados, não há roteamento entre as interfaces eth0 de domU3 e domU5. O hipervisor do Xen portanto encaminha os pacotes entre as interfaces pelo canal de E/S em função de seus endereços de Camada 2 (*Medium Access Control Addresses* - MAC Addresses). Ou seja, nessa tecnologia todos os domínios convidados criados precisarão compartilhar o mesmo domínio de colisão das pontes xenbr0 e xenbr1, que na Figura 8 estão

inseridas nas redes 30 – 10.10.30.0/24 e 300 – 172.16.30.0/24, respectivamente. Portanto, quando um pacote entra na interface eth0 da máquina física com destino à máquina virtual domU3, a VM domU5 pode “ouvir” esse tráfego. Funcionando dessa forma, a tecnologia de ponte é inapropriada para a construção de redes virtuais, uma vez que o isolamento do tráfego de pacotes entre roteadores virtuais, que podem pertencer a diferentes empresas ou equipes de trabalho (domU3 foi alocado para os pesquisadores da organização A e domU5 ao grupo de pesquisa da organização B, por exemplo), não é garantido. Um filtro de pacotes, tal como o Iptables, pode ser inserido no dom0 para resolver esse problema. Porém, essa ação só é eficaz quando combinada com a criação de Vlans exclusivas para cada grupo de trabalho.

2.3.2 Arquitetura do Xen híbrido

A falta de isolamento de tráfego entre roteadores virtuais de diferentes grupos de trabalho pode ser garantida se a arquitetura híbrida do Xen for utilizada. Nessa arquitetura, as pontes continuam existindo, contudo o tráfego é segregado através de Vlans criadas nessas pontes. A Figura 9 mostra duas Vlans criadas na ponte xenbr0 e duas criadas na ponte xenbr1 para segregar o tráfego de pacotes do domínio convidado domU3 do tráfego de pacotes do domínio domU5. Nessa solução os endereços IP de diferentes redes devem ser inseridos em sub-interfaces criadas nas *bridges*: xenbr0:30, xenbr0:50, xenbr1:300 e xenbr1:500 (Fig. 9). Assim, diversas máquinas virtuais tanto podem compartilhar o mesmo domínio de colisão de uma sub-interface xenbrA:B, quanto máquinas virtuais com domínios de colisão e broadcast exclusivos, com uma sub-interface xenbrA:B para cada Vlan, podem ser criados.

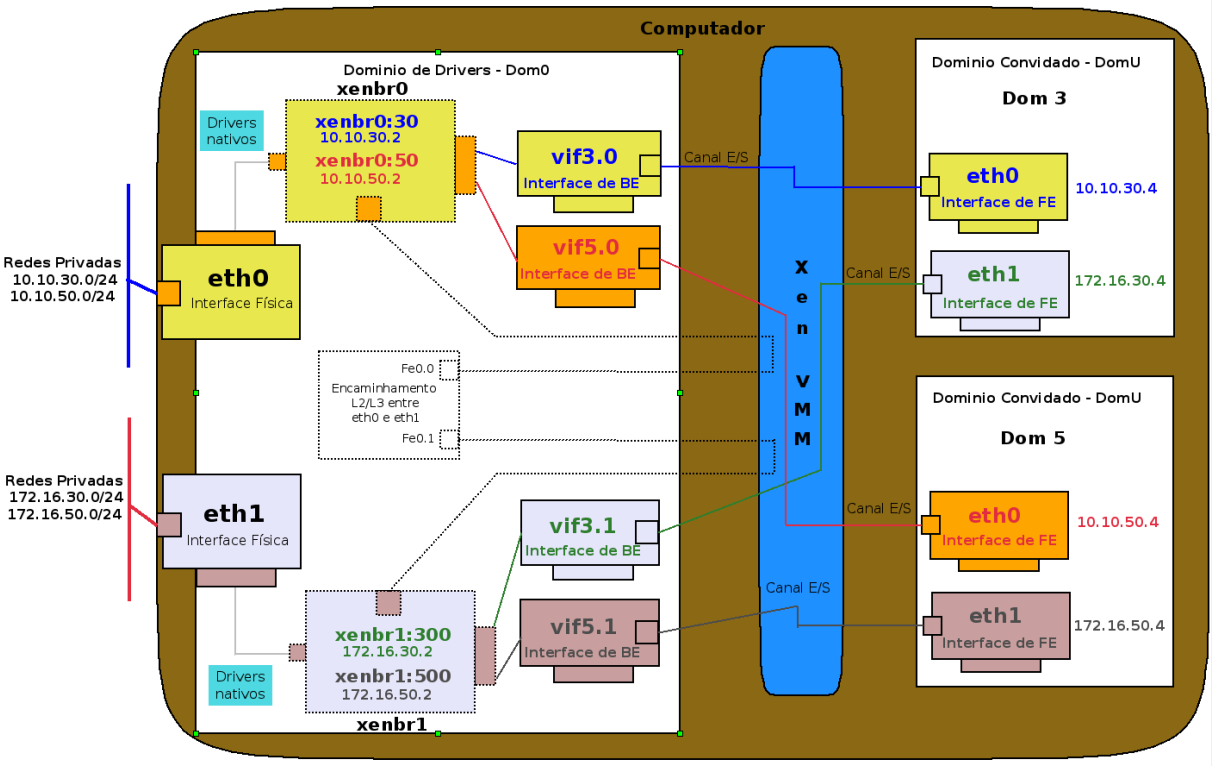


Figura 9 – Xen híbrido com o dom0 atuando como *gateway* e proxy para as VMs.

Suponha que o domU3, que utiliza as Vlans 30 e 300 deseja se comunicar com equipamentos que estão nas redes 50 e 500, criadas para o domU5. Nesse caso, apenas um roteador com interfaces nas redes citadas é capaz de permitir essa comunicação, encaminhando pacotes de uma para outra rede. Observe que, as pontes criadas no dom0 da arquitetura híbrida ainda conseguem encaminhar os pacotes entre diferentes redes através do canal de comunicação que existe entre Xenbr0 e Xenbr1, passando pelo hipervisor. No entanto, para tornar isso possível o dom0 precisa ser configurado como *gateway*, nas VMs domU3 e domU5. Porém, se o dom0 atuar como *gateway* das VMs, que foram inseridas em Vlans exclusivas, um filtro de pacotes precisará ser inserido nele (dom0) para que apenas o tráfego desejado seja roteado entre as duas VMs. O Iptables é uma aplicação capaz de realizar filtro de pacotes, e é comumente encontrado em soluções de virtualização para segregar o tráfego entre as VMs de diferentes redes virtuais que possuem como *gateway* uma interface criada dentro da própria máquina física. Porém, a execução de um filtro de pacotes no dom0 aumenta a concorrência pelos recursos físicos (memória e processamento) da máquina.

2.3.3 Arquitetura do Xen roteado

Na arquitetura roteada, o Xen também permite que diferentes redes sejam segregadas com o uso de Vlans, uma para cada grupo de pesquisa ou projeto. Para isto, as Vlans são criadas nas interfaces físicas eth0 e eth1 da máquina, ou nas interfaces virtuais vifX.Y de

acordo com o exemplo mostrado na Figura 10. Portanto, é possível configurar endereços IP de diferentes redes virtuais, em sub-interfaces ethA:B ou vifX.Y:B. Quando isso é feito, as *bridges* não precisam existir e o tráfego de pacotes entre as interfaces físicas e virtuais é roteado (Xen roteado).

Citamos anteriormente que quanto menor a sobrecarga de memória e CPU de uma solução de virtualização, melhor ela será para a construção de redes virtuais, porque esses requisitos são essenciais para o bom desempenho de um roteador. Portanto, em lugar de utilizar a topologia de híbrida, com políticas de acesso no Iptables do dom0 para garantir o isolamento do trafego de pacotes entre as VMs e *gateway* criado no dom0, é mais adequado construir roteadores virtuais utilizando-se a técnica de roteamento, com Vlans associadas às interfaces físicas ou às interfaces virtuais vifX.Y; porém com o *gateway* das Vlans inserido em um dispositivo de rede localizado do lado de fora da máquina onde as VMs são criadas (Fig. 10). Fazendo isso, a tomada de decisão de roteamento entre as diferentes Vlans criadas no *gateway* externo para segregar os roteadores virtuais de diferentes organizações ou projetos, pode ser realizada por um dispositivo que utiliza TCAMs, para acelerar essa atividade. Assim, a segurança dos dados de diferentes roteadores virtuais (VMs) criados dentro do Xen é garantida, sem que o custo de execução das políticas de acesso aplicadas no *gateway* externo interfira no desempenho das VMs criadas para agirem como roteadores virtuais. Essa solução também é válida quando a plataforma de virtualização é o KVM e é uma característica que torna possível a construção de soluções de roteamento virtual em que os roteadores criados dentro das VMs iniciadas em máquinas de plataforma x86 conseguem interagir com roteadores físicos, ainda que eles não possuam suporte a virtualização.

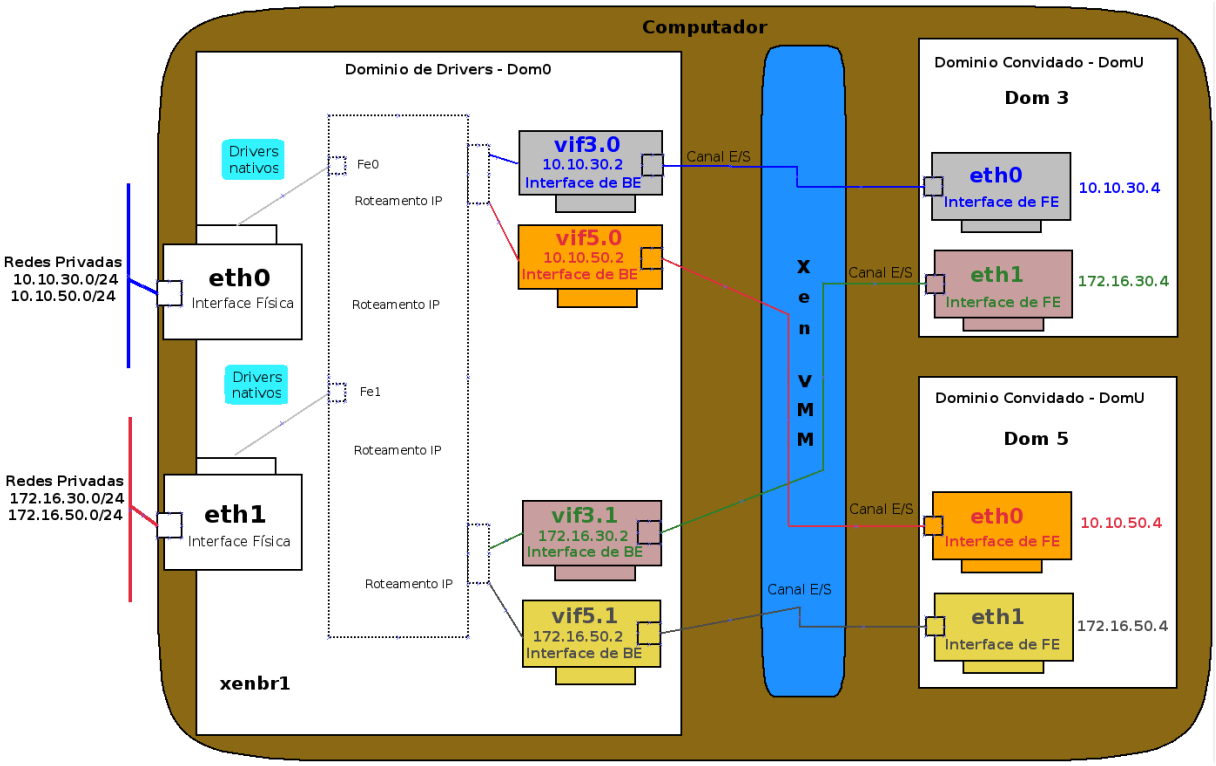


Figura 10 – Modelo de interligação com roteamento entre as interfaces de rede.

Quando a arquitetura utilizada no Xen é a roteada (Fig.10), o tráfego de pacotes entre as interfaces de rede existentes no dom0 é realizado pelas ferramentas de roteamento do *kernel*. Egi e outros [9] mostraram que o desempenho do Xen roteado é melhor que o Xen modo *bridge*. Fernandes [12] refez os testes e voltou a mostrar que é melhor utilizar o Xen roteado, entretanto também mostrou que quando as tabelas de repasse dos roteadores virtuais são copiadas para o dom0, para criar um plano de dados compartilhado, o desempenho é melhor que o encontrado quando os pacotes precisam passar por dentro da máquina virtual. Isso acontece porque o roteamento feito no dom0 ocorre da mesma maneira como é realizado no Linux nativo, sem que os pacotes precisem atravessar o hipervisor para entrar na VM; na Figura 10, quando precisam entrar pela interface eth0 de domU3 e sair pela interface eth1 dessa VM para alcançar o dispositivo que está ligado na interface física eth1 da máquina. Ou seja, se um pacote originado na rede 10.10.30.0/24 entra pela interface eth0 com destino à rede 172.16.30.0/24 o tráfego não precisa passar pelo hipervisor para consultar a tabela de roteamento de domU3 se o paradigma da separação de planos for utilizado. A consulta é realizada dentro do próprio dom0 porque ele possui uma cópia da tabela de repasse de todos os domUs.

A seguir comparamos o desempenho do Xen e do KVM em função dos requisitos identificados como essenciais para uma plataforma de roteamento virtual. Todas as avaliações

são realizadas a partir de medições obtidas em testes que são feitos com servidores reais, onde redes virtuais são criadas**.**

3 XEN *VERSUS* KVM EM UMA PLATAFORMA VIRTUAL DE ROTEAMENTO

Discutimos a importância da flexibilidade, eficiência, isolamento e extensibilidade para o bom desempenho de um ambiente que demanda roteamento virtual [24] e sabemos que o Xen e o KVM são plataformas de virtualização que podem segmentar os recursos físicos de uma máquina para criar roteadores virtuais [12]. No entanto é importante avaliar quem apresenta o melhor desempenho entre esses quesitos. Nesse capítulo vamos comparar os resultados do Xen para-virtualizado com os do KVM, utilizando os resultados do Linux nativo como referência. Em todos os casos (no Linux, no Xen e no KVM) não vamos utilizar o modo *bridge* porque diversos pesquisadores [1] [5] [9] [12] [13] já mostraram que o mecanismo de roteamento do *kernel* é mais eficiente e menos custoso que o implementado na transmissão de pacotes usando pontes. Nesse capítulo, também não vamos comparar os resultados do Xen com o paradigma de separação de planos porque ele precisa ser combinado com o OpenFlow para que esse modo de operação seja implementado. Essa não é a configuração padrão do Xen. Ou seja, todas as vezes que uma VM Xen é criada, para ser utilizada como um roteador, um plano de dados e um plano de controle exclusivos são criados dentro da VM (Fig. 4a). Por esse motivo cada pacote roteado por um roteador virtual passa sempre por dentro da VM para que a tabela de encaminhamento/roteamento dessa máquina virtual seja consultada, para viabilizar o roteamento dos pacotes. A mesma coisa acontece nas VMs criadas pelo KVM porque essas plataformas de virtualização não segmentam planos de dado e de controle. Para criar rotadores virtuais concorrentes, elas segmentam os recursos físicos de um hardware. O OpenFlow, que permite criar roteadores virtuais com PCs exclusivos e PD compartilhado (Fig. 4b), será combinado com o Xen e com o KVM no próximo capítulo deste trabalho.

3.1 Flexibilidade das plataformas de virtualização Xen e KVM

No Xen e no KVM é possível criar uma quantidade variada de sub-interfaces nas interfaces físicas (várias ethA:B) ou nas pontes (várias brA:B) associadas a elas. Ou seja, a quantidade de interfaces de rede nos roteadores virtuais não é limitada pela quantidade de interfaces de redes disponíveis na máquina física nem no Xen, onde diversas interfaces de FE podem ser criadas, nem no KVM, que através do QEMU também permite que uma quantidade variada de dispositivos de rede seja declarada para uma VM dentro de seu /dev/kvm. Portanto as duas soluções atendem a esse requisito de flexibilidade, que é essencial para uma plataforma virtual de roteamento, uma vez que seus hipervisores não impedem que a

quantidade de interfaces de rede virtuais nas VMs seja diferente das disponíveis na máquina física [18].

Se o hardware não suportar nativamente a virtualização o KVM não funciona, pois o núcleo do Linux nativo age como um hipervisor quando o KVM está instalado. Todavia, o hipervisor do Xen consegue ser executado mesmo quando a CPU não é nativamente capaz de realizar virtualização. Isso acontece porque o Xen insere o suporte necessário para que as interrupções das VMs sejam adequadamente enviadas para o hardware. Esse serviço a mais oferecido torna o Xen mais flexível que o KVM, porque permite sua instalação em máquinas x86 que possuem CPUs sem suporte a virtualização.

3.2 Eficiência das plataformas de virtualização Xen e KVM

A eficiência de um roteador é função da capacidade que ele possui para realizar operações de E/S de rede, tanto de encaminhamento quanto de roteamento de pacotes. E isso varia em função das capacidades de processamento e memória do hardware, além da taxa de transmissão das interfaces de rede do roteador. Assim, por demandar poucas operações de E/S de disco, o bom desempenho de um roteador virtual depende pouco da capacidade de acesso a disco de uma solução de virtualização. Contudo, quanto menores as sobrecargas de memória e CPU e quanto maior for a capacidade de encaminhamento e roteamento de pacotes em uma plataforma de virtualização, mais eficientes serão os roteadores virtuais criados em suas VMs.

Os primeiros testes foram realizados em máquina x86 com grande quantidade de memória e CPU para verificarmos se a eficiência dos roteadores virtuais seria diretamente proporcional à quantidade de recursos alocados para cada VM. Uma das lâminas (máquinas) do servidor Dell PowerEdge C6100 – Dell C6100 foi utilizada. Ou seja, um dos quatro servidores existentes no Dell PowerEdge C6100, que é um hardware com quatro nós servidores de 2 soquetes, criado com tecnologia HPCC (Cluster de Computação de Alto Desempenho), foi usado nos primeiros testes. Assim, a máquina utilizada nos primeiros testes desse trabalho possuía 4 CPUs Intel Xeon L5600 de seis núcleos, 12 MB de memória cache por núcleo, 50 GB de memória RAM, 900 GB de disco rígido e duas interfaces de rede Gigabit Ethernet Intel 82576. Portanto, com esse hardware é possível criar uma VM com até 24 CPUs virtuais (4 Xeon L5600 multiplicado por seis núcleos = 24), 50 GB de memória e gerar tráfego de dados de até 1 Gbits/s. Utilizamos esse hardware para saber se a eficiência no consumo de memória e CPU de uma VM melhora quando ela possui mais recursos de hardware alocados para ela. Ou seja, uma VM com 30 GB de memória e 24 CPUs virtuais é

mais eficiente que uma VM com apenas 2 GB de RAM e 4 CPUs? É aconselhável criar VMs em máquinas x86 com muita memória e muitas CPUs para que o desempenho do roteador virtual seja aumentado?

No primeiro cenário de testes a sobrecarga dos hipervisores sobre a CPU foi medida com o *benchmark* Super Pi [38]. Essa ferramenta, que realiza diversas operações aritméticas, foi executada para que o número Pi com 2^{25} casas decimais fosse calculado. No segundo cenário, o *benchmark* STREAM 5.1 foi utilizado para medir o desempenho de memória. As duas ferramentas de *benchmark* foram executados em uma VM Xen, no Linux nativo 2.6.32-358.2.1 de 64 bits e em uma máquina virtual KVM.

3.2.1 *Benchmark* Super Pi para medir a sobrecarga de CPU

A primeira VM (VM01) foi instalada com 30 GB de memória e 16 CPUs porque, apesar de ter sido possível criar uma VM com 24 CPUs virtuais no KVM, a versão mais moderna do Xen no momento em que os testes foram realizados – versão 4.2.1, não conseguiu alocar mais de 16 CPUs virtuais. Comparamos os resultados do *benchmark* de CPU obtidos na VM01 com os encontrados na VM02, que foi criada com apenas 2 GB de memória e mesma quantidade de CPUs virtuais (16). A média de 15 rodadas de testes com um intervalo de confiança de 95% é apresentada na Figura 11, onde é possível observar que utilizando uma versão *monothread* do Super Pi, principalmente no Xen, o desempenho de CPU foi pior na VM com maior quantidade de memória virtual – 30 GB. Isso mostra que a sobrecarga do hipervisor do Xen sobre a CPU foi maior na VM01, com 30 GB de RAM e 16 CPUs virtuais. Por isso, o tempo médio necessário para calcular o Pi com 2^{25} casas decimais nesta VM foi maior (2050 s) que o tempo necessário para fazer o mesmo cálculo na VM02 (2004 s), que possuía apenas 2 GB de RAM e 16 CPUs virtuais. No KVM o mesmo comportamento é observado: a VM01 com 30 GB levou mais tempo (1055 s) para fazer o cálculo do que a VM02 (1048 s) que possuía menos memória disponível. No entanto, como o tempo necessário para a execução dos cálculos foi menor no KVM (1055 s e 1048 s *versus* 2050 s e 2004 s no Xen), podemos afirmar que a sobrecarga do hipervisor do KVM sobre a CPU foi muito menor que a do hipervisor do Xen.

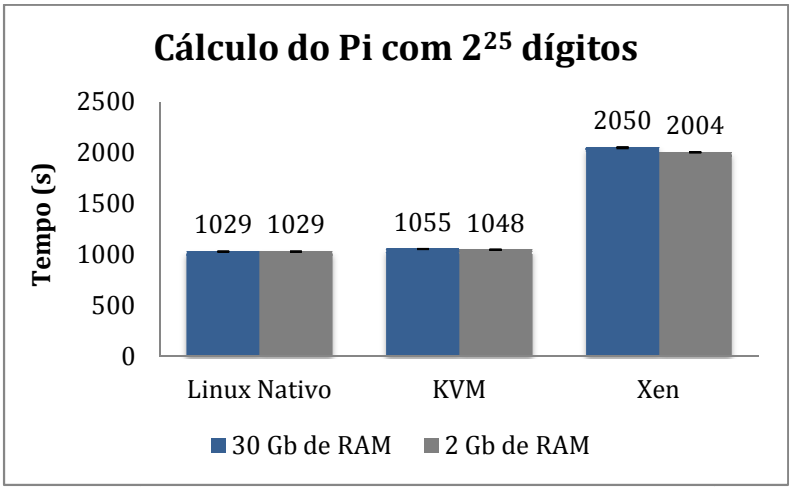


Figura 11 – Cálculo do Pi com 2^{25} casas decimais.

A Figura 11 mostra que o desempenho do KVM foi inclusive muito semelhante ao encontrado no Linux nativo. Isso aconteceu porque o hipervisor do KVM é implementado no próprio *kernel* do Linux hospedeiro e as máquinas virtuais KVM são vistas por esse núcleo como processos aguardando para serem escalonados para utilizar a CPU, pelo agendador de tarefas do SO, que também está implementado no *kernel*. No Xen, a sobrecarga do hipervisor é maior que a do KVM porque ele não é implementado no *kernel* do Linux nativo. Por isso, apesar do hipervisor ser um módulo executado no modo de acesso de núcleo, ele não é capaz de escalonar o uso da CPU para suas VMs de forma direta como acontece no KVM. Ou seja, primeiro ele mesmo precisa ser escalonado pelo agendador de tarefas do SO hospedeiro, para então encaminhar de forma controlada as chamadas de sistema que os SOs das VMs fizeram para a CPU.

3.2.2 *Benchmark* STREAM 5.1 para medir a sobrecarga de memória

Para avaliar o desempenho de memória do Xen e do KVM, o *benchmark* STREAM 5.1 [37], desenvolvido por McCalpin [25], foi utilizado. Com ele é possível, a partir do conhecimento detalhado do processador utilizado, construir um código de teste de memória simples e eficiente. Isso porque o pesquisador consegue otimizar o benchmark em função dos tamanhos dos caches dos processadores, para que eles não influenciem as contagens das operações de ponto flutuante e de acesso (leitura/escrita) dos dados na memória principal. Ou seja, é possível medir com precisão a taxa de transferência de blocos de dados para e da memória enquanto diversas operações aritméticas são realizadas, sem que as operações realizadas no cache local dos processadores influenciem os resultados encontrados.

O servidor Dell C6100, utilizado nos testes, possui CPUs Intel Xeon L5600 de seis núcleos e 12 MB de cache por processador (núcleo). Por esse motivo, o STREAM foi otimizado para que os blocos de dados utilizados nas operações aritméticas fossem formados por matrizes de 6 milhões de elementos por processador, a fim de que os 12 MB de memória cache de cada processador não interferissem no resultado dos testes [25]. Assim, os blocos de dados manipulados pelo processador durante os testes possuíam tamanho suficiente[2] para garantir que cada processador realizasse as operações aritméticas fazendo uso (cópia e escrita de dados) da memória principal, em lugar de utilizar apenas sua memória cache local. Com um tamanho de matrizes de dados suficiente para encher a memória cache de cada processador, os resultados não foram mascarados por atividades de "processamento local", onde a CPU não precisa copiar/escrever blocos de dados na/da memória principal a todo instante, porque salvou parte dos dados em sua memória cache.

A ferramenta também permite que a largura de banda sustentável, e não apenas picos de largura de banda ótimos de acesso à memória, seja medida. Para isso, ela exibe as medidas da taxa de transferência de blocos de dados de uma posição da memória para outra, depois que quatro operações de leitura e escrita na RAM são realizadas. Na primeira operação executada pelo *benchmark*, a taxa de transferência de blocos de dados de uma posição da memória para outra sem que operações aritméticas sejam realizadas (*bcopy*) é medida. Depois disso, operações aritméticas simples são inseridas nas operações de leitura e escrita de dados na memória e uma nova medição é realizada. Na terceira fase do teste, mais um operando aritmético é acrescentado nas operações e na quarta fase, operações de multiplicação e adição sobrepostas são realizadas e copiadas para/da memória pelas CPUs. Para medir a largura de banda sustentável decidimos considerar apenas as taxas de transferência das operações de leitura e escrita realizadas na quarta fase de cada teste, porque nesse momento as ações de leitura/escrita mais complexas e mais custosas estavam sendo realizadas, depois da RAM já ter sido enchida e esvaziada mais de uma vez. Por isso, apenas a média das taxas medidas nesta fase foi inserida na Figura 12.

Escrever blocos de dados na memória de forma sequencial, sem que exista intervalo entre as matrizes, é uma ação mais custosa que a escrita de blocos de dados soltos porque é necessário inserir/ler cada nova matriz a partir do ponto onde a anterior terminou de ser escrita/lida. Por esse motivo, além de configurar o tamanho dos blocos de dados, o código também foi alterado para que o *offset* entre as matrizes fosse igual a zero [25].

[2] Uma matriz com 2 milhões de elementos é grande o suficiente para satisfazer as regras de execução de um sistema que possui caches de até 4 MB, por exemplo.

Para medir os tempos de execução, um relógio externo foi utilizado porque os tempos medidos dentro dos ambientes virtualizados não são confiáveis, pois se baseiam na emulação dos dispositivos de tempo reais, uma vez que os SOs das VMs não conseguem acessar de forma direta as interrupções e os dispositivos do relógio físico [24]. Além disso, como as VMs foram criadas com 16 CPUs, o código otimizado do STREAM foi compilado com o recurso de programação paralela OpenMP (*Open Multi-Processing*) para que múltiplas *threads* pudessem ser executadas a fim de que todas as CPUs fossem ocupadas durante o *benchmark* de memória [29].

Cem baterias de teste foram realizadas em cada rodada para que a melhor largura de banda obtida fosse registrada. O primeiro valor medido foi excluído porque com a memória vazia um pico de largura de banda ótimo não sustentável poderia mascarar o resultado dos testes. O intervalo de confiança de 95% e a média de 15 rodadas de testes são apresentados na Figura 12.

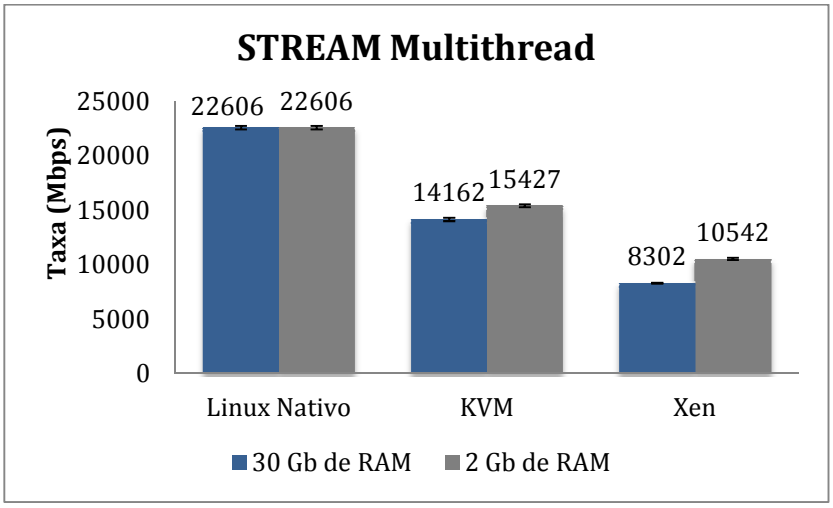


Figura 12 – Largura de banda sustentável no acesso a memória.

No Xen, a memória é fisicamente partilhada entre as máquinas virtuais no momento em que eles são criados, para garantir o isolamento; mas a arquitetura permite que um limiar seja configurado para que as VMs possam solicitar memória adicional à plataforma de virtualização quando for necessário. Um domU também pode reduzir sua reserva de memória, quando ela não estiver em uso, devolvendo as páginas de memória. Ou seja, cada SO convidado realiza sua própria paginação de memória no Xen porque o hipervisor lhes dá direito de leitura nas tabelas de página do hardware, privando-os apenas do direito de atualizações, que só podem ser executadas pelo próprio hipervisor depois de agrupadas e validadas.

Isso é feito da seguinte forma. Todas as vezes que um SO convidado cria uma nova tabela de página de hardware ele a inicializa e a aloca a partir de sua própria reserva de memória; e registra isso no Xen. Logo depois, o SO convidado faz um *downgrade* de permissão e deixa de possuir o privilégio de gravação na tabela. A partir desse momento qualquer nova atualização deve ser requisitada ao hipervisor, que a realiza depois de validá-la e autorizá-la. Dessa maneira se garante que um SO convidado conseguirá mapear apenas as suas próprias tabelas de página, porém com o controle de gravação nas tabelas realizado apenas pelo hipervisor. Assim, uma VM Xen é capaz de utilizar apenas a memória e a alocação de disco que lhe foram garantidos ainda que, na para-virtualização, um domínio conheça os endereços de hardware de seus vizinhos, porque o hipervisor implementa um controle de acesso seguro aos *namespaces* de cada VM para ser capaz de verificar, por exemplo, se um *buffer* específico pertence à reserva de memória de um domínio que deseja acessá-lo [1].

O resultado dos testes realizados e exibidos na Figura 12, mostram que o controle implementado pelo hipervisor do Xen na paginação de memória é mais custoso que o realizado pela plataforma de virtualização KVM. Vimos anteriormente que no KVM o hipervisor é implementado no *kernel* do Linux, fazendo com que a interação entre um SO virtualizado e as tabelas de página de memória do hardware aconteçam sempre no modo privilegiado, apesar da emulação do hardware. Essa é a característica que faz com que o hipervisor do KVM seja mais eficiente que o do Xen no acesso à memória.

3.2.3 Efeito da arquitetura NUMA na eficiência de roteadores virtuais criados em x86

É possível observar que, como aconteceu no *benchmark* de CPU, o desempenho de memória foi pior na VM com maior quantidade de memória virtual – 30 GB, tanto no Xen quanto no KVM. Ou seja, a sobrecarga do hipervisor do Xen sobre a memória foi maior na VM01, com 30 GB de RAM e 16 CPUs virtuais; o que fez com que a largura de banda média das operações de escrita e leitura na RAM fosse menor (8302 Mbps) nessa máquina virtual do que a taxa encontrada na VM02 (10542 Mbps), com apenas 2 GB de RAM e 16 CPUs virtuais. Nos resultados obtidos no KVM, a taxa de transferência da VM01 (14162 Mbps) também foi menor que a da VM02 (15427 Mbps). Portanto, os resultados tanto do *benchmark* de CPU quanto de memória mostraram que o desempenho das VMs de 30 GB foi pior que o das VMs com apenas 2 GB. A primeira vista esse não era o resultado esperado, mas verificamos que o problema aconteceu porque as CPUs virtuais das VMs precisaram acessar

áreas da memória compartilhada fora dos limites da memória local de cada processador quando realizavam as operações demandadas pelas ferramentas de *benchmark*.

Os primeiros computadores utilizavam a arquitetura UMA (*Uniform Memory Access*) em que todos os processadores acessavam a memória compartilhada através de um mesmo barramento (Fig. 13).

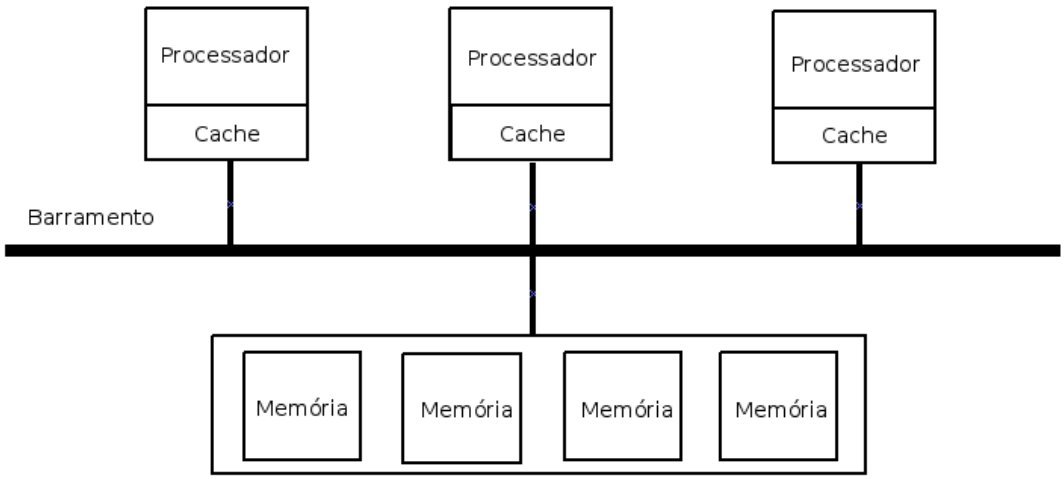


Figura 13 – Arquitetura UMA (*Uniform Memory Access*).

Quando o barramento que permitia o acesso à memória era compartilhado, o tempo de acesso a memória tendia a ser constante, independente do módulo de memória que continha um dado que estava sendo buscado pela CPU [30]. Nas máquinas modernas, com grande quantidade de RAM e processadores, as CPUs são integradas a controladores de memória para permitir a criação da arquitetura NUMA (*Non-Uniform Memory Access*) que pode ser construída em nível de hardware ou por software. Para otimizar o acesso de um processador à memória, a arquitetura NUMA divide os diversos núcleos processadores de uma máquina e a memória em pedaços, para formar agrupamentos de nós NUMA. Caso um processador precise alocar espaços de memória de fora do seu grupo, ele pode fazer o acesso remoto ao espaço de memória de outro nó; todavia a velocidade desse acesso à memória é muito menor que a alcançada quando um núcleo de processamento acessa espaços de memória locais – de seu próprio nó NUMA (Fig. 14).

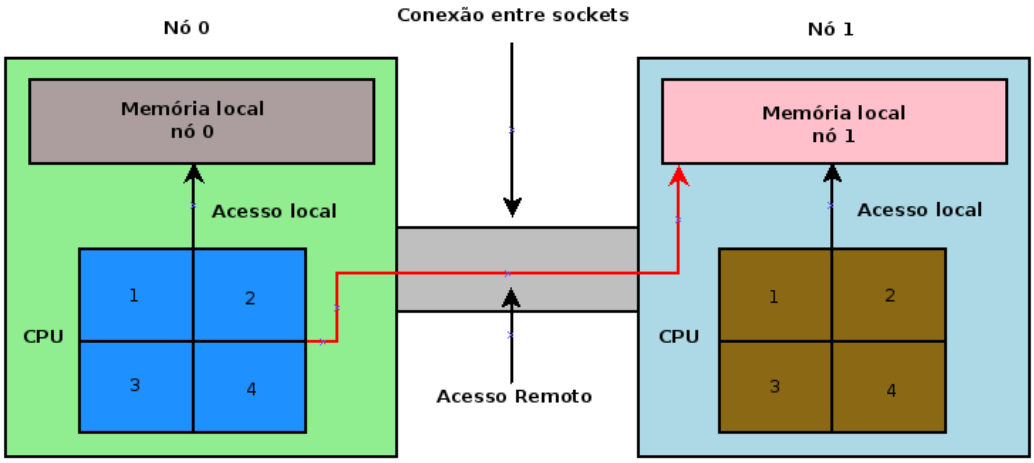


Figura 14 – Arquitetura NUMA (*Non-Uniform Memory Access*).

Os acessos dos processadores virtuais a espaços de memória remotos (NUMA) tornaram o desempenho das VMs com 30 GB pior que o das VMs com apenas 2 GB de memória. Cada máquina, de acordo com a quantidade de núcleos processadores e RAM, terá um valor máximo de memória RAM que pode ser alocada para as máquinas virtuais sem que o desempenho de acesso a memória e de CPU das mesmas seja prejudicado. Portanto, apesar de sabermos que o desempenho de um roteador é melhor quanto maior for a quantidade de memória e CPU disponibilizadas para ele, esse fato não se aplica em ambientes virtualizados. Nos diversos testes que realizamos, com diferentes quantidades de memória e CPU alocados para a VM de testes, encontramos o melhor resultado quando ela possuía no máximo 2 GB de memória. Isso aconteceu porque o limiar NUMA na máquina utilizada nos testes era igual a 50 GB / 24 núcleos = aproximadamente 2 GB.

3.2.4 Eficiência do Xen e do KVM sem o efeito negativo da arquitetura NUMA

Depois de entender o efeito que a arquitetura NUMA exerce sobre a eficiência em máquina virtuais criadas na plataforma x86, decidiu-se deixar de lado a máquina de 50 GB e 24 núcleos de processamento. A partir desse ponto as análises de desempenho do Xen e do KVM foram realizadas em três servidores Dell PowerEdge 2950, CPU Quad-Core Intel Xeon com 4 MB de cache por núcleo processador, 8 GB de memória e 876 GB de disco rígido, além de duas interfaces de rede de 1 Gbits/s [7]. Nesse cenário, o limiar NUMA ainda é de 8 GB / 4 núcleos = 2 GB. Além do mesmo hardware, as três máquinas possuíam também o mesmo SO – CentOS release 6.4 e *kernel* Linux 2.6.32-358.2.1 de 64 bits. O KVM e o Xen foram instalados nesse Sistema Operacional (CentOS 6.4) e a versão do Xen, em todos os testes realizados, continuou sendo a 4.2.1. Os *benchmarks* de CPU e memória, nos mesmos

moldes dos testes anteriormente descritos, voltaram a ser realizados em uma VM com 2 GB de RAM e 4 CPUs virtuais. Dessa vez, o Super Pi foi executado para que o número Pi com 2^{22} casas decimais fosse calculado. A média de quinze rodadas de testes, com um intervalo de confiança de 95%, é apresentada na Figura 15a. No *benchmark* de memória as mesmas cem baterias de teste foram realizadas em cada rodada de quatro operações de leitura e escrita na RAM. O valor médio da largura de banda de acesso a memória, também depois de quinze rodadas de teste e assumindo um intervalo de confiança de 95%, pode ser visualizado na Figura 15b. Pelas razões anteriormente explicadas, de acordo com o esperado, os resultados do KVM continuaram melhores que os do Xen.

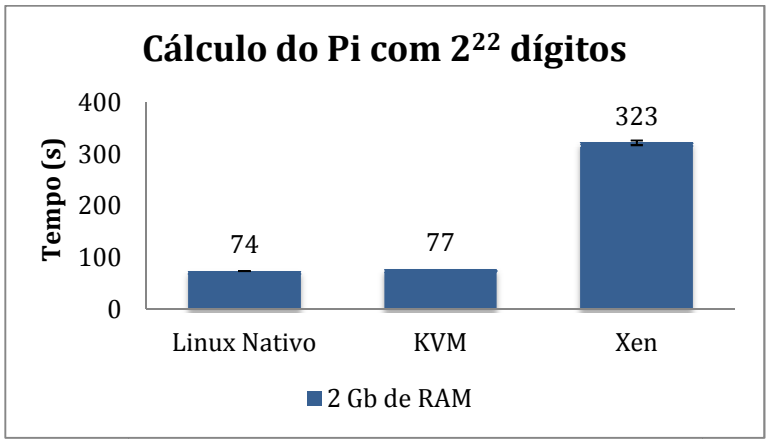


Figura 15a – Avaliação da sobrecarga de CPU das plataformas de virtualização.

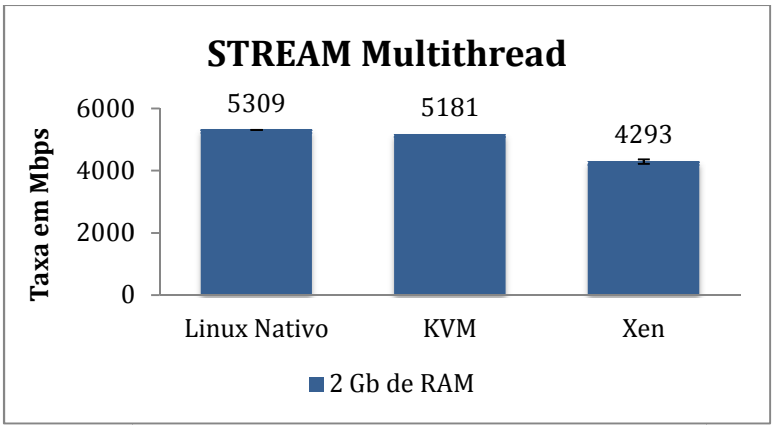


Figura 15b – Avaliação da sobrecarga de memória das plataformas de virtualização.

Mostramos que eficiência é um requisito importante para o bom desempenho dos roteadores virtuais criados em uma plataforma de virtualização. Ou seja, espera-se que a taxa das operações de E/S de rede do KVM sejam melhores que as encontradas no Xen, quando um *benchmark* de rede for executado, porque a sobrecarga de seu hipervisor sobre a memória e a CPU é menor que a exercida pelo Xen. Para verificar se isso é verdade executamos o Iperf

para comparar a capacidade de encaminhamento e roteamento de uma VM Xen e KVM com 2 GB de RAM e 4 CPUs virtuais.

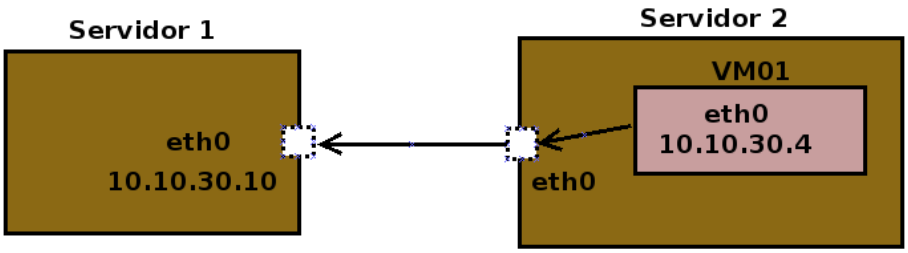


Figura 16 – Topologia do teste ponto-a-ponto.

Medimos a taxa de encaminhamento em uma conexão ponto-a-ponto, entre uma máquina geradora de tráfego para uma máquina receptora instalada com o Linux nativo 2.6.32-358.2.1 de 64 bits, com pacotes TCP de 64 bytes para que o processamento da máquina geradora fosse sobrecarregado. Em cada uma das 15 baterias de teste o tráfego de pacotes durou 10 segundos e nenhum parâmetro do TCP foi alterado. Ou seja, utilizamos a configuração padrão do Iperf em todos os testes realizados. A topologia montada pode ser visualizada na Figura 16. A geradora de tráfego no primeiro instante foi o próprio Linux nativo, depois a VM Xen e por fim a VM KVM, e na receptora de tráfego foi instalado o Linux nativo. Foram realizadas quinze rodadas de teste, assumindo um intervalo de confiança de 95%, e as taxas médias de encaminhamento de pacotes são apresentadas nas Figuras 17a e 17b. Em todos os testes realizados nesse trabalho as taxas de transmissão foram medidas sempre na máquina geradora de tráfego, onde o cliente do Iperf foi executado, e a taxa de recepção foi medida na máquina receptora, onde o servidor do Iperf foi executado.

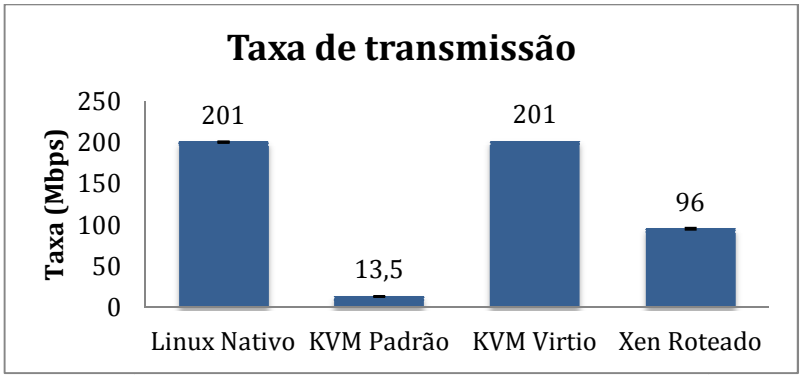


Figura 17a – Taxa de transmissão de pacotes de 64 bytes na máquina geradora da topologia ponto-a-ponto.

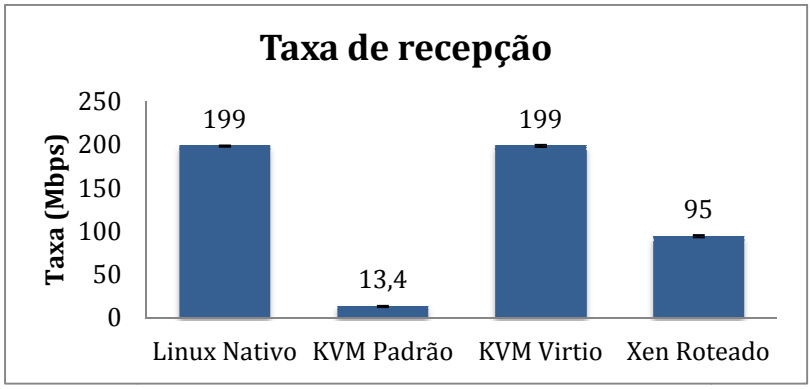


Figura 17b – Taxa de recepção de pacotes de 64 bytes na máquina receptora da topologia ponto-a-ponto.

Nos testes realizados, a libvirt foi configurada para que o padrão de virtualização VIRTIO [22] permitisse ao driver do dispositivo de rede do SO virtualizado realizar operações de E/S de rede cooperando com o QEMU. Ou seja, permitiu-se ao driver de rede do SO virtualizado saber que estava sendo executado em um ambiente virtual para que, da mesma forma como acontece na para-virtualização realizada pelo Xen, ele pudesse acessar o modo de kernel para realizar encaminhamento de pacotes fazendo chamadas privilegiadas para os dispositivos de rede. Caso isso não fosse feito, o elevado custo da emulação dos dispositivos de rede, feita pelo QEMU, tornaria o desempenho do encaminhamento de pacotes do KVM muito inferior ao do Xen. Nas Figuras 17a e 17b é possível visualizar o baixo desempenho do KVM quando o SO virtualizado executa as operações de E/S de rede no modo de usuário do SO hospedeiro, sem saber que sua interface de rede é virtual (KVM Padrão). Nelas também é possível verificar que o desempenho de encaminhamento do KVM, tanto na transmissão quanto na recepção de pacotes, foi semelhante ao encontrado no Linux nativo quando executado com VIRTIO (KVM VIRTIO). Assim, com pacotes pequenos, o desempenho de encaminhamento do Xen foi muito menor que o do KVM e do Linux nativo. Porém, as Figuras 18a e 18b mostram que seu desempenho de encaminhamento foi semelhante ao do KVM, e praticamente igual ao do Linux nativo, quando pacotes grandes de 1500 bytes foram utilizados para saturar o enlace físico. Na recepção de pacotes é possível observar que o Xen chegou a apresentar um desempenho de encaminhamento um pouco superior ao do KVM com VIRTIO.

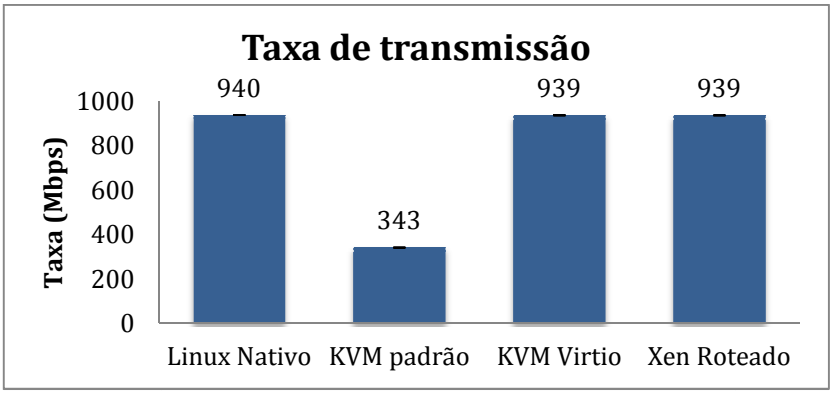


Figura 18a – Taxa de transmissão de pacotes de 1500 bytes na máquina geradora da topologia ponto-a-ponto.

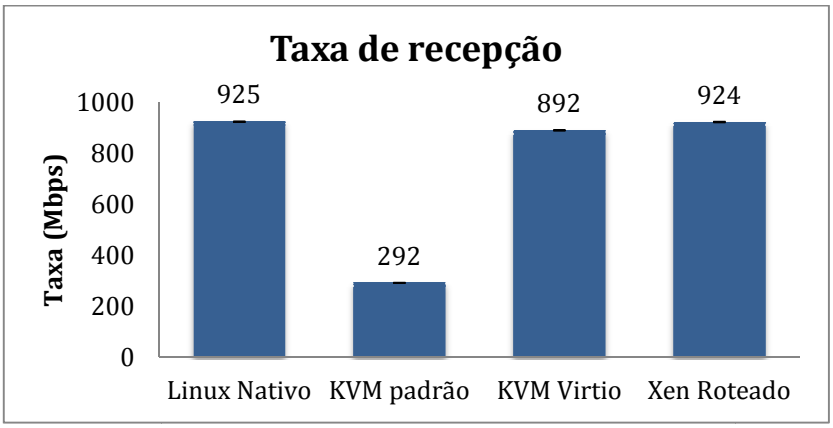


Figura 18b – Taxa de recepção de pacotes de 1500 bytes na máquina receptora da topologia ponto-a-ponto.

Alteramos a topologia para tornar a VM um roteador (Fig. 19) a fim de verificar se com o custo de roteamento de pacotes da rede A (172.16.50.0/24) para a rede B (10.10.50.0/24) o desempenho do Xen continuaria semelhante ao do KVM e Linux nativo. O mesmo tráfego de pacotes de 1500 bytes foi gerado, do servidor 3 para o servidor 1, passando pela VM01 instalada no servidor 2. Tanto o servidor 3 quanto o servidor 1 foram instalados com o Linux nativo.

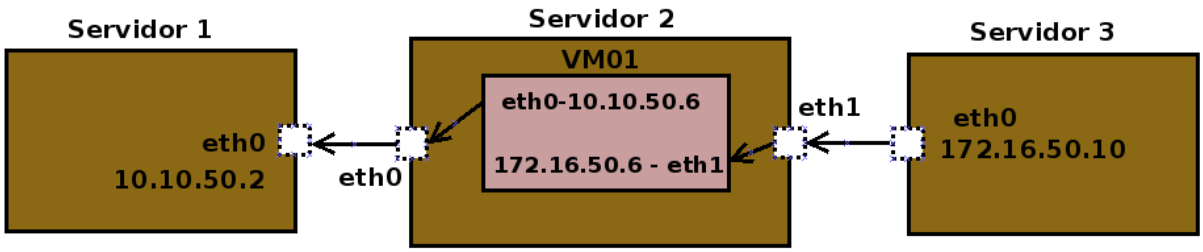


Figura 19 – Topologia do teste de roteamento.

As Figuras 20a e 20b mostram que o desempenho do roteador virtual criado no KVM é melhor que o roteador virtual criado no Xen. O KVM obteve melhor resultado na transmissão dos dados quando as operações de E/S foram virtualizadas pelo VIRTIO. No

Xen, que não realiza virtualização de operações de E/S, o desempenho foi pior provavelmente porque a tomada de decisão de roteamento realizada pelo roteador virtual do Xen demanda maior consumo de memória e maior processamento que o simples encaminhamento de pacotes em uma conexão ponto-a-ponto, onde os pacotes são encaminhados em função dos endereços de camada 2 dos dispositivos interligados. O custo é maior no Xen porque para serem roteados os pacotes atravessam o hipervisor e entram e saem do dom0 duas vezes. Na primeira vez, os pacotes entram pela interface física eth1 do servidor 2 (Fig. 19) que é visualizada apenas pelo dom0. Depois de entrar no dom0 os pacotes são direcionados, através do hipervisor (Fig.10), para dentro da VM para que a tabela de repasse do roteador virtual seja consultada. Após verificar que a rota de saída aponta para sua interface eth0, o roteador virtual encaminha o pacote de volta para o dom0. Portanto, o pacote volta a atravessar o hipervisor (Fig. 10) antes de alcançar a interface física eth0. A dupla sobrecarga do hipervisor do Xen torna a taxa de recepção no roteamento de pacotes, 553 Mbps, exibida na Figura 20b, bem menor que a encontrada quando os pacotes eram apenas encaminhados, 924 Mbps, exibida na Figura 18b.

No KVM, em primeiro lugar, quando um pacote chega na interface física eth1 do servidor 2 (Fig. 19) ele não precisa atravessar uma VM de drivers para chegar na interface do roteador virtual. O pacote atravessa apenas o hipervisor KVM que está implementado no próprio kernel do SO hospedeiro (Fig. 6). Por esse motivo, o roteamento de um pacote que entrou pela interface física da máquina para a interface de rede da VM, que está declarada dentro de uma árvore de diretórios /dev/kvm exclusiva, é realizado pelo mecanismo de encaminhamento do kernel. Essa é uma das razões para que a sobrecarga de memória e CPU seja menor que a encontrada no Xen. A sobrecarga também é menor no KVM quando o pacote precisa deixar a máquina virtual para sair pela interface eth0 do servidor 2 (Fig. 19), desde que o VIRTIO seja utilizado permitir aos drivers de rede da VM interagirem com o QEMU para serem executados no modo de kernel. Portanto, os resultados das Figuras 20a e 20b coincidem com os exibidos nos *benchmarks* de memória e CPU, realizados anteriormente, e mostram que no requisito eficiência, importante em redes que demandam roteamento virtual, o KVM é melhor que o Xen.

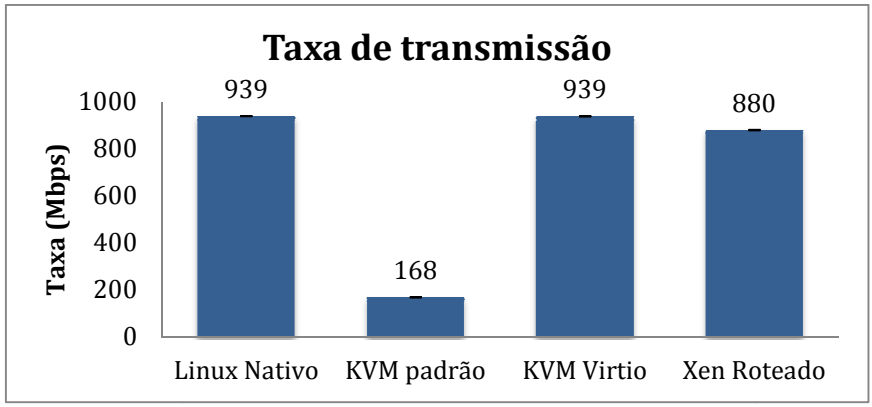


Figura 20a – Taxa de transmissão de pacotes de 1500 bytes na máquina geradora quando a VM é um roteador.

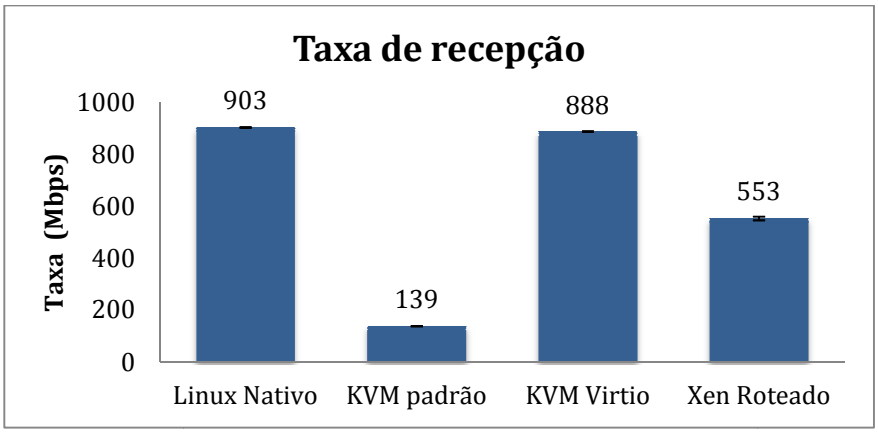


Figura 20b – Taxa de recepção de pacotes de 1500 bytes na máquina receptora quando a VM é um roteador.

3.3 Isolamento das plataformas de virtualização Xen e KVM

No Xen, apenas um roteador virtual pode provocar variações no consumo de memória e CPU do dom0 a ponto de prejudicar o desempenho de todas as VMs em execução na mesma máquina física. Por esse motivo, o isolamento entre diferentes redes virtuais criadas no Xen não é garantido. Como o KVM não possui um domínio de drivers compartilhado pelas máquinas virtuais criadas, seu isolamento é nativamente melhor que o encontrado no Xen. Para verificar o problema de isolamento do Xen, medimos o consumo de processamento no dom0, com a ferramenta top, quando pacotes foram transferidos de um domU para outro domU do mesmo nó físico, seguindo a mesma abordagem encontrada em Fernandes [11]. O Iperf foi a ferramenta utilizada nos testes, que voltaram a ser realizados com pacotes de 1500 bytes, nos mesmos moldes das avaliações realizadas anteriormente. A máquina física ainda era a mesma Dell 2950 dos testes realizados na Seção 3.2, o intervalo de confiança assumido foi de 95%, as VMs possuíam 2 GB de RAM e 4 CPUs virtuais e o dom0 foi criado sem restrição de memória.

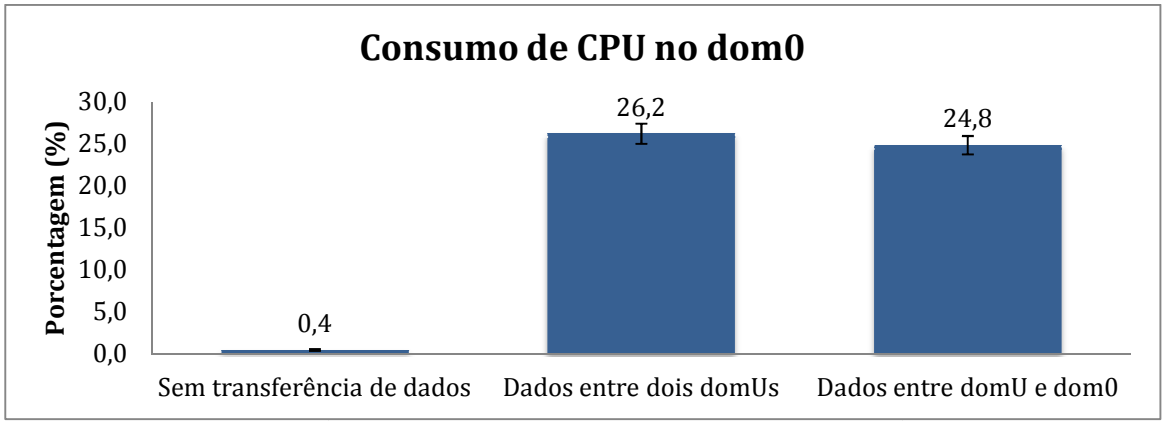


Figura 21 – Consumo de CPU no dom0 quando existe transferência de dados entre VMs.

Executamos cada rodada de testes por apenas 10 segundos, de acordo com a configuração padrão do Iperf, porque acreditamos não ser necessário variar a carga de CPU do dom0 até 100% para mostrar que o problema de isolamento existe. Os resultados da Figura 21 indicam que é possível esgotar os recursos do dom0 através de uma VM maliciosa porque o tráfego de pacotes entre dois domUs, e entre um domU e o dom0, fizeram a carga de CPU do domínio de drivers variar de aproximadamente 0 para valores superiores a 20%. Isso acontece porque o hipervisor do Xen controla o acesso do dom0 ao hardware, mas não gerencia o que acontece dentro desse domínio de drivers nem o acesso que os domUs fazem ao dom0 quando precisam realizar operações de E/S. Por esse motivo, apenas uma VM do Xen pode quebrar o isolamento e comprometer o desempenho de todas as outras VMs da máquina física afetando todas as redes virtuais criadas. Portanto, existe problema de isolamento no Xen e o dom0 é um gargalo quando a configuração do Xen roteado é utilizada para criar ambientes de roteamento virtual.

Até aqui mostramos que o KVM é mais eficiente que o Xen no consumo dos recursos físicos da máquina, no roteamento de pacotes e no isolamento entre as VMs, apesar do Xen ser mais flexível que o KVM por poder ser executado em máquinas com CPUs que não possuem suporte a virtualização. No próximo capítulo, vamos apresentar a plataforma de virtualização OpenFlow que permite virtualizar planos de dados e de controle e separá-los em diferentes dispositivos de rede. Com o OpenFlow é possível criar, apagar e migrar um plano de dados de uma rede virtual entre diferentes nós físicos [35]. Ou seja, enquanto o Xen e o KVM viabilizam a extensibilidade das máquinas virtuais que criam, permitindo que elas sejam migradas de um nó físico para outro, o OpenFlow é uma ferramenta de virtualização que viabiliza a extensibilidade de planos de dados. Com ele é possível criar planos de controle

distribuídos que utilizam um PD compartilhado configurado em apenas uma máquina ou PDs distribuídos em diversos SOs.

4 VIRTUALIZAÇÃO DE PLANO DE CONTROLE E DE DADOS COM OPENFLOW

Nos últimos anos as capacidades de processamento, memória, disco e encaminhamento de pacotes das máquinas fabricadas aumentaram significativamente. Nos *Datacenters* das empresas é fácil perceber isso quando comparamos a capacidade de hardware por unidade de rack (Una ou U) que existia há pouco tempo com as capacidades atuais. Há dez anos, por exemplo, era comum encontrar um servidor com quatro núcleos de processamento, 8 GB de RAM e 500 GB de disco ocupando quatro Us de um rack. Hoje, em apenas dois Us é possível utilizar uma máquina Dell PowerEdge C6100 que oferece até quatro nós servidores, oito soquetes com CPUs Xeon de 6 núcleos cada, RAM com até 192 GB, capacidade máxima de armazenamento local de 3 TB e possibilidade de uso de duas portas 10 Gigabit Ethernet [8]. Máquinas desse tipo, em apenas 2 Us de rack são capazes de oferecer nós com 48 núcleos de processamento e 192 GB de RAM, forçam os administradores de rede a rever a maneira como seus servidores são alocados aos projetos. Assim, para não subutilizar um nó físico de 50 GB de RAM e 24 núcleos de processamento, semelhante ao utilizado nos primeiros testes desse trabalho, diversos estudos e técnicas de virtualização têm sido desenvolvidas nos últimos anos.

Todos estão interessados em virtualizar servidores para otimizar ainda mais o espaço físico que utilizam em racks, diminuir o consumo e os gastos com energia elétrica e refrigeração em Centros de Processamento de Dados (CPDs) e *Datacenters*, etc. No entanto, de acordo com os pluralistas, a virtualização também nos permite resolver problemas antigos que o núcleo da rede possui, tais como os problemas de segurança, privacidade, suporte a nós móveis, etc. Isso é possível desde que o núcleo da rede também seja virtualizado. Assim, diversas arquiteturas de rede, com pilhas de protocolos diferentes, podem ser executadas em redes virtuais que compartilham os mesmos equipamentos. Através desse tipo de virtualização também é possível partir do zero para definir novas lógicas de encaminhamento de pacotes e novos protocolos para o núcleo da rede, mais adequados às necessidades das aplicações atuais, sem que seja necessário interromper ou afetar a lógica atualmente implementada pelo protocolo IP no núcleo da Internet.

Quando um roteador físico é utilizado como *gateway* externo das VMs criadas dentro de uma máquina x86 o custo de tomada de decisão de encaminhamento de pacotes inserido pelas ACLs é minimizado porque o dom0 do Xen, por exemplo – que é compartilhado por todos os domUs, não precisa executar essa tarefa. Como foi dito, isso acontece porque os fabricantes de roteadores modernos utilizam TCAMs para que as ACLs sejam processadas em

velocidade de linha. No entanto, se o roteador externo não apresenta suporte a virtualização, todas as redes virtuais criadas conseguirão executar apenas os protocolos de rede que esse roteador possui. Dessa forma, a flexibilidade para criar novas tecnologias e protocolos no núcleo da Internet fica comprometida. Além disso, testar novas soluções para o núcleo da rede apenas em PCs ou servidores x86 pode ser um problema, pois o desempenho de encaminhamento e roteamento dessas máquinas é muito baixo quando comparado às capacidades existentes nos equipamentos de rede modernos.

Por melhores que sejam, máquinas x86 não são construídas para possuir grande densidade de portas, necessárias em dispositivos que pretendem interligar os computadores de um *Datacenter* ou do CPD de uma universidade. Enquanto essas máquinas geralmente possuem menos de 10 interfaces de rede de 1 Gigabit Ethernet, já existem equipamentos de rede modulares, largamente utilizados em grandes *Datacenters*, que oferecem até 768 portas Ethernet de 1 Gigabit com taxas de transmissão de 10 Gbits/s, 40 Gbits/s e até 100 Gbits/s nas interfaces de interligação com outros comutadores e roteadores. Mesmo que uma máquina x86 possua duas interfaces de rede de 10 Gbits/s não é possível comparar seu desempenho com esses comutadores e roteadores modulares [4]. Por esse motivo, uma ferramenta de virtualização capaz de segmentar as redes de um roteador/comutador comercial para criar ambientes virtuais segregados, onde novas soluções de roteamento e encaminhamento possam ser implementadas, é extremamente importante. Assim, as propostas de pesquisadores para os problemas que encontramos no núcleo da rede podem ser validadas em equipamentos de produção, com tráfego de produção passando por esses equipamentos, sem que o tráfego de pesquisa interfira nos mesmos.

O OpenFlow é uma plataforma de virtualização que resolve os problemas levantados porque segmenta redes, para permitir a criação de ambientes virtuais com lógicas de encaminhamento de pacotes diferentes, em máquinas x86 e em roteadores e comutadores comerciais. Assim, um roteador comercial pode ser compartilhado por clientes concorrentes que desejam implementar novos protocolos no núcleo de uma rede de produção, desde que o fabricante do equipamento implemente suporte ao protocolo OpenFlow. Além disso, os fabricantes de hardware são capazes de implementar com facilidade o suporte à virtualização em seus dispositivos, porque suas tabelas de encaminhamento não precisam ser completamente alteradas. Elas continuam sendo utilizadas, embora a lógica de encaminhamento de pacotes não se baseie apenas no endereço IP de destino, mas na combinação de até 12 tuplas contidas no cabeçalho de um pacote OpenFlow.

4.1 Características da plataforma de virtualização OpenFlow

A plataforma de virtualização OpenFlow foi criada para possibilitar o experimento de novos protocolos no núcleo da rede utilizando os mesmos roteadores, *switches* (comutadores) pontos de acesso e servidores que estão em uso na Internet atual, sem que os novos protocolos interfiram no tráfego de produção. Assim, quando um equipamento oferece suporte ao protocolo OpenFlow, seu plano de dados (tabelas de encaminhamento e roteamento) pode ser compartilhado entre múltiplas redes virtuais que possuem lógicas de encaminhamento distintas. O plano de dados fica localizado no comutador e o plano de controle fica em um nó conhecido como controlador OpenFlow. A comunicação entre o controlador e o comutador, que o controlador gerencia e monitora, é realizada através de um canal seguro de comunicação, que deve ser estabelecido através de interfaces de controle existentes nesses equipamentos. E a lógica de encaminhamento de cada rede virtual é definida por seu controlador quando ele atualiza as tabelas de fluxo de seu(s) comutador(es) (Fig. 22).

Gude e outros [17] propuseram o controlador NOX que implementa um SO de redes e um conjunto de aplicações para permitir a interação com os comutadores. As aplicações do controlador, que podem ser executadas em uma máquina x86, interagem com o comutador quando é preciso decidir se um fluxo deve ou não ser adicionado em sua tabela de fluxo [23] [36]. Todas as vezes que os primeiros pacotes de um fluxo de dados chegam a um comutador OpenFlow ele verifica se o controlador já criou uma entrada para esse tráfego em sua tabela de fluxo. Caso isso ainda não tenha sido feito os primeiros pacotes são encaminhados para o controlador, através do canal seguro de comunicação, para que ele defina por quais comutadores e roteadores o encaminhamento e o roteamento deve ser realizado. Ou seja, o controlador processa os primeiros pacotes para inserir as rotas necessárias para esse fluxo de dados nas tabelas de repasse (no plano de dados) dos comutadores OpenFlow que fazem parte do caminho a ser percorrido [6]. Portanto, apenas os primeiros pacotes passam pelo controlador através do canal seguro.

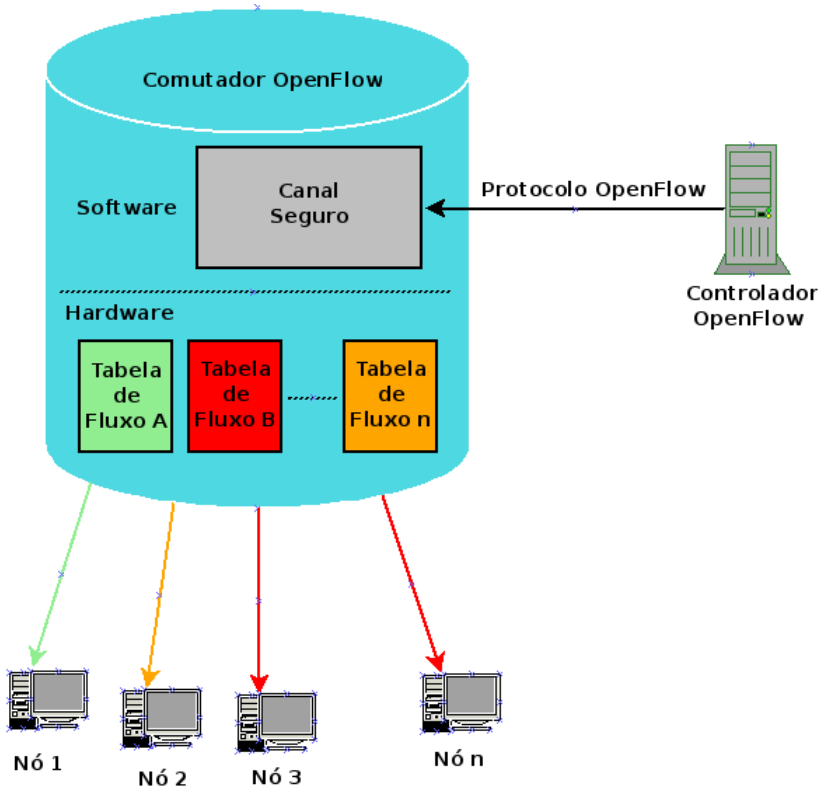


Figura 22 – Comutador OpenFlow com diferentes tabelas de fluxo gerenciadas pelo controlador remoto.

Depois que as tabelas de fluxo dos comutadores são atualizadas, os pacotes seguintes deixam de passar pelo controlador e passam a percorrer apenas os comutadores que já possuem rotas bem definidas para o fluxo de dados em questão. Ou seja, com a ferramenta de virtualização OpenFlow, o plano de dados localizado no comutador, que possui a lógica e as regras de encaminhamento dos pacotes, está separado do plano de controle, que fica no controlador; e apenas o nó controlador possui o plano de controle para que só ele seja capaz de gerenciar as tabelas de repasse dos planos de dados dos comutadores OpenFlow. Idealmente deve-se utilizar um controlador para cada rede virtual que se deseja criar [6]. Além disso, quando vários controladores usam o mesmo comutador OpenFlow, o FlowVisor [23] [36] precisa ser utilizado para atuar como proxy entre os controladores e o comutador identificando o tráfego de pacotes de cada rede – tráfego 1 pertence à rede (controlador) A e os pacotes do tráfego 2 pertencem à tabela de fluxo da rede B (Fig. 22). Quando diversos controladores compartilham um comutador OpenFlow dessa forma, a eficiência do tráfego de pacotes é maior [26].

4.2 Como um comutador OpenFlow encaminha pacotes

A lógica de encaminhamento de um comutador OpenFlow pode ser completamente diferente da implementada hoje na Internet, onde o roteamento é realizado em função do

endereço IP de destino inserido no cabeçalho do pacote [40]. Com o OpenFlow é possível realizar o encaminhamento em função da combinação de um conjunto de tuplas contidas no cabeçalho do fluxo. Na primeira versão do protocolo OpenFlow, o cabeçalho dos fluxos foi criado com as 10 tuplas mostradas na Figura 23 [26].

Porta de Entrada	Endereço Ethernet Origem	Endereço Ethernet Destino	Tipo Ethernet	ID de Vlan	Endereço IP Origem	Endereço IP Destino	Protocolo IP	Porta TCP Origem	Porta TCP Destino

Figura 23 – Campos de cabeçalho para definir o fluxo em um comutador OpenFlow.

Posteriormente, o cabeçalho do OpenFlow recebeu mais duas tuplas: Vlan Priority e IP ToS. Assim, a definição de um fluxo em um comutador OpenFlow passou a poder acontecer em função da combinação das 12 tuplas mostradas na Figura 24. O protocolo também implementa o uso de máscaras de rede nas tabelas de fluxo e um contador para que a quantidade de bytes e a duração dos fluxos de dados sejam contabilizados. Além do encaminhamento de pacotes, diferentes ações podem ser executadas por um comutador, tais como: alteração de prioridade de um pacote, alteração de endereços de origem e destino, alteração de identificadores de Vlans, etc. [12] [2].

Porta de Entrada	Endereço Ethernet Origem	Endereço Ethernet Destino	Tipo Ethernet	ID de Vlan	Prioridade Vlan	Endereço IP Origem	Endereço IP Destino	Protocolo IP	ToS IP	Porta TCP Origem	Porta TCP Destino

Figura 24 – Novo cabeçalho para definir o fluxo em um comutador OpenFlow.

Com esse conjunto de tuplas é possível encaminhar os pacotes que pertencem a rede virtual A (Fig. 22) em função apenas do endereço IP de destino do pacote, para manter a compatibilidade com a rede que existe hoje, enquanto na rede virtual B o encaminhamento poderia ser realizado em função da combinação das tuplas Vlan ID e porta de destino. Ou seja, a cada rede virtual criada, pode-se partir do zero de acordo com a abordagem *clean slate* [34], uma vez que a lógica de roteamento implementada pelo controlador da rede pode ocorrer em função de qualquer combinação de tuplas. Assim, novos formatos de endereçamento independentes podem ser otimizados para atender às necessidades de aplicações específicas, sem que o encaminhamento de pacotes da Internet atual seja desfeito [6] [2].

4.3 Vantagens da integração do OpenFlow com o Xen e o KVM

Vamos pensar no problema de isolamento do Xen que ocorre devido ao compartilhamento do dom0 pelos domUs criados na máquina física. Mostramos que para realizar operações de E/S, os domUs dependem do bom desempenho do domínio de drivers

porque apenas ele possui acesso direto ao hardware (Fig. 7); porém o consumo de CPU do dom0 pode ser fortemente afetado pelo comportamento malicioso de uma VM prejudicando o desempenho de todas as outras máquinas virtuais que compartilham a mesma máquina física (Fig. 21). Para resolver esse problema de forma definitiva é necessário criar mecanismos que consigam segmentar de forma justa as operações de E/S de rede que as VMs Xen enviam para o dom0. Não vamos propor esse tipo de solução nesse trabalho, todavia, é possível melhorar o desempenho do roteamento de pacotes (Fig. 20a e Fig. 20b) em uma solução de redes virtuais montada com a plataforma de virtualização Xen.

Se o dom0 for utilizado como comutador OpenFlow de domUs que executam um software de controle NOX, os planos de dados das VMs serão copiados para o dom0 (Fig. 25b). Ou seja, as tabelas de repasse dos controladores (domUs) são copiadas para o comutador OpenFlow (dom0). Essa abordagem foi proposta por Mattos [23]. Assim, apenas os primeiros pacotes de um fluxo de dados irão atravessar os DomUs porque, uma vez definidas as rotas a serem seguidas, as instruções necessárias para o roteamento serão escritas nas tabelas de encaminhamento contidas no domínio de drivers. Os testes de roteamento realizados na Seção 3.2.4 foram refeitos com o Xen e o OpenFlow operando em conjunto para tornar o dom0 um plano de dados compartilhado [11] [23]. Também combinamos o KVM com o OpenFlow para fazer esses testes. Nesse caso, tornamos o Linux nativo um comutador OpenFlow. Nos dois casos, o controlador foi configurado dentro das VMs: Xen e KVM e quinze rodadas de testes foram executadas assumindo-se um intervalo de confiança de 95%.

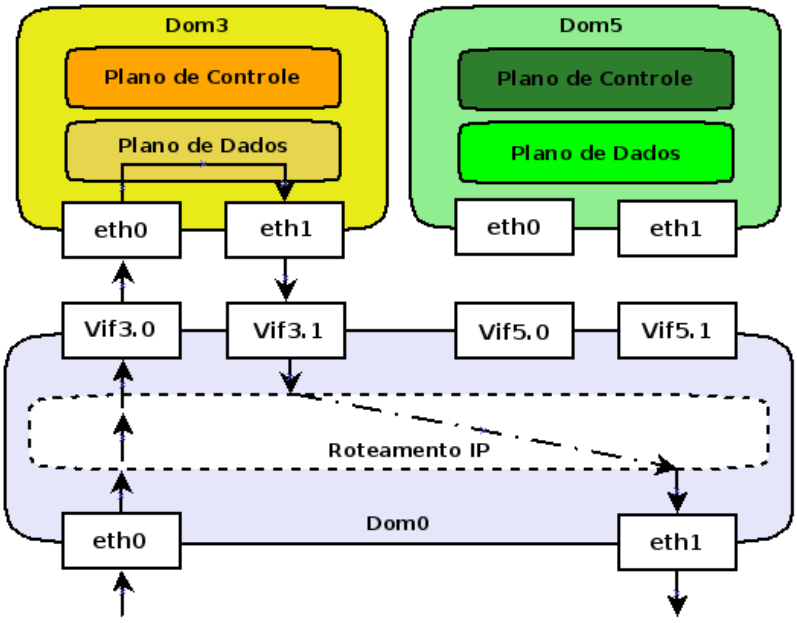


Figura 25a – Fluxo de dados no Xen roteado sem implementar o paradigma da separação de planos.

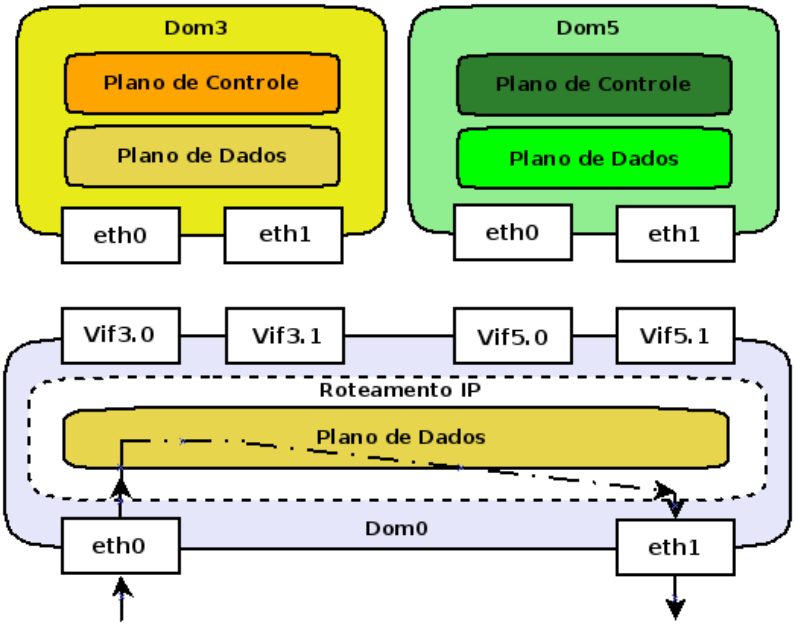


Figura 25b – Fluxo de pacotes quando o dom0 é transformado em um plano de dados compartilhado.

A Figura 26 mostra que o desempenho foi melhor quando o paradigma da separação de planos (Fig. 25b) foi utilizado, porque os pacotes deixaram de atravessar o hipervisor duas vezes como acontece na configuração padrão do Xen roteado (Fig. 10 e Fig. 25a). Sem a dupla sobrecarga do hipervisor, o dom0 utilizou as ferramentas de roteamento do *kernel*, da mesma forma que acontece no Linux nativo, e a eficiência do Xen no roteamento de pacotes melhorou significativamente, tornando-se praticamente semelhante a do Linux nativo (Fig. 26).

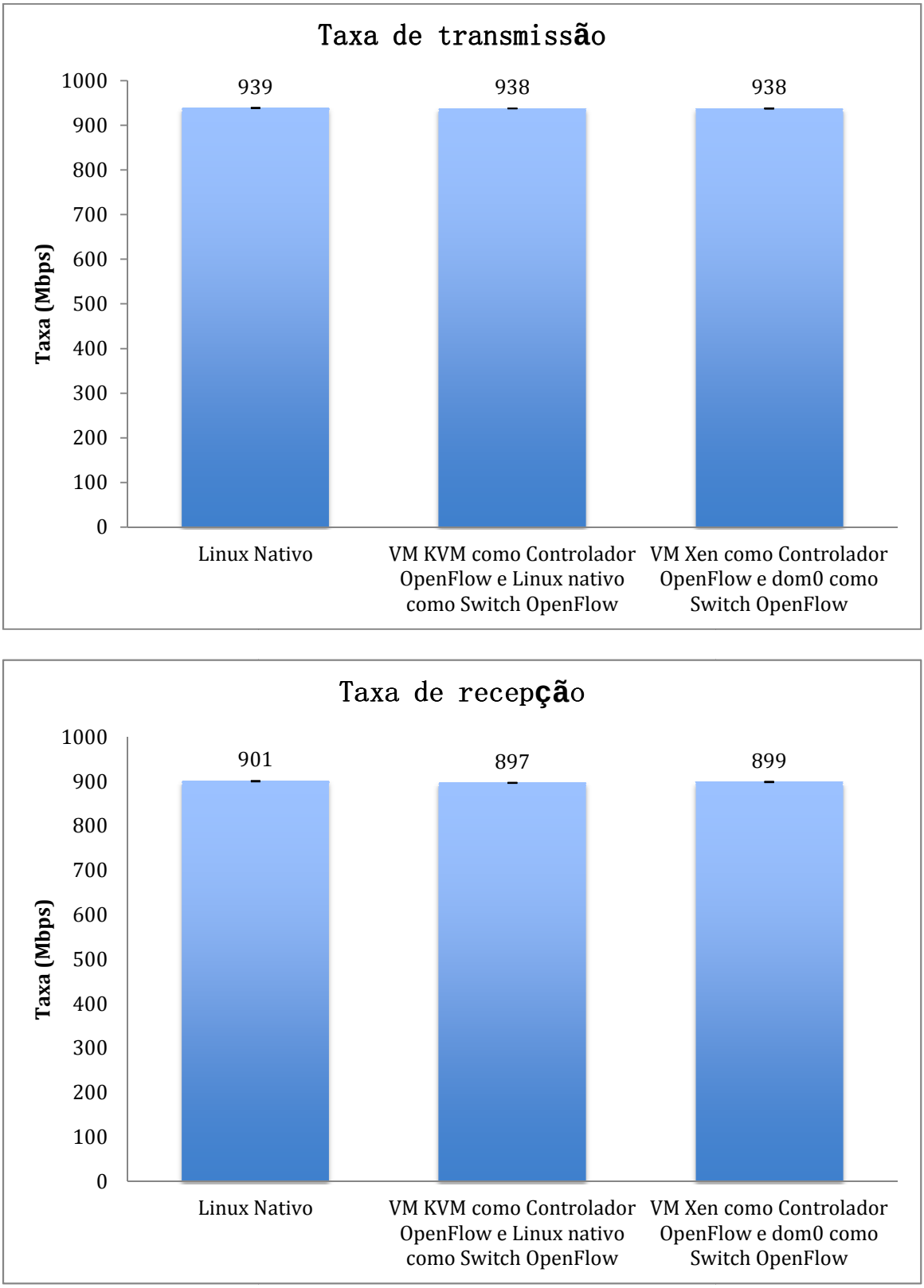


Figura 26 – Fluxo de dados combinando o OpenFlow com o Xen e com o KVM.

Os resultados da Figura 26 também mostram que, quando o KVM e o OpenFlow foram combinados para tornar a VM do KVM um controlador e o Linux nativo um comutador OpenFlow, os desempenhos de transmissão e recepção dos pacotes roteados praticamente se

igualaram ao do Linux nativo. Isso aconteceu porque, com essa técnica, apenas os primeiros pacotes de um fluxo de dados são enviados para as VMs (controladores) e o roteamento propriamente dito é realizado pelo comutador OpenFlow – que no KVM é o próprio Linux e no Xen é o dom0. Ou seja, o desempenho do OpenFlow foi praticamente semelhante ao do Linux nativo tanto quando ele foi combinado com o Xen quanto com o KVM. Portanto, se VMs são utilizadas como controladores OpenFlow e o Linux nativo (KVM) ou o dom0 (Xen) são configurados como comutadores OpenFlow, o desempenho do encaminhamento de pacotes das redes virtuais criadas é aumentado. Além disso, também é possível observar que o uso de controladores virtuais permite a extensibilidade do plano de dados. Ou seja, caso um controlador precise ser migrado de um nó físico para outro é possível fazer isso sem interromper o funcionamento dos fluxos em operação na rede virtual, porque os pacotes continuam sendo roteados pelos comutadores que possuem uma cópia do plano de dados do controlador.

Podemos utilizar o OpenFlow em conjunto com o Xen ou com o KVM para que cada máquina virtual criada atue como controlador de uma rede OpenFlow exclusiva. No entanto, também é possível criar diferentes redes OpenFlow em um dispositivo físico que não utiliza uma ferramenta de virtualização de hardware, tal como o Xen e o KVM. Para isso basta criar diferentes Vlans (domínios de broadcast) para que o fluxo de dados entre as diferentes redes seja segregado e um pesquisador não consiga enxergar o tráfego de outra rede de pesquisa. Portanto, enquanto a escalabilidade de uma plataforma de virtualização que segmenta recursos físicos, tal como o Xen ou o KVM, está relacionada ao máximo de VMs que ela consegue suportar, a do OpenFlow é definida pelo máximo de fluxos simultâneos que um controlador consegue gerenciar.

Vimos que apenas os primeiros pacotes de um fluxo de dados passam pelo controlador e poucos recursos de hardware são exigidos para que o processamento de um simples conjunto de fluxos aconteça em um comutador OpenFlow. Por esse motivo, a escalabilidade do OpenFlow, que é capaz de suportar centenas de redes virtuais paralelas, é muito maior que a do Xen e do KVM. Isso só deixa de ser verdadeiro se os fluxos de dados de todas as redes forem pequenos e novos a todo momento [6] [26] [36]. Caso isso aconteça, o desempenho do OpenFlow dependerá da capacidade de processamento da máquina em que o controlador foi instalado porque pacotes de novos fluxos a todo momento serão enviados para ela. Nessa situação, a capacidade dos enlaces, utilizados como canal seguro de comunicação entre o

controlador e os comutadores OpenFlow, também afetará o desempenho da plataforma de virtualização.

Com as técnicas apresentadas é possível permitir a programação do núcleo de uma rede de produção, que é uma ação muito mais poderosa que a realizada atualmente pelos administradores de rede quando executam apenas comandos de configuração em um roteador ou comutador. Assim, os problemas que possuímos no núcleo da Internet podem ser resolvidos, com soluções que partem do zero, mas sem que a rede tenha que ser totalmente reconstruída. Para que isso aconteça, os fabricantes de equipamentos de rede precisam oferecer suporte ao OpenFlow em seus roteadores e comutadores.

CONCLUSÕES

A importância do tamanho de um agrupamento NUMA no desempenho de uma VM

Nessa dissertação, antes de tentar verificar qual plataforma de virtualização é mais adequada para permitir tanto a economia de recursos físicos quanto a boa implementação de ambientes virtuais de encaminhamento/roteamento no núcleo da Internet, buscamos entender o efeito do aumento da capacidade de hardware sobre o desempenho das máquinas virtuais. A pergunta que esse trabalho tenta responder no primeiro momento é: todas as vezes que mais recursos de memória e CPU são alocados para uma VM o desempenho das máquinas virtuais aumenta? Mostramos que isso não é verdade em máquinas x86 de grande capacidade porque as CPUs são integradas a controladores de memória através da arquitetura NUMA (*Non-Uniform Memory Access*).

Observamos que na arquitetura NUMA, onde núcleos de CPU e pedaços de memória são agrupados para aumentar a capacidade de processamento das máquinas (Fig. 14), o desempenho de uma VM piora quando a quantidade de RAM e CPUs virtuais alocadas nela são maiores que as de um nó NUMA (Fig. 11 e Fig.12). Vimos que isso acontece porque a velocidade de acesso a espaços de memória remotos é muito menor que o acesso à memória local, que pertence ao próprio nó NUMA do processador. Por essa razão, o desempenho das VMs com 30 GB de RAM e 16 CPUs virtuais foi menor que o das VMs com apenas 2 GB de RAM quando aplicações que demandam muito processamento e memória foram executadas, porque os processadores precisaram acessar e alocar memória em nós NUMA remotos. Portanto, se um projetista ou um operador da rede desejar melhorar o desempenho de uma VM não será suficiente aumentar a quantidade de memória e processamento alocados para ela, porque o tamanho dos nós NUMA pode limitar o desempenho da máquina virtual. Ou seja, o melhor desempenho é obtido quando o tamanho dos agrupamentos NUMA é respeitado. Na Seção 3.2.4 nós mostramos como o cálculo de um nó NUMA pode ser realizado.

Desempenho do Xen e do KVM no encaminhamento e no roteamento de pacotes

Esse trabalho mostrou que a eficiência, o isolamento, a extensibilidade e a flexibilidade são os requisitos essenciais de uma plataforma virtual de roteamento, e discorreu sobre diferentes técnicas de virtualização de hardware que podem ser utilizadas em máquinas x86. Entre as diferentes técnicas e plataformas de virtualização decidimos comparar o

desempenho do Xen configurado para utilizar para-virtualização, com o desempenho do KVM, que utiliza o QEMU para fazer emulação de hardware e virtualização completa.

Em todos os testes realizados nessa dissertação, a sobrecarga de memória e CPU do hipervisor do KVM foi menor que a do hipervisor do Xen (Fig. 11, Fig. 12, Fig. 15a e Fig. 15b); e mostramos que esse resultado influencia o desempenho do roteamento de pacotes das máquinas virtuais criadas nessas plataformas de virtualização. Ou seja, mesmo sendo uma ferramenta de virtualização completa foi possível mostrar que o desempenho de roteamento de pacotes do KVM foi melhor que o do Xen para-virtualizado, quando ele foi configurado para utilizar o VIRTIO, que é uma funcionalidade da biblioteca de programação libvirt. Concluímos que isso aconteceu porque, além de fazer o núcleo do Linux agir como um hipervisor, o KVM permite, quando o VIRTIO é habilitado, que os drivers de rede das VMs saibam que estão sendo executados em um ambiente virtual. Dessa forma, eles interagem com o QEMU para realizar as operações de E/S com os privilégios de núcleo (modo de núcleo ou *kernel*) do Linux nativo utilizando seu mecanismo de encaminhamento do *kernel*. Vimos que a arquitetura da plataforma de virtualização Xen (Fig. 10) também contribuiu para que seu desempenho fosse menor que o do KVM (Fig. 20a e Fig. 20b). Combinado com o OpenFlow para operar com separação de planos o desempenho do roteamento de pacotes da VM Xen é melhor que o obtido no Xen roteado. Além disso, mostramos que no primeiro caso os pacotes atravessam o hipervisor apenas uma vez e que no segundo caso o hipervisor é atravessado duas vezes. Por isso, acreditamos que o desempenho pior acontece devido a dupla sobrecarga imposta pelo hipervisor do Xen quando os pacotes roteados pelo domU precisam atravessar o hipervisor mais de uma vez (Fig. 25a).

Quando a VM é utilizada para apenas encaminhar pacotes grandes (de 1500 bytes) em uma topologia ponto-a-ponto, o desempenho do Xen na transmissão de pacotes foi semelhante ao do KVM e um pouco superior na recepção dos dados (Fig. 18a e Fig. 18b). Neste caso, o hipervisor foi atravessado apenas uma vez porque os pacotes foram gerados dentro da VM. No entanto, para rotear pacotes de uma rede para outra uma VM precisa possuir pelo menos duas interfaces de rede, ou seja, não é possível criar roteadores virtuais com apenas uma interface de rede virtual.

Diferentes tipos de topologia de rede do Xen, o uso de Vlans e gateways externos

Este trabalho também mostrou que não é possível segregar o tráfego de pacotes de clientes concorrentes quando o Xen utiliza pontes (*bridges*) para interligar as interfaces virtuais de BE do dom0 com as interfaces de FE das VMs. Por esse motivo, a topologia

híbrida ou a roteada deve ser utilizada quando ambientes de rede virtuais são criados com o Xen. Na topologia híbrida, as pontes continuam sendo utilizadas porque Vlans são criadas nelas, para segregar o tráfego de pacotes de clientes diferentes. Outra questão explicada foi a necessidade de utilização de *gateways* externos quando uma Vlan é criada para as VMs de um ambiente virtual. No Xen, por exemplo, isso é extremamente importante porque, quando um filtro de pacotes é inserido no dom0 para limitar o tráfego entre duas Vlans, ele consome recursos de memória e CPU do domínio privilegiado, que é o gargalo dessa plataforma de virtualização. Mesmo no KVM, o desempenho de um filtro de pacotes executado na memória do Linux nativo é extremamente menor que o das ACLs processadas em um roteador que utiliza TCAMs para tomar as decisões de roteamento. Isso porque as TCAMs permitem que as tomadas de decisão aconteçam em nível de hardware nos roteadores comerciais, que por esse motivo são capazes de processar as ACLs em velocidade de linha. Portanto, esse trabalho mostra que, para garantir o bom desempenho de um ambiente de redes virtuais, *gateways* externos devem ser utilizados quando uma Vlan é criada para segregar o tráfego de pacotes de um cliente/projeto do tráfego de pacotes de outro cliente.

Os problemas de isolamento do Xen

Provamos que o Xen possui problema de isolamento ao fazer o consumo de CPU no dom0 variar significativamente, quando foi gerado tráfego de pacotes de uma VM para outra (Fig. 21). O problema existe porque o hipervisor do Xen não gerencia o acesso dos domUs ao dom0, nem a alocação de recursos de processamento que acontecem dentro desse domínio de drivers compartilhado. Por esse motivo, apenas um domU malicioso pode afetar o desempenho de outros (em operação na mesma máquina física) esgotando os recursos do dom0 (Fig. 21). Ou seja, ele pode gerar operações de E/S suficientes para ocupar todos os recursos de CPU do dom0. Este é um problema grave do Xen. Além disso, a instalação padrão do Xen roteado não oferece recursos para a virtualização das operações de E/S.

No KVM não existe uma VM de drivers compartilhada pelas outras máquinas virtuais quando elas precisam realizar operações de E/S. Cada VM KVM possui um processo QEMU exclusivo que gerencia os acessos aos dispositivos de hardware emulados. Assim, não é necessário atravessar outra máquina virtual quando se deseja realizar E/S de dados. Além disso, o QEMU executado dentro de cada VM permite que os dispositivos emulados interajam com o VIRTIO. Com o VIRTIO habilitado no KVM, os drivers das VMs passam a interagir com as interfaces de rede virtuais que o VIRTIO criou. Ou seja, as operações de E/S são virtualizadas pelo VIRTIO porque o mecanismo de encaminhamento do kernel interage com

as interfaces de rede virtuais como se elas fossem interfaces de rede físicas. Como cada operação de E/S é gerenciada por um processo QEMU exclusivo que só atua quando é escalonado pelo agendador de tarefas do kernel esta arquitetura de virtualização (KVM com VIRTIO) não apesenta problemas de isolamento.

Através das avaliações realizadas neste trabalho, é possível concluir que o KVM apresentou melhor desempenho no roteamento de pacotes porque as operações de E/S que foram virtualizadas pelo VIRTIO e a sobrecarga apresentada pelo hipervisor KVM (*kernel* do Linux agindo como hipervisor) sobre a memória e a CPU foi baixa. Por isso, acreditamos que a plataforma de virtualização de hardware KVM é adequada para a construção de ambientes de roteamento virtual.

Virtualização de redes com o OpenFlow

Vimos que é possível combinar o OpenFlow com plataformas de virtualização que segmentam recursos de hardware, tais como o Xen e o KVM, para que o desempenho de encaminhamento e roteamento das redes virtuais criadas seja aumentado. Quando isso é feito, a rede virtual definida por uma VM passa a ter capacidade semelhante à do comutador OpenFlow escolhido. Na Seção 4.3 mostramos que quando uma VM Xen ou KVM foi utilizada como controlador OpenFlow e o dom0 – ou Linux nativo no caso do KVM, foi configurado como comutador, a capacidade de roteamento de pacotes das redes criadas por essas VMs foi semelhante à do Linux nativo (Fig. 26).

Concluímos que a melhoria ocorrida no Xen deve-se ao fato de que apenas os primeiros pacotes de um fluxo de dados precisam atravessar o hipervisor para entrar no controlador (VM Xen). Assim que isso acontece, a informação necessária para que o roteamento seja realizado é copiada para a tabela de encaminhamento do comutador. A partir daí, todos os pacotes que entram no dom0 são roteados para seus destinos dentro do próprio dom0. Com essa solução a dupla sobrecarga do hipervisor é eliminada. É possível, inclusive, criar controladores virtuais (VMs Xen e KVM) para gerenciar comutadores OpenFlow instanciados em roteadores físicos de grande capacidade. Assim, concluímos que, utilizando máquinas x86 e roteadores comercias de grande capacidade, pode-se promover o avanço do núcleo da Internet através da criação de ambientes virtuais que possuem lógicas de encaminhamento de pacotes personalizadas, para atender as necessidades da aplicação que desejamos executar dentro desse ambiente virtual.

Possíveis trabalhos futuros

Como o Xen não realiza a virtualização de operações de E/S nativamente, é interessante combinar o seu uso com soluções de hardware capazes de alterar o dispositivo físico para que múltiplas interfaces de rede virtuais possam ser diretamente acessadas pelas VMs atribuídas a elas [44]. O SR-IOV (*Single Root I/O Virtualization*) é uma dessas tecnologias, proposta pelo fabricante de hardware Intel System. Dessa forma, pode-se aumentar o desempenho do encaminhamento de pacotes nas diversas redes de pesquisa que hoje operam com o Xen. O FITS, por exemplo, é um *testbed* criado pelos pesquisadores do Grupo de Teleinformática e Automação (GTA) da UFRJ que pode ser beneficiado, pois utiliza o Xen e o OpenFlow para a criação de ambientes virtuais [23]. Além disso, como o *Market Share* da Citrix é maior, de acordo com o quadrante mágico do Gartner, também é necessário avançar nas pesquisas que possibilitam resolver o problema de isolamento do Xen. É possível fazer isso incluindo funcionalidades do dom0 [11] e/ou criando um módulo para ser carregado com o *kernel* do Linux, no modo de núcleo, da mesma maneira que o Xen para-virtualizado carrega o dom0.

Como a virtualização sempre é associada à economia de recursos, diversas soluções comerciais e de código aberto tem sido desenvolvidas para que a infraestrutura de máquinas físicas sejam oferecidas como serviço (*Infrastructure as a Service* – IaaS) de uma nuvem de computadores virtuais (*Cloud*). Também é possível encontrar muitas soluções dispostas a oferecer plataformas de desenvolvimento como serviço na nuvem (*Platform as a Service* – PaaS), para que desenvolvedores não se preocupem em como seus códigos serão implementados nos servidores. Quando soluções de PaaS e IaaS são combinadas em um serviço de *Cloud*, elas permitem que um desenvolvedor se preocupe apenas em criar os códigos necessários para a solução de um problema. Se o PaaS é poliglota, como o Tsuru [41], o código pode inclusive ser construído em qualquer das linguagens de programação suportadas pela ferramenta (python, ruby etc.). Essa característica torna um PaaS extremamente poderoso porque ele é capaz de descobrir em que linguagem de programação o código foi construído, solicitar uma máquina virtual para o IaaS, instalar automaticamente as dependências da linguagem na máquina virtual iniciada e tornar a aplicação desenvolvida disponível para uso.

No entanto, as aplicações criadas pelos IaaS e PaaS das *Clouds* existentes na Internet continuam sujeitas as limitações do IP porque as redes virtuais ainda não são oferecidas como serviço. Por isso é importante que os pesquisadores, que trabalham para resolver os problemas

provocados por causa das limitações do IP, participem do desenvolvimento das soluções de *Cloud*, para que redes virtuais também possam ser oferecidas como serviço (*Networking as a Service – NaaS*) nessas APIs. Assim, além de segmentar o hardware para oferecer servidores virtuais, utilizando o protocolo OpenFlow, também será possível programar o núcleo da rede e permitir que alternativas ao IP sejam testadas em um serviço de Cloud.

REFERÊNCIAS

[1] BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the art of virtualization. In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. EUA, 2003. p. 164-177.

[2] BEHESHTI, N.; UNDERHILL, D.; HELLER, B.; BOLOUKI, S.; MCKEOWN, N.; GANJALI, Y. Experimenting with programmable routers in real networks. In: Demo Session, ACM SIGCOMM. EUA, 2008.

[3] CENTOS.ORG. Community ENTerprise Operating System: Virtualization on Cent OS. Disponível em: <http://wiki.centos.org/HowTos/Virtualization/Introduction>. Acesso em: 05 outubro, 2012.

[4] CISCO. Disponível em: <http://www.cisco.com/en/US/prod/collateral/switches/ps9441/ps9402/ps9512/Data_Sheet_C78-437763.html>. Acesso em: 15 de Janeiro, 2013.

[5] CORREA, C. N. A.; LUCENA, S. C.; ROTHENBERG, C. E.; SALVADOR, M. R. Uma avaliação experimental de soluções de virtualização para o plano de controle de roteamento de redes virtuais. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Campo Grande, Brasil: Anais do XXIX SBRC, 2011.

[6] DAS, S.; PARULKAR, G.; MCKEOWN, N. Unifying packet and circuit switched networks. In: *OPENFLOW-TR, Department of Electrical Engineering*, Stanford University, 2009.

[7] DELL. Especificações do servidor Dell PowerEdge 2950. Disponível em: <http://www.dell.com/downloads/global/products/pedge/en/PE2950_SS_072007.pdf>. Acesso em: 13 de janeiro, 2013.

[8] DELL. Especificações do servidor Dell PowerEdge C6100. Disponível em: <http://www.dell.com/downloads/global/products/pedge/en/poweredge-c6100-spec-sheet-en.pdf>. Acesso em: 13 de janeiro, 2013.

[9] EGI, N.; GREENHALGH, A.; HANDLEY, M.; HOERDT M.; MATHY, L.; SCHOOLEY, T. Evaluating Xen for router virtualization. Proc. of ICCCN 07, 2007.

p. 1256-1261.

[10] EGI, N.; GREENHALGH, A.; HANDLEY, M.; HOERDT, M.; HUICI, F.; MATHY, L. Towards high performance virtual routers on commodity hardware. In: *Proceedings of the ACM International Conference on Emerging Networking Experiments and Technologies* (CoNEXT), Madrid, Espanha, 2008.

[11] FERNANDES, N. C. Técnicas de virtualização e autoconfiguração para o projeto de redes de nova geração. Tese (Doutorado). UFRJ/COPPE, Rio de Janeiro, Brasil, 2011.

[12] FERNANDES, N. C.; MOREIRA, M. D. D.; MORAES, I. M.; FERRAZ, L. H. G.; COUTO, R. S.; CARVALHO, H. E. T.; CAMPISTA, M. E. M.; COSTA, L. H. M. K.; DUARTE, O. C. M. B. Virtual networks: isolation, performance, and trends. In: *Annals of Telecommunications*, Springer Paris: ISSN 0003-4347, DOI 10.1007/s12243-010-0208-9. 2010. p. 1-17.

[13] FERNANDES, N. C.; DUARTE, O. C. M. B. XNetMon: A network monitor for securing virtual networks. In: *Proceedings of the IEEE International Conference on Communications ICC 2011*. Japão, 2011.

[14] FITS. Future Internet Testbed with Security. Disponível em: <http://www.gta.ufrj.br/fits/>. Acesso em 15 de maio, 2013.

[15] GARTNER. Magic Quadrant for x86 server virtualization infrastructure. Disponível em:<http://www.gartner.com/technology/reprints.do?id=1-1AVRXJO&ct=120612&st=sb>. Acesso em 10 janeiro, 2013.

[16] GOUGH, C. *CCNP Routing Exam Certification Guide*. San Jose: Cisco Press, 2001.

[17] GUDE, N.; KOPONEN, T.; PETTIT, J.; PFAFF, B. NOX: Towards an Operating System for networks. In: *ACM SIGCOMM Computer Communication. Review, Vol. 38, Issue 3*, 2008.

[18] IBM. Arquitetura e vantagens da KVM: a máquina virtual do kernel Linux. Disponível em: <http://www.ibm.com/developerworks/br/library/l-linux-kvm/#resources>. Acesso em 15 de Maio, 2012.

[19] JVM. Disponível em: <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136373.html>. Acesso em 10 de junho, 2013.

[20] KUROSE, J. F.; ROSS, K. W. *Redes de Computadores e a Internet*. Pearson, 2010.

[21] KVM.ORG. Preparing to use KVM. Disponível em: <http://www.linux-kvm.org/page/FAQ>. Acesso em 12 de novembro, 2012.

[22] KVM. Using virtio_net for the guest NIC: throughput tests using Iperf. Disponível em: <http://www.linux-kvm.org/page/Using_VirtIO_NIC>. Acesso em 14 maio, 2013.

[23] MATTOS, D. M. F. Uma arquitetura de virtualização de redes orientada à migração com qualidade de serviço. Dissertação (Mestrado). UFRJ/COPPE, Rio de Janeiro, Brasil, 2012.

[24] MATTOS, D. M. F.; FERRAZ, L. H. G.; COSTA, L. H. M. K.; Duarte, O. C. M. B. Evaluating virtual router performance for a pluralist future Internet. In: *3th International Conference on Information and Communication Systems (ICICS 2012)*, Irbid, Jordânia, Abril, 2012.

[25] MCCALPIN, J. D. Memory bandwidth and machine balance in current high performance computers. In: *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, Dezembro, 1995.

[26] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: enabling innovation in campus networks. In: *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, 2008. p. 69-74.

[27] MOREIRA, M. D. D.; FERNANDES, N. C.; COSTA, L. H. M. K.; DUARTE, O. C. M. B. Internet do futuro: um novo horizonte. In: *Minicursos do Simpósio Brasileiro de Redes de Computadores*. Pernambuco, Brasil, 2009. p. 1-59.

[28] NIC.BR. Disponível em:<http://ipv6.br/estatisticas/>. Acesso em 17 dezembro, 2012.

[29] OPENMP. Apostila de treinamento: introdução ao OpenMP. UNICAMP/ CENPAD-SP, São Paulo, Brasil, 2010.

[30] OTT, D. Optimizing applications for NUMA. Intel, 2011.

[31] PETERSON, L., SHENKER, S; TURNER, J. S. Overcoming the Internet impasse through virtualization. In: *Proc. ACM Workshop on Hot Topics in Networks*

(HotNets). EUA, 2004.

[32] QEMU. A generic and open source machine emulator and virtualizer used as a machine emulator. Disponível em: <http://wiki.qemu.org/>. Acesso em 10 de junho, 2013.

[33] REDHAT. Histórico da Red Hat. Disponível em: <http://br.redhat.com/about/company/history.html>. Acesso em 16 de dezembro, 2012.

[34] REXFORD, J.; DOVROLIS, C. Future Internet Architecture: Clean slate versus evolutionary research. In: *Communications of the ACM*, vol. 53, no. 9, 2010. p. 36-40.

[35] RODRIGUES, H.; SOARES, P. V.; SANTOS, J. R.; TURNER, Y.; GUEDES, D. Isolamento de tráfego em ambientes virtualizados. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Campo Grande, Brasil: Anais do XXIX SBRC, 2011.

[36] SHERWOOD, R.; GIBB, G.; YAP, K.; APPENZELLER, G.; CASADO, M.; MCKEOWN, N. PARULKAR, G. FlowVisor: a network virtualization layer. In: *OPENFLOW-TR, Deutsche Telekom Inc. R&D Lab, Stanford University. Nicira Networks*, 2009.

[37] STREAM. Notes on the mystery of hardware cache performance counters. Disponível em: <http://blogs.utexas.edu/jdm4372/>. Acesso em 25 de janeiro, 2013.

[38] SUPER PI. A single threaded benchmark that calculates pi to a specific number of digits. Disponível em: <http://www.superpi.net/>. Acesso em 20 de janeiro, 2013.

[39] TANENBAUM, A. S. *Modern Operating Systems* (3rd Edition). Prentice Hall, 2007.

[40] TANENBAUM, A. S. *Computer Networks*. Prentice Hall, 2007.

[41] TSURU. An open source polyglot cloud application platform. Disponível em: <http://www.tsuru.io/>. Acesso em 2 de junho, 2013.

[42] TURNER, J. S.; TAYLOR, D. E. Diversifying the Internet. In: *Proc. IEEE* GLOBECOM, 2005.

[43] WOTTRICH, R.; GENEZ, T.; Pereira, W. Uma visão geral sobre sistemas virtualizados. Minicurso de arquitetura de computadores. UNICAMP, São Paulo,

Brasil, 2012. Disponível em: <http://www.ic.unicamp.br/~ducatte/mo401/1s2012/T2/G04-115168-ap.pdf>. Acesso em 10 de junho, 2013.

[44] HUAN,G Z. I/O Virtualization performance. In: *Xen Summit at Intel*. Shanghai, novembro 19-20, 2009.