MARCOS FELIPE SCHWARZ

Mecanismo para Integração de Comutadores OpenFlow
na Infraestrutura do Testbed Emulab

São Paulo

2014

MARCOS FELIPE SCHWARZ

Mecanismo para Integração de Comutadores OpenFlow
na Infraestrutura do Testbed Emulab

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do título de
Mestre em Ciências

São Paulo

2014

MARCOS FELIPE SCHWARZ

Mecanismo para Integração de Comutadores OpenFlow na Infraestrutura do Testbed Emulab

Dissertação apresentada à Escola
Politécnica da Universidade de São
Paulo para obtenção do título de
Mestre em Ciências

Área de Concentração: Engenharia
da Computação

Orientadora: Prof^a Dr^a Tereza Cristina
Melo de Brito Carvalho

São Paulo

2014

AGRADECIMENTOS

À professora Tereza Cristina Melo de Brito Carvalho, pela orientação e pelo constante apoio.

Ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e Projeto FIBRE (590022/2011-3). Ao Centro de Inovação da Ericsson Telecomunicações S.A., Brasil. À FDTE (Fundação para o Desenvolvimento Tecnológico da Engenharia) e ao LARC por terem possibilitado esta pesquisa.

À minha esposa Mariana Ventura Zini, que me deu forças nos momentos difíceis e sempre esteve ao meu lado.

E a todos que colaboraram direta ou indiretamente, na execução desta pesquisa.

RESUMO

O Emulab é um ambiente de experimentação que permite a emulação de redes com topologia arbitrária definida pelo usuário, em um ambiente controlável, previsível e reproduzível. Tal topologia definida pelo usuário é provisionada no ambiente físico de forma automatizada, por meio de nós físicos instanciados com o sistema operacional e os endereços IP solicitados e um conjunto de comutadores configurados dinamicamente para emular a topologia de rede desejada. A abordagem utilizada para a configuração dos comutadores, baseada em SNMP, impõe requisitos que limitam os modelos de comutadores que podem ser usados para a construção de um ambiente Emulab e, ao mesmo tempo, exige a criação de módulos específicos para cada novo modelo de comutador a ser suportado. Este trabalho propõe a utilização do protocolo OpenFlow no ambiente Emulab, como alternativa ao modelo atual baseado em SNMP, criando, para esse fim um módulo OpenFlow para ambiente Emulab. Esse módulo utiliza OpenFlow como interface padronizada para configurar comutadores de diferentes modelos e fabricantes que suportem o protocolo OpenFlow, removendo, dessa forma as restrições citadas anteriormente e adicionando suporte a novos modelos de comutadores. Foi efetuado um conjunto de testes para analisar o funcionamento e desempenho do uso de comutadores OpenFlow no Emulab, que comprovaram que a solução proposta adiciona suporte a diferentes modelos de comutadores OpenFlow de forma unificada e com desempenho comparável a comutadores convencionais.

ABSTRACT

Emulab is a emulated network testbed that provides user-defined network topologies in a controllable, predictable, and repeatable environment. The topology defined by the user is provisioned in the physical environment in an automated manner by means of physical nodes instantiated with the desired operating system and IP addresses and a set of switches dynamically configured to emulate the desired network topology. The approach used for switch configuration is based on SNMP and currently imposes requirements that limit the switch models that can be used to build an Emulab facility and at the same time requires the creation of specific modules for each new type of switch to be supported. This work proposes the use of the OpenFlow protocol for configuring switches in Emulab as an alternative to the current model based on SNMP. As result, it was created an OpenFlow module for Emulab environment, which uses OpenFlow as a standardized interface for configuring these devices. This new OpenFlow-based approach makes it possible to configure switches of different models and manufacturers that support the OpenFlow protocol through a single mechanism, removing thus the restrictions mentioned above, and adding support for new types of switches. A set of tests were performed to analyze the operation and performance of OpenFlow switches in Emulab, which showed that the proposed solution adds support for different models of OpenFlow switches in a unified manner with comparable performance to conventional switches.

LISTA DE ABREVIATURAS E SIGLAS

API	<i>Application Programming Interface</i>
ARP	<i>Address Request Protocol</i>
CBR	<i>Constant Bit Rate</i>
CLI	<i>Command Line Interface</i>
DNS	<i>Domain Name System</i>
DPID	<i>Data Path IDentifier</i>
FIB	<i>Forwarding Information Base</i>
FIBRE	<i>Future Internet testbeds experimentation between BRazil and Europe</i>
Frederica	<i>Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architecture</i>
GENI	<i>Global Environment for Network Innovations</i>
ICMP	<i>Internet Control Message Protocol</i>
IP	<i>Internet Protocol</i>
JSON	<i>JavaScript Object Notation</i>
MAC	<i>Media Access Control</i>
MIB	<i>Management Information Base</i>
NAT	<i>Network Address Translation</i>
NICT	<i>National Institute of Information and Communications Technology</i>
OFELIA	<i>OpenFlow in Europe: Linking Infrastructure and Applications</i>
OMF	<i>cOntrol and Management Framework</i>
PanLab	<i>Pan-European Laboratory</i>

PC	<i>Personal Computer</i>
PXE	<i>Preboot eXecution Environment</i>
QoS	<i>Quality of Service</i>
REST	<i>REpresentational State Transfer</i>
RPC	<i>Remote Procedure Call</i>
SDN	<i>Software-Defined Networking</i>
SNMP	<i>Simple Network Management Protocol</i>
SO	<i>Sistema Operacional</i>
SSH	<i>Secure Shell</i>
TCAM	<i>Ternary Content-Addressable Memory</i>
TCP	<i>Transmission Control Protocol</i>
VLAN	<i>Virtual Local Area Network</i>

LISTA DE FIGURAS

Figura 1 - Elementos da arquitetura do Emulab	20
Figura 2 - Processos da alocação de recursos de um experimento	26
Figura 3 - Arquitetura SDN.....	31
Figura 4 - Comutador OpenFlow, seus elementos e a comunicação com o controlador.....	34
Figura 5 - Processo de alocação de recursos com o módulo de suporte a OpenFlow	41
Figura 6 - Exemplo do processo de mapeamento de topologia da rede de experimentos em recursos físicos no Emulab	46
Figura 7 - Topologia para execução dos testes.....	59

LISTA DE TABELAS

Tabela 1 – Modelos de comutadores suportados na rede de experimentação do Emulab.....	23
Tabela 2 – Campos definidos no protocolo OpenFlow 1.0.....	33
Tabela 3 - Informações sobre controladores OpenFlow.....	36
Tabela 4 – Fluxos de difusão com domínio restrito por VLAN	47
Tabela 5 – Fluxos de encaminhamento por MAC/interface com tratamento de rótulos IEEE 802.1Q	48
Tabela 6 – Registros retornados para a consulta do experimento 1	50
Tabela 7 – Lista de recursos alocados no comutador OpenFlow.....	52
Tabela 8 – Mensagens trocadas entre os nós 1 e 3 durante a requisição ICMP	61
Tabela 9 – Regras de fluxos instaladas no comutador OpenFlow	62
Tabela 10 – Comparação da taxa de vazão de comutador convencional e OpenFlow	66
Tabela 11 – Tempo de resposta do primeiro pacote para módulo convencional e OpenFlow no modo proativo	68
Tabela 12 – Tempo de resposta do primeiro pacote para módulo Openflow no modo reativo	68
Tabela 13 – Comparação do tempo de resposta a partir do segundo pacote	69
Tabela 14 – Fluxos de encaminhamento com Meters.....	72
Tabela 15 – Registros na tabela de Meters.....	72
Tabela 16 – Fluxos de encaminhamento com solução da vulnerabilidade de isolamento	74
Tabela 17 – Fluxos de encaminhamento com múltiplas tabelas de fluxo.....	75

LISTA DE QUADROS

Quadro 1 – Código SQL para consulta de recursos alocados no experimento	51
Quadro 2 – Diferenças entre requisições para instalação de regras de fluxos entre Ryu e POX.....	55
Quadro 3 – Requisições HTTP para instalação de regras de fluxo para controlador Ryu.....	87

SUMÁRIO

1	Introdução	13
1.1	Motivação e Descrição do Problema	15
1.2	Objetivos.....	16
1.3	Justificativa	17
1.4	Metodologia	17
1.5	Organização do Texto	17
2	Fundamentos	19
2.1	Emulab	19
2.1.1	Arquitetura	20
2.1.2	Alocação do Experimento.....	24
2.2	SDN e OpenFlow.....	30
2.2.1	Visão Geral Sobre SDN.....	30
2.2.2	Visão Geral Sobre OpenFlow	32
2.3	Trabalhos Relacionados	37
2.4	Considerações Finais do Capítulo	39
3	Mecanismo de Suporte a OpenFlow	40
3.1	Especificação de Requisitos.....	40
3.2	Operação do Emulab com OpenFlow.....	41
3.3	Definição das Operações do Protocolo OpenFlow	42
3.4	Definição das Regras de Fluxos	45
3.5	Implementação do Protótipo.....	49
3.5.1	Diferenças Entre os Controladores.....	54
3.6	Considerações Finais do Capítulo	55
4	Avaliação do Protótipo.....	57
4.1	Recursos Utilizados	58
4.2	Execução dos Testes.....	59
4.2.1	Testes de Funcionalidades	60
4.2.2	Testes de Desempenho.....	65
4.3	Validação dos Requisitos.....	70
4.4	Limitações e Melhorias	71
4.5	Considerações Finais do Capítulo	76
5	Considerações Finais.....	77
5.1	Avaliação do Mecanismo Proposto.....	77
5.2	Contribuições.....	80
5.3	Publicações	81
5.4	Trabalhos Futuros.....	82
	Referências Bibliográficas	84
	Apêndice A.....	87

1 INTRODUÇÃO

Nas últimas décadas, as redes têm se tornado parte fundamental da infraestrutura de empresas, residências e escolas. Com isto, o trabalho dos pesquisadores da área de redes tem se tornado mais relevante, porém, a enorme base instalada de equipamentos e protocolos e a dificuldade de realizar experimentos com tráfego de produção, tem criado uma barreira para testar e validar novas ideias e soluções inovadoras em ambientes reais. Para contornar este problema, ambientes para experimentação (*testbeds*) tem sido implementados pela comunidade de redes internacional e nacional, possibilitando o desenvolvimento de novos protocolos, serviços e algoritmos em redes reais, sem interferir nos ambientes de produção (ABELÉM et al., 2010). Como exemplos destes ambientes, podemos citar: GENI (*Global Environment for Network Innovations*) (ELLIOTT, 2008), Emulab/ProtoGENI, Federica¹ (Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architecture) (SZEGEDI, 2009), AKARI² (HIRABARU, 2008) e FIBRE³ (*Future Internet testbeds experimentation between BRazil and Europe*) (SALLENT et al., 2012)

Dentre tais ambientes de experimentação de redes, destaca-se o Emulab desenvolvido pela Universidade de Utah, que possibilita que pesquisadores configurem e acessem redes compostas de nós e enlaces, emulados e simulados, de longa distância (HIBLER, et al, 2008). O Emulab funciona como um painel de junção (*patch panel*) programável, que permite a criação de uma topologia arbitrária definida pelo experimentador. Para possibilitar a configuração automatizada da rede, o Emulab utiliza mecanismos que são implementados de forma diferente em comutadores de fabricantes distintos, como por exemplo, a criação remota de VLANs (*Virtual*

¹ O Projeto FREDERICA é um projeto europeu para implementar uma infraestrutura de rede experimental para testar novas tecnologias de rede.

² O Projeto AKARI tem como meta projetar a arquitetura da nova geração de redes de computadores do Japão e é financiado pelo NICT(National Institute of Information and Communications Technology). O nome “AKARI” significa “pequena luz” (na escuridão apontando para o futuro). Esse projeto foi iniciado em 2006.

³ O FIBRE visa a concepção, implementação de um ambiente compartilhado para pesquisa em Internet do Futuro, possibilitando a experimentação conjunta de pesquisadores da comunidade europeia e brasileira.

Local Area Network) e *VLAN trunking* (EMUALB WIKI, 2013). Esta abordagem dificulta o suporte a novos modelos de comutadores na infraestrutura do Emulab, devido à necessidade de criar novos módulos específicos para configurar cada modelo.

Como alternativa para realizar a configuração de comutadores de forma padronizada, esta pesquisa propõe o uso do protocolo OpenFlow. O OpenFlow que é um componente da arquitetura SDN (*Software-Defined Networking*) em que o plano de dados e o plano de controle do equipamento de rede são separados; com isso, o plano de controle é executado em servidores externos e o equipamento de rede fica responsável apenas pela transmissão de tráfego no plano de dados (ONF, 2012). O protocolo OpenFlow define uma abordagem aberta e padronizada para o comutador se comunicar com um elemento externo, o controlador (MCKEOWN et al. 2008). O OpenFlow especifica abstrações que possibilitam uma interface padrão por meio da qual as entradas na tabela de encaminhamento do comutador podem ser configuradas pelo controlador.

Considerando as restrições observadas no ambiente Emulab, esta pesquisa propõe a utilização do protocolo OpenFlow para oferecer funcionalidades equivalentes às do ambiente atual do Emulab, que se baseia em SNMP (*Simple Network Management Protocol*) e MIBs (*Management Information Base*) proprietárias. No contexto desse trabalho, é proposta a criação de um mecanismo para suportar um conjunto de comutadores de modelos e fabricantes variados compatíveis com o protocolo OpenFlow. Essa abordagem baseia-se na criação de um módulo de suporte a OpenFlow compatível com os módulos já existentes, mantendo-se o suporte aos comutadores já existentes que não são compatíveis com o protocolo OpenFlow.

O mecanismo proposto baseia-se na utilização de um controlador OpenFlow integrado ao Emulab para efetuar a tarefa de configurar os comutadores suportados, permitindo que, por meio de um mesmo mecanismo, sejam suportados comutadores OpenFlow de diferentes fabricantes.

1.1 Motivação e Descrição do Problema

O ambiente Emulab está em operação em mais de 36 instituições de pesquisa, sendo 24 nos Estados Unidos e as 12 restantes em outros países (EMUALB WIKI, 2013). Um desses ambientes é mantido pelo LARC (Laboratório de Arquitetura e Redes de Computadores), vinculado ao Departamento de Engenharia da Computação e Sistemas Digitais da Escola Politécnica da USP. Por meio de iniciativas, como o projeto FIBRE (SALLENT et al., 2012), está sendo realizada a federação de ambientes Emulab com outros ambientes de experimentação, como por exemplo, o ambiente OMF (*cOntrol and Management Framework*) para experimentação em redes sem fio (RAKOTOARIVELO et al., 2010) e o ambiente OFELIA (*OpenFlow in Europe: Linking Infrastructure and Applications*), para experimentação em redes OpenFlow de longa distância (SUÑÉ, 2013). Iniciativas como estas promovem a utilização de ambientes Emulab existentes, a criação de novas infraestruturas de suporte ao Emulab e a expansão de ambientes já existentes.

No processo de criação e expansão de ambientes Emulab, uma das atividades de maior impacto é a definição de comutadores para compor a rede de experimentação. Estes dispositivos devem respeitar uma série de requisitos que limitam as opções, ou impõem o desenvolvimento de novos módulos de compatibilidade para o Emulab.

Isto deve-se ao fato do mecanismo atual do Emulab, que possibilita o provisionamento dos comutadores de experimentação, efetuar operações de configuração do dispositivo que são implementadas de forma diversa pelos diferentes fabricantes. Como exemplo, pode-se citar a criação de VLANs (*Virtual Local Area Networks*) e VLAN *trunking*, que é efetuada de modo diferente em equipamentos de fabricantes distintos. Estas funcionalidades estão disponíveis em alguns casos apenas em MIBs (*Management Information Base*) proprietárias do protocolo SNMP, ou em outros casos não estão disponíveis por meio de SNMP e precisam ser configurados por meio de acesso remoto ao console do dispositivo.

Estas características de implementação causam um impacto negativo no trabalho de adicionar suporte a novos modelos de comutadores ao Emulab. A longo prazo, a abordagem atual de configuração dos comutadores gerou um cenário no qual mais da metade dos modelos de comutadores suportados não são mais vendidos nem suportados por seus fabricantes; essa situação é descrita com maiores detalhes no decorrer deste trabalho.

Observadas estas restrições, esta pesquisa propõe a elaboração de um mecanismo baseado no protocolo OpenFlow, para suportar um conjunto de comutadores de modelos e fabricantes variados compatíveis com o protocolo OpenFlow, a fim de serem utilizados na rede de experimentação do Emulab.

1.2 Objetivos

Este trabalho tem como objetivo propor e avaliar um mecanismo que possibilite a utilização de comutadores OpenFlow no ambiente de experimentação Emulab. Este mecanismo será responsável por realizar a configuração do comutador de forma automatizada, provendo conectividade e isolamento entre experimentos por meio de VLANs e mantendo a compatibilidade com os mecanismos previamente existentes.

Para atingir tal objetivo, foram definidos três objetivos secundários:

1. Propor um novo módulo para o Emulab, que efetue a alocação e configuração padronizada de comutadores OpenFlow para diferentes modelos e fabricantes;
2. Implementar as funcionalidades no controlador OpenFlow necessárias para estabelecer a conectividade e o isolamento dos nós, de forma compatível com os comutadores convencionais já existentes no Emulab;
3. Efetuar medições que permitam avaliar as funcionalidades e o desempenho do módulo de suporte OpenFlow por meio de um protótipo e compará-lo com os mecanismos atuais.

1.3 Justificativa

Os trabalhos realizados nesta dissertação agregam valor ao ambiente Emulab e, por consequência, à comunidade de experimentação em redes. A criação deste novo módulo para configuração de rede baseado em OpenFlow viabiliza não somente a adição de novos modelos de comutadores à infraestrutura do Emulab, mas também a criação de uma nova plataforma para integração de serviços de redes utilizando OpenFlow. Estes novos serviços podem ser adicionados ao módulo OpenFlow em forma de componentes, que podem substituir o atual mecanismo de encaminhamento para adicionar funcionalidades que estendem os mecanismos atuais do Emulab.

1.4 Metodologia

O método de pesquisa empregado nesse trabalho é o método hipotético-dedutivo, em que a partir da definição do problema utilizam-se referências científicas da área para especificação da hipótese de solução e a avaliação dessa hipótese se dá por meio do processo de experimentação. A solução especificada é implementada por um protótipo com o objetivo de gerar os resultados que possibilitam a validação da hipótese definida. Foram efetuados experimentos para verificar o funcionamento e o desempenho do protótipo implementado. Para implementar tal protótipo, foi necessária a utilização do ambiente Emulab da USP e comutadores OpenFlow do ambiente Emulab/ProtoGENI do Projeto FIBRE/LARC.

1.5 Organização do Texto

Esta dissertação está organizada em 5 capítulos. O Capítulo 2 Fundamentos está subdividido em duas seções principais. A sua primeira seção contém a apresentação do ambiente Emulab, iniciando-se com uma visão geral sobre o seu funcionamento e os elementos da sua arquitetura,

para então se aprofundar nos recursos mais específicos como os comutadores da rede de experimentação e o módulo de configuração do Emulab, evidenciando-se as limitações e as restrições encontradas. A segunda seção do Capítulo 2, refere-se ao protocolo OpenFlow, apresentando-se os principais conceitos para o entendimento desta tecnologia e identificando-se os elementos necessários para a criação de uma rede OpenFlow. O Capítulo 3 Mecanismo de Suporte a OpenFlow descreve o módulo de suporte a OpenFlow no Emulab, que é a solução proposta nesta dissertação. É apresentada sua arquitetura e são detalhados o funcionamento do módulo e a implementação do protótipo. O Capítulo 4 Análise do Protótipo apresenta os testes realizados para verificar e validar o mecanismo proposto com relação ao seu funcionamento e seu desempenho, comparando-o com o modelo original do Emulab. O Capítulo 5 Considerações Finais, avalia o mecanismo proposto, descreve as principais contribuições do trabalho, apresenta considerações sobre limitações da solução proposta e sugere melhorias dentro do contexto de trabalhos futuros.

2 FUNDAMENTOS

Este capítulo apresenta os principais fundamentos que foram utilizados para detectar o problema levantado por esta dissertação, bem como para possibilitar a criação de uma proposta de solução. O capítulo está dividido em duas seções principais: a primeira apresenta o ambiente Emulab, seu funcionamento e os elementos da sua arquitetura, para então se aprofundar nos recursos mais específicos como os comutadores da rede de experimentação e o módulo de configuração do Emulab, evidenciando-se as limitações e as restrições encontradas. A segunda seção apresenta os principais conceitos sobre SDN (*Software-Defined Networking*) e o protocolo OpenFlow, evidenciando os elementos básicos para a criação de uma rede OpenFlow.

2.1 Emulab

Nesta seção, será detalhado o ambiente de experimentação Emulab, iniciando-se com uma visão geral de seu funcionamento e os elementos da sua arquitetura seguida da apresentação detalhada dos recursos específicos necessários, como por exemplo, os comutadores da rede de experimentação e o módulo de configuração do Emulab, evidenciando-se as limitações e as restrições encontradas.

O Emulab fornece acesso integrado, baseado na Web, para uma ampla variedade de ambientes, incluindo a simulação e a emulação voltadas para recursos de rede de longa distância. No entanto, para os fins desta dissertação, o escopo foi restringido ao ambiente de emulação de rede local, que consiste em um conjunto de nós de computadores pessoais com a interconexão de rede configurável.

2.1.1 Arquitetura

O Emulab é composto por cinco elementos principais: servidor central, servidor de usuários, nós locais, comutadores da rede de experimentação e comutadores da rede de controle. Estes elementos e suas interconexões estão representados na Figura 1.

Figura 1 - Elementos da arquitetura do Emulab

Uma descrição mais detalhada de cada um dos elementos da arquitetura do Emulab:

1. **Servidor Central (Boss):** Hospeda os componentes principais do ambiente, como banco de dados, servidor *web*, servidor DNS (*Domain Name System*) e o controle para ligar e desligar os nós.
2. **Servidor de Usuários:** Corresponde ao servidor de arquivos. Este servidor é acessível aos usuários, fornece diretórios compartilhados para armazenamento de arquivos do projeto, tais como, os resultados e monitorações dos experimentos.
3. **Nós locais:** Cada unidade do Emulab fornece um conjunto de computadores que estarão disponíveis aos usuários. Estes nós podem ter o comportamento de computadores pessoais (PCs), roteadores ou simuladores de atraso. No modo PC, o usuário pode escolher o sistema operacional (SO) que será instalado no nó (por exemplo, FreeBSD, RedHat, Fedora, Ubuntu e MS-Windows). Os pesquisadores podem, também, personalizar a imagem do SO fornecido ou carregar sua própria imagem para instalar outros sistemas operacionais. Para os nós serem usados como roteadores ou simuladores de atraso é

necessário utilizar imagens de SOs específicas que estão pré-configuradas com estas funcionalidades.

4. **Comutadores da Rede de Controle:** Esta rede interliga o servidor central, o servidor de usuários e os nós locais entre si, além de prover o acesso à Internet. Cada nó local possui uma interface ligada à rede de controle, que é fisicamente isolada da utilizada para o tráfego experimental. A rede de controle é usada para o acesso aos nós, permitindo aos usuários efetuarem a configuração dos nós locais, como por exemplo a instalação do SO.
5. **Comutadores da Rede de Experimentação:** Esta rede é exclusiva para o tráfego dos dados experimentais. Todos os nós estão ligados por meio de múltiplas interfaces de rede a esses comutadores. A rede de experimentação permite a criação de sub-redes com topologias arbitrárias e isoladas por meio de VLANs. As VLANs são usadas para permitir que múltiplos experimentos, associados a diferentes topologias, possam coexistir simultaneamente sem interferência de um experimento no outro. Esse isolamento é responsável por assegurar serviços de segurança como confidencialidade e integridade, já que um experimento não pode interferir no tráfego do outro experimento.

2.1.1.1 Comutadores da Rede de Experimentação

Esta seção descreve com mais profundidade como são utilizados os comutadores da rede de experimentação no Emulab. Esses detalhes são relevantes, dado que, os comutadores OpenFlow deverão atender aos mesmos requisitos e oferecer funcionalidades equivalentes às que estão descritas nesta seção.

Os comutadores da rede de experimentação do Emulab permitem recriar uma topologia qualquer definida pelo usuário, empregando-se os recursos físicos disponíveis. Para isolar o tráfego entre experimentos que usam o mesmo comutador são utilizadas VLANs e para interconectar nós localizados em diferentes comutadores é necessário criar túneis de VLANs, empregando-se rótulos IEEE 802.1Q entre eles (EK, 1999). O Emulab

permite, também, configurar a taxa de vazão e o tipo de controle de fluxo das interfaces ligadas a um respectivo nó. Conforme definido pela documentação oficial do Emulab, os requisitos dos comutadores da rede de experimentação são (EMUALB WIKI, 2013):

1. **Número de interfaces:** O número total de interfaces depende diretamente do número de interfaces experimentais necessárias nos nós para se criar o ambiente de teste desejado, sendo que cada nó tem em média entre duas a cinco interfaces experimentais. Por isso, faz-se necessária uma densidade grande de interfaces por comutador para otimizar a relação de número de nós por comutador.
2. **VLAN:** É necessário o suporte a VLANs e deve ser possível a sua configuração de forma dinâmica, como por exemplo, por meio do SNMP (*Simple Network Management Protocol*).
3. **VLAN trunking** (IEEE 802.1Q): Em um ambiente com múltiplos comutadores experimentais, o mecanismo de VLAN *trunking* é a forma utilizada para compartilhar VLANs entre eles. É recomendado que as interfaces físicas usadas para VLAN *trunking* sejam de pelo menos uma ordem de magnitude mais rápida que as interfaces físicas empregadas na ligação aos nós de acesso.
4. **Fabricante:** O Emulab usa MIBs (*Management Information Base*) proprietárias para criar as VLANs dinamicamente. Estas MIBs são implementadas de forma distinta por diferentes fornecedores de equipamentos, e em alguns casos, diferem entre modelos de um mesmo fabricante. Por este motivo, apenas um conjunto restrito de modelos de comutadores são suportados.

O Emulab suporta oficialmente modelos de comutadores de seis fornecedores diferentes (EMULAB WIKI, 2013). Foi efetuada uma análise do histórico de alterações no código do Emulab para definir a data em que cada modelo foi adicionado ao Emulab. Identificaram-se registros de alterações desde novembro de 2003 (modelos suportados anteriormente a essa data foram categorizados como pré-2003). A Tabela 1 apresenta um resumo da análise efetuada a partir da compilação de uma lista contendo os modelos de

comutadores suportados pelo Emulab, a data em que cada modelo foi suportado e a situação do ciclo de vida do equipamento por parte do fabricante.

Tabela 1 – Modelos de comutadores suportados na rede de experimentação do Emulab

<i>Fabricante / Modelo</i>	<i>Suportado a partir de</i>	<i>Descontinuado</i>
<i>Intel 510T</i>	<i>Pré-2003</i>	<i>Sim</i>
<i>Cisco Catalyst 29xx</i>		<i>Sim</i> ⁴
<i>Cisco Catalyst 40xx</i>		<i>Sim</i> ¹
<i>Cisco Catalyst 65xx</i>		<i>Não</i>
<i>Cisco Catalyst 55xx</i>	<i>Outubro 2003</i>	<i>Sim</i> ¹
<i>Foundry FastIron Edge 9604</i>	<i>Novembro 2004</i>	<i>Sim</i> ⁵
<i>Foundry 1500</i>		<i>Sim</i> ²
<i>Cisco Catalyst 45xx</i>	<i>Janeiro 2005</i>	<i>Não</i>
<i>Nortel Passport 1100</i>	<i>Maio 2005</i>	<i>Sim</i> ⁶
<i>Nortel/Avaya 5510, 5520, 5530</i>		<i>Não</i>
<i>Cisco Catalyst 3750</i>	<i>Julho 2007</i>	<i>Não</i>
<i>HP Procurve 2800</i>	<i>Agosto 2008</i>	<i>Sim</i> ⁷
<i>HP Procurve 3500</i>		<i>Não</i>
<i>HP Procurve 5400</i>		<i>Não</i>
<i>Cisco Catalyst 35xx</i>	<i>Janeiro 2009</i>	<i>Sim</i> ¹
<i>Arista 7500</i>	<i>Julho 2012</i>	<i>Não</i>

Com base nas informações dispostas na Tabela 1, conclui-se que o suporte a novos modelos de comutadores não possui uma frequência constante; por exemplo, apenas um modelo foi adicionado nos últimos quatro anos. Adicionalmente, observa-se que mais da metade dos modelos suportados (9 dos 16 modelos) já atingiram o fim de sua vida útil e estão descontinuados pelos fabricantes, como mostra a terceira coluna da Tabela 1.

Estas condições acarretam no comprometimento do processo de expansão das infraestruturas de Emulab já existentes e da criação de novos ambientes. Isso deve-se ao fato de que no caso da aquisição de novos equipamentos apenas sete modelos de comutadores de quatro fabricantes diferentes estão disponíveis; e no caso da adoção dos demais modelos seria necessária a aquisição de equipamentos de segunda mão. Isso decorre das restrições impostas pelo mecanismo de configuração de rede do Emulab,

⁴ Cisco – EOS and EOL Products:

http://www.cisco.com/en/US/products/hw/switches/prod_category_end_of_life.html

⁵ Brocade – Product Recycling: <http://www.brocade.com/company/corporate-responsibility/corporate-citizenship/product-recycling/index.page>

⁶ Avaya – End of Life Products: <http://downloads.avaya.com/css/P8/documents/100113214>

⁷ HP – End of Sales Switches: <http://h17007.www1.hp.com/us/en/products/eos/switches.aspx>

detalhado na Seção 2.1.2.1, que depende da criação de módulos específicos para cada modelo de comutador a ser suportado pelo Emulab. Pode-se dizer, então, que o modelo de operação atual do Emulab não é escalável em termos de número de modelos de equipamentos suportados.

2.1.2 Alocação do Experimento

A criação do experimento inicia-se quando o usuário envia o arquivo de descrição do seu experimento, empregando-se uma sintaxe similar à do simulador ns-2 (MCCANNE, et al. 1997), ou usa a interface gráfica do Emulab para criar seu experimento. Após a interpretação do arquivo carregado ou gerado a partir da interface gráfica, o experimento é criado e configurado no banco de dados do Emulab. Neste ponto, o experimento existe, mas ainda não foram alocados recursos para o mesmo.

O processo de **alocação de recursos de um experimento** ocorre quando um experimentador efetua a solicitação para instanciar o experimento e os recursos requisitados estiverem disponíveis. Se estas duas condições forem verdadeiras, o processo ocorre por meio de três sub-processos (EMULAB WIKI, 2013):

- **Mapeamento da topologia da rede:** a topologia descrita pelo experimentador é analisada e verifica-se se há recursos físicos disponíveis (nós locais e interfaces de redes) para alocar um experimento com topologia equivalente. Caso haja recursos suficientes disponíveis, é criada no banco de dados do Emulab uma lista dos recursos alocados para o experimento em questão, onde serão indicados os nós físicos que correspondem a cada nó solicitado, as interfaces de cada nó que serão configuradas com os endereços IPs requisitados, e as VLANs que foram criadas para recriar a rede deste experimento.
- **Configuração da rede:** após o termino do mapeamento da topologia de rede, inicia-se a configuração da rede de

experimentação. Utilizando a lista de nós alocados resgatada do banco de dados do Emulab e suas respectivas VLANs, é possível verificar todos os comutadores da rede de experimentação que precisam ser configurados para criar a topologia desejada pelo experimentado. As VLANs do experimento são criadas em cada comutador e as respectivas interfaces ligadas aos nós locais do experimento são associadas a cada VLAN. As interfaces entre comutadores de experimentação diferentes são conectadas utilizando-se VLAN *trunking*, para possibilitar o tráfego de pacotes de diferentes experimentos.

- **Inicialização do sistema operacional:** cada nó local alocado no experimento é inicializado pela rede pelo PXE (*Preboot eXecution Environment*) em um SO temporário carregado na memória. Em seguida, as imagens dos SOs escolhidos são transferidas do servidor central ao disco dos respectivos nós por meio da rede de controle. Esta transferência utiliza *multicast* no caso de múltiplos nós receberem a mesma imagem de SO. Este sub-processo ocorre em paralelo ao de configuração da rede de experimentação.

Esses processos estão ilustrados na Figura 2.

Figura 2 - Processos da alocação de recursos de um experimento

O processo de **alocação dos recursos de um experimento** consiste basicamente de módulos distintos, responsáveis pelos sub-processos:

1. O módulo **assign** é responsável por efetuar o mapeamento da topologia requisitada pelo usuário no *hardware* disponível;
2. O módulo **snmpit** realiza a configuração dos comutadores da rede de experimentação para prover a conectividade e o isolamento dos nós de cada experimento, por meio de VLANs.
3. O módulo **frisbee** carrega os sistemas operacionais escolhidos nos respectivos nós, e, por fim, configura o registro DNS de cada nó e o mapeamento dos diretórios de rede.

Após o ambiente de teste ser configurado por completo, é enviado um e-mail ao usuário informando quais recursos foram alocados e qual deve ser o procedimento para acessá-los. Para utilização da rede alocada, o Emulab oferece um gerador de tráfego CBR (*Constant Bit Rate*), empregando-se a ferramenta TG Tool Set (WHITE, et al, 2002).

Quando o usuário terminar o experimento, os recursos que estiverem alocados são liberados e a descrição do ambiente de teste é excluída do banco de dados. Enquanto o experimento não for terminado, é permitido ao usuário liberar ou realocar os recursos do experimento quantas vezes for desejado, considerando que os recursos requisitados estejam disponíveis.

O mecanismo proposto nesta dissertação baseia-se no módulo de configuração de rede (**snmpit**). Por este motivo, a próxima seção (2.1.2.1) apresenta detalhes sobre o funcionamento deste módulo.

2.1.2.1 Módulo de Configuração da Rede

Os comutadores da rede de experimentação são controlados usando o módulo de configuração de rede (**snmpit**), que fornece uma interface de alto nível para MIBs SNMP específicas de cada modelo de comutador. O **snmpit** possui um módulo específico para cada modelo, contendo os métodos para configuração das interfaces dos comutadores e das VLANs. A configuração da rede faz-se necessária nas etapas de alocação e liberação de um experimento e seus recursos. Um conjunto de métodos do módulo **snmpit** é executado para efetuar as operações necessárias em cada etapa. Para cada uma das etapas citadas, foram identificadas as operações realizadas e os respectivos métodos envolvidos:

a. Operações efetuadas na alocação de um experimento:

- a1. **Escolha de uma VLAN livre:** Entre as VLANs existentes em cada comutador do experimento, é verificado se há alguma VLAN que não possui interfaces associadas. Se houver, esta VLAN é escolhida. Se não houver nenhuma VLAN livre, uma nova VLAN é criada. Os métodos empregados são: findVlan, listVlans, vlanHasPorts, createVlan;
- a2. **Associação das interfaces à VLAN escolhida:** Por meio do banco de dados do Emulab, é possível resgatar a qual interface cada nó está ligado. Para cada nó do experimento, a respectiva interface é configurada na VLAN escolhida. O método utilizado é o setPortVlan;
- a3. **Extensão da VLAN entre os comutadores:** No caso dos nós do experimento serem alocados em comutadores diferentes, é habilitada a VLAN do experimento no túnel de VLANs entre

esses comutadores. Os métodos empregados são: `enablePortTrunking`, `setVlansOnTrunk`;

a4. Configuração das interfaces com os atributos desejados:

São configurados, na interface de cada nó, a taxa de vazão e o tipo do controle de fluxo (duplex ou half-duplex) descrito no experimento. Caso alguma interface do nó não esteja em uso, ela é desativada. Os métodos utilizados são: `getStats` e `portControl`.

b. Operações efetuadas na liberação de um experimento:

b1. Remoção das interfaces na VLAN: Cada interface conectada a cada nó do experimento é removida da VLAN a qual está associada. O método utilizado é o `removePortsFromVlan`;

b2. Remoção do VLAN *trunking* entre os comutadores: Se a VLAN do experimento for a única entre os comutadores, o VLAN *trunking* é removido. Caso contrário, apenas a VLAN do experimento é desconfigurada e o VLAN *trunking* é mantido para uso por VLANs dos outros experimentos existentes. Os métodos empregados são: `setVlansOnTrunk` e `disablePortTrunking`;

b3. Remoção da VLAN nos comutadores: A VLAN do experimento é removida de todos os comutadores que estavam alocados pelo usuário para o experimento. O método utilizado é o `removeVlan`;

b4. Restauração da configuração das interfaces: Os atributos das interfaces do comutador que estavam alocadas no experimento são configurados para os valores padrão e a interface é desativada. Os métodos empregados são: `getStats` e `portControl`.

Cada um dos métodos do módulo **snmpit**, destacados nas etapas citadas acima, estão detalhados a seguir:

1. **portControl**: Configura a taxa de vazão e o tipo do controle de fluxo das interfaces, assim como ativar e desativar cada interface;
2. **listPorts**: Lista todas as interfaces do comutador;
3. **getStats**: Captura estatísticas sobre as interfaces do comutador;
4. **findVlan(s)**: Informa se um conjunto de VLANs existe no comutador;
5. **vlanNumberExists**: Verifica se uma VLAN com um determinado número existe no comutador;
6. **createVlan**: Cria uma VLAN com um determinado identificador no comutador;
7. **setPortVlan**: Adiciona um conjunto de interfaces a uma determinada VLAN;
8. **delPortVlan**: Remove e desativa um conjunto de interfaces de uma determinada VLAN;
9. **removePortsFromVlan**: Remove todas as interfaces de uma determinada VLAN;
10. **removeSomePortsFromVlan**: Remove um conjunto de interfaces de uma determinada VLAN;
11. **removeVlan**: Remove uma determinada VLAN, o que implica em remover todas as interfaces e apagar qualquer registro da VLAN do comutador;
12. **vlanHasPorts**: Verifica se há alguma interface associada a uma determinada VLAN;
13. **listVlans** : Lista todas as VLANs do comutador;
14. **setVlansOnTrunk**: Adiciona e remove rótulos de VLANs (IEEE 802.1Q) a pacotes de uma determinada interface;
15. **enablePortTrunking**: Ativa VLAN *trunking* em uma determinada interface;
16. **disablePortTrunking**: Desativa VLAN *trunking* em uma determinada interface.

Todo comutador de uma rede de experimentação do Emulab deve suportar todos os métodos do módulo **snmpit**. Porém, modelos de comutadores de fabricantes diferentes não implementam da mesma forma todas as funcionalidades necessárias, como por exemplo a criação de VLANs

e VLAN *trunking*. Essas funcionalidades podem estar disponíveis em MIBs proprietárias, como no caso de equipamentos Cisco, ou não estarem disponíveis por meio do protocolo SNMP e precisarem ser configuradas por meio do acesso à console do dispositivo, como no caso dos equipamentos Nortel suportados no Emulab (EMULAB WIKI, 2013).

Sendo assim, a maior parte do trabalho de se adicionar o suporte a um novo comutador no Emulab está em descobrir o modo de configurar as funções necessárias para o módulo **snmpit** de cada dispositivo. Por exemplo, cerca de vinte MIBs proprietárias diferentes foram usadas no módulo para comutadores Cisco. De acordo com o desenvolvedor responsável pelos módulos para os comutadores HP, Nortel e Brocade (antiga Foundry), o tempo necessário para adicionar suporte a um novo modelo é estimado em duas semanas para pessoas já familiarizadas com o funcionamento do **snmpit** e dos módulos específicos já existentes; em compensação, o tempo estimado para alguém que não seja habituado a este processo é de até dois meses, baseando-se na experiência do desenvolvedor do Emulab (SKLOWER, 2008).

2.2 SDN e OpenFlow

Esta seção apresenta a arquitetura SDN (*Software-Defined Networking*) e o protocolo OpenFlow, iniciando-se com uma descrição em alto nível e apresentando-se, em seguida, os elementos dessa arquitetura. Além disso, são introduzidos os principais conceitos utilizados na descrição da arquitetura do módulo OpenFlow para o Emulab, apresentada no Capítulo 3.

2.2.1 Visão Geral Sobre SDN

Na arquitetura SDN, o plano de controle e de dados são separados. Além disso, a inteligência e a monitoração do estado da rede estão logicamente centralizados, e a informação sobre a topologia da rede pode ser exportada para diferentes aplicações. Como resultado, SDN oferece a

automação e o controle da rede, permitindo a construção de redes flexíveis (ONF, 2012).

A Figura 3 apresenta a interação entre os elementos que compõem a arquitetura SDN. A inteligência de rede é (logicamente) centralizada nos controladores que são baseados em *software*, mantendo uma visão global da rede. Como resultado, a rede é apresentada às aplicações como um único comutador lógico. A interface (*Southbound API*) entre os dispositivos de rede (comutadores e roteadores) e o controlador é realizada por meio de um protocolo de padrão aberto, como por exemplo, o OpenFlow. Enquanto a comunicação entre o controlador e as aplicações da rede é realizada por meio de uma interface de alto nível, (*Northbound API*), como por exemplo, REST API⁸ ou JSON-RPC⁹.

Figura 3 - Arquitetura SDN

As características do uso da arquitetura SDN são (ONF, 2012):

- Gerenciamento e controle centralizados de dispositivos de rede de diferentes fornecedores;

⁸ REST API define uma arquitetura para desenvolvimento de *web services*, que objetiva minimizar a latência e o número de comunicação pela rede e ao mesmo tempo busca maximizar a atomicidade e escalabilidade de cada componente (FIELDING, 2002).

⁹ JSON-RPC é um protocolo RPC (*Remote Procedure Call*) leve que utiliza o formato JSON (*JavaScript Object Notation*) (CROCKFORD, 2006) para a serialização de dados estruturados.

- Melhoria da automação do gerenciamento da rede, usando APIs comuns para abstrair os detalhes da rede para outros sistemas, como sistemas de orquestração, sistemas de provisionamento e aplicações de rede;
- Introdução rápida de inovações a partir da capacidade de fornecer novos recursos e serviços de rede, sem a necessidade de configurar dispositivos individuais e sem depender de alterações no *hardware*;
- Viabilidade de programação de funcionalidades adicionais de rede por terceiros (e não apenas pelos próprios fabricantes de equipamentos), empregando-se ambientes de programação convencionais;
- Aumento da confiabilidade da rede e de sua segurança, como resultado do gerenciamento centralizado e automatizado dos dispositivos da rede, da aplicação uniforme de políticas e da menor incidência de erros de configuração;
- Controle de rede mais granular com a capacidade de aplicar políticas abrangentes e de grande alcance.

2.2.2 Visão Geral Sobre OpenFlow

OpenFlow é a primeira interface padrão entre os elementos **comutador** e **controlador** da arquitetura SDN (ONF, 2012) (vide Figura 3). Especifica abstrações que definem uma interface padrão por meio da qual as entradas na tabela de encaminhamento do comutador podem ser configuradas utilizando-se um elemento externo, o **controlador**. O protocolo OpenFlow define uma abordagem aberta e padronizada para o comutador se comunicar por meio de um **controlador** externo (MCKEOWN et al. 2008).

O OpenFlow explora a existência de tabelas TCAM (*Ternary Content-Addressable Memory*) em comutadores Ethernet e roteadores, que

tradicionalmente são usadas para armazenar regras de funcionalidades como *firewall*, NAT¹⁰ (*Network Address Translation*), QoS (*Quality of Service*). Tais tabelas variam em formato e conteúdo entre diferentes fornecedores. Por meio da identificação de um conjunto de funções suportadas pela maioria dos comutadores e roteadores, o OpenFlow definiu um conjunto comum de abstrações para manipular de uma forma padrão as tabelas de diferentes dispositivos de rede, independentemente da implementação específica de cada fornecedor.

O funcionamento do OpenFlow baseia-se no conceito de fluxos. Um fluxo representa pacotes que possuem um conjunto de campos do cabeçalho de mesmo valor. Para descrever um fluxo, o protocolo OpenFlow na versão 1.0 (OPENFLOW SWITCH CONSORTIUM, 2009) utiliza doze campos de cabeçalho, apresentados na Tabela 2.

Tabela 2 – Campos definidos no protocolo OpenFlow 1.0

Interface	Ethernet			VLAN		IP				TCP/UDP	
Entrada	MAC de origem	MAC de destino	Tipo de quadro	Identificador	Prioridade	Origem	Destino	Protocolo	Tipo de Serviço	Porta de origem	Porta de destino

Um fluxo é descrito por 1 ou mais campos do cabeçalho. Todo campo que não for definido no fluxo recebe um valor coringa (*wildcard*); neste caso o fluxo vai corresponder a qualquer valor neste campo do pacote. Por exemplo, um fluxo que descrever apenas a porta de origem como 1 estará atribuindo o valor coringa aos demais 11 campos. Neste caso, o fluxo representa qualquer pacote que chegar ao comutador pela porta 1.

¹⁰ NAT é o método pelo qual os endereços IP de uma rede privada são mapeados em um endereço IP de outra rede, permitindo a usuários dentro da rede privada acessar de forma transparente serviços na rede externa, na maioria dos casos a Internet. (SRISURESH e EGEVANG, 2001)

2.2.2.1 Comutador OpenFlow

Um comutador OpenFlow é caracterizado por uma tabela de fluxo para encaminhamento de pacotes e um canal seguro para um **controlador** externo, representados na Figura 4. É por meio do canal seguro que o **controlador** gerencia o comutador usando o protocolo OpenFlow.

Figura 4 - Comutador OpenFlow, seus elementos e a comunicação com o controlador

A tabela de fluxo contém um conjunto de entradas de fluxo. Cada entrada de fluxo na tabela é formada pelos valores de cabeçalho para seleção dos pacotes, pelos contadores, e por um conjunto de zero ou mais ações a serem aplicadas aos pacotes correspondentes:

- **Campos de cabeçalho:** são usados para separar o tráfego da rede em fluxos que correspondam a um ou mais campos do pacote. Os campos de cabeçalhos utilizados no protocolo OpenFlow foram apresentados na Tabela 2;
- **Contadores:** armazenam estatísticas sobre cada fluxo na tabela do comutador, a fim de manter o controle do número de pacotes e de bytes de um determinado fluxo, e o tempo decorrido desde a instalação do fluxo no comutador.
- **Ações:** especifica a maneira pela qual os pacotes de um fluxo serão processados. Uma ação pode ser de um dos seguintes tipos: 1) encaminhar o pacote a uma ou mais interfaces, opcionalmente reescrevendo alguns campos do cabeçalho; 2) descartar o pacote; ou 3) encaminhar o pacote ao controlador.

Todo pacote processado pelo comutador é comparado com a tabela de fluxos. Se uma entrada correspondente ao pacote for encontrada, todas as ações relativas à respectiva entrada são realizadas sobre o pacote, por exemplo, a ação de encaminhar um pacote a uma determinada interface. Se nenhuma correspondência for encontrada, o pacote é encaminhado ao **controlador** por meio do canal seguro. O **controlador** fica responsável por determinar como lidar com pacotes sem entradas de fluxo válidas e por efetuar o gerenciamento da tabela do comutador, adicionando e removendo entradas de fluxo.

2.2.2.2 Controlador OpenFlow

O **controlador** é a entidade central responsável pelas funcionalidades do plano de controle dos comutadores OpenFlow. O **controlador** utiliza o protocolo OpenFlow como interface para criar, modificar e controlar as entradas de fluxos das tabelas dos comutadores. O **controlador** é executado em um servidor acessível aos comutadores por meio de um canal seguro (MCKEOWN et al. 2008).

Para possibilitar a criação de aplicações e serviços de rede que possam aproveitar a visão centralizada da rede criada pelos **controladores**, cada **controlador** define uma API (*Northbound API*) que exporta informações de alto nível sobre a rede e permite a manipulação dessas informações por meio de chamadas remotas (ONF, 2012).

O NOX (GUDE et Al, 2008) foi a primeira implementação de um **controlador** OpenFlow criada em Stanford pelo mesmo grupo que criou o protocolo OpenFlow. Outros grupos criaram novos **controladores** com características diferentes, como por exemplo, a linguagem de base e as funcionalidades suportadas. Na Tabela 3 foram compiladas informações sobre os controladores OpenFlow, listando a linguagem utilizada, data de criação, última versão estável e última atualização no código fonte. Essas informações são utilizadas, posteriormente, como base para a decisão sobre

as características do **controlador** a ser utilizado na implementação do módulo OpenFlow para o Emulab.

Tabela 3 - Informações sobre controladores OpenFlow

Controlador	Linguagem	Data de Criação	Última Versão¹¹	Última Atualização¹²	Northbound API
NOX	C	Mar 2009	Jul 2012	Jun 2013	REST API
POX	Python	Jun 2009	Mai 2013	Jun 2013	JSON-RPC
Beacon	Java	Abr 2010	Out 2012	Out 2012	REST API
Maestro	Java	Nov 2010	Ago 2011	Mai 2011	N/D
FlowER	Erlang	Jul 2011	N/D	Jun 2013	N/D
Trema	C/Ruby	Abr 2011	Mar 2013	Jun 2013	REST API
NodeFlow	JavaScript	Out 2011	N/D	Jun 2012	N/D
Floodlight	Java	Dez 2011	Out 2012	Jun 2013	REST API
Ryu	Python	Dez 2011	Jun 2013	Jun 2013	REST API
Jaxon	Java	Mar 2012	Mar 2012	Mar 2012	N/D
OpenDaylight	Java	Mar 2013	N/D	Jun 2013	REST API
MUL	C	Mai 2013	Fev 2013	Mar 2013	N/D

As informações apresentadas na Tabela 3 tornam evidente os projetos de comutadores que não deram continuidade ao seu desenvolvimento. Adicionalmente, é possível identificar os controladores que não implementam a *Northbound* API. Os controladores utilizados nesta pesquisa foram o POX e Ryu. O POX é um controlador em Python recomendado para o desenvolvimento e a prototipagem de softwares de controle de rede, e suporta apenas a versão 1.0 do protocolo OpenFlow (POX, 2013). O Ryu é um controlador que fornece componentes com APIs bem definidas para facilitar o desenvolvimento de aplicativos de gerenciamento de rede e controle; suporta as versões OpenFlow de 1.0 a 1.3 (RYU, 2013).

2.2.2.3 Modos de operação do controlador OpenFlow

O controlador OpenFlow apresenta dois modos de funcionamento: o reativo e o proativo (FERNANDEZ, 2013). No modo reativo, o primeiro pacote de cada fluxo recebido pelo comutador é enviado ao controlador, que, a partir do conteúdo do pacote recebido, insere uma regra de fluxo nos comutadores

¹¹ Data da última versão estável, informação referente a 25 de junho de 2013.

¹² Data da última atualização no código fonte, informação referente a 25 de junho de 2013.

OpenFlow da rede. Esta abordagem apresenta, como ponto positivo, o uso eficiente da memória da tabela de fluxo do comutador, devido ao fato de estarem instaladas no comutador apenas as regras de fluxos dos pacotes que estão trafegando na rede em um determinado período; quando estes pacotes não estão mais trafegando na rede, as regras podem ser removidas. Porém, como ponto negativo dessa abordagem, há o fato de cada novo fluxo na rede gerar um pequeno tempo de configuração adicional para instalar as regras no comutador. Outro ponto é a dependência do controlador para o funcionamento da rede, sendo que, no caso da comunicação entre o comutador e o controlador ser interrompida, o comutador não tem meios de tratar novos pacotes e o funcionamento da rede fica comprometido.

No modo proativo, o controlador preenche antecipadamente a tabela de fluxo do comutador. Nesta abordagem, não há tempo adicional para a instalação de regras de fluxos durante a execução da rede, porque todas as regras foram instaladas em uma etapa anterior. Com isto, se o comutador perde a conexão com o controlador, as regras instaladas permitem manter o funcionamento da rede. No entanto, a necessidade de se definirem todas as regras de fluxos antecipadamente pode criar problemas, como por exemplo, exceder o tamanho da tabela de fluxos do comutador. Uma estratégia para contornar esta limitação é utilizar a agregação de regras a partir dos valores coringa (*wildcard*) para descrever um conjunto fluxos utilizando apenas uma regra.

2.3 Trabalhos Relacionados

Existem algumas abordagens de utilização do OpenFlow no ambiente Emulab, mas para fins diferentes aos apresentados neste trabalho. Há implementações em *software* de comutadores OpenFlow, como o Open vSwitch (PFAFF, 2009), que podem ser instalados nos nós locais e serem utilizados para fazer experimentos em OpenFlow. Existe, também, uma implementação de um comutador OpenFlow para placas NetFPGAs (NAOUS, 2008), que é suportado e está disponível no Emulab de Utah. Estas duas

abordagens do uso do OpenFlow no Emulab são empregadas para utilizar o OpenFlow nos experimentos e não são utilizados pela infraestrutura do Emulab para oferecer novos serviços e funcionalidades.

O comutador HP ProCurve modelos 5400 e 3500, suportados pelo Emulab, possuem um *firmware* com suporte experimental ao OpenFlow. Porém, o OpenFlow é disponibilizado pelo Emulab exclusivamente para ser utilizado em experiências, e a configuração desses modelos utiliza o módulo de configuração de rede por meio do SNMP, descrito na seção 2.1.2.1.

O funcionamento do mecanismo proposto neste trabalho é comparável à utilização do controlador de virtualização FlowVisor (SHERWOOD, 2009), juntamente com um controlador OpenFlow com encaminhamento por aprendizado de endereço MAC (*L2 learning switch*). A solução apresentada nesta dissertação diferencia-se deste modelo quanto ao modo de utilização do protocolo OpenFlow. Uma solução composta pelo FlowVisor juntamente com o controlador OpenFlow utiliza o modo reativo do protocolo OpenFlow, no qual todo o primeiro pacote de cada fluxo que chega ao comutador é encaminhado ao plano de controle (FlowVisor e controlador OpenFlow), o que impõe uma carga adicional considerável em comparação ao funcionamento de um comutador tradicional. A estratégia utilizada nesta dissertação utiliza o modo proativo do OpenFlow, conforme descrito na Seção 2.2.2.3, e, por este motivo, oferece menor impacto de desempenho.

O LegacyFlow (FARIAS, 2011) busca o funcionamento de comutadores convencionais e do OpenFlow em uma mesma rede, porém utilizando uma abordagem inversa à apresentada nesta dissertação. O LegacyFlow apresenta uma arquitetura para acomodar comutadores convencionais em uma rede OpenFlow, traduzindo ações OpenFlow em configurações específicas de cada fornecedor para os respectivos comutadores. Este procedimento utiliza interfaces com SNMP ou CLI (*Command Line Interface*). O LegacyFlow apresenta uma abordagem oposta à proposta desta dissertação, pois busca implementar funções do protocolo OpenFlow em equipamentos convencionais, enquanto o objetivo

proposto nesta dissertação é implementar configurações equivalentes a de comutadores convencionais em comutadores OpenFlow.

2.4 Considerações Finais do Capítulo

A primeira seção deste capítulo serviu como base para a fundamentação da problemática alvo deste trabalho, caracterizada como a limitação da escalabilidade do Emulab por conta da utilização de interfaces não padronizadas. Além de uma visão geral sobre o ambiente de experimentação Emulab, foram apresentados com maiores detalhes: os mecanismos de alocação e liberação de experimentos e os comutadores da rede de experimentação, que são elementos fundamentais para o estudo da limitação encontrada e para o desenvolvimento de uma solução.

A segunda seção deste capítulo apresentou o SDN e o protocolo OpenFlow, que constituem uma arquitetura aberta para o desenvolvimento de novas aplicações e serviços de redes por meio de uma interface padronizada para os dispositivos de redes, como por exemplo, os comutadores. Essas tecnologias oferecem a possibilidade de efetuar a programação da tabela de encaminhamento de comutadores e de outras configurações em dispositivos de diferentes fabricantes. Essa característica possibilita o desenvolvimento de um mecanismo de configuração de rede no Emulab, que suporte um número maior de dispositivos de modelos e fornecedores variados.

3 MECANISMO DE SUPORTE A OPENFLOW

Este capítulo detalha o mecanismo proposto para integrar comutadores OpenFlow ao ambiente Emulab. O módulo OpenFlow tem por objetivo possibilitar ao *software* de controle do Emulab configurar os comutadores OpenFlow durante o processo de alocação da rede do experimento em recursos físicos disponíveis, de forma equivalente ao processo efetuado nos comutadores convencionais pelo módulo **snmpit** detalhado na Seção 2.1.2.1.

3.1 Especificação de Requisitos

A partir do estudo efetuado sobre o funcionamento do mecanismo de configuração de rede do Emulab, foram especificados os requisitos funcionais que devem ser atendidos pelo módulo OpenFlow, tomando-se como base o funcionamento dos comutadores convencionais da rede de experimentação. De forma semelhante, foram especificados os requisitos funcionais dos controladores OpenFlow e dos comutadores OpenFlow.

Os requisitos funcionais foram especificados de modo a garantir que o módulo OpenFlow tenha o funcionamento compatível com os módulos existentes do Emulab. Tais requisitos incluem:

- RF1: Possibilitar a configuração remota de comutadores OpenFlow de diferentes modelos e fabricantes;
- RF2: Aplicar configurações específicas no comutador OpenFlow durante a fase de alocação de recursos de um experimento;
- RF3: Definir o domínio de *broadcast* entre todos os nós de um mesmo experimento;
- RF4: Utilizar um mecanismo de encaminhamento de pacotes entre os nós;
- RF5: Garantir o isolamento do tráfego de um experimento para outro;
- RF6: Interligar o comutador OpenFlow aos comutadores convencionais por meio de VLANs *trunking*;

RF7: Efetuar a configuração da taxa de vazão e o tipo de controle de fluxo das interfaces do comutador.

3.2 Operação do Emulab com OpenFlow

Na etapa de configuração de rede, quando um comutador OpenFlow for alocado em um experimento, o Módulo OpenFlow é chamado no lugar do módulo **snmpit**. Cada comutador OpenFlow é, então, configurado de acordo com a topologia e a especificação do experimento usando o protocolo OpenFlow. As operações realizadas pelo protocolo OpenFlow estão descritas na Seção 3.3. Para transmitir os comandos OpenFlow ao comutador é utilizada a *Northbound API* do **controlador** OpenFlow. A Figura 5 ilustra o processo de alocação de recursos do Emulab com a inclusão do módulo OpenFlow.

Figura 5 - Processo de alocação de recursos com o módulo de suporte a OpenFlow

Depois de configurados, os comutadores OpenFlow deverão prover a conectividade e o isolamento entre os nós de um experimento, bem como atender aos requisitos detalhados na Seção 3.1.

Conforme explicado na Seção 2.2.2.3, o protocolo OpenFlow opera em dois modos: modo proativo e modo reativo. No modo proativo, todas as regras de fluxos necessárias para prover a conectividade e o isolamento da rede como um todo são instaladas no comutador OpenFlow, em uma fase anterior ao funcionamento da rede. Neste caso, o controlador não é utilizado durante o encaminhamento dos pacotes. Este modo de funcionamento não impõe tempo adicional para configuração no primeiro pacote de cada fluxo, como é o caso do modo reativo. Para atender este objetivo, foi especificado um conjunto de regras de fluxos para os comutadores OpenFlow, instaladas na fase de configuração da rede. Estas regras de fluxos foram definidas para possibilitar a utilização do modo proativo do protocolo OpenFlow. As regras de fluxos definidas estão descritas na Seção 3.4.

3.3 Definição das Operações do Protocolo OpenFlow

O protocolo OpenFlow baseia-se no conceito de fluxos para descrever e aplicar regras que possibilitam a configuração de um comutador OpenFlow, diferentemente de um comutador convencional. Para possibilitar a representação dos 16 métodos necessários para desenvolver o módulo de configuração de rede de um comutador convencional (Seção 2.1.2.1) em operações suportadas pelo protocolo OpenFlow, foram identificadas 4 grupos de operações que permitiram categorizar e descrever o funcionamento desses métodos. Cada grupo de operação e os métodos do módulo de configuração de rede que o compõem, estão descritos a seguir:

1. Consulta de informações sobre as interfaces do comutador:

Consiste na requisição de informações tais como: número de interfaces do comutador, a situação administrativa (ativada ou desativada), taxa de vazão e o tipo do controle de fluxo de cada interface (*half* ou *full*-duplex). Os métodos empregados são `listPorts`, `getStats`;

2. Consulta de informações sobre as VLANs do comutador: Consiste na requisição de informações sobre as VLANs atualmente

configuradas no comutador. Como exemplos podem-se citar: lista das VLANs e VLANs *trunking* existentes; e lista das interfaces configuradas em cada VLAN. Os métodos empregados são `findVLAN(s)`, `vlanNumberExists`, `vlanHasPorts`, `listVlans`;

3. **Configuração de VLANs no comutador:** Consiste em adicionar ou remover a VLAN ou VLAN *trunking* do comutador, bem como adicionar ou remover as interfaces às VLANs. Os métodos empregados são `createVlan`, `setPortVlan`, `delPortVlan`, `removePortsFromVlan`, `removeSomePortsFromVlan`, `removeVlan`, `setVlansOnTrunk`, `enablePortTrunking`, `disablePortTrunking`.
4. **Configuração das interfaces do comutador:** Consiste na alteração dos atributos de uma interface específica ou de um conjunto de interfaces. Os atributos modificados são: a situação administrativa, a taxa de vazão e o tipo do controle de fluxo. O método usado é `portControl`;

Por meio da análise da especificação do protocolo OpenFlow versão 1.0, foram identificadas 4 operações do protocolo OpenFlow que implementam as funcionalidades listadas acima. Cada operação do protocolo OpenFlow, enumerada a seguir, corresponde ao item de mesmo número na lista anterior:

1. **Estatísticas das interfaces do comutador:** A partir de requisições `OFPST_PORT` são obtidas informações sobre as interfaces físicas do comutador. Esta mensagem permite solicitar a lista as interfaces do comutador, as respectivas taxas de vazão e se cada interface está ativada ou desativada. Corresponde aos métodos **`listPorts`** e **`getStats`**;
2. **Estatísticas de fluxos do comutador:** Informações sobre os fluxos de cada regra são solicitadas a partir de requisições `OFPST_FLOW_STATS`. Por meio desta requisição, é possível resgatar todas as regras instaladas no comutador, a partir de uma busca na lista de regras, possibilitando:
 - Verificar se existe uma regra com uma determinada VLAN. Corresponde aos métodos **`findVLAN(s)`** e **`vlanNumberExists`**;

- Verificar quais interfaces estão associadas à VLAN de uma determinada regra. Corresponde ao método **vlanHasPorts**;
 - Listar todas VLANs do comutador. Corresponde ao método **listVlans**.
3. **Modificação de Regras de Fluxo:** Modificações na tabela de fluxos a partir do controlador são feitas empregando-se a mensagem OFPT_FLOW_MOD. Esta mensagem é usada para adicionar, modificar ou remover determinadas regras de fluxos, o que possibilita executar as seguintes operações:
- Criar uma nova regra com uma VLAN e interface específica. Corresponde ao método **createVlan**;
 - Alterar uma regra de uma determinada VLAN para adicionar uma nova interface; Corresponde ao método **setPortVlan**;
 - Adicionar múltiplas regras com a mesma interface, porém com rótulos de VLAN diferentes. Corresponde aos métodos **setVlansOnTrunk** e **enablePortTrunking**;
 - Alterar uma regra com uma VLAN específica para remover uma determinada interface. Corresponde ao método **delPortVlan**;
 - Alterar uma regra com uma VLAN específica para remover uma ou mais interfaces. Corresponde aos métodos **delPortVlan** e **removeSomePortsFromVlan**;
 - Remover uma regra com uma VLAN específica. Corresponde ao método **removeVlan, disablePortTrunking**;
4. **Modificação da configuração das interfaces do comutador:** A mensagem OFPT_PORT_MOD possibilita modificar as configurações de uma interface física, como por exemplo, definir uma interface como desativada ou ativada. O OpenFlow 1.0 não possui mecanismo para configurar a taxa de vazão; essa funcionalidade foi apenas adicionada na versão 1.3, chamada de Meters. Informações sobre esse mecanismo estão disponíveis no item **Configuração da taxa de vazão das interfaces** da Seção 5.1. Corresponde ao método **portControl**

As modificações de regras de fluxos, citadas no item 4, devem ser efetuadas para configurar os comutadores OpenFlow de forma a apresentar

um funcionamento equivalente aos comutadores convencionais configurados pelo módulo **snmpit**, provendo a conectividade e o isolamento para os experimentos. As regras de fluxos selecionadas, para efetuar o comportamento desejado, estão descritas na seção a seguir.

3.4 Definição das Regras de Fluxos

Na etapa de configuração da rede do Emulab, os nós físicos que serão utilizados no experimento já foram definidos na etapa anterior de mapeamento da topologia da rede.

Com isto, é possível consultar informações sobre cada nó alocado, mais especificamente o endereço MAC da interface de cada nó e a qual interface do comutador da rede de experimentação esse nó está conectado. Por meio destas informações, é possível definir as regras de fluxos que descrevem todos os tráfegos de rede possíveis em um determinado experimento.

Para exemplificar o processo de mapeamento da topologia da rede de experimentos em recursos físicos e evidenciar as informações disponíveis ao processo de configuração de rede, a Figura 6 apresenta o mapeamento de dois experimentos distintos em recursos físicos do Emulab. O experimento 1 é formado por 3 nós, que estão conectados por uma mesma rede local, enquanto o experimento 2 é composto por 2 nós conectados por uma segunda rede local, isolada completamente da rede do experimento.

Figura 6 - Exemplo do processo de mapeamento de topologia da rede de experimentos em recursos físicos no Emulab

Os nós de cada experimento são mapeados em PCs físicos disponíveis na infraestrutura do Emulab. A distribuição desses nós físicos na Figura 6 foi escolhida propositalmente com o objetivo de ilustrar a compatibilidade desejada entre o comutador OpenFlow e o comutador convencional.

Todos os nós de um determinado experimento encontram-se em uma mesma VLAN. No experimento 1, os nós 1, 2 e 3 fazem parte da VLAN 1, enquanto no experimento 2, os nós 4 e 5 estão na VLAN 2. Como há nós de um mesmo experimento em comutadores distintos, a VLAN do experimento é estendida entre esses comutadores por um enlace de VLAN *trunking*, representado na Figura 6 por meio de uma linha tracejada entre os comutadores.

As informações referentes ao endereço MAC de cada nó, a interface do comutador em que cada nó está conectado, e as VLANs alocadas estão disponíveis ao módulo OpenFlow e serão utilizadas para efetuar a configuração dos comutadores, estabelecer a conectividade de cada experimento e garantir o isolamento entre eles.

Para configurar o comutador OpenFlow no exemplo apresentado na Figura 6, dois tipos de regras de fluxos são utilizadas. O primeiro tipo delimita o domínio de difusão para nós em uma mesma VLAN, enquanto o segundo tipo faz o encaminhamento baseado na relação entre o endereço MAC do nó e a interface do comutador a qual este MAC está conectado.

A Tabela 4 representa as regras de fluxos necessários para restringir o domínio de difusão (*broadcast*) para o cenário apresentado na Figura 6. É necessária uma regra para cada interface do comutador que pertencer a uma determinada VLAN, sendo que este fluxo vai casar com todos os pacotes que possuírem o rótulo da respectiva VLAN e vai difundir este pacote para os demais nós do experimento. A regra 1, por exemplo, define que todo pacote que chegar ao comutador com o rótulo da VLAN 100 pela interface 1 (*vlan_id=100, in_port:1*), será encaminhado para as demais interfaces da VLAN 100, na qual se encontram os outros nós que fazem parte experimento 1. Nesse caso, as interfaces são 2 e 3 (*out_port:2, out_port:3*), que interconectam o nó 1 aos nós 2 e 3 respectivamente.

Tabela 4 – Fluxos de difusão com domínio restrito por VLAN

Regra	Experimento	Campos do cabeçalho	Ação
1	1	<i>vlan_id=100, in_port:1</i>	<i>out_port:2, out_port:3</i>
2	1	<i>vlan_id=100, in_port:2</i>	<i>out_port:1, out_port:3</i>
3	1	<i>vlan_id=100, in_port:3</i>	<i>out_port:1, out_port:2</i>
4	2	<i>vlan_id=200, in_port:3</i>	<i>out_port:4</i>
5	2	<i>vlan_id=200, in_port:4</i>	<i>out_port:3</i>

Os fluxos apresentados na Tabela 4 são suficientes para oferecer conectividade a cada experimento e simultaneamente manter o isolamento entre eles. Porém, toda transmissão está sendo efetuada por meio de difusão, o que cria uma quantidade desnecessária de tráfego. Para solucionar este problema, é utilizado um segundo conjunto de fluxos que encaminhará pacotes com um endereço MAC de destino conhecido para a interface em que se encontra o nó que possui este endereço MAC.

A Tabela 5 apresenta as regras de fluxos necessários para efetuar o encaminhamento de pacotes pela associação entre o endereço MAC e a interface, baseando-se também no cenário apresentado na Figura 6. É

necessária uma regra de fluxo para cada nó do experimento, sendo que cada regra valida o rótulo da VLAN e o endereço MAC de destino do pacote. Quando um pacote casar com uma regra, a ação efetuada é de encaminhar o pacote para a porta em que se encontra o nó que possui o respectivo endereço MAC. No caso do fluxo 1, todo pacote que possuir o rótulo da VLAN 100 (`vlan_id=100`) e tiver, como destino o endereço MAC do nó 1, (`mac_dst=00:00:00:00:00:01`) será encaminhado para a interface 1 (`out_port:1`) na qual se encontra o respectivo nó com o MAC desejado. As demais regras seguem o mesmo comportamento, para seu respectivo rótulo da VLAN, endereço MAC e interface de saída.

Tabela 5 – Fluxos de encaminhamento por MAC/interface com tratamento de rótulos IEEE 802.1Q

Regra	Experimento	Campos do cabeçalho	Ação
1	1	<code>vlan_id=100,mac_dst=00:00:00:00:00:01</code>	<code>out_port:1</code>
2	1	<code>vlan_id=100,mac_dst=00:00:00:00:00:02</code>	<code>out_port:2</code>
3	1	<code>vlan_id=100,mac_dst=00:00:00:00:00:03</code>	<code>out_port:3</code>
4	2	<code>vlan_id=200,mac_dst=00:00:00:00:00:04</code>	<code>out_port:4</code>
5	2	<code>vlan_id=200,mac_dst=00:00:00:00:00:05</code>	<code>out_port:3</code>

Da forma como estão descritas as regras de difusão (Tabela 4) e de encaminhamento (Tabela 5), é possível que um fluxo case com duas regras ao mesmo tempo. Por exemplo, um pacote que esteja saindo do nó 1 para o nó 2 irá possuir o rótulo da VLAN 100, ao chegar ao comutador pela porta 1, terá, como destino, o endereço MAC do nó 2 (`vlan_id=100,in_port:1,mac_dst=00:00:00:00:00:02`); este fluxo casa tanto com a regra 1 da Tabela 4 quanto com a regra 2 da Tabela 5. Neste caso, a regra mais adequada é o encaminhamento do pacote apenas para a porta 2 (regra 2 da Tabela 5).

Para não haver conflito entre as regras e garantir o funcionamento desejado, foi definido que os fluxos de encaminhamento possuem prioridade sobre os fluxos de difusão. O protocolo OpenFlow possui um campo para definir a prioridade de cada regra de fluxo e este recurso é utilizado no momento de inserir a regra no comutador OpenFlow. Utilizando esse recurso é possível definir a precedência das regras de encaminhamento, para utilizar o mecanismo de difusão apenas quando o MAC não for conhecido.

Com a utilização simultânea dos fluxos para difusão com domínio restrito por VLAN e dos fluxos de encaminhamento, os pacotes destinados a endereços MAC conhecidos serão encaminhados para os respectivos nós; para os demais casos os pacotes serão difundidos a todos os nós do experimento. Este comportamento é desejável para implementar o tratamento de requisições ARP (*Address Request Protocol*), transmissões *broadcast* e *multicast*. Nestes casos, o endereço MAC será FF:FF:FF:FF:FF:FF ou a faixa reservada para *multicast* será (01:00:5E:00:00:00 - 01:00:5E:7F:FF:FF). Como estes endereços MAC não estão atribuídos a nenhuma interface, o pacote será difundido por todo o domínio do experimento, caracterizando o comportamento esperado.

3.5 Implementação do Protótipo

Para avaliar o mecanismo proposto e possibilitar a realização de testes para validar os requisitos funcionais definidos, foi efetuada a implementação de um protótipo do módulo OpenFlow integrado ao ambiente Emulab da USP. As tecnologias utilizadas foram as linguagens Perl e SQL e chamadas HTTP. O código do Emulab é implementado na linguagem Perl. Logo, para facilitar a integração do módulo OpenFlow com os serviços nativos do Emulab, o código principal do protótipo foi, também, desenvolvido em Perl. A linguagem SQL foi utilizada na interação do módulo OpenFlow com o banco de dados do Emulab, possibilitando a consulta das informações necessárias para gerar as regras de fluxos dos comutadores OpenFlow. Por fim, as chamadas HTTP são utilizadas na interação entre o módulo OpenFlow e o controlador OpenFlow no processo de instalação das regras de fluxos no comutador OpenFlow. As regras de fluxos são geradas usando a sintaxe específica do controlador OpenFlow. Em seguida, é feita uma chamada remota contendo a requisição para a instalação das regras utilizando o protocolo HTTP. Essa chamada é recebida pelo controlador, que efetua a instalação dessas regras no comutador OpenFlow. O protótipo foi implementado considerando 4 procedimentos:

1. Consulta ao banco de dados para resgatar informações sobre o experimento:

O objetivo desta etapa é acessar as informações sobre os recursos que foram alocados pelo processo de mapeamento de topologia da rede, que são: os identificadores dos comutadores OpenFlow; VLANs do experimento; endereço MAC dos nós; interfaces do comutador ligadas aos nós; a taxa de vazão; e o tipo de controle de fluxo das portas. Essas informações estão disponíveis no banco de dados do Emulab e, a partir de uma consulta SQL, os dados correspondentes são obtidos, usando-se, como parâmetro, apenas o nome do experimento. A Tabela 6 apresenta, como exemplo, os registros obtidos a partir da consulta ao banco de dados referente ao experimento 1 apresentado na Figura 6.

Tabela 6 – Registros retornados para a consulta do experimento 1

DPID	Porta	VLAN	Endereço MAC	Taxa de vazão
00-04-df-1a-8c-39	1	100	00:00:00:00:00:01	1000 Mbps
00-04-df-1a-8c-39	2	100	00:00:00:00:00:02	1000 Mbps
	3	100	00:00:00:00:00:03	1000 Mbps

Na Tabela 6, o campo DPID (*Data Path ID*) representa o identificador do comutador OpenFlow; comutadores convencionais não possuem esse campo. É a partir do DPID que o controlador instala as regras de fluxo em um determinado comutador OpenFlow. A primeira linha da Tabela 6 representa as informações do nó 1, ligado ao comutador OpenFlow com DPID 00-04-df-1a-8c-39 por meio da porta 1, que possui endereço MAC '00:00:00:00:00:01' e está no experimento com a VLAN de valor 100. A linha 2 da tabela contém informações do nó 2, ligado ao mesmo comutador OpenFlow por meio da porta 2, que possui endereço MAC '00:00:00:00:00:02' e está, também, no experimento com a VLAN de valor 100. No entanto, o nó 3, cujas informações se encontram na linha 3 da Tabela 6, está conectado à porta 3 de um comutador não OpenFlow, que não possui DPID. Nesse caso, há uma condição especial que precisa ser tratada, pois a porta 3 apresentada na Tabela não corresponde a uma porta do comutador OpenFlow. Será necessário utilizar a porta de *uplink* do comutador OpenFlow para acessar o nó 3, tanto para as regras de

difusão como para as de encaminhamento. Para descobrir a porta de *uplink* do comutador OpenFlow é realizada uma nova consulta ao banco de dados na tabela *wires*, responsável por listar as conexões físicas entre os equipamentos, e utilizar o DPID do comutador como índice para a consulta. No exemplo da Tabela 6, a consulta ao banco de dados irá retornar que a porta de *uplink* é a porta 3, conforme a topologia apresentada na Figura 6.

O código SQL que gerou os registros da Tabela 6 está apresentado no Quadro 1:

Quadro 1 – Código SQL para consulta de recursos alocados no experimento

```

1. select nodes.dpid as "DPID",
2.        wires.port2 as "Porta",
3.        vlans.tag as "VLAN",
4.        interfaces.mac as "Endereço MAC",
5.        interfaces.current_speed as "Taxa de vazão"
6. from vlans inner join interfaces on vlans.members REGEXP
7.        CONCAT_WS(':', interfaces.node_id, interfaces.iface)
8.        inner join wires on interfaces.node_id = wires.node_id1
9.        and interfaces.card = wires.card1
10.       inner join nodes on wires.node_id2 = nodes.node_id
11. where vlans.eid = $eid;
```

Foi necessário recuperar campos de quatro tabelas diferentes. A tabela **vlans** contém as informações do experimento desejado representado pela variável \$eid e filtrado na linha 11. Os campos utilizados da tabela **vlans** são: o rótulo da VLAN utilizado no experimento (linha 3) e a lista dos nós físicos alocados no experimento e suas respectivas interfaces (campo **vlans.members** da linha 6). A lista de nós físicos é empregada para combinar a tabela **vlans** com a tabela **interfaces**, utilizando-se as informações do identificador e da interface do nó (campos **interfaces.node_id** e **interfaces.iface** respectivamente da linha 7). A tabela **interfaces** contém as informações de cada interface dos nós e é utilizada para consultar o endereço MAC dos nós físicos (linha 4) e a taxa de transmissão escolhida (linha 5). A tabela **wires** armazena as informações sobre a ligação entre a interface de cada nó físico e a respectiva interface do comutador ao qual ela está conectada. Desta

forma, utilizando-se as informações sobre o identificador do nó e a interface na tabela **interfaces** é possível descobrir a qual comutador e interfaces os nós físicos estão conectados (linha 8 e 9). Da tabela **wires**, são utilizados a interface do comutador (campo **wires.port2** da linha 2) e o identificador do comutador para combinar com a tabela **nodes** (linha 10). A tabela **nodes** possui as informações sobre o comutador, da qual é utilizado o campo DPID (linha 1).

2. Identificar comutadores OpenFlow do experimento e as respectivas interfaces utilizadas:

Utilizando as informações obtidas na etapa anterior, é possível verificar quais são os comutadores OpenFlow do experimento, dado que, todo comutador que for do tipo OpenFlow terá um valor não nulo no campo DPID. Para cada um dos comutadores identificados como OpenFlow será criada uma estrutura contendo uma lista dos endereços MAC e das interfaces que são utilizadas para acessar cada nó do experimento. A Tabela 7 apresenta a estrutura gerada para o comutador OpenFlow no caso do exemplo da Figura 6, que foi, também, utilizado na Tabela 6.

Tabela 7 – Lista de recursos alocados no comutador OpenFlow

DPID	VLAN	Lista de MACs	Lista de Interfaces	Taxa de vazão
00-04-df-1a-8c-39	100	00:00:00:00:00:01, 00:00:00:00:00:02, 00:00:00:00:00:03	1, 2, 3	1000 Mbps

A lista de MACs e interfaces apresentada na Tabela 7 corresponde respectivamente aos nós 1, 2 e 3. No caso do nó 3, como ele não está ligado diretamente ao nó OpenFlow, será utilizada a interface de *uplink* (porta 3) para gerar as regras.

3. Configurar a taxa de vazão e o tipo de controle de fluxo das interfaces:

Utilizando as informações referentes a lista de portas, taxa de vazão e controle de fluxo, cada comutador OpenFlow é configurado a partir de acesso SSH (*Secure Shell*) a partir da console. Como o OpenFlow 1.0 não possui uma forma de configurar a taxa de vazão, foi necessário utilizar a ferramenta *Expect*¹³ (LIBES, 1995) para automatizar o acesso ao console do comutador e efetuar a configuração desse atributo. A ferramenta *Expect* trabalha com o tratamento do texto de saída do console para automatizar a entrada de comandos. Como cada comutador possui uma sintaxe própria para configuração, essa solução precisa ser adequada para cada modelo de comutador. Essa limitação será apresentada com maior detalhes no Capítulo 5, quando serão discutidas as limitações do corrente trabalho e apresentadas propostas de trabalhos futuros.

4. Gerar e instalar as regras de fluxos para cada comutador:

A estrutura de dados apresentada na Tabela 7 do item 2 é utilizada como parâmetro para gerar as regras de fluxos do comutador OpenFlow. A sintaxe das regras geradas nessa etapa depende do controlador que será utilizado, pois cada controlador possui uma forma própria de representar e instalar as regras de fluxos. Para o protótipo implementado, foram criadas duas funções independentes para suportar as regras dos controladores POX e Ryu, sendo que a função que será utilizada dependerá do controlador escolhido. Ambas as funções são formadas por dois laços do tipo *for*: o primeiro é utilizado para criar as regras de encaminhamento e o segundo para formar as regras para difusão de pacotes.

A instalação é efetuada a partir de uma requisição HTTP feita para o controlador OpenFlow. No lado do controlador, essas requisições são recebidas por um servidor JSON-RPC no caso do POX e um servidor REST-API no caso do Ryu (vide Tabela 3 do Capítulo 2). Para desinstalar

¹³ *Expect* é uma ferramenta para automatizar interações com programas que expõem uma interface de terminal de texto como ssh, telnet, password, fsck, ftp, rlogin e tip.

as regras no controlador Ryu é utilizada a mesma sintaxe do procedimento de instalação, porém alterando o parâmetro *add* da URL da requisição HTTP para *delete*. No caso do POX, não foi encontrada nenhuma forma de remover as regras de fluxos do comutador OpenFlow remotamente.

3.5.1 Diferenças Entre os Controladores

As principais diferenças encontradas entre os controladores, que acarretaram em códigos distintos para a geração de regras, estão apresentadas a seguir e são exemplificadas no quadro ao final dessa seção:

1. Os caminhos das URLs são diferentes entre os controladores (linhas 1 e 17).
2. No POX, o tipo de transação é definido como parâmetro da *string* de dados (linha 2). No Ryu, a transação é definida na URL da requisição HTTP (linha 17).
3. O POX utiliza o DPID do comutador OpenFlow no formato hexadecimal (linha 3), enquanto o Ryu utiliza no formato decimal (linha 18).
4. Alguns dos campos das regras de fluxo são diferentes, como no caso do tipo de ação para envio de pacotes é chamado de "OFPAT_OUTPUT" no POX (linha 5) e apenas "OUTPUT" no Ryu (linha 23).
5. O POX faz a instalação de todas as regras em uma só requisição (linhas 2 a 14), enquanto o Ryu necessita de uma requisição para cada regra (linhas 18 a 34).

O Quadro 2 ilustra as diferenças listadas, destacando-se em **negrito** os itens apontados:

Quadro 2 – Diferenças entre requisições para instalação de regras de fluxos entre Ryu e POX

```

1. URL: http://192.168.189.5:8000/OF/
2. {"method":"set_table",
3.   "params":{"dpid": "00-04-df-1a-8c-39",
4.     "flows":[{"match":{"dl_dst":"00:00:00:00:00:01"},
5.       "actions":[{"type":"OFPAT_OUTPUT","port":1}]
6.     },
7.     {"match":{"in_port":1},
8.       "actions":[{"type":"OFPAT_OUTPUT","port":2},
9.         {"type":"OFPAT_OUTPUT","port":3}
10.      ]
11.    }
12.  ]
13. }
14. }
15.
16.
17. URL: http://192.168.189.5:8080/stats/flowentry/add
18. {"dpid":"20922928185",
19.   "priority":"32768",
20.   "match":{"dl_vlan":100,
21.     "dl_dst":"00:00:00:00:00:01"
22.   },
23.   "actions":[{"type":"OUTPUT","port":1}]
24. }
25.
26. {"dpid":"20922928185",
27.   "priority":"32768",
28.   "match":{"dl_vlan":100,
29.     "in_port":1
30.   },
31.   "actions":[{"type":"OUTPUT","port":2},
32.     {"type":"OUTPUT","port":3}
33.   ]
34. }

```

3.6 Considerações Finais do Capítulo

Esse capítulo apresentou a principal contribuição desta dissertação: o mecanismo proposto para suporte a comutadores OpenFlow no Emulab. Foram detalhados os requisitos funcionais do módulo OpenFlow, bem como, foi apresentado o funcionamento do Emulab com a adição do mecanismo proposto e o processo de definição das operações OpenFlow que pudessem

oferecer as funcionalidades desejadas. Por último, foram detalhados o funcionamento do protótipo implementado e as principais diferenças entre os comutadores OpenFlow utilizados.

4 AVALIAÇÃO DO PROTÓTIPO

Para validar o módulo OpenFlow e efetuar uma análise sobre seu desempenho, foi efetuada uma série de testes com o objetivo de verificar seu funcionamento e compará-lo com os módulos já existentes no Emulab. Os testes foram divididos em duas categorias para melhor definição de seus objetivos e dos mecanismos analisados:

- **Testes de funcionalidades:** Esta categoria de testes tem o objetivo de validar se a execução do módulo proposto efetua a configuração adequada dos comutadores OpenFlow. Para isto, verifica-se a conectividade entre todos os nós de um determinado experimento. O isolamento é validado por meio da execução de experimentos simultâneos e do monitoramento do tráfego que chega em cada nó. Além disso, será verificada a interoperabilidade e, ao mesmo tempo, o isolamento entre um comutador OpenFlow e um comutador convencional, por meio de experimentos que utilizem nós em ambos os tipos de comutadores simultaneamente.
- **Testes de desempenho:** Esta categoria de testes objetiva avaliar se a solução proposta baseada no uso de comutadores OpenFlow tem desempenho semelhante à solução já existente. Para tanto, foram efetuadas as medições de desempenho do comutador OpenFlow durante a execução de um experimento relativas aos itens:
 - **Taxa média de transmissão:** Foi utilizada para se obter uma visão global do desempenho do módulo OpenFlow.
 - **Tempo médio de resposta do primeiro pacote:** Foi obtido o tempo de resposta do primeiro pacote, que se diferencia dos demais pela necessidade de execução do protocolo ARP (*Address Resolution Protocol*) para resolução do IP.
 - **Tempo médio de resposta dos demais pacotes:** Foi medido com o objetivo de avaliar se a solução proposta está efetuando

de forma eficiente a utilização de fluxos de difusão e encaminhamento no comutador OpenFlow e se o desempenho desta abordagem é comparável com comutadores convencionais.

4.1 Recursos Utilizados

Para a elaboração e a execução dos testes foram utilizados os seguintes ambientes e equipamentos disponíveis para este projeto de pesquisa:

- **Ambiente Emulab da USP:** Foi utilizado para efetuar a instalação do protótipo e a execução dos testes mencionados anteriormente. Este ambiente é constituído de 1 Servidor Central, 1 Servidor de Usuários, 24 Nós locais, 2 Comutadores da Rede de Controle, e 3 Comutadores da Rede de Experimentação.
- **Comutadores convencionais da rede de experimentação:** O ambiente Emulab da USP possui 3 comutadores do modelo Cisco Catalyst 3750 na sua rede de experimentação, que foram usados para efetuar testes de interoperabilidade com o comutador OpenFlow;
- **Nós físicos do Emulab:** Foram utilizados 5 dos 24 nós do ambiente Emulab da USP para constituir as topologias necessárias para os testes de funcionalidades e desempenho;
- **Comutadores OpenFlow:** Foram utilizados, também, 5 comutadores OpenFlow de 4 fabricantes diferentes, o que possibilitou validar a compatibilidade do protótipo desenvolvido com diferentes comutadores e implementações do protocolo OpenFlow. Tais comutadores OpenFlow foram dos modelos Datacom 4001, Datacom 4100, Netgear GSM7328SO, além de um comutador Pica8/Pronto 3295 e uma placa NetFPGA programada com o firmware OpenFlow (NAOUS, 2008).

Todos esses comutadores implementam a versão 1.0 do protocolo OpenFlow.

4.2 Execução dos Testes

Para realizar os testes de funcionalidades e desempenho foi definida uma topologia que possibilite validar todos os itens desejados. Para tanto, foram criados 2 experimentos, sendo o primeiro utilizado para os testes de conectividade e desempenho; e o segundo para validar o isolamento do tráfego entre os dois experimentos. A topologia dos experimentos e os respectivos recursos físicos do Emulab estão apresentados na Figura 7.

Figura 7 - Topologia para execução dos testes

A Figura 7 representa o mesmo cenário utilizado para explicar as regras de fluxos definidas para o módulo OpenFlow na Seção 3.4, que já havia sido assim projetada para contemplar todos os cenários de conectividade, o isolamento entre experimentos e os testes de desempenho. O comutador OpenFlow foi um dos modelos mencionados anteriormente (modelo Datacom 4001), que foi integrado ao ambiente Emulab, enquanto o comutador convencional foi um Cisco Catalyst 3750 da rede de

experimentação do Emulab. Além disso, foram utilizados 5 dos 24 nós físicos disponíveis no processo de mapeamento dos recursos.

Para o teste de funcionalidades (conectividade, isolamento e interoperabilidade), foi executada uma requisição ICMP (*Internet Control Message Protocol*) entre os nós 1 e 3 do experimento 1; os demais nós foram utilizados para monitorar o tráfego e identificar o recebimento de pacotes indevidos.

No teste de desempenho, foram empregados apenas os nós 1 e 2 para efetuar as devidas medições no comutador OpenFlow. As transmissões e medições realizadas estão mais detalhadas nas subseções dos respectivos testes.

4.2.1 Testes de Funcionalidades

Este conjunto de testes foi realizado para verificar o funcionamento do módulo OpenFlow durante a execução dos experimentos. Foram efetuados dois testes: o primeiro para validar a conectividade e o isolamento do tráfego no comutador OpenFlow e a compatibilidade com comutadores convencionais existentes no Emulab; o segundo para validar se as regras do módulo OpenFlow funcionam de forma transparente para comutadores OpenFlow de diferentes marcas e fabricantes:

1. **Conectividade, isolamento e compatibilidade:** Este teste tem por objetivo validar se as regras de fluxos definidas na Seção 3.4 apresentam o comportamento esperado para estabelecer a conectividade entre os nós 1 e 3 do experimento 1 e transmitir os pacotes de ARP *request* que utiliza o mecanismo de difusão e ARP *reply*, ICMP *request* e ICMP *reply* que utilizam o mecanismo de encaminhamento. Para monitorar os pacotes trafegados na rede e detectar se a difusão, o encaminhamento e o isolamento dos pacotes estavam funcionando, conforme o esperado, foi utilizada a ferramenta

de monitoramento Tcpcmdump¹⁴ (ASRODIA e PATEL, 2012). Esta ferramenta possibilita a captura de todos os pacotes que cheguem à interface de rede do nó.

A Tabela 8 lista os cabeçalhos de cada mensagem trocada entre os nós 1 e 3 durante a requisição ICMP, e que foi usada, juntamente com a Tabela 9, para acompanhar o funcionamento das regras de fluxos de difusão e encaminhamento do comutador OpenFlow ao receber essas mensagens.

Tabela 8 – Mensagens trocadas entre os nós 1 e 3 durante a requisição ICMP

Tipo do pacote	Interface	VLAN	MAC origem	MAC destino	IP origem	IP destino
ARP request	1	100	00:....:00:01	FF:....:FF	10.0.0.1	10.0.0.3
ARP reply	3	100	00:....:00:03	00:....:00:01	10.0.0.3	10.0.0.1
ICMP request	1	100	00:....:00:01	00:....:00:03	10.0.0.1	10.0.0.3
ICMP reply	3	100	00:....:00:03	00:....:00:01	10.0.0.3	10.0.0.1

Os campos da Tabela 8 são o tipo do pacote transmitido, a interface pela qual o pacote chega ao comutador OpenFlow e as informações referentes ao cabeçalho do pacote: VLAN; endereços MAC de origem e destino; e endereços IP de origem e destino. A Tabela 9, por sua vez, contém as regras de fluxos instaladas no comutador OpenFlow após a criação dos experimentos 1 e 2 correspondentes à topologia apresentada na Figura 7.

Os campos da Tabela 9 são o número da regra, para facilitar a referência de cada regra no decorrer do texto; o número do experimento que gerou a respectiva regra; a prioridade¹⁵ da regra na tabela do comutador; os campos de cabeçalho que definem a regra de fluxo; e a ação correspondente de cada regra.

¹⁴ Tcpcmdump é um analisador de pacotes em linha de comando para sistema Linux e Unix, que permite capturar todos os pacotes que trafegam por uma determinada interface.

¹⁵ O valor padrão do campo prioridade de uma regra OpenFlow é 32768.

Tabela 9 – Regras de fluxos instaladas no comutador OpenFlow

Regra	Experimento	Prioridade	Campos do cabeçalho	Ação
1	1	32768	vlan_id=100, mac_dst=00:00:00:00:00:01	out_port:1
2	1	32768	vlan_id=100, mac_dst=00:00:00:00:00:02	out_port:2
3	1	32768	vlan_id=100, mac_dst=00:00:00:00:00:03	out_port:3
4	2	32768	vlan_id=200, mac_dst=00:00:00:00:00:04	out_port:4
5	2	32768	vlan_id=200, mac_dst=00:00:00:00:00:05	out_port:3
6	1	32767	vlan_id=100, in_port:1	out_port:2, out_port:3
7	1	32767	vlan_id=100, in_port:2	out_port:1, out_port:3
8	1	32767	vlan_id=100, in_port:3	out_port:1, out_port:2
9	2	32767	vlan_id=200, in_port:3	out_port:4
10	2	32767	vlan_id=200, in_port:4	out_port:3

Para efetuar a requisição ICMP foi utilizado o comando "ping -c1 10.0.0.3" no nó 1; este comando efetua uma requisição ICMP (parâmetro -c1) para o endereço IP do nó 3 (10.0.0.3). Para transmitir o pacote ICMP, o nó precisa traduzir o endereço IP do nó 3 para o seu respectivo endereço MAC. Como esta é a primeira transmissão entre esses nós 1 e 3, o nó 1 não possui essa informação e inicia o protocolo ARP. Com isso, o primeiro pacote transmitido é o ARP *request*, detalhado na primeira linha da Tabela 8¹⁶, que é recebido pelo comutador OpenFlow na porta 1.

Neste momento, o comutador OpenFlow varre sua tabela de fluxos para encontrar se alguma regra corresponde ao pacote recebido e executar a respectiva ação. A ordem real das regras de fluxos de mesma prioridade não é conhecida, mas as regras de prioridade superior (32768) são verificadas antes das regras de prioridade menor (32767). Nenhuma das regras de prioridade 32768 (regras de 1 a 5)

¹⁶ Considerar o campo da tabela "IP org" como o campo de endereço do emissor do cabeçalho ARP e o campo "IP dst" como o campo de endereço do receptor.

corresponde ao pacote recebido, pois o pacote ARP *request* possui o endereço MAC de destino igual ao valor de *broadcast* (FF:FF:FF:FF:FF:FF), que conflita com o endereço de destino das regras (campo *mac_dst*). Em seguida, é verificado o conjunto de regras de prioridade 32767 (de 6 a 10). Neste conjunto de regras, o pacote corresponde à regra 6, pois os dois campos da regra são iguais aos campos VLAN (100) e interface do comutador de origem (1) do pacote. Com isso, é executada a ação correspondente à regra 6 sobre o pacote que envia esse mesmo pacote para as interfaces 2 e 3 (outport:2,outport:3), o que equivale à difusão do pacote aos demais nós na VLAN do experimento 1.

Como o nó 3 não está conectado diretamente à interface 3 do comutador OpenFlow, ele recebe o pacote de ARP *request* após o comutador convencional repassar o pacote. O nó 3 processa o pacote recebido e gera a mensagem ARP *reply* detalhada na linha 2 da Tabela 8. Este pacote, por sua vez, corresponde a um tráfego *unicast*, porque possui endereço MAC de destino definido (00:00:00:00:00:01) correspondente ao nó 1. O pacote ARP *reply* é recebido pelo comutador OpenFlow pela interface 3, ao qual está conectado o comutador convencional, e novamente é varrida a tabela de fluxos. Dessa vez, a regra que corresponde ao pacote está no grupo de prioridade 32768, mais especificamente a regra 1, pois o valor da VLAN do pacote (100) e o endereço MAC de destino (00:00:00:00:00:01) coincidem com os da regra na tabela. A ação executada pela regra 1 é encaminhar o pacote pela interface 1, chegando ao nó de destino (nó 1). Conclui-se, assim, a resolução do endereço IP de destino (nó 3).

Com a resolução do endereço IP concluída, o nó 1 começa enviando um pacote ICMP. Como esse tráfego também é *unicast*, o procedimento seguido é semelhante ao do pacote ARP *reply* descrito anteriormente. O pacote ICMP *request* enviado pelo nó 1 é processado pelo comutador OpenFlow empregando-se a regra 3 (VLAN=100 e

MAC de destino igual ao do nó 3). Um pacote ICMP *reply* é enviado pelo nó 3, como resposta ao ICMP *request* recebido, e é processado pelo comutador OpenFlow empregando-se a regra 1, pelo mesmo motivo descrito no procedimento do pacote ARP *reply* (VLAN=100 e MAC de destino igual ao do nó 1).

Durante a troca de mensagens realizadas, foi possível efetuar a validação de quatro requisitos funcionais do módulo OpenFlow:

- A conectividade do domínio de difusão da VLAN 100 (RF3) foi verificada durante a transmissão do pacote ARP *request*, com a ferramenta tcpdump, pois o pacote foi recebido tanto pelo nó 2 quanto pelo nó 3.
- A validação do mecanismo de encaminhamento (RF4) foi verificado durante a transmissão dos pacotes ARP *reply*, ICMP *request* e ICMP *reply*, já que a ferramenta tcpdump detectou os respectivos pacotes apenas nos nós de destino e não foi recebido nenhum tráfego pelo nó 2.
- O isolamento do tráfego entre os experimentos 1 e 2 (VLAN 100 e 200 respectivamente) (RF5) foi validado por meio das instâncias do tcpdump nos nós 4 e 5, que não receberam nenhum pacote durante a transmissão tanto do tráfego *broadcast* quanto *unicast*.
- A compatibilidade entre o comutador OpenFlow e o convencional por meio da porta de VLAN *trunking* (RF6) foi validada tanto para o mecanismo de difusão quanto de encaminhamento, pois todas as mensagens com o nó 3, como origem ou destino, foram recebidas com sucesso.

2. **Compatibilidade do módulo OpenFlow com comutadores de diferentes fabricantes:** Este teste tem por objetivo validar se as regras geradas pelo módulo proposto são compatíveis com comutadores OpenFlow de diferentes marcas e modelos. Devido ao fato do ambiente Emulab encontrar-se no Datacenter da USP e a

entrada de equipamentos necessitar de autorização de entrada e saída, bem como a necessidade do transporte dos equipamentos entre prédios diferentes, apenas o comutador Datacom DM4001 foi integrado ao ambiente Emulab. No caso dos demais comutadores OpenFlow, o processo utilizado para avaliar a compatibilidade das regras geradas pelo módulo OpenFlow, foi a instalação manual de regras de difusão e encaminhamento em cada comutador utilizando o controlador Ryu e seguindo a mesma sintaxe demonstrada na Seção 3.5.1.

Cada comutador testado foi conectado a 4 nós, respectivamente as interfaces de 1 a 4 do comutador. As interfaces dos nós 1 e 2 foram configuradas na VLAN 100, a interface do nó 4 na VLAN 200 e a interface do nó 3 em ambas as VLANs 100 e 200 para simular a porta de VLAN *trunking* da topologia. Essa configuração permite executar a ferramenta tcpdump em todos os nós e efetuar as requisições ICMP entre os nós 1 e 3 para validar o funcionamento das regras de difusão e de encaminhamento. No Apêndice A, encontram-se as requisições HTTP para instalação das regras de fluxos no comutador OpenFlow relativas aos experimentos da Figura 7 utilizando o controlador Ryu.

Foram testados, utilizando o procedimento descrito, os comutadores Datacom 4100, Netgear GSM7328SO, Pica8/Pronto 3295 e a placa NetFPGA. Nos quatro equipamentos, as regras foram instaladas com sucesso e o funcionamento da difusão e do encaminhamento de pacotes funcionaram conforme os resultados descritos para o comutador Datacom DM4001 integrado ao Emulab e descrito no teste anterior, validando deste forma o requisito funcional 1 (RF1).

4.2.2 Testes de Desempenho

Para possibilitar a comparação de desempenho entre o comutador convencional e o comutador OpenFlow, empregando-se o módulo proposto, foi escolhido o Datacom DM4001 tanto para a medição dos valores do

comutador convencional quanto para a medição do comutador OpenFlow, dado que os comutadores Cisco Catalyst 3750 da rede de experimentação do Emulab já possuem experimentos em execução que vão influenciar nas medições de desempenho. Por ser um comutador OpenFlow híbrido, o comutador DM4001 possibilita desativar completamente o protocolo OpenFlow e trabalhar como um comutador convencional, com isso torna-se possível efetuar todos testes de desempenho utilizando o mesmo equipamento. Após a medição dos itens do módulo OpenFlow, o comutador foi configurado manualmente para o módulo convencional, com a desativação do protocolo OpenFlow, e, em seguida, foram coletadas as medições para o modo convencional.

Os testes de desempenho foram efetuados realizando-se medições entre apenas dois nós em um mesmo comutador, mais especificamente os nós 1 e 2 do cenário apresentado Figura 7.

1. **Taxa de vazão:** Este teste tem por objetivo detectar se a utilização das regras de fluxos definidas impacta de alguma forma a taxa de vazão do comutador. Para tanto, foi utilizada a ferramenta Iperf¹⁷ (TIRUMALA, COTTRELL e DUNIGAN, 2003) para medir a taxa de vazão entre os dois nós conectados ao mesmo comutador convencional e o comutador OpenFlow com o módulo OpenFlow proposto. O Iperf é executado em um dos nós no modo servidor e, assim que o Iperf é executado no outro nó, é iniciada a transmissão dos dados. A Tabela 10 apresenta o resultado das medições realizadas, empregando-se o Iperf.

Tabela 10 – Comparação da taxa de vazão de comutador convencional e OpenFlow

Protocolo	Comutador	Tamanho da Transferência	Taxa de vazão
TCP	Convencional	1.09 GB	936 Mbps
	OpenFlow		936 Mbps
UDP	Convencional	1.09 GB	1.04 Gbps
	OpenFlow		1.04 Gbps

¹⁷ Iperf é uma ferramenta para medição de taxa de vazão ponta-a-ponta entre dois equipamentos, usando transmissões TCP ou UDP.

Pode ser observado que não houve diferença na taxa da vazão entre o modo convencional e o modo OpenFlow do comutador DM4001. Os valores apresentados foram obtidos a partir da execução de dez medições com a ferramenta Iperf para cada teste, sendo que o resultado de cada teste foi constante. No caso do protocolo TCP, a taxa de vazão atingiu 93,6% da taxa nominal de 1 Gbps, enquanto no protocolo UDP foi atingido 104% da taxa nominal. O resultado obtido indica que o uso do comutador OpenFlow com o módulo Openflow proposto no modo de funcionamento proativo, com as regras de fluxos definidas, não acarreta em impacto na vazão do comutador no cenário avaliado.

2. **Tempo médio de resposta do primeiro pacote:** Este teste tem por objetivo avaliar o impacto do uso do protocolo OpenFlow no primeiro pacote da transmissão entre dois nós, utilizando requisições ICMP. O primeiro pacote foi analisado separadamente dos demais, porque na primeira transmissão entre dois nós ocorre a resolução do endereço IP por meio do protocolo ARP. Isso implica que, enquanto nas demais requisições ICMP o tempo de resposta mede apenas a troca de mensagens ICMP *request* e ICMP *reply*, a primeira requisição ICMP possui um tempo de resposta maior, pois inclui, também, o tempo de transmissão da troca de ARP *request* e ARP *reply* além das duas mensagens ICMP.

O procedimento para simular sucessivamente a transmissão do primeiro pacote foi apagar os registros da tabela ARP dos dois nós envolvidos no teste e limpar a tabela de aprendizado de MAC do comutador DM4001. Foram efetuadas 1000 requisições ICMP utilizando este procedimento, o resultado das medições está apresentado na Tabela 11.

Tabela 11 – Tempo de resposta do primeiro pacote para módulo convencional e OpenFlow no modo proativo

Comutador	Tempo de resposta (ms)			Amostra	
	Médio	Mínimo	Máximo	Desvio Padrão (ms)	Tamanho
Convencional	0,276	0,200	0,348	0,0329	1000
OpenFlow	0,279	0,202	0,351	0,0363	1000

Os dados obtidos neste teste apresentam uma pequena diferença entre as medições do comutador convencional e o comutador OpenFlow com o módulo OpenFlow proposto. A média do tempo de resposta para o caso do comutador OpenFlow mostrou-se 1,09% maior que no caso do comutador convencional. Esse resultado permite afirmar que a solução proposta apresenta tempo de resposta semelhante ao do comutador convencional. Vale salientar que o resultado obtido está diretamente ligado à escolha do modo de funcionamento proativo do protocolo OpenFlow para o módulo proposto. Caso fosse utilizado o modo reativo de funcionamento, haveria um maior tempo de resposta do primeiro pacote devido ao processo de instalação das regras de fluxos.

Para quantificar o impacto entre a escolha do modo de funcionamento proativo ou reativo, foram efetuados testes utilizando uma solução do módulo OpenFlow baseada no modo reativo para o mesmo cenário. Os dados obtidos estão apresentados na Tabela 12, considerando-se os valores médios de uma amostra de 1000 requisições.

Tabela 12 – Tempo de resposta do primeiro pacote para módulo Openflow no modo reativo

Comutador	Tempo de resposta (ms)			Amostra	
	Médio	Mínimo	Máximo	Desvio Padrão (ms)	Tamanho
OpenFlow	584,22	119	897	215,84	1000

O tempo de resposta médio obtido mostra-se aproximadamente 2000 vezes superior ao resultado obtido no modo proativo e no comutador convencional (Tabela 11). Isto deve-se, principalmente, ao tempo de comunicação adicional entre o comutador OpenFlow e o controlador para a instalação das regras de fluxos. Além disso, outro fator importante é a variação da carga de processos existentes no controlador que está localizado no servidor central do Emulab,

resultando em grande dispersão dos dados e um alto desvio padrão nesta medição.

Os dados obtidos, com o teste do modo reativo, incentivam a escolha do modo proativo na solução adotada para o módulo OpenFlow e evidenciam a proximidade dos resultados entre o modo proativo e os comutador convencional. Vale lembrar que o resultado obtido refere-se apenas à troca do primeiro pacote entre dois nós. Nas demais trocas, não houve diferença de desempenho entre o modo proativo e reativo, porque as tabelas de fluxo de difusão e encaminhamento já estão preenchidas.

3. **Tempo médio de resposta dos demais pacotes:** Este teste tem por objetivo avaliar o impacto do uso do módulo OpenFlow no tempo de transmissão dos demais pacotes entre dois nós, utilizando requisições ICMP. O teste foi executado de forma semelhante ao teste do tempo médio de resposta do primeiro pacote, porém sem o processo de apagar a tabela ARP dos nós e a tabela de MACs do comutador testado. A Tabela 13 apresenta os resultados obtidos.

Tabela 13 – Comparação do tempo de resposta a partir do segundo pacote

Comutador	Tempo de resposta (ms)			Amostra	
	Médio	Mínimo	Máximo	Desvio Padrão (ms)	Tamanho
Convencional	0,1182	0,057	0,187	0,0255	1000
OpenFlow	0,1185	0,061	0,189	0,0258	1000

As medidas do tempo médio de resposta a partir do segundo pacote, mostraram uma variação ainda menor do que no teste do primeiro pacote para o módulo do OpenFlow operando no modo proativo. O tempo médio usando OpenFlow foi apenas 0,25% maior que no comutador convencional e a diferença entre os tempos de resposta mínimos e máximos ficou entre 0,004 e 0,002 ms. Com isso, pode-se considerar que o comutador OpenFlow com o módulo OpenFlow proposto possui desempenho semelhante ao comutador convencional.

4.3 Validação dos Requisitos

Nesta seção, são revisitados os requisitos especificados no Capítulo 3, discutindo-se de que forma cada requisito definido foi atendido pela solução implementada e pelos testes realizados.

- RF1: **Possibilitar a configuração remota de comutadores OpenFlow de diferentes modelos e fabricantes:** Esse requisito foi validado pela execução do teste 2 da Seção 4.2.1, que testou a instalação de regras de fluxos em diferentes comutadores utilizando o controlador Ryu;
- RF2: **Aplicar configurações específicas no comutador OpenFlow durante a fase de alocação de recursos de um experimento:** O módulo OpenFlow foi concebido e desenvolvido de modo que, toda interação com o comutador OpenFlow para a instalação de fluxos e configuração seja feita durante o processo de mapeamento da topologia de rede do Emulab, como descrito nos itens 3 e 4 da Seção 3.5.
- RF3: **Definir o domínio de *broadcast* entre os nós de um mesmo experimento:** Foram utilizadas regras de fluxo com múltiplas portas de destino para delimitar o domínio de difusão entre os nós de um mesmo experimento, como descrito na Seção 3.4 e verificado no teste 1 da Seção 4.2.1, deste capítulo.
- RF4: **Utilizar um mecanismo de encaminhamento de pacotes entre os nós:** Foram definidas regras de fluxos que encaminham os pacotes com um MAC de destino específico à porta em que esse MAC se encontra conectado, como descrito na Seção 3.4 e verificado no teste 1 da Seção 4.2.1, deste capítulo.
- RF5: **Garantir o isolamento do tráfego de um experimento para outro:** A utilização de rótulos de VLANs em todos os pacotes que chegam ao comutador OpenFlow possibilita segregar o tráfego de cada experimento utilizando as regras de fluxo. Tanto as regras de encaminhamento quando de difusão verificam a VLAN juntamente com outros atributos do pacote antes de transmiti-lo. O teste 1 da Seção 4.2.1, deste capítulo, possibilitou validar esse requisito.

RF6: Interligar o comutador OpenFlow aos comutadores convencionais com VLANs *trunking*: Todo o tráfego dos nós ligados a comutadores OpenFlow utilizam rótulos IEEE 802.1Q. Com isso, todo pacote transmitido por um comutador OpenFlow com destino a um comutador convencional ou não já estará preparado para ser recebido pela porta de VLAN *trunking* desse comutador sem a necessidade de nenhum tratamento adicional. Esse comportamento foi verificado no teste 1 da Seção 4.2.1, deste capítulo.

RF7: Efetuar a configuração da taxa de vazão das interfaces do comutador: Como o OpenFlow 1.0 não possui um mecanismo para efetuar esse tipo de configuração, foi implementada uma solução simplificada para configurar a taxa de vazão das interfaces do comutador utilizando a ferramenta Expect, apenas para possibilitar a validação das demais funcionalidades do protótipo. Por isso, esse requisito pode ser considerado apenas parcialmente atendido. Uma proposta de solução para atender completamente a este requisito baseada na utilização do OpenFlow 1.3 está descrita no item 1 *Configuração da taxa de vazão das interfaces* da Seção 5.1, do próximo capítulo.

4.4 Limitações e Melhorias

Durante a elaboração do módulo OpenFlow, houveram questões que não puderam ser solucionadas utilizando apenas operações do protocolo OpenFlow devido principalmente a limitações da versão 1.0 deste protocolo. Duas questões principais podem ser citadas:

- **Configuração da taxa de vazão das interfaces:**

No OpenFlow 1.0, não foi encontrado nenhum mecanismo para permitir a configuração da taxa de vazão das interfaces do comutador OpenFlow. Por este motivo, foi necessário utilizar uma solução limitada por meio da ferramenta Expect para automatizar essa configuração pelo console do comutador. Como a sintaxe do console de comutadores de diferentes

fabricantes não é padronizada, esta solução apresenta o mesmo problema dos módulos tradicionais do Emulab, fazendo-se necessária a criação de um código específico para cada modelo de comutador.

Para solucionar este problema é possível utilizar uma funcionalidade adicionada ao OpenFlow a partir da versão 1.3, chamada Meters. Com este mecanismo torna-se possível controlar a taxa de vazão máxima das regras de fluxo adicionadas à tabela do comutador. Para cada regra de fluxo é criada uma Meter, que indica a taxa máxima de vazão de cada regra. As Meters são armazenadas em outra tabela, e por isso não são consideradas para efeito de conta do limite de registros da tabela de fluxos. A Tabela 14 apresenta como ficam as regras de encaminhamento do módulo OpenFlow listadas na Tabela 5 da Seção 3.4 com a utilização de Meters.

Tabela 14 – Fluxos de encaminhamento com Meters

Regra	Experimento	Campos do cabeçalho	Ação
1	1	vlan_id=100,mac_dst=00:00:00:00:00:01	meter_id:1,out_port:1
2	1	vlan_id=100,mac_dst=00:00:00:00:00:02	meter_id:2,out_port:2
3	1	vlan_id=100,mac_dst=00:00:00:00:00:03	meter_id:3,out_port:3
4	2	vlan_id=200,mac_dst=00:00:00:00:00:04	meter_id:4,out_port:4
4	2	vlan_id=200,mac_dst=00:00:00:00:00:05	meter_id:4,out_port:3

Basicamente, a única mudança foi a adição do campo meter_id na ação de cada regra. Cada Meter possui um identificador próprio que é mapeado em um registro da tabela de Meters, onde ficam armazenadas as características de cada Meter. A Tabela 15 lista os respectivos Meters para cada regra da Tabela 14.

Tabela 15 – Registros na tabela de Meters

Identificador	Unidade	Tipo	Taxa
1	KBPS	DROP	100.000
2	KBPS	DROP	100.000
3	KBPS	DROP	100.000
4	KBPS	DROP	10.000
5	KBPS	DROP	10.000

Nesta tabela, podem-se observar os seguintes campos: **identificador** corresponde a um Meter; **unidade** representa a unidade de medida do Meter; **tipo** representa como o Meter vai atuar no fluxo; e a **taxa** representa o limiar que ativa o Meter. Por exemplo, o Meter 1 define que quando a taxa de vazão

do fluxo exceder 100.000 Kbps (ou 100 Mbps) serão descartados os pacotes excedentes, controlando-se, assim, a taxa de vazão do respectivo fluxo. Um meter pode ser utilizado por mais de uma regra, mas nesse caso o limiar será compartilhado entre as taxas de vazão de fluxo, ou seja, para um Meter de 100 Mbps atribuído a dois fluxos distintos, quando a soma dos dois fluxos exceder 100 Mbps o Meter é ativado.

As regras de 1 a 3 da Tabela 14 possuem um controle da taxa de vazão de 100 Mbps cada, enquanto as regras 4 e 5 possuem o limiar de 10 Mbps. O mecanismo de Meter possibilita solucionar o caso de configuração da taxa de vazão das interfaces do comutador OpenFlow de forma padronizada e torna-se interessante a partir do momento em que sejam lançados mais modelos de comutadores compatíveis com a versão 1.3 do protocolo OpenFlow.

- **Vulnerabilidade de isolamento e escalabilidade:**

Nas regras definidas para encaminhamento não é feita a validação da interface de entrada, apenas da VLAN e do MAC de destino. Isso possibilita que um usuário mal intencionado possa injetar tráfego na VLAN de outro experimento se ele tiver acesso a qualquer nó do Emulab e souber a VLAN e o endereço MAC de um nó de outro experimento. Com essas informações, o usuário pode configurar a interface de seu nó na VLAN do outro experimento e inserir, manualmente, o endereço MAC na tabela ARP do nó. Dessa forma, o usuário mal intencionado gera um pacote com os campos `dl_vlan` e `dl_dst` válidos de outro experimento, que será encaminhado para o respectivo nó do endereço MAC utilizado. O nó mal intencionado não receberá o tráfego de resposta, mas mesmo assim ele viola o isolamento do comutador OpenFlow, pois gerou tráfego na VLAN de outro experimento.

Essa limitação não foi corrigida devido às restritas condições necessárias para reproduzi-la e ao aumento da complexidade imposto por sua solução. Da forma como foi implementado o módulo OpenFlow, é utilizada uma quantidade linear de regras de fluxos em função do número de nós do experimento. Se for aplicada a solução para a vulnerabilidade

descrita, em vez de N regras de encaminhamento, seriam necessárias N^2-N regras, onde N é o número de nós do experimento. Isso ocorre, porque todo MAC de destino deve ser validado com cada interface utilizada no experimento, empregando-se o campo interface de origem do pacote nas regras de encaminhamento. Como existem N nós no experimento, e considerando que é necessário validar esse nó com as demais interfaces do comutador OpenFlow ($N-1$ interfaces), seriam necessárias $N*(N-1) = N^2-N$ regras. Por exemplo, a Tabela 16 lista as regras de encaminhamento com a solução da vulnerabilidade de isolamento descrita, necessárias para representar as mesmas regras apresentadas na Tabela 5 da Seção 3.4.

Tabela 16 – Fluxos de encaminhamento com solução da vulnerabilidade de isolamento

Regra	Experimento	Campos do cabeçalho	Ação
1	1	vlan_id=100,mac_dst=00:00:00:00:00:01,in_port=2	out_port:1
2	1	vlan_id=100,mac_dst=00:00:00:00:00:01,in_port=3	out_port:1
3	1	vlan_id=100,mac_dst=00:00:00:00:00:02,in_port=1	out_port:2
4	1	vlan_id=100,mac_dst=00:00:00:00:00:02,in_port=3	out_port:2
5	1	vlan_id=100,mac_dst=00:00:00:00:00:03,in_port=1	out_port:3
6	1	vlan_id=100,mac_dst=00:00:00:00:00:03,in_port=2	out_port:3
7	2	vlan_id=200,mac_dst=00:00:00:00:00:04,in_port=3	out_port:4
8	2	vlan_id=200,mac_dst=00:00:00:00:00:05,in_port=4	out_port:3

Observa-se que o número de regras necessárias para o experimento 1 passou de 3 (N) para 6 (N^2-N). Já, no caso do experimento 2, manteve-se o mesmo valor pois $N = N^2-N$ para $N = 2$. A utilização de uma quantidade quadrática de regras afeta a escalabilidade do módulo OpenFlow, levando em conta que existem ambientes Emulab com mais de 500 nós, como no caso de UTAH.

Com a utilização do OpenFlow 1.1, ou superior, torna-se possível uma nova solução que não gera uma quantidade quadrática de regras de encaminhamento, por meio do recurso de múltiplas tabelas, que não existia na versão 1.0 do protocolo OpenFlow. Com esse recurso, é possível dividir e distribuir partes das regras de fluxos em diferentes tabelas, evitando casos que gerem produto cartesiano das validações, como no caso apresentado na Tabela 16. A Tabela 17 lista as regras de fluxos equivalentes às da Tabela 16, porém utilizando o recurso de múltiplas tabelas.

Tabela 17 – Fluxos de encaminhamento com múltiplas tabelas de fluxo.

Regra	Experimento	Tabela	Campos do cabeçalho	Ação
1	1	0	vlan_id=100,in_port=1	goto_table:1
2	1	0	vlan_id=100,in_port=2	goto_table:1
3	1	0	vlan_id=100,in_port=3	goto_table:1
4	2	0	vlan_id=200,in_port=3	goto_table:1
5	2	0	vlan_id=200,in_port=4	goto_table:1
6	1	1	vlan_id=100,mac_dst=00:00:00:00:00:01	out_port:1
7	1	1	vlan_id=100,mac_dst=00:00:00:00:00:02	out_port:2
8	1	1	vlan_id=100,mac_dst=00:00:00:00:00:03	out_port:3
9	2	1	vlan_id=200,mac_dst=00:00:00:00:00:04	out_port:4
10	2	1	vlan_id=200,mac_dst=00:00:00:00:00:05	out_port:3

Com este recurso, cada regra é validada em duas etapas. Na tabela 0 do comutador é validada a VLAN e a interface de origem; e na tabela 1 é validada a VLAN novamente e o MAC de destino. Por exemplo, um pacote será encaminhado para a interface 1 apenas se ele casar simultaneamente com uma das regras da tabela 0 e com a regra 6 da tabela 1. Com isso, apenas pacotes que estejam na VLAN 100, que chegarem pelas interfaces de 1 a 3 e possuírem, como MAC de destino 00:00:00:00:00:01, serão encaminhados à interface 1.

Esta solução apresenta-se como alternativa à solução apresentada na Tabela 16, com a vantagem de necessitar um número menor de regras de fluxos. A solução, utilizando múltiplas tabelas, necessita apenas de $N+N$ regras, sendo N o número de regras da tabela 0, correspondendo a cada interface de origem dos nós e sendo o segundo N o número de regras da Tabela 1 correspondendo a cada endereço MAC de destino dos nós. A interface 3 aparece duas vezes na tabela 0, porque tanto o nó 3 quanto o nó 5 estão localizados nessa interface, porém em VLANs distintas 100 e 200 respectivamente, como pode ser verificado na Figura 6.

A desvantagem desta solução dá-se pelo fato de que apenas a versão 1.0 do protocolo OpenFlow é implementada em equipamentos de diversos fabricantes, por isso este mecanismo não foi incorporado à proposta do módulo OpenFlow. Assim como o recurso de controle da taxa de vazão das interfaces, a utilização de múltiplas tabelas se torna interessante a partir do

momento em que mais modelos de comutadores sejam lançados com suporte ao OpenFlow 1.3.

4.5 Considerações Finais do Capítulo

Esse capítulo apresentou a avaliação do protótipo do módulo OpenFlow proposto que foi implementado utilizando-se o comutador DM4001 e o ambiente Emulab da USP. Os resultados do teste de funcionalidades possibilitaram validar que o comutador OpenFlow com o módulo OpenFlow proposto estabelece a conectividade e mantém o isolamento entre os experimentos, apresentando comportamento equivalente à utilização de comutadores convencionais.

A partir dos resultados obtidos nos testes de desempenho pode-se concluir que esse modulo não impacta, de forma significativa, na eficiência da rede de experimentação e que usado com um comutador OpenFlow se mostra uma alternativa viável aos comutadores convencionais.

5 CONSIDERAÇÕES FINAIS

Este trabalho apresenta a proposta de um mecanismo para possibilitar a utilização de comutadores OpenFlow na infraestrutura do ambiente de experimentação Emulab. Por meio do estudo efetuado sobre o funcionamento do ambiente Emulab e do protocolo OpenFlow, foi possível especificar e desenvolver um módulo OpenFlow para implementar este mecanismo, bem como definir um plano de testes para avaliar seu comportamento.

5.1 Avaliação do Mecanismo Proposto

O objetivo definido nesta dissertação foi propor e avaliar um mecanismo que possibilite a utilização de comutadores OpenFlow no ambiente de experimentação Emulab. Para atingir este objetivo foi necessário dividi-lo em três objetivos secundários. A seguir estão descritos os objetivos secundários e como eles foram atingidos:

1. Propor um novo módulo para o Emulab, que efetue a alocação e configuração padronizada de comutadores OpenFlow para diferentes modelos e fabricantes:
 - Com o estudo da arquitetura e o funcionamento do processo de configuração de rede do Emulab, foi possível identificar características desse mecanismo que restringiam os modelos de comutadores suportados pelo ambiente Emulab e tornavam mais complexo o processo de adicionar novos modelos de comutadores ao Emulab. Isto se dá ao fato do processo de configuração de rede depender do protocolo SNMP e necessitar da utilização de MIBs proprietárias para a criação de VLANs para os experimentos, o detalhamento desta questão se encontra na Seção 2.1. Com esta restrição em vista, foi proposto o uso do protocolo OpenFlow como interface padronizada para a criação de um novo mecanismo de

configuração de rede com suporte a comutadores de diferentes modelos de fabricantes variados. Isto foi possível a partir do estudo do funcionamento básico do protocolo OpenFlow e da arquitetura SDN (Seção 2.2) e da identificação de um conjunto de operações do protocolo OpenFlow que fossem equivalentes às funcionalidades utilizadas no processo de configuração de rede do Emulab (Seção 3.3). Estas atividades possibilitaram a criação de um módulo de suporte OpenFlow para o Emulab, que possibilitou atingir o primeiro objetivo proposto;

2. Implementar as funcionalidades no controlador OpenFlow necessárias para estabelecer a conectividade e o isolamento dos nós, de forma compatível com os comutadores convencionais já existentes no Emulab:

- Para possibilitar que comutadores OpenFlow funcionem de forma análoga e interoperável aos comutadores convencionais do Emulab, foram escolhidos: o modo de operação proativo do OpenFlow e a utilização do controlador OpenFlow Ryu. Além disso, foram definidos dois tipos de regras de fluxos (de difusão e de encaminhamento) geradas baseadas na topologia do experimento para configurar os comutadores OpenFlow. O modo proativo foi escolhido para possibilitar a configuração dos comutadores OpenFlow durante a criação do experimento do Emulab. Com esta estratégia, o uso do OpenFlow não gera tempos adicionais de configuração durante a operação da rede (Seção 3.2). Em seguida, foram considerados dois controladores para fazer parte do módulo de suporte OpenFlow proposto, o POX e o Ryu, porém apenas o controlador Ryu atendeu a todas as funcionalidades desejadas. O controlador POX não permitiu a remoção de regras de fluxos do comutador OpenFlow remotamente (Seção 3.5). Finalmente, foi necessário definir as regras de fluxos que seriam instaladas nos comutadores OpenFlow para garantir a conectividade e o isolamento dos nós. Para isso, foram definidos dois tipos de regras, o primeiro se encarregou de estabelecer o domínio de

difusão entre nós em uma mesma VLAN e o segundo foi responsável pelo encaminhamento de pacotes a partir do mapeamento do endereço MAC dos nós e a respectiva interface no comutador a que o nó está conectado. A utilização dos três itens descritos: modo proativo, comutador OpenFlow e regras de fluxos definidas possibilitou atingir o segundo objetivo proposto.

3. Efetuar medições que permitam avaliar as funcionalidades e o desempenho do módulo de suporte OpenFlow por meio de um protótipo e compará-lo com os mecanismos atuais. Este objetivo foi atingido por meio dos testes efetuados e detalhados no Capítulo 4. Os testes foram divididos em duas fases:

- A primeira para avaliar as funcionalidades do módulo de suporte OpenFlow, utilizando dois experimentos projetados para possibilitar a validação da conectividade entre nós em um mesmo experimento; o isolamento entre nós de diferentes experimentos e a interoperabilidade entre comutadores OpenFlow e comutadores convencionais do Emulab. Os resultados dos testes indicaram que o módulo de suporte OpenFlow implementado atendeu a todas as funcionalidades especificadas e com isso comprovou que esta solução é uma alternativa viável ao módulo de configuração de rede original do Emulab.
- A segunda fase foi responsável por avaliar o desempenho de comutadores OpenFlow configurados com o módulo de suporte OpenFlow, comparando com o desempenho de comutadores convencionais. Os resultados obtidos mostraram uma taxa de vazão constante entre os dois modos de operação do comutador (OpenFlow e convencional) e os testes de tempo de resposta apontaram que o modo OpenFlow apresenta tempo de resposta 1,09% maior que no teste do modo convencional. Isto significa que o comutador no modo OpenFlow, utilizando as regras geradas pelo módulo de suporte OpenFlow, apresentou

desempenho muito próximo ao comutador em modo convencional, o que indica que os comutadores OpenFlow são uma alternativa viável como equipamentos para a infraestrutura de ambientes Emulab. Os resultados das duas fases possibilitaram avaliar as funcionalidades e o desempenho do módulo de suporte OpenFlow e compará-lo com os mecanismos atuais e com isso, atingir o terceiro objetivo proposto.

Como resultados secundários obtidos com a pesquisa realizada, tem-se:

1. Pesquisa e estudo aprofundado sobre o funcionamento dos mecanismos de configuração de rede do Emulab;
2. Levantamento sobre o suporte, por parte dos fabricantes, de comutadores compatíveis com o Emulab, obtendo-se um panorama geral sobre quais modelos ainda são vendidos e suportados pelos fabricantes;
3. Estudo aprofundado sobre a especificação do protocolo OpenFlow versão 1.0;
4. Levantamento e estudo sobre controladores OpenFlow disponíveis;
5. Estudo aprofundado sobre as APIs (*Application Programming Interface*) dos controladores OpenFlow.

5.2 Contribuições

A principal contribuição desta pesquisa é a proposta do módulo para configuração de comutadores utilizando OpenFlow no ambiente de experimentação Emulab. Este mecanismo apresenta comportamento equivalente ao mecanismo atual do Emulab baseado em SNMP, porém utiliza uma abordagem inovadora para suportar modelos de comutadores baseados em OpenFlow de diferentes fabricantes, bem como reduzir o tempo para a adição de novos modelos. O código desenvolvido será cedido ao projeto do Emulab.

Como consequência do estudo realizado neste trabalho, foram levantadas informações sobre os controladores OpenFlow disponíveis na literatura, explicitando características de sua implementação como linguagem de codificação, bem como informações sobre a atividade atual¹⁸ dos projetos.

Outra contribuição desta dissertação, é a possibilidade de efetuar a programação de novos serviços de rede nos equipamentos OpenFlow em um ambiente Emulab. O mecanismo proposto neste trabalho implementa a funcionalidade de encaminhamento nos comutadores OpenFlow operando num ambiente Emulab. Esta funcionalidade é implementada na forma de um componente, que pode ser substituído por novos componentes que ofereçam novas funcionalidades para a rede, como por exemplo, roteamento e simulação de atraso.

5.3 Publicações

Como principais publicações podem-se citar:

- Artigo (*Short-Paper*) aceito e publicado com apresentação efetuada no formato de pôster. Esse artigo traz a proposta inicial do mecanismo de suporte a comutadores OpenFlow do Emulab, apresentando-se a arquitetura e definição de regras de fluxo propostas nesta dissertação:

Marcos F. Schwarz, Marco A. T. Rojas, Charles C. Miers, Fernando F. Redígolo, Tereza C. M. B. Carvalho; *Emulated and Software Defined Networking Convergence. 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM2013)*, pags 700-703. Ghent/Belgium. 2013.

- Tutorial relacionado à pesquisa desta dissertação, que foi apresentado pelo seu autor:

¹⁸ Informações relativas a 25/06/2013

Marcos F. Schwarz, Rafael V. S. Penteado, Marco T. Rojas, Fernando F. Redígolo, Tereza C. M. B. Carvalho; Experimentação em SDN/OpenFlow Utilizando o Testbed Emulab/ProtoGENI. XXXIX Conferencia Latinoamericana de Informática (CLEI). Venezuela. Outubro 2013.

5.4 Trabalhos Futuros

O protótipo do módulo de suporte a OpenFlow, desenvolvido neste trabalho, terá seu código disponibilizado como contribuição para o projeto Emulab. Desta forma, esta solução poderá ser testada com maior variabilidade de cenários, para ser aprimorada e, eventualmente, fazer parte do repositório oficial do projeto e contribuir na criação de novos ambientes Emulab e na expansão de ambientes já existentes. O código está em processo de atualização para ser integrado à versão mais nova do Emulab e ser submetido em seguida.

Para dar continuidade a esta pesquisa, torna-se interessante explorar algumas extensões do trabalho realizado, como por exemplo, oferecer OpenFlow aos experimentadores. Para isso, é possível utilizar a ferramenta FlowVisor (citada na Seção 2.3) para dar acesso aos experimentadores, de forma controlada, para configurarem os comutadores OpenFlow durante seus experimentos e possibilitar ao Emulab criar redes experimentais OpenFlow. Algumas limitações dessa proposta são: a restrição do FlowVisor de suportar apenas a versão 1.0 do protocolo OpenFlow; necessidade de limitar o número de regras de fluxos instaladas por um experimentador para evitar o mau uso dos comutadores e impedir que a tabela seja preenchida completamente, o que pode interferir na criação de novos experimentos pelo módulo OpenFlow de configuração de rede.

Outra proposta de trabalho futuro é realizar testes de compatibilidade e desempenho do módulo OpenFlow com comutadores que suportam a versão 1.3, para dessa forma, validar todos os recursos utilizados pelo módulo, como Meters e múltiplas tabelas. É possível, também, refazer os testes de

desempenho para validar o mecanismo de controle de taxa de vazão (Meters) e validar se o uso de múltiplas tabelas afetam os resultados obtidos com o módulo baseado em OpenFlow 1.0.

REFERÊNCIAS BIBLIOGRÁFICAS

- ABELÉM, A., MACHADO, I., STANTON, M., and CARVALHO, T. "Design of a testbed for R&D in network architectures". First Workshop of the Brazilian Institute for Web Science Research. Rio de Janeiro, 2010.
- ASRODIA, P.; PATEL, H. Analysis of Various Packet Sniffing Tools for Network Monitoring and Analysis. International Journal of Electrical, Electronics and Computer Engineering, v. 1, n. 1, p. 55-58, 2012.
- CHUN, B. et al. Planetlab: an overlay testbed for broad-coverage services. ACM SIGCOMM Computer Communication Review, v. 33, n. 3, p. 3-12, 2003.
- CROCKFORD, D. RFC 4627—The application/json Media Type for JavaScript Object Notation. JSON, IETF, Request for Comments, 2006.
- DATACOM. Disponível em: <<http://www.datacom.ind.br>> Acesso em: 22 jun. 2014.
- EK, Niclas. IEEE 802.1 P, Q-QoS on the MAC level. Apr, v. 24, p. 0003-0006, 1999.
- ELLIOTT, C. GENI-global environment for network innovations. In: LCN. p. 8. 2008.
- EMULAB WIKI. Emulab Documentation Wiki. Disponível em: <<https://users.emulab.net/trac/emulab/wiki>>. Acesso em: 22 jun. 2014.
- FERNANDEZ, M. Evaluating OpenFlow Controller Paradigms. In: ICN 2013, The Twelfth International Conference on Networks. 2013. p. 151-157.
- FARIAS, F. et al. LegacyFlow: Bringing OpenFlow to Legacy Network Environments. 2011.
- FDIDA, S.; FRIEDMAN, T.; PARMENTELAT, T. OneLab: An open federated facility for experimentally driven future internet research. In: New Network Architectures. Springer Berlin Heidelberg, p. 141-152. 2010.
- FEDERICA. Federated E-infrastructure Dedicated to European Researchers Innovating in Computing network Architectures. Disponível em: <<http://www.fp7-federica.eu>>. Acesso em: 22 jun. 2014.
- FIELDING, R.; TAYLOR, R. Principled design of the modern Web architecture. ACM Transactions on Internet Technology (TOIT), v. 2, n. 2, p. 115-150, 2002
- FLUX GROUP. The Flux Research Group. University of Utah. Disponível em: <<http://www.cs.utah.edu/flux/index.html>>. Acesso em 22 jun. 2014.

GENI. Global Environment for Network Innovations. Disponível em: <<http://www.geni.net/>>. Acesso em 22 jun. 2014.

GUDE, N. et al. NOX: towards an operating system for networks. ACM SIGCOMM Computer Communication Review, v. 38, n. 3, p. 105-110, 2008.

HIBLER, M. et al. Large-scale Virtualization in the Emulab Network Testbed. In: USENIX Annual Technical Conference. p. 113-128. 2008.

HIRABARU, M. et al. October 2008. New Generation Network Architecture AKARI Conceptual Design (ver1. 1). AKARI Architecture Design Project.

LANTZ, B.; HELLER, B.; MCKEOWN, N. A network in a laptop: rapid prototyping for software-defined networks. In: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks. ACM, 2010. p. 19.

LIBES, D. Exploring Expect: a Tcl-based toolkit for automating interactive programs. " O'Reilly Media, Inc.", 1995.

MCCANNE, S.; FLOYD, S. The network simulator, ns-2. 1997. <<http://www.isi.edu/nsnam/ns>> . Acesso em 22 jun. 2014.

MCKEOWN, N. et al. OpenFlow: enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, v. 38, n. 2, p. 69-74, 2008.

NAOUS, J. et al. Implementing an OpenFlow switch on the NetFPGA platform. In: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. ACM, 2008. p. 1-9

OFTEST. Validating OpenFlow Switches. Disponível em: <<http://www.projectfloodlight.org/oftest/>> Acesso em: 22 jun. 2014.

ONF - Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. ONF White Paper. Palo Alto, US: Open Networking Foundation, 2012.

OPENFLOW SWITCH CONSORTIUM et al. OpenFlow Switch Specification Version 1.0.0. 2009.

PFAFF, Ben et al. Extending Networking into the Virtualization Layer. In: Hotnets. 2009.

PETERSON, L., et al. "A Blueprint for Introducing Disruptive Technology into the Internet", In: Proceedings of ACM HotNets-I Workshop, Princeton, New Jersey, USA, October 2002.

POX. About POX. Disponível em: <<http://www.noxrepo.org/pox/about-pox/>> Acesso em: 22 jun. 2014.

PROJETO ANSP (an Academic Network at São Paulo). Disponível em: <<http://www.ansp.br/index.php/>>. Acesso em: 22 jun. 2014.

RAKOTOARIVELO, T. et al. "OMF: a control and management framework for networking testbeds". ACM SIGOPS Operating Systems Review, v. 43, n. 4, p. 54-59, 2010.

RYU. Ryu SDN Framework. Disponível em: <<http://osrg.github.io/ryu/>> Acesso em: 22 jun. 2014.

SALLENT, S. et al. FIBRE project: Brazil and Europe unite forces and testbeds for the Internet of the future. In: Testbeds and Research Infrastructure. Development of Networks and Communities. Springer Berlin Heidelberg, 2012. p. 372-372.

SCHWARZ, M. et. Al. (2013) "Emulated and Software Defined Networking Convergence". IFIP/IEEE International Symposium on Integrated Network Management, Ghent-Belgium. 2013

SKLOWER, K. Emulab Administrators Mailing list - Answer to topic: Detail requirements of switches for testbed. Julho de 2008. Disponível em: <<https://groups.google.com/d/msg/emulab-admins/dl9oek4dXs0/6rQC1A5NhmlJ>>. Acesso em: 22 jun. 2014.

SHERWOOD, R. et al. Flowvisor: A network virtualization layer. OpenFlow Switch Consortium, Tech. Rep, 2009.

SRISURESH, Pyda; EGEVANG, Kjeld. Traditional IP network address translator (Traditional NAT). 2001.

STALLINGS, W. SNMP, SNMPv2, SNMPv3, and RMON 1 and 2. Addison-Wesley Longman Publishing Co., Inc., 1998.

SUÑÉ, M. et al. "Design and implementation of the OFELIA FP7 facility: The European OpenFlow testbed". Computer Networks, 2013.

SZEGEDI, P. et al. With evolution for revolution: Managing federica for future internet research. Communications Magazine, IEEE, v. 47, n. 7, p. 34-39, 2009.

TIRUMALA, A.; COTTRELL, L.; DUNIGAN, T. Measuring end-to-end bandwidth with Iperf using Web100. In: Web100", Proc. of Passive and Active Measurement Workshop. 2003.

WAHLE, Sebastian et al. Emerging testing trends and the Panlab enabling infrastructure. Communications Magazine, IEEE, v. 49, n. 3, p. 167-175, 2011.

WHITE, B. et al. An integrated experimental environment for distributed systems and networks. ACM SIGOPS Operating Systems Review, v. 36, n. SI, p. 255-270, 2002.

APÊNDICE A

No Quadro 3 estão listadas as requisições HTTP para instalação das regras de fluxos relativas aos experimentos da Figura 7 utilizando o controlador Ryu. Cada requisição HTTP é responsável por instalar uma regra de fluxo no comutador, as dez regras apresentadas a seguir correspondem respectivamente as cinco regras de difusão apresentadas e detalhadas na Tabela 4 da Seção 3.4 e as cinco regras seguintes às regras de encaminhamento da Tabela 5 da mesma Seção. Essas regras são responsáveis por prover a conectividade e isolamento entre os experimentos.

Quadro 3 – Requisições HTTP para instalação de regras de fluxo para controlador Ryu

```

1. URL: http://192.168.189.5:8080/stats/flowentry/add
2.
3. {"dpid":"20922928185",
4.  "priority":"32767",
5.  "match":{"dl_vlan":100,
6.          "in_port":1
7.          },
8.  "actions":[{"type":"OUTPUT","port":2},
9.            {"type":"OUTPUT","port":3}
10.           ]
11. }
12.
13.
14. {"dpid":"20922928185",
15.  "priority":"32767",
16.  "match":{"dl_vlan":100,
17.          "in_port":2
18.          },
19.  "actions":[{"type":"OUTPUT","port":1},
20.            {"type":"OUTPUT","port":3}
21.           ]
22. }
23.
24.
25. {"dpid":"20922928185",
26.  "priority":"32767",
27.  "match":{"dl_vlan":100,
28.          "in_port":3
29.          },
30.  "actions":[{"type":"OUTPUT","port":1},
31.            {"type":"OUTPUT","port":2}

```

```

32.         ]
33. }
34.
35.
36. {"dpid":"20922928185",
37.  "priority":"32767",
38.  "match":{"dl_vlan":200,
39.          "in_port":4
40.          },
41.  "actions":[{"type":"OUTPUT","port":3}]
42. }
43.
44. {"dpid":"20922928185",
45.  "priority":"32767",
46.  "match":{"dl_vlan":200,
47.          "in_port":5
48.          },
49.  "actions":[{"type":"OUTPUT","port":4}]
50. }
51.
52. {"dpid":"20922928185",
53.  "priority":"32768",
54.  "match":{"dl_vlan":100,
55.          "dl_dst":"00:00:00:00:00:01"
56.          },
57.  "actions":[{"type":"OUTPUT","port":1}]
58. }
59.
60. {"dpid":"20922928185",
61.  "priority":"32768",
62.  "match":{"dl_vlan":100,
63.          "dl_dst":"00:00:00:00:00:02"
64.          },
65.  "actions":[{"type":"OUTPUT","port":2}]
66. }
67.
68. {"dpid":"20922928185",
69.  "priority":"32768",
70.  "match":{"dl_vlan":100,
71.          "dl_dst":"00:00:00:00:00:03"
72.          },
73.  "actions":[{"type":"OUTPUT","port":3}]
74. }
75.
76. {"dpid":"20922928185",
77.  "priority":"32768",
78.  "match":{"dl_vlan":200,
79.          "dl_dst":"00:00:00:00:00:04"

```



```
80.         },
81.   "actions":[{"type":"OUTPUT","port":4}]
82. }
83.
84. {"dpid":"20922928185",
85.   "priority":"32768",
86.   "match":{"dl_vlan":200,
87.           "dl_dst":"00:00:00:00:00:05"
88.         },
89.   "actions":[{"type":"OUTPUT","port":3}]
90. }
```