UMA APLICAÇÃO DE REDES DEFINIDAS POR SOFTWARE PARA A GERÊNCIA DE REDES DE DATACENTERS VIRTUALIZADOS

ROGÉRIO VINHAL NUNES

UMA APLICAÇÃO DE REDES DEFINIDAS POR SOFTWARE PARA A GERÊNCIA DE REDES DE DATACENTERS VIRTUALIZADOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

ORIENTADOR: DORGIVAL OLAVO GUEDES NETO

Belo Horizonte

Julho de 2012

Nunes, Rogério Vinhal
N972a Uma aplicação de Redes Definidas por Software para a gerência de redes de datacenters virtualizados / Rogério Vinhal Nunes. — Belo Horizonte, 2012
xvi, 96 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Dorgival Olavo Guedes Neto

1. Computação – Teses. 2. Redes de Computadores – Teses. 3. Computação em Nuvem – Teses.
I. Orientador. II. Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Uma aplicação de redes definidas por software
para a gerência de redes de datacenters virtualizados

ROGÉRIO VINHAL NUNES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. DORGIVAL OLAVO GUEDES NETO - Orientador
Departamento de Ciência da Computação - UFMG

PROF. LUIZ FILIPE MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

PROF. MARCOS AUGUSTO MENEZES VIEIRA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 31 de julho de 2012.

Dedico este trabalho aos meus pais, que não só deram apoio incondicional em todos os dias da minha vida como investiram sempre na minha educação, o que tornou tudo isso possível.

Agradecimentos

Agradeço primeiramente à minha família, que não só concedeu todo o apoio como criou as condições para que tudo isso acontecesse.

Aos amigos e demais partcipantes da minha vida que não só ajudaram com companheirismo nos curtos tempos livres durante os últimos anos como também criaram uma estrutura sólida para que fosse possível suportar as adversidades encontradas durante todo o processo (que não foram poucas).

Em seguida, mas não menos importante, ao Dorgival, não só pela orientação acadêmica convencional como também no apoio emocional, que foi imprescindível para manter o foco e gerar um trabalho de qualidade.

Também agradeço a todos os colegas de laboratório e do curso, que sempre ajudam com experiências tanto iguais como diferentes, lembrando que todos caminhamos por percursos parecidos, mas de maneira diferente. Sem qualquer um de vocês esse trabalho não se tornaria tudo o que ele se tornou e não teria a mesma importância.

"Computer Science is no more about computers
than astronomy is about telescopes."
(Edsger W. Dijkstra)

Resumo

Serviços de computação em nuvem relacionados à infraestrutura vêm se tornando cada vez mais importantes. Nessa linha de negócio, a virtualização garante uma forma de oferecer flexibilidade, isolamento e funcionalidades extras. Isso abriu espaço para o modelo de computação em nuvem para *datacenters* multi-inquilinos, onde o cliente (inquilino) não só poderia alugar uma máquina virtual como uma rede local inteira de máquinas virtuais dentro da infraestrutura do provedor.

Apesar das soluções de virtualização oferecerem boas soluções para o isolamento de recursos como CPU e memória, esse modelo de negócio requer soluções para o isolamento do tráfego de rede de cada cliente que são oferecidas de formas limitadas, caras e pouco escaláveis na atualidade, como o uso de *VLANs*. Paralelamente, a comunidade acadêmica recentemente tem focado bastante no conceito de Redes Definidas por Software (*SDNs*) por conta do padrão *OpenFlow*. Essa nova abordagem permite organizar a rede de uma forma bem mais flexível, com uma visão global centralizada da rede.

Este projeto apresenta o *DCPortals*, que permite implementar uma visão de rede virtual para cada cliente de um serviço de computação em nuvem multi-inquilino sem a utilização de hardware especializado. Ele usa o sistema operacional para redes *POX* em conjunto com um gerenciador de computação em nuvem *OpenStack* que controla máquinas com monitores de máquinas virtuais *Xen*. Essas máquinas são conectadas por *switches* virtuais *Open vSwitch* que utilizam o protocolo *OpenFlow*. Dessa forma, cada cliente poderá não só alugar suas máquinas virtuais como também interligá-las em uma rede virtual privativa isolada das demais independente de como as máquinas hospedeiras estão organizadas na infraestrutura física subjacente.

Os testes realizados comprovaram o isolamento inclusive em casos de ataques e mostram que o *overhead* se limita ao primeiro pacote dos fluxos de pacotes entre máquinas virtuais.

Palavras-chave: Datacenter, Computação em Nuvem, Virtualização, Redes Definidas por Software.

Abstract

Cloud-Computing services related to infrastructure have been drawing much attention recently. On this business model, virtualization is an efficient way to offer flexibility, isolation and several new functionalities. This technology has led to the multi-tenant datacenter model for Cloud-Computing, where the client (tenant) can rent not only a virtual machine, but a network of virtual machines in the provider infrastructure.

Despite of the recent virtualization products already offering isolation solutions for resources such as CPU and memory, this new business model require network isolation solutions for each tenant that, as of today, are offered in limited, expensive and unscalable ways like the use of VLANs. Parallely, the academic community have recently focused on the development of Software-Defined Networks (SDNs) due to the definition of the OpenFlow standard. This new approach allows the network to be organized in a more flexible way, with a global centralized vision of the network.

This project presents *DCPortals*, which implements a virtual network view for each client of a multi-tenant Cloud-Computing service without the need to use specialized hardware. It is implemented using the POX Network Operating System along with the OpenStack Cloud-Computing manager controlling machines with the Xen Virtual Machine Monitor. These machines are connected by Open vSwitch virtual switches that are capable of using the OpenFlow protocol. Therefore, each client can not only rent virtual machines, but also connect them in a private virtual network isolated from every other machine independently of how the machines are organized in the underlying physical infrastructure.

The experiments confirmed the network isolation even in attack cases and showed that the observed overhead is limited to the first packet of a flow between virtual machines.

Keywords: Datacenter, Cloud-Computing, Virtualization, Software-Defined Networks.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

No contexto de Computação em Nuvem, o modelo de Infraestrutura como Serviço (*Infrastructure-as-a-Service*, ou *IaaS*) é um modelo de negócio que tem despertado grande interesse. Nesse modelo, empresas podem alugar máquinas em um datacenter de grandes dimensões construído para esse fim e evitar os problemas de aquisição e instalação de hardware próprio. Questões de gerência, energia, refrigeração, conectividade, flexibilidade de configuração e elasticidade (alteração do número de máquinas em função de variações da demanda) são facilmente resolvidas pelo provedor de *IaaS*, simplificando a operação do serviço da empresa cliente. Por isso ser uma necessidade em crescimento no mercado, esse tipo de negócio tem se popularizado com uma velocidade muito grande.

Essa alta demanda obrigou as empresas que oferecem o serviço de *IaaS* a aumentar a facilidade e a flexibilidade no gerenciamento de recursos desse ambiente para tornar possível atender ao mercado. Isso abriu as portas para o uso da virtualização em *datacenters* que oferecem serviços de computação em nuvem. A virtualização de máquinas proporciona não só um aumento na facilidade de gerência do aluguel das mesmas como também elasticidade para que o usuário possa requisitar e pagar por uma máquina virtual exatamente do tamanho de suas necessidades. Além disso, também aumenta a eficiência do uso de recursos do *datacenter* por explorar melhor os recursos ociosos dos servidores físicos. Hoje existem já diversos serviços de aluguel de máquinas virtuais de diversos tipos em diversos ambientes independente de qual máquina real hospeda a máquina virtual, como por exemplo o *UOL Cloud* (UOL, 2012) e o *Amazon EC2* (Amazon, 2010a).

No ambiente de computação em nuvem com virtualização as empresas locadoras possuem um *datacenter* de máquinas hospedeiras que, por sua vez, podem cada uma conter dezenas de máquinas virtuais de diferentes clientes. O gerenciamento da locali-

zação, disseminação e organização dessas máquinas isoladamente por si só representa um problema complexo do ponto de vista computacional (Nurmi et al., 2008; Wood et al., 2007). Além disso, há o desafio de garantir o isolamento entre as máquinas de um cliente para que ele não tenha os dados de suas máquinas virtuais manipulados por outros clientes que compartilham a mesma infraestrutura física.

Essa demanda por isolamento surge por conta do aumento de clientes que não só necessitam alugar uma máquina para seus serviços, mas diversas máquinas que precisam se comunicar entre si. Entretanto, essa comunicação tem que ser feita sobre a rede compartilhada do *datacenter* e o isolamento não é garantido. Há casos em que sistemas armazenados na nuvem tiveram seus dados ou serviços comprometidos devido à ação maliciosa de terceiros que têm acesso à mesma infraestrutura física, como por exemplo o relato BitBucket Attack (2012). Entretanto, alugar diretamente uma rede local separada para cada cliente se torna caro e inviável devido à estrutura do *datacenter*, o custo de separação física e o número de clientes que têm essa demanda. Esse problema é teoricamente resolvido através do uso de *VLANs*[1]. O administrador de rede poderia configurar os *switches* da infraestrutura para determinar quais máquinas que ligadas a eles pertencem a qual sub-rede. Entretanto, essa solução tem problemas de gerência e escalabilidade que puderam ser constatados em contato com empresas que oferecem esse tipo de serviço[2]. Não só é necessário uma configuração dispendiosa dos *switches*, que precisam suportar o identificador destinado a *VLANs*, como também existe uma limitação prática no tamanho da parte do cabeçalho, como definido pelo padrão IEEE802.1Q (2005), que determina um campo de 12 bits para o identificador da VLAN, proporcionando no máximo 4.094 *VLANs*, retirado-se os endereços reservados, em um *datacenter*. Além disso, pelo menos uma empresa brasileira da área de *IaaS* reportou que os *switches* de sua rede só reconhecem cerca de mil *VLANs*, por limitações do hardware. Esses problemas são de grande importância, uma vez que impedem que o sistema seja completamente automatizado e transparente, assim como também limita o número de sub-redes para um número pequeno para atender a demanda das empresas que oferecem esse tipo de serviço.

Pensando em resolver esse tipo de problema com uma abordagem mais elegante e econômica, este projeto apresenta o *DCPortals*, uma ferramenta de gerenciamento e isolamento de redes em *datacenters* virtualizados. O sistema utiliza o princípio de Redes Definidas por Software (*Software-Defined Networks*, ou *SDNs*), uma nova tecnologia

[1] VLANs são separações lógicas de rede controladas pelos *switches* para permitir isolamento entre sub-redes ao se configurar os *switches* da infraestrutura física. Com isso, apesar de dividirem a mesma infraestrutura física, as máquinas são logicamente separadas pelos *switches* em redes diferentes.

[2] Comunicação pessoal.

que vem ganhando atenção na área de Redes de Computadores. No paradigma de *SDN*, os elementos de comutação de rede oferecem uma interface de programação que permite ao administrador criar uma visão abstrata de todos os fluxos na rede e determinar regras para o tratamento e encaminhamento de cada fluxo como uma aplicação (software) que controla toda a rede.

Nas próximas seções serão descritos mais detalhes sobre computação em nuvem, virtualização e *SDNs* e a contribuição do trabalho.

1.1 Computação em nuvem

O termo "computação em nuvem" é utilizado com bastante frequência nos dias de hoje, porém, costumeiramente se referindo a coisas diferentes. O termo *"cloud"*, ou "nuvem", se referindo a conceitos da área de informática começou a ser usado na década de 60 através de um artigo pioneiro (Parkhill, 1966) que previa que a computação seria usada como um serviço.

Desde então o termo foi utilizado para diversos contextos, principalmente na década de 90. Entretanto, foi depois que o *CEO* do Google, Eric Schmit, começou a utilizá-lo para um modelo de negócio de prover serviços através da Internet que o termo se popularizou. Em seguida, o termo foi utilizado inúmeras vezes para diversos significados principalmente com o objetivo de marketing.

Justamente por essa diversidade, trabalhos começaram a buscar um padrão para o termo. Vaquero et al. (2009) discorrem sobre 20 diferentes significados retirados de diversas fontes a fim de selecionar um padrão. Nos EUA, o NIST (*National Institute of Standards and Technology*) definiu *Cloud Computing* como um modelo para possibilitar acesso remoto conveniente sob demanda a um conjunto compartilhado de recursos computacionais (redes, servidores, armazenamento, serviços) que podem ser rapidamente alocados e liberados com mínimo esforço ou interação com o provedor (Mell & Grance, 2011).

Zhang et al. (2010) apresenta três camadas de serviço nas quais é possível realizar este serviço sob demanda: a camada de *aplicação*, onde os serviços são providos para o usuário final através de programas que ele executa pela Internet, sem se preocupar com sua real localização, como aplicativos Web do *Google*, *FaceBook*, *Youtube*; a camada de *plataforma*, que possibilita ao usuário expressar computação a ser realizada remotamente através de um *framework* que limita o serviço prestado a uma interface de programação bem definida, como serviços do *Microsoft Azure* (Microsoft, 2012), *Google AppEngine* (Google, 2010) e *Amazon SimpleDB* (Amazon, 2010b); e a camada de

infraestrutura, que disponibiliza servidores completos, virtualizados ou reais, para que o usuário contratante os configure como desejar como, por exemplo, o serviço *Amazon EC2* (Amazon, 2010a).

Neste trabalho abordamos o problema de configuração dos recursos físicos disponíveis, portanto essa solução mapeia mais claramente no contexto da camada de infraestrutura. Entretanto, os recursos providos podem também ser úteis no contexto da camada de plataforma.

1.2 Virtualização

No contexto de computação em nuvem, virtualização identifica o conjunto de técnicas de hardware e software que permite utilizar uma única máquina física para oferecer aos usuários a visão de que existem diversas máquinas (virtuais) disponíveis, potencialmente com recursos diferentes. Dessa forma, *CPU*, memória e discos da máquina física podem ser divididos entre diversas máquinas virtuais, que executam de forma independente, isoladas umas das outras. Para gerenciar essa alocação de recursos é criado um monitor de máquinas virtuais (Virtual Machine Monitor, ou VMM) que intermedeia a comunicação entre o sistema operacional e os recursos hardware podendo assim distribuir os recursos físicos entre diversas máquinas virtuais controladas por esse monitor (Barham et al., 2003).

A utilização da virtualização de serviços da camada de infraestrutura, em especial, vem crescendo bastante nos últimos anos por apresentar diversas vantagens:

- **Melhor utilização de recursos.** É raro um sistema de computação utilizar completamente todos os seus recursos de hardware. Normalmente há uma oscilação (Veloso et al., 2002; Shi et al., 2002; Wang et al., 2003; Chesire et al., 2001; Arlitt & Jin, 1999), portanto a utilização simultânea de mais de uma máquina virtual em uma máquina real pode ajudar a diminuir o desperdício de recursos, principalmente através do aluguel dos demais como forma de aumentar a receita.

- **Isolamento.** Os recursos de uma máquina virtual são isolados das demais, garantindo assim a possibilidade de diversos usuários utilizarem uma gama de recursos sem a interferência ou visibilidade de algum outro usuário malicioso que tem acesso ao servidor real através de outra máquina virtual, mesmo ambas as máquinas estando localizadas na mesma máquina física

- **Facilidade de gerenciamento.** Máquinas virtuais podem ser criadas, destruídas, ter seu estado copiado, ter sua capacidade estendida ou reduzida, além de

poderem ser migradas de hospedeiro (Clark et al., 2005). Essas operações eliminam qualquer necessidade de interferência física para reorganização das máquinas em qualquer demanda de modificação estrutural, tudo pode ser feito diretamente da interface de gerenciamento do monitor de máquinas virtuais.

O monitor de máquinas virtuais tem o papel de distribuir os recursos físicos da máquina real entre as máquinas virtuais. Isso pode ser feito de diversas formas; o monitor *Xen*, por exemplo, cria a abstração de um anel assíncrono de operações de entrada e saída utilizando um algoritmo de produtor/consumidor. São produzidos pedidos de acesso ao dispositivo pelo sistema operacional da máquina virtual que são consumidos pelo monitor do *Xen*. Para cada pedido de acesso o *Xen* produz uma resposta, que por sua vez é consumida pelo sistema operacional da máquina virtual. Esse mecanismo permite que diversas máquinas virtuais façam requisições aos recursos compartilhados simultaneamente enquanto que somente o monitor acessa o dispositivo físico de fato. Com isso, é garantido que apesar das requisições serem assíncronas e concorrentes, o acesso ao dispositivo é sequencial e ordenado pelo monitor.

Em relação à conexão de rede dessas máquinas, o compartilhamento ocorre com a criação de um novo nível de rede interno à máquina. É criado um *switch* virtual dentro da máquina hospedeira, ligado tanto na interface de rede de saída da máquina quanto em interfaces de rede virtuais criadas para cada máquina virtual. Dessa forma, é como se existisse uma nova camada de rede dentro da máquina física que conecta todas as máquinas virtuais com a rede externa à máquina física. Dependendo da configuração do monitor de máquinas virtuais, essa conexão pode emular um *switch ethernet* simples, incluir regras de tradução de endereço de rede (*Network Address Translation*, ou *NAT*[3]), ou implementar um roteador, criando uma sub-rede independente dentro do monitor de máquinas virtuais.

Recursos que não possuem o problema de acesso concorrente, como espaço em disco e memória, são mais fáceis de gerenciar. É simplesmente feito um particionamento desses recursos para cada máquina virtual que utiliza sua partição de forma isolada das demais. O isolamento do uso de *CPU* é obtido por meio de um escalonador de máquinas virtuais. O isolamento de operações de entrada e saída (disco e rede) também usa escalonadores para esses recursos, mas normalmente é menos preciso que os demais, por depender de elementos externos.

[3] *Network Address Translation*, ou tradução do endereço de rede, consiste no processo de traduzir diversos endereços de rede através de um roteador para que um único endereço IP externo seja compartilhado por diversas máquinas com endereços IP locais.

1.3 Redes definidas por software

O conceito de Redes Definidas por Software nasceu com a criação do protocolo *OpenFlow* (McKeown et al., 2008) para a programação de *switches*. Esse protocolo define formas para que *switches* que o suportem sejam programados por um software controlador externo, recebendo comandos para cada tipo de fluxo de pacotes observados e armazenando essas regras em suas tabelas internas.

De uma forma geral, todos *switches*, utilizam uma tabela interna para encaminhar os pacotes. O *switch OpenFlow* fornece uma interface de comunicação para a programação externa dessa tabela por um controlador que se comunique com ele utilizando um protocolo especialmente criado para isso. Dessa forma, o *switch* fornece informações, faz consultas e recebe comandos do controlador para manipular sua tabela de encaminhamento. O controlador, por sua vez, utiliza as informações recebidas de todos os *switches* para criar uma visão global da rede. Essa visão pode ser usada para definir um tratamento a ser instalado na tabela do *switch* para que todos os pacotes do mesmo fluxo que casem com essa entrada recebam o mesmo tratamento, sem a necessidade de outra consulta. Esse protocolo pode tanto ser implementado em *switches* físicos quanto em *switches* virtuais utilizados por um monitor de máquinas virtuais.

De forma abstrata, é possível dizer que uma rede definida por software (*SDN*) é caracterizada pela existência de um sistema de controle (software) que pode controlar o mecanismo de encaminhamento dos elementos de comutação da rede (*switches*) por uma interface de programação bem definida. De forma mais específica, os elementos de comutação exportam uma interface de programação que permite ao software inspecionar, definir e alterar entradas da tabela de roteamento do comutador, como ocorre, por exemplo, com comutadores *OpenFlow*. O software envolvido, apesar de potencialmente poder ser uma aplicação monolítica especialmente desenvolvida, na prática tende a ser organizado com base em um controlador de aplicação geral, em torno do qual se constroem aplicações específicas para o fim de cada rede. É possível, ainda, utilizar-se um divisor de visões para permitir que as aplicações sejam divididas entre diferentes controladores.

O conceito de *SDNs* é recente, mas isso não impede que tenha conseguido a atenção da academia e do mercado. Diversos trabalhos (Seetharaman, 2012; Vaughan-Nichols, 2011) já consideram a utilização de *SDNs* como uma forma bastante interessante e inovadora para tratar as redes e podem revolucionar a forma como *datacenters* funcionam hoje em dia. Há, sem dúvida, bastante espaço para crescimento nessa área de pesquisa. A influência dessa tendência é ilustrada pela convenção *Open Networking Summit* (ONSummit, 2012), que foi realizada em abril de 2012 em Santa Clara, Cali-

fórnia, cujo tópico principal foi discutir a influência de *SDNs* no estado atual das redes de computadores e apresentar ideias para o futuro.

A visão da rede como permitida pelas *SDNs* foi o primeiro passo para a criação do conceito que está sendo usado recentemente de Sistemas Operacionais para Redes. A criação de uma interface de programação que oferece uma forma de se controlar o encaminhamento dos pacotes em cada *switch* permitiu a criação da ideia de *Network Hypervisor* (Casado et al., 2010) assim como diversas aplicações, como a criação de um *switch* virtual distribuído, aliviamento na utilização de roteadores e redes multi-inquilinos (ou *multi-tenant network*, uma rede de computação em nuvem com diversos clientes). Esse *Network Hypervisor* é responsável pela programação dos *switches* através do protocolo *OpenFlow* e das informações globais da rede, criando assim uma separação entre as redes físicas e lógicas e definindo o conceito de *SDNs* ou redes definidas por software.

1.4 Contribuição do trabalho e organização do texto

Este trabalho aborda o problema de isolamento de redes de clientes em um *datacenter* de computação em nuvem virtualizado com uma infraestrutura física compartilhada entre os clientes através de uma abstração de redes lógicas privativas. Para isso, ele propõe o sistema *DCPortals*, que utiliza diversas ferramentas de código aberto e as integra com o conceito de Redes Definidas por Software, criando uma solução de gerenciamento de *datacenters* virtualizados com a funcionalidade de isolamento de redes para sistemas multi-inquilinos de computação em nuvem.

Além da contribuição de um sistema inovador para resolver um problema comum em provedoras desse serviço na atualidade, o projeto contribui com as ferramentas utilizadas encontrando deficiências e apresentando soluções decorrentes da integração realizada. Foram listados diversos *bugs* e sugeridas correções que foram na maior parte aceitas pelos desenvolvedores e serão implementadas em versões futuras.

Durante a definição do trabalho foi mantido contato com a empresa UOL, detentora do produto *UOL Cloud* (UOL, 2012). Esse contato ajudou a determinar a relevância do trabalho assim como algumas alterações para que ele tornasse mais relevante na visão atual do mercado através dos problemas que uma solução mercadológica já existente possui. A conclusão desse contato foi que o trabalho não só é relevante como está nos planos para o futuro da UOL uma continuação e adaptação para seu produto corrente.

O restante do texto está organizado da seguinte forma:

- **Capítulo 2 - Conceitos relacionados:** este trabalho se baseia nos conceitos de Redes Definidas por Software em ambientes virtualizados. Para descrever a solução nesse contexto, o capítulo 2 descreve em mais detalhes a estrutura atual dos *datacenters* virtualizados existentes e o protocolo *OpenFlow* e então constrói uma definição mais completa do que se entende por Redes Definidas por Software e seu elemento principal, o controlador ou *Network Hypervisor*. Naquele capítulo são discutidos também os principais trabalhos relacionados.

- **Capítulo 3 - Desenvolvimento:** uma vez definidos os conceitos, é possível descrever o papel de cada ferramenta tal como a decisão por trás da escolha da mesma para o uso no projeto. O capítulo 3 descreve as ferramentas utilizadas, como elas se encaixam, os problemas que foram resolvidos durante a implementação, decisões de implementação e arquitetura do sistema. Ao seu final é apresentado um pseudocódigo com o funcionamento simplificado do módulo implementado e um exemplo do que acontece passo a passo na comunicação entre duas máquinas virtuais de uma mesma rede lógica privativa.

- **Capítulo 4 - Avaliação:** com o sistema implementado e devidamente funcional, é necessário realizar experimentos para comprovar o funcionamento e quantificar a influência das decisões tomadas. O capítulo 4 primeiramente descreve um teste de funcionalidade exemplificando o comportamento com e sem o *DCPortals* em um ambiente de testes previamente definido. Em seguida, o capítulo mostra o impacto do *DCPortals* sobre a banda e a latência observados ao longo do tempo. Também é apresentada uma situação em que um sistema sem isolamento compromete o serviço de um cliente por um ataque de negação de serviço enquanto que o *DCPortals* consegue manter uma qualidade de serviço próxima da ideal. Por fim, serão discutidos os custos em latência inicial que a utilização de um sistema de Redes Definidas por Software acarretam.

- **Capítulo 5 - Conclusões e trabalhos futuros:** finalizando este texto, o capítulo 5 revê os pontos principais do trabalho, suas contribuições e resultados. Também apresenta uma análise criteriosa do futuro do *DCPortals*, sempre levando em consideração como melhorar os pontos fracos assim como estender suas funcionalidades.

Capítulo 2

Conceitos Relacionados

Para simplificar a descrição da solução proposta e implementada, é importante entender primeiro como operam os *datacenters* virtualizados atuais, com suas limitações, e como a tecnologia *OpenFlow* permite a criação de Redes Definidas por Sofware e seu impacto sobre a área de Redes de Computadores. Uma vez apresentados esses conceitos, este capítulo conclui com a discussão de trabalhos relacionados.

2.1 Ambientes de computação em nuvem atuais

Sistemas de computação em nuvem em *datacenters* virtualizados convencionais utilizam uma estrutura dependente de um gerenciador de nuvem e um monitor de máquinas virtuais. Com essa estrutura, é possível gerenciar um número grande de máquinas virtuais distribuídas em diversas máquinas físicas e oferecer o serviço para o cliente final.

Existem diversas ferramentas que entregam essas funcionalidades. Para este projeto foram escolhidas duas bastante utilizadas na comunidade, o *OpenStack* e o *Xen*. Adicionalmente a esse ambiente convencional, foi escolhido um *switch* virtual com suporte ao protocolo *OpenFlow*, o *Open vSwitch*, para permitir posteriormente a sua programação e a implementação do sistema no conceito de redes definidas por software.

A seguir são descritos os funcionamentos dessas ferramentas, razões para suas escolhas e a maneira como elas se relacionam para montar o ambiente propício à implementação do *DCPortals*.

2.1.1 Xen

Como monitor de máquinas virtuais foi escolhido o *Xen* (Barham et al., 2003), por ser um monitor de máquinas virtuais bastante sólido com desempenho satisfatório. *Xen* permite a divisão de recursos físicos entre máquinas *Linux*, *BSD* e *Windows*. Ele suporta tanto o conceito de para-virtualização quanto o de virtualização completa. A para-virtualização consiste na inclusão de informação no sistema virtualizado de que ele é virtualizado, enquanto que a virtualização completa é completamente independente, porém, precisa de suporte de hardware.

Na época em que foi apresentado, em 2003, o *Xen* pretendia suportar a presença de até 100 máquinas virtuais em um mesmo servidor moderno, portanto, ele possui um mecanismo de controle de recursos muito eficiente. Até hoje ele é bastante usado sendo também suportado por uma das distribuições *Linux* mais populares, o *Ubuntu*. O suporte oficial do *OpenStack* ao *Xen* também garante que o sistema não se tornará obsoleto com a modificação das ferramentas.

Uma alternativa que recentemente está se popularizando é o *KVM*, que é um monitor de máquinas virtuais baseado em uma camada de virtualização presente no *Kernel* do *Linux*. Ele apresenta uma série de extensões para virtualização completa através do hardware e para-virtualização através de *drivers* modificados para o sistema virtualizado. Apesar do *Xen* ter sido escolhido como o monitor de máquinas virtuais para o ambiente do projeto, todo o sistema foi montado de forma que uma futura transição para o *KVM* não acarrete grandes alterações, uma vez que todas as ferramentas escolhidas possuem suporte a ambos.

2.1.2 Open vSwitch

A implementação de referência do modelo *OpenFlow* é um comutador em software, executando no espaço de usuário em uma máquina *Linux*. Esse modelo tem sido utilizado como base em diversos experimentos, mas carece de um melhor desempenho. Desenvolvido para suprir essa lacuna, o *Open vSwitch* (Pfaff et al., 2009) é um *switch* virtual que segue a arquitetura *OpenFlow*, implementado em software, com o plano de dados dentro do *kernel* do sistema operacional *Linux*, enquanto o plano de controle é acessado a partir do espaço do usuário. Em particular, essa implementação foi desenvolvida especificamente para controlar o tráfego de rede da camada da virtualização em ambientes virtualizados.

Além de oferecer as funcionalidades da arquitetura *OpenFlow*, discutida na seção 2.2, o *Open vSwitch*, na sua configuração padrão, opera como um *switch ethernet* normal e é nessa condição que é atualmente utilizado. Para simplificar sua integração

com o *kernel* do *Linux*, ele emula as interfaces de rede daquele sistema e utiliza partes do código fonte de seu próprio módulo *bridge*, que implementa o *switch* de rede padrão do *Linux*. Como resultado dessas decisões de projeto, o *Open vSwitch* pode ser utilizado como um substituto imediato para os *switches* virtuais adotados pelos monitores de máquinas virtuais baseados em *Linux* mais utilizados atualmente, como *Xen*, *XenServer* e *KVM*, e vem sendo adotado como o *switch* padrão em diversas distribuições, mesmo quando as funcionalidades da arquitetura *OpenFlow* não são utilizadas. Ele também está sendo incluído no repositório padrão de *softwares* de uma das maiores distribuições de *Linux* do mundo, o *Ubuntu*. Portanto, tem uma chance muito boa de continuar sendo utilizado pela comunidade.

2.1.3 OpenStack

Uma vez montada uma infraestrutura de um *datacenter* virtualizado para serviços de computação em nuvem é necessário realizar uma sequência de operações como distribuir recursos, controlar acesso, configurar as máquinas virtuais, reunir informações sobre o sistema, dentre várias outras atividades. Esse tipo de gerência é complexa e demanda muito tempo quando é feita de forma manual, muitas vezes se tornando impraticável devido ao tamanho dos *datacenters* atuais e a necessidade por agilidade no processo.

O *OpenStack* (OpenStack, 2012) é uma ferramenta de gerenciamento de ambientes virtualizados para computação em nuvem que visa a da simplificação das operações de administração e organização da nuvem além de reunir informações sobre as mesmas em um banco de dados. Operações como a criação, destruição, migração, padronização e limitação de máquinas virtuais são algumas das funcionalidades que ele pretende fornecer de forma simples, escalável e elástica tanto para nuvens privadas quanto públicas, grandes ou pequenas. Ele é composto de três componentes: *Compute*, *Object Storage* e *Image Service*.

- O *Compute* é um controlador da infraestrutura da nuvem, usado para iniciar as máquinas virtuais, ou instâncias como são chamadas no produto, para um usuário ou um grupo. Também é usado para configurar a rede de cada instância ou projeto que contenha múltiplas instâncias.

- O *Object Storage* é um sistema que armazena objetos em um sistema massivamente escalável de alta capacidade com redundância e tolerância a falhas. Ele tem uma variedade de aplicações, como arquivar e fazer *backup* de dados, servir gráficos ou vídeos, armazenar dados estáticos secundários ou terciários, desenvolver novas aplicações com integração de armazenamento de dados e criar a elastici-

dade e flexibilidade de um armazenamento baseado em nuvem para aplicações Web.

- O *Image Service* é um sistema de pesquisa e recuperação para imagens de máquinas virtuais. Ele pode ser configurado de três formas: usar o *Object Storage* para armazenar imagens; usar a infraestrutura da *Amazon S3* diretamente para armazenar imagens; ou usar o armazenamento do *Amazon S3* com o *Object Store* como um intermediário para o acesso.

O controle de acesso à nuvem é feito por uma interface de programação (*API*) que provê recursos tanto para administração quando para acesso. Os usuários acessam diretamente a *API* sem precisar se preocupar em como a operação será realizada na infraestrutura física. Enquanto isso, o sistema se encarrega de usar o *Image Service* para publicar as imagens das máquinas virtuais, usando ou não o *Object Store*, para que o *Compute* possa distribuir, gerenciar e modificar as máquinas virtuais na infraestrutura física da melhor forma possível. Tanto esses componentes quanto os subcomponentes que os compõem se comunicam através de um servidor de filas de mensagens dedicado com registro de consumidor e produtor. Toda a comunicação feita entre os módulos passa por esse servidor de filas, de forma que qualquer processo pode produzir, consumir ou observar as mensagens cadastradas em alguma fila de mensagens existente. Isso permite que a comunicação entre os processos seja independente de implementação e que módulos externos possam interagir com o sistema apenas utilizando a organização das filas. Além disso, todas as informações da nuvem são armazenadas em um banco de dados *MySQL*, para uso de todos os módulos, que permite acessar essas informações a qualquer momento diretamente pelo servidor de banco de dados. Esse banco de dados fica localizado no controlador da nuvem, porém, todos os nós que utilizam o *Compute* o acessam diretamente.

A abstração provida pela *API* se dá por comandos simples e papéis de usuários. Um usuário com o papel de administrador tem privilégios como criar um projeto de computação em nuvem utilizando os recursos disponíveis, atrelar uma faixa de rede para esse projeto, determinar credenciais, publicar imagens de sistema e manipular a criação, destruição e migração de máquinas virtuais. A criação de uma máquina virtual em uma configuração preestabelecida se dá por um comando onde é determinada a imagem base, o tipo de máquina virtual, que determina quantos recursos ela reservará, e uma chave *RSA* para permitir acesso seguro à máquina. Enquanto isso, um usuário comum tem papel limitado como acessar uma máquina virtual através de uma chave *RSA*, dentre outros papéis que podem ser modificados ou definidos pelo administrador.

Existem outras ferramentas com o mesmo objetivo, as principais sendo o *Eucalyptus* (Nurmi et al., 2008) e o *OpenNebula* (OpenNebula, 2012). O *Eucalyptus* foi o primeiro sistema aberto de gerência de nuvem a se popularizar. Ele foi apresentado em 2008 com o intuito de apresentar uma solução para administradores de sistemas e pesquisadores para gerenciamento simplificado de uma nuvem. Foi utilizada uma interface de usuário definida pelo *Amazon EC2* para facilitar a transição de usuários desse ambiente. Por ser o precursor das ferramentas abertas de gerência de nuvens, o *Eucalyptus* popularizou e teve bastante utilização durante os últimos anos, porém, recentemente ele tem perdido muito espaço para o *OpenStack*.

O *OpenNebula* é outra ferramenta semelhante. Ele consiste em um sistema bastante semelhante ao *OpenStack*, porém divergente em algumas filosofias. Enquanto o *OpenStack* se baseia em criar um ambiente modular com um mínimo de ferramentas dependentes do ambiente possível, o *OpenNebula* tenta entregar uma solução mais completa para um sistema mais específico. Ele já tem uma participação mais sólida na comunidade, uma vez que começou de um projeto de pesquisa em 2005, bem antes do aumento do interesse na área de computação em nuvem.

Comparado a essas duas soluções, o *OpenStack* é a ferramenta do gênero que está tendo maior atenção da comunidade no momento, sendo inclusive oferecido oficialmente por distribuições famosas como o *Ubuntu* e utilizado por organizações importantes. Ele oferece serviços que além de simplificar o gerenciamento das máquinas virtuais centralizam em um só lugar diversas informações que possibilitam tanto a separação das redes virtuais como o levantamento de informações sobre separações entre elas. Ele foi escolhido por ser a ferramenta do gênero mais utilizada pela comunidade e possuir todas as informações necessárias para o desenvolvimento da aplicação.

2.1.4 Integração entre Open vSwitch e Xen

Monitores de máquinas virtuais como o *Xen* (Barham et al., 2003) apresentam um esquema de virtualização da rede que cria um *switch* virtual para conectar os dispositivos de rede de cada máquina virtual ao dispositivo de rede da máquina hospedeira, criando uma ligação entre elas e a rede real. No caso do *Xen*, esse *switch* virtual é criado com a ajuda de um módulo do *kernel* do Linux que permite a criação desse dispositivo. Esse processo é ilustrado na figura 2.1. É criado um *switch* virtual chamado *xenbr0* com diversas portas virtuais. Cada porta virtual é ligada em outro dispositivo virtual através das interfaces virtuais *vif* e a placa física *eth0* se torna o *Gateway*, enquanto que uma interface virtual também chamada de *xenbr0* é criada para que o tráfego das aplicações da máquina hospedeira também passem pelo *switch* virtual.

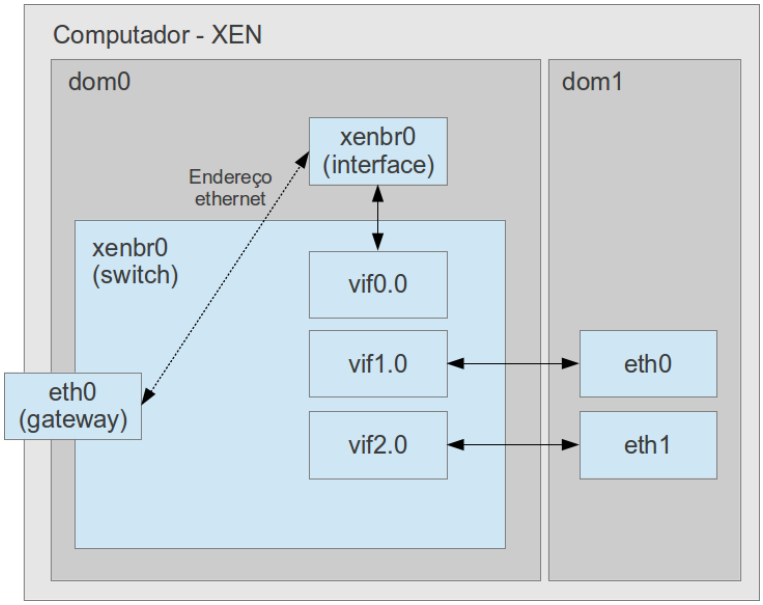

Figura 2.1. Rede no Xen

A criação do *switch* virtual e a conexão das máquinas virtuais a ele são realizadas pelo *Xen* utilizando funções do próprio sistema do *Linux* para utilizar o módulo *bridge* do *kernel*. Como citado anteriormente, o *Open vSwitch* possui uma funcionalidade de substituir esse módulo do *Linux* para que o sistema utilize o *Open vSwitch* ao invés do módulo *bridge*. Isso ocorre de forma transparente, sem que os próprios comandos do *Linux* precisem de modificações para funcionar. Portanto, ao utilizar essa funcionalidade, o *Xen* passa a utilizar de forma transparente o *Open vSwitch* para criar o seu *switch* virtual e conectar suas máquinas virtuais.

2.1.5 Integração entre OpenStack e Xen

O *OpenStack* suporta o monitor de máquinas virtuais *Xen* através de duas formas possíveis: utilizando a biblioteca de generalização de gerenciamento de máquinas virtuais *Libvirt*[1] (Bolte et al., 2010) e através de uma biblioteca de acesso à interface do produto *XenServer*[2] (Citrix, 2012). Foi escolhida a primeira, devido ao maior suporte à biblioteca *Libvirt* e sua independência de sistemas, uma vez que o *XenServer* se trata de uma solução já integrada que dificultaria a integração com os demais serviços.

[1] O *Libvirt* é uma biblioteca para gerenciamento de virtualização que promete entregar uma interface comum para gerenciar máquinas virtuais independentemente do monitor de máquinas virtuais utilizado.

[2] O *Xenserver* é um produto da empresa *Citrix* que oferece um sistema automatizado para instalação e gerência de um ambiente de *datacenter* com máquinas virtuais.

Para a biblioteca *Libvirt* integrar o *Xen* ao *OpenStack*, é necessário que o *Xen* esteja configurado para utilizar uma interface *HTTP* de acesso ao monitor. Dessa forma é possível integrar os dois serviços. Da parte do *OpenStack*, também é necessário realizar uma série de configurações específicas para o monitor *Xen*. Essas configurações estão descritas no apêndice B na explicação do roteiro de configuração.

Também é interessante separar a interface de rede pela qual o *OpenStack* comunica entre seus componentes da interfaces de rede que o *Xen* utiliza para fazer a comunicação entre as suas máquinas virtuais. Esse procedimento de separar uma interface de rede para o controle e outra para o serviço de máquinas virtuais é bastante utilizado em *datacenters* virtualizados em geral, pois ambos os tráfegos podem ser altos, e a separação do tráfego de controle aumenta a confiabilidade do sistema. Neste projeto essa abordagem foi utilizada por refletir uma situação comum no ambiente alvo e evitar problemas do uso dos dois tráfegos em uma só interface.

2.2 OpenFlow

A tecnologia *OpenFlow* surgiu como uma maneira de permitir que o comportamento dos elementos de comutação de rede (*switches* e roteadores) sejam facilmente programados para obter comportamentos diferentes dos usualmente disponíveis. Seu princípio de operação se baseia na separação dos planos de controle e dados da rede, criando uma interface de programação para o primeiro. Nesse contexto, o plano de controle consiste na lógica responsável pelo estabelecimento das regras a serem aplicadas no encaminhamento de pacotes, enquanto o plano de dados representa o hardware responsável pelo tratamento em tempo real de cada pacote recebido pelo *switch*, com base nas regras previamente definidas pelo plano de controle. A interação entre os dois planos se dá através de uma tabela de regras de encaminhamento/roteamento, que é preenchida pelo plano de controle e consultada pelo plano de dados para cada pacote recebido. *OpenFlow* define uma interface de programação e um conjunto de ações que podem ser definidas na tabela de encaminhamento, permitindo que aplicações externas controlem o encaminhamento de pacotes, sem afetar seu desempenho de forma significativa (já que o plano de dados continua sendo implementado em hardware). Essa tecnologia pode ser aplicada tanto na contrução de *switches* reais, com hardware especializado para suportá-la, quanto em ambientes virtualizados, com implementações em software.

A tabela de regras de fluxos é manipulada por um controlador externo (um PC executando um software de controle) através do protocolo definido. A figura 2.2 representa o funcionamento da comunicação do *OpenFlow*. O *switch* utiliza sua tabela

de fluxos em hardware e se comunica por software com um PC controlador através de uma conexão criptografada SSL.

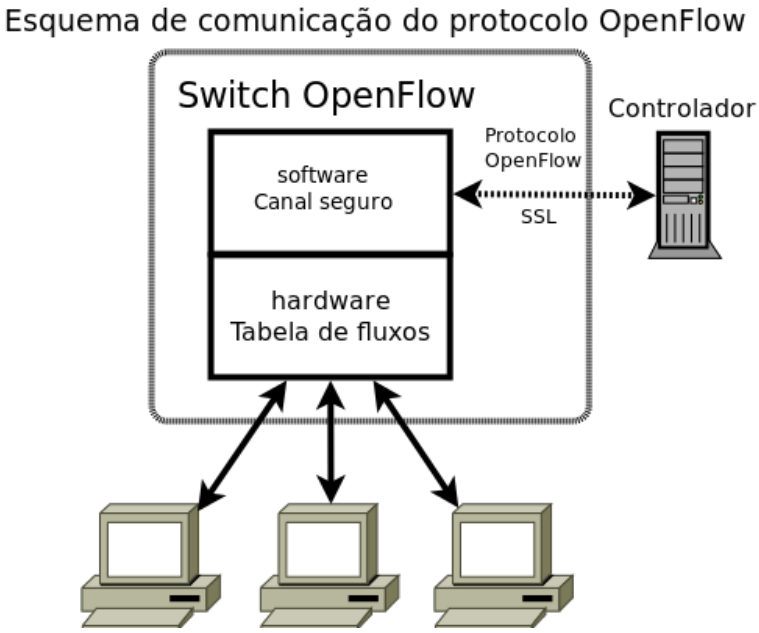

Figura 2.2. Diagrama de funcionamento da comunicação do OpenFlow

2.2.1 A API *OpenFlow*

O protocolo *OpenFlow* define uma interface de programação (*API*) comum para acessar a funcionalidade da tabela de encaminhamento dos *switches*, independente das implementações de cada fabricante. Cada entrada da tabela de encaminhamento define um padrão que pode incluir até 10 campos dos cabeçalhos usualmente encontrados em uma rede local (como Ethernet, IP e TCP). Pacotes recebidos são comparados com esse padrões e, caso haja um casamento completo, as ações associadas àquela entrada são executados. A figura 2.3 mostra os campos considerados na versão atual.

In	VLAN	Ethernet			IP			TCP	
Port	ID	SA	DA	Type	SA	DA	Proto	Src	Dst

Figura 2.3. Campos disponíveis para a identificação de fluxos na primeira versão do OpenFlow

Uma vez que um pacote case com uma das entradas da tabela, diversas ações podem ser definidas para serem aplicadas ao pacote. Essas ações se classificam em uma das seguintes categorias:

1. Encaminhar os pacotes deste fluxo a uma certa porta (ou portas) do *switch*. Isso permite que pacotes sejam roteados pela rede.

2. Re-escrever o conteúdo de campos específicos dos cabeçalhos encontrados.

3. Encapsular e encaminhar os pacotes de um fluxo para o controlador. O pacote é entregue através de um canal seguro entre o *switch* e o controlador, que pode então processá-lo e tomar decisões sobre o fluxo.

4. Descartar os pacotes deste fluxo.

Uma entrada pode definir mais de uma ação, como re-escrever alguns cabeçalhos e então encaminhar o pacote resultante por uma certa porta de saída.

Com essas ações, um programa (controlador) pode se conectar ao *switch* e determinar que pacotes que se casem com um certo padrão lhe sejam enviados. Ao receber cada pacote, o controlador pode decidir como o fluxo por ele definido deve ser processado e enviar mensagens para o *switch* para estabelecer uma nova entrada na tabela de encaminhamento específica para aquele fluxo. Essa nova entrada conteria as ações a serem executadas, como descartar o fluxo, ou encaminhá-lo para uma certa porta de saída. A especificação completa do protocolo *OpenFlow* está disponível no site oficial do *OpenFlow* (OpenFlow, 2012).

2.2.2 Controladores *OpenFlow*

Para simplificar o trabalho de desenvolvedores interessados em utilizar OpenFlow, diversos controladores já foram desenvolvidos. Esses controladores cuidam dos detalhes de implementação do protocolo propriamente dito, exportando uma interface de programação de mais alto nível que pode ser usada para representar os comportamentos específicos planejados pelo desenvolvedor.

O controlador *OpenFlow*, como descrito anteriormente, é um servidor externo executando um software que se conecta aos *switches OpenFlow* através de uma conexão criptografada SSL. Esse controlador tem o objetivo de receber pacotes encapsulados de cada fluxo dos *switches*, realizar uma computação previamente determinada e responder o *switch* com uma mensagem seguindo o protocolo *OpenFlow* determinando a ação relativa a todos os pacotes subsequentes àquele fluxo.

Esse controlador, por ser uma máquina externa completa, pode realizar qualquer tipo de computação necessária a essa lógica de tratamento de fluxos, não estando limitado aos recursos de um *switch*. Dessa forma, o controlador *OpenFlow* tem o poder de manipular toda a rede mantendo uma visão centralizada de todos os *switches*, todas as máquinas ligadas e todos os fluxos que passam por eles.

Toda essa funcionalidade permitiu a criação dos chamados *Network Hypervisors*. Esses programas consistem em um software que roda em um controlador *OpenFlow* provendo uma interface para manipulação dos fluxos com o intuito de permitir a criação de módulos que programem os *switches OpenFlow* para separar a visão da rede para as máquinas da rede física subjacente existente. Assim como um monitor de máquinas virtuais (*Hypervisor*) manipula máquinas virtuais sobre uma máquina real, um *Network Hypervisor* manipula redes virtuais sobre uma infraestrutura de rede real.

2.2.3 Controladores disponíveis

Diversos controladores já foram desenvolvidos e muitos estão disponíveis como software livre e aberto. A diversidade de controladores ilustra as diferentes abstrações disponíveis para se expressar o tratamento de fluxos de rede. Esta seção menciona alguns dos principais controladores existentes durante o período de desenvolvimento desta dissertação.

O *NOX* (Gude et al., 2008) é considerado o controlador original *OpenFlow*. Apresentado pelos criadores do *OpenFlow* como um *Network Hypervisor* bastante extensível e capaz de proporcionar um ambiente para a programação de aplicações modulares que reagem a diversos eventos levantados originalmente pela rede ou pelos próprios módulos. Ele pretende oferecer uma interface de programação para o acesso à rede e a capacidade de manipular pacotes *OpenFlow* para os *switches*. Ele tem como principal função hoje o desenvolvimento de controladores eficientes em *C++*. Opera sobre o conceito de fluxos de dados checando o primeiro pacote de cada fluxo e procurando na tabela de fluxos a política a ser aplicada. Atualmente a maioria dos projetos de pesquisa na área são baseados no *NOX*, que é um sistema operacional simples para redes e que provê primitivas para o gerenciamento de eventos bem como funções para a comunicação com *switches*.

O *NOX* obteve uma grande aceitação entre os pesquisadores da área de *SDN*. A existência de duas interfaces, *C++* e *Python*, permite que o mesmo ambiente seja utilizado em situações que exigem alto desempenho e em casos onde a capacidade de expressão de *Python* agilizam o desenvolvimento e simplificam o código resultante. Posteriormente, foi também apresentado pela *UC Berkeley* o *POX*, que foi desenvolvido

com base no modelo do *NOX*, mas com a premissa de ser completamente escrito em *Python*, resultando em uma interface mais elegante e simples dentro daquela linguagem. Os desenvolvedores do *POX* acreditam que este seja adequado para substituir o *NOX* nos casos em que *Python* é utilizado, enquanto o *NOX* ainda permanece adequado para implementações que tenham demandas mais elevadas em termos de desempenho.

O *NOX* e o *POX* são os *Network Hypervisors* mais importantes para o projeto por serem mais sólidos e apresentarem um conjunto de funcionalidades que comportam as funcionalidades descritas. Entretanto, existem outros *Network Hypervisors* que são criados para uma finalidade específica, ou apenas apresentando uma forma de abstração diferente para resolver um problema específico que merecem ser mencionados.

O *Beacon* (Beacon, 2012) é outro controlador baseado em *Java* que suporta tanto a operação baseada em eventos quanto em *threads*. O projeto vem sendo desenvolvido já há um ano e meio, sendo considerado estável. Seu registro de operação inclui a gerência de uma rede com 100 *switches* virtuais e 20 *switches* físicos em um *datacenter* experimental. O sistema tem uma estrutura modular que permite que o controlador seja atualizado em tempo de execução sem interromper outras atividades de encaminhamento de pacotes. O pacote opcionalmente incorpora o servidor web *Jetty* e um arcabouço extensível para o desenvolvimento de interfaces de usuário personalizadas.

Frenetic (Foster et al., 2011) é um sistema baseado em linguagem funcional desenvolvido para programar redes *OpenFlow*. *Frenetic* permite que o operador da rede, ao invés de manualmente configurar cada equipamento de rede, programe a rede como um todo. Ele é implementado sobre o *NOX*, em *Python*, e foi projetado para resolver problemas de programação com o *OpenFlow/NOX*. Ele faz isso introduzindo abstrações funcionais para permitir programas modulares e a composição desses programas.

Existem ainda diversos outros *Network Hypervisors*, cada um com suas particularidades, como uma preocupação maior com desempenho, tolerância a falhas ou facilidade de escrita de aplicações (Trema, 2012; Cai et al., 2010; Hinrichs et al., 2009; Koponen et al., 2010; Kohler et al., 2000; Mundada et al., 2009; Floodlight, 2012; SNAC, 2012; Ganguly, 2011). Essa variedade existe porque muitas vezes o ambiente alvo tem demandas diferentes, como a necessidade de redundância em sistemas que dependem muito da qualidade de serviço ou o desempenho em sistemas que exigem uma resposta mais rápida.

2.2.4 Redes Definidas por Software

Um dos impactos da definição da tecnologia *OpenFlow* e de *Network Hypervisors* é a definição do conceito de Redes Definidas por Software, ou *SDNs* (*Software Defined*

Networks). Com base nos recursos providos pela tecnologia *OpenFlow*, SDNs pregam a construção de uma visão global da rede, sobre a qual diferentes aplicações podem ser criadas. A figura 2.4 demonstra a comparação de uma rede comum e uma SDN. Em uma rede tradicional, cada elemento de comutação (*switch* ou roteador) é uma unidade de processamento independente, com uma arquitetura de software que determina seu comportamento. Um roteador tem seu sistema operacional e executa aplicações que definem o protocolo de roteamento a ser utilizado, por exemplo. Em uma SDN, os elementos de comutação são ligados a um controlador de rede (*Network Hypervisor*), que podem então construir uma visão completa da rede. Sobre essa visão completa, aplicações podem ser desenvolvidas de forma mais simples e as ações por elas exigidas são transformadas pelo controlador em comandos que programam cada elemento de comutação. A figura 2.5 ilustra com mais detalhes essa organização.

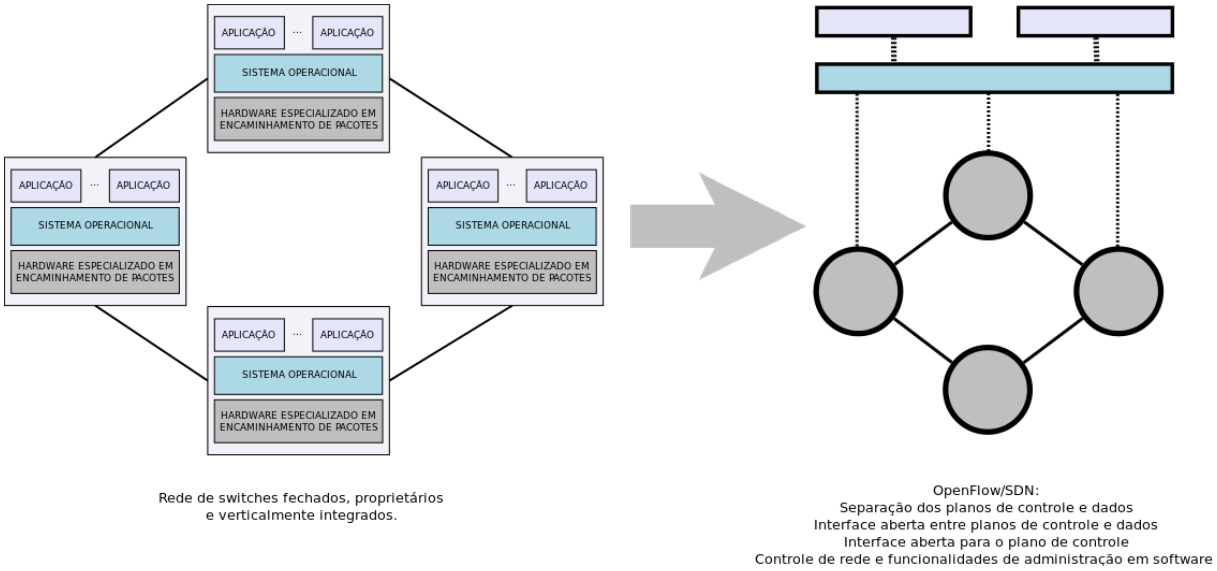

Figura 2.4. Diferenças entre modelo comum e modelo com SDN

A *SDN*, como é possível ver na figura 2.5, separa todas essas preocupações do plano de controle, como o roteamento, para uma aplicação separada que opera sobre a abstração da visão global da rede. Essa abstração é implementada como um sistema operacional da rede que proporciona recursos, como a visão lógica da rede, para módulos responsáveis por realizar essas operações, deixando para os nós de dados apenas a responsabilidade de encaminhar os pacotes seguindo uma tabela de fluxos.

Essa abstração, como discutido anteriormente, é alcançada através do uso de *switches OpenFlow* e um controlador que programa os mesmos através de regras *OpenFlow* e visão global da rede. Essa visão global pode ser implementada também considerando-

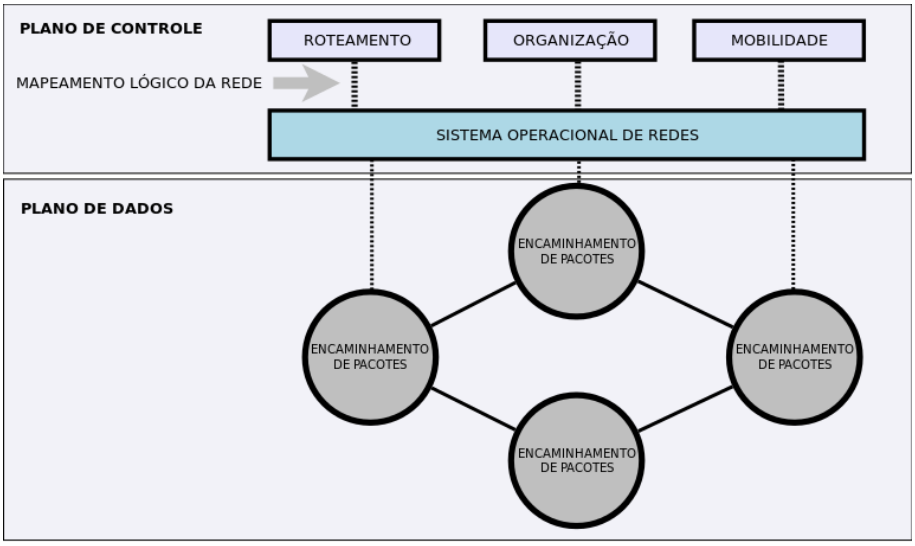

Figura 2.5. Separação entre planos de controle e dados em uma SDN

se uma visão de rede com duas camadas, onde os comutadores da borda sejam completamente programáveis (*OpenFlow*), enquanto os comutadores internos obedecem a regras de encaminhamento mais simples (p.ex., MPLS). Essa organização se torna particularmente interessante se considerarmos que, em um *datacenter* virtualizado, a borda da rede é completamente formada por *switches* virtuais em sofware (*Open vSwitch*, por exemplo, que implementa *OpenFlow*). Esses *switches* podem ser usados para implementar diversos comportamentos complexos, desde que se mantenha em mente que dentro do núcleo da rede o processamento precisa continuar obedecendo ao comportamento usual da rede *Ethernet*. Apesar dessa limitação restringir o tipo de solução possível, ela oferece uma solução de compromisso importante que não exige a substituição do hardware existente em uma grande instalação.

2.2.5 POX

Para gerenciar os *switches OpenFlow* neste projeto, foi escolhido como *Network Hypervisor* o *POX* da *UC Berkeley*. O *POX* vem sendo desenvolvido como um sucessor natural do *NOX* para iniciativas de pesquisa e ensino. O objetivo dos desenvolvedores é que ele venha a substituir o *NOX* nos casos onde desempenho não seja um requisito crítico — essencialmente, nos casos onde a interface *Python* é utilizada. Nesse aspecto, *POX* traz uma interface mais moderna e uma implementação mais elegante, além de oferecer melhor desempenho que o *NOX* com *Python* como pode ser visto na figura 2.6.

NOX e outros controladores de primeira geração são construídos ao redor do

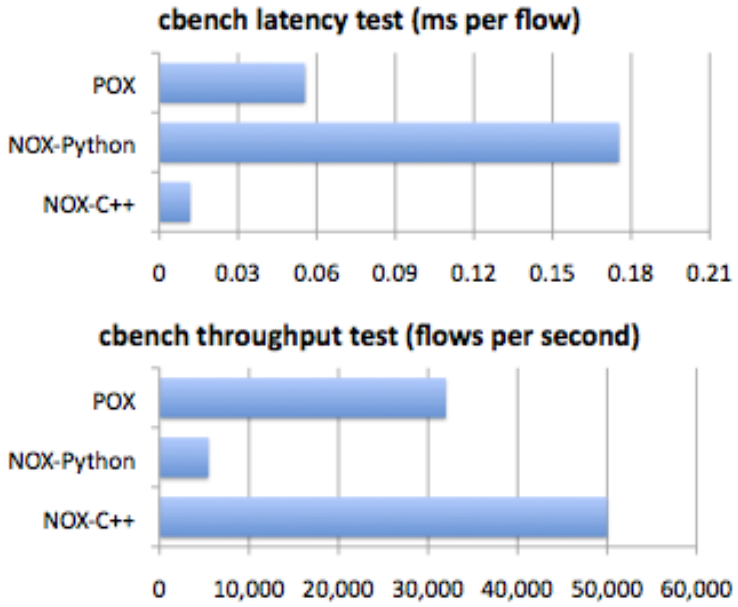

Figura 2.6. Comparação de desempenho entre POX e NOX, com suas duas interfaces (C++ e Python). Figura publicada por Murphy McCauley no site `http://www.noxrepo.org/2012/03/introducing-pox/`.

conceito de mensagens *OpenFlow* como primitivas básicas. Assim sendo, as aplicações se organizam ao redor de ciclos de recebimento/tratamento/geração de mensagens. O *POX* está sendo organizado ao redor da noção de visão global da rede: aplicações não precisarão necessariamente de receber e enviar mensagens diretamente, mas poderão consultar a visão corrente da rede e atuar diretamente sobre ela para obter seus objetivos (os recursos para esse fim ainda estão em desenvolvimento). Outros elementos que fazem parte do contexto de projeto do *POX* incluem a previsão de recursos para a distribuição dessa visão global e para a depuração de aplicações desenvolvidas sobre essa visão.

POX funciona com o conceito de orientação a eventos para gerenciar a rede formada por *switches OpenFlow*. Os componentes, que são as aplicações desenvolvidas sobre o *POX*, podem reagir a eventos definidos pelo núcleo do *POX* ou levantar e reagir a eventos criados pelos próprios componentes. São exemplos de eventos definidos pelo *POX* a entrada ou saída do *switch* na rede através de conexão ao *Network Hypervisor*, a identificação de um novo fluxo por um *switch* pertencente à rede ou modificações de portas ou remoção de um fluxo de um *switch* devido a inatividade. O *POX* também oferece uma biblioteca para a criação de mensagens *OpenFlow* para programar os *switches* da rede de acordo com as instruções geradas pela computação do controlador com visão global da rede.

Por exemplo, quando um *switch OpenFlow* se conecta à rede é gerado um evento de *ConnectionUp*, como definido pelo *POX*. O componente do *POX* pode se cadastrar para reagir a esse evento e adicionar informações sobre esse *switch* à sua visão global da rede. Em seguida, o primeiro pacote que for identificado pelo *switch* conectado gera um evento de *PacketIn* e o componente pode reagir a ele utilizando as informações do pacote identificado para tomar uma decisão com base na visão global da rede para gerar uma mensagem *OpenFlow* que define uma ação que será instalada na tabela de fluxos daquele *switch*. Essa entrada garante que todos os pacotes do mesmo fluxo que o seguirem tenham o mesmo tratamento programado pelo controlador para o primeiro pacote.

Para este projeto foi escolhido o *POX* por ser uma ferramenta de *Network Hypervisor* livre com facilidade de criação de módulos e por possuir no futuro um suporte maior dos desenvolvedores. O apêndice A demonstra uma implementação de um *switch ethernet* simples usando o *POX* para ilustrar a facilidade de programação e demonstrar algumas bibliotecas do *POX*. Este projeto também gerou um *tutorial* dando mais detalhes sobre o *POX* em uma publicação para um minicurso para o Simpósio Brasileiro de Redes de Computadores (SBRC, 2012).

2.3 Trabalhos relacionados

Recentemente vários trabalhos têm sido publicados com a intenção de entender melhor e oferecer soluções para o ambiente de *datacenters* principalmente no que diz respeito a soluções que geram um retorno econômico em virtude do uso de hardware mais barato.

Greenberg et al. (2009) reuniram um estudo bastante interessante sobre os custos de um *datacenter* de computação em nuvem. Alguns fatores muito importantes em relação a esse tipo de ambiente são descritos, como a tendência de possuir um número extremamente elevado de máquinas para um pessoal comparativamente menor (hoje são comuns *datacenters* com mais de 100.000 servidores com frações de pessoas de 1:1000 máquinas.). Também são apresentadas conclusões bastante interessantes sobre a rede de *datacenters* como a diferença de preço notável entre switches L2 e L3, fazendo com que a engenharia de organização da rede seja uma tarefa bastante importante, e o uso pesado de *VLANs* para realizar isolamento e disseminação de pacotes específicos.

Através dessa análise é possível perceber que os custos de manutenção do uso de *VLANs* é bastante considerável em virtude da fração de pessoas/máquinas e a frequência com que sua manutenção é ocasionada. Isso em um *datacenter* convencional já representa um peso importante devido à organização intrínseca à infraestrutura

física. Em *datacenters* que oferecem o serviço de computação em nuvem esse peso é muito maior devido à frequência com que essa manutenção deveria ocorrer à medida que clientes modificam suas redes e demandam pelo isolamento na comunicação de suas máquinas virtuais. Sem contar na necessidade de migração de uma máquina virtual devido a possíveis pontos específicos da infraestrutura que ficam sobrecarregados com muitas máquinas demandando processamento, o que é um problema grave e tratado por algoritmos como o *Sandpiper* (Wood et al., 2007). Todas essas evidências apontam para um custo muito grande e pouco escalável para a utilização de *VLANs* no isolamento de *datacenters* de serviços de computação em nuvem.

Já Fares et al. (2008) apresentam um algoritmo de roteamento e uma organização de rede a fim de diminuir os custos de infraestrutura em *switches* de um *datacenter*. Ele consegue reduzir o custo da arquitetura de *switches* através da utilização de aparelhos baratos ao invés de aparelhos caros especializados, mantém a confiabilidade com a organização de redundância programada da hierarquia e garante o desempenho através do uso de algoritmos de roteamento modificados visando essa organização.

Alizadeh et al. (2010) trabalham numa solução de nível mais baixo, apresentando uma modificação no algoritmo do *TCP* com o intuito de evitar o esgotamento dos *buffers* dos *switches* através da marcação dos pacotes e a auto-detecção proativa dos lados da transferência para reduzir o tráfego sem o enchimento da fila. O *switch* quando começa a detectar uma ocupação do seu *buffer* perto do esgotamento marca os pacotes *TCP* que passam por ele com uma *tag*. O receptor da transferência detecta a frequência que esses pacotes marcados estão chegando e gera uma mensagem para o emissor para ele diminuir a velocidade de transmissão explicitamente. Com o melhor uso dos *buffers* dos *switches* a rede do *datacenter* aumenta a confiabilidade, latência e utilização de recursos além de diminuir perda de pacotes, atraso e fenômenos como o *TCP Incast* (Chen et al., 2009; Vasudevan et al., 2009), que é um problema causado por um alto número de requisições externas a um ponto comum de uma só vez, gerando muita colisão, atraso e retransmissões devido ao algoritmo de congestionamento do *TCP*. Isso beneficia *datacenters* de forma geral, mas principalmente os que possuem menos hardware especializado com capacidades maiores.

Portanto, abrir mão de hardware especializado e caro para utilizar soluções mais elegantes utilizando aparelhos de baixo custo é uma situação bastante comum que apresenta ganhos significativos. Muitas empresas, principalmente quando estão abandonando o posto de "pequenas", precisam aumentar a oferta de seus serviços mas nem sempre dispõem do investimento necessário para fazer isso da forma convencional ou ainda precisam apresentar uma estratégia mais eficiente para competir com empresas maiores. Isso trabalha na mesma linha da solução proposta pelo *DCPortals*.

Na questão de separação de redes, Cabuk et al. (2008) apresentam um estudo comparativo para a virtualização das redes. Eles apresentaram pela primeira vez o conceito de virtualizar as redes e apresentaram três técnicas para fazer isso. Uma delas é o *MAC Rewriting*, uma técnica que visa separar os endereços de *ethernet* das máquinas virtuais da infraestrutura real através da re-escrita do campo de endereço *ethernet* dos pacotes quando eles deixam ou chegam às máquinas físicas que mantêm máquinas virtuais. Uma tabela que mapeia o endereço *ethernet* para o endereço IP das máquinas virtuais é mantida em todas as máquinas físicas para que um módulo possa fazer essa escrita quando for necessário. Assim que essa tradução é feita, também é adicionada uma lógica no monitor de máquinas virtuais para controlar o acesso entre redes virtuais não conectadas. Outra técnica discutida é o *EtherIP*, que consiste em um protocolo que encapsula o endereço *ethernet* das máquinas virtuais em um cabeçalho sobre os pacotes IP e gerando uma camada de rede *ethernet* por cima da camada de rede IP. Assim como a técnica anterior, existe a necessidade que o monitor de máquinas virtuais tenha uma inteligência programada sobre a separação de redes para que o isolamento ocorra no momento do encapsulamento de desencapsulamento dos pacotes. A última técnica apresentada é o uso automatizado de *tags VLAN*, que, como já foi discutido, esbarra em diversas limitações de escalabilidade e gerência. Todas as técnicas discutidas utilizam também a programação de endereços *multicast* para mapear pacotes de *broadcast* e *multicast* dentro da rede virtual determinada.

Essas técnicas não só representam a importância de uma separação lógica de redes em uma infraestrutura compartilhada como também apresentam técnicas interessantes que podem ser utilizadas em sistemas mais complexos. Cabuk et al. (2007) apresentam um trabalho bastante semelhante em objetivo com este projeto. Começando por definir o conceito de domínios virtuais confiáveis (*Trusted Virtual Domains*, ou *TVDs*), que são separações lógicas isoladas da rede independentes da infraestrutura, assim como neste projeto definimos como redes virtuais ou redes lógicas privativas. Para implementar esse isolamento é criado um módulo interno à máquina virtual para interceptar os pacotes direto na interface de rede e outro módulo na máquina física para realizar o processamento relativo à separação das redes. Duas técnicas de virtualização de redes apresentadas no trabalho anterior são utilizadas e comparadas, as *tags VLAN* e o *EtherIP*. Esse trabalho apresenta um sistema com um objetivo semelhante ao deste projeto, o que é interessante do ponto de vista de importância do serviço e a necessidade de resolver esse problema. Entretanto, tanto as técnicas quanto o sistema apresentado se baseiam em modificações bastante intrusivas no monitor de máquinas virtuais e em limitações bastante relevantes já discutidas como o número de *tags VLAN* e endereços de *multicast* IP.

O *DCPortals* introduz o conceito de Redes Definidas por Software para resolver esse problema de isolamento de redes lógicas privativas. Com isso é possível apresentar uma solução mais elegante, através do uso do conceito de *Network Hypervisor* como um ponto centralizado de visão global que programa os *switches* para obedeceram suas políticas. Dessa forma, toda a lógica do isolamento é de competência do controlador e não depende de nenhuma interferência nas máquinas virtuais nem nos monitores de máquinas virtuais. Também é importante mencionar a importância que o uso de redes definidas por software representa para um sistema desses. Essa abordagem fornece uma nova gama de possibilidades para desenvolvimento de mecanismos mais elaborados, como os descritos posteriormente para eliminar as limitações que os sistemas anteriores possuíam com o uso de *tags VLAN* e endereços de *multicast* IP. Também existe a possibilidade de criação de outros mecanismos futuros que podem tirar proveito das vantagens das *SDNs*, como algoritmos de roteamento que usem a condição de uso do *datacenter* em consideração ou garantam um melhor uso dos recursos computacionais do mesmo como um todo.

Capítulo 3

Desenvolvimento

A criação dos *Network Hypervisors* foi uma evolução muito interessante que forneceu um ferramental para lidar com vários problemas de *datacenters* virtualizados. Embora já existam alguns *switches OpenFlow* comerciais disponíveis no mercado, eles ainda são poucos e caros. Os *datacenters* virtualizados atuais possuem uma infraestrutura já existente com um número grande de *switches ethernet* convencionais, portanto, o investimento para trocar todos eles para migrar para um ambiente completamente compatível com a tecnologia *OpenFlow* muitas vezes não é viável.

Porém, os monitores atuais de máquinas virtuais no ambiente Linux costumam utilizar um módulo do *kernel* para criar um *switch* virtual interno às máquinas hospedeiras para conectar as interfaces de rede das máquinas virtuais. Se apenas esses *switches* virtuais forem compatíveis com a tecnologia *OpenFlow*, é possível desenvolver uma aplicação que tenha conhecimento de que somente os *switches* finais serão compatíveis enquanto que existirão *switches* convencionais ligando a infraestrutura real de rede entre eles.

Implementar *SDNs* em *datacenters* virtualizados usando os *switches* virtuais de borda e mantendo o uso dos *switches ethernet* comuns possibilita uma gama de novas funcionalidades, além de melhorar o uso dos recursos da infraestrutura sem a necessidade de investimento em troca de aparelhos. Para isso, é necessário a utilização de um *switch* virtual compatível com o protocolo *OpenFlow* que gerenciaria as máquinas virtuais e uma aplicação do controlador que gerenciaria esses *switches* virtuais de forma que o fato de os *switches* reais da infraestrutura serem *switches ethernet* comuns não influencie no seu funcionamento. Além disso, é interessante manter informações centralizadas sobre a localização, disparo e funcionamento das máquinas virtuais para que seja possível a criação de um sistema de redes virtuais multi-inquilinos, que é o foco de *datacenters* de computação em nuvem. Portanto, é uma limitação desse ambiente que

todas as máquinas comercializadas estarão por trás de um *switch* virtual de borda, não possibilitando uma arquitetura heterogênea com outras máquinas reais ou máquinas reais fora de *switches* virtuais programados pelo sistema.

Mais que aproveitar a infraestrutura existente, o uso de *SDNs* pode implementar algoritmos que melhorem o aproveitamento de recursos computacionais dessa infraestrutura. Por exemplo, é possível aproveitar a visão global centralizada da *SDN* para implementar uma técnica de re-escrita de endereços *MAC* (*MAC Rewriting*) para esconder os endereços *ethernet* das máquinas virtuais da rede física, diminuindo bastante a carga sobre os *switches ethernet* convencionais da infraestrutura subjacente.

Além disso, como citado por Greenberg et al. (2009) e discutido no capítulo 1, a rede de *datacenters* depende muito do uso pesado de *VLANs* para isolamento de disseminação de mensagens. O controlador *OpenFlow* pode diminuir ou até eliminar a necessidade do uso de *VLANs* utilizando SDNs.

Essa abordagem apresenta diversas vantagens, algumas delas estão descritas a seguir:

- **Menor uso da memória dos *switches* convencionais:** É possível re-escrever os endereços *ethernet* das máquinas virtuais para separar logicamente as redes virtuais da rede subjacente, evitando o acúmulo de rotas e endereços *ethernet* de máquinas virtuais não importantes para os *switches* reais. Com isso, pode ser evitado um problema que é o armazenamento em *switches* de endereços das máquinas virtuais, o que tem se tornado um problema maior devido ao número elevado de máquinas virtuais em um *datacenter* por conta da alta capacidade das máquinas físicas recentes e o alto número de servidores em *datacenters*.

- **Possibilidade da criação de redes virtuais privadas multi-inquilinos:** A limitação do tamanho das *tags VLAN* discutida anteriormente já é um problema para *datacenters* maiores. Com uma SDN alguma outra forma de isolamento poderia ser desenvolvida para que não exista nenhum limitante para o isolamento de redes específicas além da própria eficiência dos equipamentos.

- **Melhor uso da infraestrutura subjacente:** Com um sistema operacional próprio para desenvolver aplicações para mapear a visão lógica da rede para a infraestrutura física, é disponibilizado um *framework* para a implementação de vários algoritmos com visão global da rede para melhorar a comunicação tendo em vista tanto a organização física quanto quaisquer outras políticas que possam ser interessantes para o *datacenter*.

- **Possibilidade de criação de módulos que integrem operações do *datacenter* com a organização da rede:** Com a flexibilidade de manter uma visão virtual da rede para as máquinas finais, vários serviços podem ser integrados com essa abstração da rede a fim de melhorar o uso de recursos, como uma migração física de máquinas virtuais de acordo com a proximidade entre elas das redes virtuais, a fim de otimizar a comunicação entre máquinas de uma mesma rede virtual.

Este trabalho propõe o *DCPortals*, uma aplicação que executa sobre um *Network Hypervisor* utilizando informações de um gerenciador de computação em nuvem e programando uma série de *switches* virtuais compatíveis com *OpenFlow* internos a máquinas hospedeiras para poder entregar um serviço de isolamento de redes virtuais multi-inquilinos em um *datacenter* sem a necessidade de trocar equipamentos físicos da infraestrutura. Ele consiste na integração e manipulação de ferramentas de código livre para entregar um serviço de gerenciamento de máquinas virtuais capaz de criar redes virtuais multi-inquilinos com isolamento da rede física e das demais redes virtuais melhorando também a utilização dos recursos de rede subjacentes.

Neste capítulo é apresentada a arquitetura proposta, as ferramentas utilizadas e os detalhes de implementação.

3.1 Arquitetura do Sistema

O *DCPortals* foi implementado como um módulo que executa sobre o controlador *POX*. Ele acessa diretamente o banco de dados do *OpenStack* para poder retirar as informações sobre localização e isolamento das redes e das máquinas virtuais. Uma vez consultadas essas informações, ele cria mensagens *OpenFlow* para instruir aos *Open vSwitches* como lidar com cada fluxo de pacotes identificado das máquinas virtuais. O *OpenStack* controla o monitor de máquinas virtuais *Xen* em cada máquina hospedeira e o *Xen*, por sua vez, utiliza um *switch* virtual *Open vSwitch* para conectar suas máquinas virtuais à rede real. A figura 3.1 ilustra a relação entre o módulo e as ferramentas.

O administrador do sistema utiliza a interface de programação do *OpenStack* para disparar uma máquina virtual e identifica a rede a qual ela vai pertencer. O *OpenStack* define qual máquina física dentre as disponíveis que será escolhida para hospedar a máquina virtual e envia os recursos necessários para que o *Xen* inicie a máquina virtual e a conecte ao *Open vSwitch* configurado naquela máquina física. Assim que a nova máquina virtual envia o primeiro pacote para a rede, o *Open vSwitch* identifica um novo fluxo e, com o protocolo *OpenFlow*, o *POX* recupera as informações desse pacote.

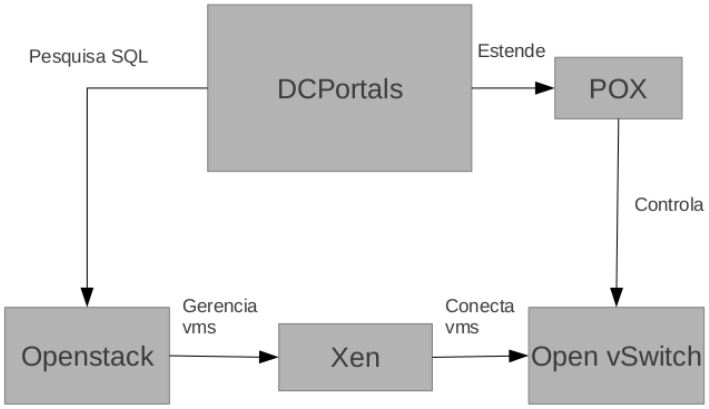

Figura 3.1. Arquitetura do sistema

Com isso, o *DCPortals* trata esse pacote utilizando informações do banco de dados do *OpenStack* e utiliza o *POX* para enviar uma mensagem *OpenFlow* que programa o *Open vSwitch* para seguir o mesmo tratamento para todos os pacotes seguintes do mesmo fluxo.

A seguir serão descritos detalhes do modelo implementado tal como os problemas que o módulo trata para poder entregar as funcionalidades esperadas.

3.2 Integração com o POX

O *DCPortals* é uma aplicação que executa sobre o *POX*. Ele utiliza as bibliotecas definidas pelo *POX* para gerar e enviar mensagens *OpenFlow* para programar os *Open vSwitches* presentes na rede de acordo com os fluxos que forem identificados das máquinas virtuais conectadas neles.

Na implementação foi definido que o *DCPortals* reagiria a dois eventos definidos pelo *POX*, o *ConnectionUp*, que é levantado sempre que um novo *switch* se conecta à rede e o *PacketIn*, que é levantado sempre que um novo pacote que não casa com nenhum fluxo instalado é identificado por um *switch*. O primeiro evento é tratado com a criação de uma estrutura de dados que identifica o novo *switch* na rede. Essa estrutura será responsável por armazenar as informações daquele *switch*, como os endereços *ethernet* das máquinas que estão conectadas a ele. Já o segundo evento é tratado com a aplicação da lógica de isolamento através da instalação de fluxos nos *switches*. Detalhes de como essas operações são realizadas serão discutidos posteriormente.

O *DCPortals* é levantado no carregamento do *POX* através de seus parâmetros

de execução, da mesma forma que são levantadas quaisquer aplicações que executem sobre ele.

3.3 Integração com o OpenStack

A aplicação criada realiza acessos ao banco de dados *MySQL* criado pelo *OpenStack* com informações sobre as máquinas virtuais para realizar as decisões para implementar o isolamento. Durante o tratamento dos novos fluxos identificados são feitas algumas pesquisas *SQL* para reunir informações sobre as máquinas virtuais:

- **Chave RSA de uma máquina virtual:** dado um endereço *MAC* de uma máquina, pesquisa no banco de dados se esse *MAC* corresponde a uma máquina virtual e qual a chave *RSA* com a qual ela foi registrada no momento de seu disparo no *OpenStack*. Essa chave será utilizada no processo de isolamento, como será descrito posteriormente.

- **Endereço MAC de uma máquina virtual:** dado um endereço IP, pesquisa se ele corresponde a uma máquina virtual e recupera o endereço *MAC* dessa máquina caso ela seja virtual.

- **Endereço MAC do hospedeiro de uma máquina virtual:** dado um endereço IP, pesquisa se ele corresponde a uma máquina virtual e recupera o *MAC* do hospedeiro dessa máquina caso ela seja virtual.

Além das informações recuperáveis pelo banco de dados do *OpenStack*, foi necessário criar um arquivo de configuração para o *DCPortals*. Esse arquivo reúne não só as informações necessárias para fazer a conexão *SQL*, como o usuário e senha do banco de dados, como também informações complementares que não estão presentes no banco de dados. Um exemplo desse tipo de informação é o endereço *MAC* das interfaces de rede das máquinas físicas dedicadas à comunicação das máquinas virtuais. Como descrito anteriormente, é comum haver uma separação entre o tráfego das máquinas virtuais e das interfaces de controle em um *datacenter*. Dessa forma, a interface pela qual o *OpenStack* se comunica com as máquinas hospedeiras nem sempre corresponde à mesma interface que as máquinas hospedeiras dedicam para a comunicação entre máquinas virtuais. Como para o sistema é importante recuperar o endereço *MAC* das interfaces dedicadas para as máquinas virtuais, uma listagem estática desses endereços é necessária. Mais detalhes sobre esse arquivo de configuração estão listados no apêndice B.

3.4 Abstração de rede virtual

A abstração de rede virtual, ou rede lógica privativa, consiste em criar uma representação clara para a separação lógica de diversas redes isoladas compartilhando a mesma infraestrutura, independente de sua organização física. As máquinas conectadas em uma dessas redes podem se comunicar livremente, como se estivessem em uma rede local privada simples conectada por somente um *switch*, embora estejam compartilhando uma rede física complexa, com diversos níveis de *switches*, com outras redes semelhantes. Essa abstração é importante por permitir que um ambiente compartilhado possa ter um isolamento sem qualquer interferência na visão das máquinas, mantendo a segurança de uma rede local e ainda não exigindo qualquer conhecimento da infraestrutura real nas máquinas virtuais.

Essa abstração pode ser alcançada neste sistema configurando cada *switch* virtual para utilizar o controlador *OpenFlow* que estará executando o módulo desenvolvido. O módulo deverá tratar a ocorrência de cada fluxo de pacotes poder instruir os *switches* a encaminhar ou não cada pacote a seu destino, escondendo completamente detalhes da rede virtual das demais. Com isso, para as máquinas virtuais, será como se todas elas estivessem conectadas a um só *switch* em uma rede local simples independente de como elas estiverem organizadas na infraestrutura física.

Essa operação apresenta alguns desafios, que são detalhados com suas soluções a seguir.

3.4.1 Como identificar máquinas de uma mesma rede

Para poder separar as máquinas virtuais de um sistema compartilhado em redes virtuais distintas é necessário criar um identificador que determina a rede virtual à qual uma máquina virtual está conectada. Como descrito anteriormente, no *OpenStack* ao criar uma máquina virtual é atrelada uma chave *RSA* à mesma para prover acesso. Essa chave pode até ser configurada internamente na máquina virtual para permitir login sem senha através de um servidor *SSH*[1]. Inclusive, esse acesso por *SSH* é automaticamente configurado em imagens disponibilizadas oficialmente pelo *OpenStack*.

Como essa chave é cadastrada anteriormente e é necessária para todas as máquinas virtuais disparadas pelo *OpenStack*, é possível utilizá-la como o identificador para determinar a rede virtual. Dessa forma, cada cliente do serviço de computação em nuvem ao alocar sua rede virtual terá uma chave *RSA* cadastrada no sistema para identificar sua rede virtual e a receberá para acessar as máquinas virtuais isoladamente.

[1] *SSH*, ou *Secure Shell* é um protocolo que permite a conexão remota a uma máquina através de um fluxo criptografado.

Em relação ao comportamento do sistema, uma vez interceptado um pacote, é necessário identificar se as máquinas de origem e destino são ou não da mesma rede virtual para permitir a entrega ou descartar do pacote. O *DCPortals* deverá realizar uma pesquisa no banco de dados do *OpenStack* para verificar qual a chave *RSA* atrelada às máquinas de origem e destino e conferir se elas são iguais para permitir ou não que o pacote siga para o destino correto.

Portanto, caso duas máquinas sejam disparadas com a mesma chave *RSA*, elas serão consideradas como conectadas em uma mesma rede virtual e o cliente receberá essa chave para ter acesso a elas enquanto que o sistema usará essa informação internamente para a diferenciação das redes virtuais. Essa opção não só aumenta a segurança como evita a criação de um campo extra no banco de dados.

3.4.2 Como isolar as redes virtuais da rede física

O completo isolamento da rede virtual da rede física possui diversas vantagens, como evitar problemas de segurança e aliviar a utilização dos recursos de rede subjacentes da rede física. Para resolver este problema foi utilizada uma estratégia de re-escrita de endereço *MAC* (*MAC Rewriting*) utilizando o endereço *IP* como identificador.

No nosso ambiente, cada máquina virtual é cadastrada com um *IP* único. Portanto, através do *IP* é possível identificar qual a máquina virtual, qual o seu hospedeiro e a qual rede virtual ela está conectada por causa da visão global da rede provida pelo *DCPortals*. Dessa forma, é possível isolar os endereços *ethernet* das máquina virtuais da rede física re-escrevendo o endereço *MAC* das máquinas virtuais com os endereços das suas respectivas máquinas hospedeiras, mantendo o endereço *IP* da máquina virtual, e desfazendo essa re-escrita quando o pacote chegar na máquina física que hospeda a máquina virtual de destino. Dessa forma, as máquinas virtuais enviam e recebem pacotes de forma transparente, como se estivessem em uma rede local isolada e os *switches* da rede física não tomam conhecimento dos endereços das máquinas virtuais, mantendo somente os endereços das máquinas físicas em suas tabelas e aproveitando melhor os seus recursos.

Essa re-escrita é feita da seguinte forma: quando uma máquina virtual envia um pacote para uma outra máquina virtual da mesma rede virtual, inicialmente o pacote contém o endereço *MAC* das máquinas virtuais de origem e destino. Quando esse pacote é interceptado pelo *switch* virtual, o *DCPortals* programa esse *switch* para que ao enviar o pacote para a porta de saída, re-escreva tanto o endereço de origem quanto de destino com os *MACs* das máquinas físicas hospedeiras, mantendo os endereços *IP* das duas. Quando o pacote chega na máquina física que hospeda a máquina virtual de

destino, o processo contrário é feito. O *DCPortals* utiliza o *IP* da origem e destino e re-escreve novamente os endereços *MAC* com os endereços das máquinas virtuais. Esse processo é simétrico em relação ao tratamento, ou seja, somente com o endereço *IP* e o *MAC* é possível identificar qual endereço deve ser re-escrito, independente do pacote estar saindo ou chegando no *switch*. Esse processo não acontece quando a máquina virtual de destino está mesma máquina hospedeira que a máquina virtual de origem, já que nesse caso o pacote nunca chega à rede física.

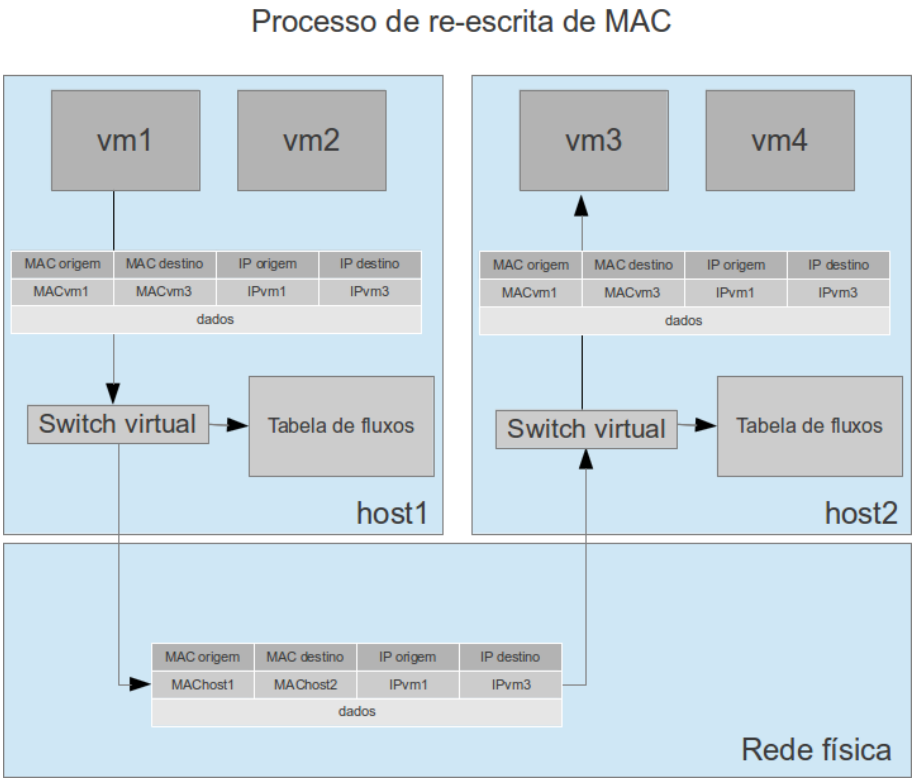

Figura 3.2. Processo de re-escrita do MAC

A figura 3.2 ilustra o processo de re-escrita de *MAC* de um pacote posterior ao primeiro pacote do fluxo, portanto, as regras de re-escrita já estão instaladas nas tabelas de fluxos dos *switches*. No caso, uma máquina virtual *vm1* hospedada em uma máquina física *host1* envia um pacote para a máquina virtual *vm3* hospedada em outra máquina física *host2*. Antes de sair de *host1*, o *switch* virtual consulta sua tabela de fluxos onde a regra de re-escrita está instalada e re-escreve os *MACs* de origem e destino com *MAChost1* e *MAChost2* respectivamente e envia o pacote para a rede física. Ao chegar em *host2*, o mesmo processo é repetido no *switch* virtual e os endereços de origem e destino são revertidos para *MACvm1* e *MACvm3* respectivamente. Os endereços *IP* de origem e destino se mantém *IPvm1* e *IPvm3* durante todo o processo e servem de

identificador para que o *DCPortals* instale os fluxos de re-escrita corretos nas tabelas de fluxos dos *switches* virtuais.

É importante lembrar que esse processo utiliza as informações de *IP* para virtualizar a camada de enlace, portanto, ele só funcionará para pacotes que trabalhem sobre o protocolo *IP*.

3.4.3 Como tratar broadcasts somente para as máquinas de uma rede

Quando um cliente aluga uma rede de máquinas virtuais, ele deseja que pacotes enviados para o endereço de *broadcast* sejam entregues a todas as suas máquinas, e somente a elas. Entretanto, em um ambiente complexo e compartilhado isso não ocorre da forma como deveria, pois pacotes para o endereço de *broadcast* são entregues a todas as máquinas da rede física.

A figura 3.3 mostra a diferença entre o que acontece no ambiente tradicional e com a solução do *DCPortals*. Na figura, as máquinas virtuais estão separadas em redes de acordo com suas cores, a máquina *vm2* envia uma mensagem de *broadcast* que somente deveria ser entregue para outras máquinas da sua mesma rede, as máquinas *vm1*, *vm4* e *vm8*. Sem o *DCPortals*, no modelo convencional, todas as máquinas, indepentende de usuário ou rede virtual, recebem o pacote de *vm2*. Já com a solução proposta, o *broadcast* é entregue apenas às máquinas que fazem parte da mesma rede virtual.

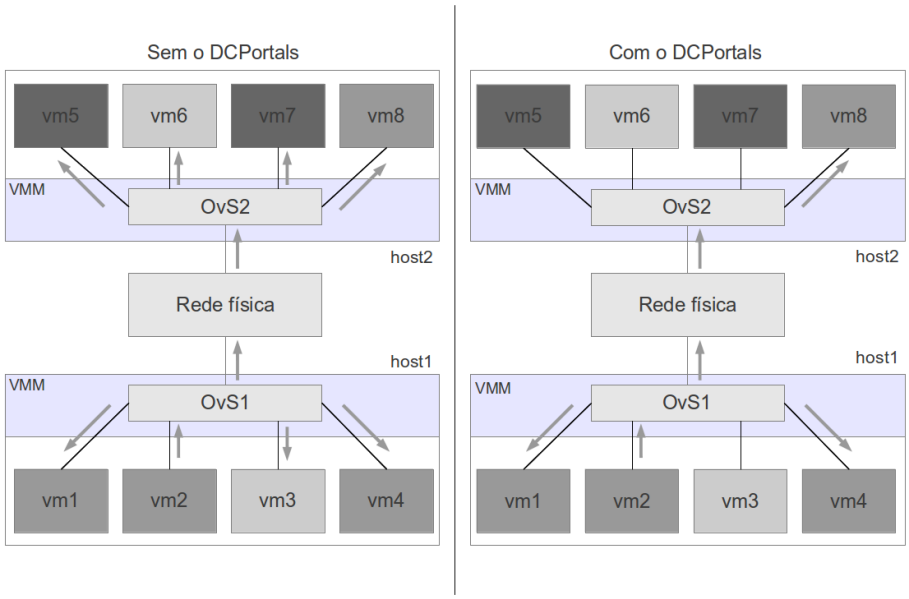

Figura 3.3. Diferença entre envio de pacotes para o endereço de broadcast com e sem o DCPortals

Para realizar esse tratamento, primeiramente é interceptado o pacote de *broadcast* no *switch* virtual da máquina de origem. O controlador pesquisa quais portas desse *switch* estão conectadas a máquinas pertencentes à mesma rede do emissor, além de adicionar a porta que liga o *switch* ao resto da rede física, permitindo que a mensagem de *broadcast* seja disseminada para os outros *switches* virtuais. No caso do exemplo, o controlador instala o fluxo no *switch OvS1* para que ele entregue o pacote nas portas que conectam as máquinas *vm1*, *vm4* e a rede física.

Quando o pacote deixa a máquina física hospedeira da máquina virtual de origem, o processo de re-escrita de *MAC* funciona normalmente para isolar o *MAC* da máquina virtual de origem e mantém o endereço *ethernet* de *broadcast* no destino do pacote. Isso permitirá que os *switches* convencionais encaminhem o pacote para as demais portas.

Ao chegar novamente a um *switch* virtual, o controlador desfaz a re-escrita de *MAC* e pesquisa novamente em quais portas estão conectadas máquinas virtuais da mesma rede virtual do emissor e redireciona o pacote para elas, tomando o cuidado de também não redirecionar o pacote de volta para a rede física nesse caso. No caso do exemplo, o controlador instala o fluxo no *switch OvS2* para entregar o pacote somente na porta ligada à máquina *vm8*.

Dessa forma, a mensagem continua sendo uma mensagem de *broadcast*, porém, ela só será entregue de fato às máquinas da mesma rede virtual do emissor, garantindo que nenhuma outra máquina virtual tenha acesso a esse pacote.

3.4.4 Como interceptar pesquisas ARP

A re-escrita dos endereços *MAC* descrita anteriormente substitui os endereços das máquinas virtuais por endereços das máquinas físicas em um pacote que já contém esses endereços inicialmente. Entretanto, para que as máquinas virtuais montem os pacotes dessa forma, é necessário que elas conheçam o endereço *MAC* da máquina de destino. Para fazer isso ela utiliza o protocolo de resolução de endereços (*Address Resolution Protocol*, ou *ARP*). O *ARP* é um protocolo de resolução de nomes que consiste consiste basicamente de dois tipos de mensagens: o *ARP Request* e o *ARP Reply*. A primeira mensagem é utilizada quando não se sabe o endereço *MAC* de um certo *IP*, portanto é enviada uma mensagem para o endereço de *broadcast* da rede para que o responsável pelo endereço *IP* responda com um *ARP Reply* contendo seu endereço *MAC*.

Para que a separação da rede virtual da rede física que o *DCPortals* oferece ocorra, não é possível que esses pacotes transitem pela rede física com os endereços *MAC* das máquinas virtuais. O *DCPortals* resolve esse problema através da simples

interceptação desses pacotes e a criação de uma resposta com o *MAC* da máquina virtual pesquisada. Dessa forma o pacote nunca chega à rede física e a resposta chega de forma transparente à máquina virtual que realiza a pesquisa.

Esse procedimento ocorre da seguinte forma: assim que um pacote *ARP Request* é interceptado na rede, o *DCPortals* verifica se a máquina que o enviou está na mesma rede virtual que o endereço que ela está pesquisando. Caso as duas máquinas estejam na mesma rede virtual, o próprio *DCPortals* cria um pacote *ARP Reply* encapsulado com o endereço *MAC* da máquina virtual desejada e envia para o *switch* virtual utilizando o protocolo *OpenFlow*. Dessa forma, o *ARP Reply* chega de forma transparente para a máquina emissora e nenhum dos dois passam pela rede física, evitando que qualquer um dos *MACs* de origem ou destino sejam vistos fora das máquinas virtuais.

3.4.5 Como conectar cada rede virtual ao mundo externo

É necessário que cada máquina tenha acesso ao mundo externo nem que seja apenas para que seja possível ao usuário acessar sua própria rede de máquinas ou que elas possam executar alguma aplicação web, por exemplo.

Essa operação depende muito da estrutura de cada *datacenter*, sendo possível, por exemplo, a conexão de uma interface extra de rede em uma das máquinas da rede com um endereço *IP* válido na Internet. Outra solução consiste na criação de *gateways* reservados para cada rede para rotear seu tráfego através de um ponto comum.

Para simplificar, o *DCPortals* elegeu uma máquina, no caso a mesma que executa o controlador *OpenFlow*, para ser um roteador compartilhado de todas as redes virtuais. Todas as máquinas são configuradas para usar esse roteador como o *gateway* de sua rede. Ele, por sua vez, tem acesso a todas as máquinas de qualquer rede virtual e pode estabelecer regras de *firewall* para aplicar regras de redirecionamento utilizando um mecanismo de tradução de endereços (*NAT*) para que as máquinas possuam serviços acessíveis externamente.

3.5 Pseudocódigo do DCPortals

Um trecho em pseudocódigo foi criado a fim de facilitar o entendimento do fluxo de tratamento que um pacote tem ao ser processado pelo *DCPortals* até ser encaminhado de volta para o *switch* virtual

O funcionamento do sistema é representado pelo pseudocódigo a seguir.

```
1  # Funcao que retorna o MAC a ser re-escrito se for necessario re-escreve-lo
```

```
2  def consultaMACRW(MAC,pacote)
3     ...
4
5  # Funcao que envia um ARP Reply com o MAC especificado:
6  def enviaARPReply(MACresposta,pacote)
7     ...
8  # Funcao que instala um fluxo para entregar o pacote na determinada porta
9  def instalaFluxoEntregaPacote(porta,pacote)
10    ...
11
12 # Funcao que entrega um pacote sem instalar um fluxo
13 def entregaPacote(porta,pacote)
14    ...
15
16 # Loop principal do controlador
17 para cada fluxo:
18    portaOrigem[MACvmorigem] = pacote.porta
19
20    # Garante que maquinas de redes diferentes nao se comunicam
21    se (nao mesmaRedeVirtual(MACvmorigem,MACvmdestino)):
22       descartar(pacote)
23       retorna
24
25    # Somente aceita pacotes IP e pacotes ARP Request
26    se pacote.ePacoteIP():
27       # Realiza a re-escrita de MAC se necessario
28       pacote.MACdestino = consultaMACRW(pacote.MACdestino,pacote)
29       pacote.MACorigem = consultaMACRW(pacote.MACorigem,pacote)
30    senao se pacote.ePacoteARPRequest():
31       enviaARPReply(MACvmdestino,pacote)
32    senao:
33       descartar(pacote)
34
35    # Intercepta os broadcasts e entrega somente para portas na mesma rede
36    se pacote.destino.eBroadcast():
37       para cada switch.porta:
38          se switch.porta != pacote.porta:
```

```
39              se mesmaRedeVirtual(switch.porta.MACconectado,pacote.MACorigem):
40                 instalaFluxoEntregaPacote(switch.porta)
41
42    # Entrega os pacotes nao-broadcast
43    senao:
44       # Se nao sabe qual porta, entrega para todas sem instalar o fluxo
45       se MACdestino nao esta em macParaPorta:
46          entregaPacote(todasPortas,pacote)
47
48       # Se sabe a porta, instala fluxo para entregar os pacotes
49       senao:
50          instalaFluxoEntregaPacote(portaOrigem[MACvmdestino],pacote)
```

Das linhas 1-10 são definidas algumas funções mais importantes para o entendimento do algoritmo. As demais funções não foram listadas pois seus nomes são suficientes para entender seu funcionamento. A função *consultaMACRW(MAC,pacote)* realiza uma pesquisa com o *MAC* especificado e retorna o *MAC* que deverá ser utilizado no processo de re-escrita. Ela pesquisa o *MAC* do hospedeiro e da máquina virtual através do *IP* do pacote e determina qual dos dois deve utilizar na situação. A função *enviaARPReply(MACresposta,pacote)* cria um pacote de *ARP Reply*, o encapsula e envia de volta para o emissor do pacote de *ARP Request* utilizando o protocolo *OpenFlow*. Já as funções *entregaPacote(porta,pacote)* e *instalaFluxoEntregaPacote(porta,pacote)* utilizam o protocolo *OpenFlow* para enviar o pacote para a porta especificada com ou sem a criação de uma entrada na tabela de fluxos do *switch* correspondente para que os demais pacotes do fluxo sigam o mesmo destino.

O *loop* principal começa na linha 17. A linha 18 alimenta um dicionário que mapeia endereços *MAC* das máquinas que originaram os pacotes para as portas do *switch* nas quais elas estão instaladas, para simular o comportamento de um *switch ethernet* convencional.

As linhas 21-23 implementam o isolamento das redes virtuais. Os *MACs* de origem e destino são testados com os valores do banco de dados para verificar que ambos pertencem à mesma rede virtual. Caso eles pertençam, o processo continua e em caso contrário, a mensagem é descartada. É importante lembrar que no caso de comunicação com o *gateway* da rede, o retorno da função será positivo e a mensagem não será descartada.

Nas linhas 26-29 é feita a operação de re-escrita de *MAC*. No caso é passado o *MAC* do pacote de origem e a função retorna ou o *MAC* do anfitrião, caso o original

seja de uma máquina virtual, o *MAC* da máquina virtual, caso o original seja de um anfitrião que já foi re-escrito anteriormente, ou o próprio *MAC* passado, caso ele não seja o *MAC* de uma máquina virtual ou um anfitrião. Essa operação é repetida tanto para a origem quanto para o destino, visto que ambos podem precisar ser modificados dependendo da situação da mensagem. Caso o pacote não seja um pacote *IP*, não é possível realizar a re-escrita, portanto ele será descartado caso também não seja um *ARP Reply*, que é tratado posteriormente.

As linhas 30-31 implementam a interceptação dos pacotes de *ARP Request* e enviam o pacote *ARP Reply* correspondente com o *MAC* da máquina virtual correspondente ao *IP* requisitado. Esse processo é feito encapsulando o pacote e enviando-o de volta para a origem do *ARP Request* através do protocolo *OpenFlow*. Caso o pacote não seja nem um pacote *IP* nem um pacote *ARP Request*, ele é descartado na linha 33.

As linhas 36-40 tratam os pacotes de *broadcast*. Nesse caso, tanto se a origem estiver ou não no mesmo *switch*, basta repassar o pacote para todas as outras portas que possuem conectam uma máquina que possui um *MAC* da mesma rede da máquina de origem. O pacote também é repassado para a porta de saída caso o pacote não tenha originado dela, garantindo assim que o mesmo comportamento chegará aos outros *switches* da rede.

Finalmente as linhas 43-50 realizam a entrega do pacote e a instalação do fluxo correspondente nos *switch* programado. Caso a porta do *switch* que deve ser utilizada já seja conhecida, o pacote é entregue e o fluxo é instalado para que os pacotes seguintes sigam o mesmo comportamento. Caso a porta não seja conhecida, o pacote é entregue em todas as portas do *switch* mas nenhum fluxo é instalado, para que posteriormente o processo possa ser repetido e o fluxo instalado caso a porta correta já tenha sido descoberta.

3.6 Exemplo do funcionamento integrado

Portanto, para ilustrar o funcionamento do sistema será descrito a seguir a sequência de acontecimentos que é esperado da comunicação entre duas máquinas virtuais de uma mesma rede virtual passando por todas as ferramentas descritas. As máquinas virtuais são identificadas como **vm1** e **vm2**, seus hospedeiros como **h1** e **h2**, o controlador como **portal** e os *switches* virtuais como **ovs1** e **ovs2**.

1. **vm1** executa um comando de *ping* para verificar se consegue alcançar **vm2**.

2. **vm1** envia um pacote *ARP Request* em *broadcast* para identificar o endereço *MAC* de **vm2** utilizando o IP de **vm2**.

3. **ovs1** intercepta o *ARP Request* e envia uma mensagem a **portal** identificando o novo fluxo e aguardando instruções para lidar com ele.

4. **portal** identifica o *ARP Request*, pesquisa no banco de dados do *OpenStack* o endereço *MAC* do *IP* requisitado e envia a **ovs1** um evento *OpenFlow* com uma mensagem *ARP Reply* encapsulada com o endereço *MAC* de **vm2**, sem que o mesmo chegue à rede física fora de **ovs1**.

5. **ovs1** recebe a resposta de **portal** e encaminha o *ARP Reply* sintetizado para **vm1**.

6. **vm1** utiliza o *MAC* de **h1** para criar o pacote *ICMP* do *ping* e envia o pacote.

7. **ovs1** intercepta o pacote e envia uma mensagem a **portal** para decidir o que fazer com esse fluxo.

8. **portal** verifica se as duas máquinas virtuais que querem se comunicar pertencem à mesma rede virtual através de uma pesquisa ao banco de dados do *OpenStack* pela chave *RSA* utilizada em cada uma das máquinas em seu disparo. Ao verificar que as duas máquinas pertencem à mesma rede, **portal** re-escreve o *MAC* de origem do pacote com o *MAC* de **h1** ao invés do de **vm1**, re-escreve o *MAC* de destino do pacote com o *MAC* de **h2** e indica a porta que **ovs1** deverá enviar o pacote para que ele chegue a **h2**, instalando o fluxo para que esse comportamento se repita para os demais pacotes.

9. **h2** recebe o pacote *ICMP* e **ovs2** o intercepta, enviando um pedido a **portal** sobre o que deverá fazer com ele.

10. **portal** identifica que tanto a origem quanto o destino são da mesma rede virtual e re-escreve o *MAC* de destino com o *MAC* de **vm2** e o *MAC* de origem com o *MAC* de *vm1*, indicando também a **ovs2** qual porta ele deverá encaminhar o pacote para que ele chegue a **vm2**, instalando o fluxo para que esse comportamento se repita para os demais pacotes.

11. **ovs2** recebe a resposta de **portal** e encaminha o pacote *ICMP* a **vm2**.

É importante perceber que nesse processo nenhuma máquina fora dos hospedeiros teve qualquer acesso aos endereços *MAC* das máquinas virtuais através do processo de

re-escrita de endereços *MAC*. Isso alivia a ocupação das tabelas dos *switches* no meio do caminho e garante um isolamento mais eficiente da rede.

3.7 Contribuições para outros projetos

Durante o desenvolvimento do *DCPortals*, as ferramentas *OpenStack*, *Open vSwitch* e o próprio *POX* apresentaram alguns *bugs* principalmente durante o processo de integração das ferramentas. Todos esses *bugs* foram devidamente cadastrados nas respectivas ferramentas de rastreamento oficiais do sistema e sugestões de correções foram discutidas e muitas vezes aceitas como correções para versões futuras.

Espera-se que todas as correções sejam aceitas e que este projeto possa deixar uma contribuição também para as ferramentas que foram utilizadas. Dessa forma, espera-se que este projeto possa ser visto não só como um consumidor de artefatos intelectuais disponíveis através da comunidade de código livre como também um contribuidor para o desenvolvimento dos mesmos.

Algumas sugestões de correção são realizadas localmente no momento, como descrito no roteiro de instalação no apêndice B, porque não foram adotadas a tempo de serem usadas no projeto ou só foram adotadas para versões futuras. Ainda assim, todas foram reportadas e tiveram a solução adotada sugerida.

Capítulo 4

Avaliação

Neste capítulo serão descritos os testes, experimentos e ambientes utilizados assim como serão discutidos seus resultados.

Com base no protótipo implementado, este capítulo descreve os resultados dos experimentos de avaliação realizados.

4.1 Ambiente de testes

Para realizar os testes e experimentos foram utilizadas três máquinas dedicadas somente para esse fim. Elas possuíam duas interfaces de rede, cada uma conectada a um *switch* diferente. O sistema foi configurado com a separação da comunicação de rede em duas interfaces, sendo uma para a comunicação entre o controlador e as máquinas hospedeiras e a outra para comunicação entre as máquinas virtuais. Esta configuração, é bastante comum em *datacenters* comerciais, como já foi comentado anteriormente.

Essa configuração, além de proporcionar que um serviço não interfira no outro, evita uma colisão causada por uma decisão de desenho do *Open vSwitch* em relação à comunicação entre ele e o controlador. Essa decisão consiste na instalação de um fluxo fixo, que tem prioridade maior sobre os instaláveis pelo controlador *OpenFlow*, para permitir o tráfego entre *Open vSwitch* e controlador sem a necessidade de perguntar ao próprio controlador sobre o destino desse fluxo. Embora essa decisão seja óbvia em termos de funcionalidade, se fosse usada com uma só interface, ela ocasionaria a falta de registro de alguns fluxos que deveriam ser capturados, como pacotes *ARP* que chegassem à rede física com os endereços das máquinas virtuais.

A rede de controle usa endereços na faixa 10.0.254.0/24[1]. Foi configurada a

[1]Segundo a definição do *CIDR (Classless Inter-Domain Routing)*, uma faixa 10.0.254.0/24 significa que os primeiros 24 bits do endereço representam a rede enquanto que os demais até o total de 32bits

Máquina Virtual	IP	Rede Virtual	Hospedeiro
vm2	10.0.20.2	rede1	host1
vm3	10.0.20.3		
vm4	10.0.20.4		host2
vm5	10.0.20.5	rede2	
vm6	10.0.20.6		

Tabela 4.1. Distribuição das máquinas virtuais e respectivas redes nas máquinas físicas

faixa de *IP* 10.0.20.0/24 única para distribuir os *IPs* entre as máquinas virtuais nas interfaces de rede dedicadas à comunicação de máquinas virtuais e duas redes virtuais isoladas chamadas de *rede1* e *rede2*. A escolha de uma única faixa de *IP* para todas as máquinas foi feita por ser a configuração mais frágil em termos de isolamento da rede, porque em relação à sub-rede, não existe nenhuma limitação que impediriam as máquinas virtuais de comunicarem entre si. Foram criadas três máquinas virtuais pertencentes à *rede1* e duas pertencentes à *rede2*. Elas foram distribuídas nas máquinas reais de acordo com a configuração descrita na tabela 4.1 e ilustrada na figura 4.1. Essa distribuição inclui os casos de máquinas virtuais de uma mesma rede virtual em uma mesma máquina hospedeira assim como em máquinas hospedeiras diferentes misturadas com máquinas de outras redes virtuais. Essa configuração permitiria que todas as máquinas se comunicassem, caso não existisse o serviço de isolamento, já que haveriam rotas para todas e elas utilizam a mesma faixa de *IPs*.

As máquinas físicas foram configuradas utilizando o sistema operacional *Ubuntu* versão 11.10 e os pacotes do *OpenStack*, *Xen* e *Open vSwitch* dos repositórios oficiais. Já as máquinas virtuais foram instaladas utilizando uma imagem padrão do *OpenStack* com o sistema *Ubuntu* na versão 10.10. Mais detalhes sobre a configuração desse sistema estão presentes no apêndice B.

Sobre esse ambiente montado foram realizados quatro tipos de testes e experimentos. O primeiro consistiu em um teste de isolamento simples utilizando o comando *ping*, que é uma ferramenta que envia um pacote pela rede e recebe uma resposta para comprovar o alcance e o tempo de ida e volta utilizando o protocolo *ICMP* (IETF, 1981), tendo como destino o endereço de *broadcast* da rede. O segundo avaliou a interferência do sistema no atraso da comunicação entre as máquinas utilizando a mesma ferramenta *ping* e registrando o tempo de resposta dos pacotes. O terceiro simulou um ataque comum de negação de serviço, que consistiu em uma máquina enviando muitos pacotes para inviabilizar a comunicação de máquinas de outro cliente na mesma rede

representam uma máquina daquela rede. Portanto, 10.0.254.0/24 significa que todos os *IPs* entre 10.0.254.1 e 10.0.254.254 estão na mesma rede.

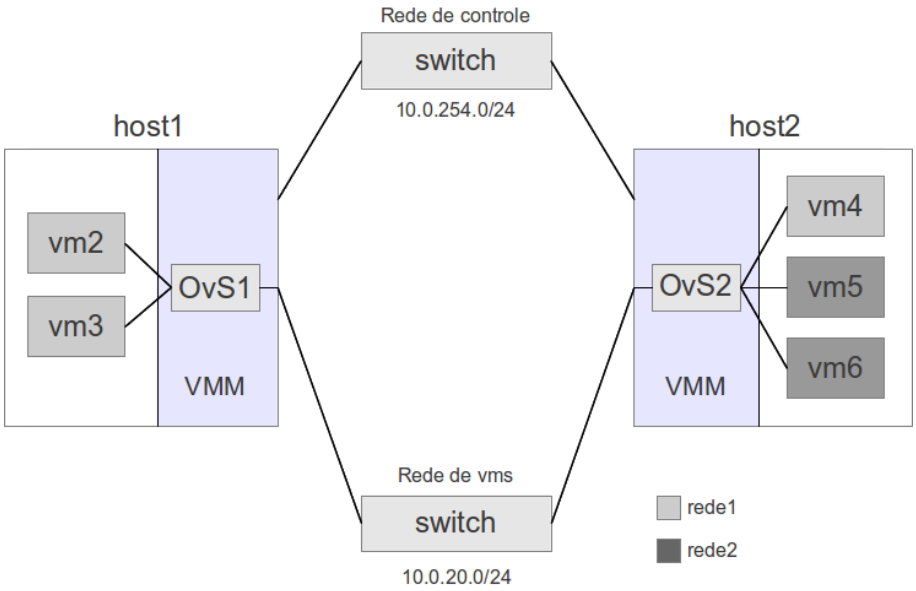

Figura 4.1. Distribuição das máquinas virtuais nas máquinas físicas. As cores separam as redes virtuais das máquinas virtuais.

física, para mostrar os ganhos do isolamento do *DCPortals* em um ataque desse tipo. Por fim, o quarto experimento quantificou os custos do ambiente baseado em Redes Definidas por Software no momento do estabelecimento dos fluxos.

4.2 Comprovação de isolamento

Esse experimento teve como objetivo comprovar o funcionamento correto do isolamento das redes virtuais e garantia de comunicação entre máquinas de uma mesma rede virtual.

O experimento consistiu em utilizar a ferramenta *ping* para enviar um pacote pedindo uma resposta para o endereço de *broadcast* da rede a partir de uma máquina pertencente a cada rede virtual. O comportamento esperado com esse uso da ferramenta *ping* seria que todas as máquinas que estivessem presentes na mesma rede respondessem à máquina emissora com um pacote de resposta à requisição inicial. No caso do *DCPortals*, mesmo com as máquinas usando uma mesma faixa de endereços de rede, com o mesmo endereço de *broadcast*, e compartilhando a mesma infraestrutura física, apenas máquinas virtuais da mesma rede virtual deveriam receber o pedido e, consequentemente, respondê-lo.

Uma preparação necessária a esse experimento é a verificação se todas as máquinas envolvidas estão configuradas para responder mensagens do comando *ping* para

endereços de *broadcast*. Os sistemas atuais, por padrão, preferem desabilitar essa funcionalidade para evitar congestionamento da rede e evitar tentativas de ataques de negação de serviço utilizando o protocolo *ICMP*. Para habilitar essa funcionalidade no sistema utilizado nas máquinas virtuais do ambiente de testes bastou modificar o valor da variável do *kernel* do *Linux* *net.ipv4.icmp_echo_ignore_broadcasts* para 0.

Para comprovar a funcionalidade esperada foi executado um comando *ping* para o endereço de *broadcast* da rede configurada para as máquinas virtuais (10.0.20.255) partindo da máquina virtual *vm2*, pertencente à rede *rede1*, e da máquina virtual *vm5*, pertencente à rede *rede2*, como descrito pela tabela 4.1. A seguir são relatados os resultados com e sem o controle do *DCPortals*:

Sem o *DCPortals*:

```
vm2:

PING 10.0.20.255 (10.0.20.255) 56(84) bytes of data.
64 bytes from 10.0.20.2: icmp_req=1 ttl=64 time=0.027 ms
64 bytes from 10.0.20.4: icmp_req=1 ttl=64 time=2.60 ms (DUP!)
64 bytes from 10.0.20.3: icmp_req=1 ttl=64 time=3.21 ms (DUP!)
64 bytes from 10.0.20.6: icmp_req=1 ttl=64 time=3.22 ms (DUP!)
64 bytes from 10.0.20.5: icmp_req=1 ttl=64 time=8.47 ms (DUP!)


vm5:

PING 10.0.20.255 (10.0.20.255) 56(84) bytes of data.
64 bytes from 10.0.20.5: icmp_req=1 ttl=64 time=0.027 ms
64 bytes from 10.0.20.6: icmp_req=1 ttl=64 time=1.00 ms (DUP!)
64 bytes from 10.0.20.3: icmp_req=1 ttl=64 time=1.86 ms (DUP!)
64 bytes from 10.0.20.2: icmp_req=1 ttl=64 time=5.91 ms (DUP!)
64 bytes from 10.0.20.4: icmp_req=1 ttl=64 time=10.5 ms (DUP!)
```

No ambiente sem o isolamento do *DCPortals* todas as máquinas virtuais se alcançam e podem se comunicar, independente da sua organização dentro do *DCPortals*. Por isso que tanto o *ping* proveniente da *vm2* quanto o da *vm5* apresentaram respostas de todas as 5 máquinas virtuais levantadas no ambiente. Todas as máquinas receberam o pacote de requisição para o endereço de *broadcast* e enviaram a resposta diretamente à máquina emissora sem nenhuma separação lógica por estarem utilizando a mesma infraestrutura física.

Com o *DCPortals*:

```
vm2:

PING 10.0.20.255 (10.0.20.255) 56(84) bytes of data.
64 bytes from 10.0.20.2: icmp_req=1 ttl=64 time=0.026 ms
64 bytes from 10.0.20.3: icmp_req=1 ttl=64 time=66.7 ms (DUP!)
64 bytes from 10.0.20.4: icmp_req=1 ttl=64 time=88.5 ms (DUP!)

vm5:

PING 10.0.20.255 (10.0.20.255) 56(84) bytes of data.
64 bytes from 10.0.20.5: icmp_req=1 ttl=64 time=0.032 ms
64 bytes from 10.0.20.6: icmp_req=1 ttl=64 time=103 ms (DUP!)
```

Já no ambiente com o *DCPortals*, o *ping* da máquina virtual *vm2* apenas recebeu respostas de outras máquinas cadastradas na rede virtual *rede1*, independentemente de qual máquina física elas estavam hospedadas, e o mesmo aconteceu com as máquinas da rede virtual *rede2*. Isso mostra que apesar de dividir a mesma infraestrutura física, o mesmo endereço de *broadcast* e as mesmas máquinas hospedeiras, as redes virtuais estão totalmente isoladas entre si. O mesmo é comprovado ao analisar um registro de todos os pacotes que passam pela rede nas máquinas físicas e nas máquinas virtuais.

Em relação ao endereço *ethernet* e o processo de *MAC Rewriting*, foi feita a captura dos pacotes do *ping* dentro e fora das máquinas virtuais. Para exemplificar esse caso serão mostrados os pacotes de *ICMP Request* da máquina *vm2* que está hospedada na máquina *host1* e o *ICMP Reply* correspondente da máquina *vm4*, que está hospedada na máquina *host2*. A tabela 4.2 lista os endereços *ethernet* de todas as máquinas envolvidas. A figura 4.2 mostra a listagem dos cabeçalhos das mensagens capturadas dentro da máquina virtual *vm2* enquanto que a figura 4.3 mostra a mesma listagem capturada na máquina hospedeira, ou seja, o que de fato foi propagado pela rede física.

É possível ver nos campos realçados das figuras 4.2 e 4.3 que quando o pacote ainda está na máquina virtual e quando a resposta chega, os endereços *ethernet* registrados são os mesmos das máquinas virtuais. Já quando os pacotes deixam a máquina virtual, os endereços registrados correspondem somente às máquinas hospedeiras independente de qual máquina virtual se trata.

Máquina	Endereço ethernet
vm2	02:16:3e:30:ce:4b
vm3	02:16:3e:35:37:85
vm4	02:16:3e:17:91:9d
vm5	02:16:3e:2e:e3:e0
vm6	02:16:3e:3e:eb:10
host1	f4:ec:38:80:69:86
host2	54:e6:fc:81:9e:ce

Tabela 4.2. Endereço ethernet de cada uma das máquinas virtuais ou físicas

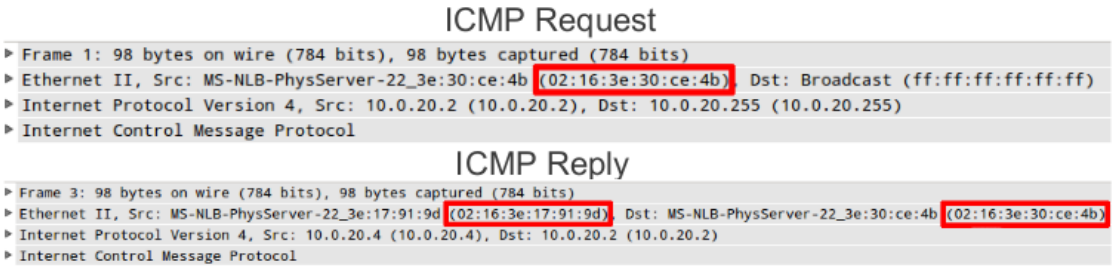

Figura 4.2. Pacotes de ICMP Request e Reply entre vm2 e vm4 capturados dentro da máquina virtual vm2

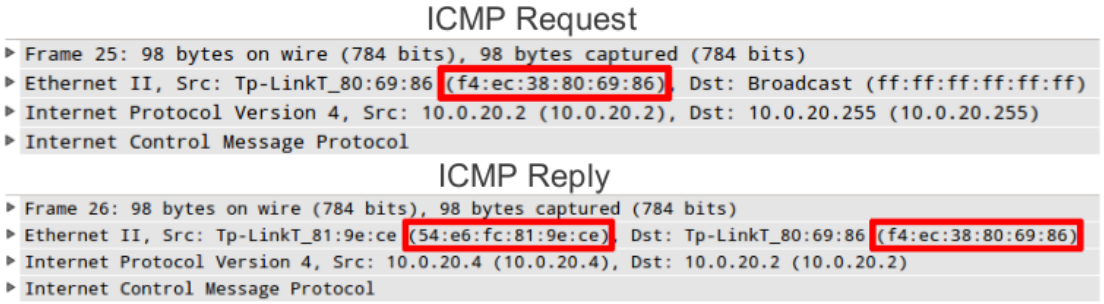

Figura 4.3. Pacotes de ICMP Request e Reply entre vm2 e vm4 capturados fora da máquina virtual vm2

4.3 Verificação de latência

A separação dos planos de controle e dados realizado pela *SDN* através da programação dos *switches* modifica consideravelmente a forma como a comunicação da rede funciona. Portanto, é interessante realizar um experimento para quantificar o impacto que o *DCPortals* tem sobre a latência de um fluxo já estabelecido.

Para este experimento foi utilizada novamente a ferramenta *ping*, porém, dessa vez para calcular o tempo entre a requisição e a resposta. Foram feitas comparações

com o sistema convencional sem *SDN* e com uma implementação de *switch ethernet* comum sobre um controlador *OpenFlow* para identificar se existe alguma interferência considerável do controlador *DCPortals* sobre a latência da comunicação das máquinas virtuais.

Como o foco desse experimento é apenas a latência dos pacotes trocados pelo *ping*, serão instaladas entradas estáticas na tabela *ARP* para que não seja necessário fazer o mapeamento dos endereços IP para endereços *ethernet*. Com isso podemos ter certeza que todo atraso registrado é devido somente ao tráfego dos pacotes do *ping*.

Foram avaliados *pings* entre máquinas virtuais hospedadas em máquinas diferentes e na mesma máquina. Essas duas configurações foram avaliadas em um sistema apenas com *switches ethernet* comuns, um sistema com a implementação de *switch L2 ethernet* convencional encontrado no código do *POX* e com o *DCPortals*. As figuras 4.4 e 4.5 demonstram como esses experimentos são feitos no ambiente de teste em uma mesma máquina física e em diferentes máquinas físicas respectivamente.

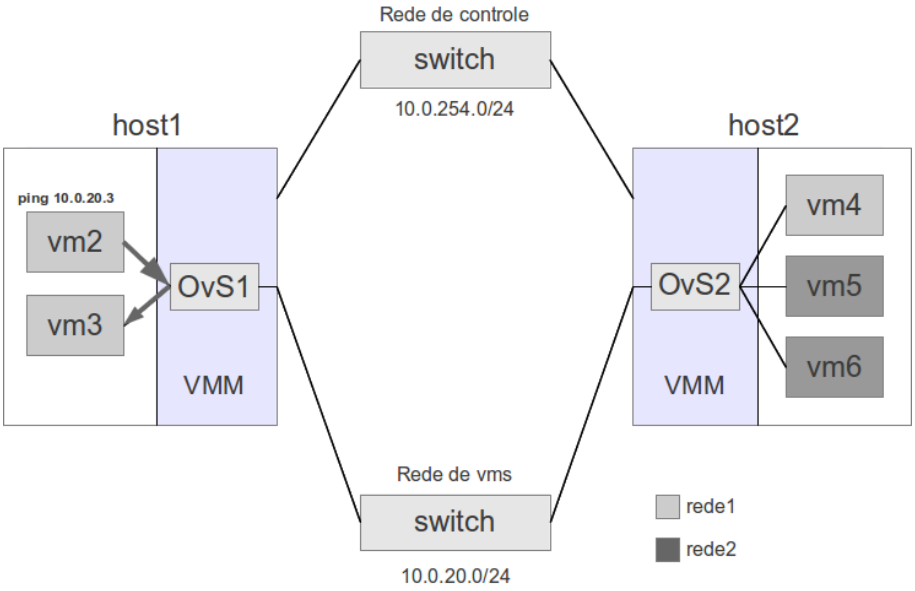

Figura 4.4. Esquema do experimento de latência entre duas máquinas virtuais hospedadas na mesma máquina física. A máquina vm2 realiza um ping para a máquina vm3.

Os resultados estão descritos nas tabelas 4.3 e 4.4, a primeira com o resultado entre duas máquinas virtuais hospedadas em uma mesma máquina hospedeira enquanto que a segunda entre duas máquinas virtuais hospedadas em diferentes máquinas físicas. Cada *ping* foi repetido 10.000 vezes e os resultados foram calculados com um intervalo de confiança de 99%. Foi desconsiderado o primeiro pacote de cada experimento devido

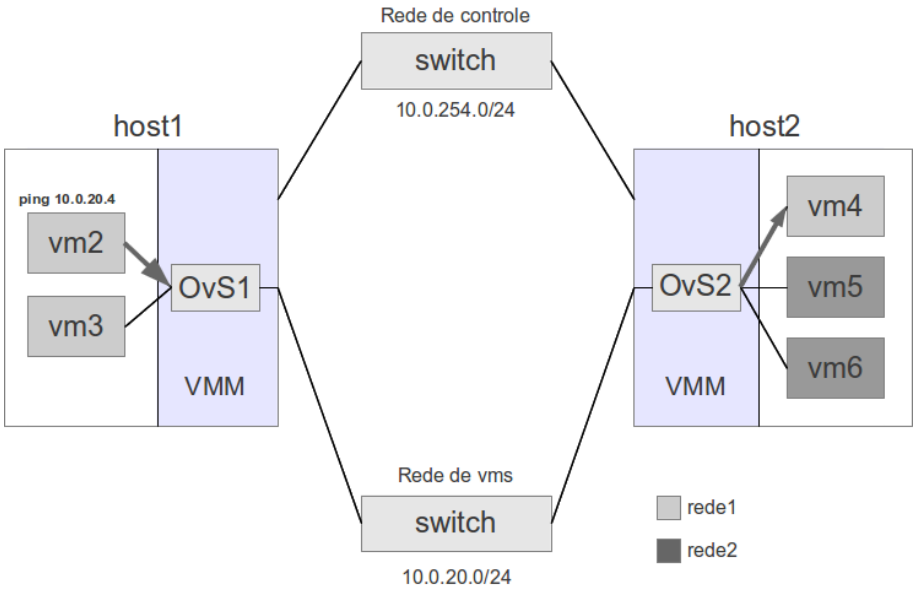

Figura 4.5. Esquema do experimento de latência entre duas máquinas virtuais hospedadas em diferentes máquinas físicas. A máquina vm2 realiza um ping para a máquina vm4.

ao atraso esperado pela consulta ao controlador e instalação do fluxo já que o objetivo nesse caso era avaliar o impacto para um fluxo já estabelecido. A análise dos custos de estabelecimento dos fluxos foi feita na seção 4.5.

Ambiente	Média (μs)	$\pm$ erro (μs)
A	148,98	0,38
B	149,26	0,38
C	150,23	0,36

Tabela 4.3. Latências médias registradas com intervalo de confiança de 99% entre máquinas virtuais hospedadas na mesma máquina física. Os ambientes são: A: com switches ethernet comuns, B: POX com switch L2 comum e C: *DCPortals*.

Ambiente	Média (μs)	$\pm$ erro (μs)
A	196,58	0,54
B	196,6	0,41
C	198,17	0,48

Tabela 4.4. Latências médias registradas com intervalo de confiança de 99% entre máquinas virtuais hospedadas em máquinas físicas diferentes. Os ambientes são: A: com switches ethernet comuns, B: POX com switch L2 comum e C: *DCPortals*.

Também foi feito um levantamento estatístico da diferença entre os 3 sistemas pareados para visualizar a diferença entre eles. A finalidade desse levantamento é quantizar quanto um sistema é mais lento que o outro, avaliando o intervalo estatístico da diferença entre eles. Os dados utilizados foram os mesmos para os levantamentos anteriores; a diferença foi estimada também com um intervalo de confiança de 99%. Os resultados foram expressos nas tabelas 4.5 e 4.6, a primeira com o resultado entre duas máquinas virtuais hospedadas em uma mesma máquina física, e a segunda entre duas máquinas virtuais hospedadas em máquinas físicas diferentes.

É importante salientar que apesar da diferença do *DCPortals* para os ambientes sem *SDN* e o *switch L2* comum do *POX* ser diferente de zero, é um valor muito pequeno, da ordem de microssegundos, o que em uma rede local é pouco em comparação aos valores médios, que são da ordem de centenas de microssegundos, como é possível reparar nos valores dos sistemas isolados. Fazendo a comparação do *switch L2* comum do *POX* para o ambiente sem *SDN* não há diferença estatística porque o intervalo de confiança contém zero. É possível explicar essa diferença mínima do *DCPortals* para os demais ambientes por conta das operações de re-escrita do endereço de *ethernet* dos pacotes, que é uma operação que não é realizada nos demais. Essa diferença se mantém semelhante estatisticamente quando as duas máquinas virtuais estão hospedadas na mesma máquina física ou hospedadas em máquinas físicas diferentes, já que em um experimento pareado o intervalo de confiança tem sobreposições.

Ambiente	Média (μs)	$\pm$ erro (μs)
C − A	1,25	0,53
C − B	0,97	0,52
B − A	0,28	0,55

Tabela 4.5. Diferenças entre latências médias registradas com intervalo de confiança de 99% entre máquinas virtuais hospedadas na mesma máquina física. Os ambientes são: A: com switches ethernet comuns, B: POX com switch L2 comum e C: *DCPortals*.

Ambiente	Média (μs)	$\pm$ erro (μs)
C − A	1,59	0,73
C − B	1,57	0,64
B − A	0,02	0,64

Tabela 4.6. Diferenças entre latências médias registradas com intervalo de confiança de 99% entre máquinas virtuais hospedadas em máquinas físicas diferentes. Os ambientes são: A: Sem SDN, B: POX com switch L2 comum e C: *DCPortals*.

Pode-se dizer que essa diferença desprezível significa que o *DCPortals* não compromete o funcionamento de aplicações dependentes de latência em um sistema semelhante ao presente nos experimentos uma vez que os fluxos são estabelecidos.

4.4 Ataque de negação de serviço

Uma preocupação bastante comum com serviços ofertados pela rede é a ameaça de queda do serviço por um ataque de negação de serviço (*Denial of Service* ou *DoS*). Esse tipo de ataque consiste em maliciosamente criar tráfego exagerado para o provedor de um serviço até que o servidor fique sobrecarregado e comece a negar requisições válidas dos clientes. Esse tipo de ataque pode ser realizado de várias formas, criando diversas conexões simultâneas, estourando o limite da banda, utilizando um ou mais focos de ataque, dentre outras formas.

No ambiente de um *datacenter* de computação em nuvem onde diversos clientes dividem uma mesma infraestrutura existe uma forma muito simples de fazer um ataque de negação de serviço. Qualquer cliente pode criar um fluxo de dados muito grande utilizando o protocolo de datagramas *UDP* e enviar esse fluxo para o endereço de *broadcast* da rede. Assim, o atacante pode esgotar a banda para cada um dos demais clientes e impossibilitar o uso de serviços que dependam da rede. É possível até que ocorram casos como esse sem a intenção de gerar um ataque, já que um dos clientes pode ter um sistema defeituoso que pode enviar incorretamente muito tráfego para um endereço incorreto.

Existem casos relatados de ataques semelhantes que aconteceram no serviço do *Amazon EC2*, como o caso do serviço de hospedagem *BitBucket*, que é relatado no pronunciamento BitBucket Attack (2012). Basicamente, no período em que o *BitBucket* estava utilizando os serviços do *Amazon EC2*, ele sofreu um ataque de negação de serviço utilizando tráfego *UDP* para esgotar sua banda. A *Amazon* precisou intervir diretamente através do suporte para bloquear esse tráfego indesejado, porém, no final do processo a *BitBucket* decidiu abandonar o *Amazon EC2* e partir para outro serviço de computação em nuvem para hospedar seu sistema.

Esse tipo de problema não aconteceria se o tráfego entre clientes fosse isolado, como é a proposta do *DCPortals*. Portanto, a ideia desse experimento é mostrar a interferência que um ataque simples de tráfego excessivo de pacotes *UDP* realiza no desempenho de uma conexão *TCP* e como o isolamento do *DCPortals* atenua esse problema quase completamente.

Para gerar e monitorar o tráfego foi utilizada uma versão modificada da ferra-

menta *iperf* para permitir o tráfego *UDP* para um endereço de *broadcast*. O *iperf* é uma ferramenta bastante utilizada para gerar tráfego e monitorar a velocidade de transmissão entre dois pontos da rede, ele cria um tráfego e mostra as taxas a cada intervalo de tempo informado. Foram criados dois fluxos de pacotes utilizando o *iperf*, sendo que um deles era responsável pelo ataque e o outro por uma transferência *TCP* comum entre duas máquinas virtuais. A taxa de transferência de cada máquina foi limitada para 1 Gbps em cada máquina virtual, uma configuração comum, já que este é o limite de banda da rede.

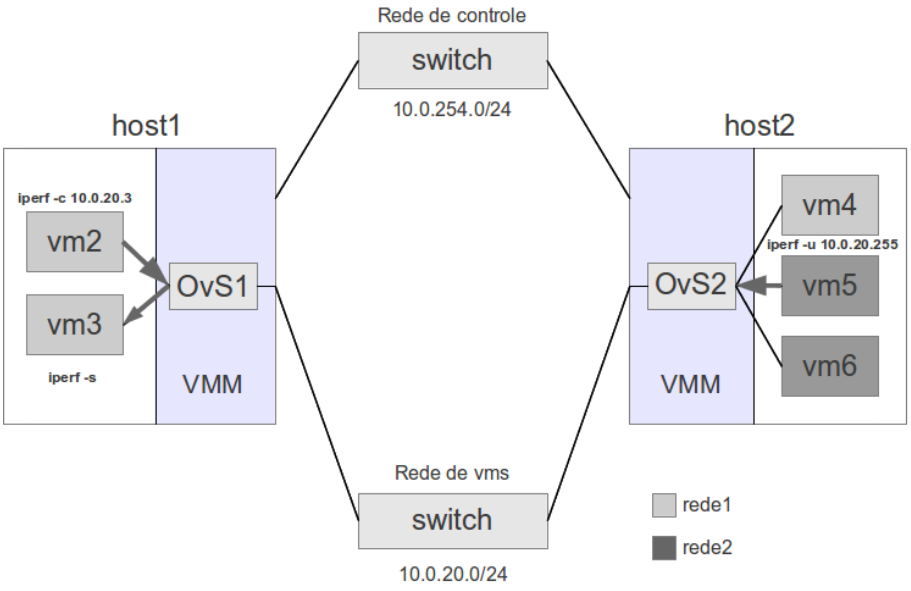

Figura 4.6. Esquema do experimento de ataque de negação de serviço. A máquina vm5 realiza um flood UDP para o endereço de broadcast da rede enquanto que as máquinas vm2 e vm3 realizam um fluxo TCP.

A figura 4.6 mostra o experimento realizado no ambiente de testes. As máquinas virtuais *vm2* e *vm3*, localizadas na máquina física *host1*, realizaram uma transferência *TCP* comum entre elas e registram a largura da banda durante essa transferência. Enquanto isso, a máquina virtual *vm5*, hospedada na máquina física *host2* envia o máximo de tráfego possível *UDP* para o endereço de *broadcast* da rede. Como *UDP* não possui controle de congestionamento, o fluxo continuará máximo mesmo ao esgotar a capacidade da interface e não receber nenhuma confirmação de recebimento. Como o tráfego é enviado para o endereço de *broadcast* comum da rede, no caso da falta de isolamento, todas as máquinas virtuais na rede devem receber esses pacotes e esgotar sua banda, o que ativaria o algoritmo de congestionamento do *TCP* e diminuiria a velocidade de transmissão do fluxo *TCP*. Além do experimento do ataque foi realizado

Ambiente	Média (Mbps)	± erro (Mbps)
A	928,29	0,16
B	928,21	0,22
C	41,21	12,99
D	910,11	0,28

Tabela 4.7. Largura de banda dos fluxos TCP registrados nos diferentes ambientes utilizando um intervalo de confiança de 99%. Os ambientes são: A: ambiente sem isolamento e sem ataque, B: *DCPortals* sem ataque, C: ambiente sem isolamento e D: *DCPortals*.

Ambiente	Média (Mbps)	± erro (Mbps)
D − C	868,91	12,98
B − A	-0,08	0,18

Tabela 4.8. Diferença das larguras de banda dos fluxos TCP registrados nos diferentes ambientes utilizando um intervalo de confiança de 99%. Os ambientes são: A: ambiente sem isolamento e sem ataque, B: *DCPortals* sem ataque, C: ambiente sem isolamento e D: *DCPortals*

um experimento de controle com o tráfego *TCP* sem a interferência do tráfego *UDP* para comparação. Todas as transferências duraram 1.000 segundos cada uma e tiveram a banda medida a cada 3 segundos pelo *iperf*. A banda dos primeiros 3 segundos foi descartada por ser afetada pelo mecanismo de estabelecimento da conexão *TCP*.

Os resultados estão apresentados na tabela 4.7. Além deles também foi gerada uma comparação estatística da diferença entre as bandas no ambiente sem isolamento e no ambiente com o *DCPortals*. Essa diferença está expressa na tabela 4.8.

Observando as comparações entre o ambiente sem ataques já é possível perceber a diferença dos dois casos. Enquanto o sistema sem isolamento sofre uma perda de cerca de 95% na banda observada, o *DCPortals* sofre apenas cerca de 5% de banda. Com o isolamento das redes, apenas máquinas da mesma rede virtual do mesmo cliente receberão os pacotes desse tipo de ataque, por isso que a interferência não acontece em máquinas de outros clientes. A pequena queda no serviço dos outros clientes se dá pela utilização da infraestrutura local, porém, nesse tipo de serviço as conexões são limitadas assim como foi feito nesse ambiente para evitar exatamente o caso de que o tráfego de um cliente prejudique o tráfego dos demais. Portanto, é esperado que a interferência continue pequena mesmo em um ambiente real.

Em relação às comparações das diferenças entre os sistemas, primeiro é importante comentar que não existe diferença estatística entre o *DCPortals* e o sistema sem isolamento quando não existe a ocorrência de um ataque, já que o intervalo de confiança de 99% da diferença dos dois contém zero. Isso demonstra que o *DCPortals*

também não apresenta nenhuma deficiência de largura de banda em relação ao sistema sem isolamento. Além disso, a comparação dos dois ao sofrer o ataque demonstra uma diferença de banda a favor do *DCPortals* de cerca de 93% da banda observada sem a ocorrência do ataque. Essa diferença é extremamente relevante e demonstra que o *DCPortals* apresenta um nível de segurança bastante superior a ataques semelhantes.

Também é interessante comentar que para esse teste foi encontrada uma limitação em sistemas reativos em ambientes de Redes Definidas por Software que determinou uma alteração manual no funcionamento do sistema para realizar o experimento. Ao iniciar o ataque *UDP*, como não há controle de congestionamento, diversos pacotes chegam no *switch* e são encaminhados constantemente ao controlador antes que a resposta ao primeiro pacote do fluxo chegue ao *switch*. Esse comportamento persiste até que alguma memória seja estourada ou algum limite seja alcançado. O modelo proativo sugerido no final deste capítulo elimina esse problema ao instalar o fluxo antecipadamente e evitando que os pacotes cheguem quando o fluxo é iniciado. Para resolver esse problema no protótipo implementado foi antecipada a resposta do controlador e instalada manualmente a entrada na tabela de fluxos dos *switches* relativa ao tratamento esperado para o fluxo *UDP*.

4.5 Custos do ambiente de Redes Definidas por Software

Nos experimentos anteriores sempre foram consideradas as comunicações do fluxo de pacotes eliminando a configuração inicial do sistema necessária em um ambiente de Redes Definidas por Software. Como descrito anteriormente, quando um novo fluxo é identificado em um ambiente de Redes Definidas por Software, o controlador é consultado e uma entrada é instalada na tabela de fluxos de um *switch* para programá-lo. Além disso, no caso do *DCPortals* é feito um tratamento especial para pacotes *ARP*, interceptando pacotes desse tipo e criando respostas sintetizadas para garantir o isolamento que o sistema oferece.

Para medir os custos relacionados a esse ambiente foi feito um experimento que compara o tempo de ida e volta dos primeiros pacotes de um fluxo utilizando a ferramenta *ping*. Foram tratadas duas situações: uma situação em que somente um pacote *ARP* é sintetizado e outra em que somente as entradas nas tabelas de fluxos de um *switch* são instaladas. Essas duas situações foram medidas em um ambiente sem o uso de *SDNs*, utilizando o *switch L2* comum do *POX* e com o *DCPortals*.

Para medir o custo de sintetização do pacote *ARP* foram fixadas entradas nas

Ambiente	Média (ms)	± erro (ms)
A	7,45	0,1
B	10,72	2,67
C	15,31	3,57

Tabela 4.9. Atrasos do *ARP* nos ambientes utilizando um intervalo de confiança de 99%. Os ambientes são: A: ambiente sem *SDN*, B: *switch L2* comum do *POX* e C: *DCPortals*.

Ambiente	Média (ms)	± erro (ms)
C − A	7,85	3,57
C − B	4,581	4,99
B − A	3,27	2,63

Tabela 4.10. Diferenças nos atrasos do *ARP* nos ambientes utilizando um intervalo de confiança de 99%. Os ambientes são: A: ambiente sem *SDN*, B: *switch L2* comum do *POX* e C: *DCPortals*.

tabelas de fluxo do *switch* virtual de uma máquina hospedeira para tratar os fluxos de pacote que seriam gerados pelos pacotes do *ping* utilizado para o experimento. Também foi fixada uma entrada na tabela *ARP* de uma das máquinas envolvidas na comunicação para que o atraso extra gerado seja exclusivo do tempo de criação de somente uma resposta *ARP*. Para forçar a criação desse pacote *ARP*, a tabela *ARP* foi esvaziada manualmente antes da execução do comando *ping*.

Esse processo foi repetido 30 vezes em cada ambiente e os resultados estão apresentados na tabela 4.9. Também foi gerada a diferença estatística entre os atrasos dos ambientes na tabela 4.10. Ambos os resultados foram gerados considerando um intervalo de confiança de 99%.

Observando os tempos de atrasos é possível perceber que todos os ambientes possuem um atraso considerável relacionado ao pacote *ARP*, já que no experimento de latência discutido anteriormente os valores são bem menores. O *POX*, mesmo sem a técnica de sintetização de pacotes *ARP* do *DCPortals*, já adiciona um atraso extra considerável em comparação ao sistema sem *SDN*. Isso acontece porque, apesar de não ter que instalar nenhum fluxo relacionado ao *ping*, ainda são necessários fluxos relacionados aos pacotes *ARP*, que fluem livremente pela rede e por isso precisam de tratamento em cada fase da comunicação. Já o *DCPortals* não possui esse tempo de instalação de fluxos porque intercepta a comunicação *ARP* no início e impede que os pacotes passem pela rede. Entretanto, o tratamento para a sintetização da resposta que é enviada ao emissor da requisição *ARP* é considerável e também adiciona um atraso em comparação ao sistema sem *SDN*. O valor do atraso do *DCPortals* não pode

Ambiente	Média (ms)	± erro (ms)
A	0,20	0,03
B	29,83	8,36
C	47,05	8,96

Tabela 4.11. Atrasos da criação de entradas nas tabelas de fluxos nos ambientes utilizando um intervalo de confiança de 99%. Os ambientes são: A: ambiente sem *SDN*, B: *switch L2* comum do *POX* e C: *DCPortals*.

Ambiente	Média (ms)	± erro (ms)
C − A	46,85	8,96
C − B	17,22	11,91
B − A	29,63	8,36

Tabela 4.12. Diferenças nos atrasos da criação de entradas nas tabelas de fluxos nos ambientes utilizando um intervalo de confiança de 99%. Os ambientes são: A: ambiente sem *SDN*, B: *switch L2* comum do *POX* e C: *DCPortals*.

ser considerado maior que o atraso do *switch L2* do *POX*, uma vez que o intervalo da diferença entre os experimentos pareados de ambos contém zero.

Para medir o custo da instalação de fluxos foram fixadas as entradas na tabela *ARP* de duas máquinas virtuais hospedadas em uma mesma máquina fixa e foram executados *pings* com um intervalo grande o suficiente para que as entradas da tabela de fluxos correspondentes ao *ping* anterior tenham seu tempo de expiração alcançado. Dessa forma, nenhum pacote *ARP* será emitido durante os experimentos e cada *ping* resultará na criação de duas entradas na tabela de fluxos do *switch* virtual da máquina hospedeira, uma para a requisição e outra para a resposta.

Esse processo foi repetido 30 vezes em cada ambiente e os resultados estão apresentados na tabela 4.11. Também foi gerada a diferença estatística entre os atrasos dos ambientes na tabela 4.12. Ambos os resultados foram gerados considerando um intervalo de confiança de 99%.

O atraso relacionado à instalação de entradas nas tabelas de fluxos é bem mais considerável. Isso ocorre por causa da necessidade da consulta ao controlador *OpenFlow*, que soma o tempo de comunicação do protocolo e o tempo de processamento interno do controlador. O ambiente sem *SDN* não possui nenhuma interferência, somente a comunicação do *ping*, por isso possui um resultado comparável aos observados no experimento de latência discutido anteriormente. Na comparação dos ambientes de redes definidas por software, o *DCPortals* apresenta um atraso comprovadamente maior que o *switch L2* comum do *POX*. Isso se deve à necessidade de pesquisa ao banco de dados do *OpenStack* para garantir o isolamento da rede, o que não é feito no *switch*

L2 comum do *POX*, por ele somente emular o comportamento de um *switch ethernet* convencional com aprendizado.

Os atrasos observados nos ambientes de redes definidos por software são garantidamente superiores a um ambiente sem *SDN*. Porém, não são atrasos muito relevantes para a comunicação principalmente porque acontecem somente uma vez, no caso da instalação de entradas nas tabelas de fluxos, ou somente quando a tabela *ARP* não contém uma entrada necessária para a comunicação do fluxo. Também não são atrasos de uma magnitude elevada, já que variam na casa das dezenas de milissegundos, enquanto o atraso comum do processo de pesquisa *ARP* em ambientes sem *SDN* já apresentam um atraso de cerca de sete milissegundos.

Mesmo esse atraso não tendo um impacto significativo, é possível alterar a arquitetura do *DCPortals* para evitá-lo. Apesar de ter-se optado neste trabalho por não implementar essa alteração, ela é registrada aqui.

4.5.1 Modelo proativo

Como implementado, *DCPortals* apresenta uma abordagem reativa para o controlador, ou seja, a programação dos *switches* de borda é feita no momento em que um fluxo é observado entre duas VMs de um mesmo usuário. Como visto, essa forma de operação implica em um atraso mais elevado para o primeiro pacote trocado entre cada para de VMs. Como a localização de VMs no sistema é controlado pelo *OpenStack*, é possível prever os fluxos que devem ser observados no sistema, assumindo-se que em algum momento cada par de máquinas de um cliente pode vir a trocar mensagens.

O problema do modelo reativo é que o tempo que as consultas ao banco de dados necessárias para se decidir como tratar um fluxo levam para completar pode ser prejudicial ao desempenho da rede. Além disso, a sensação de tempo de resposta é muito mais sentida por conta do processamento ser feito na hora que os pacotes estão em trânsito.

Para agilizar a resposta dos pacotes de instalação de fluxos *OpenFlow*, é interessante utilizar uma abordagem mais proativa para recuperar os dados das máquinas virtuais. Ao contrário do modelo discutido, onde o *DCPortals* consulta o banco de dados do *OpenStack* a cada momento, é possível criar uma tabela em memória com as informações importantes e atualizá-la no momento que cada configuração da rede é modificada para manter a consistência dos dados. Ao mesmo tempo, o *DCPortals* seria responsável por atualizar todos os fluxos instalados nas tabelas de fluxos dos *switches* virtuais da rede a fim de antecipar o comportamento esperado que o controlador realizaria no modelo reativo.

Essa modificação resultaria na extinção das pesquisas *SQL* adotadas no modelo original e implicaria na criação de um novo módulo que receberia diretamente as informações do *OpenStack* assim que elas acontecessem. Esse novo módulo armazenaria somente as informações necessárias sobre as máquinas virtuais e possibilitaria ao *DC-Portals* realizar as operações em todas as máquinas no momento que a modificação acontecesse.

A nova arquitetura do modelo proativo pode ser ilustrada pela figura 4.7.

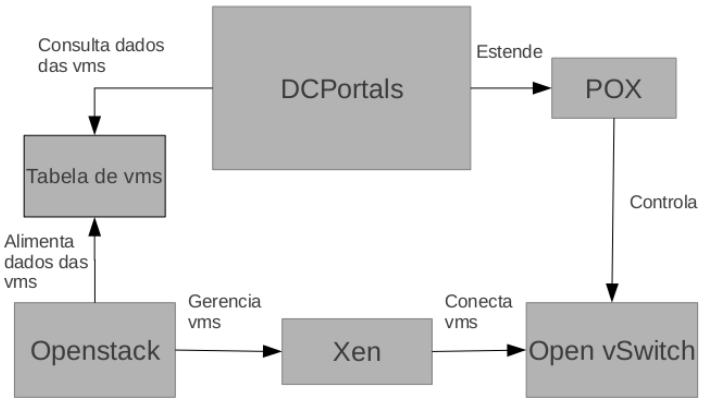

Figura 4.7. Arquitetura do sistema no modelo proativo

Para realizar a implementação do modelo proativo pode-se utilizar o mecanismo já existente do *OpenStack* de servidor de filas de mensagens, implementado pelo servidor *RabbitMQ* (VMWare, 2012). Todas as operações de comunicação entre o a *API* do *OpenStack* e o módulo *OpenStack Compute*, que é o módulo responsável pelas operações de gerenciamento de máquinas virtuais, são realizadas através desse servidor de filas. Portanto, é possível criar um módulo que se cadastre como observador da fila do módulo *OpenStack Compute* e detecte cada operação de configuração da rede.

Esse novo módulo deveria identificar quais mensagens são importantes, como criação, remoção e migração de máquinas virtuais e adquirir as informações necessárias para alimentar a tabela em memória no momento em que elas são disparadas. Uma rápida pesquisa provou que praticamente todos os dados estão disponíveis nas próprias mensagens relativas a essas operações e os demais podem ser adquiridos facilmente através de uma pesquisa *SQL* da mesma forma que o modelo atual faz.

Esse modelo apresenta uma nova preocupação que é a necessidade de atualizar todos fluxos já instalados em todas as máquinas físicas que possuem máquinas virtuais da rede que teve uma modificação no momento que ela ocorre. Essa operação consiste

na remoção dos fluxos já instalados para a configuração anterior e na instalação de novos fluxos de acordo com a nova configuração. Para resolver esse problema, é necessário que a tabela citada mantenha também um registro dos fluxos instalados, o que não é um problema muito grande, visto que é um registo da ordem de $O(n^2)$, onde n é o número de máquinas virtuais.

Capítulo 5

Conclusões e trabalhos futuros

Este trabalho apresenta o sistema *DCPortals*, desenvolvido com o intuito de oferecer uma solução de isolamento para *datacenters* de computação em nuvem que utilizam virtualização em um modelo de clientes multi-inquilinos. Foi proposta uma arquitetura utilizando ferramentas populares na área e o conceito de Redes Definidas por Software. Em sequência, foi implementado um protótipo que por fim foi avaliado funcionalmente.

Foram utilizadas as seguintes ferramentas: *OpenStack Compute* como gerenciador de computação em nuvem, *Open vSwitch* como *switch* virtual que suporta o protocolo *OpenFlow*, *Xen* como monitor de máquinas virtuais e o *POX* como o *Network Hypervisor* sobre o qual foi desenvolvida a aplicação. Essas ferramentas foram integradas e manipuladas de forma a reunir informações e funcionalidades suficientes para que o *DCPortals* pudesse realizar seu isolamento.

Os experimentos realizados demonstraram que o sistema não apresenta limitações consideráveis em relação a um sistema sem isolamento. Além disso, os testes confirmaram que ele apresenta uma resistência considerável a problemas que sistemas sem isolamento sofrem, como o ataque de negação de serviço considerado nos experimentos.

Além do sistema desenvolvido, durante o desenvolvimento do projeto várias contribuições foram feitas para as ferramentas utilizadas através da detecção de *bugs* e sugestão de correções. A maioria das sugestões foram acatadas e corrigidas na versão corrente ou o serão na próxima versão do software.

Em conversas com empresas da área, foi observada a relevância dele para empresas provedoras desse serviço, que sofrem do problema que o sistema soluciona e costumeiramente implementam soluções mais caras e menos eficazes para conseguir realizar algo parecido, porém ainda mais limitado, com o resultado observado.

O projeto foi apresentado durante um *workshop* do *inWeb* (InWeb, 2012), realizado com o intuito de discutir a distribuição de informação na *Web* em todos os níveis

da rede. Ele também resultou na publicação de um minicurso durante o *SBRC* (SBRC, 2012) sobre a aplicação de Redes Definidas por Software e a importância delas para o futuro da infraestrutura das redes de computadores.

Trabalhos futuros caminham na direção de avaliar alternativas melhores para o desempenho, como o modelo proativo descrito. Ao invés de realizar todas as pesquisas nos bancos de dados assim que o fluxo é iniciado, esse modelo prevê identificar os eventos necessários para alimentar uma tabela em memória para que a agilidade da resposta do controlador seja aumentada, melhorando o serviço ofertado através da diminuição da latência observada.

Outros trabalhos futuros interessantes incluem a extensão do sistema para trabalhar com outras ferramentas conhecidas, como o monitor de máquinas virtuais *KVM* e diferentes *Network Hypervisors*. A utilização de outras ferramentas pode ser benéfica não só por tornar o sistema mais geral como para poder utilizá-lo em uma variedade maior de ambientes, tirando proveito da particularidade de cada um.

Também é interessante pensar na possível integração futura de outras funcionalidades ao sistema que podem ser interessantes para o contexto alvo. O conceito de visão global da rede física proposta pela *SDN* permite que muitas outras políticas sejam aplicadas e integradas para que o ambiente alvo tenha mais benefícios. Dentre elas estão a implementação de algoritmos que modifiquem o comportamento da rede de acordo com critérios como a modificação do roteamento para melhorar o aproveitamento dos recursos, mecanismos de redundância e balanceamento de carga, dentre outros serviços que melhoraram o aproveitamento de recursos da infraestrutura.

Referências Bibliográficas

Alizadeh, M.; Greenberg, A.; Maltz, D. A.; Padhye, J.; Patel, P.; Prabhakar, B.; Sengupta, S. & Sridharan, M. (2010). Data center TCP (DCTCP). Em *SIGCOMM '10: Proceedings of the ACM SIGCOMM 2010 Conference on SIGCOMM*, pp. 63–74, New York, NY, USA. ACM.

Amazon (2010a). Amazon elastic compute cloud. `http://aws.amazon.com/ec2/`. Acessado em julho de 2012.

Amazon (2010b). Amazon SimpleDB. `http://aws.amazon.com/simpledb/`. Acessado em julho de 2012.

Arlitt, M. & Jin, T. (1999). Workload characterization of the 1998 world cup web site. Relatório técnico HPL-1999-35R1, HP Labs.

Barham, P.; Dragovic, B.; Fraser, K.; Hand, S.; Harris, T.; Ho, A.; Neugebauer, R.; Pratt, I. & Warfield, A. (2003). Xen and the art of virtualization. Em *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, New York, NY, USA. ACM.

Beacon (2012). The beacon controller. `https://openflow.stanford.edu/display/Beacon/Home`. Acessado em julho de 2012.

BitBucket Attack (2012). Bitbucket amazon ddos attack. `http://blog.bitbucket.org/2009/10/04/on-our-extended-downtime-amazon-and-whats-coming/`. Acessado em julho de 2012.

Bolte, M.; Sievers, M.; Birkenheuer, G.; Niehörster, O. & Brinkmann, A. (2010). Non-intrusive virtualization management using libvirt. Em *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pp. 574–579, 3001 Leuven, Belgium, Belgium. European Design and Automation Association.

Cabuk, S.; Dalton, C. I.; Eduards, A. & Fischer, A. (2008). A comparative study on secure network virtualization. Relatório técnico HPL-2008-57, HP Laboratories.

Cabuk, S.; Dalton, C. I.; Ramasamy, H. & Schunter, M. (2007). Towards automated provisioning of secure virtualized networks. Em *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pp. 235–245, New York, NY, USA. ACM.

Cai, Z.; Cox, A. L. & Ng, T. S. E. (2010). Maestro: A system for scalable OpenFlow control. Relatório técnico TR10-08, Rice University.

Casado, M.; Koponen, T.; Ramanathan, R. & Shenker, S. (2010). Virtualizing the network forwarding plane. Em *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pp. 8:1–8:6, New York, NY, USA. ACM.

Chen, Y.; Griffith, R.; Liu, J.; Katz, R. H. & Joseph, A. D. (2009). Understanding TCP incast throughput collapse in datacenter networks. Em *WREN '09: Proceedings of the 1st ACM Workshop on Research on Enterprise Networking*, pp. 73–82, New York, NY, USA. ACM.

Chesire, M.; Wolman, A.; Voelker, G. M. & Levy, H. M. (2001). Measurement and analysis of a streaming media workload. Em *USENIX Symposium on Internet Technologies and Systems (USITS)*, pp. 1–12. USENIX.

Citrix (2012). Xenserver. `http://www.citrix.com.br/products/xenserver.php`. Acessado em julho de 2012.

Clark, C.; Fraser, K.; Hand, S.; Hansen, J. G.; Jul, E.; Limpach, C.; Pratt, I. & Warfield, A. (2005). Live migration of virtual machines. Em *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pp. 273–286, Berkeley, CA, USA. USENIX Association.

Fares, M. A.; Loukissas, A. & Vahdat, A. (2008). A scalable, commodity data center network architecture. Em *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, pp. 63–74, New York, NY, USA. ACM.

Floodlight (2012). Floodlight. `http://floodlight.openflowhub.org/`. Acessado em julho de 2012.

Foster, N.; Harrison, R.; Freedman, M. J.; Monsanto, C.; Rexford, J.; Story, A. & Walker, D. (2011). Frenetic: a network programming language. *SIGPLAN Notices*, 46(9):279–291.

Ganguly, S. (2011). NEC ProgrammableFlow: Redefining cloud network virtualization with OpenFlow.

Google (2010). Google appengine. `http://code.google.com/intl/pt-BR/appengine/`. Acessado em julho de 2012.

Greenberg, A.; Hamilton, J.; Maltz, D. A. & Patel, P. (2009). The cost of a cloud: research problems in data center networks. *SIGCOMM Computer Communication Review*, 39(1):68–73.

Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N. & Shenker, S. (2008). NOX: towards an operating system for networks. *SIGCOMM Computer Communication Review*, 38:105–110.

Hinrichs, T. L.; Gude, N. S.; Casado, M.; Mitchell, J. C. & Shenker, S. (2009). Practical declarative network management. Em *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pp. 1–10, New York, NY, USA. ACM.

IEEE802.1Q (2005). IEEE standard for local and metropolitan area networks: Virtual bridged local area networks. Disponível em: `http://OpenFlowSwitch.org`. Acessado em julho de 2012.

IETF (1981). Rfc 792. `http://www.ietf.org/rfc/rfc792.txt`. Acessado em julho de 2012.

InWeb (2012). Workshop InWeb. `http://www.inweb.org.br/`. Acessado em julho de 2012.

Kohler, E.; Morris, R.; Chen, B.; Jannotti, J. & Kaashoek, M. F. (2000). The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297.

Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T. & Shenker, S. (2010). Onix: a distributed control platform for large-scale production networks. Em *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pp. 1–6, Berkeley, CA, USA. USENIX Association.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. & Turner, J. (2008). Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74.

Mell, P. & Grance, T. (2011). NIST definition of cloud computing v15. `http://www.nist.gov/itl/cloud/upload/cloud-def-v15.pdf`. Acessado em julho de 2012.

Microsoft (2012). Microsoft windows azure platform. `http://www.microsoft.com/windowsazure/`. Acessado em julho de 2012.

Mundada, Y.; Sherwood, R. & Feamster, N. (2009). An OpenFlow switch element for Click. Em *Symposium on Click Modular Router*.

Nurmi, D.; Wolski, R.; Grzegorczyk, C.; Obertelli, G.; Soman, S.; Youseff, L. & Zagorodnov, D. (2008). The eucalyptus open-source cloud-computing system. Em *Proceedings of Cloud Computing and Its Applications*.

ONSummit (2012). Open Networking Summit 2012. `http://opennetsummit.org/`. Acessado em julho de 2012.

OpenFlow (2012). OpenFlow specification. Disponível em: `http://OpenFlowSwitch.org`. Acessado em julho de 2012.

OpenNebula (2012). OpenNebula. `http://opennebula.org/`. Acessado em julho de 2012.

OpenStack (2012). OpenStack cloud software. `http://www.openstack.org/`. Acessado em julho de 2012.

Parkhill, D. F. (1966). *The challenge of the computer utility*. Addison-Wesley Publishing Company Reading Massachussets.

Pfaff, B.; Pettit, J.; Koponen, T.; Amidon, K.; Casado, M. & Shenker, S. (2009). Extending networking into the virtualization layer. Em *8th ACM Workshop on Hot Topics in Networks (HotNets-VIII)*.

SBRC (2012). Simpósio Brasileiro de Redes de Computadores. `http://sbrc2012.dcc.ufmg.br/`. Acessado em julho de 2012.

Seetharaman, S. (2012). OpenFlow/SDN: How it works, and what it means for networks beyond the campus. Em *National Fiber Optic Engineers Conference*, p. NTu1E.4. Optical Society of America.

Shi, W.; Wright, Y.; Collins, E. & Karamcheti, V. (2002). Workload characterization of a personalized web site and its implications for dynamic content caching. Em *Proceedings of the 7th International Workshop on Web Caching and Content Distribution (WCW'02)*, pp. 1–16.

SNAC (2012). Simple network access control. `http://www.openflow.org/wp/snac/`. Acessado em julho de 2012.

Trema (2012). Trema: Full-stack OpenFlow framework for Ruby and C. `http://trema.github.com/trema/`. Acessado em julho de 2012.

UOL (2012). UOL Host. `http://www.uolhost.com.br/uol-cloud-computing.html`. Acessado em julho de 2012.

Vaquero, L. M.; Rodero-Merino, L.; Caceres, J. & Lindner, M. (2009). A break in the clouds: towards a cloud definition. *SIGCOMM Computer Communication Review*, 39(1):50–55.

Vasudevan, V.; Phanishayee, A.; Shah, H.; Krevat, E.; Andersen, D. G.; Ganger, G. R.; Gibson, G. A. & Mueller, B. (2009). Safe and effective fine-grained TCP retransmissions for datacenter communication. Em *Proceedings of ACM SIGCOMM*, Barcelona, Spain.

Vaughan-Nichols, S. J. (2011). Openflow: The next generation of the network? *IEEE Computer*, 44(8):13–15.

Veloso, E.; Almeida, V.; Meira, W.; Bestavros, A. & Jin, S. (2002). A hierarchical characterization of a live streaming media workload. Em *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pp. 117--130, New York, NY, USA. ACM.

VMWare (2012). Rabbit mq. `http://www.rabbitmq.com/`. Acessado em julho de 2012.

Wang, Q.; Makaroff, D.; Edwards, H. K. & Thompson, R. (2003). Workload characterization for an e-commerce web site. Em *Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, CASCON '03, pp. 313–327. IBM Press.

Wood, T.; Shenoy, P.; Venkataramani, A. & Yousif, M. (2007). Black-box and gray-box strategies for virtual machine migration. Em *Proceedings of the 4th USENIX*

conference on Networked Systems Design and Implementation, NSDI'07, pp. 17–17, Berkeley, CA, USA. USENIX Association.

Zhang, Q.; Cheng, L. & Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18.

Apêndice A

Exemplo de switch ethernet comum no POX

A interface do *POX* é bastante simples e direta. Para implementar um componente basta criar um arquivo com um método *launch* e registrar uma função para um ou mais eventos cadastrados. Por exemplo, o código a seguir implementa um *switch ethernet* comum com aprendizado. Assim que um pacote passa pelo *switch*, ele armazena a origem daquele pacote e mapeia o endereço *ethernet* dele à porta por onde ele veio. A seguir, verifica se o destino é conhecido, para que possa ser encaminhado para a porta correta, ou envia o pacote para todas as portas para que o pacote chegue a seu destino.

```
1  from pox.core import core
2  from pox.openflow.libopenflow_01 import *
3
4  log = core.getLogger()
5
6  macToPort = {}
7
8  def flood(event):
9     msg = ofp_packet_out()
10    msg.actions.append(ofp_action_output(port = OFPP_FLOOD))
11    msg.buffer_id = event.ofp.buffer_id
12    msg.in_port = event.port
13    event.connection.send(msg)
14
15 def installFlow(packet,event):
```

```
16    msg = ofp_flow_mod()
17    msg.match.dl_dst = packet.dst
18    msg.match.dl_src = packet.src
19    msg.buffer_id = event.ofp.buffer_id
20    msg.actions.append(ofp_action_output(port = macToPort[packet.dst]))
21    event.connection.send(msg)
22
23 def handle_PacketIn (event):
24  packet = event.parse()
25
26  if packet.dst.isMulticast():
27      flood(event)
28      return
29
30  if packet.src not in macToPort:
31    macToPort[packet.src] = event.port
32    log.info("Aprendeu localizacao de %s",packet.src)
33
34  if packet.dst not in macToPort:
35    log.info("Fazendo flood do pacote de %s para %s",packet.src,packet.dst)
36    flood(event)
37  else:
38    installFlow(packet,event)
39    log.info("Instalando fluxo de %s para %s",packet.src,packet.dst)
40
41 def launch ():
42  core.openflow.addListenerByName("PacketIn", handle_PacketIn)
```

As linhas 1-6 importam a biblioteca *core* do *POX* além de inicializar as variáveis de *log* e o dicionário de tradução de endereço *MAC* para porta do *switch*. É importante atentar que esse exemplo não cria instâncias para diferentes *switches*, portanto, não serve para um ambiente com mais de um *switch OpenFlow* conectado ao controlador.

As linhas 8-13 definem a sub-rotina *flood*, que tem o papel de criar uma mensagem *OpenFlow* para apenas enviar o pacote para todas as portas do *switch*. É importante também comentar que o tipo de mensagem *OpenFlow PacketOut* utilizado nessa mensagem não registra um fluxo no *switch*, apenas encaminha o pacote. Dessa forma, na próximo pacote desse fluxo o controlador será consultado novamente para ver se possui

o destino correto do pacote e possa instalar o fluxo para que todos os pacotes seguintes sejam encaminhados diretamente para o destino sem que exista a necessidade de consultar o controlador novamente.

Que é o que de fato acontece nas linhas 15-21, onde é definida a sub-rotina *installFlow*. Essa sub rotina instala um fluxo na tabela do *switch OpenFlow* para que todos os pacotes a partir do registrado com o *buffer_id* correspondente que sigam o mesmo fluxo sejam encaminhados pela mesma porta do *switch*. Nesse caso apenas foi definido como critérios de validação do fluxo os endereços de origem e destino do cabeçalho *ethernet*, mas diversos outros campos poderiam ser definidos.

As linhas 23-39 implementam a sub-rotina *handle_PacketIn*, que foi definida como uma sub-rotina que executa a lógica do *switch* e que será chamada assim que um evento de *PacketIn* for levantado pelo *POX*, ou seja, sempre que um pacote que não bate em qualquer fluxo em um *switch OpenFlow* for encaminhado ao controlador. Ele somente realiza o comportamento esperado de um *switch ethernet* convencional, encaminhado *multicasts* para todas as portas, verificando se já contém o mapeamento do endereço da origem do pacote e armazenando-o caso não contenha, verificando se contém o endereço do destino, instalando o fluxo caso conheça e enviando o pacote para todas as portas caso contrário.

Já as linhas 41-42 definem a função *launch*, que é executada assim que o módulo é chamado pelo *POX*. No caso, apenas registra que a função *handle_PacketIn* deverá ser chamada sempre que o evento *PacketIn* for levantado.

Apêndice B

Roteiro de instalação

Resumo

Este roteiro descreve como preparar o ambiente para o gerenciamento de uma nuvem formada por máquinas virtuais utilizando o Hypervisor Xen. Ele foi escrito em conjunto com o aluno de iniciação científica Raphael Luciano de Pontes. Para o desenvolvimento desse sistema, é preciso realizar a integração de algumas ferramentas. São elas:

- Monitor de máquinas virtuais Xen
- OpenVswitch
- OpenStack
- POX + DCPortals

B.1 Monitor de máquinas virtuais Xen

B.1.1 Instalação no Ubuntu 10.04 server

Para instalar o Xen nesta versão do Ubuntu é preciso compilar, manualmente, um kernel próprio para o Xen, uma vez que o kernel oficial da distribuição não possui suporte próprio. Foi usado, nesse ambiente, uma versão do kernel 2.6 com os patches do Xen aplicados. Para compilar esse kernel, é necessário instalar as seguintes dependências:

```
# apt-get install build-essential zlib1g-dev gettext libssl-dev \
libx11-dev bin86 bcc libncurses5-dev python-dev texinfo \
bridge-utils fig2ps bison flex gcc g++ xorg eterm xterm zlib1g zlib1g-dev \
libssl-dev uuid-dev xserver-xorg-dev gettext libncurses5-dev \
```

```
bin86 bcc g++-multilib iasl python2.7-dev texinfo ocaml-findlib \
git-core kernel-package
```

obs: O pacote "python2.7-dev" não vem por padrão nos repositórios do Ubuntu. É necessário adicionar um repositório ppa para ter acesso a essa versão: https://launchpad.net/ fkrull/+archive/deadsnakes

Descompacte o kernel na pasta `/usr/src/`. Agora será preciso compilar e instalar o novo kernel. E depois fazer com que ele carregue por padrão no bootloader. Para isso, faça os seguintes comandos (estando dentro da pasta do kernel compilado):

```
# make menuconfig
# make -j5
# make install
# make modules_install
# mkinitramfs -o /boot/initrd.img-<versao> <versao>
```

(Para ver a versão, basta olha a versão do arquivo "vmlinuz" criado na pasta `/boot`)

Altere o tempo de espera do grub no arquivo `/etc/default/grub` para que seja possível escolher outro kernel em caso de erro. Para isso, altere as duas linhas que serão explicitadas abaixo.

Linha original:
`GRUB_HIDDEN_TIMEOUT=0`
Mude para:
`GRUB_HIDDEN_TIMEOUT=10`

Linha original:
`GRUB_HIDDEN_TIMEOUT_QUIET=true`
Mude para:
`GRUB_HIDDEN_TIMEOUT_QUIET=false`

Em seguida, será necessário instalar o Hypervisor Xen. A versão usada em nosso ambiente é a 4.1.1. Ele foi baixado no site `http://www.xen.org/`. Descompacte o arquivo compactado do Xen na máquina desejada. Leia o arquivo README e/ou INSTALL, presentes no diretório descompactado do Xen, sobre como instalá-lo.

Primeiramente instale todas as dependências que o manual indicar. Depois siga os comandos abaixo para compilar e instalar o Xen.

```
# make xen
# make tools
# make stubdom
# make docs
# make install-xen
# make install-tools PYTHON_PREFIX_ARG=
# make install docs
```

É também preciso adicionar uma entrada para subir o Hypervisor do Xen no bootloader junto com o kernel do Linux. Para isso é preciso seguir algumas etapas.

Copie, do arquivo `/boot/grub/grub.cfg`, o bloco de comandos do "menuentry" do kernel compilado e cole nas últimas linhas do arquivo `/etc/grub.d/40_custom`. Abaixo esta o exemplo do bloco de comandos do "menuentry" usado no nosso ambiente.

```
menuentry "Ubuntu, Xen 4.1.1 Linux <VERSÃO DO KERNEL COMPILADO>"
{
    set root=(hd0,1)
    multiboot /boot/xen-4.1.1.gz dummy=dummy
    module /boot/vmlinuz-<VERSÃO DO KERNEL COMPILADO> root=/dev/sda1
    root=/dev/sda1 ro
    module /boot/initrd.img-<VERSÃO DO KERNEL COMPILADO>
}
```

Logo depois é preciso fazer com que a entrada do Ubuntu com o Hypervisor Xen seja a padrão no boot. Para isso, abra o arquivo `/etc/default/grub` e modifique a seguinte linha:

```
GRUB_DEFAULT=0
```

Mude-a, informando o nome da entrada "menuentry" criada para o Ubuntu com o Xen no boot padrão do sistema. A linha deverá ficar parecida com a abaixo:

```
GRUB_DEFAULT="Ubuntu, Xen 4.1.1 Linux <VERSÃO DO KERNEL COMPILADO>"
```

Agora é só fazer um `# update-grub` e o Xen subirá junto com o kernel, permitindo utilizar suas funcionalidades.

Por fim inicie os serviços abaixo (ou os configure como serviços padrões do sistema):

```
# /etc/init.d/xencommons start
# /etc/init.d/xend start
```

B.1.2 Instalação no Ubuntu 11.10 server

Nessa versão o Hypervisor Xen já faz parte dos pacotes do Ubuntu. Logo para instalá-lo é necessário apenas alguns comandos:

Xen Hypervisor no AMD64:

```
# apt-get install xen-hypervisor-4.1-amd64
```

Xen Hypervisor no i386:

```
# apt-get install xen-hypervisor-4.1-i386
```

É também preciso adicionar uma entrada para subir o Hypervisor do Xen no bootloader junto com o kernel do Linux, assim como na versão anterior do Ubuntu. Siga o mesmo procedimento descrito anteriormente para criação de uma entrada extra no arquivo `/etc/grub.d/40_custom` e a configuração para usá-la como padrão no `/etc/default/grub`.

B.1.3 Configuração da rede no Xen 4.1.1

A partir da versão 4.1, o Xen modificou a forma como configura a rede. A partir dessa versão ele deixa para o sistema criar a interface de rede e o bridge que conecta a interface de rede física às máquinas virtuais. Para configurar esse bridge para iniciar automaticamente no Ubuntu, deve-se editar o arquivo /etc/network/interfaces e substituir qualquer configuração da interface de rede física conectada na rede das máquinas virtuais pela seguinte definição de bridge:

```
auto xenbr0
iface xenbr0 inet static
        bridge_ports eth1
        bridge_stp off
```

```
bridge_maxwait 0
bridge_fd 0
address 10.0.0.2
netmask 255.255.255.0
broadcast 10.0.0.255
```

Nesse caso, a interface de rede física é a eth1 e a faixa de rede das interfaces físicas conectadas à rede das máquinas virtuais é 10.0.0.0/24. Essa faixa de rede é diferente da que será configurada para as máquinas virtuais.

B.2 OpenvSwitch

O OpenVswitch é uma ferramenta de código livre que cria um switch virtual, conectando várias máquinas virtuais que estão hospedadas em uma determinada máquina real. Ele realiza a função de um switch, só que virtualizado, no ambiente das máquinas virtuais. Ele também pode ser usado para substituir o módulo de bridge do kernel do Linux e suportar múltiplos monitores de máquinas virtuais, como Xen, KVM e VirtualBox.

Como este switch fará a conexão das vm's, ele só precisa ser instalado nos nós-compute. Para instalá-lo, é preciso instalar os pacotes do OpenVswitch nos nós-compute:

```
# apt-get install openvswitch-brcompat openvswitch-switch
openvswitch-datapath-dkms
```

Foi preciso fazer uma pequena modificação no arquivo de configuração `/etc/default/openvswitch-switch` para que o módulo que substitui o kernel do Linux seja habilitado ao subir o serviço. Abaixo está a linha que deve ser alterada no arquivo:

Linha original:
BRCOMPAT=no

Mudar para "yes"onde esta "no":
BRCOMPAT=yes

Para que o OpenVswitch funcione corretamente, o módulo bridge padrão do Linux não pode estar rodando. Então é preciso removê-lo manualmente com o

comando:

```
# rmmod bridge
```

Mesmo após esse comando o módulo pode não ser removido. Para verificar se o módulo bridge realmente foi removido use o comando abaixo para listar os módulos. O modulo bridge não deve ser listado e se estiver o comando acima deve ser repetido.

```
# lsmod | grep bridge
```

Agora precisamos criar a bridge xenbr0 e adicionar uma porta eth (verifique antes qual porta eth está disponível para a rede das máquinas virtuais). Para isso uso com os comandos ovs-vsctl:

Cria bridge xenbr0:

```
# ovs-vsctl add-br xenbr0
```

Adiciona porta ethx na bridge xenbr0:

```
# ovs-vsctl add-port xenbr0 ethx
```

Nesse ponto, ao criar vm's no controlador, o OpenVswitch deverá adicionar automaticamente, na máquina hospedeira, as interfaces referentes de cada vm criada. Para verificar isso, após criar uma vm, vá até o nó-compute onde ela foi instanciada e dê o comando "`# ovs-vsctl show`". Ele retornará informações sobre a bridge existente e as interfaces de cada vm adicionada na bridge (bridge xenbr0, nesse caso).

Como exemplo, abaixo temos o retorno do comando "`# ovs-vsctl show`" para um nó-compute que possui 4 máquinas virtuais hospedadas (interfaces das vm's são, respectivamente, vif10.0, vif11.0, vif12.0 e vif13.0).

```
Bridge "xenbr0"
        Port "xenbr0"
            Interface "xenbr0"
            type: internal
        Port "vif11.0"
            Interface "vif11.0"
        Port "vif10.0"
```

```
        Interface "vif10.0"
    Port "vif13.0"
        Interface "vif13.0"
    Port "vif12.0"
        Interface "vif12.0"
    Port "eth0"
        Interface "eth0"
    ovs_version: "1.2.0+build0"
```

Esse bloco significa que existe uma bridge chamada `xenbr0` (adicionada manualmente), e dentro dela, existem as portas:

`xenbr0`	(criada automaticamente ao criar a bridge xenbr0)
`vif11.0`	(adicionada automaticamente pelo Xen)
`vif10.0`	(adicionada automaticamente pelo Xen)
`vif13.0`	(adicionada automaticamente pelo Xen)
`vif12.0`	(adicionada automaticamente pelo Xen)
`eth0`	(adicionada manualmente)

As informações retornadas pelo comando ovs-vsctl, referentes às portas e à bridge, devem estar em coerência com as informações retornadas pelo comando "`# brctl show`". Ambas devem retornar a mesma configuração de portas e redes virtuais (VIF's).

B.3 OpenStack

O OpenStack é um gerenciador de computação em nuvem que será instalado em um controlador e em todas as máquinas hospedeiras, que serão os nós-compute. Os nós-compute são as máquinas físicas onde estarão hospedadas as máquinas virtuais enquanto que o controlador receberá os comandos da API de configuração para gerenciar o ambiente.

A seguir serão descritos os processos de instalação do controlador e dos nós-compute, além de um requisito para seu funcionamento, que é a sincronização dos horários entre as máquinas utilizando o protocolo NTP. Ao final da seção serão descritos alguns problemas comuns durante o processo de configuração e como corrigi-los.

B.3.1 Sincronização NTP

O NTP (Network Time Protocol) é um protocolo para sincronização dos relógios de um conjunto de computadores em redes de dados com latência variável. O NTP permite manter o relógio dos computadores em rede com a hora sempre certa e com grande exatidão. É necessário sincronizar o horário, do controlador e dos nós-compute, com o protocolo ntp. Para isso foi preciso fazer o controlador se tornar um servidor de horário, e os nós-compute se tornarem clientes desse servidor. Então instale o ntp em ambos, controlador e nós.

```
# apt-get install ntp
```

B.3.1.1 Configuração no Controlador

Foi preciso fazer algumas alterações no controlador, no arquivo de configuração do ntp, para informar qual será o servidor de horário com o qual ele será sincronizado. Modifique, no arquivo `/etc/ntp.conf`, a linha referente ao servidor desejado. Abaixo esta o exemplo do nosso ambiente.

```
driftfile /var/lib/ntp/ntp.drift
statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org
server ntp.ubuntu.com
server 127.127.1.0
fudge 127.127.1.0 stratum 10
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery
restrict 127.0.0.1
restrict ::1
```

obs: A linha com o endereço *server ntp.ubuntu.com* indica o servidor de horário com o qual o nosso controlador irá sincronizar. Nesse caso o servidor é o oficial do

Ubuntu, mas também é possível sincronizar com o servidor do local de trabalho.

Também é necessário configurar um bridge para que o controlador possa se comunicar com as máquinas virtuais. Assim como na configuração o bridge xenbr0 criado para o Xen, esse bridge precisa substituir a configuração da interface de rede física conectada na rede de máquinas virtuais. É importante também que ele compartilhe o mesmo nome dos bridges dos nós-compute. O exemplo utilizado a seguir demonstra como o arquivo `/etc/network/interfaces` foi configurado em nosso ambiente:

```
auto xenbr0
iface xenbr0 inet static
        bridge_ports eth1
        bridge_stp off
        bridge_maxwait 0
        bridge_fd 0
        address 10.0.50.1
        netmask 255.255.255.0
        broadcast 10.0.50.255
```

Note que ao contrário da configuração do bridge relativo à rede do Xen, esse bridge está na faixa de redes 10.0.50.0/24, que é a mesma faixa de rede que configuramos para as máquinas virtuais. Isso é necessário porque o controlador do OpenStack deve se comunicar com as máquinas virtuais.

B.3.1.2 Configuração nos Nós-Compute

Nos nós-compute é preciso fazer as modificações no arquivo `/etc/ntp.conf`, para ele ficar parecido com o exemplo abaixo:

```
driftfile /var/lib/ntp/ntp.drift
statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable
# ABAIXO ESTA O IP DO SERVIDOR LOCAL
server 10.0.254.1
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery
restrict 127.0.0.1
```

```
restrict ::1
```

obs: A linha com o endereço *10.0.254.1* corresponde ao IP do servidor ao qual esse cliente ntp estará sendo sincronizado, que nesse caso, é o controlador do OpenStack. É preciso trocar essa linha para o IP do seu controlador.

B.3.1.3 Sincronização

Em seguida é necessário fazer uma sincronização forçada entre os clientes ntp e o controlador ntp (servidor local). Esse comando é necessário quando a diferença entre o cliente e servidor é muito grande, após uma sincronização forçada as diferenças pequenas serão corrigidas automaticamente pelo serviço do ntp.
Estando logado no controlador, pare o serviço ntp:

```
# /etc/init.d/ntp stop
```

Sincronize o horário com o servidor:

```
# ntpdate servidor ntp.speed.dcc.ufmg.br
```

E reinicie o serviço:

```
# /etc/init.d/ntp start
```

Agora, estando logado nos nós-compute, pare o serviço:

```
# /etc/init.d/ntp stop
```

Sincronize o horário com o controlador:

```
# ntpdate <ip_controlador>
```

Reinicie o serviço:

```
# /etc/init.d/ntp start
```

Ao fim de cada sincronização (`# ntpdate ip`) a mensagem retornada deve ser

algo do tipo:

```
27 Mar 15:17:33 ntpdate[15263]:  adjust time server 10.0.254.1 offset
0.000085 sec
```

Pronto, agora as máquinas estão com o horário sincronizado via ntp.

B.3.2 Instalação nos Nós-Compute

O serviço nova-compute deve ser instalado em todos as máquinas que irão hospedar máquinas virtuais, ou nós-compute. Instale o serviço nova-compute em todos os "nós-compute".

```
# apt-get install nova-compute
```

Foi preciso alterar o arquivo de configuração `/etc/nova/nova.conf` para que ficasse configurado de acordo com o ambiente criado, alterando IPs, usuários Mysql, banco de dados, etc. Abaixo está o exemplo que seguimos no nosso ambiente.

```
-dhcpbridge_flagfile=/etc/nova/nova.conf
-dhcpbridge=/usr/bin/nova-dhcpbridge
-logdir=/var/log/nova
-state_path=/var/lib/nova
-lock_path=/var/lock/nova
-verbose
-s3_host=10.0.254.1
-rabbit_host=10.0.254.1
-cc_host=10.0.254.1
-ec2_url=http://10.0.254.1:8773/services/Cloud
-sql_connection=mysql://nova:notnova@10.0.254.1/nova
-network_manager=nova.network.manager.FlatDHCPManager
-network_host=10.0.254.1
-flat_network_bridge=xenbr0
-libvirt_type=xen
-xenapi_remap_vbd_dev=true
-glance_api_servers=10.0.254.1:9292
-image_service=nova.image.glance.GlanceImageService
```

```
-nouse_cow_images
```

As linhas com o endereço *10.0.254.1* são relativas ao IP da máquina controladora, então deve-se alterar esse IP para o IP do controlador que você está criando.

A linha com os valores *nova:notnova* são informações referentes ao banco de dados do Mysql. Note que são duas palavras separadas por `:` (dois pontos). A primeira palavra é o usuário do mysql e a segunda palavra é a senha desse usuário (ambas serão criadas posteriormente). Substitua esse login e senha pelo login e senha desejado.

Foi preciso fazer configurações em algumas linhas de outros três arquivos. Seguem abaixo as instruções referentes a cada modificação necessária em cada um deles:

A primeira modificação é no arquivo `/etc/xen/xend-config.sxp`, habilite a interface http descomentando e modificando as linhas comentadas nos dois blocos abaixo:

O primeiro bloco é o abaixo:

```
#(xend-http-server no)
#(xend-unix-server no)
#(xend-tcp-xmlrpc-server no)
#(xend-unix-xmlrpc-server yes)
#(xend-relocation-server no)
#(xend-relocation-ssl-server no)
#(xend-udev-event-server no)
```

Ele deve ser modificado para ficar da seguinte maneira:

```
(xend-http-server yes)
#(xend-unix-server no)
#(xend-tcp-xmlrpc-server no)
#(xend-unix-xmlrpc-server yes)
(xend-relocation-server yes)
#(xend-relocation-ssl-server no)
#(xend-udev-event-server no)
```

O segundo bloco é o abaixo:

```
# Port xend should use for the HTTP interface, if xend-http-server
is set.
```

```
#(xend-port 8000)

# Port xend should use for the relocation interface, if xend-
# relocation-server is set.
#(xend-relocation-port 8002)
```

Ele deve ser modificado para ficar da seguinte maneira:

```
# Port xend should use for the HTTP interface, if xend-http-server
is set.
(xend-port 8000)

# Port xend should use for the relocation interface, if xend-
# relocation-server is set.
(xend-relocation-port 8002)
```

obs: Basicamente o que deve ser feito é descomentar as linhas (retirar o # na frente da linha) e mudar para "yes", caso tenha um "no", no final da linha.

O segundo arquivo a ser modificado é o `/usr/share/pyshared/nova/virt/libvirt/connection.py` para corrigir um bug da versão OpenStack contida no Ubuntu 11.10. Será preciso alterar o seguinte bloco:

```
def list_instances(self):
    return [self._conn.lookupByID(x).name()
        for x in self._conn.listDomainsID()]
```

Para o seguinte:

```
def list_instances(self):
    return [self._conn.lookupByID(x).name()
        for x in self._conn.listDomainsID() if x != 0]
```

O terceiro arquivo a ser modificado é o `/usr/share/pyshared/nova/virt/libvirt.xml.template`. Onde será corrigida uma incompatibilidade do libvirt com máquinas Xen.
Nele devem ser feitas modificações em dois lugares.

O primeiro bloco a ser modificado é o seguinte:

```
#else
     #if $type == 'xen'
          #set $disk_bus = 'scsi'
               <type>linux</type>
               #set $root_device_name = $getVar('root_device_name',
'/dev/xvda')
               <root>$root_device_name</root>
```

Ele deve ficar da seguinte maneira:

```
#else
     #if $type == 'xen'
          #set $disk_bus = 'scsi'
               <type>linux</type>
               #set $root_device_name = '/dev/sda'
               <root>$root_device_name</root>
```

O segundo bloco a ser modificado é o seguinte:

```
#else
     #if $getVar('kernel', None)
     <kernel>$kernel</kernel>
          #if $type == 'xen'
               <cmdline>ro</cmdline>
          #else
               #set $root_device_name = $getVar('root_device_name',
'/dev/vda')
               <cmdline>root=$root_device_name console=ttyS0</cmdline>
```

Ele deve ficar da seguinte maneira:

```
#else
     #if $getVar('kernel', None)
     <kernel>$kernel</kernel>
```

```
        #if $type == 'xen'
            <cmdline>ro</cmdline>
        #else
            #set $root_device_name = $getVar('root_device_name',
'/dev/vda')
            <cmdline>root=/dev/sda console=ttyS0</cmdline>
```

Algumas vezes o módulo xen_gntdev não estava sendo iniciado corretamente no boot. Caso esse seja o caso, é necessário adicionar uma entrada para que ele seja levantado automaticamente. Para isso, abra o arquivo /etc/modules e adicione a linha abaixo no final do arquivo.

```
xen_gntdev
```

B.3.3 Instalação no Controlador

É necessário, antes de começar a instalação do controlador, instalar alguns pacotes de dependências:

```
# apt-get install python-software-properties
# add-apt-repository ppa:openstack-release/2011.3
# apt-get update
```

```
# apt-get install -y rabbitmq-server python-greenlet python-mysqldb \
euca2ools unzip
```

Instale todos os serviços "nova" abaixo (menos o nova-compute) e o servidor de imagens `glance`:

```
# apt-get install nova-volume nova-vncproxy nova-api nova-ajax-console-proxy \
nova-doc nova-scheduler nova-objectstore nova-network glance
```

B.3.3.1 Instalação e Configuração do MYSQL

O OpenStack irá usar o banco de dados Mysql para armazenar informações e configurações da rede e da nuvem. Abaixo segue a instalação do MYSQL, configuração do banco de dados "nova" e configuração do usuário "nova", juntamente com suas permissões.

Primeiramente é preciso configurar algumas variáveis de sistema para facilitar o processo de definição de senha.

```
bash
MYSQL_PASS=nova
NOVA_PASS=notnova
cat «MYSQL_PRESEED | debconf-set-selections
mysql-server-5.1 mysql-server/root_password password $MYSQL_PASS
mysql-server-5.1 mysql-server/root_password_again password $MYSQL_PASS
mysql-server-5.1 mysql-server/start_on_boot boolean true
MYSQL_PRESEED
```

Em seguida, instale o MySQL com o seguinte comando:

```
# apt-get install -y mysql-server
```

Para habilitar acesso externo ao servidor mysql é necessário editar o arquivo `/etc/mysql/my.cnf` para mudar o bind-address de 127.0.0.1 para 0.0.0.0:

```
# sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
```

Agora reinicie o serviço mysql:

```
# service mysql restart
```

Para configurar o MySQL, crie o banco de dados nova:

```
# mysql -u root -p$MYSQL_PASS -e 'CREATE DATABASE nova;'
```

É necessário criar o usuário "nova", dar permissões dele alterar o banco e criar sua senha:

```
# mysql -u root -p$MYSQL_PASS -e "GRANT ALL PRIVILEGES ON *.* TO 'nova'@'%' WITH GRANT OPTION;"
# mysql -u root -p$MYSQL_PASS -e "SET PASSWORD FOR 'nova'@'%' = PASSWORD('$NOVA_PASS');"
```

Foi necessário alterar também o arquivo de configuração `/etc/nova/nova.conf`, para que ficasse configurado de acordo com o nosso ambiente. Abaixo está o exemplo seguido para nos nossos testes:

```
-dhcpbridge_flagfile=/etc/nova/nova.conf
-dhcpbridge=/usr/bin/nova-dhcpbridge
-logdir=/var/log/nova
-state_path=/var/lib/nova
-lock_path=/var/lock/nova
-verbose
-s3_host=0.0.0.0
-rabbit_host=10.0.254.1
-cc_host=10.0.254.1
-ec2_url=http://10.0.254.1:8773/services/Cloud
-fixed_range=10.0.50.0/24
-network_size=100
-FAKE_subdomain=ec2
-routing_source_ip=10.0.254.1
-verbose
-sql_connection=mysql://nova:notnova@10.0.254.1/nova
-network_manager=nova.network.manager.FlatDHCPManager
-flat_network_dhcp_start=10.0.50.2
-flat_network_bridge=xenbr0
-glance_api_servers=10.0.254.1:9292
-image_service=nova.image.glance.GlanceImageService
-use_deprecated_auth=true
```

O endereço *10.0.254.1* é o IP da máquina controladora, então é preciso alterar esse IP para o IP do controlador que esta sendo criado.

O IP *10.0.50.2* é o endereço inicial da faixa de IP destinada as vm's que serão criadas no OpenStack. A mesma faixa deverá ser escolhida para a rede configurada no projeto do OpenStack.

Os valores *nova:notnova* são informações referente ao banco de dados do Mysql. Note que são duas palavras separadas por : (dois pontos). A primeira palavra é o usuário do mysql e a segunda palavra é a senha desse usuário (ambos foram criados na etapa anterior). Basta substituir esse login e senha pelo login e senha criados.

B.3.3.2 Criando o ambiente para o OpenStack

A primeira etapa consiste em assegurar que o esquema do seu banco de dados está atualizado. Execute o comando:

```
# nova-manage db sync
```

É necessário que se tenha um usuário administrador no OpenStack.
Crie o usuário administrador com o comando abaixo:

```
# nova-manage user admin <nome_de_usuario>
```

Agora crie um projeto, utilizando o usuário administrador criado acima, com o comando:

```
# nova-manage project create <nome_do_projeto> <nome_de_usuario>
```

Por último, é necessário criar e configurar a rede na qual as máquinas virtuais serão distribuídas. Para isso, execute o comando:

```
# nova-manage network create <nome-da-rede> <faixa-de-ip> \
<numero-de-redes> <enderecos-na-rede>
```

Abaixo segue, como exemplo, a configuração usada em nosso ambiente:

```
# nova-manage db sync
# nova-manage user admin cloudypants
# nova-manage project create wpscales cloudypants
# nova-manage network create novanet 10.0.50.0/24 1 256
```

B.3.3.3 Criando as credenciais

É preciso criar as credenciais para os usuários. Elas serão a certificação para podermos criar instâncias, agrupar imagens e usar todas as outras funções da API.

Primeiro crie a pasta onde estarão as credenciais, no diretório root:

```
# mkdir -p /root/creds
```

Use o nova-manage para criar o arquivo zip, que conterá informações sobre o projeto (wpscales), com o usuário administrador (cloudypants). Ambos, projeto e usuário, foram criados na etapa anterior).

```
# nova-manage project zipfile wpscales cloudypants
/root/creds/novacreds.zip
```

Descompacte o .zip:

```
# unzip /root/creds/novacreds.zip -d /root/creds/
```

Para simplificar o uso da interface do OpenStack, adicione as informações das credenciais criadas para seu arquivo .bashrc.

```
# cat /root/creds/novarc »  /.bashrc
# source  /.bashrc
```

Use os comandos abaixo para liberar o "ping" e o acesso "ssh" para as vm's.

```
# euca-authorize -P icmp -t -1:-1 default
# euca-authorize -P tcp -p 22 default
```

Para checar se esta tudo funcionando corretamente até este ponto, use o comando abaixo. Ele deve retornar uma saída mostrando detalhes dos nós-compute que estão acessíveis ao controlador. Segue um exemplo utilizado no ambiente criado nesse teste:

```
# euca-describe-availability-zones verbose
```

AVAILABILITYZONE nova available
AVAILABILITYZONE |- cloudinha1
AVAILABILITYZONE | |- nova-network enabled :-) 2012-04-02 18:12:19
AVAILABILITYZONE | |- nova-scheduler enabled :-) 2012-04-02 18:12:18
AVAILABILITYZONE | |- nova-compute enabled XXX 2012-03-01 21:13:13
AVAILABILITYZONE |- cloudinha2
AVAILABILITYZONE | |- nova-compute enabled :-) 2012-04-02 18:12:17
AVAILABILITYZONE |- cloudinha3
AVAILABILITYZONE | |- nova-compute enabled XXX 2012-04-02 18:06:04

AVAILABILITYZONE |- cloudinha4
AVAILABILITYZONE | |- nova-compute enabled XXX 2012-03-12 18:15:14

É possível perceber que em algumas das linhas aparece o simbolo :-) ou XXX. O :-) significa que a o nó-compute, da instância em questão, está ativo. O XXX significa que a instância em questão não foi levantada porque ocorreu algum problema.

B.3.3.4 Publicando a Imagem

Agora é preciso publicar uma imagem de máquina virtual para ser usada na criação das máquinas virtuais do OpenStack.

Para essa etapa o OpenStack disponibiliza uma imagem do Ubuntu para download. Caso você não possua uma imagem, os dois comandos seguintes irão fazer o download dela, já pronta para a publicação (terceiro comando seguinte).

O comando abaixo apenas salva, na variável "image", o nome da imagem a ser publicada:

```
# image="ubuntu1010-UEC-localuser-image.tar.gz"
```

Faça o download da imagem:

```
# wget http://c0179148.cdn1.cloudfiles.rackspacecloud.com/
ubuntu1010-UEC-localuser-image.tar.gz
```

Publique a imagem para que ela possa ser acessada pelos nós-compute que, por sua vez, irão levantar as vm's com ela. Use o seguinte comando:

```
# cloud-publish-tarball $image wpbucket amd64
```

obs: O último parâmetro (amd64, nesse caso) vai depender do ambiente que esta utilizando.

Depois desse último comando, você irá obter como retorno três referências: emi, eri e eki. São siglas em que: emi significa a imagem da máquina, eri significa a imagem ramdisk e eki significa imagem do kernel. Então para garantir que tudo foi bem sucedido, neste caso deve ser retornado algo do tipo:

```
Qui Mar 29 14:52:56 BRT 2012:  ====== done ======
emi="ami-00000002"; eri="none"; eki="aki-00000001";
```

obs: Lembre-se de que os serviços nova precisam estar todos rodando, e os nós compute precisam estar com o módulo xen_ gntdev levantado.

B.3.3.5 Inicializando Instâncias

Para criar as máquinas virtuais, é necessário criar uma chave RSA que garante uma autenticação segura para as instâncias e que será utilizada no projeto como identificador de rede lógica privativa.

O seguinte comando cria uma chave privada RSA e cadastra a chave pública no sistema. Guarde o arquivo com a chave privada com uma permissão restritiva:

```
# euca-add-keypair mykey > /root/mykey.priv
# chmod 0600 /root/mykey.priv
```

Agora crie instâncias de máquinas virtuais utilizando a chave mykey.
Para isso utilize o comando a seguir:

```
# euca-run-instances ‘‘nome_da_instancia’’ -k mykey -t m1.tiny
```

Faça login na máquina virtual criada:

```
# ssh usuario@ip
```

obs: Se a imagem utilizada foi a disponibilizada pelo OpenStack, esta imagem já vem, por padrão, com o usuario=ubuntu e senha=ubuntu. Ela também aceita o login sem senha utilizando a chave privada criada no comando anterior. Os IPs das máquinas virtuais criadas podem ser encontrados com o comando euca-describe-availability-zones verbose.

B.3.4 Erros comuns de configuração

Ao publicar a imagem (cloud-publish-tarball $image wpbucket amd64) podem ocorrer alguns erros que envolvem a configuração dos arquivos nova.conf e o novarc (extraído no diretório /root/creds/novacreds.zip). Como foram erros muito recorrentes, eles serão comentados a seguir.

Na publicação, ocorreu um erro retornando a seguinte mensagem:

```
failed to upload bundle to wpbucket/maverick-server-uec-amd64-vmlinuz-virtual.manifest.xml

failed:  euca-upload-bundle -bucket wpbucket -manifest /tmp/cloud-publish-image.heQeT3
/maverick-server-uec-amd64-vmlinuz-virtual.manifest.xml

Checking bucket:  wpbucket
[Errno 111] Connection refusedfailed to upload kernel
```

A primeira coisa feita foi verificar se as variáveis do ambiente EC2 estavam corretas, especialmente a EC2_URL e a S3_URL, e os serviços nova-api e nova-objectstore estavam rodando. Esses valores estão no arquivo novarc (gerado ao extrair o novacreds.zip e posteriormente adicionados ao .bashrc). As variáveis devem estar com uma configuração parecida com a abaixo:

```
EC2_URL="http://192.168.1.77:8773/services/Cloud"
S3_URL="http://192.168.1.77:3333"
```

Também é necessário verificar se os serviços estão vinculados às portas corretas:

```
[root@c46074 ]#         netstat -tnap | grep 3333

(retorno do comando):
tcp 0 0 0.0.0.0:3333 0.0.0.0:* LISTEN 2283/python

[root@c46074 ]#         netstat -tnap | grep 8773

(retorno do comando):
tcp 0 0 0.0.0.0:8773 0.0.0.0:* LISTEN 19867/python
```

Após reiniciar os serviços nova-api e nova-objectstore, a imagem deve ser publicada normalmente.

```
# /etc/init.d/nova-api restart;/etc/init.d/nova-objectstore restart
```

B.4 POX e DCPortals

O POX é um *Network Hypervisor* que permite que você crie controladores OpenFlow para utilizar o conceito de redes definidas por software. O DCPortals é o módulo controlador criado para implementar o isolamento de redes lógicas privativas determinado pelo projeto.

Copie o código fonte do POX com o módulo do sistema para a máquina que realiza o papel de controlador do sistema. Coloque o arquivo openstack_interface.conf no diretório `/etc/`.

Agora levante o POX com o devido controlador. O primeiro comando abaixo se refere à um switch L2 comum com aprendizado, já o segundo se refere ao controlador do DCPortals.

```
# ./pox.py forwarding.l2_learning
# ./pox.py dcportals
```

É necessário fazer os switches de cada um dos nós-compute apontarem para o controlador. O comando abaixo deve ser executado em todos os nós-compute e permite que os openvswitches de cada um seja programado pelo controlador utilizando o protocolo OpenFlow:

```
# ovs-vsctl set-controller xenbr0 tcp:10.0.254.1:6633
```

obs: No comando acima é preciso utilizar o IP do controlador do ambiente que esta sendo usado. No nosso ambiente o controlador possui o IP 10.0.254.1.

É preciso editar o arquivo de configuração /etc/openstack_interface.conf, atualizando os dados do controlador e dos nós compute. Abaixo será descrito um exemplo do arquivo de configuração /etc/openstack_interface.conf. A numeração não faz parte do arquivo, apenas foi adicionada para simplificar a explicação.

Exemplo de configuração do arquivo "openstack_interface.conf":

```
1  [mysql\_server]\\
2  mysql\_server\_address = 10.0.254.1\\
```

```
3  mysql\_user = nova\\
4  mysql\_user\_passwd = notnova\\
5  openstack\_dbname = nova\\
6  controller\_MAC\_address = 00:15:17:d9:74:7f\\
7  controller\_IP\_address = 10.0.50.1\\
8  network = 10.0.50.0/24\\
9  [compute\_nodes\_macs]\\
10 cloudinha2 = 00:15:17:ed:cc:01\\
11 cloudinha3 = 48:5b:39:86:ce:a4\\
12 cloudinha4 = 48:5b:39:86:cb:62\\
```

Na linha (2) do arquivo, é preciso informar o IP da máquina que será a controladora (máquina onde esta rodando o POX).

As linhas (3) e (4) são as partes do "mysql_user"e "mysql_user_passwd". Seria apenas colocar o usuário e senha criados na seção de criação do MYSQL.

Na linha (5) se refere à parte do "openstack_dbname". Coloque o nome do banco de dados criado, também, na parte de criação do MYSQL.

Na linha (6) é preciso informar o endereço MAC da interface de rede do controlador dedicada à rede de máquinas virtuais (Essa informação é encontrada com o comando ifconfig).

A linha (7) se refere ao IP da interface de rede do controlador dedicada às vm's.

A linha (8) se refere à faixa de IP reservada para as vm's

As linhas (10), (11) e (12) são os nomes/MACs das máquinas pertencentes ao grupo de nós-computes configurados para esse controlador. Os nomes devem ser resolvidos por todas as máquinas da rede, seja por um servidor DNS ou por um arquivo no /etc/hosts. Coloque os MACs relativos às interfaces de rede que cada máquina física esta ligada na bridge xenbr0 (você pode ver essas informações com o comando ifconfig).

B.5 Resultado

Se todas as etapas tiverem funcionado corretamente, é esperado que o sistema esteja funcionando com o isolamento esperado do DCPortals.

Para realizar um teste, crie uma outra chave diferente da chave “mykey” já criada e inicie diversas máquinas virtuais com cada chave. Em seguida, a partir do controlador logue utilizando ssh em alguma máquina virtual de uma das duas redes. Essa máquina só deve conseguir se comunicar com máquinas virtuais da mesma rede e o controlador, inclusive em pacotes de broadcast.

Ao se capturar o tráfego fora das máquinas virtuais, somente os endereços MAC das máquinas físicas deve ser identificado. Enquanto que dentro das máquinas virtuais os MACs das máquinas virtuais devem estar corretos.