Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Um estudo analítico sobre funcionamento de uma Rede Definida por Software

Pedro Henrique Cruz Amaral

JUIZ DE FORA
AGOSTO, 2013

Resumo

Redes de Computadores podem ser consideradas soluções tecnológicas de grande sucesso, alcançando larga utilização ao redor do mundo, principalmente, através de uma representante de destaque: a Internet. Entretanto, a crescente difusão desta tecnologia, exige melhor desempenho e confiabilidade de sua arquiquetura já preocupantemente comprometida nestes aspectos, além de representar também relevante problema em termos de alterações da sua estrutura, pois acarretariam impactos de alta complexidade.

No contexto apresentado, surge, recentemente, o paradigma de Redes Definidas por Software, uma ideia inovadora que despertou o interesse de muitos pesquisadores e empresários da área. A relativa simplicidade de sua implementação, o baixo impacto sobre o *hardware*, somados à real possibilidade de solução para vários problemas presentes nos sistemas de rede atuais, colocam este paradigma como principal proposta para o que convencionou-se denominar "Internet do Futuro".

Contudo, pouco foi de fato desenvolvido sobre este paradigma e muitas ainda são as dúvidas em torno das possibilidades que ele pode oferecer. O trabalho aqui resumido desenvolve uma aplicação de Redes Definidas por Software em um ambiente simulado, detalhando as necessidades e idiossincrasias de sua arquitetura e demonstrando alguns de seus benefícios.

Na simulação desenvolvida, observou-se o poder de programação sobre os fluxos de pacotes que a arquitetura oferece, executando-se, por exemplo, descartes de pacotes do tipo *ICMP* pelos *switches OpenFlow*, programados através do controlador da rede.

Palavras-chave: Redes Definidas por Software (SDN), virtualização, protocolo *OpenFlow*, *Open vSwitch*, *POX*, *OpenStack*, *Xen Hypervisor*

Sumário

1 Introdução

As redes de computadores são hoje de fundamental importância para o cotidiano de empresas e pessoas ao redor do mundo. É raro encontrar negócios ou situações que não necessitem de qualquer interação com esta tecnologia, seja ela de grande, médio ou pequeno porte. A Internet, por exemplo, alcançou sucesso absoluto e abrange grande parte das transações financeiras, de *marketing* e até mesmo das relações interpessoais. Esta abrangência é tal que seria impensável para os dias de hoje, abrir um empreendimento que não se relacionasse, de alguma forma, com este meio de comunicação.

Como consequência desta grande difusão, surgiram muitos desafios para a comunidade científica. A medida que aumentam a complexidade e criticidade de performance das redes de computadores, aumenta também a necessidade de melhores dispositivos de *hardware*, protocolos e meios de transmissão. No entanto, a enorme utilização das tecnologias e padronizações atuais limitam consideravelmente as possibilidades de alteração em sua infraestrutura, pois acarretariam, na maioria dos casos, alto investimento financeiro e enorme complexidade ao processo de mudança, que teria alcance mundial. Um exemplo da referida complexidade pode ser observado no processo de transição dos protocolos IPv4 para IPv6, discutido detalhadamente em (Waddington e Chang, 2002). Além disso, essa inflexibilidade força pesquisadores a executarem e testarem seus experimentos de novas tecnologias em ambientes virtualizados, que são abstrações simplificadas da realidade e não conseguem fornecer fidedignamente a situação dos problemas em redes reais.

A alta complexidade de tarefas como organizar e gerenciar uma rede de computadores atualmente, tornou-se problema preocupante para muitos administradores, principalmente com a crescente utilização de máquinas virtualizadas. Exemplo relevante pode ser observado nos sistemas de rede de *datacenters* que utilizam a tecnologia de virtualização de computadores para prover recursos computacionais sob demanda, onde o cliente pode ter acesso a uma estrutura de redes e máquinas privada, exclusiva e remota, de acordo com sua necessidade. *Datacenters* deste tipo costumam ter centenas de máquinas reais, com dezenas de *switches* conectando-as, além de milhares de máquinas virtuais, com centenas de *switches* virtuais interligando-as. É preciso gerenciar não somente esta enorme estrutura, mas também cada rede e máquina de cada cliente, isolando os fluxos de pacotes para fornecer-lhes privacidade e segurança. Normalmente, nas empresas que oferecem este tipo de serviço, como *UOL* e *Amazon*, utiliza-se *VLANs*. Mas esta solução exige dispendiosa configuração sobre cada *switch* e ainda apresenta uma limitação prática no tamanho do cabeçalho à disposição, como discutido em (Nunes et al., 2013).

O paradigma de Redes Definidas por Software (*Software Defined Networks* ou, simplesmente, *SDNs*) foi recentemente idealizado e já recebe grande atenção. Propõe uma estrutura de rede programável com uma visão globalizada que permite intervenções e manipulações em seu funcionamento. Apoiando-se comumente sobre o protocolo *OpenFlow*, que especifica uma interface de comunicação direta com os comutadores de pacotes (*switches*), ele consegue oferecer uma possibilidade de solução elegante, eficiente e com baixo impacto sobre as tecnologias de rede em uso, abrindo caminho para novas formas de tratamento de fluxos de pacotes que podem viabilizar muitos benefícios em termos de gerência, organização, desempenho e segurança de redes de computadores.

1.1 Contextualização

Inúmeras iniciativas, ao longo dos últimos anos, visaram atender a crescente necessidade por melhores formas de organizar e administrar os sistemas de redes de computadores. Para alcançar esta melhora, no entanto, é preciso que cada dispositivo de rede possua maiores recursos que possibilitem sua programação. Em (Tennenhouse e Wetherall, 2002), apresenta-se as *active networks*, que oferecem grande poder de programação dos dispositivos da rede, executando até mesmo cálculos sobre os dados dos usuários. Entretanto, mesmo com o visível potencial de melhora no que se propunha, esta idéia teve pouca aceitação devido à necessidade de alteração profunda dos dispositivos de encaminhamento de rede para que pudessem se tornar programáveis.

Mais recentemente, encontramos iniciativas como o *PlanetLab* (Peterson e Roscoe, 2006) e o *GENI* (Turner, 2006; Elliott e Falk, 2009). A primeira propõe um sistema de rede em escala mundial, construído e mantido por pesquisadores de 25 países e usado como ambiente teste para aplicações e serviços distribuídos geograficamente. Embasa-se este em um conceito de virtualização distribuída, onde várias máquinas virtuais são agrupadas e tratadas como uma única entidade do sistema, facilitando a organização e manipulação dos fluxos de pacote na rede. A segunda aposta, similarmente, em recursos de virtualização para facilitar a transição gradual para novas tecnologias. Ambas são consideradas boas soluções a longo prazo, mas enfrentam problemas de performance para manter o nível de exigência dos padrões atuais de serviços largamente difundidos na grande rede.

Assim, a atenção dos pesquisadores voltou-se para soluções a curto prazo que mantivessem a estrutura do *hardware* pouco alterada e conseguissem garantir um nível aceitável de performance. Como o gargalo dos elementos centrais de rede situa-se no processo de encaminhamento de pacotes, focou-se principalmente em facilitá-lo com intervenções via *software*. Propostas deste tipo fundamentaram-se fortemente em uma tecnologia já utilizada que emprega rótulos programáveis na identificação de fluxos, popularizada principalmente pelo *MPLS (Multi-protocol Label Switching)* (Davie e Farrel, 2008; Kempf et al., 2011).

Atualmente, a iniciativa nesse sentido com melhor aceitação baseia-se em interfaces de comunicação que utilizam o protocolo *OpenFlow* (McKeown et al., 2008). Este protocolo estabelece regras para comunicação direta com os comutadores de rede que o implementam, tornando possível programá-los através de uma camada extra de *software*. Esta camada pode ainda armazenar informações sobre a rede e sobre os fluxos de pacote que por ela trafegam. Com esta abordagem, a tarefa de encaminhamento permanece eficiente pois continua sendo executada pelo *hardware* de alto desempenho, mas as decisões acerca dos fluxos passam para um nível superior, permitindo uma programação interventiva que pode proporcionar muitas facilidades para a administração e organização deste sistema. As referidas soluções ficaram conhecidas como Redes Definidas por Software (*SDNs*) e serão tema de pesquisa do presente trabalho.

1.2 Justificativa

As *SDNs* tornaram-se rapidamente alvo de muitos estudos por parte de pesquisadores da área a qual se relaciona. O *Open Networking Summit*[1], importante encontro entre pesquisadores e profissionais da indústria, organizado pela Universidade de Stanford e a empresa Nicira, é focado principalmente neste tipo de solução, e teve uma procura por inscrições muito acima da esperada, resultando em mais de 250 nomes de participação

[1] http://opennetsummit.org, acessado em julho de 2013

efetiva e em outros 250 para uma lista de espera (Guedes et al., 2012). Publicações em conferências da área que abordem algum aspecto de *SDNs* são cada vez mais numerosas. Em 2012, dez trabalhos apresentados no Simpósio Brasileiro de Redes de Computadores tinham este tipo de enfoque (Guedes et al., 2012).

Isto se deve, principalmente, ao caráter promissor destas soluções, que combinam real poder de programação sobre os fluxos, possibilidade de inovações em vários aspectos e relativa simplicidade de implantação. *SDNs* constituem muito mais do que, simplesmente, as possibilidades trazidas pela especificação do padrão *OpenFlow*, pois permitem desenvolver aplicações que manipularão vários aspectos dos nós de uma rede, com benefícios impensáveis até então. Abrem-se portanto muitas oportunidades para desenvolvimento de trabalhos relacionados ao tema.

Entretanto, pouco foi desenvolvido de fato sobre o paradigma de *SDN* até o momento e, consequentemente, há ainda muitos aspectos desconhecidos sobre o assunto. Existem gargalos significativos em um sistema como esse? Eles são comprometedores em um futuro próximo? A performance, aspecto crítico das redes atuais, será comprometida sob alguma circunstância? Haverá possibilidade de melhora na segurança? A maioria destas perguntas permanece ainda com pouca ou nenhuma elucidação.

Assim, o trabalho aqui desenvolvido visa contribuir com o desenvolvimento das *SDNs*, permitindo melhor compreensão de questões ainda "obscuras", principalmente aquelas relativas a implementação deste inovador modelo de redes.

1.3 Objetivos

1.3.1 Objetivos Gerais

Os objetivos gerais deste trabalho são: fornecer uma apresentação do paradigma de *SDNs*, identificando suas características, componentes principais, capacidades e limitações; mostrar a lógica de funcionamento e troca de mensagens do protocolo *OpenFlow*; estender o entendimento acerca do *Network Hypervisor POX* e o que é necessário para codificar aplicações em seu contexto; identificar as dificuldades e necessidades para implementação de uma *SDN*.

1.3.2 Objetivos Específicos

Objetiva-se especificamente que este trabalho:

- Apresente possibilidades acerca da arquitetura de uma *SDN*;
- Mostre como utilizar o controlador *POX* para implementar soluções neste contexto;
- Forneça uma ideia básica de como implementar uma *SDN*, da instalação e configuração de seus principais componentes;
- Sirva como referência e facilitador para futuros trabalhos de implementação sobre o paradigma tratado;
- Por fim, se possível, possa identificar imperfeições e sugerir melhorias às versões dos softwares utilizados para implementar a arquitetura da aplicação desenvolvida.

1.4 Metodologia

Este trabalho fará uma contextualização do cenário atual das redes de computadores, enfatizando as propostas recentes de solução das imperfeições observadas em sua arquitetura e as dificuldades impostas pela criticidade de seu funcionamento e completa difusão de suas tecnologias e padronizações por todo o mundo.

Abordará também a promissora ideia de Redes Definidas por *Software*, desbravando suas possibilidades e descrevendo seu funcionamento e composição, além de discutir a tecnologia na qual se apoia, o *OpenFlow*.

O trabalho implementará os detalhes da arquitetura deste modelo de redes, utilizando ambiente virtual simulado e desenvolvendo aplicação didática e relevante. Desta forma, pretende-se estender o entendimento do paradigma e confirmar as afirmações observadas nas referências quanto a facilidade de implantação.

1.5 Estrutura do Trabalho

Baseado no que foi anteriormente apresentado, o presente trabalho organiza-se como detalhado a seguir:

1. Capítulo introdutório finalizado pela presente seção, cujo conteúdo baseou-se em discussões iniciais que descreveram e ambientaram o tema, além de apresentar características primárias do trabalho desenvolvido;

2. Redes Definidas por *Software* - dedica-se aqui o esforço para fundamentar o entendimento do paradigma de *SDNs*, abordando em detalhes cada componente, seu papel na arquitetura e a forma como estes se inter-relacionam;

3. Desenvolvimento - apresenta o processo que resultou na implementação do presente trabalho, passando por aspectos de instalação e configuração de cada componente, descrição do ambiente de *hardware* e *software* utilizado e da aplicação codificada;

4. Conclusão - finaliza-se por tal cada discussão abordada, apresentando as considerações finais, pontos de vista do autor e ideias para trabalhos futuros.

2 Redes Definidas por *Software*

2.1 Definição

Redes Definidas por *Software* são redes programáveis que possuem componentes e aplicações de *software* capazes de compor uma separação entre o plano de controle e o plano de dados de sua estrutura. O plano de controle é responsável pelas tomadas de decisões acerca dos fluxos baseado em informações da rede armazenadas em uma base de dados, acessada pela aplicação desenvolvida para confeccionar tabelas de encaminhamento nos dispositivos de rede. Já o plano de dados é responsável pura e simplesmente pela comutação e repasse dos pacotes de rede (Guedes et al., 2012). Alguns dos dispositivos da rede participam ativamente de ambos os planos definidos anteriormente: são aqueles que possuem uma interface de comunicação direta com as *APIs* implementadas, as quais situam-se em nó(s) conhecido(s) como controlador(es) da *SDN*.

Para que esta comunicação ocorra, é necessário algum tipo de padronização, que poderia partir do ponto de vista das *APIs* desenvolvidas. Mas esta abordagem seria limitada pois as ações do plano de controle estariam restringidas àquelas previstas pelo *software* do dispositivo de rede. Surge assim o *OpenFlow*, que especifica um protocolo para esta interface de comunicação entre o plano de controle e o plano de dados desta arquitetura (Guedes et al., 2012). Detalhes sobre o universo *OpenFlow* serão discutidos na terceira seção deste capítulo.

Portanto, as *SDNs* integram a abordagem pluralista de arquiteturas de rede que conceituam o que convencionou-se denominar "Internet do Futuro". Apoiada sobre a tecnologia *OpenFlow*, o paradigma de *SDN* mostra-se uma proposta promissora, que provê alto desempenho, controle sobre a rede e grande flexibilidade para seu núcleo.

2.2 Componentes

As *SDNs* possuem uma arquitetura característica que é composta por diversas entidades de *software*, cada uma com um papel bem definido. A seguir, discute-se sobre as mais importantes, analisando as possibilidades de disposição e organização das mesmas.

2.2.1 Elementos de Comutação Programáveis

Os elementos de comutação programáveis constituem a premissa básica do paradigma de Redes Definidas por *Software*. É através desta programação que as aplicações implementadas no controlador conseguem executar manipulações simples nos pacotes trafegantes, construindo uma estrutura de rotas simplificadas para o *hardware* do dispositivo, baseando-se em conceitos de fluxos definidos pela aplicação controladora (Guedes et al., 2012).

Comutadores comuns realizam o encaminhamento de pacotes de acordo com uma sequência bem definida de ações:

1. O pacote recebido em uma das interfaces é inspecionado;
2. Uma busca pelo endereço extraído do cabeçalho do pacote é executada na tabela de encaminhamento;

3. Em caso de sucesso, o comutador identifica para qual de suas interfaces deve enfileirar o pacote tratado e o faz. Em caso de inssucesso, normalmente o pacote é encaminhado a todas as interfaces do comutador (a depender de prévia configuração).

A diferença existente entre o processo de encaminhamento em comutadores comuns e comutadores programáveis é que nestes são disparados alguns eventos intercalados entre as ações anteriormente descritas, desencadeando uma troca de mensagens entre o comutador programável e o controlador da rede *SDN*, o que permite a este último manipular os pacotes de acordo com as decisões da sua aplicação. Percebe-se portanto que este tipo de abordagem fornece grande poder de controle sobre a rede.

É válido salientar que os comutadores programáveis não precisam ser, necessariamente, dispositivos reais. Como exemplo, na implementação desenvolvida em (Nunes et al., 2013), nenhum dispositivo físico (comutador) possuia uma interface de comunicação capaz de executar a troca de mensagens anteriormente descrita. As programações sobre aquela *SDN* eram, portanto, direcionadas apenas a comutadores virtuais especialmente configurados, do tipo *Open vSwitch*[1], que implementam o protocolo *OpenFlow*. Estes comutadores virtuais conectavam as várias máquinas virtuais presentes na estrutura do *datacenter* enquanto os comutadores físicos conectavam as máquinas físicas somente, abstraindo-se destes toda a complexidade da aplicação de controle em funcionamento neste modelo.

2.2.2 Controlador

Definida a interface de comunicação com os comutadores de rede programáveis, pode-se codificar aplicações que se comuniquem diretamente com aquela, operando sobre cada comutador separadamente. Essa abordagem, entretando, tornaria o *software* resultante muito dependente da implementação do *hardware* do dispositivo, desfavorecendo consideravelmente a flexibilidade, manutenção e portabilidade do sistema.

Para conseguir abstrair de forma eficiente esta comunicação e interdependência, desenvolveu-se um tipo de entidade de *software* que pudesse tratar esta complexidade, levando ao desenvolvedor uma visão de mais alto nível. Operando como facilitador e controlador base desta comunicação, este *software* assemelha-se a um sistema operacional, que isola detalhes intrínsecos de cada componente (Guedes et al., 2012).

Assim, um controlador (também chamado de *Network Hypervisor*) é um componente que pode concentrar a comunicação entre as aplicações e os comutadores programáveis, fornecendo inclusive uma visão global da rede, que facilita a identificação e solução de problemas, além da tomada de deciões (Casado et al., 2010). Vale frisar que esta abstração de visão global pode ser implementada de forma distribuída, ou seja, a ideia de centralização e unicidade do controlador limita-se a uma visão de conveniente simplicidade (Guedes et al., 2012).

2.2.3 Aplicações

Apresentado o controlador *SDN*, que pode ser comparado a um "Sistema Operacional da rede", compreende-se mais facilmente como as ações de decisão serão programadas nos comutadores de rede. As aplicações desenvolvidas são mantidas em execução no controlador e, fazendo uso de informações sobre recursos da rede armazenadas possivelmente em uma base de dados comum, conseguem desempenhar os mais variados tipos de controle

[1] http://www.openvswitch.org, acessado em agosto de 2013

no tráfego de pacotes, como, por exemplo, "implementar soluções de roteamento especialmente desenhadas para um ambiente particular, controlar a interação entre os diversos comutadores, oferecendo para os computadores a eles ligados a impressão de estarem ligados a um único *switch*, ou a um único roteador IP" (Guedes et al., 2012).

Estas aplicações podem ser, portanto, especialmente desenvolvidas para a rede tratada, possibilitando configurações bem particulares. Podem estar espalhadas pelos diversos controladores presentes na rede, o que permite também melhor divisão e aproveitamento dos recursos computacionais disponíveis.

A variedade de benefícios que podem ser proporcionados pela implementação de aplicações neste contexto é com certeza algo pouco explorado. Pode-se obter soluções para muitos problemas presentes hoje nos sistemas de rede computacionais, como, por exemplo, na detecção de ataques do tipo *Distributed Denial Of Service* (*DDOS*) abordada em (Braga et al., 2010). Identifica-se portanto muitas possibilidades para desenvolvimento de pesquisas e projetos futuros.

2.3 *OpenFlow*

As *SDNs*, como visto anteriormente, são capazes de controlar o plano de encaminhamento de pacotes da rede, possibilitando abstrações e programações sobre seu funcionamento que podem simplificá-lo e torná-lo mais eficiente. Entretanto, para que seja possível o referido controle, é necessária a definição de uma interface de comunicação que permita às aplicações do controlador "ensinarem"os comutadores de rede acerca de suas regras e proposta de tratamento dos pacotes. Grande motivadora deste paradigma de redes, a interface *OpenFlow* (McKeown et al., 2008) tornou-se, desde o início da idealização desta arquitetura, a mais promissora tecnologia para prover a comunicação descrita.

Desenvolvido pela Universidade de Stanford, um de seus principais objetivos é atender a demanda por validação de novas propostas de arquitetura e protocolos de rede sobre equipamentos comerciais. Através do *OpenFlow*, pode-se implementar uma tecnologia capaz de abrigar testes em cenário real sem prejudicar ou comprometer a rede de produção, pois consegue-se separar desta a rede de testes, mantendo ambas em funcionamento isolado e paralelo (Guedes et al., 2012).

Esta seção visa apresentar as principais características do universo *OpenFlow*, detalhando o funcionamento da arquitetura que o utiliza (*SDNs*) e a especificação de seu protocolo de comunicação.

2.3.1 Padrão *OpenFlow*

O *OpenFlow* é um padrão aberto que consiste de uma interface de comunicação capaz de prover ao desenvolvedor controle direto sobre os elementos de encaminhamento de pacotes. Foi proposto por pesquisadores que desejavam desenvolver seus experimentos em cenários reais de rede, sem que estes testes atrapalhassem o funcionamento da rede de produção. Com isto, conseguiriam prover maior confiabilidade a suas propostas de novas tecnologias junto à indústria (McKeown et al., 2008).

Como o *OpenFlow* pode abranger todos os comutadores de pacote da rede especialmente estruturada, torna-se possível executar o isolamento entre tráfego de produção e experimental, sem que este comprometa o bom funcionamento daquele. Esta característica dá grande credibilidade ao padrão, visto que facilita o desenvolvimento e teste de novas propostas tecnológicas sem ferir a criticidade e necessidade de confiabilidade dos serviços largamente difundidos na Internet.

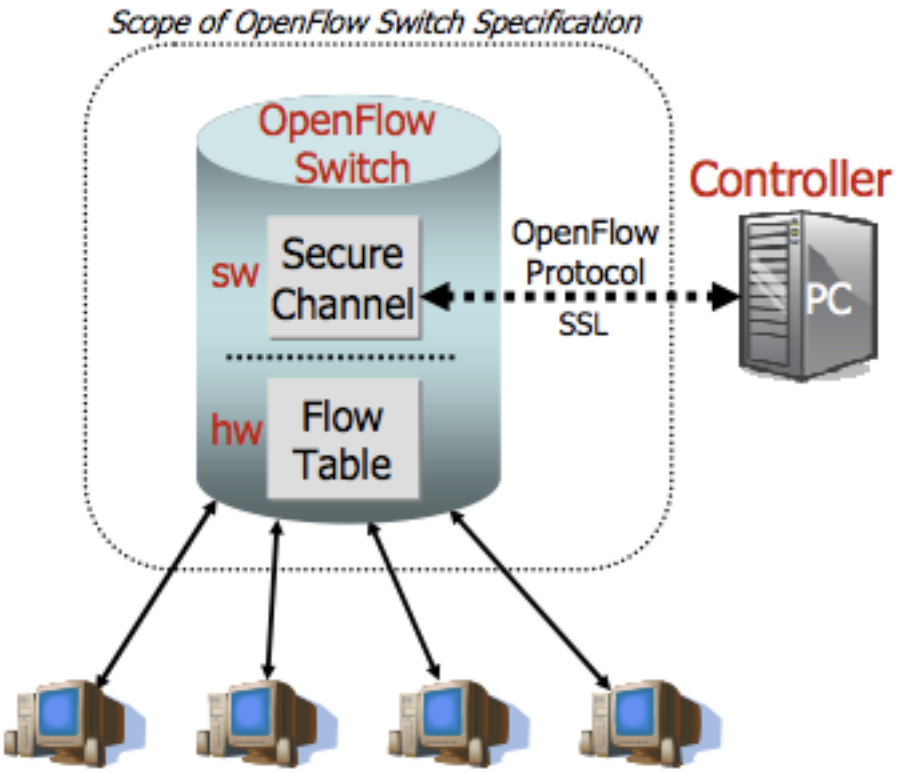

Figura 2.1: Diagrama de funcionamento do padrão *OpenFlow* (Nunes et al., 2013)

Outro grande trunfo deste padrão é manter o projeto do equipamento de *hardware* de encaminhamento atual quase intacto, sendo apenas necessário adicionar as funcionalidades de comunicação *OpenFlow*. Isto é animador para a indústria, visto que os equipamentos continuarão com baixo custo e semelhante performance, permitindo sua adoção por parte dos administradores de redes.

Oferecendo uma separação entre o plano de controle e o plano de dados da rede, o padrão *OpenFlow* é forte proposta a ser adotada em curto tempo. O plano de dados fica responsável pelo encaminhamento dos pacotes, podendo inclusive ser implementado por *switches* e roteadores comuns, como no sistema baseado em *SDNs* apresentado em (Nunes et al., 2013). O plano de controle fica, por sua vez, responsável pela programação das tabelas de encaminhamento dos dispositivos de rede que implementam o padrão aqui discutido, atuando também na manipulação de pacotes e na definição de possíveis fluxos, utilizando-se da visão global da rede fornecida pelo controlador *SDN*.

Deve-se ressaltar que a troca de mensagens entre controlador *SDN* e cada comutador *Openflow* é efetuada através de um canal seguro de comunicação, implementado normalmente (não obrigatoriamente) com a utilização do *SSL* (*Secure Socket Layer*[2]).

2.3.2 *API OpenFlow*

A *API OpenFlow*, definida em seu protocolo, permite acesso às funcionalidades de manipulação das tabelas de encaminhamento dos comutadores de pacote, independente de particularidades de cada fabricante. As entradas das tabelas contêm regras de encaminhamento que podem utilizar até 10 campos dos cabeçalhos de pacote para identificação de condutas sobre ele. Caso ocorra casamento completo das entradas com os dados obtidos

[2]RFC 6101, disponível em http://tools.ietf.org/html/rfc6101 - acessado em julho de 2013

do pacote, ações anteriormente definidas podem ser executadas (Guedes et al., 2012).

Tais ações podem ser classificadas dentre as seguintes categorias:

1. Executar o encaminhamento de pacotes a alguma interface específica do comutador, permitindo roteamento de pacotes pela rede.
2. Alterar convenientemente campos contidos no cabeçalho dos pacotes.
3. Descartar o pacote.
4. Encapsular e enviar pacotes de um fluxo para o controlador, a fim de que possam ser processados para tomadas de decisões.

Assim, esta conexão direta com o controlador permite processamentos antes impossíveis, capazes de organizar, identificar e coordenar as tomadas de decisões sobre um fluxo, simplesmente atribuindo novas entradas à tabela de encaminhamento.

3 Desenvolvimento

Os surgimentos do protocolo *OpenFlow* e dos *Network Hypervisors* representaram uma possibilidade real de evolução para vários sistemas de rede de computadores, fornecendo meios para que problemas atuais pudessem ser tratados de forma simples e elegante. Entretanto, mesmo que já existam *switches OpenFlow* no mercado, eles são ainda poucos e caros. Além disto, muitas redes complexas possuem infraestrutura dotada de muitos *switches ethernet* comuns, tornando dispendiosa a troca de todos eles.

Utilizando a tecnologia de virtualização, tão comum atualmente em infraestruturas de rede, é possível adotar *switches* virtuais para facilitar a organização desta. Esta abordagem é encontrada em muitas aplicações computacionais, como nas arquiteturas de *datacenters* virtualizados abordadas por (Nunes et al., 2013). Fazendo com que estes *switches* virtuais sejam compatíveis com a tecnologia *OpenFlow*, torna-se possível implementar uma *SDN*.

Este trabalho implementa uma *SDN* utilizando recursos de virtualização, integrando a ferramenta para gerenciamento de computação em nuvem *OpenStack* com o *Network Hypervisor POX*, e programando *switches* virtuais *OpenvSwitch*, que suportam o padrão *OpenFlow*. Neste capítulo são apresentados os detalhes da implementação da rede e das configurações nas ferramentas utilizadas.

3.1 Arquitetura da Aplicação

O controlador *POX* possui uma *API* simples, possibilitando a implementação de muitos módulos que podem manipular os comutadores programáveis da rede. São estes módulos os responsáveis pela criação de mensagens *OpenFlow* que podem programar os *OpenvSwitches* como desejado, podendo ainda utilizar de informações sobre as redes virtuais através da base de dados do *OpenStack*. É este quem controla o monitor de máquinas virtuais *Xen* presente em cada máquina da rede (exceto naquela com o papel de controlador na *SDN*).

As máquinas virtuais que compõem a rede são instanciadas a partir da interface do *OpenStack*. Ele envia recursos para que o *Xen* dispare a máquina virtual e conecte-a ao *OpenvSwitch* configurado. Assim que esta nova máquina inicia uma comunicação na rede, os módulos implementados sobre o *POX* já são capazes de manipular as trocas de pacote pelos eventos disparados no *OpenvSwitch* e identificados pelo controlador *POX*.

3.2 Monitor de Máquinas Virtuais *Xen*

O *Xen* (Barham et al., 2003) é um monitor de máquinas virtuais suportado por uma distribuição *Linux* muito popular, o *Ubuntu*, sendo, por isso, escolhido para o desenvolvimento deste trabalho. Engloba conceitos de para-virtualização e virtualização completa e possui um mecanismo de controle de recursos com desempenho bastante satisfatório.

Este software possui suporte oficial por parte do *OpenStack*, o que garante certa longevidade às aplicações que utilizam os recursos destas ferramentas em conjunto. Possui versões para vários sistemas operacionais do mercado, permitindo o controle e manipulação de recursos de *hardware* em máquinas *Linux*, *BSD* e *Windows* (Barham et al., 2003).

Nesta seção são discutidos detalhes de sua instalação e configuração para integração com uma implementação virtualizada de *SDN*.

3.2.1 Instalação no *Ubuntu*

Para o desenvolvimento do presente trabalho, utilizou-se a versão 12.04.2 do *Ubuntu Server*. Nesta versão, o *Hypervisor Xen* faz parte dos pacotes de instalação do sistema operacional, o que pode não acontecer em versões mais antigas. Assim, para a instalação do software nas máquinas comuns da rede (denominadas nós-compute), basta um comando simples:

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

Destaca-se que esta implementação foi desenvolvida sobre máquinas com processador *Intel*. Caso o ambiente seja diferente, é preciso escolher a *release* mais adequada do *Xen*.

3.2.2 Configurações no *Xen*

Para o que *Xen* seja iniciado junto com o *kernel* do *Linux*, é necessário adicionar uma entrada no *bootloader* do sistema. Para tal, deve-se copiar do arquivo `'/boot/grub/grub.cfg'` o bloco de comandos 'menuentry' do *kernel* relativo ao *Xen*, normalmente nomeado por algo como `'Ubuntu GNU/Linux, with Xen 4.1 and Linux VERSAO_DO_KERNEL'`, e colar ao final do arquivo `'/etc/grub.d/40_custom'`. Em seguida, é preciso fazer com o que esta entrada seja a padrão no *boot*, alterando em `'/etc/default/grub'` o valor da variável `'GRUB_DEFAULT'`, deixando-a parecida como abaixo:

```
GRUB_DEFAULT='Ubuntu GNU/Linux, with Xen 4.1 and Linux
   VERSAO_DO_KERNEL'
```

Para concluir esta configuração, deve-se executar o comando `'# update-grub'`. Assim, com a reinicialização do sistema, o *hypervisor Xen* será carregado. Ele pode também ser carregado através do comando `'# /etc/init.d/xend start'`. Pode-se conferir se o monitor de máquinas virtuais *Xen* está em operação no sistema, executando o comando `'# xm info'`, que retornará, em caso de funcionamento correto do *software*, várias informações sobre a execução e configuração do mesmo no sistema operacional.

A interface de rede e o *bridge* necessário para a conexão com as máquinas virtuais devem ser configurados no arquivo `'/etc/network/interfaces'`, substituindo qualquer outra pela seguinte:

```
1 auto xenbr0
2 iface xenbr0 inet static
3      bridge_ports ethX
4      bridge_stp off
5      bridge_maxwait 0
6      bridge_fd 0
7      address ENDERECO_IP_FIXO_DA_MAQUINA
8      netmask MASCARA_REDE
9      broadcast ENDERECO_IP_BROADCAST
```

Na listagem anterior, os termos em caixa alta devem ser substituídos pelos valores de configuração do ambiente alvo, incluindo o termo `'X'` em `'ethX'`, o qual deve ser trocado pelo número identificador da porta disponível para a comunicação com a rede de máquinas virtuais.

É preciso editar o arquivo `'/etc/xen/xend-config.sxp'` para habilitar a interface 'http' do *Xen* , descomentando e alterando o valor das linhas como nos blocos abaixo:

```
1 (xend-http-server yes)
2 #(xend-unix-server no)
3 #(xend-tcp-xmlrpc-server no)
4 #(xend-unix-xmlrpc-server yes)
5 (xend-relocation-server yes)
6 #(xend-relocation-ssl-server no)
7 #(xend-udev-event-server no)
```

Em outro ponto do mesmo arquivo:

```
1 # Port xend should use for the HTTP interface, if xend-http-
     server is set.
2 (xend-port 8000)
3
4 # Port xend should use for the relocation interface, if xend-
     relocation-server is set.
5 (xend-relocation-port 8002)
```

O arquivo `'/usr/share/pyshared/nova/virt/libvirt.xml.template'` também deve ser modificado para corrigir uma incompatibilidade do *libvirt* com as máquinas *Xen*. São duas as modificações necessárias já inseridas nas listagens abaixo. A primeira em:

```
1 #else
2      #if $type == 'xen'
3           #set $root_disk_bus = 'scsi'
4           #set $ephemeral_disk_bus = 'scsi'
5                <type>linux</type>
6                #set $root_device_name = '/dev/sda'
7                <root>${root_device_name}</root>
```

A segunda em:

```
1 #else
2      #if $getVar('kernel', None)
3           <kernel>${kernel}</kernel>
4           #if $type == 'xen'
5                <cmdline>ro</cmdline>
6           #else
7                #set $root_device_name = $getVar('root_device_name
                    ', '/dev/vda')
8                <cmdline>root=/dev/sda console=ttyS0</cmdline>
```

3.3 *OpenvSwitch*

O OpenvSwitch (Pfaff et al., 2009) é um switch virtual de código livre, capaz de conectar dezenas de máquinas virtuais e apresentando desempenho satifatório. Foi implementado em *software*, com o plano de dados dentro do *kernel* do sistema operacional *Linux*, enquanto seu plano de controle é acessado pelo espaço do usuário.

Apesar de ser dotado das funcionalidades da tecnologia *OpenFlow*, o *OpenvSwitch* é capaz de operar como um *switch ethernet* comum, simplificando sua integração com o *kernel* do *Linux*, uma vez que não compromete o funcionamento de uma rede padrão.

Como este *switch* conectará somente as máquinas virtuais da rede, ele deverá ser instalado somente nos nós-compute, através do comando a seguir:

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

Para que o módulo deste *switch* possa substituir aquele presente no *kernel* do *Linux*, é preciso descomentar e modificar no arquivo `'/etc/default/openvswitch-switch'` uma variável de configuração, deixando-a como indicado abaixo:

```
BRCOMPAT=yes
```

Visando-se o bom funcionamento do *OpenvSwitch*, o módulo *bridge* padrão do *Linux* deve ser removido através do comando:

```
# rmmod bridge
```

Deve-se, em seguida, criar a *bridge* que será utilizada na rede, adicionando também uma porta 'eth' que esteja disponível para comunicação com as máquinas virtuais. Para tal, os comandos são:

```
# ovs-vsctl add-br xenbr0

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

3.4 *OpenStack*

O *OpenStack*[1] é um gerenciador para ambiente virtualizado que permite distribuir recursos, controlar acessos, configurar máquinas virtuais e reunir informações sobre a nuvem. Esta ferramenta consegue prover relevante simplificação das operações de administração e organização do ambiente virtualizado, abstraindo toda a complexidade destas no que se refere a infraestrutura física e disponibilização de recursos. O *software* é composto por três componentes principais descritos a seguir:

- *Compute*: gerenciador da infraestrutura da nuvem, onde manipulam-se as instâncias de máquinas virtuais de um usuário, configuram-se as redes destas ou de projetos, que são aglomerações de instâncias voltadas ao mesmo fim.

[1] http://www.openstack.org/, acessado em agosto de 2013

- *Object Storage*: sistema de armazenamento de objetos de alta capacidade com redundância e tolerância a falhas. É capaz de gerar *backup* de dados, armazenar dados estáticos secundários ou terciários, prover flexibilidade e elasticidade para armazenamento baseado em nuvem de computação, entre outros serviços.

- *Image Service*: aplicação de pesquisa e recuperação para imagens de máquinas virtuais.

A *API* do *OpenStack* fornece uma abstração baseada em comando simples e níveis de usuários. Usuários administradores possuem privilégios como criar projetos, determinar credenciais, publicar imagens, manipular máquinas virtuais, etc. Usuários comuns possuem privilégios limitados, baseados principalmente em acessar máquinas virtuais, a depender do que seja estabelecido pelo administrador. Informações sobre toda a nuvem são armazenadas em um banco de dados *MySQL*[2] localizado no controlador da nuvem, mas são também acessíveis a todos os nós que utilizam o componente *Compute*.

O *OpenStack* tem recebido grande atenção da comunidade atualmente e, por ser oferecido oficialmente pela distribuição *Linux* denominada *Ubuntu*, mostrou-se como boa opção para uso na aplicação desenvolvida.

3.4.1 Instalação e Configuração nos Nós-Compute

O serviço *nova-compute* deve ser instalado em cada nó-compute através do comando:

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

É preciso também configurá-lo de acordo com o ambiente criado, de forma similar à listagem a seguir:

```
1  -dhcpbridge_flagfile=/etc/nova/nova.conf
2  -dhcpbridge=/usr/bin/nova-dhcpbridge
3  -logdir=/var/log/nova
4  -state_path=/var/lib/nova
5  -lock_path=/var/lock/nova
6  -verbose
7  -s3_host=ENDERECO_IP_CONTROLADOR
8  -rabbit_host=ENDERECO_IP_CONTROLADOR
9  -cc_host=ENDERECO_IP_CONTROLADOR
10 -ec2_url=http://ENDERECO_IP_CONTROLADOR:8773/services/Cloud
11 -sql_connection=mysql://USUARIO_MYSQL:
      SENHA_USUARIO_MYSQL@ENDERECO_IP_CONTROLADOR/nova
12 -network_manager=nova.network.manager.FlatDHCPManager
13 -network_host=ENDERECO_IP_CONTROLADOR
14 -flat_network_bridge=xenbr0
15 -libvirt_type=xen
16 -xenapi_remap_vdb_dev=true
17 -glance_api_servers=ENDERECO_IP_CONTROLADOR:9292
18 -image_service=nova.image.glance.GlanceImageService
19 -nouse_cow_images
```

[2]http://www.mysql.com, acessado em agosto de 2013

Para o funcionamento da *SDN* é necessário fazer com que todos os nós-compute da rede apontem para o controlador. Portanto, deve-se executar um comando como o abaixo em cada nó-compute, permitindo que os *OpenvSwitches* que conectam suas máquinas virtuais sejam programáveis pelo controlador (lembrando que o termo em caixa alta na listagem a seguir deve ser substituído pelo endereço *IP* do controlador no ambiente):

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

O horário em todos os nós da rede precisam estar em sincronização constante, para que os serviços em funcionamento possam operar corretamente. Para tal, fez-se necessária a instalação e configuração do protocolo *NTP* (*Network Time Protocol*)[3], que permite uma sincronização quase exata dos relógios de um conjunto de computadores em rede de latência variável. O comando a seguir executa a instalação mencionada, que deve ser feita em cada nó-compute:

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

São necessárias modificações no arquivo `'/etc/ntp.conf'` para atualizá-lo com as informações do ambiente, deixando-o similar ao exemplo a seguir:

```
1  driftfile /var/lib/ntp/ntp.drift
2  statistics loopstacks peerstats clockstats
3  filegen loopstacks file loopstacks type day enable
4  filegen peerstats file peerstats type day enable
5  filegen clockstats file clockstats type day enable
6  server ENDERECO_IP_CONTROLADOR
7  restrict -4 default kod notrap nomodify nopeer noquery
8  restrict -6 default kod notrap nomodify nopeer noquery
9  restrict 127.0.0.1
10 restrict ::1
```

3.4.2 Instalação e Configuração no Controlador

No controlador, além de vários serviços *nova*, são necessárias instalações de algumas dependências:

```
1  # apt-get install python-software-properties
2  # apt-get install -y rabbitmq-server python-greenlet python-
      mysqldb euca2ools unzip
3  # apt-get install nova-volume nova-vncproxy nova-api nova-ajax-
      console-proxy nova-doc nova-scheduler nova-objectstore nova-
      network nova-cert nova-objectstore novnc nova-consoleauth nova-
      compute
```

Em seguida, deve-se instalar e configurar o banco de dados *MySQL*, que será usado pelo *OpenStack* para armazenar informações da nuvem:

[3]http://www.ntp.org/, acessado em agosto de 2013

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

A fim de habilitar acesso externo ao servidor *MySQL*, é necessário alterar o *bind-address* do valor '127.0.0.1' para '0.0.0.0' no arquivo `'/etc/mysql/my.cnf'`. Isto pode ser executado pelo comando:

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

A configuração do banco de dados que será acessado pelos vários serviços do *OpenStack* deve ser feita por comandos como os descritos a seguir:

```
1  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e 'CREATE DATABASE
       nova;'
2  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e 'CREATE USER
       novadbadmin;'
3  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e "GRANT ALL
       PRIVILEGES ON nova.* TO 'novadbadmin'@'%';"
4  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e "SET PASSWORD FOR '
       novadbadmin'@'%' = PASSWORD('SENHA_USUARIO_NOVADBADMIN_MYSQL');
       "
5  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e 'CREATE DATABASE
       glance;'
6  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e 'CREATE USER
       glancedbadmin;'
7  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e "GRANT ALL
       PRIVILEGES ON glance.* TO 'glancedbadmin'@'%';"
8  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e "SET PASSWORD FOR '
       glancedbadmin'@'%' = PASSWORD('
       SENHA_USUARIO_GLANCEDBADMIN_MYSQL');"
9  # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e 'CREATE DATABASE
       keystone;'
10 # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e 'CREATE USER
       keystonedbadmin;'
11 # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e "GRANT ALL
       PRIVILEGES ON keystone.* TO 'keystonedbadmin'@'%';"
12 # mysql -uroot -pSENHA_USUARIO_ROOT_MYSQL -e "SET PASSWORD FOR '
       keystonedbadmin'@'%' = PASSWORD('
       SENHA_USUARIO_KEYSTONEDBADMIN_MYSQL');"
```

O arquivo `'/etc/nova/nova.conf'` deve ser modificado para que fique com configurações similares às descritas na listagem abaixo:

```
1 --dhcpbridge_flagfile=/etc/nova/nova.conf
2 --dhcpbridge=/usr/bin/nova-dhcpbridge
3 --logdir=/var/log/nova
4 --state_path=/var/lib/nova
5 --lock_path=/run/lock/nova
6 --allow_admin_api=true
7 --use_deprecated_auth=false
8 --auth_strategy=keystone
9 --scheduler_driver=nova.scheduler.simple.SimpleScheduler
```

```
10 --s3_host=ENDERECO_IP_CONTROLADOR
11 --ec2_host=ENDERECO_IP_CONTROLADOR
12 --rabbit_host=ENDERECO_IP_CONTROLADOR
13 --cc_host=ENDERECO_IP_CONTROLADOR
14 --nova_url=http://ENDERECO_IP_CONTROLADOR:8774/v1.1/
15 --routing_source_ip=ENDERECO_IP_CONTROLADOR
16 --glance_api_servers=ENDERECO_IP_CONTROLADOR:9292
17 --image_service=nova.image.glance.GlanceImageService
18 --iscsi_ip_prefix=192.168.4
19 --sql_connection=mysql://novadbadmin:
      SENHA_USUARIO_NOVADBADMIN_MYSQL@ENDERECO_IP_CONTROLADOR/nova
20 --ec2_url=http://ENDERECO_IP_CONTROLADOR:8773/services/Cloud
21 --keystone_ec2_url=http://ENDERECO_IP_CONTROLADOR:5000/v2.0/
      ec2tokens
22 --api_paste_config=/etc/nova/api-paste.ini
23 --libvirt_type=xen
24 --libvirt_use_virtio_for_bridges=true
25 --start_guests_on_host_boot=true
26 --resume_guests_state_on_host_boot=true
27 # vnc specific configuration
28 --novnc_enabled=true
29 --novncproxy_base_url=http://ENDERECO_IP_CONTROLADOR:6080/
      vnc_auto.html
30 --vncserver_proxyclient_address=ENDERECO_IP_CONTROLADOR
31 --vncserver_listen=ENDERECO_IP_CONTROLADOR
32 # network specific settings
33 --network_manager=nova.network.manager.FlatDHCPManager
34 --public_interface=ethX
35 --flat_interface=ethY
36 --flat_network_bridge=xenbr0
37 --fixed_range=RANGE_ENDERECOS_IP_PRIVADOS
38 --floating_range=RANGE_ENDERECOS_IP_PUBLICOS
39 --network_size=TAMANHO_REDE
40 --flat_injected=False
41 --force_dhcp_release
42 --iscsi_helper=tgtadm
43 --connection_type=libvirt
44 --root_helper=sudo nova-rootwrap
45 --verbose
```

Deve-se então sincronizar o serviço com sua base de dados e disponibilizar um intervalo de endereços *IP* para associar com as instâncias que serão criadas na rede. Assim:

```
# nova-manage db sync
# nova-manage network create private --fixed_range_v4=
   RANGE_ENDERECOS_IP_PRIVADOS --num_networks=1 --bridge=xenbr0 --
   bridge_interface=ethX --network_size=TAMANHO_REDE
```

Os *nova services* devem ser reinicializados para que as configurações sejam aplicadas:

```
# restart libvirt-bin;
# restart nova-network;
# restart nova-compute;
# restart nova-api;
# restart nova-objectstore;
# restart nova-scheduler;
# restart nova-volume;
# restart nova-consoleauth;
```

No tocante, os serviços do *OpenStack* devem ser instalados e configurados separadamente, visando maior facilidade e organização de tais processos. Primeiramente, o serviço de identificação e autenticação do ambiente da nuvem denominado *Keystone* deve ser instalado:

```
# apt-get install keystone python-keystone python-keystoneclient
```

O arquivo de configuração do serviço localizado em `'/etc/keystone/keystone.conf'` deve ser modificado:

```
admin_token = admin
...
connection = mysql://keystonedbadmin:
   SENHA_USUARIO_KEYSTONEDBADMIN@ENDERECO_IP_CONTROLADOR/keystone
```

Em seguida, deve-se reinicializar o serviço para que sejam carregadas as alterações de configuração e sincronizar o banco de dados:

```
# service keystone restart
# keystone-manage db_sync
```

O serviço de imagens do *OpenStack* denominado *Glance* deve ser também instalado e configurado no ambiente. A instalação deve ser feita através do comando a seguir:

```
# apt-get install glance glance-api glance-client glance-common
   glance-registry python-glance
```

Faz-se necessária uma alteração no arquivo `'/etc/glance/glance-registry.conf'` para que o *Glance* seja capaz de acessar sua base de dados no servidor *MySQL*:

```
sql_connection = mysql://glancedbadmin:
   SENHA_USUARIO_GLANCEDBADMIN@ENDERECO_IP_CONTROLADOR/glance
```

Para que o *Glance* use o serviço de identificação e autenticação *Keystone*, é preciso adicionar as linhas abaixo ao final dos arquivos `'/etc/glance/glance-registry.conf'` e `'/etc/glance/glance-api.conf'`:

```
[paste_deploy]
flavor = keystone
```

Deve-se, em seguida, reinicializar o serviço *Glance* e sincronizá-lo com sua base de dados:

```
# restart glance-api
# restart glance-registry
# glance-manage version_control 0
# glance-manage db_sync
```

Para que o protocolo *NTP* funcione corretamente, devemos também instalá-lo e configurá-lo no controlador, o qual fará o papel de servidor de sincronização para os nós-compute. Para a instalção, basta executar o comando a seguir:

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

No controlador, o arquivo `'/etc/ntp.conf'` também deve ser modificado para que fique similar à listagem 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
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
```

3.4.3 Sincronização *NTP*

Uma primeira sincronização forçada dos relógios na rede faz-se necessária, independentemente da diferença existente entre os clientes (nós-compute) e o servidor (controlador). Após esta operação, as pequenas diferenciações que possam vir a aparecer serão corrigidas automaticamente pelo serviço *NTP*. No controlador, deve-se primeiro parar o serviço através do comando a seguir:

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

Em seguida, sincroniza-se o horário do controlador com o servidor *NTP* (caso aja algum na rede do ambiente, pode-se optar por ele substituindo o endereço de serviço no comando abaixo):

```
# ntpdate ntp.ubuntu.com
```

Basta agora reinicializar o serviço:

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

Os mesmos passos descritos devem ser repetidos em cada nó-compute da rede, mas substituindo-se o endereço do controlador para o comando de sincronização, através da sintaxe expressa a seguir:

```
# ntpdate ENDERECO_IP_CONTROLADOR
```

Assim, todas as máquinas estarão com o horário sincronizado através do protocolo *NTP*.

3.4.4 Criação e Configuração do ambiente *OpenStack*

Inicialmente, para configuração do ambiente *OpenStack*, necessita-se indentificar e autenticar os serviços utilizados pela nuvem através daquele responsável por isso, o *Keystone Service*. Portanto, os papéis, usuários e inquilinos da estrutura devem ser criados neste serviço:

```
1 # keystone tenant-create --name admin
2 # keystone tenant-create --name service
3 # keystone user-create --name admin --pass
     SENHA_USUARIO_ADMIN_OPENSTACK
4 # keystone user-create --name nova --pass
     SENHA_USUARIO_NOVA_OPENSTACK
5 # keystone user-create --name glance --pass
     SENHA_USUARIO_GLANCE_OPENSTACK
6 # keystone role-create --name admin
7 # keystone role-create --name member
```

Os usuários criados precisam ser associados a um papel em um determinado inquilino. Para esta associação, precisa-se obter o ID de cada entidade designado no *Keystone*, que pode ser obtido através dos comandos `'# keystone user-list'`, `'# keystone tenant-list'` e `'# keystone role-list'`. Com os IDs em mãos, deve-se associar todos os usuários representantes de serviços com o papel 'admin' do inquilino 'admin', como nos comandos:

```
1 # keystone user-role-add --user ID_USUARIO_ADMIN_OPENSTACK --role
     ID_PAPEL_ADMIN --tenant_id ID_INQUILINO_ADMIN
2 # keystone user-role-add --user ID_USUARIO_NOVA_OPENSTACK --role
     ID_PAPEL_ADMIN --tenant_id ID_INQUILINO_ADMIN
3 # keystone user-role-add --user ID_USUARIO_GLANCE_OPENSTACK --
     role ID_PAPEL_ADMIN --tenant_id ID_INQUILINO_ADMIN
```

Os serviços devem também ser criados para identificação com o *Keystone*, executando-se:

```
1 # keystone service-create --name nova --type compute --
     description 'OpenStack Compute Service'
```

```
2 # keystone service-create --name volume --type volume --
     description 'OpenStack Volume Service'
3 # keystone service-create --name glance --type image --
     description 'OpenStack Image Service'
4 # keystone service-create --name keystone --type identity --
     description 'OpenStack Identity Service'
5 # keystone service-create --name ec2 --type ec2 --description '
     EC2 Service'
```

A criação de *endpoints* para cada serviço precisa ser efetuada no *Keystone*. Serão necessários os IDs dos serviços anteriormente criados, podendo ser obtidos através de '`# keystone service-list`'. Para o *nova-compute*:

```
# keystone endpoint-create --region REGIAO --service_id
   ID_SERVICO_NOVA_COMPUTE --publicurl 'http://
   ENDERECO_IP_CONTROLADOR:8774/v2/$(tenant_id)s' --adminurl 'http
   ://ENDERECO_IP_CONTROLADOR:8774/v2/$(tenant_id)s' --internalurl
    'http://ENDERECO_IP_CONTROLADOR:8774/v2/$(tenant_id)s'
```

Para o *nova-volume*:

```
# keystone endpoint-create --region REGIAO --service_id
   ID_SERVICO_NOVA_VOLUME --publicurl 'http://
   ENDERECO_IP_CONTROLADOR:8776/v1/$(tenant_id)s' --adminurl 'http
   ://ENDERECO_IP_CONTROLADOR:8776/v1/$(tenant_id)s' --internalurl
    'http://ENDERECO_IP_CONTROLADOR:8776/v1/$(tenant_id)s'
```

Para o *glance*:

```
# keystone endpoint-create --region REGIAO --service_id
   ID_SERVICO_GLANCE --publicurl 'http://ENDERECO_IP_CONTROLADOR
   :9292/v1' --adminurl 'http://ENDERECO_IP_CONTROLADOR:9292/v1'
   --internalurl 'http://ENDERECO_IP_CONTROLADOR:9292/v1'
```

Para o próprio *keystone*:

```
# keystone endpoint-create --region REGIAO --service_id
   ID_SERVICO_KEYSTONE --publicurl http://ENDERECO_IP_CONTROLADOR
   :5000/v2.0 --adminurl http://ENDERECO_IP_CONTROLADOR:35357/v2.0
    --internalurl http://ENDERECO_IP_CONTROLADOR:5000/v2.0
```

Para o *ec2*:

```
# keystone endpoint-create --region REGIAO --service_id
   ID_SERVICO_EC2 --publicurl http://ENDERECO_IP_CONTROLADOR:8773/
   services/Cloud --adminurl http://ENDERECO_IP_CONTROLADOR:8773/
   services/Admin --internalurl ENDERECO_IP_CONTROLADOR:8773/
   services/Cloud
```

Com estes comandos executados, todos os serviços do *OpenStack* podem ser identificados e autenticados na nuvem. No entanto, é necessário ainda atualizar os dados criados no *Keystone* para os arquivos de configuração dos serviços. As linhas presentes na lis-

tagem a seguir devem ser adicionadas ao final dos arquivos `'/etc/glance/glance-api-paste.ini'` e `'/etc/glance/glance-registry-paste.ini'` referentes ao *glance*:

```
1 admin_tenant_name = service
2 admin_user = glance
3 admin_password = SENHA_USUARIO_GLANCE_OPENSTACK
```

Alteração similar deve ser feita também no arquivo `'/etc/nova/api-paste.ini'`, inserindo ao seu final linhas como as descritas a seguir:

```
1 admin_tenant_name = service
2 admin_user = nova
3 admin_password = SENHA_USUARIO_NOVA_OPENSTACK
```

Estes serviços devem ser reinicializados através de comandos como os já apresentados anteriormente na última subseção. Concluídas estas configurações, o ambiente *OpenStack* pode ser criado. Deve-se para tal exportar algumas informações sobre as credenciais de acesso dos serviços, facilitando o uso da interface. Assim, os seguintes comandos podem ser executados (as variáveis exportadas a seguir podem também ser condicionadas em um arquivo e adicionadas ao *.bashrc*):

```
1 # export SERVICE_ENDPOINT="http://localhost:35357/v2.0"
2 # export SERVICE_TOKEN=admin
3 # export OS_TENANT_NAME=admin
4 # export OS_USERNAME=admin
5 # export OS_PASSWORD=SENHA_USUARIO_ADMIN_OPENSTACK
6 # export OS_AUTH_URL="http://localhost:5000/v2.0/"
```

Pode-se agora criar um projeto que relacionará todos os dados da implementação na arquitetura do *OpenStack*, além da rede que distribuirá as máquinas virtuais, utilizando-se para tal os comandos:

```
# nova-manage project create NOME_DO_PROJETO admin
# nova-manage network create NOME_DA_REDE
   RANGE_ENDERECOS_IP_PRIVADOS NUMERO_DE_REDES
   QUANTIDADE_ENDERECOS_REDE
```

Os comando a seguir podem ser executados para garantir que as máquinas virtuais possam utilizar serviços como *ping* e *ssh*:

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

Deve-se ainda editar o arquivo `'/etc/openstack_interface.conf'`, atualizando-o com os dados relativos ao ambiente da implementação, similarmente ao descrito abaixo (os nomes dos nós-compute devem ser resolvidos por qualquer máquina da rede. Isso pode ser feito por um servidor *DNS* ou, como no caso do presente trabalho, através de entradas no arquivo `'/etc/hosts'`. Os *MACs* destes mesmos nós devem ser aqueles relativos às interfaces de rede conectadas na *bridge* `'xenbr0'`):

```
[mysql\_server]\\
mysql\_server\_address = ENDERECO_IP_CONTROLADOR\\
mysql\_user = USUARIO_MYSQL\\
mysql\_user\_passwd = SENHA_USUARIO_MYSQL\\
openstack\_dbname = NOME_BANCO_DE_DADOS_MYSQL\\
controller\_MAC\_address = ENDERECO_MAC_CONTROLADOR\\
controller\_IP\_address = ENDERECO_IP_CONTROLADOR\\
network = ENDERECO_DA_REDE\\
[compute\_nodes\_macs]\\
NOME_NO_COMPUTE_UM = ENDERECO_MAC_NO_COMPUTE_UM\\
NOME_NO_COMPUTE_DOIS = ENDERECO_MAC_NO_COMPUTE_DOIS\\
NOME_NO_COMPUTE_TRES = ENDERECO_MAC_NO_COMPUTE_TRES\\
...\\
```

O funcionamento correto da ferramenta pode ser observado através do comando abaixo. Ele retorna detalhes dos nós-compute acessíveis ao controlador e dos serviços ativos com status de funcionamento. Abaixo deste será apresentado o resultado de sua execução no ambiente desenvolvido por este trabalho a título de exemplo:

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

AVAILABILITYZONE nova available
AVAILABILITYZONE |- vmone
AVAILABILITYZONE ||- nova-network enabled :-) 2013-08-08 18:02:08
AVAILABILITYZONE ||- nova-scheduler enabled :-) 2013-08-08
   18:02:07
AVAILABILITYZONE ||- nova-compute enabled :-) 2013-08-08 18:02:05
AVAILABILITYZONE |- vmtwo
AVAILABILITYZONE ||- nova-compute enabled XXX 2013-08-06 17:44:19
AVAILABILITYZONE |- vmthree
AVAILABILITYZONE ||- nova-compute enabled XXX 2013-08-06 17:59:36
```

É preciso publicar uma imagem para ser usada na criação de máquinas virtuais a partir da interface do *OpenStack*. Imagens do sistema operacional *Ubuntu* podem ser obtidas pelo comando `'# wget'` utilizando o endereço do repositório oficial de *releases*[4]. O *OpenStack* disponibiliza uma imagem do *Ubuntu* para *download*. Os comandos a seguir baixam a imagem e a publicam para que possa ser acessada pelos nós-compute:

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

Em seguida, para inicializar instâncias dessa máqina virtual, faz-se necessária a criação de uma chave *RSA*, garantindo uma autenticação segura. O comando a seguir cria uma chave privada e cadastra a chave pública correspondente no sistema:

```
# euca-add-keypair NOME_DA_CHAVE
```

[4]http://releases.ubuntu.com, acessado em agosto de 2013

O comando abaixo utilizará a chave anteriormente criada para finalmente inicializar uma instância da máquina virtual (pode-se repetí-lo para criação de outras instâncias, diferenciando-as pelo nome):

```
# euca-run-instances ‘‘NOME_DA_INSTANCIA’’ -k NOME_DA_CHAVE -t m1
   .tiny
```

3.5 O *Network Hypervisor POX*

O *POX*[5] vem sendo desenvolvido para substituir o *NOX* (Gude et al., 2008), essencialmente onde a interface *Python* é utilizada (Guedes et al., 2012). Funciona com um conceito de orientação a eventos disparados por *switches OpenFlow*, onde os módulos desenvolvidos sobre ele podem reagir e executar ações interventivas na rede. Oferece também uma biblioteca para criação de mensagens no padrão *OpenFlow*, possibilitando a programação dos *switches* da *SDN*.

Para utilizar o *network hypevisor POX* é preciso adquirir seu código fonte[6], que hoje está incluído no controlador *GitHub*. É neste código que devem ser implementados os módulos para controlar e intervir nos fluxos de pacote da rede. Ativa-se um módulo *POX* através de um comando simples, como no exemplo abaixo:

```
# ./pox.py NOME_DO_MODULO
```

No presente trabalho, optou-se por desenvolver um módulo teste com proposta simples, para que pudessem ser avaliadas as possibilidades da arquitetura implementada. Informações armazenadas na base de dados do gerenciador *OpenStack* poderiam ser consultadas via *SQL* para que manipulações mais profundas pudessem ser efetuadas nos fluxos de pacote, como aquelas necessárias em grandes *datacenters* e implementadas em (Nunes et al., 2013). Entretanto, a estrutura simples aqui desenvolvida não deixaria margens para testes mais complexos, decidindo-se portanto por funções mais didáticas que podem ambientar melhor o leitor com a proposta do controlador.

O código desenvolvido e exposto a seguir embasa-se na implementação de um *switch L2* com aprendizado proposta em (Guedes et al., 2012). Nele foram adicionadas funcionalidades que pudessem ser observadas concretamente na rede. Exemplo de tal foi o bloqueio de tráfgo *ICMP*, que anteriormente foi liberado pela configuração no *OpenStack* e passou a ser efetuado nos *Openvswitches* que conectam as máquinas virtuais, através de programação em eventos *POX* e troca de mensagens *OpenFlow*. Assim (importações de bibliotecas foram omitidas para simplicação do código):

```
1 class TCC (EventMixin):
2     def __init__(self):
3         self.listenTo(core.openflow)
4
5     def _handle_ConnectionUp(self, event):
6         LearningSwitch(event, connection)
7
8     def launch():
```

[5]https://openflow.stanford.edu/display/ONL/POX+Wiki, acessado em agosto de 2013
[6]https://github.com/noxrepo/pox, acessado em agosto de 2013

```
9           core.registerNew(TCC)
```

O método *launch()* é quem torna o módulo executável pelo controlador *POX*. Ele foi definido, na realidade, em um arquivo a parte na pasta, denominado '__init__.py', para possibilitar a inicialização do módulo através de seu nome, como exemplificado anteriormente. A classe 'TCC' funciona como um fábrica de *switches*, associando um objeto da classe 'LearningSwitch' a cada conexão de um novo comutador detectada pelo evento tratado. Esta última classe é detalhada na listagem adiante:

```
1  class LearningSwitch (EventMixin):
2       def __init__(self, connection):
3            self.connection = connection
4            self.macToPort = {}
5            self.listenTo(connection)
6
7       def _handle_PacketIn (self, event):
8            packet = event.parse()
9
10           if packet.src not in self.macToPort:
11                macToPort[packet.src] = event.port
12
13           if packet.dst.isMulticast():
14                flood(event)
15                return
16
17           if packet.dst not in self.macToPort:
18                flood(event)
19                return
20
21           if packet.type == packet.IP_TYPE:
22                ip_packet = packet.payload
23                if ip_packet.protocol == ip_packet.ICMP_PROTOCOL
24                     drop()
25                     return
26
27           msg = of.ofp_flow_mod()
28           msg.match = of.ofp_match.from_packet(packet)
29           msg.idle_timeout = 10
30           msg.hard_timeout = 30
31           msg.actions.append(of.ofp_action_output(port = port))
32           msg.buffet_id = event.ofp.buffer_id
33           self.connection.send(msg)
34
35      def flood (event):
36           msg = ofp_packet_out()
37           msg = actions.append(ofp_action_output(port =
                OFPP_FLOOD))
38           msg.buffer_id = event.ofp_buffer_id
39           msg.in_port = event.port
40           event.connection.send(msg)
41
42      def drop (duration = None):
43       if duration is not None:
```

```
44          if not isinstance(duration, tuple):
45            duration = (duration,duration)
46          msg = of.ofp_flow_mod()
47          msg.match = of.ofp_match.from_packet(packet)
48          msg.idle_timeout = duration[0]
49          msg.hard_timeout = duration[1]
50          msg.buffer_id = event.ofp.buffer_id
51          self.connection.send(msg)
52        elif event.ofp.buffer_id is not None:
53          msg = of.ofp_packet_out()
54          msg.buffer_id = event.ofp.buffer_id
55          msg.in_port = event.port
56          self.connection.send(msg)
```

Na classe anteriormente apresentada, define-se de fato o comportamento dos *switches OpenFlow* que conectam as máquinas virtuais da *SDN* implementada. As linhas iniciais compõem o construtor da classe, que identifica a conexão de controle, cria um dicionário para registrar endereços conhecidos e registra o próprio objeto para responder a eventos gerados no comutador. O método '`_handle_PacketIn`' possui as ações que serão executadas sempre que um novo pacote chegar ao comutador, disparado pelo evento '`PacketIn`'. Estas ações incluem:

- Atualizar o dicionário de endereços caso o endereço de origem não esteja presente no mesmo.
- Enviar o pacote a todas as interfaces caso o endereço de destino seja *Multicast*.
- Enviar o pacote a todas as interfaces caso o endereço de destino não esteja no dicionário de endereços.
- Descartar o pacote caso o mesmo contenha mensagem com protocolo *ICMP*.
- Enviar o pacote à interface adequada.

O método '`flood`' garante o envio a todas as interfaces do comutador, enquanto o método '`drop`' efetua o descarte do pacote tratado.

Inicializando-se o controlador *POX* com o módulo descrito, observou-se que a comunicação entre as máquinas virtuais era completa com exceção de mensagens com protocolo *ICMP*, mesmo quando este tráfego havia sido explicitamente autorizado durante o processo de criação da rede virtual com o *OpenStack*. Isto comprova a interferência do módulo implementado na rede através de comando direto no comutador de pacotes *OpenvSwitch* que conecta as máquinas virtuais instanciadas na rede.

4 Considerações Finais

O paradigma de Redes Definidas por Software trouxe grandes possibilidades para o desenvolvimento de aplicações no cenário de redes de computadores. Seu potencial, porém, apenas começou a ser explorado. Vários são os desafios de pesquisa na área e, certamente, novas soluções e projetos baseados nesta arquitetura estão a caminho. Provavelmente, os mais revolucionários.

Um dos grandes benefícios da arquitetura de *SDN* é a possibilidade real de obter-se uma visão global da rede, representada pela ideia do controlador. Observou-se que os controladores atualmente disponíveis ainda não possuem esse conceito como algo intrínseco de suas funcionalidades, apesar de possibilitarem sua definição através de informações processadas em trocas de mensagens ou, como mencionado no caso da implementação aqui desenvolvida, pelas informações armazenadas na ferramenta gerenciadora da nuvem de recursos (*OpenStack*).

Outro ponto importante advindo deste paradigma é a forma de se enxergar as redes. Incluem-se por ele noções de princípios organizacionais abstratos, muito comuns em Sistemas Operacionais, tornando ultrapassada a visão atual de simples conjunção de artefatos tecnológicos. A possibilidade de organização e manipulação destes recursos através de uma camada controladora abstrata deverá revolucionar a forma de se implementar uma rede em um futuro próximo.

A forma ideal de organização e distribuição dos ativos de *software* e de processamento e armazenamento de informações em uma *SDN* pode ser colocada como um dos desafios da área. Observou-se que a centralização destas tarefas em um nó controlador pode não ser a melhor escolha em termos de eficiência e escalabilidade, além de representar uma possível preocupação para segurança e confiabilidade da rede.

Com a análise deste trabalho, espera-se que o leitor seja capaz de compreender a arquitetura de uma *SDN*, além de ter uma relevante referência para futuras implementações e configurações de ambiente. Algumas ferramentas utilizadas apresentam detalhes de configuração não triviais e acredita-se que este trabalho possa ser uma boa fonte para o entendimento necessário.

Espera-se também que o conceito de visão global da rede provido pelo paradigma abordado, possa abrir caminho para muitas outras políticas benéficas ao ambiente alvo.

Referências Bibliográficas

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

Braga, R.; Mota, E. ; Passito, A. **Lightweight ddos flooding attack detection using nox/openflow**. In: Local Computer Networks (LCN), 2010 IEEE 35th Conference on, p. 408–415. IEEE, 2010.

Casado, M.; Koponen, T.; Ramanathan, R. ; Shenker, S. **Virtualizing the network forwarding plane**. In: Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow, p. 8. ACM, 2010.

Davie, B. S.; Farrel, A. **MPLS: Next Steps: Next Steps**, volume 1. Morgan Kaufmann, 2008.

Elliott, C.; Falk, A. An update on the geni project. **SIGCOMM Comput. Commun. Rev.**, v.39, n.3, p. 28–34, Jun 2009.

Gude, N.; Koponen, T.; Pettit, J.; Pfaff, B.; Casado, M.; McKeown, N. ; Shenker, S. Nox: towards an operating system for networks. **ACM SIGCOMM Computer Communication Review**, v.38, n.3, p. 105–110, 2008.

Guedes, D.; Vieira, L.; Vieira, M.; Rodrigues, H. ; Nunes, R. V. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. **Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012**, v.30, n.4, p. 160–210, 2012.

Kempf, J.; Whyte, S.; Ellithorpe, J.; Kazemian, P.; Haitjema, M.; Beheshti, N.; Stuart, S. ; Green, H. **Openflow mpls and the open source label switched router**. In: Proceedings of the 23rd International Teletraffic Congress, p. 8–14. ITCP, 2011.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. ; Turner, J. Openflow: enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, v.38, n.2, p. 69–74, Mar. 2008.

Nunes, R. V.; Pontes, R. L. ; Guedes, D. Uma arquitetura baseada em redes definidas por software para isolamento de redes em datacenters virtualizados. **31 Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, p. 703–716, 2013.

Peterson, L.; Roscoe, T. The design principles of planetlab. **SIGOPS Oper. Syst. Rev.**, v.40, n.1, p. 11–16, Jan. 2006.

Pfaff, B.; Pettit, J.; Amidon, K.; Casado, M.; Koponen, T. ; Shenker, S. **Extending networking into the virtualization layer.** In: Hotnets, 2009.

Tennenhouse, D.; Wetherall, D. **Towards an active network architecture**. In: DARPA Active NEtworks Conference and Exposition, 2002. Proceedings, p. 2–15, 2002.

Turner, J. **A proposed architecture for the geni backbone platform**. In: Architecture for Networking and Communications systems, 2006. ANCS 2006. ACM/IEEE Symposium on, p. 1–10, 2006.

Waddington, D.; Chang, F. Realizing the transition to ipv6. **Communications Magazine, IEEE**, v.40, n.6, p. 138–147, 2002.