MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÃO E COMUNICAÇÕES

CENTRO BRASILEIRO DE PESQUISAS FÍSICAS

Coordenação de Formação Científica

Pedro Henrique Diniz da Silva

VCFLOW: SOLUÇÃO PARA PROVISIONAMENTO DE CIRCUITOS VIRTUAIS DINÂMICOS DE CAMADA 2 EM REDES SDN/OPENFLOW HÍBRIDAS

Rio de Janeiro - RJ

2017

Pedro Henrique Diniz da Silva

VCFLOW: SOLUÇÃO PARA PROVISIONAMENTO DE CIRCUITOS VIRTUAIS DINÂMICOS DE CAMADA 2 EM REDES SDN/OPENFLOW HÍBRIDAS

Dissertação apresentada à Coordenação de Formação Científica do Centro Brasileiro de Pesquisas Físicas, como requisito para obtenção do título de Mestre em Física com ênfase em Instrumentação Científica.

Orientador(es): Prof. *D. Sc.* Márcio Portes de Albuquerque (CBPF - RedeRio/FAPERJ)

Prof. *D. Sc.* Nilton Alves Jr. (CBPF - RedeRio/FAPERJ)

Rio de Janeiro - RJ

2017

Centro Brasileiro de Pesquisas Físicas
Rua Doutor Xavier Sigaud, 150, Rio de Janeiro, Brasil
Tel.: +55 21 2141-7100 Fax.: +55 21 2141-7400 - CEP:22290-180
http://www.cbpf.br
CBPF

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA,
INOVAÇÕES E COMUNICAÇÕES

"VCFLOW: SOLUÇÃO PARA PROVISIONAMENTO DE CIRCUITOS VIRTUAIS DINÂMICOS DE CAMADA 2 EM REDES SDN/OPENFLOW HÍBRIDAS"

PEDRO HENRIQUE DINIZ DA SILVA

Dissertação de Mestrado Profissional em Física com ênfase em Instrumentação Científica apresentada no Centro Brasileiro de Pesquisas Físicas do Ministério da Ciência, Tecnologia, Inovação e Comunicação. Fazendo parte da banca examinadora os seguintes professores:

Márcio Portes de Albuquerque - Presidente/Orientador/CBPF

Nilton Alves Júnior – Coorientador/CBPF

José Ferreira de Rezende – Titular/UFRJ

Ulisses Barres de Almeida – Titular/CBPF

Natália Castro Fernandes – Suplente externo/UFF

Marcelo Portes de Albuquerque – Suplente/CBPF

Alexandre Leib Grojsgold – Convidado especial/LNCC

Luís Felipe Magalhães de Moraes – Convidado especial/UFRJ

Rio de Janeiro, 23 de junho de 2017.

AGRADECIMENTOS

Primeiramente, a Deus, senhor de todo o conhecimento e sabedoria. Obrigado, por permitir a obtenção dessa conquista, por estender suas mãos nos momentos de desânimo e por me apresentar a pessoas tão especiais no decorrer da vida.

À minha família, em especial, a duas pessoas que foram as mais especiais nessa longa jornada. Aos meus avós Selma Diniz (*in memorian*) e Hélio Diniz (*in memorian*). Ambos não estão mais aqui para compartilhar a conclusão dessa etapa de vida, mas foram essenciais para que pudesse acontecer. Obrigado por tudo. Espero poder um dia reencontrá-los. Amo vocês.

À pessoa mais especial de todo o mundo, Luana Barbosa. Jamais poderia ter encontrado melhor em lugar algum do mundo. Obrigado, por compartilhar comigo cada momento de vida. Desde o início dessa jornada cada vitória, cada tropeço, cada choro, cada mágoa, cada felicidade e, infelizmente, cada perda. Obrigado, por cada puxão de orelha e por abdicar de tantos momentos de lazer e diversão juntos para que tudo pudesse dar certo. Obrigado por compartilhar um pouquinho de sua vida comigo. Essa vitória é dela também.

À equipe da COTEC (Coordenação de Desenvolvimento Tecnológico) do CBPF/MCTIC (Centro Brasileiro de Pesquisas Físicas/Ministério de Ciência Tecnologia, Inovação e Comunicações), em especial a três pessoas essenciais para a conclusão desse trabalho. Ao Prof. *D. Sc.* Nilton Alves Jr., um professor, um orientador, um poeta e principalmente um amigo. Ao Sandro Luiz Pereira, por ser não só um grande amigo e parceiro, mas uma pessoa que está sempre zelando pelos seus companheiros. Ao Prof. *D. Sc.* Márcio Portes de Albuquerque, por coordenar toda a equipe da CEO (Coordenação de Engenharia de Operações) da RedeRio/FAPERJ e o comitê técnico da Redecomep-Rio (RNP – Rede Nacional de Ensino e Pesquisa) de forma tão sóbria, apesar da situação turbulenta pela qual passamos atualmente. Obrigado pela paciência, por todas as discussões, análises de resultados, revisões e o suporte para que todo esse trabalho pudesse ser concluído. Muito obrigado, por ser mais do que um orientador, por ser um amigo sempre zelando pelo futuro.

Agradeço à COTEC do CBPF/MCTIC, à Coordenação de Engenharia de Operações da RedeRio/FAPERJ (Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro), à Coordenação Técnica do Projeto Redecomep-Rio pelo suporte e acesso aos laboratórios e equipamentos utilizados no desenvolvimento deste trabalho.

À toda equipe do CEO da RedeRio/FAPERJ, aos ainda aqui não citados Marita, Marcelo e Jabur (#ficajabur) pelo apoio no desenvolvimento do projeto.

À equipe da Verto Technologies, em especial ao George Argiros, sem ele o ponta pé inicial para esse trabalho não teria sido possível.

RESUMO

As redes acadêmicas atuais contribuem para o estudo e o desenvolvimento de projetos entre instituições que necessitam da troca de um grande volume de informações por meio de redes de alta velocidade. No entanto, esse tipo de aplicação também demanda serviços de transporte privativo com alta disponibilidade, tendo em vista as características dos dados que gerados por essas aplicações. Esses serviços, denominados de transporte de camada 2 (do modelo de referência TCP/IP), são utilizados para provisionamento de circuitos virtuais privados capazes de isolar o tráfego transportado em relação aos demais. Atualmente, entretanto, esses serviços são implementados normalmente através de soluções proprietárias, distribuídas pelos diversos elementos de rede, inserindo uma maior complexidade na operação da infraestrutura. O novo paradigma de Redes Definidas por *Software* (SDN), no entanto, pode fornecer soluções alternativas para o provimento desses serviços. As SDNs introduzem o conceito de controladores centralizados para gerenciamento do comportamento do encaminhamento dos elementos de rede. Esse conceito permite que diversos serviços em redes de *backbone* sejam simplificados facilitando a operação da rede. Este trabalho propõe uma nova solução denominada *Virtual Circuits Flow* (VCFlow) com a finalidade de provisionar circuitos virtuais L2VPN em redes de *backbone*. A solução VCFlow provisiona serviços L2VPN *Ethernet*/VLAN em redes SDN híbridas, mais especificamente em redes que adotam o protocolo de comunicação OpenFlow. Além disso, o VCFlow tem como suas principais características: (i) suporte a privacidade (por meio do isolamento lógico do tráfego transportado); (ii) baixa complexidade de operação (estabelecimento de circuitos virtuais de modo transparente); e (iii) alto desempenho e disponibilidade (operando sob demanda e de modo resiliente). O VCFlow foi implementado utilizando o arcabouço Ryu como controlador OpenFlow e foi estendido para lidar com: (a) operação de uma topologia SDN/OpenFlow híbrida; (b) cálculo de menores caminhos através do algoritmo SPF; (c) processo de VLAN *stitching* para estabelecimento de circuitos virtuais; e (d) prevenção de *loops* baseado na técnica de *Split Horizon*. A solução proposta foi caracterizada e teve seu desempenho avaliado através de cenários emulados por meio do emulador Mininet e de cenários de estudo de caso em uma rede virtual paralela à rede de produção do *backbone* da RedeRio Metropolitana (Redecomep-Rio). Os resultados mostraram intervalos de tempo médio de estabelecimento de circuitos inferiores a um segundo e de recuperação inferiores a dois segundos. Esses valores atendem a requisitos de alta disponibilidade para várias aplicações. É possível com este estudo estabelecer qual é a resiliência da infraestrutura de rede para atendimento a demandas especificas. Através dos resultados obtidos nas avaliações, é possível concluir que o serviço de provisionamento de circuitos virtuais baseado na solução VCFlow permite a adoção de serviços de transporte de camada 2 com alto desempenho e baixa complexidade em uma topologia de rede SDN/OpenFlow híbrida.

Palavras-chave: VCFlow; Circuitos Virtuais; L2VPN; Redes Definidas por *Software*; OpenFlow; Controlador Ryu.

ABSTRACT

Current academic computer networks contribute to the development of research and education projects among institutions that require the exchange of a large volume of information through high-speed computer networks. However, this type of application also demands private transport services with high availability, taking into consideration the characteristics of the data generated by these applications. These services, called Layer 2 (of the TCP/IP reference model) transport services, are used for the provision of private virtual circuits that are capable of isolating the traffic transported in relation to the others. Currently, though, these services are generally implemented through proprietary solutions, distributed in the various network elements, what inserts a greater complexity in the operation of the network infrastructure. Moreover, the new paradigm of Software Defined Networks (SDN) can provide alternative solutions for the provisioning of these services. SDNs introduce the concept of centralized controllers to manage the forwarding behavior of network elements. This concept allows that multiple services on backbone networks to be simplified by facilitating network operation. We propose a new solution called Virtual Circuits Flow (VCFlow) for the purpose of provisioning L2VPN virtual circuits in backbone networks. The VCFlow solution provides L2VPN Ethernet/VLAN services in hybrid SDN networks, more specifically in networks that adopt the OpenFlow communication protocol. In addition, VCFlow has as its main characteristics: (i) support for privacy (through the logical isolation of carried traffic); (ii) low complexity of operation (transparently establishment of virtual circuits); and (iii) high performance and availability (operating on demand and in resilient mode). VCFlow was implemented using the Ryu framework as OpenFlow controller and it was extended to deal with: (a) operation of a hybrid SDN/OpenFlow topology; (b) calculation of shortest paths through the SPF algorithm; (c) VLAN stitching process for establishing of virtual circuits; and (d) prevention of loops based on the Split Horizon technique. We characterized the proposed solution and evaluated its performance through emulated scenarios using the Mininet emulator and case study scenarios in a virtual network working in parallel to the production network of the backbone of RedeRio Metropolitana (Redecomep-Rio). The results have shown average time intervals of circuit establishment less than one second and of recovery less than two seconds. These values meet high availability requirements for multiple applications. It is possible with this study to define the resilience of the network infrastructure to meet specific demands. Through the results that we have obtained in the evaluations, it is possible to conclude that the virtual circuit provisioning service based on the VCFlow solution allows the adoption of layer 2 transport services with high performance and low complexity in a hybrid SDN/OpenFlow network topology.

Keywords: *VCFlow; Virtual Circuits; L2VPN; Software Defined Network; OpenFlow; Ryu Controller.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS, ACRÔNIMOS E SIGLAS

AL2S - *Internet2's Advanced Layer-2 Service*

API - *Application Programming Interface*

ARP - *Address Resolution Protocol*

BoD - *Bandwidth on Demand*

Caltech - *California Institute of Technology*

CBPF - Centro Brasileiro de Pesquisas Físicas

CDF - *Cumulative Distribution Function*

CE - *Customer Edge*

Cefet-RJ - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

CPU - *Central Processing Unit*

dest. - destino

DPID - *DataPath IDentification*

DWDM - *Dense Wavelength Division Multiplexing*

DYNES - *DYnamic NEtwork System*

DynPaC - *Dynamic Path Computation*

ESnet - *Energy Sciences Network*

Eth. - *Ethernet*

EUA - Estados Unidos da América

FermiLab - *Fermi National Accelerator Laboratory*

Fibre - *Future Internet Brazilian Environment for Experimentation*

Fiocruz - Fundação Oswaldo Cruz

GB - GigaByte

GMPLS - *Generalized Multi Protocol Label Switching*

GRE - *Generic Routing Encapsulation*

GUI - *Graphic User Interface*

HP - *Hewlett-Packard*

IBM - *International Business Machines*

ICMP - *Internet Control Message Protocol*

ID - *Identification*

IDCP - *Inter Domain Controller Protocol*

IEEE - *Institute of Electrical and Electronics Engineers*

IP - *Internet Protocol*

IplanRio - Empresa Municipal de Informática da Cidade do Rio de Janeiro

ISP - *Internet Service Provider*

JVM - *Java Virtual Machine*

L2TP - *Layer 2 Tunneling Protocol*

L2VPN - *Layer 2 Virtual Private Network*

L3VPN - *Layer 3 Virtual Private Network*

LAN - *Local Area Network*

LHC - *Large Hadron Collider*

LLDP - *Link Layer Discovery Protocol*

LTS - *Long Term Support*

MAC - *Media Access Control*

MIX - *Municipal Internet eXchange*

MPLS - *Multi Protocol Label Switching*

NEC - *Nippon Electric Company*

NOS - *Network Operating System*

NSI - *Network Services Interface*

NSI-CS - *Network Services Interface - Connection Service*

ODL - *OpenDayLight*

OESS - *Open Exchange Software Suite*

OF - *OpenFlow*

ONE - *Open Network Environment*

onePK - *Open Network Environment Plataform Kit*

ONF - *Open Networking Foudation*

ONOS - *Open Networking Operating System*

ori. - origem

OSCARS - *On-Demand Secure Circuits and Advanced Reservation System*

OvS - *OpenVSwitch*

P - *Provider*

PBB - *Provider Backbone Bridges*

PBR - *Policy Based Routing*

PCP - *Priority Code Point*

PE - *Provider Edge*

POF - *Protocol-Oblivious Forwarding*

PUC-RJ - Pontifícia Universidade Católica do Rio de Janeiro

PVC - *Permanent Virtual Circuit*

Redecomep - Redes Comunitárias de Educação e Pesquisa

REN - *Research and Education Networks*

RNP - Rede Nacional de Educação e Pesquisa

RPM - Rotações Por Minuto

SAS - *Serial Attached SCSI*

SCSI - *Small Computer System Interface*

SDN - *Software Defined Networks*

SIX - *State Internet eXchange*

SOAR - *Southern Astrophysical Research*

SPF - *Shortest Path First*

STP - *Spanning Tree Protocol*

TCP - *Transport Control Protocol*

UERJ - Universidade do Estado do Rio de Janeiro

UFF - Universidade Federal Fluminense

UFRJ - Universidade Federal do Rio de Janeiro

Unirio - Universidade Federal do Estado do Rio de Janeiro

VFI - *Virtual Forwarding Instance*

VID - *VLAN Identifier*

VLAN - *Virtual Local Area Network*

VM - *Virtual Machine*

VMFS - *Virtual Machine File System*

VRF - *Virtual Routing and Forwarding*

WAN - *Wide Area Network*

WLCG - *Worldwide LHC Computing Grid*

SUMÁRIO

Capítulo 1

1 INTRODUÇÃO

O elo entre a física e a computação tem seu marco nos laboratórios da empresa *Bell Labs*, no final da década de 40, quando do invento do transistor, dando início à era da microeletrônica. Nestas últimas seis décadas, essa união tem gerado avanços significativos para ambas as áreas. Por exemplo, a física utiliza a computação para a realização de simulações numéricas e aquisição de dados complexos, e a computação usa as descobertas científicas da física para construir dispositivos e computadores cada vez mais elaborados, complexos e eficientes. Ademais, inúmeros projetos com aplicações tecnológicas nasceram ou tiveram importantes aplicações em laboratórios de física em todo o mundo. A popular "*Worldwide Web"*, por exemplo, nasceu da necessidade de manipular e divulgar grandes volumes de dados no CERN, o laboratório de física europeu para pesquisas nucleares (Genebra, Suíça). Aplicações em ambientes de computação paralela, na qual várias tarefas são realizadas simultaneamente por um computador, tiveram grande avanço devido às simulações numéricas de colisões de partículas realizadas em laboratórios de pesquisa em física de altas energias, como o Fermilab (Estados Unidos).

Recentemente, diversas pesquisas e desenvolvimento (P&D) de tecnologias para operação de redes a 100 Gbps, (Kissel *et al.*, 2013; Garzoglio *et al.*, 2014; Hoeft e Petzold, 2015) têm sido desenvolvidas no ambiente cooperativo mundial do CERN. O CBPF tem atuado nesta linha de P&D, seja através do desenvolvimento de instrumentos e técnicas avançadas para a comunidade de física em cooperação com o CERN (em projetos como, por exemplo, o LHCONE[1]), na operação da RedeRio de Computadores (a rede avançada para pesquisa, ensino e inovação do governo do Estado do Rio de Janeiro) ou na coordenação técnica da rede de fibras óticas do projeto Redecomep-Rio (em um projeto colaborativo

[1] O LHCONE é um serviço de Rede Privada Virtual de Camada 3 (L3VPN, do inglês *Layer 3 Virtual Private Network*) que prove conectividade entre todos os *sites Tier-1* e *Tier-2* do WLCG. Esse serviço utiliza técnicas de *Virtual Routing and Forwarding* (VRF) e o *Border Gateway Protocol* (BGP) para prover um *backbone* dedicado e reservado somente a análise e transferência de dados voltados à física de altas energias. O CBPF integra desde 2015 esse projeto (RNP, 2015) e serviu como instituição piloto para sua implementação na América Latina.

coordenado pela RNP e pela FAPERJ e com a participação de diversas instituições na região metropolitana do Rio de Janeiro).

A Redecomep-Rio, inaugurada em 2014, implantou uma malha de fibras óticas de mais de 350Km, e opera um *backbone* de 10Gbps sobre um suporte tecnológico de até 1,9Tbps em DWDM[2] (Pinto *et al.*, 2002)[3], (Alves Jr. *et al.*, 2004; Alves Jr., 2007; Esteves, 2013; Miranda, 2013; Miranda *et al.*, 2013; Diniz e Alves Jr., 2014; Macedo, 2016; Silva, 2016). No entanto, os desafios para as redes acadêmicas de alta velocidade, em especial como suporte à instrumentação científica remota em física, são ainda enormes. Apesar da infraestrutura local ter melhorado significativamente e as redes terem atingido um desempenho de altíssima capacidade em seu ponto final, obter um ótimo desempenho entre as redes envolvidas fim a fim, algumas vezes em escala global, é ainda um desafio.

Atualmente, grandes projetos de pesquisa científica em várias áreas da física, como por exemplo, em física de altas energias - *Worldwide LHC Computing Grid* (WLCG)[4], em astrofísica - *Brazilian Science Data Center* (BSDC) construção do centro de dados centrado em informações de astrofísica de alta energia), em cosmologia - o *Southern Astrophysical Research* (SOAR), dado as suas característica de grandes colaborações internacionais, fazem com que tanto pesquisadores, quanto instrumentação, armazenamento de dados e as ferramentas de processamento e análise estejam geralmente geograficamente distribuídos em escala planetária.

As atividades que antes eram realizadas em um único local, agora dependem muito do acesso a redes de alta capacidade para a troca de informações. Essa característica traz enormes desafios para os provedores de serviços de redes acadêmicas. A capacidade da rede e o gerenciamento de tráfego são pontos chave nesse quesito e um balanceamento adequado entre ambos é necessário para prover fluxos de rede de alta capacidade para transferência de grandes volumes de dados com ótimo desempenho, assim como para manter as atividades tradicionais dos usuários, como acesso *web*, por exemplo (Zurawski *et al.*, 2012).

Desse modo, os provedores de serviços de redes acadêmicas têm buscado o desenvolvimento de modelos de serviços diferenciados que permitam atender as demandas

---

[2] O DWDM, do inglês *Dense Wavelength Division Multiplexing*, é uma tecnologia de multiplexação de canais ópticos que permite a operação de até 64 canais de frequência em uma mesma fibra óptica.

[3]Em junho de 2014, foi inaugurada a RedeRio Metropolitana (Redecomep-Rio), uma parceria da RedeRio/FAPERJ com a RNP/MCTIC. Esta é uma infraestrutura de fibras óticas próprias que formam uma rede de alta velocidade para as instituições de ensino, ciência, tecnologia, inovação e de governo na cidade do Rio de Janeiro. A Redecomep-Rio interconecta 86 pontos, pertencentes a 51 instituições acadêmicas na região metropolitana do Rio de Janeiro estendendo-se por mais de 300 quilômetros em um *backbone* de 10Gbps em tecnologia DWDM (Multiplexação Densa por Divisão de Comprimento de Onda) atingindo uma banda agregada de até 1,9 Tbps. A coordenação e o projeto técnico da Redecomep-Rio ficaram sob a responsabilidade do CBPF (para mais informações vide: http://www.redecomep.rnp.br/?consorcio=2).

[4] O WLCG é uma colaboração global de aproximadamente 200 centros de supercomputação interconectados mundialmente, os quais proveem recursos de computação necessários para armazenar, distribuir e analisar o volume massivo de dados gerados pelo experimento *Large Hadron Collider* (LHC) localizado na Organização Europeia a Pesquisa Nuclear (CERN), com sede na divisa entre Suíça e França.

de troca de grandes volumes de dados científicos, enquanto também dão suporte a outros tipos de tráfego simultaneamente. Os serviços de provisionamento de circuitos virtuais dinâmicos, também conhecidos como serviços de transporte de camada 2, proveem redes escaláveis e flexíveis onde os seus usuários podem criar circuitos de camada 2 entre *endpoints* para atender a necessidades específicas, como os fluxos de rede de alta capacidade fim a fim.

Do ponto de vista dos provedores de serviços o advento de serviços de provisionamento de circuitos virtuais dinâmicos, no entanto, permite a criação de canais de rede com garantia de banda e duração pré-determinada, que levam a utilização mais eficiente da infraestrutura de rede. Diversos projetos de redes dedicadas a projetos de pesquisa na área de física, como o LHC *Optical Private Network* (LHCOPN) (WLCG, 2016) e o LHC *Open Network Environment* (LHCONE) (Martelli e Stancu, 2015; WLCG, 2016) podem tirar vantagem dos recursos de rede disponíveis através da otimização da utilização de banda disponível fazendo com que esses circuitos virtuais sejam utilizados como atalhos na topologia da rede. Nesse caso, o produto largura de banda por atraso (BDP, do inglês *Bandwidth-Delay Product*)[5], métrica importante para otimização de transferências massivas de dados, é reduzido pela minimização dos atrasos de enfileiramento de pacote nos *switches* presentes no caminho fim a fim. Todos esses avanços quando combinados com modelos de projetos de rede como o *Science DMZ*[6] (Dart *et al.*, 2014), podem otimizar a transferência de grandes volumes de dados fim a fim.

Nos últimos anos, a comunidade de Redes de Educação e Pesquisa (REN, do inglês *Research and Education Networks*) tem investido no desenvolvimento de diversas arquiteturas, protocolos e *softwares* controladores para suportar os serviços de provisionamento de circuitos virtuais (Rao *et al.*, 2005; Zheng *et al.*, 2005; Bobyshev *et al.*, 2006; Guok *et al.*, 2006; Yang *et al.*, 2006; Katramatos *et al.*, 2007), com enfoque principalmente voltado a projetos de *grids* computacionais para física de altas energias. O projeto *Lambda Station* (Bobyshev *et al*., 2006), idealizado pelo *Fermi National Accelerator Laboratory* (FermiLab) e o *California Institute of Technology* (Caltech), foi uma das primeiras iniciativas para provisionamento dinâmico de circuitos para aplicações científicas e utilizava técnicas de *Policy Based Routing* (PBR) para encaminhamento de tráfego entre *clusters* de computadores. Já o projeto *Terapaths* (Katramatos *et al.*, 2007) , financiado pelo Departamento de Energia dos Estados Unidos, propôs a criação de caminhos virtuais fim a fim com garantias de banda, combinando o uso de redes locais (LAN) baseadas em técnicas de *DiffServ* e redes de longa distância (WAN) baseadas em túneis *Layer 2 Virtual Private*

[5] O BDP representa a quantidade de dados necessária a ser enviada por um servidor antes de uma confirmação de recepção dos dados em uma conexão do *Transport Control Protocol* (TCP) para manter 100% de utilização da capacidade da rede.

[6] O ScienceDMZ é um modelo de projeto de topologia de rede desenvolvido pela rede de educação e pesquisa norte-americana ESnet e voltado para instituições de pesquisa, as quais necessitam de ambientes de conectividade customizados para aplicações científicas que demandam alto desempenho computacional. Recentemente, o CBPF também integra o projeto intitulado "DMZ Científica: Infraestrutura de Suporte para Aplicações de e-Ciência" da Rede Nacional de Ensino e Pesquisa, que visa implantar infraestruturas de ScienceDMZ em universidades e centro de pesquisa pelo país visando a otimização de transferências de grandes volumes de dados voltados a aplicações científicas.

Network (L2VPN) *Pseudowire* em redes *Multi Protocol Label Switching* (MPLS). Essa solução se baseava na utilização de técnicas de estabelecimento de circuitos conhecidas como L2VPN que permitem a criação de túneis virtuais sobrepostos a uma rede IP por exemplo, onde para o seu funcionamento é necessário a utilização de protocolos de encapsulamento. Por consequência, esse tipo de protocolo opera através de cabeçalhos adicionais incorporados aos pacotes tradicionais, o que faz com que mais recursos de rede sejam consumidos para o estabelecimento dos circuitos virtuais fim a fim. Soluções como UltraScience (Rao *et al.*, 2005), também financiado pelo Departamento de Energia dos Estados Unidos, e CHEETAH (Zheng *et al.*, 2005), proposto por pesquisadores da Universidade da Virginia e do Laboratório Nacional de *Oak Ridge*, se baseavam no estabelecimento de canais óticos dedicados, também conhecidos como L1VPNs óticos, em redes *Synchronous Optical Networking* (SONET) através do protocolo *Generalized Multi Protocol Label Switching* (GMPLS). Entretanto, esse tipo de implementação carece também de um conjunto de *softwares* e protocolos de sinalização e controle (*e.g.* OSPF-TE e RSVP-TE) que possuem grande complexidade de serem implementados para permitir o provisionamento de canais dedicados de forma dinâmica, além de haver a necessidade de cabeçalhos adicionais para encapsulamento.

No caso de redes IP convencionais, que não implementam nenhuma técnica para provisionamento de circuitos dinâmicos ou VPNs (seja uma rede de *campus* ou até mesmo um provedor de serviços que não implementa MPLS), muitas das vezes um *software* de controle tem de ser empregado para prover algum tipo de circuito estático exercendo funções de monitoramento e configuração dos *switches*, roteadores e enlaces. Tipicamente, esse tipo de *software* é implementado utilizando tecnologias proprietárias, que permitem o acesso somente aos operadores de rede do provedor, além de carecerem na maioria das vezes de configurações distribuídas realizadas manualmente em vários *switches* e roteadores no caminho (dispositivos esses, em geral, produzidos pelo mesmo fabricante do *software*). Além disso, cada tipo de técnica adotada para o provisionamento de circuitos virtuais de maneira estática demanda um conjunto de procedimentos específicos que depende do protocolo/solução que é implementado para o fornecimento do mesmo serviço. Exemplos dessas técnicas são os casos de utilização do *Layer 2 Tunneling Protocol* (L2TP) e de túneis *Generic Routing Encapsulation* (GRE). Por essa razão, esse mecanismo de funcionamento acaba por aumentar muito a complexidade de implantação de serviços de provisionamento de circuitos virtuais em redes IP convencionais.

Atualmente, os serviços de provisionamento de circuitos virtuais existentes e em operação em redes acadêmicas pelo mundo baseiam-se em técnicas de policiamento de tráfego para prover Qualidade de Serviço (QoS, do inglês *Quality of Service*) em redes que operam por meio de diferentes tipos de tecnologia, como *Ethernet*, SONET e MPLS. Exemplos de soluções para provimento desse tipo de serviço são o OSCARS (Guok *et al.*, 2006; ESnet, 2016), adotado pelo *backbone* da *Energy Sciences Network* (ESnet) do Departamento de Energia dos Estados Unidos, e o AutoBAHN (GEANT, 2017), adotado pela rede de educação e pesquisas da Europa denominada GEANT. Ambas as soluções se baseiam, basicamente, na utilização de protocolos abertos como *Inter Domain Controller Protocol* (IDCP) e *Network Services Interface* (NSI), mas cada uma delas disponibiliza um conjunto de *Application Programming Interfaces* (API) desenvolvidas para cada cenário onde são adotadas com suas especificidades, o que dificulta sua utilização em outras redes.

Entretanto, novas soluções para provisionamento de circuitos virtuais fim a fim têm surgido mais recentemente a partir da introdução do novo paradigma de Redes Definidas por *Software* (SDN, do inglês *Software Defined Networks*). Esse paradigma reduz a complexidade para implantação de serviços de provisionamento de circuitos virtuais dinâmicos já que permite o desacoplamento do *software* de plano de controle do dispositivo de encaminhamento. Isso faz com que esse *software* de controle, que era integrado fortemente aos equipamentos individuais de rede, possa ser implementado em controladores SDN externos (logicamente centralizados) em vez de nos próprios *switches*, permitindo a abstração da infraestrutura de rede para as aplicações e serviços de rede (Chaves Filho, 2015). Esse mecanismo de funcionamento permite que os controladores SDN tenham uma visão centralizada da rede e regras de encaminhamento sejam aplicadas aos *switches*, baseando-se em uma classificação de tráfego em fluxos de rede a partir somente das informações dos protocolos tradicionalmente utilizados em LANs e WANs, como *Ethernet*, IP e TCP. No momento atual, o OpenFlow é o padrão de interfaces de SDN para conexão entre controladores e dispositivos de encaminhamento mais amplamente aceito e implementado por fabricantes de dispositivos de rede, que disponibilizam recursos de SDN. Esse padrão prove, basicamente, a especificação para o canal de comunicação entre *switches* e controladores, e é baseado nele que a maioria das soluções/aplicações de rede disponibilizadas pela comunidade se baseiam (Kreutz *et al.*, 2015).

Essa abordagem de serviços de provisionamento de circuitos baseada em SDN/OpenFlow traz como uma de suas principais vantagens a não necessidade de protocolos de encapsulamento adicionais para estabelecimento de circuitos, visto que podem fazer uso de técnicas de modificação dos cabeçalhos já utilizados por padrão em redes locais, como é o caso do cabeçalho de VLAN. Outra vantagem também muito importante é a possibilidade de que quando esse serviço de provisionamento for utilizado, por exemplo para transferências de arquivos, as perdas de pacotes se mantenham baixas, visto que os recursos computacionais de rede podem ser estritamente atribuídos a fluxos de dados específicos, levando a disponibilização de serviços de *Bandwidth on Demand* (BoD). Exemplos desse tipo de solução de provisionamento baseado em SDN/OpenFlow são o OESS, (Tepsuporn *et al.*, 2015; GlobalNOC, 2016), utilizado inicialmente no *backbone* de educação e pesquisas da Internet2 nos Estados Unidos, e o DynPaC (Mendiola, Astorga, *et al.*, 2015), desenvolvido para utilização no *backbone* da GEANT.

Contudo, as soluções baseadas em SDN/OpenFlow como OESS e DynPaC, visam o provisionamento desse tipo de circuitos em redes **integralmente** SDN/OpenFlow. Essa característica é um problema visto que para o estabelecimento desse tipo de rede é, em geral, necessário a aquisição de novos recursos, tanto de novos enlaces para operar em paralelo a rede de produção IP tradicional quanto de novos equipamentos (*switches*) que suportem tecnologias de SDN/OpenFlow. Uma solução para esse problema é a utilização de redes SDN/OpenFlow **híbridas**, onde o tráfego IP tradicional de propósitos gerais roteado tradicionalmente pode ser encaminhado em paralelo ao tráfego destinado a uma aplicação específica, mas que se baseia nas regras de decisão de SDN/OpenFlow. Esse tipo de solução permite a utilização de equipamentos que já estejam em operação, mas que proveem algumas funcionalidades de SDN/OpenFlow.

Este trabalho, então, apresenta o desenvolvimento e caracterização de uma solução para serviços de provisionamento de circuitos virtuais dinâmicos do tipo L2VPN

Ethernet/VLAN em redes de *backbone* SDN/OpenFlow híbridas já em operação, em que o encaminhamento dos pacotes dos circuitos virtuais compartilha os mesmos recursos de rede já disponíveis, como enlaces físicos e roteadores/*switches*. Essa solução, denominada de ***Virtual Circuits Flow*** **(VCFlow),** é aberta e destinada a redes de *backbone* de provedores de serviço de rede em geral (não somente voltada a redes de educação e pesquisa) ou até mesmo para redes de *campus*. O VCFlow inclui como uma de suas principais vantagens um serviço de conectividade privada sem custos adicionais e de alta capacidade baseada na agregação das vantagens oferecidas por uma tecnologia bem compreendida, como o *Ethernet*, aliada ao novo paradigma de SDN/OpenFlow.

Através dessa solução, visa-se eliminar os problemas de: (a) adoção de protocolos de encapsulamento para estabelecimento de L2VPNs - baseando as decisões de encaminhamento somente no cabeçalho VLAN para criação dos circuitos L2VPN, e por consequência minimizando o *overhead* na rede; (b) configurações distribuídas pelos diversos elementos de rede - minimizando a intervenção humana através de uma aplicação baseada nos conceitos de SDN/OpenFlow; (c) necessidade da aquisição de novos equipamentos com suporte às funcionalidades de SDN/OpenFlow, baseando-se na sua implantação em redes híbridas.

O presente trabalho foi motivado pelo desejo de se desenvolver alternativas e melhorias para os serviços de alta capacidade de encaminhamento e de melhor esforço do IP provido pelo projeto RedeRio/FAPERJ e Redecomep-Rio, adicionando-se novos serviços para provisionamento de caminhos virtuais protegidos do tráfego IP de propósitos gerais. Tudo isso, com o intuito de prover segurança para transferência de dados entre as instituições afiliadas ao projeto e de prover também possíveis melhorias nas transferências de grandes volumes de dados fim a fim em redes SDN/OpenFlow híbridas, aliando os princípios de resiliência de redes tradicionais à programabilidade de SDN.

1.1 Objetivos

1.1.1 Objetivos Gerais

Propor uma solução de provisionamento de circuitos virtuais L2 do tipo L2VPN *Ethernet* VLAN resiliente e sob demanda, explorando as capacidades de controle centralizado da rede proposto pelo paradigma de SDN. Essa solução opera de uma forma mais simples do que a abordagem tradicional (como, por exemplo, *pseudowire*-MPLS), por meio de uma abordagem voltada para redes SDN/OpenFlow híbridas, visando facilitar e otimizar o processo de operação de uma rede de *backbone*.

1.1.2 Objetivos Específicos

Desenvolver uma solução de provisionamento de circuitos virtuais com as seguintes características:

1. Garantir o provisionamento rápido de circuitos virtuais com intervalo de tempo de estabelecimento inferior a um segundo;

2. Permitir a recuperação automática de falhas de circuitos virtuais;

3. Permitir o estabelecimento de circuitos virtuais em diferentes interfaces eletrônicas nos equipamentos da rede;

4. Permitir a criação de circuitos virtuais que interconectem VLANs (com suporte a diferentes "*tags*") em pontos distintos da rede e operando de forma transparente para o cliente/instituição/projeto;

5. Evitar a complexidade de protocolos de encapsulamento para estabelecimento de circuitos fim a fim;

6. Minimizar a intervenção humana na rede para configuração de circuitos;

7. Centralizar as configurações dos circuitos virtuais em um único ponto da rede, permitindo um melhor gerenciamento da rede, dos circuitos virtuais e da segurança das informações para a operação da rede;

8. Manter a escalabilidade da rede, assim como ocorre nos serviços de melhor esforço do *Protocolo Internet* (IP), através de sua implantação em uma topologia de rede SDN/OpenFlow híbrida.

1.2 Organização

Nesse contexto, esse documento descreve a concepção e funcionamento da solução VCFlow, desenvolvida nesse trabalho para dar suporte ao provisionamento de circuitos virtuais resilientes em redes SDN/OpenFlow híbridas. Também apresenta uma demonstração da sua implantação no *backbone* da Redecomep-Rio e a caracterização de seu funcionamento, apontando suas principais características e limitações. O texto é organizado conforme descrito a seguir.

No Capítulo 2, é abordado a fundamentação teórica dos temas necessários para a clara compreensão do funcionamento da solução VCFlow. Os fundamentos de redes de *backbone*/núcleo e sua arquitetura são inicialmente abordados e uma descrição do funcionamento de VPNs é também realizada, com intuito de apresentar sumariamente como operam as L2VPNs. Em sequência, o paradigma de SDN é apresentado, com especial enfoque ao protocolo OpenFlow. Por fim, as soluções de provisionamento de circuitos virtuais em redes SDN/OpenFlow existentes são apresentadas.

No Capítulo 3, é apresentada uma visão geral da solução VCFlow, descrevendo o mecanismo de configuração do circuito virtual por parte do operador de rede, o mecanismo de estabelecimento de circuito virtual fim a fim através do processo de VLAN *stitching* e o mecanismo de descoberta de topologia desenvolvido para operação em redes SDN/OpenFlow híbridas.

No Capítulo 4, são detalhados os experimentos para caracterização da solução VCFlow, os quais incluem: (i) avaliação da capacidade dos controladores OpenFlow, para decisão de qual opção mais adequada para o desenvolvimento do VCFlow; (ii) demonstração do sistema protótipo operando no *backbone* da Redecomep-Rio; (iii) avaliação do tempo de

convergência do sistema protótipo em ambientes emulados e tomando como estudo de caso a implementação do sistema no *backbone* da Redecomep-Rio; (iv) avaliação do tempo de estabelecimento de circuitos fim a fim e de processamento de mensagens de controle do protocolo OpenFlow pela solução VCFlow, também realizando as avaliações em ambientes emulados e como estudo de caso a Redecomep-Rio. Todos os resultados dos experimentos descritos no Capítulo 4 são apresentados e discutidos no Capítulo 5, demonstrando também as limitações de sua implementação.

Por fim, o Capítulo 6 finaliza esse trabalho trazendo as suas conclusões e os trabalhos futuros.

Capítulo 2

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo, é apresentada a fundamentação teórica dos assuntos essenciais para a compreensão tanto da concepção quanto do funcionamento da solução VCFlow, desenvolvida nesse trabalho, voltada a provisionar circuitos virtuais L2VPN *Ethernet*/VLAN em redes SDN/OpenFlow híbridas. Inicialmente, é explicitado o que é uma rede de *backbone* e qual a sua arquitetura básica. Em sequência, é descrito o funcionamento de VPNs e as diferenças entre L2VPNs e L3VPNs. Posteriormente, os conceitos essenciais de SDN são expostos, dando enfoque especial ao OpenFlow. Ao final, as soluções já existentes baseadas nos conceitos de SDN/OpenFlow voltadas ao provisionamento de circuitos virtuais dinâmicos são apresentadas, exibindo as suas principais limitações que justificam o desenvolvimento da solução VCFlow.

2.1 Redes de *Backbone* ou Núcleo

Uma rede de *backbone*, também denominada de rede de núcleo ou simplesmente *backbone*, é parte de uma rede de computadores capaz de interconectar diversas redes, provendo um caminho para o transporte de informações entre diferentes redes locais (LAN, do inglês *Local Area Networks*) ou sub-redes. Um *backbone* pode unir diversos tipos de redes em um mesmo local, em locais distintos em um ambiente de campus, ou até mesmo em áreas distantes geograficamente como o caso de uma *Wide Area Network* (WAN). Exemplos de redes de *backbone* são o *backbone* da Internet (Moss e Townsend, 2000), o *backbone* da Rede Nacional de Ensino e Pesquisa do Brasil (RNP), conhecida também como redeIpê [7] e o *backbone* acadêmico metropolitano da RedeRio de Computadores (rede acadêmica para pesquisa e ensino do Estado do Rio de Janeiro) e da Redecomep-Rio (que

[7] Mais informações podem ser obtidas em: https://www.rnp.br/servicos/conectividade/rede-ipe

foi inaugurada recentemente com um *backbone* de 10Gbps sobre um suporte tecnológico de até 1,9Tbps em DWDM[8]).

2.1.1 RedeRio Metropolitana (Redecomep-Rio)

A RedeRio Metropolitana (Redecomep-Rio) (Moraes *et al.*, 2015) é uma infraestrutura de fibras óticas próprias que formam uma rede de alta velocidade para as instituições de ensino, ciência, tecnologia, inovação e governo na cidade do Rio de Janeiro. A Redecomep-Rio, que opera como uma rede de núcleo IP tradicional puramente roteada, apesar de ser um *backbone* que atende instituições acadêmicas, também atende a diversos outros tipos de instituições, as quais incluem empresas ligadas à prefeitura, ao estado e ao governo federal.

Além de serviços de conectividade de alta velocidade e de alta confiabilidade a Internet, essas instituições também carecem de serviços que simulem redes privadas em cima dessa rede de núcleo compartilhada entre os diversos afiliados. Esse tipo de rede privada é também conhecido como Rede Privada Virtual (VPN, do inglês *Virtual Private Network*).

2.2 *Virtual Private Networks* (VPN)

Uma VPN é um conjunto de políticas que controlam a conectividade e a qualidade de serviço de uma rede privada. São redes que compartilham um meio físico comum, porém possuem, em geral, privacidade dos dados através de criptografia. São redes virtuais, pois não possuem enlaces ou linhas dedicadas entre suas extremidades. No entanto, para os usuários e clientes, essas redes são transparentes e possuem as mesmas funcionalidades de segurança de enlaces dedicados. As VPNs consistem em soluções simples e flexíveis, tratando-se de uma poderosa ferramenta de tunelamento oferecida pelos Provedores de Serviço de Internet (ISPs, do inglês *Internet Service Providers*) (Oliveira *et al.*, 2012). As VPNs foram originalmente introduzidas para permitir aos provedores de serviços o uso de uma mesma infraestrutura comum na emulação de enlaces ponto-a-ponto entre os *sites* dos clientes. Elas foram desenvolvidas nos moldes de uma rede geograficamente distribuída ou WAN, podendo abranger uma ampla área geográfica, com todos os atributos de segurança, gerenciamento e processamento (Ricci, 2007).

As VPNs são uma área de crescimento importante na Internet. Apesar de um dos objetivos da Internet ser proporcionar comunicação entre os nós da rede de modo irrestrito, existem muitas situações de uso em que se exige privacidade das informações e conectividade controlada. Exemplos dessas situações são transações financeiras, pagamentos, e até mesmo transferência de dados de pesquisa científica. A maneira de baixo custo e eficiente de obter tal resultado é a utilização de VPNs. Usando protocolos padronizados, as empresas são capazes de utilizar as VPNs como uma alternativa viável e de baixo investimento para substituição de circuitos privados e dedicados de dados, o que vem a contribuir com reduções significativas em seus custos de operação (Guimaraes *et al.*,

[8] O DWDM, do inglês *Dense Wavelength Division Multiplexing*, é uma tecnologia de multiplexação de até 64 canais de frequência em uma mesma fibra óptica.

2006). Portanto, a grande motivação das VPNs é fazer uso dos serviços existentes das redes IPs espalhadas mundialmente para estender as redes corporativas de uma empresa a pontos distantes, como outros escritórios, filiais, parceiros, e até mesmo residências, ao invés de se utilizar um grande número de linhas dedicadas para a interconexão entre os diversos pontos (Oliveira *et al.*, 2012).

O crescimento da utilização das VPNs tem sido acompanhado por uma explosão nas técnicas disponíveis para prover essa função. A maior parte dessas técnicas utiliza abordagens padronizadas, mas cada uma utiliza protocolos diferentes e possui suas próprias vantagens e desvantagens.

Desse modo, vários tipos de VPNs estão disponíveis para a comunidade atualmente e existem diversos tipos de classificação para tais, como as classificações baseadas em tipos de implementação (*e.g. intranet* ou *extranet*) ou em tipos de dispositivos (*hardware* ou *software*). Uma dessas classificações é a divisão em dois modelos de serviços (Knight e Lewis, 2004): VPNs de camada 2 do modelo de referência TCP/IP (L2VPNs do inglês *Layer 2 Virtual Private Networks*); e VPNs de camada 3 (L3VPNs do inglês *Layer 3 Virtual Private Networks*).

A principal aplicação da tecnologia de L2VPN tem sido a criação de topologias do tipo estrela, malha total ou malha parcial construídas a partir de conexões ponto-a-ponto para interconectar *sites* VPN. Isso difere do modelo L3VPN que oferece um serviço ponto-a-multiponto, onde cada conexão do dispositivo do cliente para a rede do provedor vê simplesmente um roteador IP adjacente como um *gateway* para todos ou alguns destinos (Knight e Lewis, 2004). Uma explicação mais detalha do funcionamento das L3VPNs e L2VPNs pode ser encontrada nas Seções 2.2.1 e 2.2.2, respectivamente.

Ambos os modelos de serviço de VPNs citados se baseiam na tecnologia de tunelamento. Essa tecnologia pode ser definida como o processo de encapsular um protocolo dentro de outro, através de protocolos como *Layer 2 Tunneling Protocol* (L2TP), *Generic Routing Encapsulation* (GRE), *Multi Protocol Label Switching* (MPLS), entre outros. Nesses casos, o pacote encapsulado é enviado pela Internet até o destino, onde é desencapsulado, para entrega ao destinatário. Após alcançar o destino, o pacote desencapsulado é encaminhado ao seu destino final, conforme delineado na Figura 2.1. O túnel é a denominação do caminho lógico que é percorrido pelos pacotes ao longo da rede.

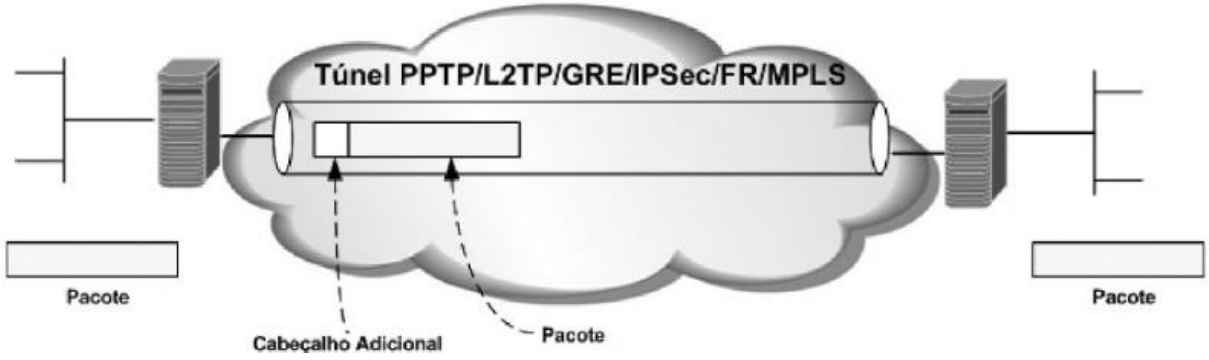


Figura 2.1: Processo generalizado de tunelamento para VPNs. Fonte:(Oliveira et al., 2012)

Dentro do contexto de VPNs, os seguintes termos são utilizados no restante do documento e definidos a seguir. A Figura 2.2 ilustra o escopo de alguns destes termos.

- *Customer Edge* (CE): Dispositivos de rede na localidade do cliente para estabelecimento de VPNs.
- *Provider (P):* Dispositivo de rede gerenciado pelo provedor de serviços para transporte dos pacotes referentes a cada túnel VPN.
- *Provider Edge* (PE): Dispositivo de rede na borda da rede do provedor de serviços para estabelecimento de VPNs.
- *Site:* um conjunto de usuários em uma rede local.
- Cliente VPN: uma empresa, organização ou instituição controlando múltiplos *sites*.

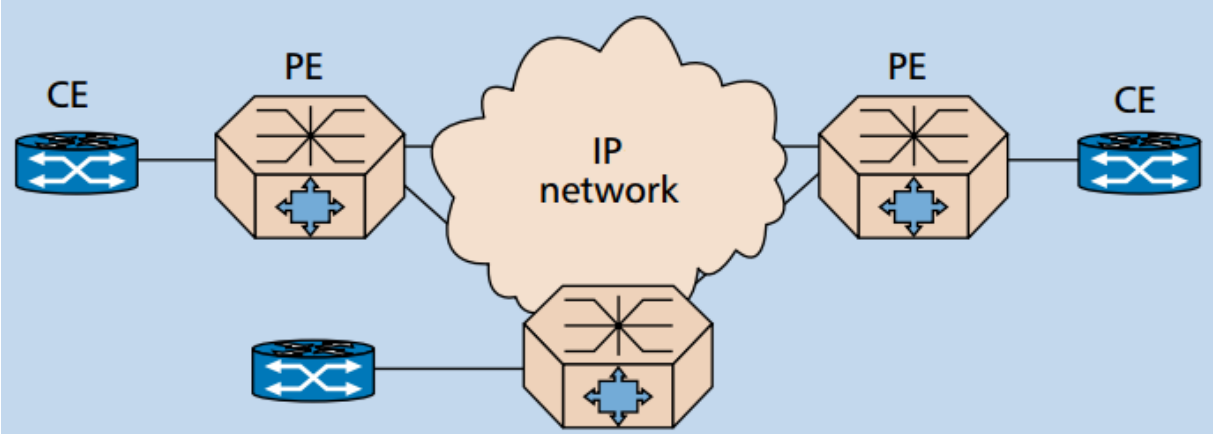


Figura 2.2: Arquitetura simplificada de uma VPN. Imagem adaptada de: (Knight e Lewis, 2004).

As L3VPNs e L2VPNs (objeto de estudo nesse trabalho) serão descritas de modo simplificado a seguir.

2.2.1 *Layer 3 Virtual Private Networks* – L3VPN

As L3VPNs referem-se à comunicação de camada 3 entre um conjunto de sites que fazem uso de uma infraestrutura de rede compartilhada (*e.g.* rede de um ISP). Elas podem ser divididas em dois tipos de serviços: (i) L3VPNs baseadas em PE (PE-*based*), também conhecida como L3VPNs baseadas em rede (*e.g.* BGP/MPLS IP VPNs (Rosen e Rekhter, 2006) e *Virtual Router* IP VPNs (Bao-Liang *et al.*, 2005)); (ii) L3VPNs baseadas em CE (*e.g.* CE-*based* IPSec VPNs (Seo e Kent, 2005)).

Nas L3VPNs baseadas em rede, o provedor de serviços configura os seus dispositivos PE para estabelecer as conexões VPN para cada cliente, sendo o dispositivo CE isento de quaisquer configurações adicionais para estabelecer as VPNs. Nesse caso, os dispositivos CE têm como requisitos somente possuir funcionalidades de roteamento básicas. A Figura 2.3 ilustra os elementos necessários para uma L3VPN baseada nos dispositivos PE. Nesse tipo de serviço, o *site* do cliente recebe o serviço de conectividade IP do provedor de serviços, podendo o dispositivo PE ser conectado através de enlaces de acesso a um ou mais dispositivos CE. O PE, então, é responsável por encaminhar os pacotes de dados dos usuários

clientes baseados no cabeçalho IP, como os endereços IPv4 ou IPv6. O dispositivo CE enxerga o dispositivo PE simplesmente como um dispositivo de camada 3, ou seja, como um roteador IPv4 ou IPv6 tradicional. Para estabelecer uma L3VPN de modo isolado para cada cliente, os *switches* PE contêm uma *Virtual Forwarding Instance* (VFI), também conhecida como *Virtual Routing and Forwarding* (VRF), para cada L3VPN que é entregue por esses dispositivos. As VFIs podem ser utilizadas como terminação de túneis para interconexão com outras VFIs e também como terminação para conexões de acesso para dispositivos CE. A VFI inclui as informações de roteamento para a L3VPN e permite que as funções de roteador dedicadas a uma VPN em particular sejam entregues, como a separação entre as funcionalidades de comutação e roteamento, e o suporte a espaços de endereçamento IP sobrepostos. Logo, os protocolos de roteamento nos dispositivos PE e CE interagem para popular a VFI.

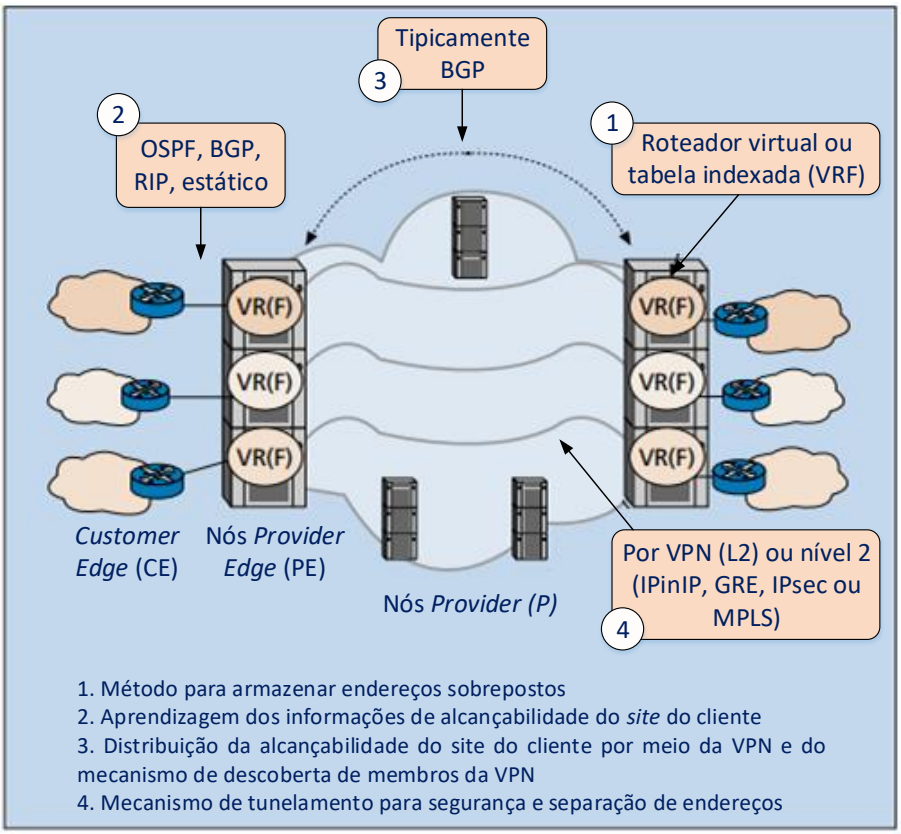


Figura 2.3: Componentes das L3VPNs baseadas em rede. Imagem adaptada de (Knight e Lewis, 2004)

Em contraste, as L3VPNs baseadas em CE podem ser implantadas pelos clientes sem nenhum requisito adicional por parte do provedor de serviços, além do fornecimento do acesso à Internet. Entretanto, nesse caso, os dispositivos CE necessitam suportar todas as funcionalidades necessárias para estabelecer a VPN, como o suporte a IPSec por exemplo.

A Figura 2.4 ilustra o modelo de L3VPN baseado em CE. Nesse tipo de serviço, um único tipo de túnel (como o IPSec por exemplo) termina em pares de CEs. Como os dispositivos CE, em geral, servem a um único site cliente, então o encaminhamento e roteamento é fisicamente separado de todos os outros clientes. Desse modo, os dispositivos PE não têm ciência da participação de dispositivos CE específicos em alguma VPN em

particular. Portanto, as funções da VPN são implementadas com as configurações provisionadas nos dispositivos CE, e a rede do provedor de serviços é utilizada somente para prover as funcionalidades de roteamento e comutação que suportam os *endpoints* do túnel entre os dispositivos CE.

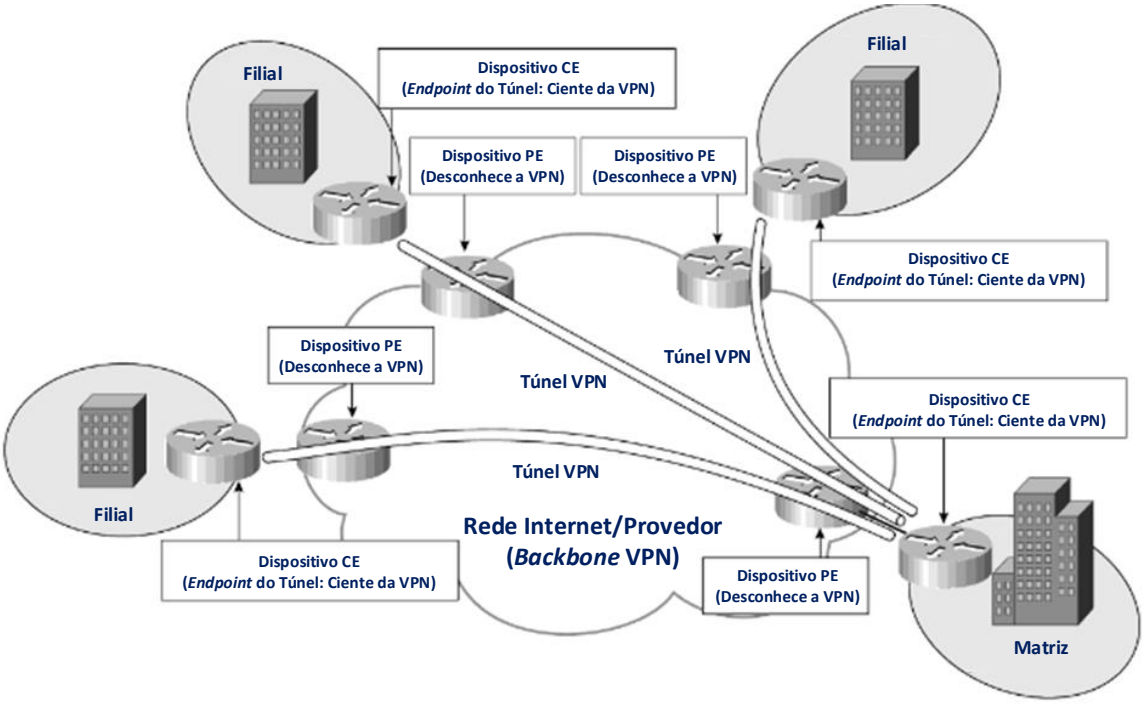


Figura 2.4: Exemplo de L3VPN baseada em CE site a site. Imagem adaptada de (Lewis, 2006).

2.2.2 *Layer 2 Virtual Private Networks* – L2VPN

As L2VPNs proveem uma conexão fim a fim de camada 2 entre os sites participantes, transportando quadros de camada 2 (e.g. *Ethernet* e ATM) entre eles. Uma de suas principais vantagens está no fato de suportar diferentes tipos de protocolos de mais alto nível (*e.g.* IPX e IPSec) (Chowdhury e Boutaba, 2010).

No caso de L2VPNs, o tráfego L2 dentro de uma conexão de uma rede privada, termo também denominado de circuito virtual, é transportado através do núcleo de uma rede (IP ou MPLS, por exemplo) em *pseudowires* ou pseudofios (emulações de enlaces físicos *Ethernet* baseadas em pacotes IP, por exemplo), que são transportados dentro de túneis. Essa abordagem ajuda a tornar serviços mais escaláveis, suportando um número muito grande de clientes, cada um deles com muitos *sites*. Uma vez que os dispositivos de roteamento dentro da rede de pacotes do provedor só precisam saber sobre os túneis entre os dispositivos de borda (PE), não é necessário saber de todas as conexões L2 individuais que existem entre os dois dispositivos de borda. Em resumo, a carga útil de L2 é encapsulada com um cabeçalho de *pseudowire*, que é encapsulado adicionalmente dentro de um cabeçalho de túnel. O cabeçalho do *pseudowire* atua como o campo para demultiplexação na terminação do túnel (Knight e Lewis, 2004). Todo esse processo pode ser sumarizado na Figura 2.5.

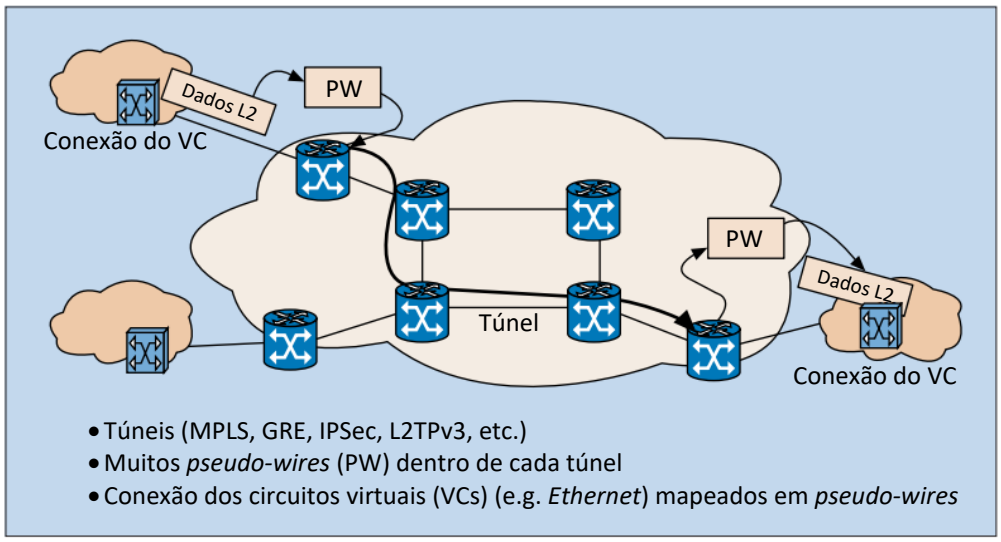


Figura 2.5: Arquitetura de L2VPNs. Imagem adaptada de: (Knight e Lewis, 2004).

Atualmente, há basicamente dois tipos de modelos de serviços de L2VPN que um provedor de serviços pode oferecer (Chowdhury e Boutaba, 2010): (i) ponto-a-ponto, denominado de *Virtual Private Wire Service* (VPWS); (ii) ponto-a-multiponto, denominado de *Virtual Private LAN Service* (VPLS). O modelo VPWS provê a emulação de conexões de camada 2 para enlaces *Ethernet* ou *Permanent Virtual Circuits* (PVC) ATM, por exemplo. Já o modelo VPLS, oferecido em geral para redes *Ethernet*, provê o mecanismo de aprendizagem de endereços MAC e replicação de pacotes para fazer com que o serviço de conectividade WAN oferecido pelo provedor de serviços aparente ser um *switch* conectado a cada *site* dos seus clientes (Knight e Lewis, 2004).

Conforme citado anteriormente, as L2VPN se baseiam em mecanismos de tunelamento proporcionados por protocolos como L2TP, GRE e MPLS. Entretanto, esses mecanismos geram consequências negativas para o seu pleno funcionamento, como *overhead* devido à adição de cabeçalhos adicionais em cada pacote para estabelecimento dos túneis, existência de mecanismos de configuração complexos e com informações espalhadas por toda a rede, e dificuldade de gerenciamento, monitoramento e *troubleshooting*. No entanto, a solução proposta neste trabalho visa justamente otimizar o processo de estabelecimento desse tipo de túneis e simplificar todo o processo de operação de uma rede de *backbone* de um provedor de Internet para provisionamento de circuitos virtuais do tipo L2VPN *Ethernet* VLAN, baseando-se no paradigma de Redes Definidas por *Software*.

2.3 Redes Definidas Por *Software* – SDN

As redes de computadores tradicionais são complexas e difíceis de se administrar (Benson *et al.*, 2009; Feamster *et al.*, 2014). Existem muitos tipos de equipamentos sendo utilizados nessas redes, como roteadores, *switches*, *firewalls* e servidores balanceadores de carga e para expressar as políticas de rede, os operadores de rede precisam configurar cada dispositivo separadamente através de comandos de baixo nível (protocolos individuais e configuração de cada interface, por exemplo), muitas das vezes específicos de cada fabricante. Além disso, as redes IP tradicionais são verticalizadas ou verticalmente

integradas. Isto é, o plano de controle dos dispositivos (que decide como lidar com o tráfego de rede) e o plano de dados ou de encaminhamento (que encaminha o tráfego de acordo com as decisões tomadas pelo plano de controle) são integradas de modo unificado nos dispositivos de rede, aumentando a complexidade e dificultando a inovação das redes (Feamster *et al.*, 2014).

As Redes Definidas por Software (SDN, do inglês *Software Defined Networking*) estão mudando a maneira de se projetar e gerenciar redes de computadores (Mckeown, 2011; Shenker *et al.*, 2011; Feamster *et al.*, 2014). As SDN possuem duas características principais as quais buscam endereçar as limitações de infraestrutura das redes atuais. Primeiramente, as SDN separam o plano de controle do plano de dados, quebrando a integração vertical dos dispositivos de rede fazendo com que roteadores e *switches* tornem-se simples dispositivos de encaminhamento de tráfego. Segundo, as SDN permitem consolidar o plano de controle, de modo que um único programa de controle centralizado seja capaz de controlar múltiplos elementos de planos de dados. Dessa maneira, esse programa de controle centralizado ou controlador simplifica a aplicação de políticas, a reconfiguração e a evolução da rede, dado que os operadores de rede não têm de configurar todos os dispositivos de rede individualmente para realizar modificações no comportamento da rede. Em vez disso, as decisões de encaminhamento de tráfego são tomadas em um local logicamente centralizado (Kim e Feamster, 2013).

O plano de controle SDN exerce controle direto sobre o estado de cada elemento de plano de dados (i.e., roteadores e *switches*) por meio de uma *southbound Application Programming Interface* (API), também conhecida como interface *southbound*. Além disso, existe também o conceito de interface *northbound.* A interface *northbound* determina como expressar tarefas operacionais e políticas de rede, e também como transformá-las em uma forma através da qual o controlador SDN ou Sistema Operacional de Rede (NOS, do inglês *Network Operating System*) possa entendê-las (Kim e Feamster, 2013; Feamster *et al.*, 2014). A Figura 2.6 ilustra a arquitetura de SDNs. Essas interfaces abertas permitem aos controladores de rede programar dispositivos de encaminhamento completamente heterogêneos, algo que é muito difícil de se obter em redes IP tradicionais devido à grande variedade de interfaces fechadas e proprietárias e à natureza distribuída do plano de controle.

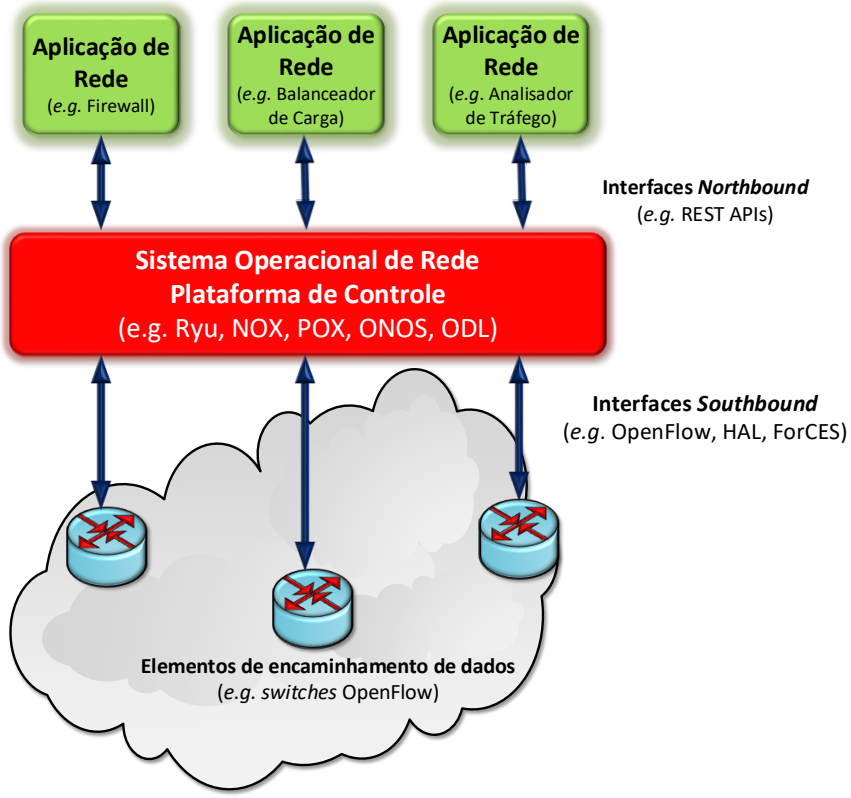


Figura 2.6: Visão simplificada da arquitetura de SDN.

As interfaces *southbound* são o elo de conexão entre os elementos de controle (controlador SDN) e os elementos de encaminhamento (*switches*), sendo um elemento crucial para a separação entre as funcionalidades dos planos de dados e de controle. Atualmente, muitas propostas de interfaces *southbound* como ForCES (Doria *et al.*, 2010), *Protocol-Oblivious Forwarding* (POF) (Song, 2013), OpFlex (Smith *et al.*, 2014) e *Hardware Abstraction Layer* (HAL) (Belter *et al.*, 2014; Parniewicz *et al.*, 2014) estão disponíveis. Entretanto, o OpenFlow (Mckeown *et al.*, 2008) é um dos padrões de interfaces *southbound* mais amplamente aceitas e implementadas para SDN (Kreutz *et al.*, 2015). Desse modo, a seção a seguir descreve detalhadamente o OpenFlow, visto que é objeto de estudo e desenvolvimento nesse trabalho.

2.3.1 OpenFlow – Visão Geral

O OpenFlow (Mckeown *et al.*, 2008) é um exemplo proeminente de interface *southbound*. Muito fabricantes, incluindo *Hewlett-Packard* (HP), *Nippon Eletric Company* (NEC), NetGear, *International Business Machines* (IBM), Cisco e muitos outros produzem *switches* que suportam OpenFlow e que estão disponíveis hoje no mercado. A fundação *Open Networking Foundation* (ONF) (ONF, 2016) é a responsável pela sua padronização e foi fundada por grandes empresas como Google, Facebook, Yahoo, Microsoft, Verizon e Deutsche Telekom

Na arquitetura do SDN/OpenFlow, há dois elementos principais que são os controladores e os dispositivos de encaminhamento, conforme ilustrado na Figura 2.7

adaptada de (Kreutz *et al.*, 2015). Um dispositivo de encaminhamento, também conhecido como dispositivo de plano de dados ou *switch* OpenFlow, é um elemento de *hardware* ou *software* especializado em realizar o encaminhamento de pacotes, enquanto que o controlador é um *software* rodando em uma plataforma de *hardware commodity*. Um dispositivo de encaminhamento OpenFlow funciona baseado em um *pipeline*[9] de tabelas de fluxos em que cada entrada dessas tabelas possui três partes: (i) regras; (ii) ações a serem executadas nos pacotes correspondentes a cada regra; (iii) contadores que mantêm estatísticas dos pacotes. Em síntese, um *switch* OpenFlow possui uma ou mais dessas tabelas contendo regras para tratamento de pacotes. Cada uma dessas regras corresponde a um determinado tipo de tráfego e realiza determinadas ações nesse tipo de tráfego, onde essas ações incluem descartar, encaminhar ou realizar inundação dos pacotes. Dependendo de cada tipo de regra instalada, o *switch* OpenFlow pode atuar como um dispositivo de rede distinto, como por exemplo um roteador, um *firewall* ou um balanceador de carga.

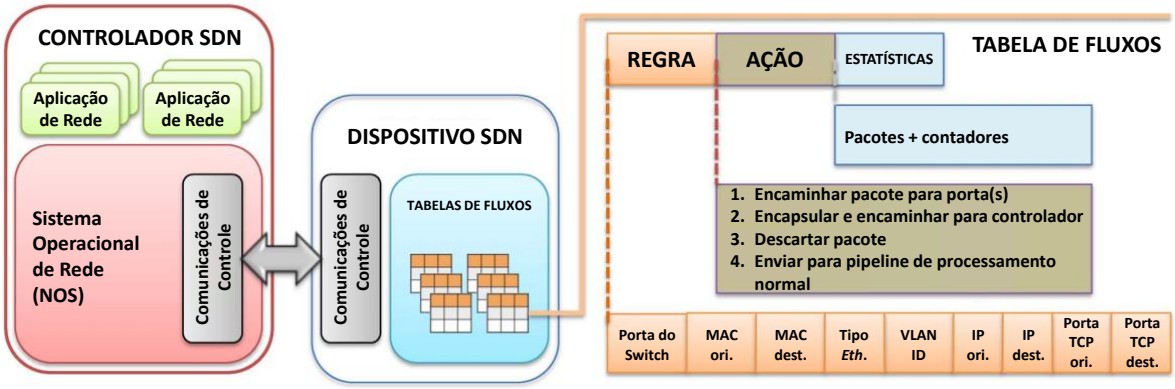


Figura 2.7: Arquitetura do SDN/OpenFlow. Imagem adaptada de (Kreutz et al., 2015).

Atualmente, diferentes versões do protocolo OpenFlow estão disponíveis. A primeira versão a ser lançada foi a 0.2.0 em março de 2008. Após esse período várias outras versões foram lançadas, como as versões 1.0.2, 1.1.0, 1.2, 1.3.5, 1.4.1 e no momento de desenvolvimento desse trabalho a mais recentemente lançada é a versão 1.5.1 de março de 2015 (ONF, 2015b). Apesar de atualmente a versão 1.0 ser a mais amplamente difundida e implementada, a proposta deste trabalho é baseada na versão 1.3, pois disponibiliza uma característica importante que a diferencia das versões anteriores, conhecida como "*feature Meter"*. A "*feature Meter"* permite que um *switch* OpenFlow seja capaz de medir e controlar a taxa de encaminhamento de pacotes para garantia de *Quality of Service* (QoS). Apesar da gama de possibilidades de aplicações que essa característica permite, ela não foi utilizada no trabalho descrito nesse documento, devido a limitações de *hardware* impostas pelos equipamentos onde a solução protótipo foi implantada. Entretanto, dado o potencial dessa funcionalidade a solução VCFlow foi desenvolvida inicialmente baseada na utilização de OpenFlow 1.3, visa fazer uso dessa característica futuramente.

[9] *Pipeline* é definido em (ONF, 2015a) como o conjunto de tabelas de fluxo vinculadas entre si que fornecem correspondência, encaminhamento e modificação de pacotes em um *switch* OpenFlow.

2.3.2 OpenFlow – Tipos de *Switches:* Híbridos ou Puros

Os *switches* OpenFlow são disponibilizados de dois modos: OpenFlow puros e OpenFlow híbridos. Os *switches* OpenFlow puros suportam somente as operações disponibilizadas pelo OpenFlow, em que todos os pacotes são processados pelo *pipeline* OpenFlow e não podem ser processados de nenhuma outra maneira.

No caso dos *switches* OpenFlow híbridos, eles suportam ambas as operações de encaminhamento baseado em OpenFlow e de comutação *Ethernet* tradicional. Nesse caso, esses *switches* proveem um mecanismo de classificação fora do escopo da especificação do OpenFlow, capaz de rotear o tráfego para ambos os *pipelines* OpenFlow ou de encaminhamento tradicional (ONF, 2015a). Por exemplo, um *switch* pode utilizar uma *tag* de VLAN ou uma porta de entrada do pacote para decidir como processar o pacote, utilizando o *pipeline* OpenFlow ou o *pipeline* tradicional.

Existem, basicamente, dois modos de operação dos *switches* OpenFlow híbridos. Em ambos os casos, duas instâncias de planos de dados podem ser criadas, sendo uma delas referente ao *pipeline OpenFlow* e outra ao *pipeline* de encaminhamento tradicional. Em um dos modos de operação (também conhecido por alguns fabricantes como *hybrid switch mode*) cada porta pode ser associada a somente uma dessas instâncias. Nesse caso, todo o tráfego em uma determinada porta é encaminhado baseado ou no *pipeline* OpenFlow ou no *pipeline* tradicional. No outro modo de operação (também conhecido por alguns fabricantes como *hybrid port mode*), ambas as instâncias de planos de dados podem ser associadas a uma mesma porta. Porém, cada instância é associada a um tipo de tráfego. Por exemplo, o tráfego *Ethernet* com *tags* de VLAN 802.1q pode ser associado à instância OpenFlow enquanto que todo o resto é associado à instância tradicional. A Figura 2.8 ilustra os modos de operação dos *switches* OpenFlow híbridos.

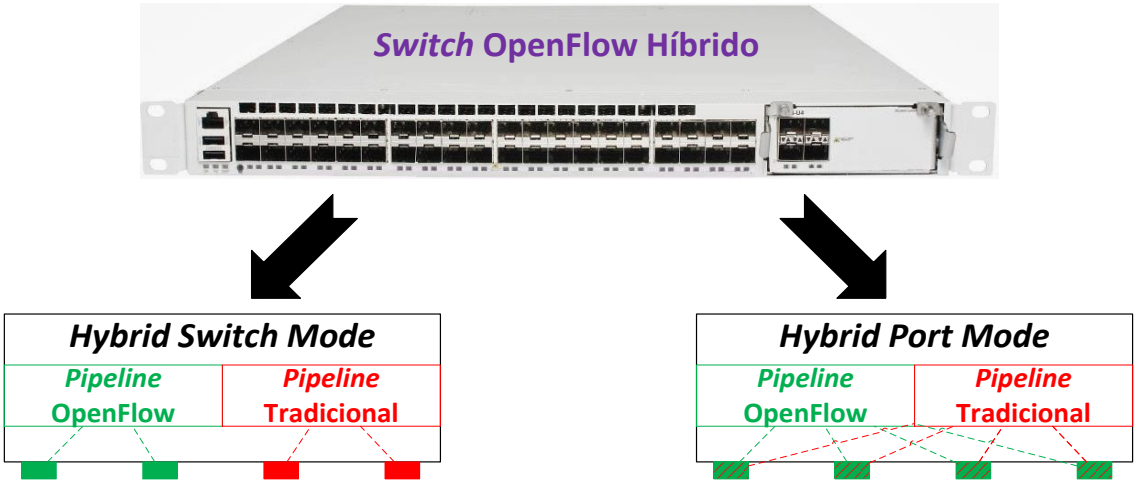


Figura 2.8: Modos de operação dos switches OpenFlow híbridos.

A partir desse momento, uma rede que opera por meio de *switches* OpenFlow híbridos em que ambas as instâncias de plano de dados OpenFlow e tradicional compartilham a mesma porta em *hybrid port mode* é denominada como uma rede SDN/OpenFlow híbrida ou simplesmente uma rede híbrida.

2.3.3 OpenFlow – Processamento de Tabelas de Fluxo

Dentro de um dispositivo OpenFlow, um caminho através de uma sequência de tabelas de fluxo define como os pacotes devem ser manipulados. Quando um novo pacote chega, o processo de pesquisa é iniciado na primeira tabela e termina com uma correspondência em uma das tabelas do *pipelin*e ou com uma falha (quando nenhuma regra de fluxo é encontrada para esse pacote). Uma regra de fluxo pode ser definida combinando diferentes campos de correspondência, como ilustrado na Figura 2.7. Se não houver nenhuma regra padrão para o caso de falha, o pacote é descartado. No entanto, o que é feito em geral é instalar uma regra padrão que indica ao *switch* para enviar o pacote para o controlador. A ordem em que as regras são processadas segue o número de sequência natural das tabelas e a prioridade das entradas em cada tabela de fluxo. As ações possíveis para cada entrada da tabela de fluxos incluem: encaminhar o pacote para a(s) porta(s) de saída; encapsulá-lo e encaminhá-lo para o controlador; descarta-lo; enviá-lo para o *pipeline* de processamento normal (no caso de *switches* OpenFlow híbridos); enviá-lo para a próxima tabela de fluxo ou para tabelas especiais, como tabelas de grupo ou de medição.

No caso do OpenFlow versão 1.3, uma tabela de fluxos consiste de entradas ou regras, as quais contêm os itens que podem ser vistos na Tabela 2.1, e são descritos na sequência.

Tabela 2.1: Principais componentes de uma entrada de fluxo em uma tabela de fluxo para o OpenFlow versão 1.3.

Match Fields	*Priority*	*Counters*	*Instructions*	*Timeouts*	*Cookie*	*Flags*

- *Match Fields*: campos dos cabeçalhos dos pacotes para correspondência. Consiste da porta de entrada e dos cabeçalhos dos pacotes, e opcionalmente campos de outros *pipelines* como metadados especificados por tabelas anteriores.
- *Priority*: precedência de correspondência da entrada de fluxo definida pelo controlador.
- *Counters*: contador atualizado quando ocorre correspondência de pacotes.
- *Instructions*: modifica o conjunto de ações a serem tomadas sobre os pacotes ou o processamento de *pipelines*.
- *Timeouts*: quantidade máxima de tempo ou de tempo ocioso antes de uma entrada de fluxo ser expirada.
- *Cookie*: valor arbitrário escolhido pelo controlador e que pode ser utilizado pelo controlador para filtrar entradas de fluxo afetadas por estatísticas de fluxo, modificações de fluxos e requisições de deleção de fluxo, não sendo utilizada para processamento dos pacotes pelo *switch.*
- *Flags:* altera o modo como as entradas de fluxo são gerenciadas.

Cada entrada da tabela de fluxos é identificada pelos itens *Match Fields* (Campos de Correspondência) e *Priority* (Prioridade), os quais identificam unicamente uma entrada em uma tabela de fluxos específica. Os campos de correspondência podem se referenciar a valores de campos de cabeçalhos extraídos dos cabeçalhos dos pacotes ou são valores

atribuídos ao pacote para o processamento do *pipeline* e não são associados aos cabeçalhos dos pacotes. Exemplos de campos de correspondência extraídos dos cabeçalhos são endereço *Ethernet* de origem e campo VLAN ID e de campos para processamento é o *in_port* que representa a porta lógica de entrada dos pacotes no *switch*. Cabe ressaltar, um *switch* OpenFlow deve suportar somente alguns desses campos obrigatoriamente. Além disso, esses campos obrigatórios não precisam ser implementados em todas as tabelas de fluxos e não precisam todos ser implementados na mesma tabela de fluxos. A Tabela 8.1, presente na Seção 8. Apêndice A – Campos de Correspondência de Fluxos para OpenFlow versão 1.3, descreve em detalhes cada um desses campos.

Cada entrada da tabela de fluxos contém também um conjunto de instruções o qual contém ações a serem executadas nos pacotes. Esse conjunto de instruções é executado quando um pacote corresponde a uma entrada da tabela de fluxos, sendo que essas instruções resultam em modificações no pacote, no conjunto de ações e/ou no processamento do *pipeline*. Nem todos os tipos de instruções são obrigatoriamente suportados pelos *switches* OpenFlow versão 1.3, somente os marcados pelo caractere asterisco (*) ao lado de cada um dos itens da Tabela 2.2 a seguir.

Tabela 2.2: Tipos de conjuntos de instrução executadas quando um pacote corresponde a uma entrada da tabela de fluxos OpenFlow versão 1.3.

Tipos de Instrução	Descrição
Meter <meter_id>	Direciona o pacote para o *meter_id* da tabela de medição (*Meter Table*). Um *meter_id* é uma entrada da tabela de medição responsável por medir a taxa de pacotes associada a essa entrada e permite controlar a taxa desses pacotes. Cada *meter_id* pode ser atribuído diretamente às entradas de fluxo. Como resultado da medição o pacote pode ser descartado.
Apply-Actions <action(s)>	Aplica a ação (*action*) especificada imediatamente, sem nenhuma modificação ao conjunto de ações. Deve ser utilizada para modificar o pacote durante o processamento entre duas tabelas ou para executar múltiplas ações do mesmo tipo. As ações são especificadas como uma lista de ações.
*Clear-Actions**	Limpa todas as ações no conjunto de ações imediatamente.
*Write-Actions <action(s)>**	Funde o conjunto de ações especificado no conjunto de ações atual referente à entrada da tabela de fluxos associada a alguns dos campos de correspondência de fluxos. Se uma ação do dado tipo existe no conjunto atual, sobrescreve, senão adiciona. Se uma ação do tipo *set-field* com um dado tipo de campo existe no conjunto atual, sobrescreve, senão adiciona.
Write-Metadata <metadata/mask>	Escreve o valor dos metadados com máscara no campo de metadados. A máscara especifica quais *bits* do registrador de metadados que devem ser modificados.

*Goto-Table <next-table-id>**	Indica o identificador da próxima tabela (*next-table-id*) no processamento do *pipeline*. O próximo *table-id* deve ser maior do que o *table-id* atual, referente à entrada da tabela de fluxos.

Um conjunto ou lista de ações é associado a cada pacote, sendo que esse conjunto é vazio por padrão. Uma entrada de fluxo pode modificar o conjunto de ações utilizando as instruções *Write-Action* ou *Clear-Action* associadas a um determinado tipo de fluxo. O conjunto de ações é carregado entre tabelas. Quando um conjunto de instruções referente a uma entrada de fluxo não contém uma instrução do tipo *Goto-Table*, o processamento do *pipeline* para e as ações no conjunto de ações do pacote são executadas. As ações da lista de ações são executadas na ordem especificada pela lista e são aplicadas imediatamente aos pacotes. Os tipos de ações disponíveis em *switches* OpenFlow versão 1.3 podem ser encontrados na Tabela 2.3, porém nem todas as ações são obrigatórias. Somente as ações marcadas pelo caractere asterisco (*) ao lado de cada um dos itens da Tabela 2.3 é obrigatória.

Tabela 2.3: Tipos de ações do OpenFlow versão 1.3.

Tipos de Ações	Descrição
*Output <port_no>**	Encaminha um pacote para uma porta lógica ou física do *switch* OpenFlow.
*Group <group_id>**	Processa o pacote através da entrada *<group_id>* da tabela de grupos.
*Drop**	Não há ação explícita para descarte de pacotes. Os pacotes cujo conjunto de ações não possui nenhuma ação de encaminhamento a uma porta de saída e nenhuma ação associada à tabela de grupos devem ser descartados.
Set-Queue <queue_id>	Estabelece o identificador *queue_id* da fila para um pacote. Quando pacote é encaminhado para uma porta utilizando uma ação do tipo *Output,* o *queue_id* determina qual fila associada a porta especificada pela ação *Output* deve ser usada para escalonamento e encaminhamento de pacotes. O comportamento do encaminhamento é direcionado pela configuração da fila e é utilizado para prover suporte a Qualidade de Serviço (QoS).
Push-Tag/Pop-Tag <ethertype>	Os *switches* devem oferecer suporte à adição/remoção (*push/pop*) de *tags* referentes aos protocolos VLAN, *Q-in-Q*, MPLS e *Provider Backbone Bridges* (PBB).
Set-Field <field_type> <value>	Modificam os valores dos campos dos cabeçalhos de alguns protocolos no pacote. As diversas ações do tipo *Set-Field* são identificadas pelo campo *field_type,* conforme Tabela 8.1. Em princípio, todos os campos descritos na Tabela 8.1 podem ser utilizados.

Change-TTL <ttl>	Os vários tipos de ações *Change-TTL* modificam os valores dos campos TTL do IPv4, *Hop Limit* do IPv6 e TTL do MPLS no pacote. Apesar de não ser obrigatório, esse tipo de ação aumenta muito a utilidade dos *switches* OpenFlow para implementação de funções de roteamento.

2.3.4 OpenFlow – Mensagens de Sinalização e Controle entre *Switch* e Controlador

O protocolo OpenFlow também implementa um conjunto de mensagens de sinalização e controle trocadas entre o controlador e o *switch*. Essas mensagens são responsáveis por realizar diversas ações como: verificação de características, configuração, modificação de estados, leitura de estados, envio de pacotes e mensagens de barreira.

Na presente seção, são descritas as mensagens tidas aqui como essenciais para a compreensão da operação da solução VCFlow proposta nesse documento e descrita no Capítulo 3. Proposta de Solução de Provisionamento de Circuitos Virtuais em Redes SDN/OpenFlow Híbridas. Dentre os tipos de mensagens mais relevantes estão o *Packet-In*, o *Packet-Out* e o *Flow-Mod*.

O *Packet-In* é uma mensagem assíncrona enviada do *switch* para o controlador para notificar a chegada de um fluxo não classificado. O pacote referente a esse fluxo é, em geral, "bufferizado" pelo *switch*, e um conjunto de informações sumarizadas são enviadas ao controlador. Cada mensagem possui um conjunto de campos a serem utilizados pelo controlador para definir a classificação dos fluxos. Esses campos podem ser vistos na Tabela 2.4. Após o controlador definir as ações que devem ser tomadas sobre esse tipo de fluxo, o controlador encaminha mensagens do tipo *Packet-Out* para o *switch*, para executar ações sobre o pacote "bufferizado", e/ou encaminha mensagens do tipo *Flow-Mod* para definir uma entrada na tabela de fluxos.

Tabela 2.4: Campos da mensagem Packet-In do protocolo OpenFlow versão 1.3.

Campos da Mensagem *Packet-In*	**Descrição**
Buffer Id	Valor de identificação único atribuído pelo *switch* OpenFlow para identificar um pacote "bufferizado". Caso o *switch* não implemente "bufferização", os pacotes são encapsulados e encaminhados integralmente ao controlador para análise.
Total Length	Tamanho total do pacote que levou ao encaminhamento da mensagem de *Packet-In*.
Reason	Razão pela qual a mensagem é enviada ao controlador. As possibilidades são: (a) *NO_MATCH* - nenhuma entrada correspondente ao fluxo foi encontrada na tabela de fluxos; (b) *ACTION* - ação explícita de encaminhamento para o controlador; *(c) INVALID_TTL* - pacote tem campo TTL inválido.
Table Id	Valor identificador da tabela de fluxos em que o processamento do pipeline está sendo realizado.

Cookie	Contém o *cookie* da entrada de fluxo que levou o pacote a ser encaminhado ao controlador.
Match Fields	Contém os campos de correspondência existentes no pacote que levou ao encaminhamento da mensagem de *Packet-In*, conforme descritos na Tabela 8.1.

O *Packet-Out* é uma mensagem enviada do controlador para o *switch*, em resposta a um *Packet-In*, indicando qual ação deve ser tomada para aquele pacote. Assim como a mensagem de *Packet-In*, as mensagens de *Packet-Out* são definidas por um conjunto de campos utilizados para indicar qual ação a ser tomada sobre os pacotes. A Tabela 2.5 descreve esses campos.

Tabela 2.5: Campos da mensagem Packet-Out do protocolo OpenFlow versão 1.3.

Campos da Mensagem *Packet-Out*	**Descrição**
Buffer Id	Assim como nas mensagens de *Packet-In*, um valor de identificação único é atribuído pelo *switch* OpenFlow para identificar um pacote "bufferizado". Caso o pacote não tenha sido "bufferizado", os dados do pacote são incluídos no *array* de dados e encaminhados ao *switch.* O pacote encapsulado na mensagem é, então processado pela lista de ações da mensagem.
In Port	Especifica a porta de entrada que deve ser associada aos pacotes para o processamento OpenFlow.
Actions Length	Tamanho da lista de ações em *bytes*.
Actions	Lista de ações definindo como o pacote deve ser processado pelo *switch.* Uma lista das ações disponíveis pode ser encontrada na Tabela 2.3.
Data	Campo vazio quando o *Buffer Id* é especificado. Quando o *Buffer Id* não é especificado o campo *Data* contém o pacote a ser processado. O formato do pacote é um quadro *Ethernet*, incluindo o cabeçalho *Etherne*t e a sua carga útil.

Já o *Flow-Mod* é também enviado do controlador para o *switch* para modificar o estado do mesmo, podendo realizar diversos comandos como adição e modificação de entradas na tabela de fluxos. Assim como as mensagens de *Packet-Out*, as mensagens do tipo *Flow-Mod* possuem campos específicos para realização das modificações nas tabelas de fluxos. Esses campos podem ser encontrados a seguir na Tabela 2.6.

Tabela 2.6: Campos da mensagem Flow-Mod do protocolo OpenFlow versão 1.3.

Campos da Mensagem *Flow-Mod*	**Descrição**
Cookie	Valor aleatório atribuído pelo controlador para identificar uma entrada da tabela de fluxos. Pode ser utilizado para filtrar estatísticas de fluxos, realizar modificação de fluxos e deleção de fluxos.
Cookie Mask	Utilizado pelo campo *Cookie* para restringir a correspondência de fluxos enquanto modifica ou deleta entradas da tabela de fluxos.
Table Id	Especifica a tabela em que a entrada de fluxo deve ser inserida, modificada ou deletada.
Command	Indica o tipo de comando que deve ser executado pela mensagem de *Flow Mod*. São eles: (a) *ADD* - adiciona nova entrada de fluxo; (b) *MODIFY - modifica* todos os fluxos que correspondam ao campo *Match; (c) MODIFY STRICT* - modifica a entrada que corresponde estritamente ao campo *Match* e ao campo *Priority*; (d) *DELETE* - deleta todos os fluxos correspondentes; (e) *DELETE STRICT* - deleta a entrada que corresponde estritamente aos campos *Match* e *Priority.*
Idle Timeout	Utilizado pelo mecanismo de expiração de fluxos. A entrada de fluxo expira após *Idle Timeout* segundos em que nenhum tráfego recebido.
Hard Timeout	Utilizado também pelo mecanismo de expiração de fluxos. A entrada de fluxo expira em *Hard Timeout* segundos independentemente se pacotes referentes à entrada são recebidos ou não.
Priority	Indica a prioridade da entrada de fluxo dentro da tabela de fluxos especificada. Números maiores indicam maior prioridade quando ocorre correspondências na tabela de fluxos.
Buffer Id	Identificador do pacote “bufferizado” no *switch* e enviado ao controlador pela mensagem de *Packet-In. Se a mensagem Flow Mod* inclui um *Buffer Id* válido, o pacote correspondente é removido do *buffer* e processado através de todo o *pipeline* OpenFlow após o fluxo ser inserido, iniciando a partir da primeira tabela de fluxos.
Out Port	Filtra opcionalmente o escopo das mensagens de *Flow Mod* do tipo *DELETE* e *DELETE STRICT* pela porta de saída.
Out Group	Filtra opcionalmente o escopo das mensagens de *Flow Mod* do tipo *DELETE* e *DELETE STRICT* pelo grupo de saída.

Flags	Mapa de *bits* que inclui as indicações de: (a) *SEND FLOW REM* - envia mensagens de fluxo removido para o controlador quando um fluxo é removido ou expirado; (b) *CHECK OVERLAP* - checa a sobreposição de entradas primeiramente; (c) *RESET COUNTS* - Reinicializa os contadores de *bytes* e pacotes da entrada de fluxo; (d) *NO PKT COUNTS* - Não realiza contagem de pacotes; (e) *NO BYT COUNTS* - Não realiza contagem de *bytes.*
Match	Contém a estrutura de campos de correspondência conforme a Tabela 8.1, definindo como a entrada de fluxo corresponde aos pacotes. A combinação dos campos *Match* e *Priority* identificam unicamente uma entrada na tabela de *fluxos.*
Instructions	Contém o conjunto de instruções para a entrada de fluxo quando se realiza adição ou modificação das entradas, conforme a Tabela 2.2. Cada conjunto de instruções pode conter uma lista de ações conforme a Tabela 2.3.

2.3.5 OpenFlow – Descoberta de Topologia

Não há nenhuma funcionalidade dedicada nos *switches* OpenFlow para realizar a descoberta de topologia, sendo de inteira responsabilidade do controlador implementar esse serviço. Além disso, não há ainda nenhuma padronização oficial que define um método de descoberta de topologia em SDNs baseadas em OpenFlow. Entretanto, o mecanismo referido como *OpenFlow Discovery Protocol* (OFDP), baseado no *Link Layer Discovery Protocol* (LLDP) padrão IEEE 802.1AB (IEEE, 2009), é o mecanismo utilizado mais comumente pelos controladores OpenFlow (Pakzad *et al.*, 2014). Como ainda se carece de um termo oficial para a descoberta de topologia, no restante deste documento será utilizado o LLDP.

Esse protocolo opera na camada de enlace do modelo de referência TCP/IP permitindo que informações como nome de *host*, versão do Sistema Operacional, endereço MAC da interface, entre outras, sejam aprendidas de modo dinâmico pelos equipamentos diretamente conectados.

No caso de redes SDN/OpenFlow, o procedimento de descoberta de topologia ocorre da seguinte forma, conforme a Figura 2.9:

1) Controlador envia mensagem de *Packet-Out* requisitando ao *switch* 1 enviar pacote LLDP em uma de suas interfaces.
2) *Switch* 1 envia pacote LLDP para o *switch* vizinho. O pacote LLDP que é enviado possui uma *tag* com um *DataPathID* (DPID) (identificador único para cada *switch* OpenFlow, que em geral é baseado no endereço MAC da interface de gerencia associada ao controlador) e o número da interface através da qual o pacote é enviado.
3) *Switch* 2 vizinho recebe o pacote, porém não tem entrada na tabela de fluxos.

4) *Switch* 2 envia mensagem de *Packet-In* para o controlador, indicando: o recebimento de pacote LLDP, a porta em que o pacote é recebido e o *DataPathID* do *switch* 2.
5) Controlador processa o *Packet-In* e estabelece uma adjacência entre os *switches* através dos seguintes passos: (a) descobre que o *switch* 1 é vizinho do *switch* 2, por meio dos DPIDs dos *switches*; (b) descobre a porta na qual o *switch* 1 está conectado ao *switch* 2, por meio da *tag* no pacote LLDP e da porta de entrada na mensagem de *Packet-In*, respectivamente.
6) Controlador repete o processo para todos os *switches* em todas as interfaces.

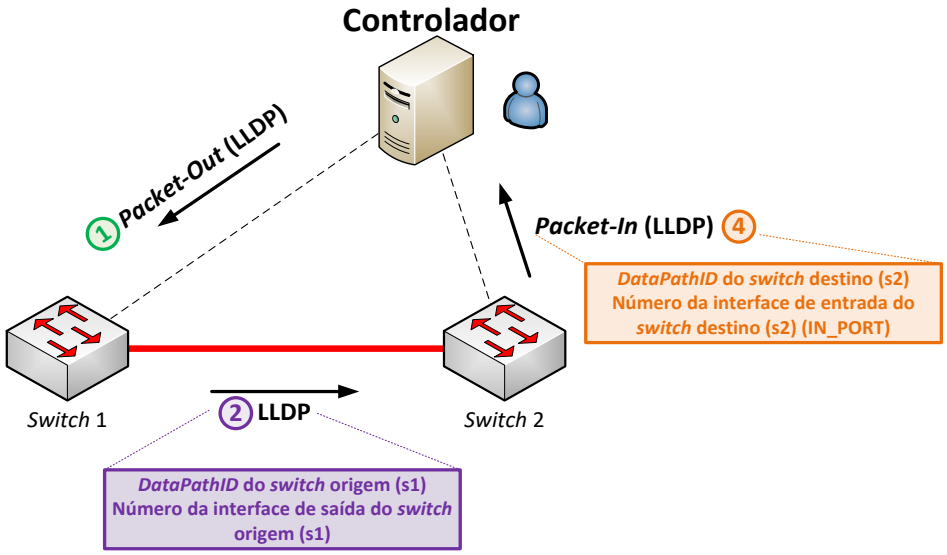


Figura 2.9: Esquema de funcionamento da descoberta de topologia em um ambiente OpenFlow através do protocolo LLDP.

2.4 Provisionamento de Circuitos Virtuais em Redes Definidas por *Software*

O tema de provisionamento de circuitos virtuais já foi tratado por iniciativas baseadas em SDN, entretanto cada uma delas se destina a aplicação em um cenário específico. Nas próximas seções suas características, contribuições e limitações serão abordadas.

2.4.1 DynPaC

O projeto *Dynamic Path Computation* (DynPaC) (Mendiola, Astorga, *et al.*, 2015) é um *arcabouço* para provisionamento de serviços de camada 2 resilientes sob demanda com limitação de banda, através de computação dinâmica de caminhos em redes SDN/OpenFlow. Esse projeto foi concebido como uma Atividade de Pesquisa Conjunta (*Joint Research Activity*) focada em serviços de rede para Internet do futuro e foi financiado pela rede de educação e pesquisa da Europa GÉANT. Implementado, inicialmente, por meio do controlador OpenFlow OpenDaylight (ODL) (Medved *et al.*, 2014), o DynPaC provê três funcionalidades principais: resiliência; agendamento e monitoramento de serviços; e desagregação de fluxos. O DynPaC visa aplicar técnicas de engenharia de tráfego baseadas nos conceitos de SDN para obter otimização da utilização de recursos de rede, fazendo uso

de mecanismos de agregação e desagregação de fluxos para acomodar fluxos elefante (*i.e.* tráfego de transferência de grandes volumes de dados) e camundongo (*e.g.* tráfego *web*) (Papagiannaki *et al.*, 2002) mais facilmente.

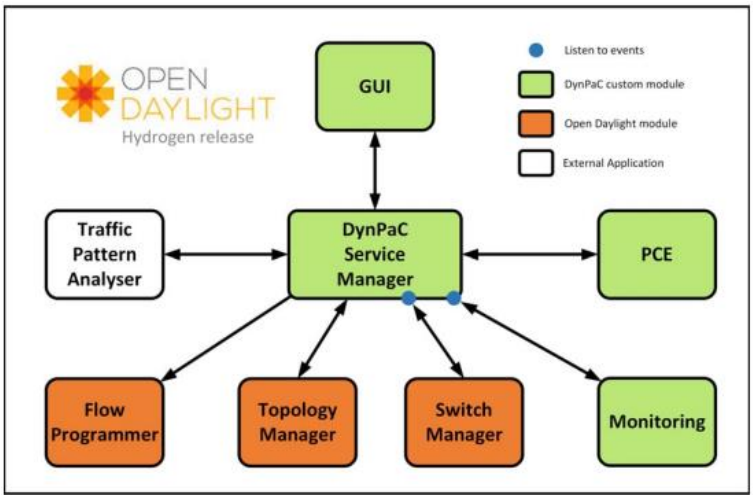


Figura 2.10: Arquitetura do arcabouço DynPaC.

A arquitetura do DynPaC é exibida na Figura 2.10, a qual fora baseada no controlador ODL. O DynPaC faz uso dos módulos *Topology Manager*, *Switch Manager* e *Flow Rules Manager* incluídos por padrão no ODL, e estende suas capacidades com quatro módulos customizados adicionais e uma interface gráfica de usuário (GUI). Esses módulos são descritos brevemente a seguir.

- DynPaC *Service Manager* (DSM): gerencia os recursos de rede consumidos durante o tempo e trata também do serviço de resiliência.
- *Path Computation Element* (PCE): realiza a computação de caminhos para os serviços requisitados, através de dois diferentes tipos de algoritmos. O primeiro deles é utilizado durante o período de inicialização do DynPaC, e computa todos os possíveis caminhos entre todos os nós de origem e destino. O segundo algoritmo é o de desagregação de serviços, o qual utiliza as informações fornecidas pelo módulo *Traffic Pattern Analyser* para desagregar os serviços já instalados na rede em um número mínimo de sub-serviços. Esses sub-serviços são, então, realocados em caminhos alternativos para permitir a liberação de recursos suficientes para possibilitar o provisionamento de novos serviços.
- *Monitoring*: monitora a banda ocupada pelos serviços rodando na rede. Esse monitoramento ocorre através das informações obtidas através das mensagens de estatísticas de fluxos do OpenFlow para computar a banda média ocupada por fluxo, garantindo assim que os serviços não estejam utilizando mais banda do que foi requisitado. Caso o serviço exceda o limite de banda pré-estabelecido, o módulo de monitoramento notifica o DSM sobre o ocorrido.
- *Graphic User Interface* (GUI): permite realizar requerimentos de novos serviços através de um formulário *web*. Também prove informações sobre os serviços já requisitados.
- *Traffic Pattern Analyser*: permite a obtenção de informações de padrões de tráfego e sua distribuição.

Recentemente conforme descrito em (Mendiola *et al.*, 2016), o DynPaC fora migrado para operar através do controlador OpenFlow ONOS (Berde *et al.*, 2014) sendo também capaz de prover serviços de *Bandwitch on Demand* (BoD) ou Banda sob Demanda com suporte a SDN/OpenFlow em topologias de rede multi-domínio por meio do protocolo *Network Services Interface-Connection Service* (NSI-CS) (Roberts *et al.*, 2013).

Entretanto, o DynPaC não provê suporte a redes híbridas. Isso acaba por se tornar um limitador no caso de sua implantação em redes em operação que visam fazer uso das vantagens provenientes da utilização dos conceitos de SDN, mas que não permitem ou não desejam operar todos os seus serviços integralmente através de SDN/OpenFlow.

2.4.2 OESS

O *Open Exchange Software Suite* (OESS) (Tepsuporn *et al.*, 2015; GlobalNOC, 2016) é um controlador SDN que gerencia os *switches* via protocolo OpenFlow. Ele é utilizado pela rede de educação e pesquisa norte-americana Internet2[10] para configuração e controle de circuitos virtuais L2 dinâmicos do tipo VLAN, através do serviço denominado *Internet2's Advanced Layer-2 Service* (AL2S) (Internet2, 2017b). Desenvolvido a partir do projeto norte-americano denominado *DYnamic NEtwork System* (DYNES)[11] (Zurawski *et al.*, 2012), o OESS provê provisionamento rápido de circuitos virtuais com tempo de estabelecimento inferior a 1 segundo, recuperação de falhas de circuitos automática, permissões de circuito por interface e estatísticas automáticas por VLAN, além de suportar a integração com o *On-Demand Secure Circuits and Advanced Reservation System* (OSCARS)[12] (Guok *et al.*, 2006; ESnet, 2016) , da *Energy Sciences Network* (ESnet)[13], para provisionamento de circuitos inter-domínios.

O mecanismo de provisionamento de circuitos no OESS opera conforme descrito em sequência. Inicialmente, um usuário que deseja estabelecer um circuito seleciona através de uma interface de usuário *web:* os *endpoints* do circuito virtual, a taxa de transferência máxima do circuito, o instante de inicialização, sua duração e opcionalmente também

[10] A Internet2 (Internet2, 2017a) é a maior rede de educação e pesquisa dos Estados Unidos da América (EUA), sendo responsável por interconectar instituições de educação, pesquisa, indústrias e governo através de um *backbone* nacional, operando em parceria com mais de 60 redes de educação e pesquisa locais.

[11] O projeto DYNES visa a implantação de uma arquitetura híbrida de redes voltada para o transporte de pacotes e circuitos de alta capacidade para tráfego de e-ciência nos EUA, por meio do fomento à aquisição de dispositivos de rede (e.g. *switches*) e de servidores para transferência de dados, e ao desenvolvimento de um controlador de redes.

[12] O OSCARS é um *arcabouço* para provisionamento de circuitos virtuais L2 com segurança e garantia de reserva de banda em redes IP tradicionais com suporte a MPLS. O *arcabouço* opera através dos protocolos *Inter-Domain Controller Protocol* (IDCP) (OGF/NSI-WG, 2010) e NSI-CS versão 2.0 (Roberts *et al.*, 2013) para provisionamento de circuitos inter-domínios e tem como principais características: autenticação, autorização, auditoria, mediação de *hardware* e coordenação de processos.

[13] A ESnet, assim como a Internet2, é uma das maiores redes de educação e pesquisa dos EUA e opera um *backbone* nacional servindo ao Escritório de Ciências do Departamento de Energia norte-americano sendo gerenciada pela equipe do Laboratório Nacional de *Lawrence Berkeley*

especifica um caminho (com um possível caminho redundante) para interconexão dos *endpoints*. Quando todos os detalhes do circuito são especificados, a interface de usuário do OESS encaminha uma requisição ao controlador SDN/OpenFlow. O controlador, então, calcula todas as mensagens de controle *FlowMod* necessárias para estabelecimento do circuito fim a fim.

Essas mensagens do tipo *FlowMod* constam basicamente da estrutura de campos de correspondência *Match* associada aos campos porta de entrada (IN_PORT) e VLAN ID (VLAN_VID). As *Actions* de cada mensagem constam da modificação do campo de cabeçalho VLAN ID, através da ação *Set Field set_vlan_id*, e do encaminhamento a uma porta de saída, por meio da ação *Output*. Em alguns casos a ação de remoção de *tag* de VLAN é também utilizada por meio da ação *Pop-Tag*, para o caso de circuitos não "tageados". Além disso, o OESS utiliza um mecanismo de descoberta de topologia através do protocolo denominado *OpenFlow Discovery Protocol* (OFDP), que opera de maneira similar ao *Link Layer Discovery Protocol* (LLDP), descrito na Seção 2.3.5. OpenFlow – Descoberta de Topologia. Esse mecanismo permite principalmente a identificação de adjacências entre dois dispositivos em portas específicas, além de permitir também a detecção de quedas de enlaces e inserção de novos nós, o que possibilita ao OESS mover automaticamente milhares de circuitos virtuais com mínima intervenção humana.

Apesar de tudo, o controlador OESS também sofre com a limitação do não suporte a redes SDN híbridas assim como no caso do *arcabouço* DynPaC, o que torna mais restritivo o seu campo de utilização. Desse modo, essa restrição é uma das maiores motivações para o desenvolvimento da solução VCFlow, descrita no próximo capítulo intitulado 3. Proposta de Solução de Provisionamento de Circuitos Virtuais em Redes SDN/OpenFlow Híbridas.

Capítulo 3

3 PROPOSTA DE SOLUÇÃO DE PROVISIONAMENTO DE CIRCUITOS VIRTUAIS EM REDES SDN/OPENFLOW HÍBRIDAS

Neste capítulo, é apresentada inicialmente uma visão geral da solução VCFlow, descrevendo de maneira geral como o VCFlow opera. Em seguida, é descrito em detalhes o processo de verificação dos *endpoints* e identificação de circuito virtual, responsável por validar se o circuito virtual pode ser estabelecido fim a fim e identificar qual ID de VLAN será utilizado para transporte pela rede de núcleo do provedor. Em sequência, é descrito o processo de escolha do menor caminho, VLAN *stitching* e estabelecimento do circuito virtual, ou seja, como o VCFlow se baseia no algoritmo *Shortest Path First* (SPF) e do ID de VLAN para transporte pela rede de núcleo para realizar o estabelecimento do circuito virtual. É descrito como ocorre o processo de desenvolvido para descoberta de topologia em redes SDN/OpenFlow híbridas, por meio de pacotes LLDP com *tag* de VLAN. Por fim, é descrito o mecanismo desenvolvido para prevenção de *loops* de encaminhamento baseado em técnica de *split horizon*.

3.1 Visão Geral da Solução VCFlow

Uma rede de *backbone* IP consiste de múltiplos roteadores do tipo *Provider Edge* (PE), que conectam o roteador de borda *Customer Edge* (CE) do cliente a rede de núcleo do provedor, e roteadores do tipo *Provider* (P), os quais funcionam como um roteador de trânsito na rede de núcleo entre os PE, conforme mostrado na Figura 3.1.

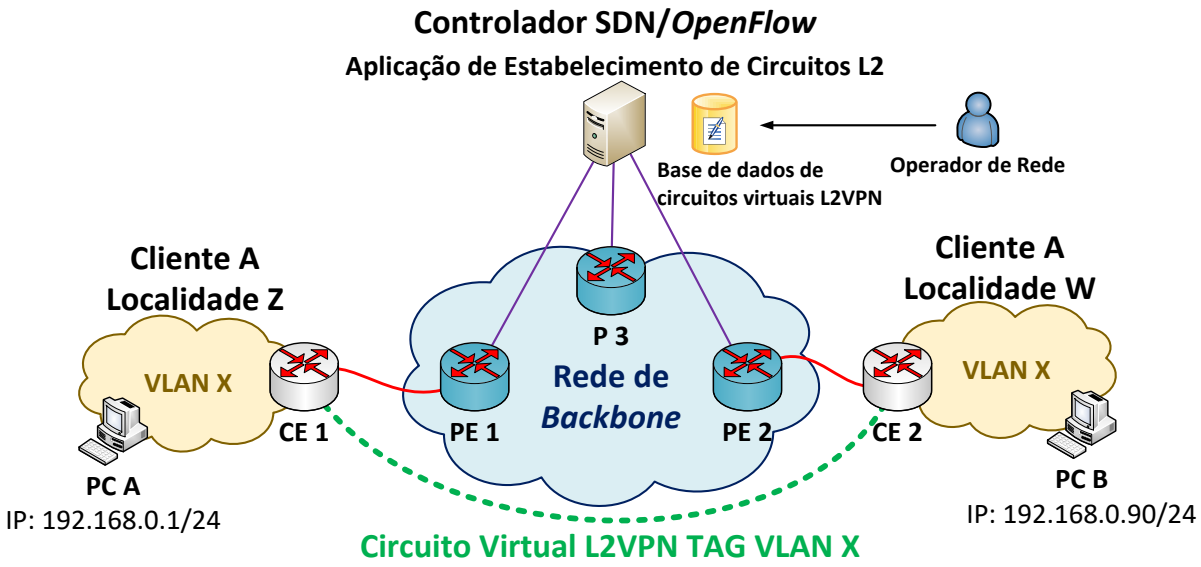


Figura 3.1: Modelo básico da solução de provisionamento de circuitos virtuais L2.

A solução aqui proposta e denominada como *Virtual Circuits Flow* (VCFlow) otimiza o mecanismo de provisionamento de circuitos virtuais através de sua simplificação, não havendo necessidade dos operadores realizarem configurações distribuídas pelos diversos elementos de rede nem havendo a necessidade de se utilizar protocolos de tunelamento. Desse modo, o VCFlow é capaz de minimizar a intervenção humana para estabelecimento de cada circuito, permitindo que os administradores de rede do cliente informem aos operadores de rede do provedor somente as *tags* de VLAN 802.1q (IEEE, 2014) utilizadas em cada localidade nas quais desejam estabelecer um circuito virtual L2VPN *Ethernet*/VLAN. Os operadores, então, somente precisam armazenar as informações do circuito (como, por exemplo, *tag* de VLAN e roteadores PE envolvidos) em uma base de dados. A solução é responsável por direcionar o tráfego de maneira adequada através da rede de núcleo, baseando-se nas informações obtidas através da base de dados e utilizando o algoritmo de escolha de menor caminho *Shortest Path First* (SPF) (Misa e Frana, 2010). O modelo básico de estabelecimento de circuitos virtuais via VCFlow pode também ser visto na Figura 3.1.

O VCFlow identifica quais os circuitos estão associados a cada *switch* PE (*endpoints* do circuito virtual) e relaciona qual o caminho (*path*) que cada fluxo deve seguir para se estabelecer o circuito. Cada *endpoint* origem insere uma *tag* identificadora do circuito no pacote recebido de cada cliente. Essa *tag* é obtida simplesmente modificando-se a *tag* original do pacote com cabeçalho VLAN por uma *tag* identificadora única no núcleo da rede para cada circuito. Então, o encaminhamento em todos os *switches* intermediários ocorre por meio dessa *tag* identificadora até alcançar o destino. No destino, essa *tag* é novamente alterada para entregar ao *switch* CE. No entanto, essa *tag* alterada pode ser a mesma recebida do *endpoint* origem ou pode ser outra qualquer. Essa característica permite que um circuito virtual seja estabelecido para uma mesma sub-rede IP distribuída em localidades distintas, mas que adota *tags* de VLAN diferentes em cada localidade atendida.

Cada *switch* OpenFlow controlado pela solução VCFlow opera também como um *learning switch.* Isto é, para cada novo pacote recebido por um *switch* PE o endereço MAC do *host* de origem é associado a interface de entrada desse *switch* PE. Como cada *endpoint* conhece apenas um único caminho (calculado pelo algoritmo SPF) para o outro *endpoint* de um mesmo circuito virtual, uma regra por circuito por *host* é instalada nas tabelas de fluxos dos *switches* do caminho. Cada uma dessas regras é identificada pelo endereço MAC do *host* origem e por uma *tag* de VLAN, sendo essa *tag* diferente dependendo do tipo de *switch* em que a regra é instalada, ou seja, se é um *switch* tipo P ou PE. Nos *switches* PE, essa *tag* de identificação de fluxo refere-se ao VLAN ID da rede do cliente associado ao *endpoint* do circuito virtual em que o *host* origem está conectado. As entradas na tabela de fluxo possuem como ação o encaminhamento pela porta calculada pelo algoritmo SPF e a modificação do VLAN ID da rede do cliente pelo VLAN ID de encaminhamento no núcleo. Nos *switches* P intermediários, a *tag* refere-se ao VLAN ID utilizado para transporte dos pacotes no núcleo. Nesse caso, as tabelas de fluxos possuem um menor número de ações em relação aos *switches* PE, pois recebem somente as ações de comutação dos *paths* associadas às *tags* de identificação do circuito no núcleo e ao endereço MAC do *host* origem, sem que haja a necessidade de nenhuma alteração nos pacotes.

No entanto, essa abordagem em que as regras de encaminhamento se baseiam também no endereço MAC de origem, traz limitações em termos de escalabilidade, pois dado que o *pipeline* OpenFlow é finito, ao passo que se aumenta o número de *hosts* em cada circuito, se diminui o número máximo de circuitos virtuais que podem ser atendidos. Porém, apesar dessa limitação, essa abordagem permite que a solução seja implementada em quaisquer *switches* OpenFlow independentemente da implementação de cada fabricante. Nesse caso, avaliações anteriores mostraram que a identificação de cada circuito somente pela sua *tag* de VLAN (sem considerar o endereço MAC de origem) não garante o encaminhamento dos pacotes de modo correto em *switches* híbridos.

Desse modo, o VCFlow permite que nenhum cabeçalho adicional seja adicionado aos pacotes de cada circuito, como no caso de tunelamento via protocolo GRE ou comutação baseada em rótulos via MPLS. Essa característica otimiza o desempenho de utilização de enlaces de rede, pois diminui o *overhead* de informações de cabeçalho. Diferentemente de redes tradicionais em que as informações de circuito são descentralizadas gravadas em cada *switch* do caminho, o VCFlow mantém as informações de modo centralizado no controlador SDN/OpenFlow, simplificando também o gerenciamento dos circuitos. Além disso, o VCFlow foi concebido para operar em redes híbridas, as quais permitem aliar as vantagens de encaminhamento baseado em entrega por melhor esforço do IP em redes tradicionais, aliado à potencialidade da programabilidade proveniente do paradigma de SDN.

Portanto, nesta seção, será dada uma descrição detalhada do funcionamento do VCFlow. Primeiramente, é descrito como o operador de rede configura sua base de dados de estabelecimento de circuitos L2VPN. Em sequência, é descrito a utilização do algoritmo de menor caminho SPF e o processo de VLAN *stitching* no núcleo da rede para o estabelecimento do circuito fim a fim. Em seguida, é descrito como ocorre o processo de descoberta de topologia da rede em redes híbridas, necessário para o cálculo de cada *path* pelo SPF. Por fim, o mecanismo de prevenção de ocorrência de *loops* baseado na técnica de *Split Horizon* é descrito.

3.2 Verificação dos *Endpoints* e Identificação de Circuito Virtual

O VCFlow se baseia nos dados armazenados em duas bases de dados:

(a) "Base de Configuração": gerenciada pelo operador de rede para armazenar a configuração do circuito virtual;
(b) "Base de Identificação": utilizada para o processo de VLAN *stitching* no núcleo da rede.

Na base de dados de configuração, cada entrada é representada por uma tupla de seis valores: porta *switch* PE 1, *switch* PE 1, VLAN ID de acesso no *switch* PE 1, porta *switch* PE 2, *switch* PE 2 e VLAN ID de acesso no *switch* PE 2.

Para lidar com as situações em que um mesmo cliente utiliza *tags* de VLANs distintas nos locais em que é atendido, e para decisão de qual *tag* é utilizada no núcleo da rede sem que haja sobreposição, outra base de dados denominada de "Base de Identificação" é utilizada. Nela, cada circuito passa a ser identificado de modo único por uma tupla de cinco valores, onde o ID de backbone (utilizado para transporte no núcleo da rede) é acrescentado a cada entrada da base de configuração: *switch* PE 1, VLAN ID de acesso no *switch* PE 1, *switch* PE 2, VLAN ID de acesso no *switch* PE 2 e ID de *backbone*.

No caso da "Base de Configuração", cada entrada de configuração é checada em sequência a cada cinco segundos em busca de modificações. Esse período foi adotado de modo a otimizar o período de tempo de estabelecimento de circuitos sem sobrecarregar o *hardware* do controlador com acessos demasiados a disco. A partir dos dados de identificação de cada circuito, seleciona-se sequencialmente em função das *tags* já em uso no núcleo, qual a próxima a ser utilizada e armazena-se essa informação na própria "Base de Identificação", conforme ilustrado na Figura 3.2.

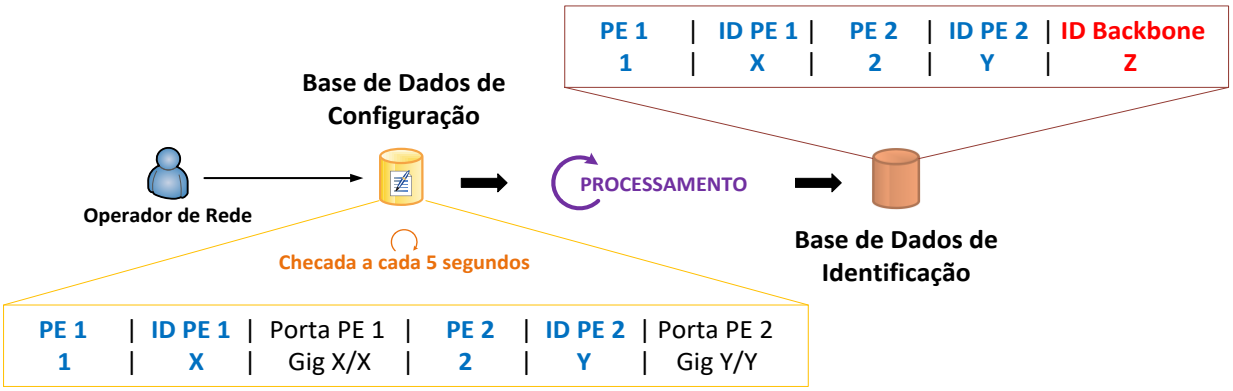


Figura 3.2: Visão geral do processo de identificação de cada circuito virtual.

O fluxograma de código simplificado do processo de verificação dos *endpoints* e identificação de circuito virtual pode ser encontrado na Figura 3.3 a seguir.

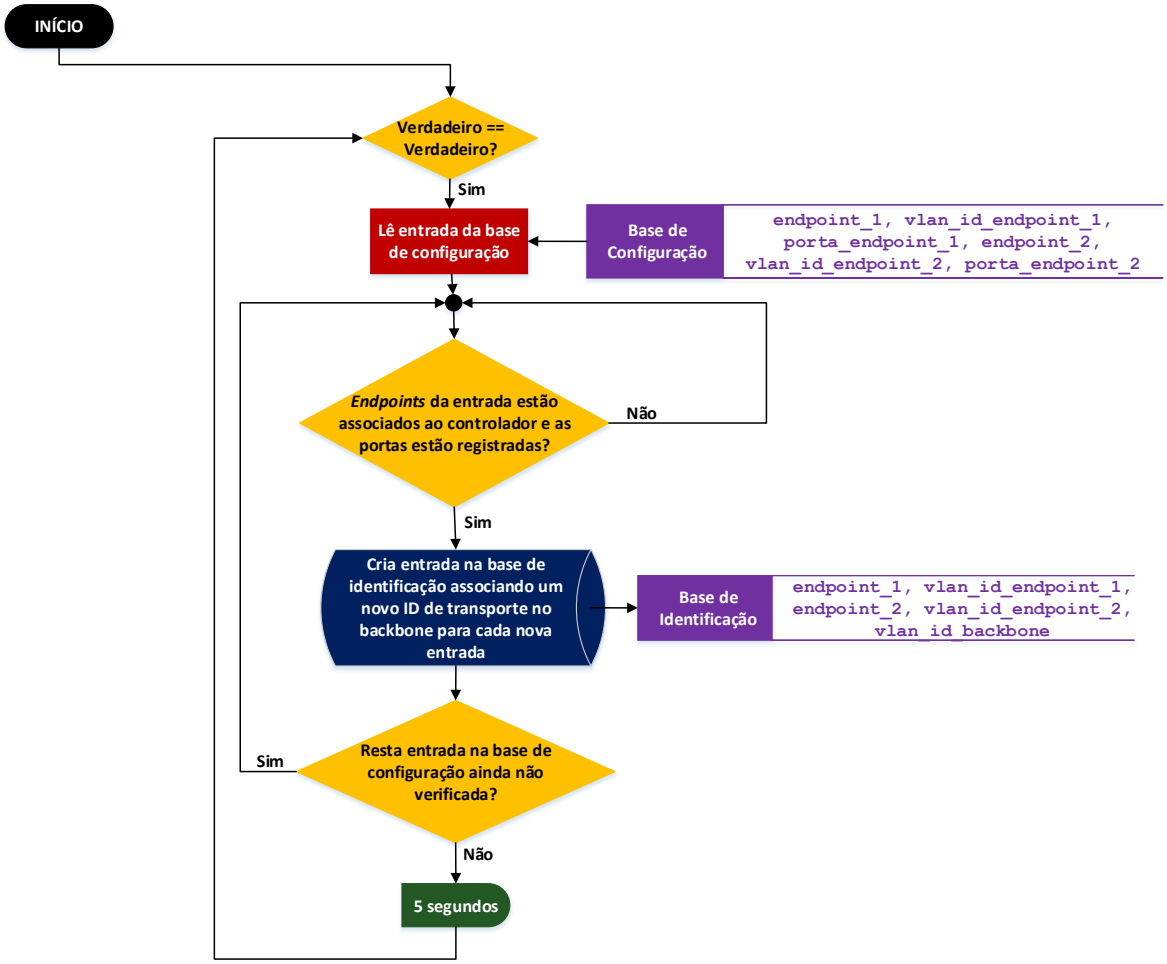


Figura 3.3: Fluxograma de código simplificado do processo de verificação de base de dados de configuração.

3.3 Escolha do Menor Caminho, VLAN *Stitching* e Estabelecimento do Circuito Virtual

A cada pacote do tipo *Packet-In* proveniente do recebimento de um pacote VLAN em um dos *endpoints* de um circuito virtual, o controlador analisa as informações dos cabeçalhos do pacote, verificando para cada entrada de configuração se:

(i) O *switch* de entrada do pacote é um dos *endpoints* de algum dos circuitos;
(ii) A porta de entrada do *switch* é associada a algum dos circuitos;
(iii) A t*ag* de VLAN do pacote é igual ao VLAN ID de entrada de algum circuito.

Caso essas informações se confirmem, o controlador configura o *switch* para ele:

(i) Associar o endereço MAC de origem do pacote ao *endpoint* de entrada do circuito;
(ii) Substituir a *tag* de entrada pela ID de *backbone* do circuito a partir do ID de *backbone* da "Base de Identificação", por meio da ação *set_field* do OpenFlow 1.3;
(iii) Encaminhar o pacote em direção ao outro *endpoint* do circuito.

Ao chegar no *endpoint* de destino, a operação inversa é realizada substituindo-se o ID de *backbone* pelo ID do circuito no *endpoint* de destino.

Caso o *switch* que recebe o pacote não seja um *endpoint*, mas a *tag* seja associada a um ID de circuito no *backbone* e o MAC de origem esteja vinculado a um dos *endpoints*, o pacote é simplesmente encaminhado para o outro *endpoint* sem realizar nenhuma modificação no pacote. Além disso, para estabelecer o caminho entre o *switch* que envia a mensagem de *Packet-In* e o *switch* PE de destino do circuito, a aplicação utiliza o algoritmo de *Shortest Path First* (Misa e Frana, 2010), descrito na Seção 9. Apêndice B – Algoritmo *Shortest Path First* Adotado pelo VCFlow para Cálculo de Menor Caminho.

Todo esse procedimento de VLAN *stitching*, ou seja, o processo de costurar o circuito virtual L2VPN/VLAN através da rede, ocorre proativamente em cada direção do circuito (nas direções de *endpoint* 1 para *endpoint* 2 e de *endpoint* 2 para *endpoint* 1). Isto é, as ações de encaminhamento nos *switches* do caminho de cada circuito entre *endpoint* origem e *endpoint* destino são instaladas de uma única vez a partir do *Packet-In* proveniente do *endpoint* de origem. Nesse caso, o fluxo é encaminhado à porta de saída de cada *switch*, onde a informação referente a essa porta é obtida em função do algoritmo SPF em conjunto com as bases de dados de configuração e de identificação de circuitos. Por consequência, somente são gerados eventos de *Packet-In* ao entrarem pacotes não classificados nos *endpoints* dos circuitos. Dessa forma, é possível otimizar o processo de estabelecimento de circuitos fim a fim, minimizando o número de pacotes enviados dos *switches* ao controlador. Essa abordagem proativa está de acordo com estudos em literatura, que indicam melhor desempenho dos controladores OpenFlow ao se utilizar esse tipo de abordagem em relação a uma abordagem integralmente reativa (Fernandez, 2013). Além de tudo, para evitar que ocorram *loops* na topologia utiliza-se também um mecanismo baseado na técnica de *Split Horizon*, conforme descrito na Seção 3.5. Prevenção de *Loops* de Encaminhamento Utilizando Técnica de *Split Horizon*

O VCFlow também instala regras com *soft-timeout* de 60 segundos para poder lidar com o processo de alteração da configuração de um circuito. Então, no caso de mudança de configuração de um mesmo circuito por parte do operador de rede, para que ocorra o seu estabelecimento há a necessidade de se aguardar: (a) um período de 60 segundos de inatividade para que a entrada da tabela de fluxo de cada *switch* no caminho seja deletada; (b) um período de 5 segundos para que a configuração possa ser verificada novamente e o novo circuito possa ser estabelecido a partir de novas mensagens de *Packet-In* - podendo ambos os intervalos (a) e (b) ocorrerem concorrentemente.

A Figura 3.4 exibe uma visão geral e a **Erro! Fonte de referência não encontrada.** exibe um fluxograma de código simplificado do mecanismo de VLAN *stitching* e estabelecimento de circuitos virtuais fim a fim em função das bases de dados de configuração e de identificação.

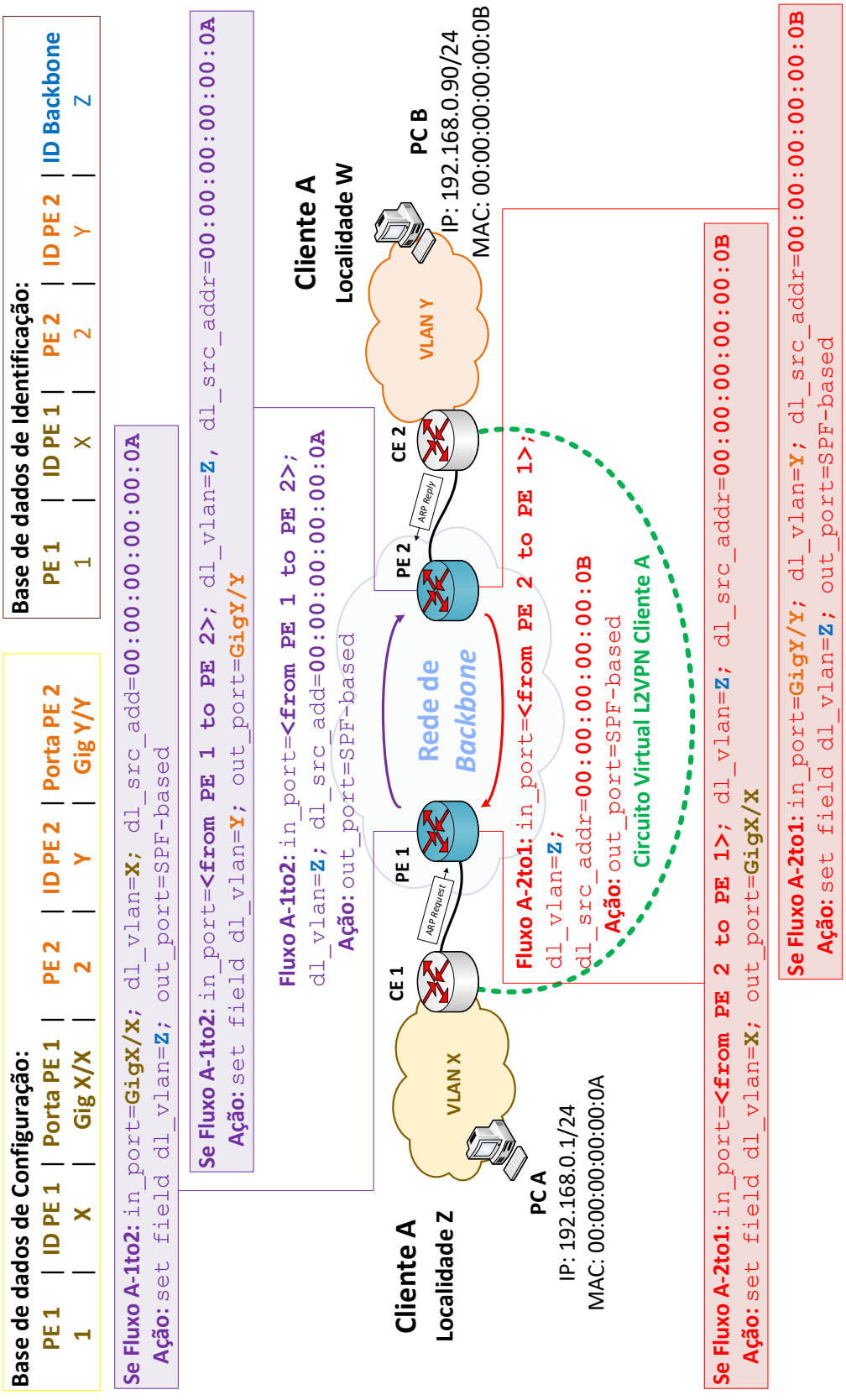


Figura 3.4: Modelo básico do funcionamento da solução VCFlow para o processo de VLAN stitching e estabelecimento de circuitos.

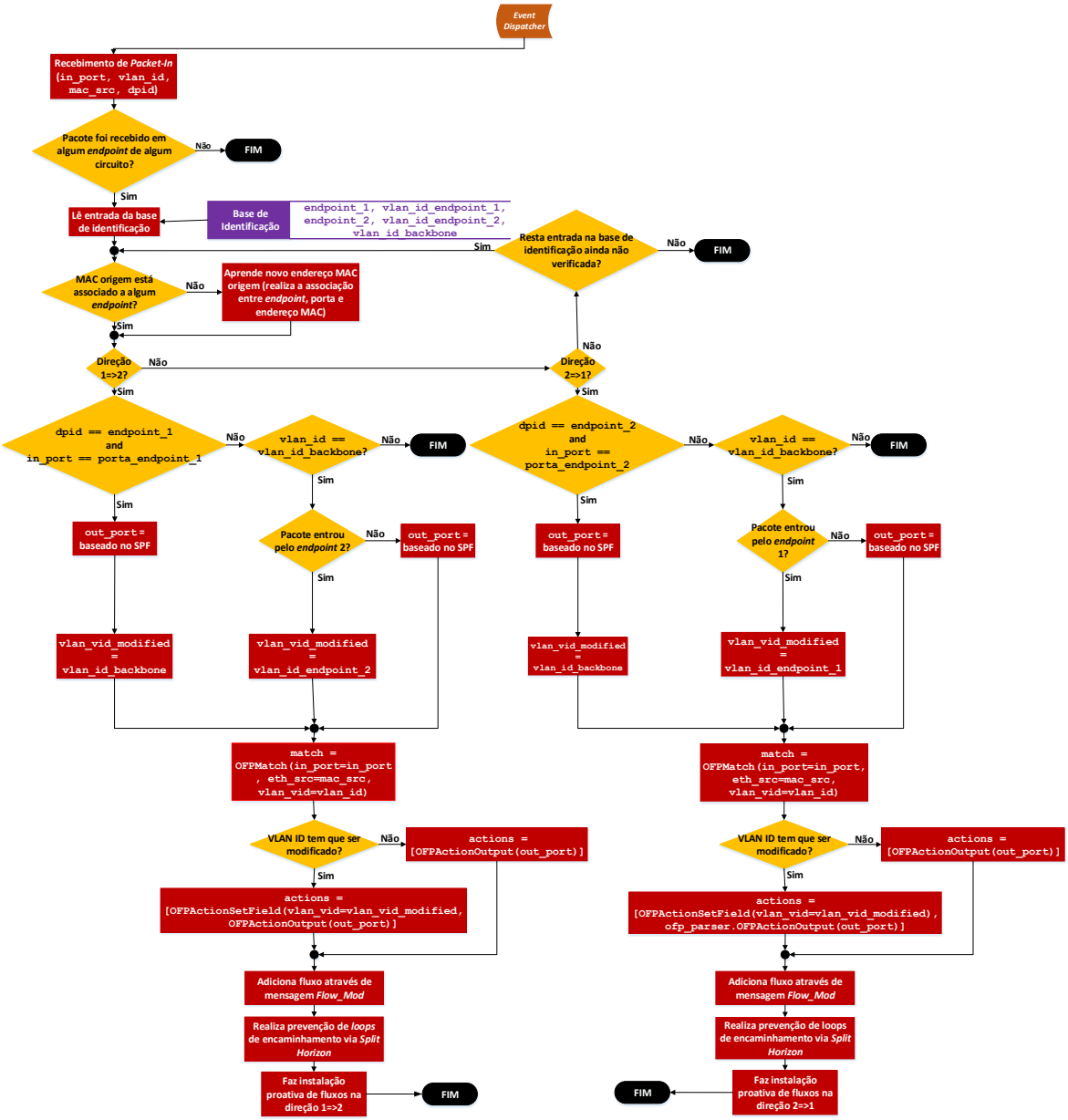


Figura 3.5: Fluxograma de código simplificado do processo de VLAN stitching e estabelecimento de circuito virtual fim a fim.

3.4 Descoberta de Topologia em Redes SDN/OpenFlow Híbridas

O VCFlow utiliza como característica principal o fato de que em uma rede definida por *software* o controlador tem uma visão holística da rede. Isto é, o controlador tem conhecimento de todos os nós da rede e suas ligações (para efeito de cálculo dos caminhos de encaminhamento de cada circuito). Então, o processo de descoberta de topologia em uma rede híbrida é essencial, haja visto que o processo de encaminhamento tradicional de tráfego não "tageado" e o processo de encaminhamento de tráfego "tageado" baseado em SDN/OpenFlow compartilham os mesmos enlaces físicos.

Tendo isso em vista, o VCFlow faz uso do mecanismo de descoberta de topologia baseado no *Link Layer Discovery Protocol* (LLDP), conforme descrito na Seção 2.3.5. OpenFlow – Descoberta de Topologia, porém com uma pequena modificação: o tráfego LLDP é encaminhado também com uma *tag* de VLAN. Nesse caso, os pacotes LLDP são encaminhados em interfaces virtuais dedicadas ao encaminhamento de quadros *Ethernet* com cabeçalho VLAN 802.1q e campo de prioridade *Priority Code Point* (PCP) para controle de rede. Essas interfaces são associadas ao plano de dados OpenFlow dos *switches* híbridos. Assim sendo, os pacotes LLDP são encaminhados em quadros *Ethernet* com cabeçalho VLAN, e campos VLAN *Identifier* (VID) igual a zero[14] e PCP igual a sete[15].

Já o tráfego para o estabelecimento dos circuitos virtuais é encaminhado em interfaces virtuais associadas também ao plano de dados OpenFlow, porém dedicadas a quadros *Ethernet* com cabeçalho VLAN 802.1q e campo VLAN *Identifier* (VID) variando de dois até 4094. Os quadros *Ethernet* sem cabeçalho VLAN são encaminhados pelas interfaces físicas padrão do *switch*, associadas a seu plano de dados tradicional. A Figura 3.6 ilustra como o tráfego é encaminhado em uma topologia de rede híbrida com o VCFlow, através da separação dos planos de encaminhamento OpenFlow e tradicional nos *switches* híbridos.

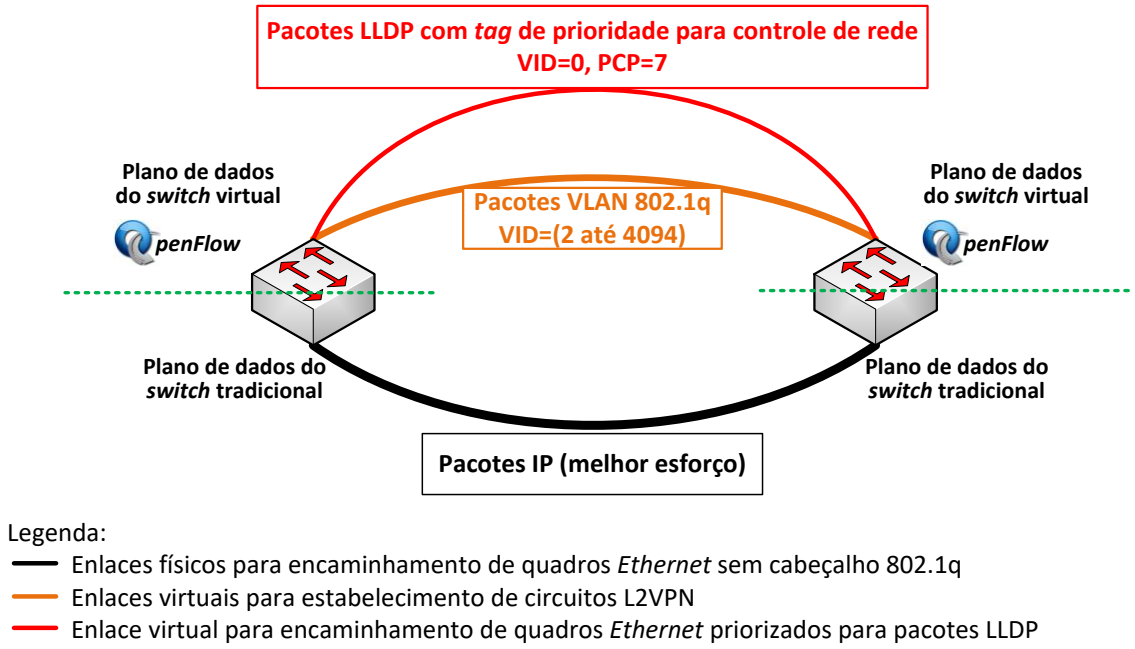


Figura 3.6: Mecanismo de encaminhamento de pacotes em uma rede híbrida com o VCFlow.

[14] O VID é um campo de 12 bits (de 0 a 4095) do cabeçalho VLAN, que identifica a VLAN a qual o quadro *Ethernet* pertence. Os valores 0 e 4095 são reservados e todos os outros 4094 valores podem ser utilizados como identificadores de VLAN. O VID 0 (zero) indica que o quadro não carrega um identificador de VLAN, mas especifica somente uma prioridade. O VID 1 (um) é em geral reservado para utilização como VLAN de gerenciamento, porém a sua implementação depende do fabricante.

[15] O PCP é um campo de prioridade de 3 *bits* (de 0 a 7) do cabeçalho VLAN e é responsável por mapear o nível de prioridade dos quadros *Ethernet*. O nível sete é utilizado para controle de rede.

Além de tudo, esse mecanismo de encaminhamento baseado em múltiplas interfaces virtuais com funções específicas para cada uma delas permite também que sejam aplicados métodos de monitoramento, filtragem ou espelhamento de tráfego de modo independente em relação ao tipo de tráfego. Isto é, cada tipo de tráfego pode ser filtrado por interface específica associada a um tipo de tráfego, simplificando o *troubleshooting* em caso de problemas.

3.5 Prevenção de *Loops* de Encaminhamento Utilizando Técnica de *Split Horizon*

O *Split Horizon* é um método de prevenção de *loops* de roteamento em uma rede. O seu princípio básico de funcionamento é simples: as informações de atualização de uma rota nunca são enviadas na direção em que elas são recebidas.

Em geral, essa técnica é utilizada para evitar *loops* em redes que adotam protocolos de roteamento do tipo vetor distância, como é o caso do *Routing Information Protocol* (RIP), e que podem sofrer com o problema de contagem ao infinito (Peterson e Davie, 2012b). Entretanto, o mesmo princípio pode ser adotado em redes voltadas para encaminhamento de pacotes L2. Nesse caso, todo o tráfego recebido em uma interface não é encaminhado de volta nessa mesma interface onde o tráfego foi recebido.

O VCFlow adota uma técnica de prevenção de *loops* de encaminhamento similar ao *Split Horizon*, porém com algumas características agregadas. Essa técnica adotada faz com que um fluxo recebido em uma porta não seja encaminhado de volta por essa mesma porta. Esse fluxo é somente encaminhado pela porta calculada a partir do protocolo SPF e para todas as outras portas os pacotes desse fluxo são descartados. Apesar de ser o próprio VCFlow responsável por controlar o caminho dos pacotes de cada *host* de cada circuito virtual, adota-se o mecanismo de prevenção de *loop*s, pois assume-se que a rede não é integralmente OpenFlow.

A Figura 3.7 ilustra um exemplo em que a técnica de *Split Horizon* desenvolvida para o VCFlow é adotada. Nesse exemplo, uma rede em topologia *full-mesh* com quatro *switches* PE OpenFlow A, B, C e D, em que dois *switches* CE X e Y, ambos configurados com *tag* de VLAN H, são conectados aos *switches* B e D, respectivamente. Um circuito virtual visa ser provisionado entre os *switches* X e Y. Então, na ocorrência de um evento de *Packet-In* gerado pelo *endpoint* B (*switch* PE B) do circuito virtual, o fluxo identificado pela tupla `<in_port=1, dl_vlan=H e dl_src_addr=00:00:00:00:00:0B>` é encaminhado pela `out_port=2` e o VLAN ID é também alterado para transporte no núcleo da rede através da ação `set_field dl_vlan=2`. Para garantir que não ocorram *loops* os fluxos identificados pelas tuplas `<in_port=2, dl_vlan=2 e dl_src_addr=00:00:00:00:00:0B>`, `<in_port=3, dl_vlan=2 e dl_src_addr=00:00:00:00:00:0B>` e `<in_port=4, dl_vlan=2 e dl_src_addr=00:00:00:00:00:0B>` (ou seja, os pacotes que são encaminhados pela porta dois do *switch* B, mas que podem ser recebidos por quaisquer outras portas na ocorrência de *loops*) são descartados. Dessa forma, garante-se que os pacotes sejam imediatamente descartados caso ocorra um *loop*, caso em que os pacotes encaminhados pelo *switch* B são recebidos novamente nesse mesmo *switch*. O mesmo processo ocorre para os outros *switches* que fazem parte do caminho do circuito virtual, nesse caso o *switch* D. No

caminho inverso, ou seja, na ocorrência de um evento de *Packet-In* gerado pelo *endpoint* D (*switch* PE D), o mesmo procedimento também ocorre. A Figura 3.7 exibe a topologia descrita acima e todas as regras instaladas nos *switches* para o estabelecimento do circuito virtual e para o mecanismo de *split horizon* desenvolvido para o VCFlow e ilustrado nesse exemplo.

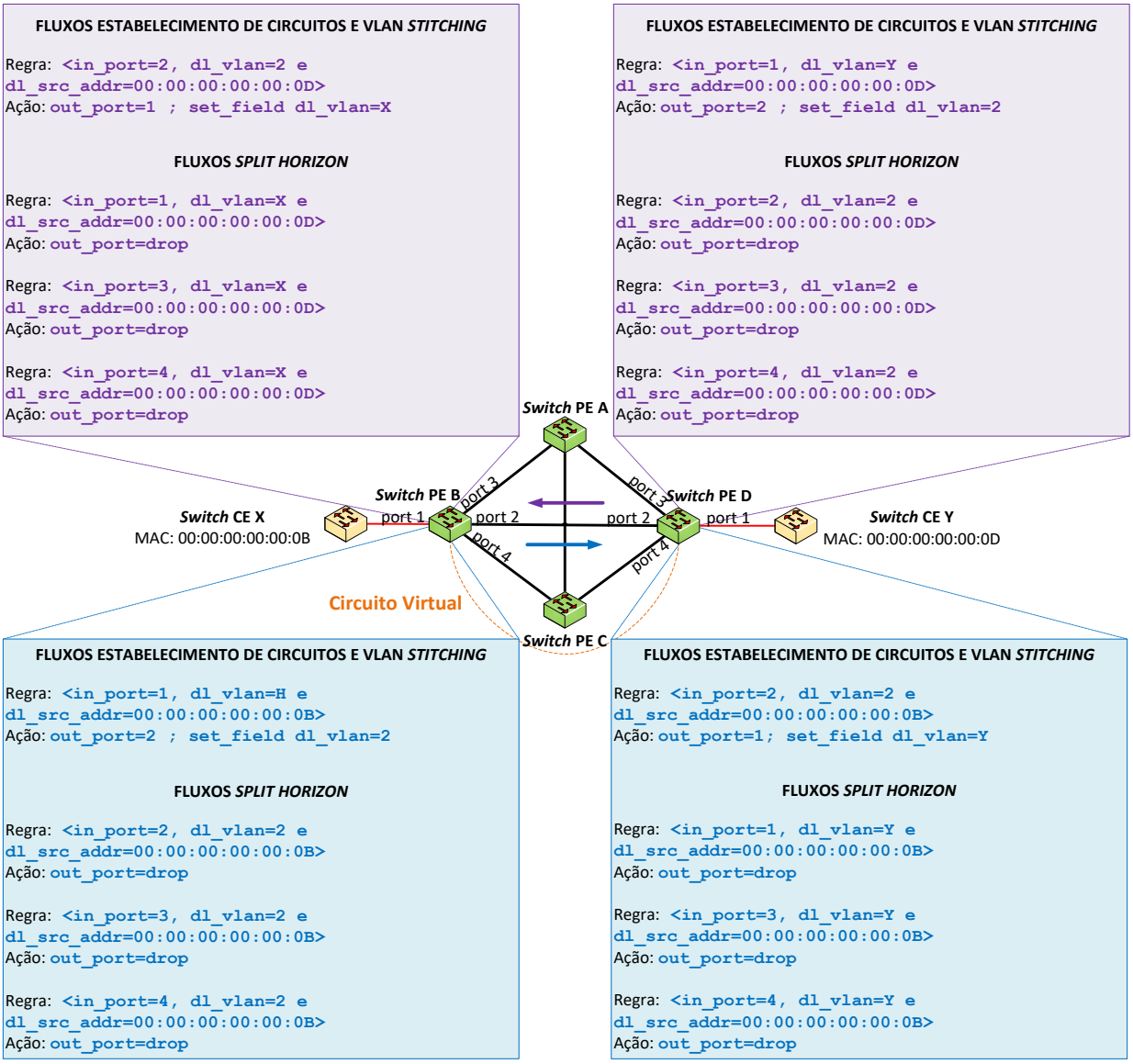


Figura 3.7: Exemplo de operação do mecanismo de Split Horizon para prevenção de loops pela solução VCFlow.

Capítulo 4

4 CARACTERIZAÇÃO DO VCFLOW

No presente capítulo, será descrito a metodologia para caracterização da solução VCFlow. O VCFlow foi caracterizado em quatro etapas distintas, cujos objetivos são descritos sumariamente a seguir:

I. Avaliação de controladores OpenFlow.
- **Objetivo**: definir qual o controlador mais adequado para desenvolvimento do protótipo da solução VCFlow por meio da análise de desempenho, de características e funcionalidades.

II. Demonstração do sistema protótipo desenvolvido em operação em rede SDN/OpenFlow híbrida real em produção.
- **Objetivo**: confirmar elegibilidade, viabilidade e funcionalidade da solução VCFlow para operar em uma topologia de rede SDN/OpenFlow híbrida real.

III. Caracterização dos intervalos de tempo de convergência para cenários reais de operação de rede.
- **Objetivo**: determinar o intervalo de tempo máximo fim a fim para o início do encaminhamento de pacotes entre os *endpoints* de um circuito virtual em dois cenários distintos de operação de rede – (i) Início do Funcionamento da Aplicação e (ii) Recuperação em Caso de Falhas.

IV. Caracterização dos intervalos de tempo de processamento das mensagens de sinalização e controle do protocolo OpenFlow, e de estabelecimento de circuitos virtuais fim a fim.
- **Objetivo**: determinar o desempenho do algoritmo de estabelecimento de circuitos e VLAN *stitching* do sistema protótipo.

Desse modo, neste capítulo é descrito primeiramente a avaliação da capacidade de diversos controladores OpenFlow, objetivando a escolha do mais adequado para o desenvolvimento da solução. Em sequência, são descritos os procedimentos e a plataforma de teste para demonstração do funcionamento da solução. Posteriormente, a metodologia adotada para avaliação do intervalo de tempo de convergência da solução é descrita. Por fim,

a metodologia adotada para avaliação do intervalo de processamento das mensagens de sinalização do protocolo OpenFlow e de tempo de estabelecimento de circuitos virtuais em diversos cenários distintos é definida.

4.1 Avaliação de Controladores OpenFlow

Como os controladores SDN/OpenFlow não são definidos por padrão pelo protocolo, o seu desempenho depende inteiramente de sua implementação específica. Como consequência, alguns controladores são mais adequados para certos tipos de tarefas (Jarschel *et al.*, 2014). Desse modo, visando identificar qual a opção dentre os muitos controladores SDN/OpenFlow seria a mais adequada para o desenvolvimento do VCFlow, a capacidade de alguns desses controladores OpenFlow foi avaliada. Essa avaliação foi realizada por meio de uma análise comparativa de desempenho e de uma pequena análise de algumas características e funcionalidades de cada um dos controladores avaliados.

Atualmente, diversos controladores SDN são disponibilizados tanto pela comunidade científica (como Maestro e Beacon**)** quanto pela indústria (como OpenDaylight e ONOS). Uma lista de diversos controladores SDN pode ser visualizada na Tabela 4.1, que é uma versão modificada da Tabela I apresentada em (Khattak *et al.*, 2014).

Tabela 4.1: Controladores OpenFlow. Adaptada de (Khattak et al., 2014).

Controlador	Linguagem de Programação	Desenvolvido por
Nox	C++	*Nicira Networks*
Maestro	Java	Universidade de Stanford – EUA
Beacon	Java	Universidade Rice – EUA
Floodlight	Java	*Big Switch Networks*
Trema	Ruby e C	NEC
Node.Flow	JavaScript	DreamersLab
OpenDaylight	Java	Cisco e OpenDaylight
Ryu	Python	*Nippon Telegraph and Telephone Corporation*
Open Networking Operating System (ONOS)	Java	*Open Networking Laboratory*

Diversos estudos foram realizados buscando apresentar análises comparativas entre vários controladores OpenFlow, como em (Shah *et al.*, 2013), (Tootoonchian *et al.*, 2012), (Fernandez, 2013) e (Khondoker *et al.*, 2014). Entretanto, nenhum desses estudos se propõe a avaliar em uma mesma análise comparativa de desempenho os controladores ODL e ONOS, utilizados pela arcabouço de estabelecimento de circuitos virtuais DynPaC, conforme descrito em (Mendiola, Astorga, *et al.*, 2015; Mendiola, Urtasun, *et al.*, 2015) e (Mendiola *et al.*, 2016), respectivamente. O arcabouço DynPaC fora desenvolvido para utilização na rede de educação e pesquisa da Europa, denominada GÉANT, e é também apresentado na Seção 2.4.1. DynPaC. Além disso, visto que um dos objetivos da solução descrita nesse trabalho é buscar a otimização do intervalo de tempo de convergência da aplicação, e Metter demonstra em (Metter *et al.*, 2015) que o controlador Ryu apresenta um

ótimo desempenho em relação ao intervalo de tempo de processamento de mensagens de sinalização, optou-se por avaliá-lo. Apresenta-se também as análises em relação ao controlador Floodlight, visto que foi o primeiro controlador SDN de código aberto a ganhar atenção em pesquisa e na indústria segundo (Berde *et al.*, 2014), e por possuir diversas análises em literatura.

Assim sendo, o desempenho de um controlador OpenFlow é, em geral, medido como o número de eventos de mensagens *Packet-In* que um controlador pode processar e responder por segundo e o intervalo de tempo médio que ele leva para processar cada evento. Atualmente, o padrão para avaliação de desempenho de controladores OpenFlow é a aplicação *Cbench* (Sherwood e Kok-Kiong, 2010)**.** O *Cbench* permite que testes de vazão e latência sejam executados, visto que é capaz de emular diversos *switches* virtuais OpenFlow (v1.0) localmente, gerar mensagens de *Packet-In* destinadas ao controlador com campos pré-determinados e contabilizar as respostas de mensagens *Packet-Out* e/ou *Flow-Mod* que são recebidas. Os testes de vazão visam avaliar quantas mensagens de controle o controlador OpenFlow é capaz de processar por segundo, enquanto que os testes de latência visam avaliar em quanto tempo que cada mensagem de controle é processada e respondida. Dessa forma, optou-se por utilizar o *Cbench* como ferramenta para análise de desempenho dos controladores OpenFlow.

Cabe ressaltar que uma das características principais da aplicação VCFlow é a rápida convergência [16] . Então, o intervalo de tempo de processamento das mensagens de sinalização, também conhecido como latência do controlador, é tido aqui como o principal parâmetro de desempenho avaliado.

Entretanto, a análise de desempenho por si só não é suficiente para a escolha do controlador que atende as necessidades de cada situação de uso. Desse modo, além da análise de desempenho foi realizada uma pequena análise comparativa que levou em consideração as características e funcionalidades dos controladores SDN/OpenFlow avaliados. Como exemplos dos tópicos avaliados em termos de características pode-se citar: (i) arquitetura do sistema do controlador; e (ii) situações de uso para os quais cada controlador foi desenvolvido. Em termos de funcionalidades pode-se destacar: (i) suporte a Interface Gráfica de Usuário (GUI); (ii) suporte a API REST; e (iii) produtividade dos controladores em termos de velocidade de se desenvolver códigos, baseado no suporte a linguagens de programação e disponibilidade de bibliotecas de código.

A seguir, é descrito a plataforma utilizada para avaliação de desempenho e como os testes de vazão e latência foram executados.

4.1.1 Plataforma de Experimentação

Tendo em vista que a solução VCFlow apresentada nesse trabalho visa ser implementada em um controlador operando em um sistema virtualizado, optou-se por executar os experimentos apresentados nessa seção na mesma plataforma de virtualização

[16] O termo convergência de rede é defino na Seção 4.3. Caracterização do Intervalo de Tempo de Convergência .

em que a solução visa ser colocada em operação. Essa plataforma consta da seguinte configuração:

- Processador: Intel Xeon E5-2430 2.20 GHz
- Memória: 64 GB
- Armazenamento: 4TB Huawei OceanStor S220T (7200 RPM NL SAS)
- Sistema Operacional: VMware vSphere 5 Essentials Plus
- Sistema de Arquivos: VMware *Virtual Machine File System* (VMFS)

Os experimentos foram executados em uma configuração conforme apresentada na Figura 4.1, em que foi utilizado uma máquina virtual dedicado para execução do controlador e outra máquina virtual dedicada a execução da ferramenta de análise *Cbench*. Ambas as máquinas foram configuradas a 1Gbps compartilhando a mesma sub-rede IP. As configurações de cada uma delas podem ser vistas na Tabela 4.1.

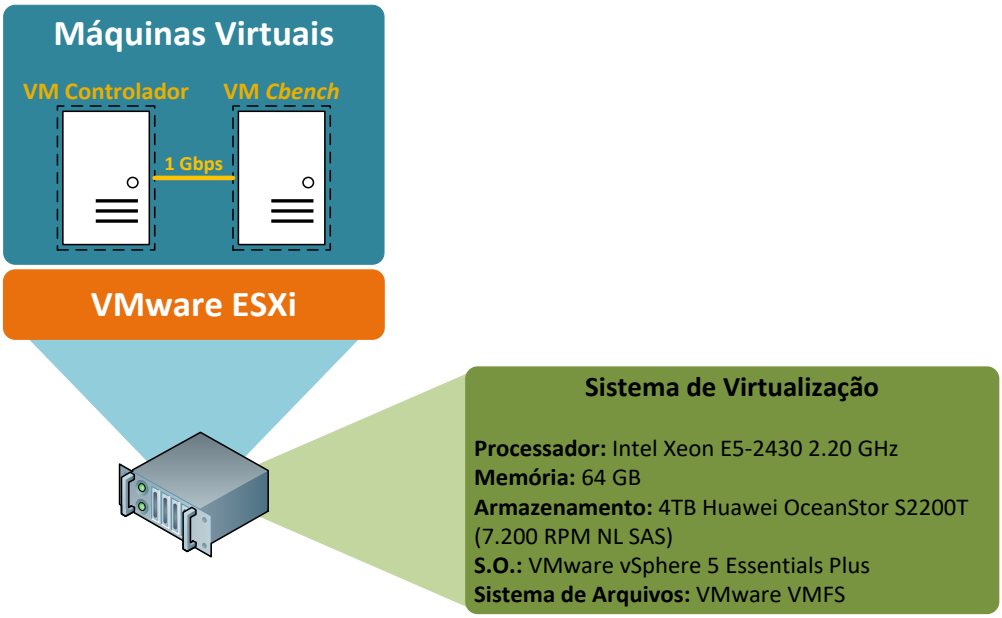


Figura 4.1: Visão geral do cenário para avaliação comparativa de controladores SDN/OpenFlow.

Tabela 4. 1: Configurações das máquinas controladora OpenFlow e de benchmarking Cbench para análise comparativa de controladores.

Característica	Controlador	Cbench
Processador	Intel(R) Xeon(R) CPU E5-2430 0 @ 2.20GHz (2 CPU x 6 *cores*)	Intel(R) Xeon(R) CPU E5-2430 0 @ 2.20GHz (1 CPU x 2 *cores*)
Memória	4GB	4GB
Armazenamento	19GB	6GB
Sistema Operacional	Ubuntu 14.04.3 LTS	Ubuntu 14.04.3 LTS

Os controladores foram executados com a aplicação de *learning switch* tradicional. Essa aplicação realiza o mapeamento de endereços MAC, associando a porta em que o pacote é recebido no *switch* OpenFlow ao endereço MAC do *host* de origem do pacote. Então, para cada mensagem de *Packet-In*, o *switch* encontra a porta de saída baseado no endereço MAC de destino e realiza o encaminhamento do pacote nessa porta.

As versões de cada um dos controladores analisados e as referências em literatura são disponibilizadas a seguir:

- Ryu: versão 4.6 (Ryu, 2016)
- Floodlight: versão 1.2 (Floodlight Project, 2016)
- OpenDaylight: versão *Beryllium* SR1 (Medved *et al.*, 2014)
- ONOS: versão 1.3.0 *Drake* (Berde *et al.*, 2014)

4.1.2 Testes de Vazão de Mensagens de Controle

No campo da física mais precisamente na área de mecânica dos fluidos, a vazão é definida como a relação entre o volume e o tempo. A vazão pode ser determinada a partir do escoamento de um fluido através de determinada seção transversal de um conduto livre (tubulação aberta) ou de um conduto fechado (tubulação com pressão). Isto é, a vazão representa a rapidez com a qual um volume escoa. Na área de sistemas de informação e redes de computadores, a vazão pode ser definida como o número de *bits* que podem ser transmitidos em uma transferência de informação em certo intervalo de tempo. Usualmente, o termo vazão é utilizado para tratar do desempenho medido de um sistema, ou seja, o número de *bits* por segundo que se pode transmitir em um enlace de rede (Peterson e Davie, 2012a).

Em se tratando da avaliação de desempenho de controladores OpenFlow, nos testes de vazão os controladores são avaliados por quantas mensagens *Packet-In* eles podem processar em um segundo. Nesse tipo de análise, a ferramenta *Cbench* envia uma grande quantidade de tráfego de controle e checa o número de respostas enviadas pelo controlador.

Nos experimentos realizados, cujos resultados serão descritos adiante, os controladores foram avaliados com n diferentes números de *switches* virtuais OpenFlow emulados pelo *Cbench*, onde $n = 1, 2, 4, 6, \ldots, 20$ *switches*. Cada *switch* possuiu associado a ele 100 MACs virtuais de *hosts*. O *Cbench* foi configurado para executar 70 testes por rodada, cada teste durando 1 segundo, ou seja, cada rodada durou 70 segundos. A média das últimas 60 respostas foi calculada para produzir o resultado final[17]. Para obter resultados estatisticamente significativos cada rodada de testes foi repetida 30 vezes.

O comando utilizado para os testes de vazão pode ser verificado a seguir:

[17] Os primeiros 10 testes de cada rodada foram considerados como um período de esquenta para que as linguagens baseadas em VM (*e.g. Java Virtual Machine* – JVM) em que alguns controladores são desenvolvidos, pudessem realizar qualquer otimização adaptativa necessária a seu pleno funcionamento. Esses 10 primeiros resultados foram desconsiderados do cálculo do final.

```
> cbench -c <IP_controlador> -p <porta_controlador> -l 70 -m 1000 -s
<numero_de_switches_virtuais> -t -w 10 -M 100
```

4.1.3 Testes de Latência de Mensagens de Controle

A latência corresponde ao intervalo de tempo que uma mensagem leva para ser transportada de um ponto a outro de uma rede. A avaliação da latência pode ter como foco a medição do intervalo de tempo de entrega de mensagens em diversos cenários distintos. Esses cenários podem ser um enlace simples, um canal fim a fim, em um único sentido (p.ex. origem para destino) ou até mesmo no caso de uma mensagem ser enviada ao destino e uma resposta ser entregue de volta a origem.

No caso de avaliação de desempenho dos controladores OpenFlow, os testes de latência têm por objetivo caracterizar o intervalo de tempo de processamento de um único pacote pelo controlador. Isto é, esses testes permitem checar a capacidade que o controlador tem de processar as mensagens *Packet-In* o mais rápido possível de modo a responder aos *switches*.

Para medir a latência do controlador, o *Cbench* foi executado em modo latência. Ou seja, o *switch* virtual encaminha uma mensagem de *Packet-In* ao controlador e aguarda por uma resposta do controlador antes de encaminhar um novo *Packet-In*. O número total de respostas/segundo é então contabilizado para medir a latência média por pacote do controlador. Assim sendo, dado que o controlador responde a $x\ mensagens\ por\ segundo\ por\ switch$, então a latência média é aqui definida como $\frac{1}{x} segundos\ por\ mensagem\ por\ switch$.

Nas avaliações de latência, cujos resultados serão descritos posteriormente, os controladores foram avaliados aumentando-se progressivamente a densidade de *switches* virtuais OpenFlow. Assim como nos testes de vazão foram utilizados de 1 a 20 *switches* com 100 MACs virtuais de *hosts* associados a cada um. O *Cbench* também foi configurado para executar 70 testes por rodada, cada teste durando 1 segundo, totalizando 70 segundos por rodada. A média das últimas 60 respostas foi calculada para produzir o resultado final. Para obter resultados estatisticamente significativos cada rodada de testes foi também repetida 30 vezes.

O comando utilizado para os testes de latência pode ser verificado a seguir:

```
> cbench -c <IP_controlador> -p <porta_controlador> -l 70 -m 1000 -
s <numero_de_switches_virtuais> -w 10 -M 100
```

4.2 Demonstração do Sistema Protótipo em Operação

Buscando confirmar a elegibilidade, viabilidade e funcionalidade da solução VCFlow foram realizadas diversas medições em cenários emulados e reais. Inicialmente, buscou-se apresentar uma prova de conceito da solução após a decisão pelo controlador a ser utilizado no sistema protótipo. Desse modo, foi realizada a medição da vazão e taxa de perdas de transferências de arquivos executadas através do *backbone* da Redecomep-Rio.

Assim foi possível verificar o mecanismo de recuperação de falhas em operação sobre uma rede de *backbone* SDN/OpenFlow híbrida real em produção.

O cenário de demonstração consistiu de nove roteadores híbridos (encaminhamento tradicional + OpenFlow) espalhados por cada um dos nove PoPs da Redecomep-Rio a citar: CBPF, RNP, Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ), Empresa Municipal de Informática da Cidade do Rio de Janeiro (IplanRio), Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (Cefet-RJ), Universidade do Estado do Rio de Janeiro (UERJ), Fundação Oswaldo Cruz (Fiocruz), Universidade Federal do Rio de Janeiro (UFRJ) e Universidade Federal do Estado do Rio de Janeiro (SIX/Unirio). Como pode ser visto na Figura 4.2, foram utilizados também dois *switches* CE para a demonstração, assim como os equipamentos que são usados em instituições clientes na Redecomep-Rio[18], sendo um deles conectado ao PoP-CBPF e outro conectado ao PoP-UERJ. Esses dispositivos são: (i). PoP-UERJ: 1 x Cisco Catalyst WS-C3560G-24TS rodando IOS 12.2(50)SE5; (ii). PoP-CBPF: 1 x Cisco Catalyst WS-C3750G-24TS-1U rodando 12.2(25)SEE2. Conectados a cada um dos *switches* CE utilizou-se servidores modelo *Raspberry* Pi 2 Model B para geração de tráfego de medição. Cabe ressaltar que a associação entre os roteadores Cisco ASR9000 e o controlador SDN/OpenFlow ocorreu *in-band*, sendo o tráfego de controle encaminhado pelo *pipeline* tradicional.

[18] Cabe salientar que nenhum dos dois *switches* utilizados na demonstração apresenta qualquer *feature* de OpenFlow.

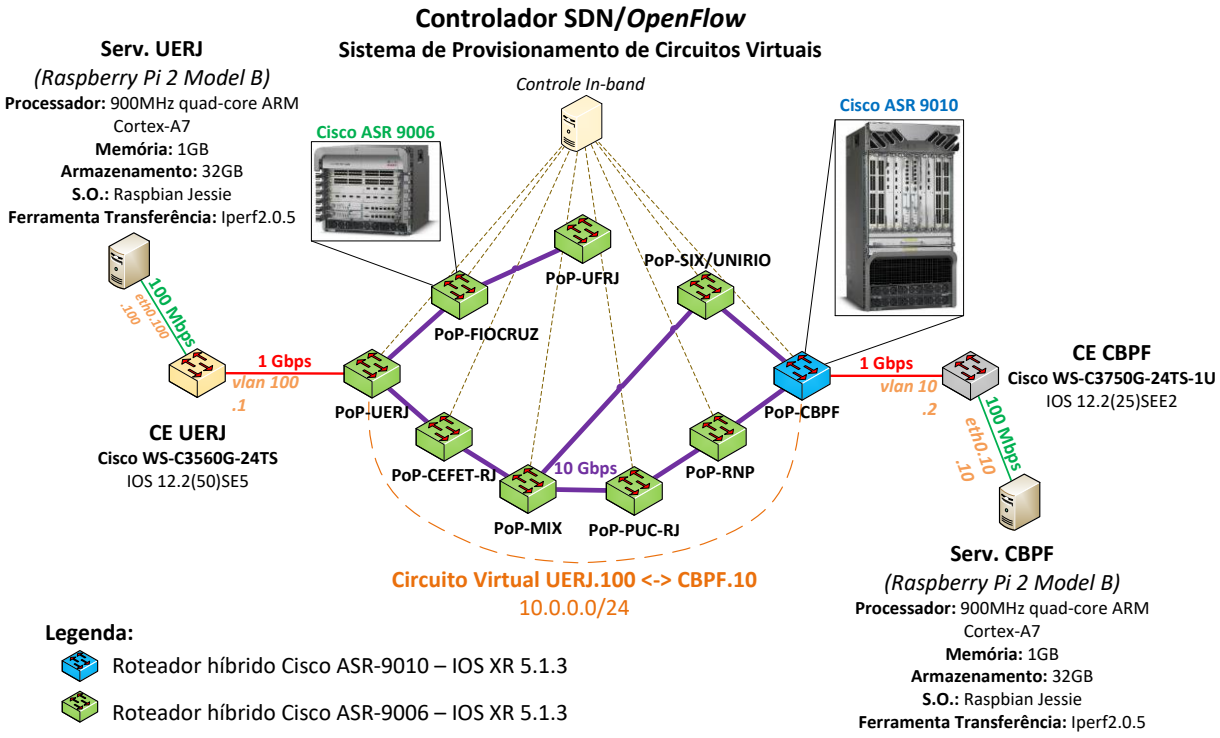


Figura 4.2: Cenário de demonstração da aplicação de provisionamento de circuitos virtuais operando sobre a infraestrutura da Redecomep-Rio.

A configuração dos *switches* PE híbridos OpenFlow e dos servidores utilizados para essa demonstração pode ser encontrada na Tabela 4.2 adiante.

Tabela 4.2: Configuração dos equipamentos utilizados para demonstração do funcionamento da aplicação.

Equipamento	*Hardware*			S.O.
Roteadores Cisco ASR 9010 e 9006	Supervisora:	A9K-RSP440-TR		Cisco IOS XR *Software*, *Version* 5.1.3
		Memória Interna:	6-GB *Error-Correcting Code* (ECC)-*protected* DRAM	
		Armazenamento:	2 x 16-GB SSDs	
	Linecard:	A9K-MOD80-TR		
		(80G *Modular Linecard*, *Packet Transport Optimized*)		
	Módulos 10Gbps:	A9K-MPA-2X10GE		
		(ASR 9000 2-*port* 10GE *Modular Port Adapter*)		
	Módulos 1Gbps:	A9K-MPA-20X1GE		
		(ASR 9000 20-*port* 1GE *Modular Port Adapter*)		
Servidores UERJ e CBPF (Raspberry Pi 2 Model B)	Processador:	QUAD Core Broadcom BCM2836 900MHz		Raspbian Jessie (Debian 8.6) Kernel 4.4.21-v7+
	Memória RAM:	1 GB		
	Armazenamento:	32 GB microSDHC Class 10		

Os roteadores ASR9000 representam dispositivos PE tipicamente utilizados como roteadores de agregação em *backbones* de ISPs (*Internet Service Providers*) e também na Redecomep-Rio. Todos os *switches* PE foram conectados entre si utilizando os enlaces de 10Gbps da rede em produção da Redecomep-Rio, conforme citado anteriormente. Cabe frisar ainda que em todas as avaliações especificadas nas Seções 4.2, 4.3 e 4.4 foi utilizado um servidor controlador com especificações similares às utilizadas nos testes apresentados na Seção 4.1.1, sendo elas: máquina virtual operando sobre sistema VMware vSphere 5

Essential Plus; processador Intel(R) Xeon(R) CPU E5-2430 0 @ 2.20GHz (6 CPU x 2 cores); memória RAM de 4GB; armazenamento de 6GB; sistema operacional Ubuntu 14.04.3 LTS; kernel 3.13.0-24-generic.

Através do mecanismo de configuração da aplicação descrito na Seção 3.2. Verificação dos Endpoints e Identificação de Circuito Virtual, um circuito virtual foi configurado entre os *switches* CE localizados na UERJ e no CBPF. Em cada um dos servidores foi utilizado o mecanismo de configuração padrão de interfaces de rede do sistema operacional Ubuntu 14.04.3 LTS utilizado. Nesse caso, foi adotado tráfego não "tageado" para acesso remoto aos dispositivos e para representar o tráfego Internet dos clientes. As VLANs 802.1q de ID 10 e 100 foram adotadas como identificação dos *endpoints* do circuito virtual no CBPF e na UERJ, respectivamente. Além disso, como as conexões de acesso dos servidores foram ativadas em enlaces dedicados a 100 Mbps, nenhum mecanismo de limitação de tráfego teve de ser implementado.

A ferramenta *iperf* v2.0.5 foi utilizada para testes de transmissão de dados do tipo memoria-a-memoria via *User Datagram Protocol* (UDP), já que essa ferramenta permite a avaliação de vazão e número de perdas a cada segundo (Diniz e Alves Jr., 2014). Os testes foram executados em um período de 60 segundos. Durante o segundo de número 30 foi gerada uma queda de enlace entre o PoP-CBPF e o PoP-SIX/Unirio, de modo a desviar o tráfego antes direcionado pelo caminho $PoP-CBPF \leftrightarrow PoP-SIX/UNIRIO \leftrightarrow PoP-MIX \leftrightarrow PoP-CEFET-RJ \leftrightarrow PoP-UERJ$ para o caminho $PoP-CBPF \leftrightarrow PoP-RNP \leftrightarrow PoP-PUC \leftrightarrow PoP-MIX \leftrightarrow PoP-CEFET-RJ \leftrightarrow PoP-UERJ$. Por fim, os resultados da ferramenta *iperf* foram utilizados para gerar os gráficos apresentados no Capítulo 5. Medidas e Resultados. Dessa maneira, foi possível verificar o mecanismo de provisionamento de circuitos virtuais com VLANs distintas em cada *endpoint* e o mecanismo de recuperação de falhas em ação, confirmando assim a elegibilidade e viabilidade do sistema.

4.3 Caracterização do Intervalo de Tempo de Convergência

Convergência de rede é um termo amplamente utilizado em situações distintas. Antes de prosseguir com a definição do método de avaliação do intervalo de tempo de convergência é importante definir o termo "convergência de rede". A convergência da rede e/ou, no caso de ambientes SDN, a convergência da aplicação é definida neste caso como o processo de sincronização das tabelas de fluxos dos *switches* OpenFlow após uma mudança de topologia da rede. A rede é dita como convergida quando nenhuma das tabelas de fluxos é modificada por um determinado período de tempo. Então, desse momento em diante os termos "convergência da rede" ou "convergência da aplicação" ou "convergência do sistema" são equivalentes.

Para realizar a caracterização do tempo de convergência da aplicação buscou-se determinar o intervalo de tempo máximo fim a fim para o início do encaminhamento de pacotes entre os *endpoints* de um circuito virtual. Para isso, dois cenários distintos de operação foram definidos, são eles:

i. Cenário 1: Início do Funcionamento da Aplicação
ii. Cenário 2: Recuperação em Caso de Falhas

Cada um desses cenários denota uma situação diferente de mudança de topologia de rede, então a sua caracterização se faz importante. O cenário (i) é relevante para que no caso de uma falha total do sistema o operador de rede seja capaz de identificar qual o intervalo de tempo mínimo de convergência da rede após a reinicialização da aplicação e assim será possível tomar as medidas cabíveis. Já o cenário (ii) permite caracterizar o mecanismo de recuperação de falhas, identificando assim o número médio de erros de transmissão e o intervalo de indisponibilidade dos circuitos causados por cada alteração na topologia de rede.

Para cada um dos cenários citados, as métricas avaliadas e os métodos de avaliação são descritos a seguir.

4.3.1 Cenário 1: Início do Funcionamento da Aplicação

Dentre os possíveis cenários de convergência da aplicação, o início do funcionamento é o mais crítico, pois engloba a maior quantidade de passos para estabelecimento do circuito fim a fim. Esses passos incluem: (i) Descoberta de topologia; (ii) Verificação da base de dados de circuitos; (iii) Convergência do protocolo SPF; e (iv) Convergência do mecanismo de prevenção de *loops* de encaminhamento baseado em *Split Horizon*.

A seguir são definidos a métrica avaliada e o método de avaliação.

- **Métrica:** Intervalo de tempo entre o início do funcionamento da aplicação e o momento em que a aplicação converge. Isto é, o intervalo de tempo de convergência do algoritmo SPF, do algoritmo do mecanismo de *Split Horizon* e do algoritmo proposto de estabelecimento de circuitos virtuais e VLAN *stitching* foi avaliado.

- **Método de avaliação:** Emulou-se o processo de estabelecimento de uma conexão entre dois *hosts* de um mesmo cliente, assumindo que o circuito já estava previamente descrito na base de dados de configuração. Desta forma, a metodologia consistiu em medir o intervalo de tempo entre o início do funcionamento da solução VCFlow e o primeiro pacote ARP *Reply* recebido por um *host* origem. Para isso uma requisição ARP entre dois *hosts* conectados a *switches* CE distintos já estava em curso antes do início do funcionamento da solução. Para geração dos pacotes ARP para estabelecimento do circuito, a ferramenta *arp-scan* v1.8.1 (Hills, 2011) (uma descrição mais detalhada da ferramenta pode ser encontrada na Seção de Apêndices 10.1. *arp-scan*) foi utilizada com intervalos entre mensagens ARP *Request* de 25ms e *timeout* de 0ms. O instante de tempo de inicialização do VCFlow foi obtido a partir do *log* da aplicação. Os intervalos entre os pacotes ARP foram checados através da ferramenta *tcpdump* (Fuentes e Kar, 2005), capturando pacotes na interface do *host* de origem (uma descrição mais detalhada da ferramenta pode ser encontrada na Seção de Apêndices 10.3. *tcpdump*). A Figura 4.3 exibe uma visão geral da metodologia descrita.

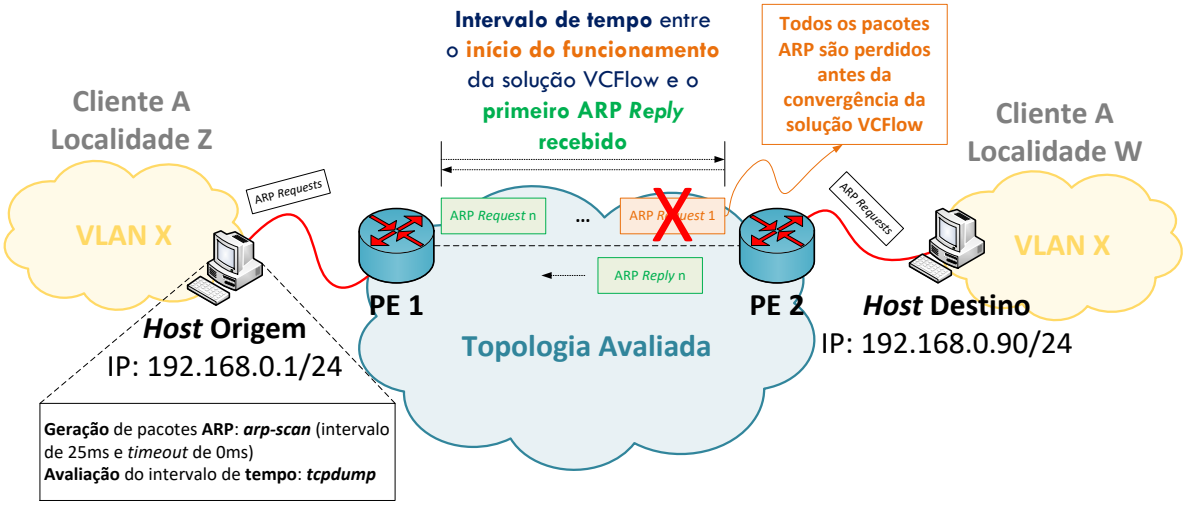


Figura 4.3: Metodologia de avaliação do tempo de convergência da solução VCFlow para o cenário de início de funcionamento.

Os resultados foram subdivididos em dois tipos de topologias distintas, que são: (a) topologias emuladas e (b) estudo de caso da implantação no *backbone* da Redecomep-Rio.

Para o caso das topologias emuladas, utilizou-se o *Open vSwitch* (um *switch* OpenFlow via *software*) por meio do emulador de redes Mininet v2.2.1 (Lantz *et al.*, 2010) (uma descrição mais detalhada da ferramenta pode ser encontrada na Seção de Apêndices 10.4. Mininet). Foi utilizado um protótipo da aplicação VCFlow capaz de instalar as regras nos *switches* do caminho do circuito virtual por meio do protocolo OpenFlow v1.3, assim como o utilizado na demonstração descrita na Seção 4.2. Demonstração do Sistema Protótipo. Nesse protótipo, todos os procedimentos de verificação do circuito e de escolha de menor caminho foram desenvolvidos conforme descritos nas Seções 3.2 e 3.3. Cabe frisar que o procedimento de descoberta de topologia em redes híbridas, conforme descrito na Seção 3.4, não foi utilizado, tendo em vista que a topologia emulada não era híbrida. O procedimento de descoberta de topologia utilizado foi o tradicional, em que os pacotes LLDP de descoberta de topologia são encaminhados sem campo de VLAN no cabeçalho *Ethernet*.

(a) Topologias Emuladas:

Para realizar a caracterização inicial da convergência da aplicação para o Cenário 1: Início do Funcionamento da Aplicação, emulou-se quatro diferentes topologias. Utilizou-se duas topologias padrão do Mininet e duas customizadas, conforme descritas em detalhes adiante.

Topologia Linear com 2 Switches: Como avaliação inicial, a topologia mais simples disponível no emulador com dois *switches* foi utilizada. Os *switches* foram conectados diretamente entre si e dois *hosts* foram conectados cada um a um dos *switches*, conforme pode ser visualizado na Figura 4.4.

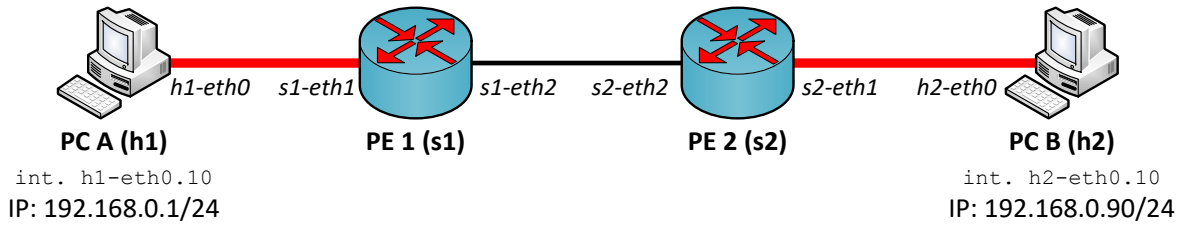


Figura 4.4: Layout da topologia linear com dois switches e dois hosts.

Topologia Tree com Profundidade 3 e Fanout 2: A topologia *tree* é um tipo de topologia voltada para *datacenters*, onde cria-se uma árvore com uma determinada profundidade k (número de níveis da árvore) e *fanout* n, onde cada *switch* possui uma conexão ao seu *switch* pai e há n filhos por *switch***.** Realizou-se a avaliação de funcionamento em uma topologia de profundidade $k = 3$ e *fanout* $n = 2$, ou seja, três níveis com sete *switches* e oito *hosts* conectados de acordo com a Figura 4.5.

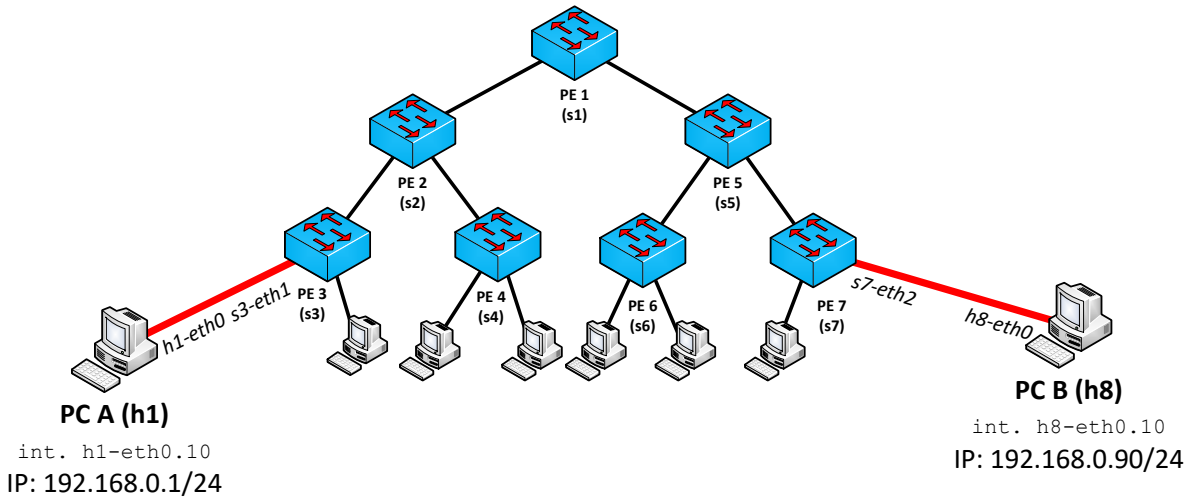


Figura 4.5: Layout da topologia Tree com profundidade três e fanout dois.

Topologia Fat-Tree com Profundidade 3 e Fanout 2: Assim como a topologia *Tree*, a *Fat-Tree* é uma topologia de rede em árvore onde os servidores são conectados à camada mais inferior de nós. A característica que a diferencia em relação à topologia *Tree* tradicional é que para qualquer *switch*, o número de enlaces para baixo em direção a seus *switches* filho é igual ao número de enlaces em direção ao seu *switch* pai. Portanto, os enlaces vão se tornando mais gordos (do inglês *fat*) ao passo que se vai em direção ao nó raiz da árvore. Através dessa topologia, é possível notar a potencialidade de utilização do sistema não só em redes de *backbone* ou redes de *campus*, como também possíveis aplicações em redes de *datacenters*. Do mesmo modo que no caso da topologia *Tree*, realizou-se a avaliação de funcionamento em uma topologia de profundidade $k = 3$ e *fanout* $n = 2$, conforme pode ser visto na Figura 4.6. Cabe frisar também que como existem múltiplas redundâncias de enlaces entre os *switches* ocorrem *loops*, onde o mecanismo de prevenção de *loops* baseado em *Split Horizon* torna-se essencial para o pleno funcionamento da rede.

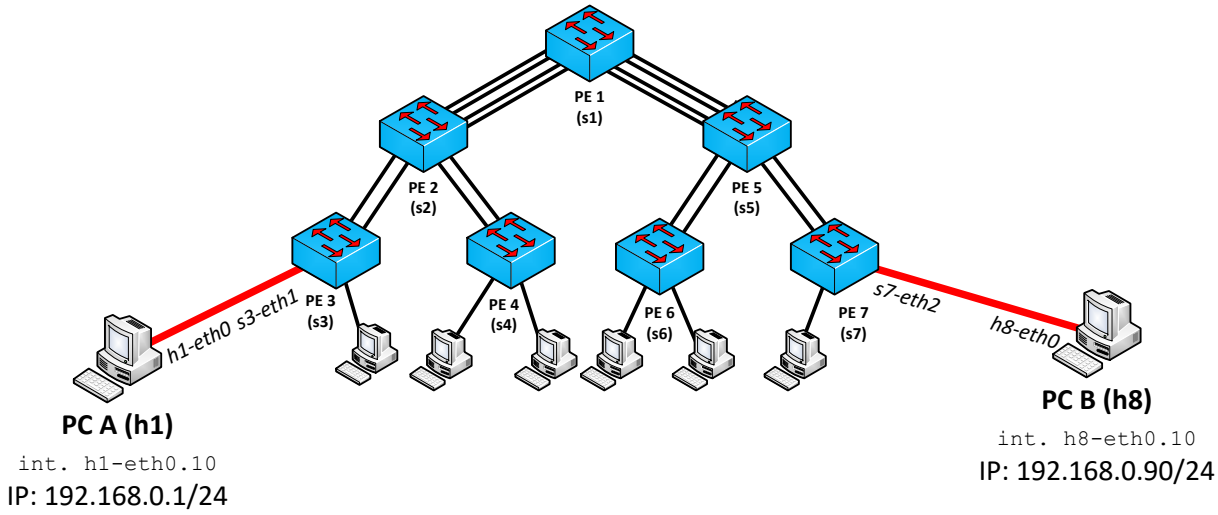


Figura 4.6: Layout da topologia Fat-Tree com profundidade três e fanout dois.

Topologia Emulada do Backbone Redecomep-Rio: Como a aplicação visa ser implementada na rede em produção da Redecomep-Rio, optou-se por realizar uma emulação de uma infraestrutura similar a sua topologia real. A topologia de *backbone* da Redecomep-Rio, atualmente, é composta por nove Pontos de Presença espalhados por uma topologia em anel com múltiplas redundâncias, simplificadamente descrito na Seção 4.2. Demonstração do Sistema Protótipo e na Figura 4.2. A Figura 4.7 exibe o *layout* da topologia da rede em produção simplificada utilizada na emulação, em que foram conectados nove *switches* com dois *hosts*. Nesse cenário, cada *host* faz o papel dos *switches* CE que devem ser conectados aos *switches* PE de *backbone*. Vale ressaltar que como parte da topologia é em anel, ocorrem *loops*.

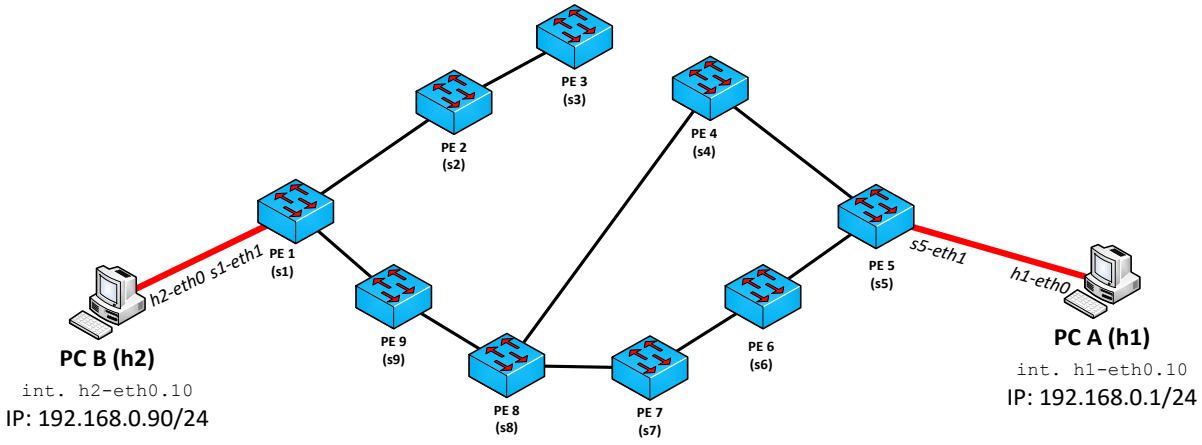


Figura 4.7: Layout da topologia emulada da Redecomep-Rio com nove switches e dois hosts para cenário de avaliação de início de funcionamento da solução VCFlow.

(b) Estudo de Caso da Implantação no Backbone Redecomep-Rio

A seguir é apresentada a avaliação do sistema protótipo na Redecomep-Rio. O estudo de caso foi executado em uma configuração com equipamentos híbridos similar a topologia descrita na Seção 4.2. Demonstração do Sistema Protótipo. Entretanto, em vez de se utilizar *tags* de VLAN com VLAN ID diferentes em cada localidade, foi adotado o mesmo VLAN ID 10 em ambas as localidades. Do ponto de vista do *switch* CE, a conexão entre ele e o roteador PE ocorreu através de uma interface do tipo tronco com permissão de tráfego das VLANs 802.1q ID 10 para o circuito virtual e ID 99 para acesso remoto ao dispositivo e para emular o tráfego Internet. Também do ponto de vista do *switch* CE, a sua conexão com os PCs ocorreu também através de uma interface do tipo tronco, com permissão de tráfego "tageado" com VLAN ID 10 para o circuito virtual e não "tageado" para acesso remoto aos servidores. A Figura 4.8 exibe o *layout* da topologia avaliada, onde o servidor conectado ao *switch* CE CBPF (PC A) foi utilizado como origem e o servidor conectado ao *switch* CE UERJ (PC B) como destino nas medições realizadas.

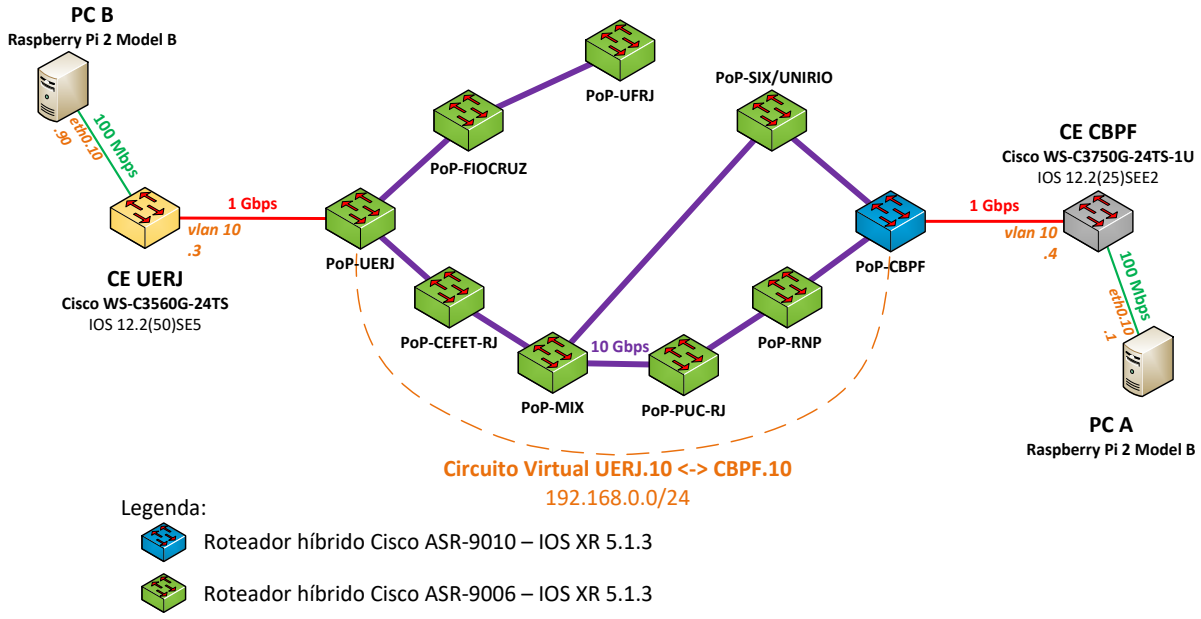


Figura 4.8: Layout da topologia do estudo de caso da aplicação de estabelecimento de circuitos virtuais na Redecomep-Rio para o cenário de início de funcionamento da aplicação.

4.3.2 Cenário 2: Recuperação em Caso de Falhas

Diferentemente do cenário de início do funcionamento da aplicação, o procedimento para reestabelecimento do circuito fim a fim envolve uma quantidade menor de passos, visto que não carece da verificação da base de dados de circuitos. Esses passos são então classificados como: (i) Identificação de falha de enlace e descoberta de nova topologia; (ii) Convergência do protocolo SPF; e (iii) Convergência do mecanismo de prevenção de *loops* de encaminhamento baseado na técnica de *Split Horizon*. Cabe destacar que em uma situação real de funcionamento este é o cenário mais provável de ocorrer.

A seguir é definido a métrica avaliada e o método de avaliação.

- **Métrica:** Intervalo de tempo entre a detecção da ocorrência de uma falha de rede do ponto de vista da aplicação que faz uso do circuito virtual (FPING) e o momento em que a solução de estabelecimento de circuitos VCFlow converge novamente. Ou seja, esse intervalo de tempo engloba a convergência dos algoritmos do SPF, do *Split Horizon* e do algoritmo proposto de estabelecimento de circuitos virtuais e VLAN *stitching*.

- **Método de avaliação:** Emulou-se o processo de reestabelecimento de uma transmissão entre dois *hosts* de um mesmo cliente. Assumiu-se que o circuito já estava configurado entre todos os *switches* da topologia avaliada e uma checagem de alcançabilidade via ICMP já estava em curso. Desse modo, a metodologia consistiu em medir o intervalo de tempo entre o último pacote ICMP *Echo Reply* a ser recebido pelo *host* de origem antes da primeira perda de pacotes e o primeiro pacote ICMP *Echo Reply* recebido pelo *host* origem após a primeira perda. Para geração dos pacotes para a avaliação foi utilizada a ferramenta FPING[19] com intervalos entre mensagens ICMP *Echo Request* de 25ms e *timeout* de 0ms. Para checagem do intervalo de tempo dos pacotes foi também utilizada a ferramenta *tcpdump* capturando pacotes na interface do *host* de origem, além dos *logs* do VCFlow terem sido também analisados. A Figura 4.9 exibe uma visão geral da metodologia descrita.

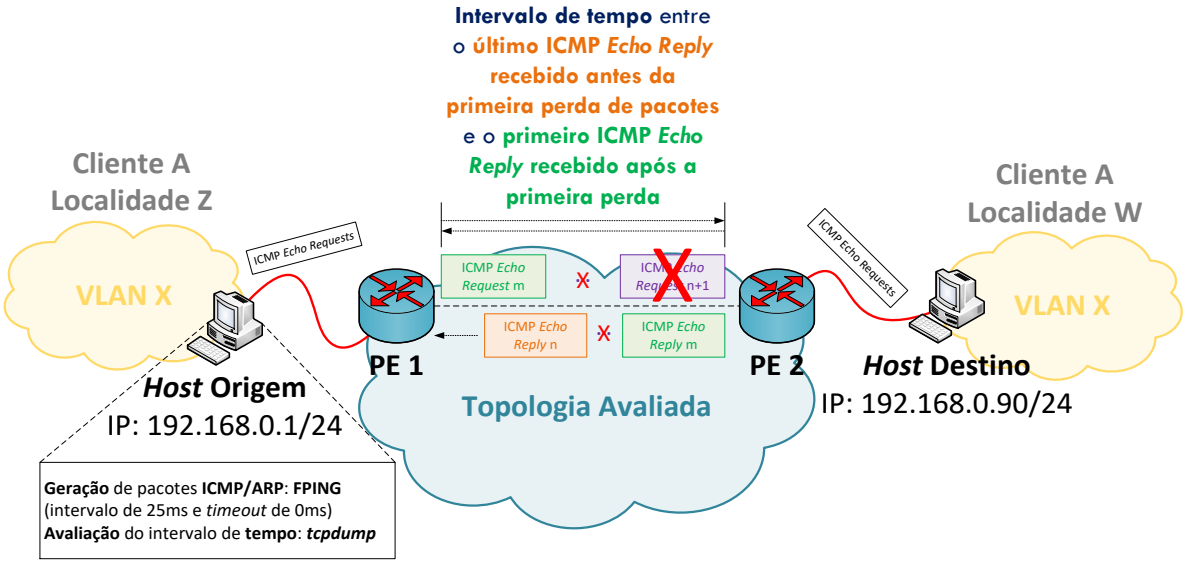


Figura 4.9: Metodologia de avaliação do tempo de convergência da aplicação para o cenário de recuperação em caso de falhas.

[19] A opção pela adoção da ferramenta FPING em vez do *arp-scan*, como nas avaliações de início de funcionamento da aplicação, ocorreu pelo fato dos cabeçalhos ICMP possuírem um campo de número de sequência. Desse modo, foi possível identificar quais pacotes foram enviados, mas não obtiveram resposta, baseado nesses números de sequência. Isso não seria possível de ser realizado se fossem utilizadas ferramentas que encaminham pacotes ARP, como é o caso do *arp-scan*.

Assim como na Seção 4.3.1. Cenário 1: Início do Funcionamento da Aplicação, os resultados foram subdivididos em dois tipos de topologias distintas, que são: (a) topologias emuladas e (b) estudo de caso da implantação no *backbone* da Redecomep-Rio. As mesmas características de ferramentas de emulação foram utilizadas.

(a) Topologias Emuladas:

Para realizar a caracterização da convergência da aplicação para o Cenário 2: Recuperação em Caso de Falhas, também foram emuladas três diferentes topologias. Utilizou-se três topologias customizadas, conforme descritas em detalhes a seguir. Cabe destacar que como todas as topologias emuladas possuem redundâncias então ocorrem *loops*, logo o mecanismo de *Split Horizon* é essencial para o pleno funcionamento da rede.

Topologia Linear com 2 Switches e 2 Enlaces: Como avaliação primária, emulou-se uma topologia similar a topologia linear com 2 *switches* descrita na Seção 4.3.1. Cenário 1: Início do Funcionamento da Aplicação. Porém, em vez de um único enlace entre os dois *switches*, foram utilizados dois enlaces. A topologia pode ser vista na Figura 4.10.

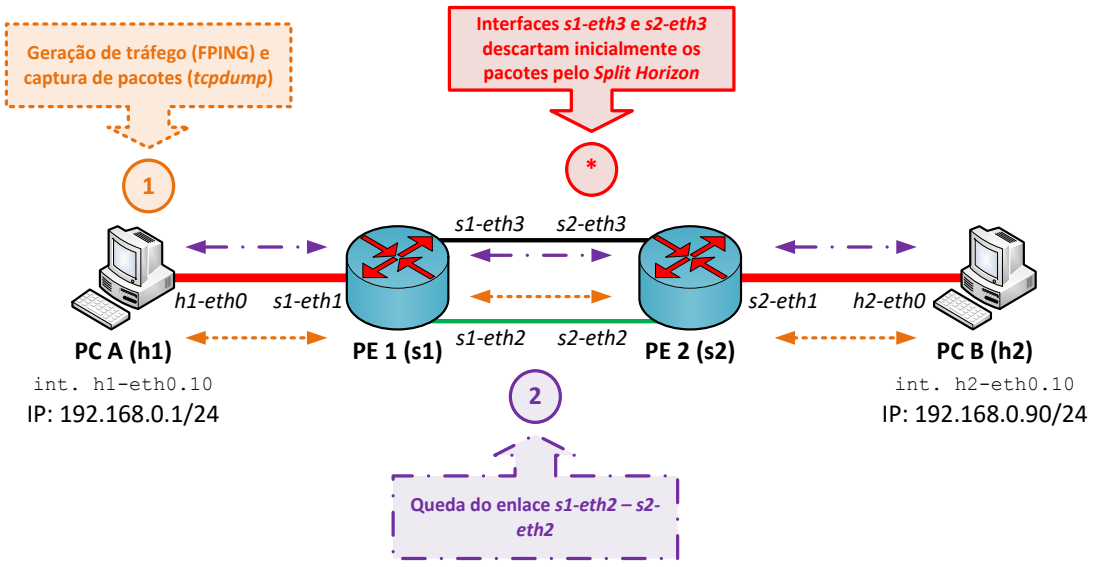


Figura 4.10: Layout da topologia linear com dois switches, dois enlaces entre switches e dois hosts.

O procedimento de teste para essa topologia ocorreu conforme as etapas descritas abaixo:

1. Início da geração de tráfego ICMP via FPING a partir do *host* origem *h1*. Tráfego encaminhado pelo caminho a seguir, descrito pelas interfaces de origem de cada nó no sentido *h1* para *h2*: *h1-eth0*; *s1-eth2*; *s2-eth1*. O tráfego é simétrico.
2. Queda no enlace *s1-eth2 – s2-eth2*. O tráfego é então desviado pelo caminho*:* *h1-eth0*; *s1-eth3*; *s2-eth1*.

Topologia Fat-Tree com Profundidade 3 e Fanout 2: Assim como no caso do Cenário 1: Início do Funcionamento da Aplicação, uma topologia do tipo *Fat-Tree* com parâmetros

de profundidade $k = 3$ e *fanout* $n = 2$ foi utilizada. A topologia descrita e o procedimento de teste pode ser visto na Figura 4.11.

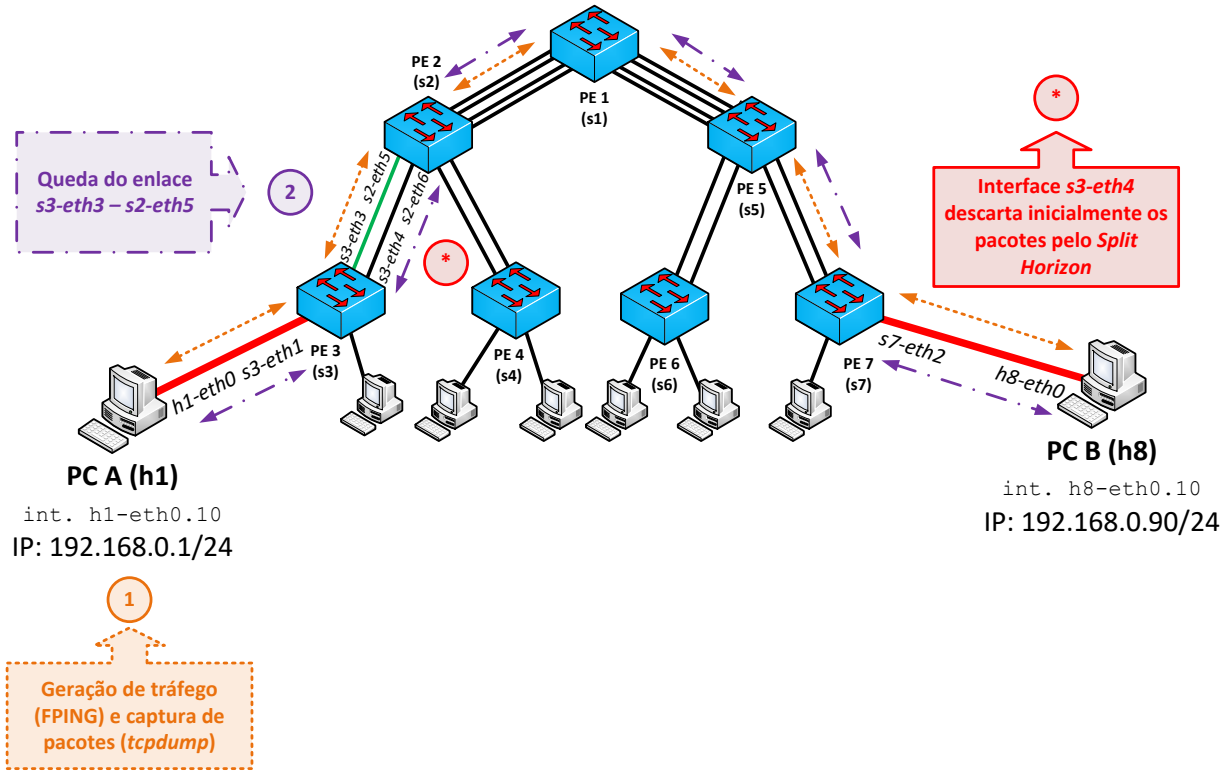


Figura 4.11: Layout da topologia Fat-Tree com profundidade três e fanout dois.

O procedimento de teste para a topologia *Fat-Tree* ocorreu conforme descrito abaixo:

1. Início da geração de tráfego ICMP via FPING a partir do *host* origem *h1*. Tráfego encaminhado pelo caminho a seguir no sentido *h1* para *h8*, considerando tráfego simétrico: *h1*; *s3-eth3*; *s2; s1; s5; s7; h8*.
2. Queda no enlace *s3-eth3 – s2-eth5*. O tráfego é então desviado pelo caminho*:* *h1*; *s3-eth4*; *s2; s1; s5; s7; h8*.

Topologia Emulada do Backbone Redecomep-Rio: Assim como no caso do Cenário 1: Início do Funcionamento da Aplicação, emulou-se uma topologia simplificada da Redecomep-Rio. A topologia emulada e o procedimento de teste pode ser visto na Figura 4.12.

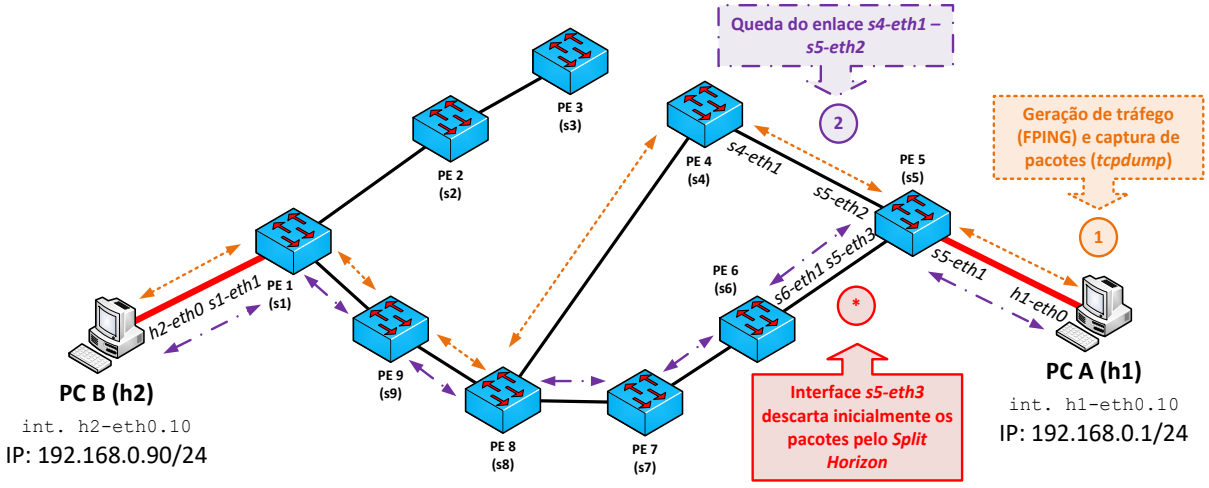


Figura 4.12: Layout da topologia emulada da Redecomep-Rio com nove switches e dois hosts para cenário de avaliação de início de recuperação em caso de falhas.

O procedimento de teste para a topologia emulada da Redecomep-Rio ocorreu assim como descrito a seguir:

1. Início da geração de tráfego ICMP via FPING a partir do *host* origem *h1*. Tráfego encaminhado pelo caminho a seguir no sentido *h1* para *h2*, considerando tráfego simétrico: *h1; s5; s4; s8; s9; s1; h2*.
2. Queda no enlace *s4-eth1 – s5-eth2*. O tráfego é então desviado pelo caminho: *h1; s6; s7; s8; s9; s1; h2*.

(b) Estudo de Caso da Implantação no Backbone Redecomep-Rio

Do mesmo modo que no Cenário 1: Início do Funcionamento da Aplicação o sistema protótipo foi também avaliado na Redecomep-Rio. No presente cenário, assim como no anterior ambos os *endpoints* do circuito virtual estabelecido são *switches* CE utilizados em instituições clientes da Redecomep-Rio. Os equipamentos e a topologia são os mesmos descritos na Seção 4.3.1 (b). A Figura 4.13 demonstra a topologia utilizada nesse estudo de caso e o seu respectivo procedimento de teste.

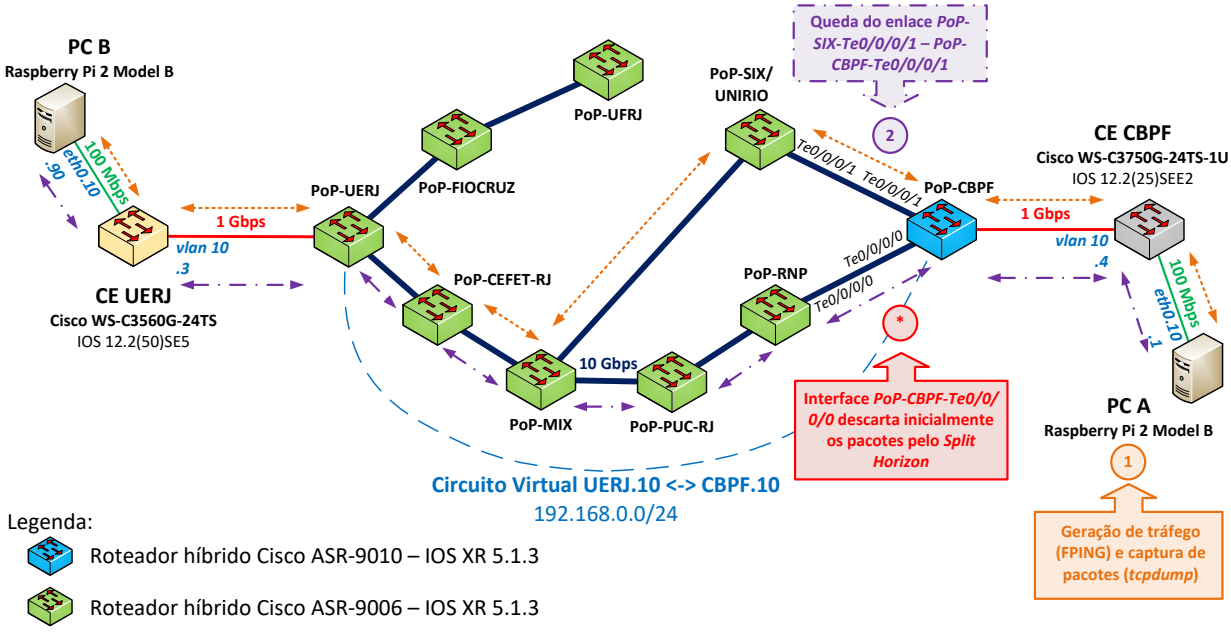


Figura 4.13: Layout da topologia do estudo de caso da aplicação de estabelecimento de circuitos virtuais na Redecomep-Rio para o cenário de recuperação em caso de falhas.

O procedimento de teste para a topologia de estudo de caso da Redecomep-Rio para o cenário de recuperação em caso de falhas pode ser encontrado a seguir:

1. Início da geração de tráfego ICMP via FPING a partir do *host* origem *PC-A*. Tráfego encaminhado pelo caminho a seguir no sentido *PC-A* para *PC-B*, considerando simetria de tráfego: PC-A; CE-CBPF; PoP-CBPF; PoP-SIX; PoP-MIX; PoP-Cefet-RJ; PoP-UERJ; CE-UERJ; PC-B.
2. Queda no enlace PoP-CBPF-*Te0/0/0/1* – PoP-SIX-*Te0/0/0/1*. O tráfego é então desviado pelo caminho: PC-A; CE-CBPF; PoP-CBPF; PoP-RNP; PoP-PUC-RJ PoP-MIX; PoP-Cefet-RJ; PoP-UERJ; CE-UERJ; PC-B.

4.4 Caracterização do Intervalo de Tempo de Processamento das Mensagens de Sinalização e de Estabelecimento de Circuitos Fim a Fim

Como descrito por Metter, não somente a implementação de uma aplicação OpenFlow (OF) pode influenciar em seu desempenho, mas a arquitetura de cada controlador pode também interferir (Metter *et al.*, 2015). Então, visando identificar o desempenho do algoritmo de estabelecimento de circuitos e VLAN *stitching* do sistema protótipo VCFlow em uma rede em que as tabelas de fluxos já convergiram, procurou-se avaliar duas métricas:

i. Tempo de processamento das mensagens de sinalização OF

ii. Tempo de estabelecimento de circuitos

O tempo de processamento das mensagens de sinalização é aqui definido como: o intervalo entre (1) o instante em que os pacotes OF do tipo *Packet-In* são recebidos pela aplicação no controlador e (2) o instante em os pacotes OF do tipo *Flow-Mod* são encaminhados pela aplicação no controlador ao último *switch* do caminho do circuito virtual,

ou seja, o *endpoint* do circuito. Deve-se ter em vista que o sistema proposto instala as regras de fluxo proativamente em cada direção (como descrito na Seção 3.3. Escolha do Menor Caminho, VLAN *Stitching* e Estabelecimento do Circuito Virtual) e adota o esquema de prevenção de *loops* de encaminhamento baseado na técnica de *Split Horizon* (como descrito na Seção 3.5. Prevenção de *Loops* de Encaminhamento). Por esse motivo é realizada uma análise do tempo de processamento em cada direção de *endpoint* 1 para *endpoint* 2 e vice-versa. Uma ilustração para a metodologia de avaliação do intervalo de tempo de processamento pode ser encontrada na Figura 4.14. Para realizar as avaliações, os mesmos procedimentos e topologias de medição descritos na Seção 4.3.1 foram também adotados, sendo que a identificação dos instantes de recebimento e encaminhamento de pacotes de sinalização OF foi realizada integralmente através da análise dos *logs* do controlador.

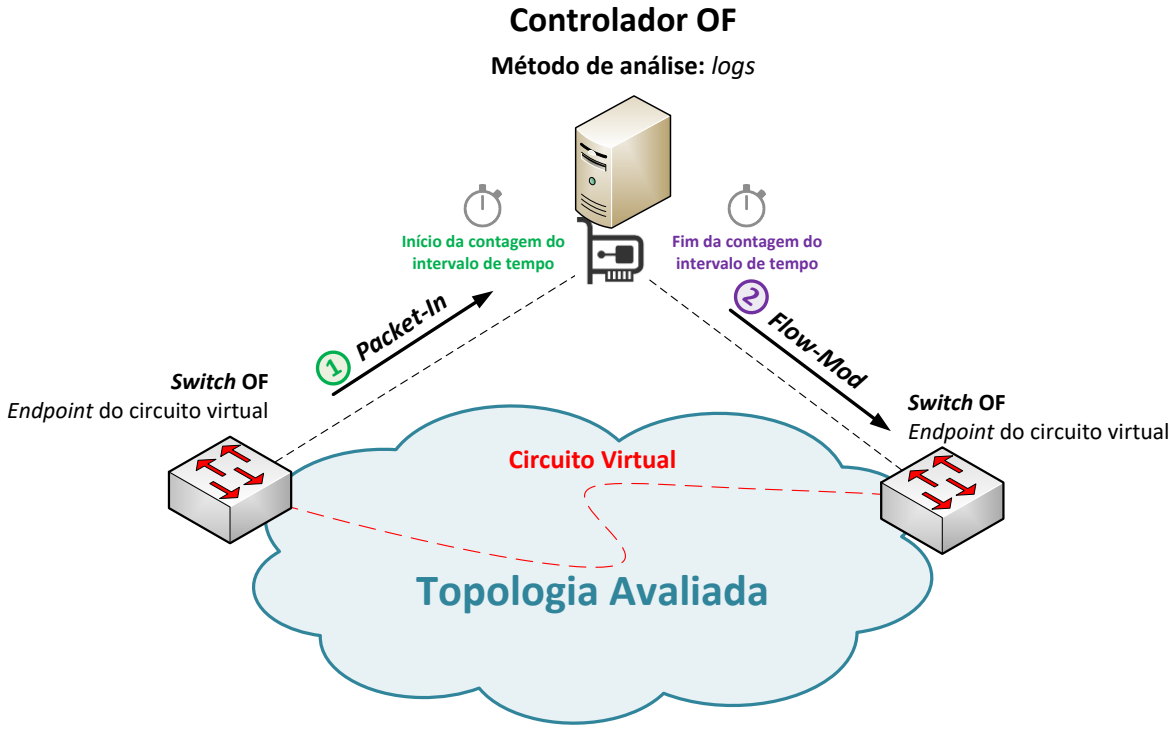


Figura 4.14: Visão geral da metodologia de avaliação do tempo de processamento de mensagens de sinalização OpenFlow.

O intervalo de tempo de estabelecimento de circuitos foi definido como o intervalo de tempo para encaminhamento de pacotes fim a fim através de um circuito virtual em uma rede convergida. Isto é, dado que todo o processo de descoberta de topologia e cálculo de tabelas de roteamento baseados no funcionamento dos protocolos LLDP e do SPF já ocorreu, procurou-se avaliar o intervalo de tempo entre o início de encaminhamento de pacotes ARP *Request* em um *host* de origem e o momento de recepção do pacote ARP *Reply* proveniente de um *host* de destino. Para tal, assim como na Seção 4.3.1, o utilitário *arp-scan* foi utilizado para geração de pacotes ARP com intervalo de 25ms e *timeout* de 0ms e o utilitário *tcpdump* foi utilizado para captura de pacotes no *host* de origem. A Figura 4.15 exibe uma visão geral da metodologia descrita. Cabe ressaltar também que as mesmas topologias emuladas e de estudo de caso descritas na Seção 4.3.1. Cenário 1: Início do Funcionamento da Aplicação foram utilizadas.

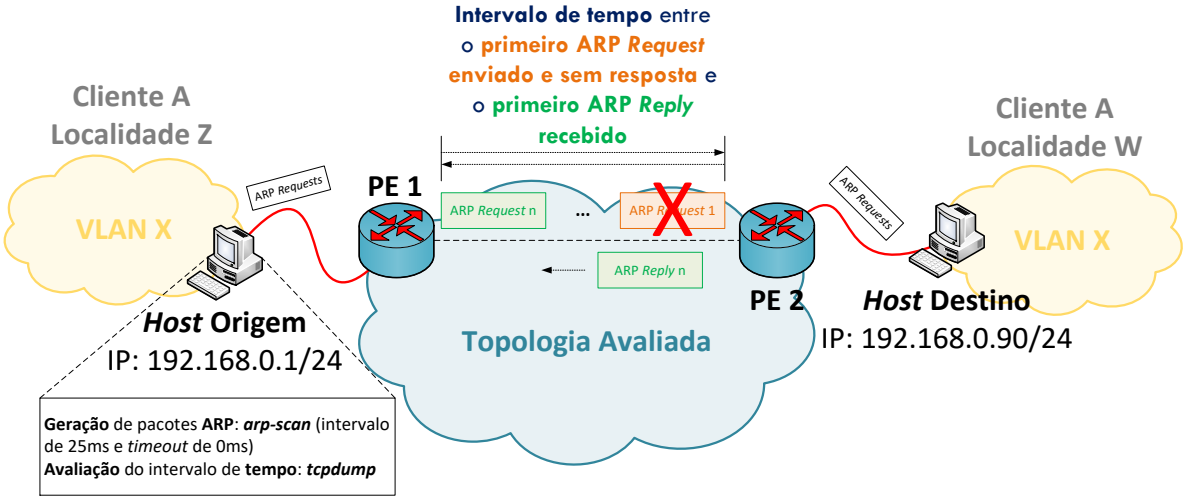


Figura 4.15: Metodologia de avaliação do tempo estabelecimento de circuitos em uma rede convergida.

Capítulo 5

5 MEDIDAS E RESULTADOS

Neste capítulo, serão apresentados os resultados dos experimentos descritos em detalhes no Capítulo 4. Caracterização do VCFlow realizados nos laboratórios do CBPF/MCTIC, responsável pela área de operações da Rede-Rio e também responsável pela coordenação técnica do projeto Redecomep-Rio. Esses experimentos tiveram como objetivo realizar a caracterização da solução proposta VCFlow para provisionamento de circuitos virtuais em redes híbridas, através da: (1) avaliação de controladores OpenFlow disponíveis, por meio da análise de desempenho, características e funcionalidades, visando a seleção do controlador mais adequado para o desenvolvimento e implementação da solução VCFlow; (2) demonstração do funcionamento da solução VCFlow, operando em uma rede SDN/OpenFlow híbrida em paralelo a uma rede de produção, como o *backbone* da Redecomep-Rio; (3) avaliação do intervalo de tempo de convergência da solução VCFlow; (4) avaliação do intervalo de tempo de processamento das mensagens de sinalização e controle do protocolo OpenFlow pela solução VCFlow e do intervalo de tempo de estabelecimento de circuitos virtuais fim a fim.

A seguir, serão apresentados os resultados obtidos para cada um dos experimentos de (1) a (4) citados acima, cujas metodologias de avaliação são apresentadas em detalhes nas Seções 4.1 a 4.4, respectivamente.

5.1 Avaliação dos Controladores OpenFlow

Nesta seção, é realizada uma pequena comparação em termos de desempenho, características e funcionalidades dos controladores OpenFlow: Floodlight, Ryu, OpenDaylight (ODL) e ONOS. O desempenho dos controladores é verificado através de testes de vazão e de latência das mensagens de sinalização e controle. Os testes de vazão buscam avaliar a quantidade de mensagens *Packet-In* que podem ser processadas em um segundo, enquanto que os testes de latência buscam avaliar o intervalo de tempo de processamento de um único pacote de *Packet-In* pelo controlador. A análise de características e funcionalidades foi realizada baseado: (a) nas experiências obtidas a partir

dos experimentos realizados com os controladores; (b) em buscas realizadas nos *websites* de cada controlador; e (c) principalmente em literatura.

A plataforma de experimentação e a metodologia adotada nos testes de vazão e latência são apresentadas nas Seções 4.1.1, 4.1.2 e 4.1.3, respectivamente. Os resultados são apresentados nos gráficos a seguir e exibem o desempenho de cada controlador em função da quantidade de *switches* OpenFlow por topologia. As topologias adotadas foram emuladas e avaliadas através da ferramenta *Cbench*. Cada rodada de testes foi repetida 30 vezes para obter resultados estatisticamente significativos e seu resultado é apresentado com intervalo de confiança de 95% da média. Ao final da presente subseção são apresentadas as justificativas para adoção do controlador utilizado para o desenvolvimento do VCFlow.

5.1.1 Testes de Vazão

Conforme apresentado na Seção 4.1.2. Testes de Vazão de Mensagens de Controle, os testes de vazão visam avaliar quantas mensagens *Packet-In* os controladores OpenFlow podem processar em um segundo. Nos testes cujos resultados são apresentados a seguir, são contabilizadas as mensagens *Packet-In* que recebem respostas do tipo *Flow-Mod*. As mensagens que não recebem respostas são simplesmente desconsideradas.

A Figura 5.1 demonstra a vazão média dos diferentes controladores avaliados em relação a escalabilidade dos *switches*, variando entre 1 e 20 *switches.* A vazão média para o controlador Ryu se manteve estável independentemente do número de *switches*, apresentando como resultado aproximadamente 4.900 pacotes/s. Já o ODL apresentou um pequeno aumento da vazão entre 1 e 2 *switches*, variando de 69381 pacotes/s a 90208 pacotes/s. A partir de 4 *switches* o ODL começou a apresentar diminuição do desempenho, o qual se manteve estável a partir de 10 *switches,* em torno de 40000 pacotes/s. Os controladores ONOS e Floodlight apresentam uma curva de crescimento em função do número de *switches* com as mesmas características, onde o desempenho aumenta até cerca de 10 *switches* e se mantém constante a partir de então. Entretanto, o Floodlight apresenta valores médios de vazão maiores do que o ONOS, sendo de cerca de 310.000 pacotes/s e 44.000 pacotes/s, respectivamente.

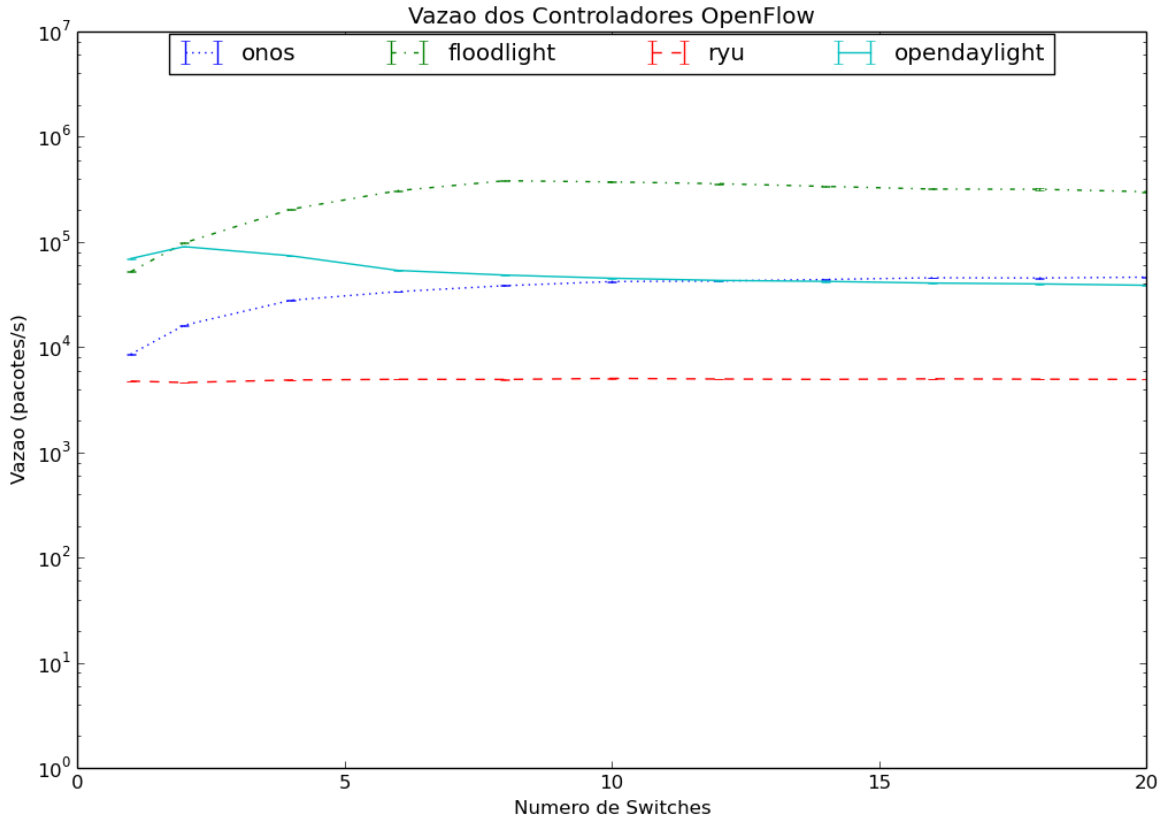


Figura 5.1: Vazão dos Controladores OpenFlow: ONOS, Floodlight, Ryu e ODL.

5.1.2 Testes de Latência

Conforme descrito na Seção 4.1.3. Testes de Latência de Mensagens de Controle, os testes de latência visam avaliar o intervalo de tempo de processamento de uma única mensagem do tipo *Packet-In* pelo controlador. Nesse caso, o *switch* virtual encaminha uma mensagem de *Packet-In* ao controlador e aguarda por uma resposta antes de encaminhar um novo *Packet-In*. Desse modo, a aplicação de mediação *Cbench* avalia o número de mensagens *Packet-In* que obtiveram resposta do tipo *Flow-Mod* em um segundo. A partir desse resultado é possível obter o intervalo de tempo médio de processamento.

A Figura 5.2 exibe a latência média de cada controlador OpenFlow, sendo também avaliada em relação a escalabilidade de *switches*, ou seja, o número de *switches* na topologia. O Floodlight apresenta um resultado estável de cerca de 0,65ms independentemente do número de *switches* na topologia. O ODL também apresenta resultados estáveis de cerca de 1,00 ms até 16 *switches*, quando a partir de então apresenta resultados de 1,38 ms e 1,66 ms para 18 e 20 *switches*, respectivamente. Já o ONOS apresenta resultados bastante similares ao ODL em topologias com até 16 *switches*, porém a partir desse tamanho de topologia há um grande crescimento na latência, variando de 0,65ms com até 16 *switches* para 3,00 ms e 6,90 ms nos casos de 18 e 20 *switches*, respectivamente.

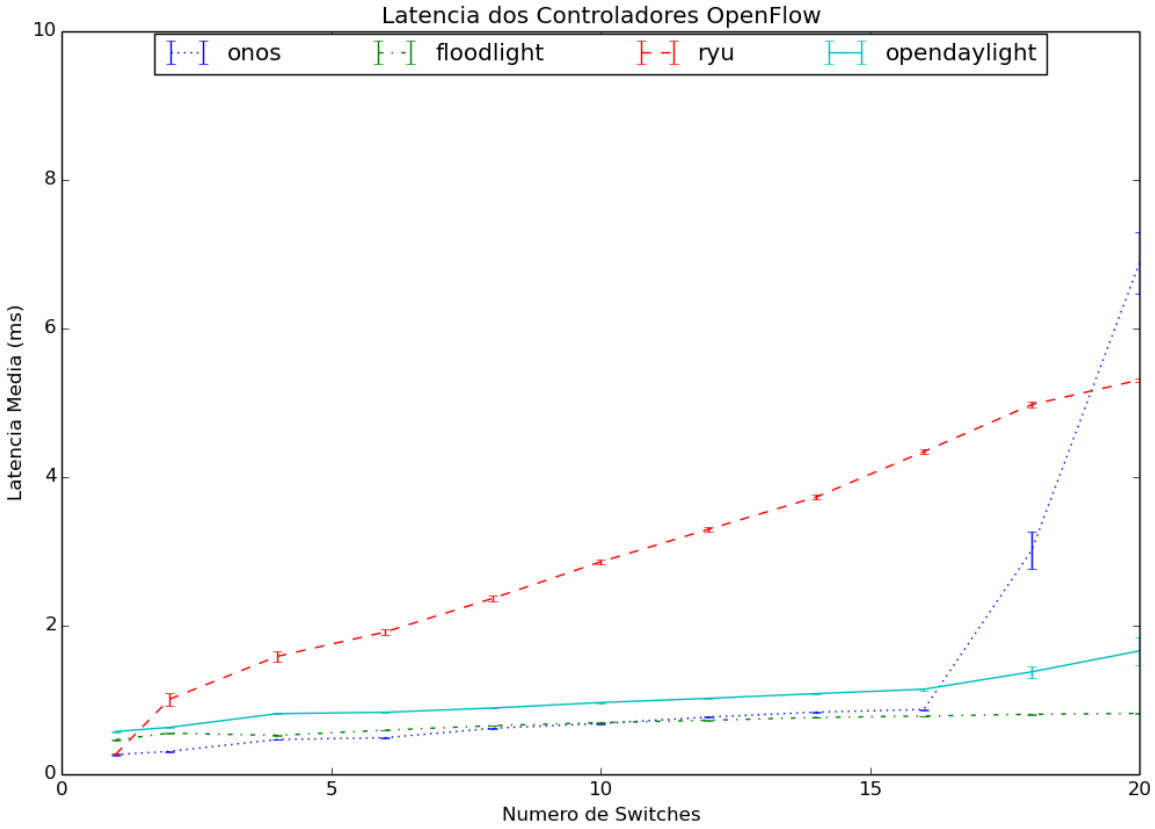


Figura 5.2: Latência dos controladores OpenFlow: ONOS, Floodlight, Ryu e ODL.

5.1.3 Definição do Controlador para o Sistema Protótipo

Os resultados apresentados indicam que o controlador centralizado[20] Floodlight alcança a maior vazão dentre os controladores avaliados. Este controlador é desenvolvido sobre uma arquitetura de sistema altamente concorrente, baseado em sistemas *multi-threading* explorando arquiteturas *multi-core* (Kreutz *et al.*, 2015). Esse tipo de controlador é projetado, principalmente, para atender às demandas de redes densas (*i.e.* com muitos pontos atendidos, e.g. em ambientes corporativos/empresariais) e de *data centers*, uma vez que esse tipo de rede opera com uma grande quantidade de novas requisições aos seus servidores a todo instante e por consequência um grande número de novos fluxos não classificados são também gerados.

Entretanto, apesar de apresentar o melhor desempenho em termos de vazão a análise de algumas das características e funcionalidades influenciou negativamente na escolha do Floodlight como controlador. O Floodlight não provê suporte a linguagens de *scripting* de mais alto nível como Ruby ou Python, somente provê suporte a Java. Esse fato leva a um maior tempo de aprendizado de desenvolvimento e consequentemente uma menor

[20] Controladores centralizados é um termo também utilizado para fazer referência a controladores SDN que não são desenvolvidos para operar de forma distribuída, com múltiplas instâncias de controle para redundância.

produtividade na geração de códigos computacionais. Aliado a isso, o Floodlight também não provê componentes de código facilmente modificáveis, *i.e.*, que permitam customizações baseado em suas situações de uso, como o caso de redes SDN/OpenFlow híbridas em que é necessário a modificação dos cabeçalhos dos pacotes LLDP. Dessa forma, esses fatores levaram a desconsiderar o Floodlight como plataforma de controle para desenvolvimento do sistema VCFlow protótipo.

Diferentemente dos controladores centralizados (*e.g.* Floodlight e Ryu), os controladores distribuídos apresentam como uma de suas principais características poderem ser escalados para irem de encontro aos requisitos de qualquer ambiente de rede, desde pequenas LANs a grandes WANs. Controladores distribuídos podem ser implantados em topologias de *cluster* centralizado de nós de controle ou em um conjunto de elementos distribuídos fisicamente (Jain *et al.*, 2013). Plataformas de controle como ONOS e ODL são exemplos de controladores distribuídos (Kreutz *et al.*, 2015). Contudo, apesar de apresentar como uma de suas principais vantagens a redundância de controle, muitos dos controladores distribuídos apresentam problemas relacionados a um termo conhecido como semântica de consistência de dados. No que tange ao tema de SDN, esse termo se refere a eventos na topologia de rede que geram atualização de dados de diferentes nós da rede, mas que podem eventualmente não ser atualizados em todos os nós controladores consistentemente. Isso implica na ocorrência de um período de tempo em que alguns nós da rede podem ler diferentes valores da mesma propriedade de controle, causando diversos problemas, como atualizações erradas das tabelas de fluxos (Reitblatt *et al.*, 2011). Entretanto, o enfoque deste trabalho é o projeto, desenvolvimento e implantação de um sistema protótipo para uma solução de provisionamento de circuitos virtuais em uma rede SDN/OpenFlow híbrida em um ambiente totalmente centralizado. Ambos os controladores ONOS e ODL têm como uma de suas principais características a sua utilização em ambientes distribuídos (Ferreira, 2016), assim como também são desenvolvidos integralmente em Java como o Floodlight, o que leva da mesma forma um a maior tempo de aprendizado de desenvolvimento. Assim sendo, como o projeto da solução VCFlow também não visa sua aplicação em ambientes distribuídos, então os controladores ONOS e ODL foram descartados para o desenvolvimento do sistema protótipo.

Apesar de tudo, alguns controladores centralizados como o Ryu são desenvolvidos almejando a aplicação em outros tipos ambientes de rede, como infraestruturas de rede de *backbone* de grandes provedores de serviço (Kreutz *et al.*, 2015). Esse controlador foi projetado baseado em uma arquitetura *single-threaded* e por esse motivo alcançou um desempenho menor nas avaliações realizadas. Apesar disso, os resultados para os testes de latência se mostraram adequados dentro do limite proposto inicialmente (*i.e.* o tempo de estabelecimento do circuito ser inferior a 1s), levando em consideração a complexidade da topologia. Entretanto, esse controlador oferece serviços de baixo nível (componentes de código como descoberta de topologia) que podem ser modificados e/ou estendidos de forma simples, permitindo ao desenvolvedor de rede construir novas aplicações baseadas nesses serviços de acordo com requisitos específicos (Khondoker *et al.*, 2014; Ferreira, 2016), assim como é o caso do VCFlow. Além disso, outros fatores relacionados às características disponibilizadas pelos controladores, tais como: (i) facilidade de desenvolvimento e boa documentação do arcabouço do controlador; (ii) menor tempo no aprendizado de desenvolvimento, levando em consideração o suporte a linguagens de programação; (iii) modularidade do sistema; (iv) suporte a GUI; (v) suporte a APIs REST; (vi) suporte a

virtualização (utilização de *switches* virtuais tanto para desenvolvimento quanto para uso em produção) e (vii) maturidade do projeto; levaram a escolha do controlador Ryu, para desenvolvimento da solução VCFlow (Khondoker *et al.*, 2014).

Desta forma, é descrito a seguir a arquitetura do sistema protótipo da solução VCFlow baseada na plataforma de controle Ryu.

5.1.4 Arquitetura VCFlow Baseada no Arcabouço do Controlador Ryu

A Figura 5.3 exibe a arquitetura da solução VCFlow, baseada no controlador Ryu. Mais especificamente, essa solução foi desenvolvida como uma aplicação do controlador Ryu. O VCFlow faz uso dos módulos OpenFlow *Parser/Serializer* e *Event Dispatcher*, e das bibliotecas de código disponibilizadas pelo arcabouço do Ryu. O VCFlow também estende o controlador com três funções principais que são descritas a seguir:

- *Hybrid Topology Discovery* (HTD): responsável pelo processo de descoberta de topologia por meio de pacotes LLDP "tageados", assim como descrito na Seção 3.4. Descoberta de Topologia em Redes SDN/OpenFlow Híbridas. Encapsula os pacotes LLDP com *tag* de VLAN e lida com os eventos de adição e deleção de enlaces na topologia de rede. Dessa forma, o tráfego dos circuitos virtuais L2VPN VLAN passam a ser tratados exclusivamente pelo *pipeline* OpenFlow e o tráfego de propósitos gerais pode ser tratado pelo *pipeline* tradicional, garantindo o isolamento do tráfego em uma topologia que adote *switches* SDN/OpenFlow híbridos.
- *Shortest Path Calculation* (SPC): responsável pelo processo de cálculo de menores caminhos entre *endpoints* na topologia de rede, baseando-se no algoritmo SPF. Após a descoberta de topologia os caminhos entre todos os *endpoints* são calculados. Gera-se, então, uma tabela de encaminhamento do VCFlow referente aos menores caminhos entre todos esses *endpoints* na topologia. Uma descrição detalhada do mecanismo de cálculo pode ser encontrada na Seção Apêndice B – Algoritmo *Shortest Path First* Adotado pelo VCFlow para Cálculo de Menor Caminho. Após a descoberta de topologia os caminhos entre todos os *endpoints* são calculados e por esse fator o SPC é responsável por lidar com o serviço de resiliência em conjunto com o HTD. Além disso, o SPC é também responsável pelos processos de VLAN *stitching* no núcleo da rede e de estabelecimento de circuitos virtuais fim a fim.
- *Loops Prevention*: responsável pelo processo de garantia de não ocorrência de *loops* na topologia de rede, baseando-se na técnica de *Split Horizon*, conforme descrito na Seção 3.5. Prevenção de *Loops* de Encaminhamento Utilizando Técnica de *Split Horizon*.

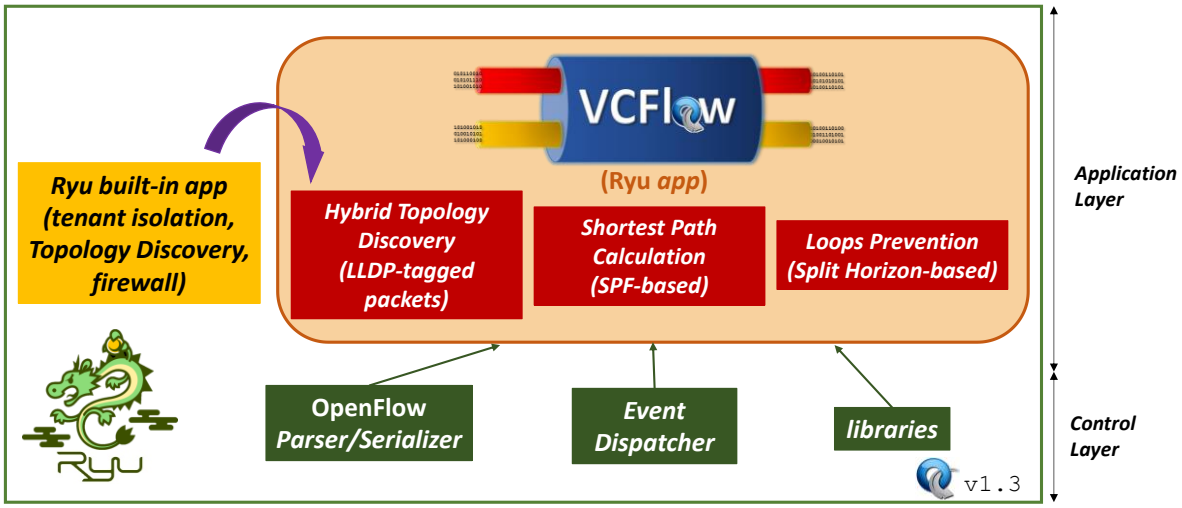


Figura 5.3: Arquitetura da Solução VCFlow.

Todos os resultados em sequência são apresentados em função da utilização do sistema protótipo VCFlow desenvolvido através do arcabouço do controlador Ryu.

5.2 Demonstração do Sistema Protótipo em Operação

O objetivo principal desta demonstração é exibir como a solução VCFlow suporta o provisionamento resiliente de serviços de estabelecimento de circuitos virtuais L2 em uma rede SDN/OpenFlow híbrida real em produção, como o *backbone* da Redecomep-Rio. A demonstração consistiu de um experimento focado em duas das funcionalidades principais providas pelo VCFlow: resiliência e suporte a circuitos virtuais com ID distintos em cada localidade atendida.

A Figura 5.4 exibe como que o mecanismo de resiliência/recuperação funciona de modo a garantir o serviço de provisionamento de circuitos virtuais mesmo na eventualidade de falhas de enlaces e o suporte ao estabelecimento de circuitos virtuais com VLANs distintas por localidade. A Figura 5.4 demonstra como a transmissão de pacotes UDP através da ferramenta *iperf2* ocorre sem falhas no primeiro caminho programado nos dispositivos de rede ($PoP-CBPF \leftrightarrow PoP-SIX/UNIRIO \leftrightarrow PoP-MIX \leftrightarrow PoP-CEFET-RJ \leftrightarrow PoP-UERJ$) e no caso de falha gerada no segundo de número 30, as entradas de fluxos associadas ao caminho primário são deletadas e as entradas de fluxos associadas ao caminho secundário ($PoP-CBPF \leftrightarrow PoP-RNP \leftrightarrow PoP-PUC \leftrightarrow PoP-MIX \leftrightarrow PoP-CEFET-RJ \leftrightarrow PoP-UERJ$) são instaladas. No caso da ocorrência da falha, é possível notar o mecanismo de recuperação de falhas, que adota os mecanismos de descoberta de topologia e cálculo de caminhos baseado em SPF em operação (conforme descrito nas Seções 3.4. Descoberta de Topologia em Redes SDN/OpenFlow Híbridas e 3.3. Escolha do Menor Caminho, VLAN *Stitching* e Estabelecimento do Circuito Virtual, respectivamente), leva entre um e dois segundos para recuperação do circuito virtual fim a fim. O intervalo de recuperação de circuitos é caracterizado na Seção 5.3.2. Cenário 2: Recuperação em Caso de Falhas. Além disso, é também possível observar como o circuito virtual é estabelecido para uma mesma sub-rede IP (10.0.0.0/24), a qual adota VLANs distintas em cada localidade onde o cliente é atendido (VLAN ID 10 no CBPF e VLAN ID

100 na UERJ). Esse circuito virtual opera de forma transparente para o cliente e também sem perdas de pacotes durante os intervalos de transferência em que não ocorreram falhas de enlace, apesar do encaminhamento nos *switches* PE ocorrer em modo OpenFlow híbrido e de serem realizadas ações de modificação de cabeçalho. Esse fato permitiu também validar o esquema de funcionamento do conjunto *switches* Cisco ASR 9000 em modo OpenFlow híbrido e solução VCFlow para sua implantação como serviço de provisionamento de circuitos virtuais na rede em produção da Redecomep-Rio.

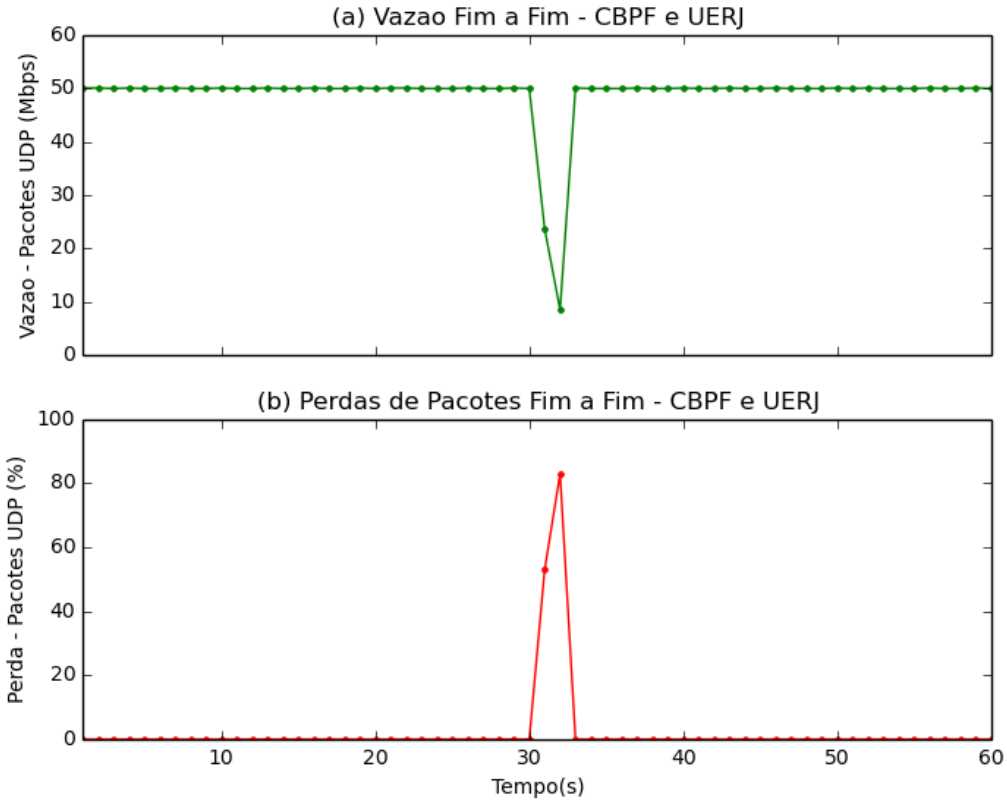


Figura 5.4: (a) Vazão de pacotes UDP e (b) taxa de perdas de pacotes UDP reportadas pela ferramenta iperf2 para o circuito virtual estabelecido entre CBPF e UERJ para demonstração da solução VCFlow em operação no backbone da Redecomep-Rio.

5.3 Caracterização do Intervalo de Tempo de Convergência do Sistema Protótipo

5.3.1 Cenário 1: Início do Funcionamento da Aplicação

Inicialmente, foi avaliado o intervalo de tempo de convergência do sistema VCFlow para o cenário de inicialização do sistema.

A Figura 5.5 exibe os intervalos de tempo médios de convergência fim a fim, de descoberta de topologia e de verificação de base de dados de configuração. Esses resultados são apresentados em função de cada topologia avaliada e da complexidade da rede, ou seja, em função do número de nós (*switches* PE) em cada topologia. Os resultados são expressos como média das 30 repetições do experimento e intervalo de confiança de 95%. O intervalo de convergência fim a fim foi calculado entre o instante de tempo de inicialização da aplicação e o instante em que o primeiro pacote ARP *Reply* é recebido pelo *host* origem,

assim como descrito na Seção 4.3.1. Cenário 1: Início do Funcionamento da Aplicação. O intervalo de descoberta de topologia foi calculado entre o último evento de adição de enlaces na topologia e o instante de inicialização do VCFlow. Já o intervalo de verificação de base de dados de configuração foi calculado entre o instante de detecção de entradas válidas nessa base e o instante de inicialização do VCFlow.

Cabe destacar que para que ocorra a validação de uma entrada na base de configuração, as interfaces dos *endpoints* associados a um circuito virtual qualquer cadastrado na base de configuração têm de estar registradas no controlador. Caso esse processo não ocorra, a entrada na base de configuração associada ao circuito virtual não é validada e uma entrada na base de dados de identificação não é criada para esse circuito. Assim não é possível que o processo de VLAN *stitching* ocorra no núcleo da rede e, por consequência, o circuito não pode ser estabelecido fim a fim, necessitando aguardar mais um período de cinco segundos para que ocorra uma nova verificação da base de configuração após o registro. Por exemplo, quando um *endpoint* não está associado ao controlador antes do instante de leitura da base de dados de configuração, as entradas que fazem referência a esse *endpoint* não podem ser validadas e o circuito virtual não pode ser estabelecido.

A Figura 5.6 exibe o intervalo de associação entre *switches* e controlador OpenFlow. Esse intervalo foi medido entre o instante de tempo de inicialização da aplicação e o instante em que o último *switch* da topologia se associa ao controlador. Os resultados exibidos nessa figura são apresentados como uma função de distribuição acumulada (CDF, do inglês *Cumulative Distribution Function)* das 30 repetições do experimento para as topologias avaliadas.

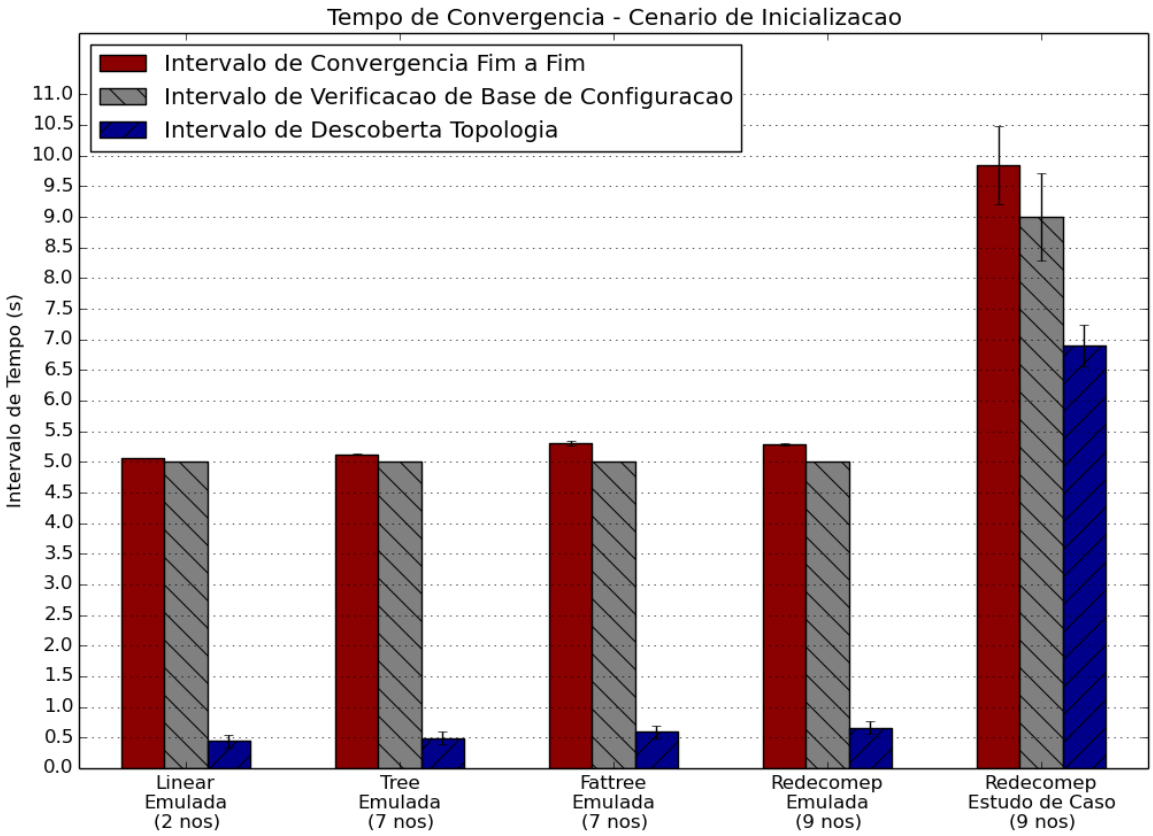


Figura 5.5: Intervalo de tempo de convergência para o cenário de inicialização do VCFlow.

Na Figura 5.5, observa-se que para todas as topologias emuladas o intervalo de tempo total de convergência fim a fim se mostrou similar, com resultados de 5,064s, 5,123s, 5,30s e 5,29s para as topologias Linear, *Tree*, *Fat-tree* e Redecomep, respectivamente. Pode-se notar que as topologias *Redecomep* e *Fat-tree* emuladas apresentaram resultados médios ligeiramente maiores do que as outras topologias emuladas por um único motivo em ambos os casos ocasionado por dois fatores distintos, sendo um em cada topologia. No primeiro caso da topologia Redecomep, há um maior número de nós e no segundo caso da topologia *Fat-tree* cada nó possui maior número de interfaces. Esses fatores fazem com que em ambos os casos o tempo de instalação das regras de prevenção de *loops* via mecanismo de *Split Horizon* aumente, conforme pode ser visto na Figura 5.7(b). No entanto, tanto os intervalos de descoberta de topologia (variando de 0,4s para topologia Linear a 0,7s para topologia Redecomep), quanto de verificação de base de configuração (5,00s para todos casos) também se mostraram similares independentemente da topologia emulada. Em comparação aos resultados de topologias emuladas, a topologia de estudo de caso da Redecomep apresentou um aumento de 86% no intervalo de convergência fim a fim (média de 9,8s) em relação a topologia da Redecomep emulada. Nesse mesmo caso comparativo, os resultados para a topologia Redecomep de estudo de caso para o intervalo de descoberta de topologia (média de 6,9s) e de verificação de base de configuração (média de 9,0s) tiveram aumentos de 945% e 80%, respectivamente. Isso ocorreu pelo fato da grande diferença do intervalo entre tentativas de associação entre os *switches* emulados e o controlador OpenFlow, e entre os *switches* Cisco ASR 9000 e o controlador OpenFlow. No caso dos *switches* virtuais *OpenVSwitch* (OvS) utilizados pelo emulador Mininet, o intervalo de associação é de um segundo, enquanto que no caso dos *switches* Cisco ASR 9000 esse intervalo é de oito

segundos. Como pode ser visto na Figura 5.6, o intervalo de associação de um segundo para os *switches* emulados pode ser percebido no pior dos casos de qualquer uma das topologias emuladas, e o intervalo de associação de oito segundos do roteador Cisco ASR 9000 também pode ser percebido no pior dos casos na topologia Redecomep Estudo de Caso. Então, conforme descrito anteriormente, o mecanismo de validação da base de dados de configuração depende estritamente dos *switches* associados ao controlador no momento de leitura da base e a verificação da base de configuração ocorre em intervalos de cinco segundos. Portanto, caso os *endpoints* do circuito virtual não estejam associados ao controlador no momento da primeira validação (cinco segundos após início da aplicação), consequentemente somente após mais um intervalo de verificação, isto é, dez segundos desde a inicialização, é que ocorrerá uma leitura válida desse circuito na base de configuração permitindo a continuação do processamento do fluxo. Como o circuito somente pode ser estabelecido quando a base de identificação possui uma entrada válida para esse circuito, então isso impacta diretamente no tempo total de convergência da rede para estabelecimento dos circuitos fim a fim, justificando as diferenças encontradas entre os casos emulados e de estudo de caso.

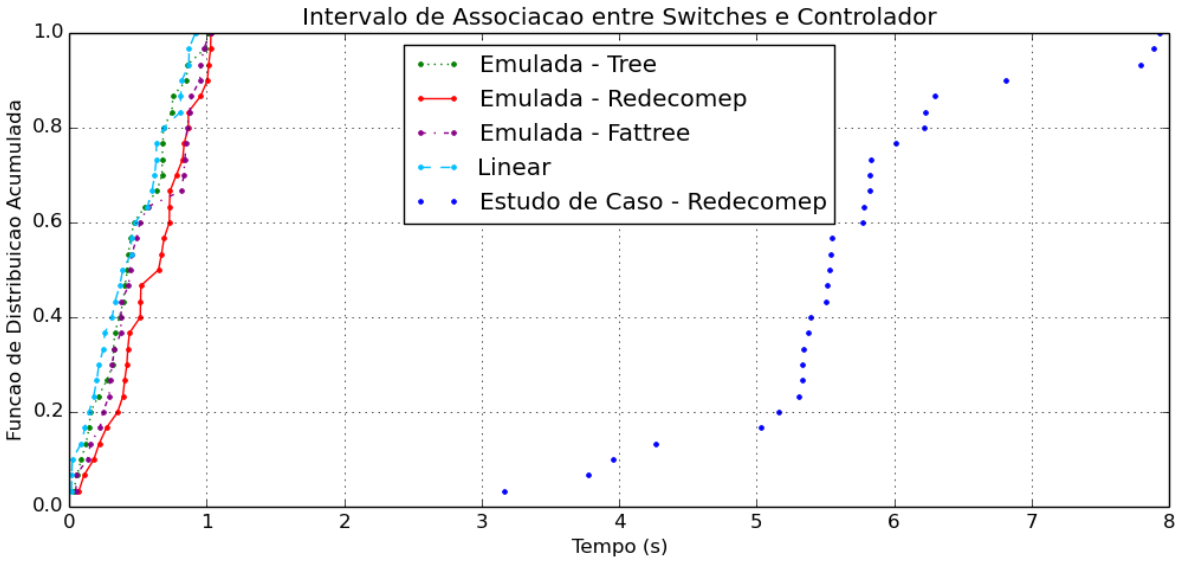


Figura 5.6: Intervalo de associação entre switches e controlador OpenFlow.

A Figura 5.7 mostra o (a) intervalo de tempo de convergência do algoritmo de *Shortest Path First* (SPF) e o (b) intervalo de tempo de convergência do mecanismo de prevenção de ocorrência de *loops* baseado na técnica de *Split Horizon* ou simplesmente intervalo de convergência do *Split Horizon*. O intervalo de convergência do algoritmo SPF foi calculado entre o último evento de adição de enlace gerado pelo arcabouço do controlador Ryu após a descoberta de topologia e o instante de atualização da tabela de encaminhamento do VCFlow. Já o intervalo de convergência do *Split Horizon* foi calculado entre o instante de recepção do *Packet-In* referente ao pacote VLAN do circuito virtual e o instante de encaminhamento do último pacote de *Flow-Mod* para descarte de pacotes direcionado ao último *switch* que pertence ao caminho do circuito virtual. Os resultados exibidos nessas imagens são apresentados como uma função de distribuição acumulada das 30 repetições do experimento, conforme descrito na Seção 4.3.1. Cenário 1: Início do Funcionamento da Aplicação, para cada topologia avaliada.

Na Figura 5.7(a), pode-se notar um pequeno aumento do intervalo de convergência do SPF entre os casos de topologia emulada Linear e todas as outras topologias. Isso ocorre pelo simples fato da diferença da complexidade da topologia entre os casos avaliados, sendo de apenas dois nós na topologia Linear, de sete nós nas topologias *Tree* e *Fat-tree* e de nove nós no caso da topologia Redecomep. Para o melhor dos experimentos realizados (convergência mais rápida) os resultados apresentados para cada topologia foram de: **i.** Emulada Linear: 0,262ms; **ii.** Emulada *Tree*: 0,764ms; **iii.** Emulada *Fat-tree*: 0,847ms; **iv.** Emulada Redecomep: 1,230ms; **v.** Estudo de Caso Redecomep: 1,001ms. É também possível identificar, claramente, que não há nenhuma diferença significativa entre os resultados referentes à topologia Redecomep emulada e de estudo de caso, visto que do ponto de vista do controlador OpenFlow as topologias são as mesmas.

Na Figura 5.7(b), para o experimento de melhor resultado para as topologias emuladas, os resultados foram: *Linear*: 29,888ms; *Tree* 56,469ms; *Fat-tree* 158,317ms; e Redecomep 199,240ms. De modo geral, pode-se observar uma pequena diferença entre os intervalos de convergência do *Split Horizon* para a topologia Linear em relação a todas as outras topologias emuladas. Isso ocorreu pelo simples fato de que na topologia Linear existe uma única interface por direção do caminho do circuito para a qual é instalada uma regra de descarte pelo mecanismo de prevenção de *loops*. É possível notar também uma diferença entre os resultados referentes à topologia *Tree* e as topologias *Fat-tree* e Redecomep Emulada. Essa situação foi ocasionada pelo fato da topologia *Fat-tree* possuir maior número de interfaces por nó em relação à topologia *Tree* e pelo fato da topologia Redecomep possuir maior complexidade entre todas as topologias avaliadas. Ambos os fatos acarretam maior número de mensagens de *Flow-Mod* necessárias para descartar pacotes nas interfaces dos *switches* que não pertencem ao caminho do circuito virtual e assim garantir a prevenção de ocorrência de *loops*. Além disso, é possível também observar uma pequena diferença de 131,364ms no melhor dos casos entre as topologias Redecomep Emulada e Redecomep de Estudo de Caso, onde a topologia de estudo de caso apresentou o maior resultado de 330,604ms entre os melhores casos. Isso ocorre devido a dois fatores distintos. O primeiro deles é devido à maior latência entre os *switches* Cisco ASR 9000 e o controlador OpenFlow em relação à latência entre os *switches* virtuais OvS (emulados pelo Mininet) e o controlador OpenFlow, ambos os últimos dispositivos tendo sido executados na mesma máquina. O segundo fator é o fato das regras demorarem mais para serem instaladas nos *switches* Cisco ASR 9000 do que nos *switches* OvS após o recebimento das mensagens de controle do tipo *Flow-Mod* provenientes do controlador OF.

Esse fato das regras demorarem mais para serem instaladas nos *switches* Cisco ASR 9000 ocorre devido a esses equipamentos não suportarem as funcionalidades de *switches* OpenFlow diretamente em seu sistema operacional[21]. Essas funcionalidades são garantidas através de uma aplicação que roda nativamente sobre o sistema operacional IOS-XR desses equipamentos, e é responsável por emular um ou mais *switches* virtuais OpenFlow. Mais especificamente, essa aplicação roda sobre o Cisco *Open Network Environment* (ONE) *Plataform Kit* ou onePK. O onePK é um arcabouço que inclui um conjunto de APIs e

[21] Experimentos não exibidos nesse documento comprovaram o fato descrito, porém análises posteriores carecem de serem realizadas para identificar a diferença de desempenho quanto a instalação de regras entre os *switches* Cisco ASR 9000 e outros equipamentos que suportem o protocolo OpenFlow.

bibliotecas de programação proprietárias, permitindo aos programadores desenvolverem aplicações que se integrem a um ambiente de equipamentos Cisco. Essa aplicação, que provê as funcionalidades do protocolo OpenFlow, é responsável por se conectar ao controlador OpenFlow e converter as suas mensagens para a API onePK correspondente. Desse modo, como há uma camada de virtualização responsável por traduzir as mensagens OpenFlow em um formato proprietário suportado pelo onePK, para que então as regras sejam efetivamente instaladas em *hardware*, os *switches* Cisco ASR 9000 demoram mais para iniciarem o encaminhamento do tráfego.

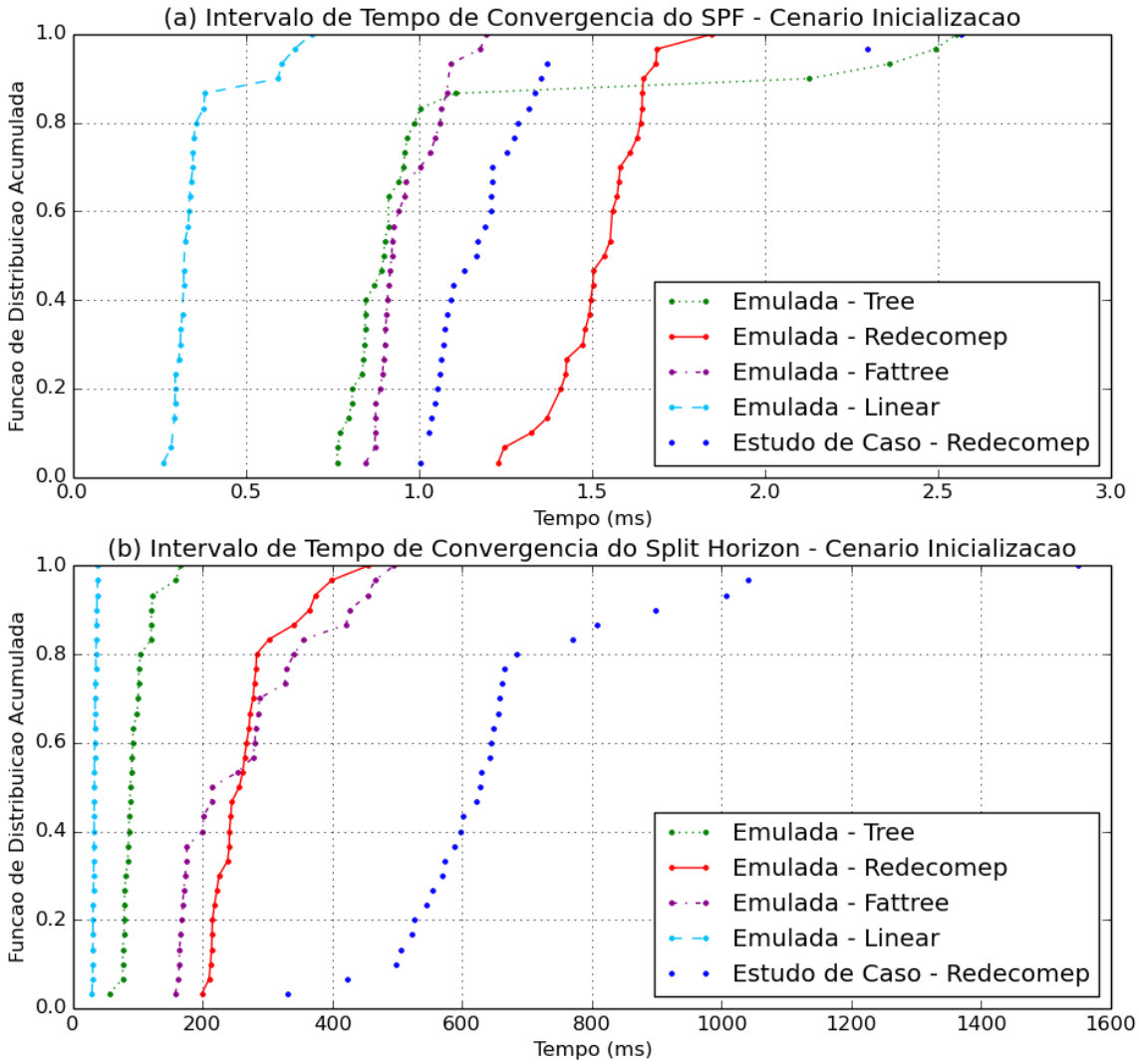


Figura 5.7: (a) Intervalo de tempo de convergência do algoritmo do Shortest Path First (SPF) e (b) Intervalo de tempo de convergência do mecanismo de prevenção de loops baseado no Split Horizon para cenário de inicialização do VCFlow.

5.3.2 Cenário 2: Recuperação em Caso de Falhas

Após a avaliação do VCFlow no cenário de inicialização, foram realizadas as avaliações do tempo de convergência do sistema VCFlow para o cenário de recuperação em

caso de falhas. Esse cenário é o de maior relevância durante a operação da rede, tendo em vista sua maior probabilidade de ocorrência.

A Figura 5.8 exibe o intervalo de tempo médio de recuperação fim a fim medido entre o instante de recepção do último pacote ICMP *Echo Reply* pelo *host* de origem antes da primeira perda de pacotes e o primeiro pacote ICMP *Echo Reply* recebido pelo *host* origem após a primeira perda, assim como descrito na Seção 4.3.2. Cenário 2: Recuperação em Caso de Falhas. Esse resultado é apresentado em função de cada topologia avaliada e da complexidade da rede. O resultado é expresso como média das 30 repetições do experimento e intervalo de confiança de 95%.

A Figura 5.9 mostra o (a) intervalo de tempo de convergência do algoritmo SPF, o (b) intervalo de tempo de convergência do mecanismo de prevenção de ocorrência de *loops* após a detecção de alteração da topologia e o (c) intervalo de tempo de detecção de alteração da topologia. O intervalo de convergência do algoritmo SPF foi medido entre o evento de deleção de enlace gerado pelo arcabouço do controlador Ryu após a detecção de alteração de topologia e o instante de atualização da tabela de encaminhamento do VCFlow. O intervalo de convergência do *Split Horizon*, de modo similar ao apresentado na Figura 5.7(b), foi medido entre o instante de recepção do *Packet-In* (referente ao pacote VLAN do circuito virtual recebido após a detecção de alteração de topologia e após o cálculo da tabela de encaminhamento pelo SPF), e o instante de encaminhamento do último pacote de *Flow-Mod* (com regras para descarte de pacotes) ao último *switch* que pertence ao caminho do circuito virtual. O intervalo de detecção de alteração de topologia foi medido entre a recepção da mensagem de controle OpenFlow de *PortStatus*, que indica a alteração do estado de uma porta no *switch* OpenFlow, e o evento de deleção de enlace gerado pelo arcabouço do controlador Ryu. Os resultados exibidos nessas imagens são apresentados como uma função de distribuição acumulada das 30 repetições do experimento para cada topologia avaliada, conforme descrito na Seção 4.3.2. Cenário 2: Recuperação em Caso de Falhas.

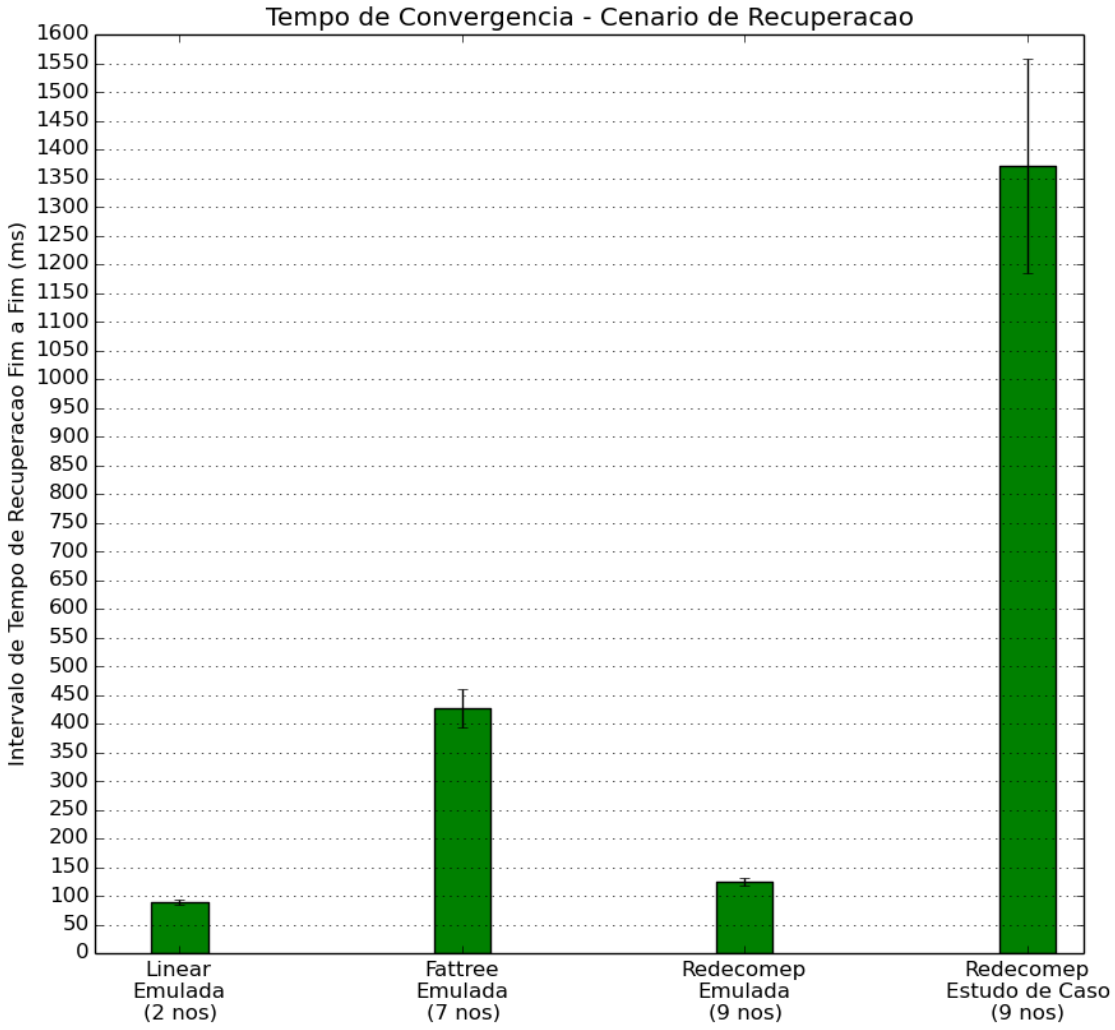


Figura 5.8: Intervalo de tempo de convergência do VCFlow para o cenário de recuperação em caso de falhas.

Na Figura 5.8, pode-se observar os seguintes resultados de recuperação médios para cada uma das topologias avaliadas: **i.** Linear: 89ms; **ii.** *Fat-tree*: 430ms; **iii.** Redecomep Emulada: 124ms; **iv**. Redecomep Estudo de Caso: $1{,}4 \times 10^3$ms. Dentre as topologias emuladas, pode-se notar um aumento de 380% entre a topologia *Fat-tree* e a Linear, e um aumento de 243% entre a topologia *Fat-tree* e a Redecomep. Essa situação de maior tempo de recuperação da topologia *Fat-tree* dentre as topologias emuladas ocorreu devido principalmente ao fato dos *switches* na topologia *Fat-tree* possuírem grande quantidade de interfaces cada. Esse fato implica em maior tempo de instalação de regras de descarte de pacotes pelo mecanismo de *Split Horizon* para garantir a não ocorrência de *loops*, conforme pode ser visto na Figura 5.9(b). Assim como houve uma significativa diferença entre o intervalo de recuperação em caso de falhas médio entre a topologia *Fat-tree* e as outras topologias emuladas, também houve um aumento de 1002% entre a topologia Redecomep de estudo de caso e a topologia *Fat-tree* emulada. Esse grande aumento foi ocasionado em razão da latência entre o controlador OpenFlow e os *switches* OpenFlow Cisco ASR 9000 ser maior do que a latência entre o controlador e o *switch* virtual OvS (adotado pelo emulador Mininet), e, principalmente, em razão do maior tempo que cada *switch* OpenFlow Cisco ASR 9000 leva para instalar as regras na tabela de fluxo após receber a mensagem de *Flow-Mod* do controlador. Isso implica em maior tempo para instalar as regras de prevenção de

ocorrência de *loops* em todos os *switches* da topologia como pode ser visto na Figura 5.9(b). Como consequência esse fato acarreta em aumento no tempo de recuperação de circuitos fim a fim em caso de falhas. Além de tudo, o resultado de recuperação médio obtido para a topologia Redecomep de estudo de caso também condiz com os resultados apresentados na demonstração e que podem ser vistos na Figura 5.4, onde pode-se notar um intervalo de recuperação entre 1,00s e 2,00s.

Enfim, nas Figura 5.9(a) e Figura 5.9(c) pode-se reparar que não há nenhuma diferença significativa para os intervalos de convergência do SPF e de detecção de alteração de topologia para todas as topologias avaliadas. Sendo que em ambas as figuras pode-se notar um melhor desempenho de recuperação do VCFlow na topologia Redecomep de estudo de caso do que na emulada. Pode-se notar também a similaridade entre os resultados para o intervalo de convergência do SPF para a topologia Redecomep Estudo de Caso para ambos os cenários de inicialização e de recuperação. Nessa topologia, em 100% dos experimentos realizados os resultados ficaram abaixo de 2,5ms tanto no cenário de inicialização quanto no cenário de recuperação, como pode ser observado na Figura 5.7(a) e na Figura 5.9(a), respectivamente. Apesar do fato de que em 93% dos experimentos da topologia de estudo de caso os resultados estarem entre 1,00ms e 1,36ms no cenário de inicialização, e em 100% dos resultados estarem entre 1,8ms e 2,4ms no cenário de recuperação.

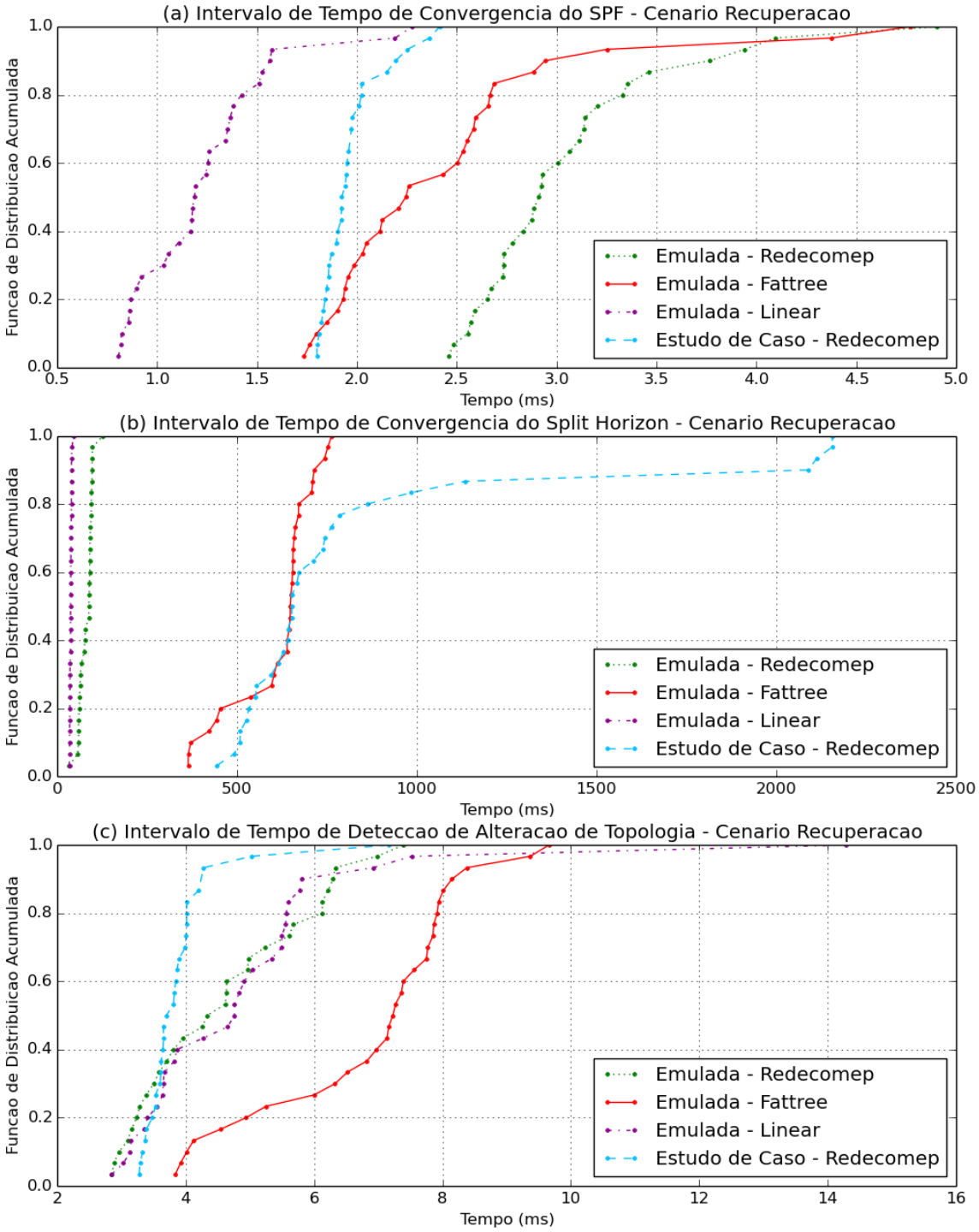


Figura 5.9: (a) Intervalo de tempo de convergência do algoritmo de SPF, (b) Intervalo de tempo de convergência do mecanismo de prevenção de ocorrência de loops após a detecção de alteração da topologia e (c) Intervalo de tempo de detecção de alteração da topologia para cenário de recuperação em caso de falhas.

5.4 Caracterização do Intervalo de Tempo de Processamento das Mensagens de Sinalização e de Estabelecimento de Circuitos Fim a Fim

Por fim, foram realizadas as avaliações do intervalo de tempo de processamento das mensagens de sinalização do tipo *Packet-In* e de tempo de estabelecimento de circuitos fim a fim.

5.4.1 Intervalo de Tempo de Processamento das Mensagens de Sinalização OpenFlow

A Figura 5.10 mostra o (a) intervalo de tempo de processamento das mensagens de controle OF *Packet-In* na direção de *switch* PE 1 (onde o *host* origem está conectado) para o *switch* PE 2 (onde o *host* destino está conectado), e também exibe o (b) intervalo de tempo de processamento das mensagens de controle OF *Packet-In* na direção de *switch* PE 2 para *switch* PE 1. Uma visão geral da metodologia adotada pode ser visualizada na Figura 4.15. O intervalo de tempo de processamento das mensagens de controle na direção de *switch* PE 1 para *switch* PE 2, ou simplesmente direção (1) => (2), foi medido entre o instante de recepção do *Packet-In* (referente ao pacote VLAN do circuito virtual recebido pelo *switch* PE 1), e o instante de encaminhamento do pacote de *Flow-Mod* destinado ao último *switch* nessa direção do circuito virtual, isto é, o *switch* PE 2. O intervalo de tempo de processamento na direção de *switch* PE 2 para *switch* PE 1 ou direção (2) => (1) foi avaliado de modo inverso, ou seja, foi avaliado entre o instante de recepção do *Packet-In* proveniente do *switch* PE 2 e o instante de encaminhamento do pacote *Flow-Mod* destinado ao *switch* PE 1. Uma ilustração generalizada desse método de medição, e que se aplica a ambos os casos de direção (1) => (2) e (2) => (1), pode ser visto na Figura 4.14.

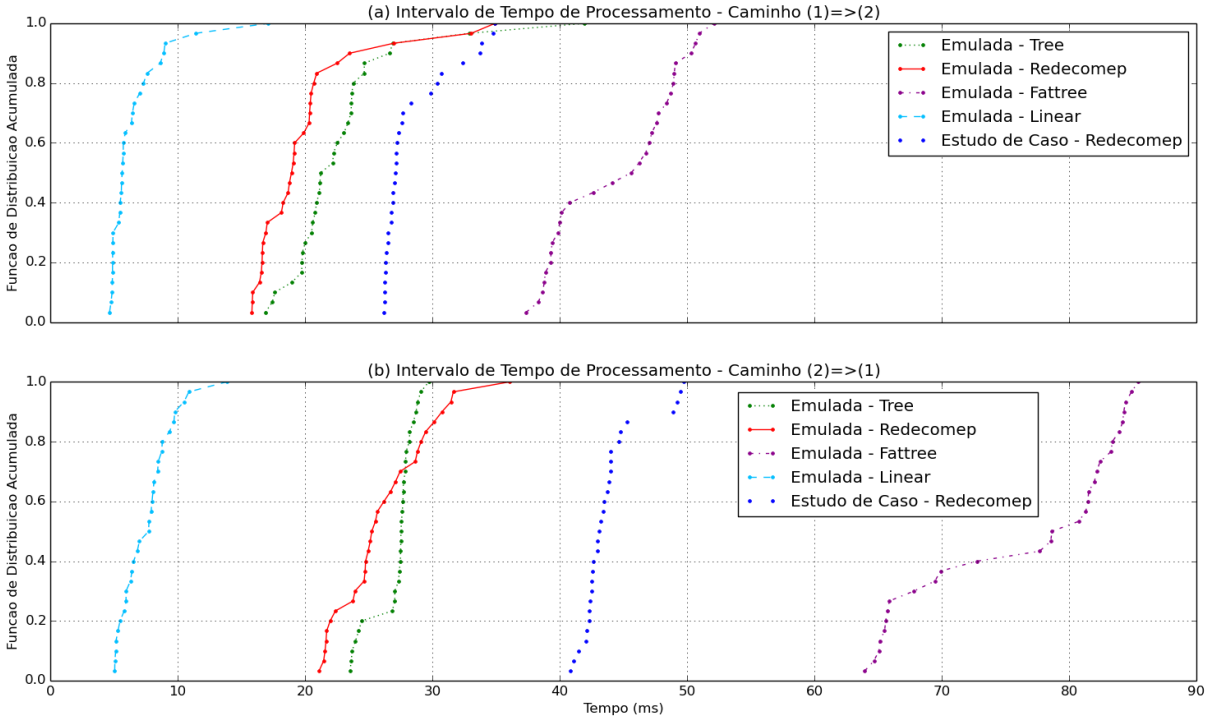


Figura 5.10: Intervalo de tempo de processamento das mensagens de controle OpenFlow Packet-In (a) na direção de switch PE 1 para switch PE 2 e (b) na direção de switch PE 2 para switch PE 1.

Pode-se observar que pela FIGURA(b) que o intervalo de tempo de processamento na direção (2) => (1) foi maior em relação à direção (1) => (2). Esse fato ocorre devido a como o algoritmo de estabelecimento de circuito virtual e VLAN *stitching* foi implementado de modo sequencial, conforme pode ser visto na **Erro! Fonte de referência não encontrada.**. Nesse caso, primeiramente verifica-se se o pacote é referente a algum dos circuitos na direção (1) => (2) para depois verificar se o pacote pertence a algum dos circuitos na direção (2) => (1).

5.4.2 Intervalo de Tempo Estabelecimento de Circuitos Fim a Fim

A Figura 5.11 exibe o intervalo de tempo médio de estabelecimento de circuitos fim a fim determinado entre o instante de encaminhamento do primeiro pacote ARP *Request* pelo *host* origem e o instante em que o primeiro pacote ARP *Reply* é recebido como resposta pelo *host* origem, dado que os processos de verificação de base de dados de configuração, de descoberta de topologia e de cálculo de tabelas de encaminhamento pelo SPF já ocorreram. A metodologia de teste adotada nesse experimento pode ser encontrada na Seção 4.4. Caracterização do Intervalo de Tempo de Processamento das Mensagens de Sinalização e de Estabelecimento de Circuitos Fim a Fim. Esse resultado é apresentado em função de cada topologia avaliada e da complexidade da rede. O resultado é expresso como média das 30 repetições do experimento e intervalo de confiança de 95%.

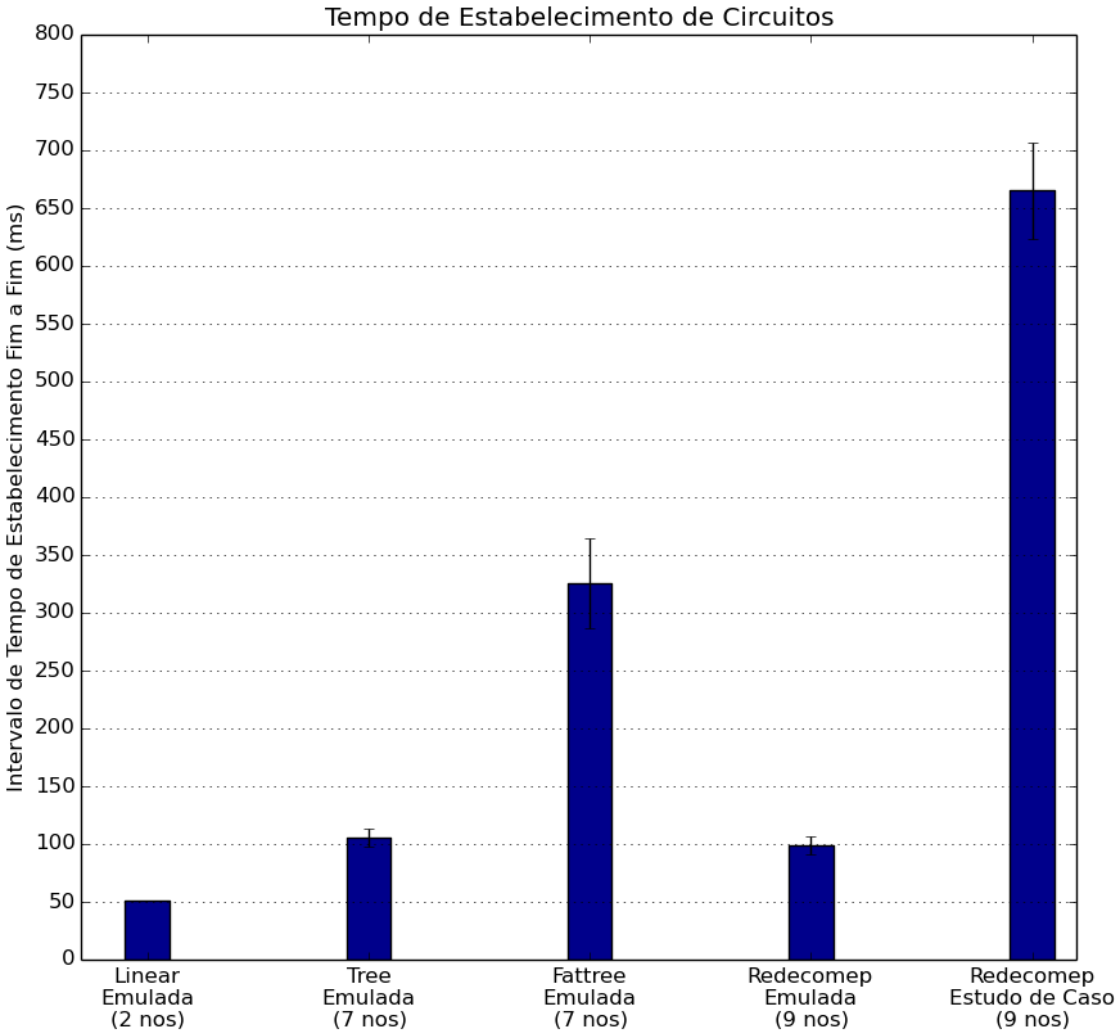


Figura 5.11: Intervalo de tempo de estabelecimento de circuitos virtuais fim a fim pelo VCFlow.

Na Figura 5.11, pode-se observar os seguintes resultados de tempo de estabelecimento médio para cada uma das topologias avaliadas: **i.** Linear: 50,74ms; **ii.** *Tree*: 106ms; **iii.** *Fat-tree*: $0{,}33 \times 10^3$ms; **iv.** Redecomep Emulada: 99ms; **v**. Redecomep Estudo de Caso: $0{,}67 \times 10^3$ms. Pode-se notar um comportamento de aumento do intervalo de estabelecimento fim a fim entre as topologias avaliadas similar aos resultados obtidos para o cenário de recuperação em caso de falhas, que podem ser vistos na Figura 5.8. Pode-se observar, dentre as topologias emuladas, um aumento do intervalo de tempo de estabelecimento fim a fim de 542% entre *Fat-tree* e Linear, um aumento de 208% entre *Fat-tree* e *Tree* e um aumento de 228% entre *Fat-tree* e Redecomep. Esse fato, assim como no caso do tempo de convergência no cenário de recuperação em caso de falhas, ocorreu devido, principalmente, aos *switches* na topologia *Fat-tree* possuírem grande quantidade de interfaces cada. Isso implica em maior tempo para instalar as regras para garantir a não ocorrência de *loops*, pois essas regras são instaladas logo após as regras voltadas ao processo de VLAN *stitching* e estabelecimento de circuitos virtual fim a fim em cada um dos *switches* do caminho do circuito. As etapas de processamento das mensagens de controle em que as regras são instaladas nos *switches* podem ser vistas no fluxograma de código do processo de VLAN *stitching* na **Erro! Fonte de referência não encontrada.** da Seção 3.3. Escolha do

Menor Caminho, VLAN *Stitching* e Estabelecimento do Circuito Virtual. As Figura 5.10(a) e Figura 5.10(b) demonstram justamente que o fato dos *switche*s virtuais possuírem maior número de interfaces cada implica em aumento do tempo total de processamento das mensagens de controle.

Além disso, é possível observar também o aumento de 571% entre a topologia Redecomep de Estudo de Caso e a topologia Redecomep Emulada. Esse aumento, no entanto, é motivado por dois motivos principais: (i) a latência entre controlador OpenFlow e os *switches* OpenFlow Cisco ASR 9000 é maior do que a latência entre o controlador e o *switch* virtual OvS adotado pelo emulador Mininet; (ii) cada *switch* OpenFlow Cisco ASR 9000 leva mais tempo do que os *switches* virtuais OvS para instalar as regras na tabela de fluxo após receber a mensagem de *Flow-Mod* do controlador. Ambos os fatos são fatores que influenciaram a diferença nos resultados obtidos para o cenário de recuperação em caso de falhas, e por esse motivo carecem de avaliações mais apuradas que não foram realizadas nesse trabalho.

Capítulo 6

6 CONCLUSÃO

Este trabalho endereça o problema de provisionamento de circuitos virtuais principalmente voltados para redes de *backbone* IP. Atualmente, os operadores de rede de provedores de serviços podem levar horas, dias e até meses para testar e validar a implementação desse tipo de circuito ao utilizar técnicas de L2VPNs tradicionais, que variam muito de fabricante para fabricante de equipamentos de rede e de plataforma para plataforma também. Dependendo muitas das vezes de políticas locais desses provedores de serviços, da técnica adotada, da extensão da rede onde o circuito visa ser implantado e devido à forma distribuída como as configurações de circuito são realizadas pelos diversos elementos de rede esse tempo pode variar ainda mais. Aliado a isso, problemas relacionados à ineficiência de utilização da infraestrutura de rede causada pelo *overhead* que essas técnicas geram e dificuldades de *troubleshooting* relacionadas a forma como cada técnica é implementada com informações espalhadas pelos diversos elementos de rede são também um grande problema para a operação de redes IP.

Então, visando mitigar esses problemas, este trabalho propõe o desenvolvimento de uma solução de provisionamento de circuitos virtuais tomando como base recentes trabalhos baseados no paradigma de Redes Definidas por *Software* (SDN). Porém, diferentemente das soluções SDN tradicionais, foi desenvolvido um sistema que visa sua utilização em redes SDN/OpenFlow híbridas. Nas redes SDN/OpenFlow híbridas, é possível aliar os princípios de resiliência de redes tradicionais ao princípio de programabilidade das SDNs. Nesse caso, o encaminhamento dos pacotes dos circuitos virtuais pode compartilhar os mesmos recursos de rede (como enlaces físicos e roteadores/*switches*) já disponíveis e em operação adotados pelas redes tradicionais. Isso permite que o tráfego IP de propósitos gerais seja tratado pelo *pipeline* tradicional dos dispositivos de rede, com todas as suas vantagens, e o tráfego de circuitos virtuais seja tratado somente pelo *pipeline* OpenFlow.

Foi desenvolvido uma solução denominada de ***Virtual Circuits Flow*** **(VCFlow)** para serviços de transporte de camada 2 visando o provisionamento de circuitos virtuais dinâmicos e resilientes do tipo L2VPN *Ethernet*/VLAN em redes de *backbone* SDN/OpenFlow híbridas. O VCFlow é uma solução aberta (desenvolvida sob o conceito de *software* livre) que possui como principal vantagem o provimento de um serviço de

conectividade privada de alta capacidade baseado na agregação das vantagens oferecidas por uma tecnologia bem difundida atualmente como o *Ethernet*, operando em conjunto com técnicas de comutação permitidas pelo novo paradigma de SDN/OpenFlow. Este trabalho foi motivado pelo propósito de desenvolver alternativas e melhorias para os serviços de alta capacidade de encaminhamento provido pelo projeto Redecomep-Rio, e tomou como estudo de caso a caracterização do funcionamento do VCFlow em sua rede em produção. Além disso, outra grande vantagem deste trabalho é permitir o planejamento de uma migração da operação da rede nos meios tradicionais para uma rede com base integralmente em tecnologia SDN.

O VCFlow faz uso de uma das principais características dos controladores SDN/OpenFlow que é possuir uma visão holística da rede. Isto é, o controlador SDN possui conhecimento de todos os dispositivos de rede e as ligações entre eles. Isso permite ao controlador criar e manter uma visão centralizada da topologia e calcular os caminhos de encaminhamento de forma simplificada. O algoritmo SPF é adotado para calcular os caminhos entre cada *endpoint* dos circuitos virtuais e adota um mecanismo de prevenção de ocorrência de *loops* baseado na técnica de *Split Horizon*. O VCFlow também opera como um *learning switch*, de forma que o endereço MAC do *host* de origem é associado a interface de entrada do *switch* PE onde ele está conectado e as regras para estabelecimento de circuitos virtuais se baseiam nesse endereço MAC.

O VCFlow minimiza a intervenção humana (outro grande fator gerador de problemas em redes IP causadas por erros de configuração, por exemplo) através da adoção de uma base de dados de configuração centralizada, a partir da qual o operador de rede necessita informar: (i) os *endpoints* do circuito virtual; (ii) as interfaces dos *endpoints* que fazem parte do circuito; e (iii) os ID de VLAN que são adotados em cada *endpoint*. O VCFlow também possui como características principais: (i) a capacidade de evitar a complexidade de protocolos de encapsulamento para estabelecimento de circuitos fim a fim, uma vez que este adota o mecanismo de comutação baseado nas próprias *tags* de VLAN dos pacotes dos circuitos; (ii) o suporte ao transporte de até 4094 circuitos virtuais simultaneamente; e (iii) o suporte ao estabelecimento de circuitos virtuais com ID distintos em cada localidade atendida de modo transparente para o cliente, através da alteração dos VLAN ID dos circuitos em cada *endpoint* por meio da ação *Set Field* do OpenFlow.

Este trabalho teve também o objetivo de avaliar e identificar o controlador mais adequado para a implantação do VCFlow. Os controladores SDN/OpenFlow avaliados foram: ODL, ONOS, Floodlight e Ryu. As avaliações foram realizadas por meio de análise comparativa de desempenho, de suas características e funcionalidades. Tendo em vista as limitações técnicas impostas pelo desenvolvimento para cada tipo de controlador, suas características (*e.g.* facilidade de desenvolvimento, qualidade da documentação, suporte a interface gráfica de usuário, a APIs REST e a virtualização), e por apresentar um desempenho de latência dentro dos limites estabelecidos pelo escopo deste trabalho, optou-se pelo uso do arcabouço do controlador Ryu.

O VCFlow foi avaliado em duas fases principais: (i) demonstração do funcionamento da solução em um caso real de uso na Redecomep-Rio/Rede-Rio; e (ii) caracterização dos intervalos de tempo de convergência, de processamento de mensagens de sinalização e controle, e de estabelecimento de circuitos virtuais fim a fim. Inicialmente, visando garantir a elegibilidade e viabilidade do VCFlow, a operação de estabelecimento de circuitos virtuais

foi realizada em uma rede paralela à rede de produção da Redecomep-Rio/Rede-Rio. O experimento foi realizado efetuando-se uma medição de vazão de pacotes UDP a 50Mbps através da ferramenta *iperf2* em um circuito virtual operando sobre o *backbone* da Redecomep-Rio entre os PoP-UERJ e PoP-CBPF. Para verificar o mecanismo de resiliência do VCFlow foi gerada uma falha de enlace no caminho do circuito virtual. O experimento permitiu verificar o funcionamento do circuito virtual fim a fim em operação na rede em produção da Redecomep-Rio, medir o intervalo de tempo de recuperação, verificar que não houveram perdas de pacotes durante a transferência nos intervalos de circuitos estabelecidos e também o mecanismo de recuperação em caso de falhas.

Posteriormente, foi caracterizado o funcionamento do VCFlow, por meio de experimentos que buscaram caracterizar o tempo de convergência para o cenário de inicialização da aplicação e de recuperação em caso de falhas. Em ambos os cenários, foram adotadas topologias tanto emuladas quanto no caso real da Redecomep-Rio/Rede-Rio. Os resultados das medições para cada um dos cenários propostos foram apresentados na Seção 5.3. Caracterização do Intervalo de Tempo de Convergência do Sistema Protótipo. No cenário de inicialização do VCFlow, foram avaliadas cinco topologias: Linear Emulada, *Tree* Emulada, *Fat-tree* Emulada, Redecomep Emulada e Redecomep Estudo de Caso. A convergência do VCFlow para o cenário de inicialização ocorreu em média em 9,8s para a topologia Redecomep de Estudo de Caso, enquanto que na topologia Redecomep Emulada esse intervalo foi de 5,29s. Essa diferença pode ser explicada pelo intervalo de associação dos *switches* com o controlador; i.e., esta é oito vezes superior no cenário de estudo de caso real do que no cenário emulado, ocasionado pelas características de operação do *switch*/roteador Cisco ASR 9000 (equipamentos da infraestrutura da RedeRio/FAPERJ e Redecomep-Rio, em funcionamento desde 2014). Esse fato trouxe à tona um problema: a falta de documentação desses equipamentos (assunto discutido mais adiante nesta seção).

Assim como no cenário de inicialização, o VCFlow foi avaliado quanto ao tempo de convergência em eventos de recuperação em caso de falhas na topologia (*e.g.*, rompimento das conexões, falhas das interfaces de comunicação etc.). Para isso foram avaliadas quatro topologias: Linear Emulada, *Fat-tree* Emulada, Redecomep Emulada e Redecomep Estudo de Caso. Nesse caso, pôde-se verificar que o intervalo de tempo de recuperação em caso de falhas foi de 1,4s na topologia Redecomep de Estudo de caso, enquanto que na topologia Redecomep Emulada esse resultado foi de 124ms. Entretanto, essa diferença ocorreu devido aos roteadores modelo Cisco ASR 9000 levarem mais tempo do que os *switches* emulados para instalar as regras na tabela de fluxos, após recepção das mensagens de *Flow-Mod* e pela latência entre os equipamentos Cisco e o controlador OpenFlow ser maior do que a latência entre os *switches* emulados e o controlador.

Como avaliação final, os tempos de estabelecimento de circuito fim a fim e o tempo de processamento das mensagens de sinalização *Packet-In* foram medidos para as cinco topologias: Linear Emulada, *Tree* Emulada, *Fat-tree* Emulada, Redecomep Emulada e Redecomep Estudo de Caso. Desse modo, foi possível verificar o intervalo de tempo total em que o VCFlow é capaz de estabelecer um circuito virtual fim a fim, dado que o sistema convergiu, ou seja, a topologia foi totalmente descoberta e a tabela de encaminhamento calculada. Foi possível também verificar o intervalo tempo de estabelecimento fim a fim é dominado pelo tempo de processamento das mensagens de controle OpenFlow. O intervalo de tempo de estabelecimento médio foi de $0{,}67 \times 10^3$ms para a topologia Redecomep Estudo

de Caso e de 99ms para a topologia Redecomep Emulada. Essa diferença é também justificada pelo tempo de instalação das regras na tabela de fluxo e pela latência de comunicação entre controlador e *switches*. Cabe ressaltar que no pior caso, o tempo de processamento das mensagens de controle representa 7,5% do tempo total de estabelecimento médio de circuito fim a fim, para a topologia Redecomep de Estudo de Caso (*i.e.* a Redecomep-Rio real).

Contudo, apesar dos resultados apresentados terem sido satisfatórios outras técnicas para determinação do tempo de resiliência da rede poderiam ter sido utilizadas. Dentre elas pode-se citar medições de tráfego baseadas em fluxos de testes TCP e UDP através de *streams*[22] de dados TCP/UDP emulando tráfego POP3, HTTP ou DNS, por exemplo. Porém, ferramentas capazes de gerar esse tipo de tráfego de dados, como *IxChariot Endpoint servers* (Ixia, 2017), são proprietárias e têm um custo alto para o projeto. Assim, optou-se por utilizar ferramentas confiáveis de código livre e de mais simples utilização como o FPING, *arp-scan* e o *tcpdump*.

Cabe destacar também que no decorrer de todo o processo de desenvolvimento do VCFlow foi percebido que um dos principais problemas foi a falta ou insuficiência de documentação técnica dos roteadores modelo Cisco ASR 9000. Nesse caso, parâmetros básicos de funcionamento (*e.g.* intervalos de tempo de associação) e a forma correta de configuração dos *switches* virtuais OpenFlow para sua ativação nesses equipamentos são mal ou não são documentados. Esse fato gera problemas adicionais ao desenvolvedor de rede durante o processo de implantação de um novo serviço (aplicação), pois esses parâmetros podem influenciar negativamente no funcionamento da rede. Isso faz com que esse profissional necessite empregar tempo adicional de desenvolvimento para determinação desses parâmetros e assim seja capaz de determinar as características do serviço desenvolvido.

Na realidade, foi necessário despender bastante tempo do projeto na compreensão do funcionamento do *switch* virtual OpenFlow nos roteadores Cisco ASR 9000. Tempo adicional àquele de pesquisa para a definição da melhor forma de operação desses *switches* virtuais em uma topologia SDN/OpenFlow híbrida. Essa forma de operação deveria levar em consideração que o processamento do *pipeline* OpenFlow não deveria interferir no processamento do *pipeline* tradicional. Além disso, o fato dos *switches* virtuais não operarem integralmente em conformidade com a especificação do protocolo OpenFlow e como o fabricante Cisco implementa esse protocolo de forma distribuída em múltiplos *pipelines* distintos, somente foi possível utilizar um conjunto limitado de correspondências e ações nas tabelas de fluxos pelo VCFlow.

A limitação de não conformidade também não permitiu a agregação de outras funcionalidades ao VCFlow. Por exemplo, a criação de circuitos virtuais com garantias de banda em um regime de BoD (*Bandwidth on Demand* - Banda sob Demanda) não foi possível devido ao roteador Cisco ASR9000 não implementar múltiplas tabelas no *pipeline* OpenFlow (como o caso da tabela de medição *Meter Table)*. Entretanto, espera-se que com a evolução do protocolo OpenFlow e do sistema operacional adotado por esse modelo de

[22] *Stream* é um fluxo de dados em um sistema computacional.

equipamentos, novas funcionalidade do protocolo sejam adicionadas a esses dispositivos, permitindo também a complementação de ações ao VCFlow.

Como sugestão para trabalhos futuros e que poderiam ser comparadas a este trabalho, é possível realizar medições do intervalo de tempo de recuperação em caso de falhas da rede em soluções tradicionais como *Pseudowire*-MPLS. Existirá, no entanto, uma dificuldade adicional em realizar experimentos reais como os apresentados por este trabalho, pois pôde-se beneficiar da característica de tratamento isolado das interfaces associadas ao *pipeline* OpenFlow. Esse fato permitiu que os eventos de queda de enlaces da rede SDN/OpenFlow não fossem reconhecidos pelo *pipeline* tradicional e não interferissem no tráfego em produção, visto que esse tráfego e o tráfego SDN/OpenFlow compartilham a mesma infraestrutura física de fibras óticas e interfaces.

No caso dos circuitos virtuais do tipo *Pseudowire* em um *backbone* MPLS, os procedimentos de avaliação seriam muito mais complexos de serem realizados, tendo em vista que esses procedimentos poderiam influenciar diretamente no tráfego dos clientes da rede em produção. Isso ocorre devido às interfaces associadas aos circuitos *Pseudowire*-MPLS e o tráfego de propósitos gerais serem tratadas por um único *pipeline*. Essa característica faz com que eventos de queda de enlace sejam reconhecidos por esse *pipeline*, levando-o a desviar todo o tráfego dos clientes da rede e não somente o tráfego dos circuitos virtuais.

Como sugestão de trabalho adicional, tendo em vista os resultados obtidos para o intervalo de tempo de convergência (no cenário de recuperação em caso de falhas) e para o intervalo de tempo de estabelecimento de circuito fim a fim percebeu-se também a necessidade de medir o intervalo de tempo entre a recepção das mensagens de controle OpenFlow do tipo *Flow-Mod* e a efetiva instalação da regra na tabela de fluxos. Isso porque a forma como cada fabricante implementa o protocolo OpenFlow deve impactar no intervalo de tempo em que cada mensagem de controle é processada por cada *switch*.

Além disso, outra sugestão de desenvolvimentos futuros para o VCFlow, é a criação de uma interface web que possua dentre suas principais funcionalidades: (i) um mecanismo de agendamento de serviços em que clientes autenticados sejam capazes de configurar circuitos virtuais por períodos pré-determinados; (ii) um mecanismo de monitoração dos circuitos virtuais já estabelecidos, em que seja possível monitorar os recursos de rede consumidos por cada circuito.

Em função dos resultados obtidos, a operacionalização do VCFlow em todo o *backbone* em produção da Redecomep-Rio/RedeRio está em fase final de avaliação para entrar em operação. Neste momento, o VCFlow já se encontra em fase de operação em modo experimental. Nesse caso, ele é utilizado para fornecer circuitos virtuais privados de alta capacidade (1Gbps) na Redecomep-Rio/RedeRio para o *testbed* do projeto *Future Internet Brazilian Environment for Experimentation* (Fibre)[23], organizado e gerenciado pela RNP no

[23] O projeto Fibre é um *testbed* para experimentação em Internet do futuro e adota os conceitos de SDN/OpenFlow. Universidades e centro de pesquisa por todo o Brasil participam do *testbed* Fibre, onde essas instituições abrigam "ilhas de experimentação", que funcionam como *mini-testbeds* locais. Dado suas características de colaboração, o Fibre opera como uma federação de "ilhas de experimentação" interconectadas através do *backbone* nacional da RNP. Cada instituição que faz parte do projeto tem autonomia

Brasil. Adotou-se para tal como projeto piloto o estabelecimento dos circuitos virtuais necessários para interconexão da "ilha de experimentação" do projeto Fibre na Universidade Federal Fluminense (UFF) ao *backbone* nacional da RNP. Esse projeto piloto encontra-se em operação na Redecomep-Rio/RedeRio desde janeiro de 2017.

Por fim, a tecnologia SDN traz um novo paradigma para o uso e gerenciamento das redes de computadores, sendo a sua característica principal a programabilidade centralizada e a possibilidade de uso de equipamentos de *hardware commodity* ou *switches* brancos (*whitebox*). A versatilidade trazida pela programação da rede por meio de *softwares* sofisticados abre possibilidades imensas. No caso de redes com enfoque acadêmico é possível pensar em diversas aplicações envolvendo criação de circuitos e técnicas específicas envolvendo projetos, instituições, segurança, controle de banda, etc. No entanto, a utilização permanente da tecnologia SDN na Redecomep-Rio/RedeRio é limitada hoje pelas características dos equipamentos que compõem o seu *backbone*. A proposta apresentada pelo VCFlow permite trazer solução para um problema de estabelecimento de circuitos de camada 2 neste nível da rede, integrando inclusive com circuitos equivalentes em redes parcerias. É possível vislumbrar ainda aplicações dessa tecnologia (em camada 2 ou outras) em grandes redes colaborativas mundiais, como por exemplo as redes dedicadas à pesquisa em física de altas energias, cosmologia ou astrofísica.

Possibilidades futuras seriam ainda identificar serviços na rede de produção com potencial de migração para serem atendidas por tecnologias de SDN/OpenFlow. Esses serviços seriam, por exemplo: roteamento de tráfego de propósitos gerais segregados de circuitos virtuais específicos; aplicação de QoS na rede, por exemplo para otimizar desempenho de aplicações que geram grandes volumes de dados (inclusive em redes inter-domínios); gerir de forma mais otimizada o tráfego da rede; manter redes experimentais em paralelo com redes de produção, etc. Esses serviços poderiam ser implementados por meio da utilização de aplicações de virtualização de redes, como por exemplo o *FlowVisor*, que faz um fatiamento (*slicing*) da rede física em diversas redes logicas, garantindo que cada controlador tenha conhecimento somente dos *switches* e recursos atribuídos a ele.

sobre seus recursos locais, ao mesmo tempo em que usa recursos de outras ilhas para montar experimentos de rede. Para mais informações: www.fibre.org.

7 REFERÊNCIAS

ALVES JR., N. **Caracterização de redes complexas aplicação à modelagem relacional entre Sistemas Autônomos da Internet** 2007. (Tese de Doutorado em Modelagem Computacional). IPRJ, Universidade do Estado do Rio de Janeiro, Nova Friburgo, RJ, Brasil.

ALVES JR., N. et al. **Topologia e Modelagem Relacional da Internet Brasileira**. Nota Técnica do CBPF - NT-004/4. Rio de Janeiro, RJ, Brasil 2004.

BAO-LIANG, Z. et al. A network-based VPN architecture using virtual routing. **Wuhan University Journal of Natural Sciences,** v. 10, n. 1, p. 161-164, 2005. ISSN 1007-1202.

BELTER, B. et al. Hardware abstraction layer as an SDN-enabler for non-OpenFlow network equipment. 2014 Third European Workshop on Software Defined Networks, 2014, IEEE. p.117-118.

BENSON, T.; AKELLA, A.; MALTZ, D. A. Unraveling the Complexity of Network Management. USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2009, Boston, Massachusetts. p.335-348.

BERDE, P. et al. ONOS: towards an open, distributed SDN OS. Proceedings of the third workshop on Hot topics in software defined networking, 2014, ACM. p.1-6.

BOBYSHEV, A. et al. Lambda Station: On-demand Flow Based Routing for Data Intensive Grid Applications over Multitopology Networks. 2006 3rd International Conference on Broadband Communications, Networks and Systems, 2006, IEEE. p.1-9.

CHAVES FILHO, G. D. A. **Comutação baseada em caminhos: uma solução SDN para problema de migração de máquinas virtuais em Data Centers**. 2015. (Mestrado em Informática). Centro de Ciências Exatas e Tecnologia, Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

CHOWDHURY, N. M. K.; BOUTABA, R. A survey of network virtualization. **Computer Networks,** v. 54, n. 5, p. 862-876, 2010. ISSN 1389-1286.

DART, E. et al. The Science DMZ: A Network Design Pattern for Data-Intensive Science. **Scientific Programming,** v. 22, n. 2, p. 173-185, 2014. ISSN 1058-9244.

DINIZ, P. H.; ALVES JR., N. **Ferramenta IPERF: geração e medição de tráfego TCP e UDP**. Centro Brasileiro de Pesquisas Físicas - Notas Técnicas. 4**:** 1–13 p. 2014.

DORIA, A. et al. **Forwarding and control element separation (ForCES) protocol specification**. Internet Engineering Task Force. 2010. (RFC 5810)

ESNET (ENERGY SCIENCE NETWORK). OSCARS: On-Demand Secure Circuits and Advance Reservation System. 2016. Disponível em: < http://www.es.net/engineering-services/oscars/ >. Acesso em: 02 de janeiro de 2017.

ESTEVES, A. M. B. **Sistema de monitoramento de redes baseado nos protocolos SNMP e SpanningTree** 2013. (Dissertação de Mestrado em Física - Instrumentação Científica). Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, RJ, Brasil.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The Road to SDN: an Intellectual History of Programmable Networks. **ACM SIGCOMM Computer Communication Review,** v. 44, n. 2, p. 87-98, 2014. ISSN 0146-4833.

FERNANDEZ, M. P. Comparing OpenFlow controller paradigms scalability: Reactive and proactive. Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, 2013, IEEE. p.1009-1016.

FERREIRA, C. C. **Análise Comparativa de Controladores para Redes Definidas por Software de Classe Carrier Grade**. 2016. (Dissertação de Mestrado em Ciência da Computação). Faculdade de Computação, Universidade Federal de Uberlândia Uberlândia, MG, Brasil.

FLOODLIGHT PROJECT. Floodlight. 2016. Disponível em: < http://www.projectfloodlight.org/floodlight/ >. Acesso em: 07 de outubro de 2016.

FUENTES, F.; KAR, D. C. Ethereal vs. Tcpdump: a comparative study on packet sniffing tools for educational purpose. **Journal of Computing Sciences in Colleges,** v. 20, n. 4, p. 169-176, 2005. ISSN 1937-4771.

GARZOGLIO, G. et al. Big data over a 100G network at Fermilab. Journal of Physics: Conference Series, 2014, IOP Publishing. p.062017.

GEANT. About AutoBAHN. 2017. Disponível em: < http://geant3.archive.geant.net/service/autobahn/about_autoBAHN/Pages/About_autoBAHN.aspx >. Acesso em: 11 de janeiro de 2017.

GLOBALNOC. OESS: Open Exchange Software Suite. 2016. Disponível em: < http://globalnoc.iu.edu/sdn/oess.html >. Acesso em: 31 de dezembro de 2016.

GONT, F.; PIGNATARO, C. **Formally Deprecating Some ICMPv4 Message Types**. 2013. (2070-1721)

GUIMARAES, A. G.; LINS, R. D.; DA OLIVEIRA, R. C. **Segurança em Redes Privadas Virtuais-VPNs**. Brasport, 2006. ISBN 8574522899.

GUOK, C. et al. Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System. 2006 3rd International Conference on Broadband Communications, Networks and Systems, 2006, IEEE. p.1-8.

HILLS, R. arp-scan - Linux man page. 2011. Disponível em: < https://linux.die.net/man/1/arp-scan >. Acesso em: 10 de novembro de 2016.

HOEFT, B.; PETZOLD, A. 100G Deployment@(DE-KIT). Journal of Physics: Conference Series, 2015, IOP Publishing. p.052017.

IEEE (INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS). **IEEE 802: Local and Metropolitan Area Network Standards, IEEE Standard 802.1ab**: IEEE 2009.

_____. **IEEE 802: Local and Metropolitan Area Network Standards**. IEEE Standard 802.1q 2014.

INTERNET2. Internet2 - About Us. 2017a. Disponível em: < http://www.internet2.edu/about-us/ >. Acesso em: 02 de janeiro de 2017.

_____. Layer 2 Services. 2017b. Disponível em: < http://www.internet2.edu/products-services/advanced-networking/layer-2-services/#service-overview >. Acesso em: 02 de janeiro de 2017.

IXIA. IxChariot: Instant performance assessment of complex networks from pre- to post-deployment., 2017. Disponível em: < https://www.ixiacom.com/products/ixchariot >. Acesso em: 14 de fevereiro de 2017.

JAIN, S. et al. B4: Experience with a Globally-Deployed Software Defined WAN. **ACM SIGCOMM Computer Communication Review,** v. 43, n. 4, p. 3-14, 2013. ISSN 1450320562.

JARSCHEL, M. et al. OFCProbe: A platform-independent tool for OpenFlow controller analysis. Communications and Electronics (ICCE), 2014 IEEE Fifth International Conference on, 2014, IEEE. p.182-187.

KATRAMATOS, D. et al. The TeraPaths Testbed: Exploring End-to-End Network QoS. Testbeds and Research Infrastructure for the Development of Networks and Communities, 2007. TridentCom 2007. 3rd International Conference on, 2007, IEEE. p.1-7.

KHATTAK, Z. K.; AWAIS, M.; IQBAL, A. Performance evaluation of OpenDaylight SDN controller. 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2014, IEEE. p.671-676.

KHONDOKER, R. et al. Feature-based comparison and selection of Software Defined Networking (SDN) controllers. Computer Applications and Information Systems (WCCAIS), 2014 World Congress on, 2014, IEEE. p.1-7.

KIM, H.; FEAMSTER, N. Improving Network Management with Software Defined Networking. **IEEE Communications Magazine,** v. 51, n. 2, p. 114-119, 2013. ISSN 0163-6804.

KISSEL, E. et al. Efficient wide area data transfer protocols for 100 Gbps networks and beyond. Proceedings of the Third International Workshop on Network-Aware Data Management, 2013, ACM. p.3.

KNIGHT, P.; LEWIS, C. Layer 2 and 3 Virtual Private Networks: Taxonomy, Technology, and Standardization Efforts. **IEEE Communications Magazine,** v. 42, n. 6, p. 124-131, 2004. ISSN 0163-6804.

KREUTZ, D. et al. Software-Defined Networking: A Comprehensive Survey. **Proceedings of the IEEE,** v. 103, n. 1, p. 14-76, 2015. ISSN 0018-9219.

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

LEWIS, M. **Comparing, designing, and deploying VPNs**. Indianapolis, IN: Cisco Press, 2006. ISBN 1587051796.

LIEBEHERR, J.; ZARKI, M. E. **Mastering Networks: An Internet Lab Manual**. Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0201781344.

LINUX FOUNDATION. Open vSwitch: Features. 2016. Disponível em: < http://openvswitch.org/features/ >. Acesso em: 14 de fevereiro de 2017.

MACEDO, E. L. C. **Previsão de Tráfego em Enlaces de Rede Utilizando Séries Temporais**. 2016. (Dissertação de Mestrado em Engenharia de Sistemas e Computação). Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

MARIANI, A. C. Algoritmo de Dijkstra para cálculo do Caminho de Custo Mínimo. 2017. Disponível em: < http://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html >. Acesso em: 14 de fevereiro de 2017.

MARTELLI, E.; STANCU, S. LHCOPN and LHCONE: Status and Future Evolution. Journal of Physics: Conference Series, 2015, IOP Publishing. p.052025.

MCKEOWN, N. How SDN will Shape Networking. **Open Networking Summit**, Palo Alto, Califórnia, 2011. Disponível em: < https://www.youtube.com/watch?v=c9-K5O_qYgA >. Acesso em: 14 de dezembro de 2016.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **ACM SIGCOMM Computer Communication Review,** v. 38, n. 2, p. 69-74, 2008. ISSN 0146-4833.

MEDVED, J. et al. Opendaylight: Towards a model-driven sdn controller architecture. Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014, 2014.

MENDIOLA, A. et al. DynPaC: A Path Computation Framework for SDN. 2015 Fourth European Workshop on Software Defined Networks, 2015, IEEE. p.119-120.

_____. Multi-domain bandwidth on demand service provisioning using SDN. NetSoft Conference and Workshops (NetSoft), 2016 IEEE, 2016, IEEE. p.353-354.

_____. **A solution for connection-oriented multi-domain SDN based on NSI-CS and the DynPaC framework**. Terena Networking Conference. Porto, Portugal 2015.

METTER, C. et al. Investigating the impact of network topology on the processing times of SDN controllers. 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, IEEE. p.1214-1219.

MININET TEAM. Mininet Overview. 2016. Disponível em: < http://mininet.org/overview/ >. Acesso em: 14 de fevereiro de 2017.

MIRANDA, E. F. **Desenvolvimento de Sitema para Monitoramento de Redes de Computadores e Servidor Looking Glass** 2013. (Dissertação de Mestrado em Física - Instrumentação Científica). Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro, RJ, Brasil.

MIRANDA, E. F.; ALVES JR., N.; SOUZA, M. G. M. Desenvolvimento de um Repositório RIB-BGP. **Notas Técnicas do CBPF**, Rio de Janeiro, RJ, Brasil**,** v. 3, n. 2, p. 6, 2013. ISSN 0101-9201. Disponível em: < http://cbpfindex.cbpf.br/publication_pdfs/nt00413apub.2013_08_02_09_04_39.pdf >.

MISA, T. J.; FRANA, P. L. An interview with Edsger W. Dijkstra. **Communications of the ACM,** v. 53, n. 8, p. 41-47, 2010. ISSN 0001-0782.

MORAES, L. F. M. D.; ALBUQUERQUE, M. P. D.; RIBEIRO FILHO, J. L. Infraestrutura Redes de Alta Velocidade no Rio de Janeiro: história e estado da arte. In: (Ed.). **A História da Telessaúde da Cidade para o Estado do Rio de Janeiro**. Rio de Janeiro: EdUERJ, 2015. ISBN 978-85-7511395-0.

MOSS, M. L.; TOWNSEND, A. M. The Internet backbone and the American metropolis. **The information society,** v. 16, n. 1, p. 35-47, 2000. ISSN 0197-2243.

OGF/NSI-WG, O. G. F. O. N. S. I. W. G. N.-W. **Inter-Domain Controller Protocol (IDCP) Specification** 2010.

OLIVEIRA, J. M.; LINS, R. D.; MENDONÇA, R. **Redes MPLS: Fundamentos e Aplicações**. Rio de Janeiro: Brasport, 2012. ISBN 978-85-7452-539-6.

ONF (OPEN NETWORK FOUNDATION). **OpenFlow Switch Specification Version 1.3.5 (Protocol version 0x04)** 2015a.

_____. **OpenFlow Switch Specification Version 1.5.1 (Protocol version 0x06)** 2015b.

_____. Open Networking Foundation. 2016. Disponível em: < https://www.opennetworking.org/ >. Acesso em: 14 de dezembro de 2016.

PAKZAD, F. et al. Efficient topology discovery in software defined networks. Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on, 2014, IEEE. p.1-8.

PAPAGIANNAKI, K. et al. A pragmatic definition of elephants in internet backbone traffic. Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment, 2002, ACM. p.175-176.

PARNIEWICZ, D. et al. Design and implementation of an openflow hardware abstraction layer. Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing, 2014, ACM. p.71-76.

PETERSON, L. L.; DAVIE, B. S. Chapter 2: Getting Connected. In: ADAMS, R. e MCFADDEN, N. (Ed.). **Computer networks: a systems approach**. 5ª ed.: Elsevier, 2012a. ISBN 978-0-12-385059-1

_____. Chapter 3: Internetworking. In: ADAMS, R. e MCFADDEN, N. (Ed.). 5ª ed. : Elsevier 2012b.

PINTO, J. D. O. et al. **DWDM em Redes Metropolitanas**. Nota Técnica do CBPF – NT001/02. Rio de Janeiro, Brasil 2002.

PLUMMER, D. C. RFC 826: An Ethernet Address Resolution Protocol. **InterNet Network Working Group**, 1982.

POSTEL, J. RFC 792: Internet control message protocol. **InterNet Network Working Group**, 1981.

RAO, N. S. et al. UltraScienceNet: Network Testbed for Large-Scale Science Applications. **IEEE Communications Magazine,** v. 43, n. 11, p. S12-S17, 2005. ISSN 0163-6804.

REITBLATT, M. et al. Consistent updates for software-defined networks: Change you can believe in! , Proceedings of the 10th ACM Workshop on Hot Topics in Networks, 2011, ACM. p.7.

RICCI, B. **Rede Segura: VPN Linux**. Rio de Janeiro: Editora Ciência Moderna, 2007.

RNP (REDE NACIONAL DE ENSINO E PESQUISA). Brasil passa a fazer parte da rede LHCONE. 2015. Disponível em: < https://www.rnp.br/destaques/brasil-passa-fazer-parte-rede-lhcone >. Acesso em: 09 de janeiro de 2017.

ROBERTS, G. et al. NSI Connection Service v2. 0. Open Grid Forum, GWD-RP, NSI-WG, 2013.

ROSEN, E. C.; REKHTER, Y. RFC 4364: BGP/MPLS IP virtual private networks (VPNs). **The Internet Society**, 2006.

RYU. Framework Community: Ryu SDN controller. 2016. Disponível em: < https://osrg.github.io/ryu/ >. Acesso em: 03 de março de 2017.

SCHEMERS, R. J. FPING Man Page. 2007. Disponível em: < http://fping.sourceforge.net/man/ >. Acesso em: 14 de outubro de 2016.

SEO, K.; KENT, S. Security architecture for the internet protocol. 2005.

SHAH, S. A. et al. An architectural evaluation of SDN controllers. 2013 IEEE International Conference on Communications (ICC), 2013, IEEE. p.3504-3508.

SHENKER, S. et al. The Future of Networking, and The Past of Protocols. **Open Networking Summit**, Palo Alto, Califórnia, 2011. Disponível em: < https://www.youtube.com/watch?v=YHeyuD89n1Y >. Acesso em: 14 de dezembro de 2016.

SHERWOOD, R.; KOK-KIONG, Y. Cbench: an open-flow controller benchmarker. 2010. Disponível em: < http://archive.openflow.org/wk/index.php/Oflops >. Acesso em: 30 de novembro de 2016.

SILVA, V. L. P. D. **Identificação de Anomalias em Fluxos de Rede Utilizando Previsões em Séries Temporais pelo Método de Holt Winters**. 2016. (Dissertação de Mestrado em Engenharia de Sistemas e Computação). Universidade Federal do Rio de Janeiro Rio de Janeiro, RJ, Brasil.

SMITH, M. et al. **OpFlex control protocol**. Internet Engineering Task Force. 2014. (Internet Draft)

SONG, H. Protocol-Oblivious Forwarding: Unleash the power of SDN through a future-proof forwarding plane. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, 2013, ACM. p.127-132.

TEPSUPORN, S. et al. A multi-domain SDN for dynamic layer-2 path service. Proceedings of the Fifth International Workshop on Network-Aware Data Management, 2015, ACM. p.2.

TOOTOONCHIAN, A. et al. On Controller Performance in Software-Defined Networks. Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services, 2012.

WLCG (WORLDWIDE LHC COMPUTING GRID). LHC Optical Private Network. 2016. Disponível em: < http://lhcopn.web.cern.ch/lhcopn/ >. Acesso em: 09 de janeiro de 2017.

WLCG (WORLDWIDE LHC COMPUTING GRID). LHC Open Network Environment. 2016. Disponível em: < http://lhcone.net/ >. Acesso em: 09 de janeiro de 2017.

YANG, X. et al. GMPLS-based Dynamic Provisioning and Traffic Engineering of High-Capacity Ethernet Circuits in Hybrid Optical/Packet Networks. Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, 2006, IEEE. p.1-5.

ZHENG, X. et al. CHEETAH: Circuit-switched High-speed End-to-End Transport ArcHitecture testbed. **IEEE Communications Magazine,** v. 43, n. 10, p. S11-S17, 2005.

ZURAWSKI, J. et al. The DYNES Instrument: A Description and Overview. Journal of Physics: Conference Series, 2012, IOP Publishing. p.042065.

8 APÊNDICE A – CAMPOS DE CORRESPONDÊNCIA DE FLUXOS PARA OPENFLOW VERSÃO 1.3

A Tabela 8.1 descreve em detalhes cada um dos campos de correspondência do OpenFlow versão 1.3 utilizados nas tabelas de fluxos para tratamento dos pacotes. Os campos que são obrigatórios no *pipeline* do *switch* e devem ser suportados em pelo menos uma das tabelas de fluxos são indicados pelo caractere asterisco (*) ao lado de cada um desses itens.

Tabela 8.1: Tipos de campos de correspondência de fluxos para OpenFlow versão 1.3.

Campos de Correspondência de Fluxos	Descrição
IN_PORT*	Porta lógica de entrada no *switch.* Representação numérica da porta de entrada do pacote, começando em 1. Pode ser uma porta física ou uma porta lógica definida pelo *switch*
IN_PHY_PORT	Porta física de entrada no *switch.* Porta física subjacente quando o pacote é recebido em uma porta lógica
METADATA	Metadados passados entre tabelas
ETH_DST*	Endereço *Ethernet* (MAC) de destino
ETH_SRC*	Endereço Ethernet (MAC) de origem
ETH_TYPE*	Campo "Tipo" do cabeçalho *Ethernet*
VLAN_VID	Identificação de VLAN
VLAN_PCP	Prioridade de VLAN
IP_DSCP	IP DSCP (6 bits no campo ToS)
IP_ECN	IP ECN (2 bits no campo ToS)
IP_PROTO*	Campo Protocolo do cabeçalho IP
IPV4_SRC*	Endereço IPv4 de origem
IPV4_DST*	Endereço IPv4 de destino
TCP_SRC*	Porta TCP de origem
TCP_DST*	Porta TCP de destino
UDP_SRC*	Porta UDP de origem
UDP_DST*	Porta UDP de destino
SCTP_SRC	Porta SCTP de origem
SCTP_DST	Porta SCTP de destino
ICMPV4_TYPE	Campo "Tipo" do cabeçalho ICMP
ICMPV4_CODE	Campo "Código" do cabeçalho ICMP
ARP_OP	Campo "Opcode" do cabeçalho ARP
ARP_SPA	Endereço IPv4 de origem do cabeçalho ARP
ARP_TPA	Endereço IPv4 de alvo do cabeçalho ARP
ARP_SHA	Endereço de *hardware* de origem do cabeçalho ARP
ARP_THA	Endereço de hardware de destino do cabeçalho ARP
IPV6_SRC*	Endereço IPv6 de origem
IPV6_DST*	Endereço IPv6 de destino
IPV6_FLABEL	Campo "Rótulo de Fluxo" do cabeçalho IPv6
ICMPV6_TYPE	Campo "Tipo" do cabeçalho ICMPv6
ICMPV6_CODE	Campo "Código" do cabeçalho ICMPv6

IPV6_ND_TARGET	Endereço IPv6 de alvo do cabeçalho ND
IPV6_ND_SLL	Endereço de *hardware* de origem do cabeçalho ND
IPV6_ND_TLL	Endereço de hardware de destino do cabeçalho ND
MPLS_LABEL	Rótulo MPLS
MPLS_TC	Campo "Classe de Tráfego (TC)" do cabeçalho MPLS
OFPXMT_OFP_MPLS_BOS	Bit de "Início da Pilha (BoS)" do cabeçalho MPLS
PBB_ISID	PBB I-SID
TUNNEL_ID	Metadados de porta lógica

9 APÊNDICE B – ALGORITMO *SHORTEST PATH FIRST* ADOTADO PELO VCFLOW PARA CÁLCULO DE MENOR CAMINHO

A presente seção descreve brevemente o funcionamento do algoritmo de cálculo de menor caminho adotado para cálculo das tabelas de encaminhamento entre os *endpoints* de uma topologia de rede controlada pelo VCFlow. Ao fim da seção são apresentadas as linhas de código do VCFlow referentes a esse algoritmo.

O algoritmo de Dijkstra ou algoritmo *Shortest Path First* (SPF), desenvolvido pelo pesquisador holandês E.W. Dijkstra em 1956, é um dos algoritmos que calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo (Mariani, 2017). Basicamente, sendo executado para cada um dos nós, ele permite descobrir a menor distância até todos os possíveis destinos. Ele é bastante simples e apresenta um bom nível de desempenho, por esse motivo foi adotado na solução VCFlow.

O algoritmo é executado seguindo-se os seguintes passos:

1. Atribui-se valor zero à estimativa do custo mínimo do vértice **s** (a raiz da busca) e infinito às demais estimativas;
2. Atribui-se o peso do enlace à estimativa para todos os vizinhos da raiz;
3. Marca-se a raiz como precedente de todos os seus vizinhos;
4. Marca-se a raiz como visitada;
5. Enquanto existirem vértices não visitados:
    a. Escolhe-se um vértice **k** ainda não visitado, cuja estimativa seja a menor dentre todos os vértices não visitados;
    b. Para todos os vizinhos de **k**:
        i. Soma-se a estimativa do vértice **k** com o custo do arco que une **k** a **j**;
        ii. Caso essa soma seja melhor que a estimativa anterior para o vértice **j**, substitui-se ela e anota-se **k** como precedente de **j**;
    c. Marca-se **k** como visitado.

A seguir podem ser encontradas as linhas de código desenvolvidas para implementar o algoritmo SPF na solução VCFlow.

```
class calculo_dijkstra: # Retorna um objeto com a tabela de roteamento
    def __init__(self, topologia):
        self.tabela_dijkstra = {}
        self.tabela_roteamento_completa = {}
        verificador = []
        prosseguimento = "sim"
        self.logger = logging.getLogger('spf_calculation_application')
        if not len(self.logger.handlers):
            self._set_logger()
```

```
        method_str = {'method': '__init__'}

        self.logger.info('Routing table calculation has been initiated.',
extra=method_str)
        self.logger.info('Topology object is: %s.', str(topologia),
extra=method_str)

        """Calcula a tabela dijkstra para todos os switches da
topologia"""
        """Para realizar o calculo tem que verificar se para cada switch
dest de cada switch origem existe um switch origem (verifi
ca se existe o par switch_origem:switch_destino e
switch_destino_switch_origem)"""
        for switch_origem in topologia.keys():
            for switch_destino in topologia[switch_origem]:
                if topologia.has_key(switch_destino):
                    if topologia[switch_destino].has_key(switch_origem):
                        verificador.append("ok")
                    else:
                        verificador.append("nok")
                else:
                    verificador.append("nok")

        for i in verificador:
            if (i == "nok"):
                self.logger.info('Proceeding could not be completed due
to topology mismatch.', extra=method_str)
                prosseguimento = "nao"

        if (prosseguimento == "sim"):
            for switch in topologia.keys():
                self.tabela_dijkstra[switch] =
self.calcula(self.inicializacao(topologia), topologia, switch)

            self.tabela_roteamento_completa =
self.monta_tabela_roteamento(self.tabela_dijkstra, topologia)

    def _set_logger(self):
        self.logger.setLevel(logging.INFO)
        self.logger.propagate = False
        hdlr = logging.StreamHandler()
        fmt_str = '[SPF][%(levelname)s] method=%(method)s %(asctime)s:
%(message)s'
        hdlr.setFormatter(logging.Formatter(fmt_str))
        self.logger.addHandler(hdlr)

    def __repr__(self): # Metodo que retorna a maneira como o objeto da
classe e printado na tela
        return  str(self.tabela_roteamento_completa)

    def inicializacao(self, topologia): # faz todos os vertices
        method_str = {'method': 'inicializacao'}
        # Cria uma um dicionario em que as chaves sao topologia.keys() e
os valores sao o objeto dados_dijkstra
        # Essa tabela e utilizada para o calculo do algoritmo Dijkstra
(uma tabela por topologia)
        tabela_dijkstra = {}
        for switch in topologia.keys():
```

```
            tabela_dijkstra[switch] = dados_dijkstra(None, float("inf"),
"nao")

        self.logger.debug('Routing table has been initiated. %s',
tabela_dijkstra, extra=method_str)
        return tabela_dijkstra

    def calcula(self, tabela_inicializada, topologia, switch_raiz):
        #Algoritmo para calculo da tabela dijkstra
        """Atribua valor zero a estimativa do custo minimo do vertice s
(raiz da busca) e infinito as demais estimativas)"""
        """Marque a raiz como precedente da propria raiz"""
        tabela_inicializada[switch_raiz].estimativa = 0
        tabela_inicializada[switch_raiz].precedente = switch_raiz

        for switch_vizinho_raiz in topologia[switch_raiz].keys():
            tabela_inicializada[switch_vizinho_raiz].estimativa =
topologia[switch_raiz][switch_vizinho_raiz].custo # ******* atrib
ui o custo
            tabela_inicializada[switch_vizinho_raiz].precedente =
switch_raiz

        """Marque a raiz como visitada"""
        tabela_inicializada[switch_raiz].visitado = "sim"

        switches_nao_visitados = 1
        """Enquanto existirem vertices (switches) nao visitados"""
        while switches_nao_visitados == 1:
            """Escolha um vertice k ainda nao visitado cuja estimativa
seja a menor dentre todos os vertices nao visitados"""
            #Cria um dicionatio com as estimativas do switches vizinhos
nao visitados
            estimativas = {}
            for switch in tabela_inicializada.keys():
                if (tabela_inicializada[switch].visitado == "nao"):
                    estimativas[switch] =
tabela_inicializada[switch].estimativa
            # Referencia: stackoverflow get key with the least value from
a dictionary
            switch_vizinho_menor_estimativa = min(estimativas,
key=estimativas.get) #retorna o switch com menor estimativa

            """Para todos os vizinhos de k"""
            for switches_vizinhos_switch_menor_estimativa in
topologia[switch_vizinho_menor_estimativa].keys():
                """Some a estimativa do vertice k com o custo do arco que
une k (switch_vizinho_menor_estimativa) a j(switches_vizi
nhos_switch_menor_estimativa)"""
                nova_estimativa =
tabela_inicializada[switch_vizinho_menor_estimativa].estimativa +
topologia[switches_vizinhos_swi
tch_menor_estimativa][switch_vizinho_menor_estimativa].custo
                if ( nova_estimativa <
tabela_inicializada[switches_vizinhos_switch_menor_estimativa].estimativa
):

tabela_inicializada[switches_vizinhos_switch_menor_estimativa].estimativa
= nova_estimativa
```

```
tabela_inicializada[switches_vizinhos_switch_menor_estimativa].precedente
= switch_vizinho_menor_estimativa

            """Marque k (switch_vizinho_menor_estimativa) como
visitado"""
            tabela_inicializada[switch_vizinho_menor_estimativa].visitado
= "sim"

            """Enquanto existirem vertices (switches) nao visitados"""
            switches_nao_visitados = 0
            for switch in tabela_inicializada.keys():
                if (tabela_inicializada[switch].visitado == "nao"):
                    switches_nao_visitados = 1

        return tabela_inicializada

    def monta_tabela_roteamento(self, tabela_todos_switches, topologia):
        #Exemplo do recebimento {1: {1: {'visitado': 'sim', 'precedente':
1, 'estimativa': 0}, 2: {'visitado': 'sim', 'precedente':
 1, 'estimativa': 1}, 3: {'visitado': 'nao', 'precedente': 2,
'estimativa': 2}, 4: {'visitado': 'sim', 'precedente': 1, 'estimativa
': 1}, 5: {'visitado': 'sim', 'precedente': 1, 'estimativa': 1}}
        #tabela_roteamento = {switch_origem:{switch_destino:[switch,
porta_saida; switch, porta_saida; ultimo_switch, None]}}
        tabela_roteamento = {}

        # Faz verificacao se a topologia esta dividida em duas ou mais
partes. Se estiver retorna a tabela de roteamento vazia
        for sw_ori in tabela_todos_switches.keys():
            for sw_dest in tabela_todos_switches[sw_ori].keys():
                if ( tabela_todos_switches[sw_ori][sw_dest].precedente ==
None ): # Se o precedente de qualquer switch na tabela_di
jkstra for None (ou seja, ele nao entrou no calculo de vizinhos, entao a
topologia esta desmenbrada) entao nao continua
                    return tabela_roteamento

        for switch_origem in tabela_todos_switches.keys():
            tabela_roteamento[switch_origem] = {}
            for switch_destino in
tabela_todos_switches[switch_origem].keys():
                if ( switch_destino != switch_origem ):
                    #print "A topologia antes do encontra caminho e: ",
topologia
                    #print "Antes do encontra caminho o switch origem e:
", switch_origem, "e o switch destino e: ", switch_destino
                    tabela_roteamento[switch_origem][switch_destino] =
self.encontra_caminho(switch_origem, switch_destino, tabela_
todos_switches, topologia)

        return tabela_roteamento

    def encontra_caminho(self, sw_origem, sw_destino,
tabela_todos_switches, topologia):
        """Encontra caminho entre um switch de origem e um destino
atraves da topologia"""
        #Exemplo do recebimento tabela_todos_switches {1: {1:
{'visitado': 'sim', 'precedente': 1, 'estimativa': 0}, 2: {'visitado'
```

```
: 'sim', 'precedente': 1, 'estimativa': 1}, 3: {'visitado': 'nao',
'precedente': 2, 'estimativa': 2}, 4: {'visitado': 'sim', 'prece
dente': 1, 'estimativa': 1}, 5: {'visitado': 'sim', 'precedente': 1,
'estimativa': 1}}

        caminho_switches = [] #Lista com o dpid dos switches com o
caminho de tras para frente, do sw_destino para o sw_origem
        salto_anterior = sw_destino
        while ( salto_anterior != sw_origem ):
            caminho_switches.append(salto_anterior)
            salto_anterior =
tabela_todos_switches[sw_origem][salto_anterior].precedente

        caminho_switches.append(salto_anterior) # Faz o append do switch
origem
        caminho_switches = caminho_switches[::-1] # Inverte a lista com o
caminho para obter o caminho na ordem de sw_origem a sw_d
estino

        caminho = []
        contador_saltos = 0
        for posicao_switch_saida in range(len(caminho_switches)):
            if ( caminho_switches[posicao_switch_saida] != sw_destino ):
                interface_saida =
topologia[caminho_switches[posicao_switch_saida]][caminho_switches[posica
o_switch_saida+1]].int_s
aida_src

            else:
                interface_saida = None

            caminho.append(saida(caminho_switches[posicao_switch_saida],
interface_saida, contador_saltos))
            contador_saltos = contador_saltos + 1

        return caminho
```

10 APÊNDICE C – FERRAMENTAS UTILIZADAS PARA CARACTERIZAÇÃO DA SOLUÇÃO VCFLOW

A presente seção tem como objetivo dar uma breve descrição das ferramentas utilizadas no Capítulo 4 - Caracterização do VCFlow . São elas: *arp-scan*, FPING, *tcpdump*, *Mininet*.

10.1 *arp-scan*

O *arp-scan* é uma ferramenta de interface de linha de comando (CLI) utilizada, principalmente, para encontrar todos os dispositivos que possuem endereçamento IPv4 ativo em uma sub-rede, por meio do escaneamento através de pacotes ARP. O *Address Resolution Protocol* (ARP) (Plummer, 1982) utiliza um formato simples de mensagem contendo uma requisição para resolução de endereços (mensagem ARP *request*) ou uma resposta a uma requisição (mensagem ARP *reply*). As mensagens ARP *request* visam descobrir qual endereço MAC está associado a um determinado endereço IP. Em uma rede *Ethernet*, essas mensagens são encaminhadas, em geral, a todos os *hosts* de uma mesma sub-rede IP. O cabeçalho ARP dessas requisições contém o endereço MAC da placa de rede de origem, o endereço IP de origem, o endereço MAC de destino 00:00:00:00:00:00 e o endereço IP de destino. Já as mensagens ARP *reply* encaminhadas em resposta às requisições contêm em seus cabeçalhos os endereços MAC de origem e de destino e os endereços IP de origem e de destino.

A ferramenta *arp-scan* encaminha pacotes ARP *request* a um *host*, a um conjunto de *hosts*, ou a todos os *hosts* em uma rede local e exibe as respostas recebidas. Por padrão, os pacotes ARP são encaminhados ao endereço de *broadcast Ethernet* FF:FF:FF:FF:FF:FF, apesar dessa opção poder ser modificada. O *arp-scan* permite também a alteração do intervalo e do *timeout* entre pacotes ARP *request* de modo simplificado, uma das principais características que motivou sua utilização. Caso seja recebida uma resposta após o encaminhamento de uma requisição a um endereço IPv4, as mensagens param de ser encaminhadas para esse endereço IP e o escaneamento prossegue. Caso seja o último *host* da lista de escaneamento, a ferramenta encerra sua execução (Hills, 2011).

10.2 FPING

A ferramenta FPING é muito semelhante à aplicação “Ping”, nativa de sistemas operacionais como Windows e Linux. O FPING é também uma aplicação com interface de linha de comando utilizada para, dentre outras funções, encontrar máquinas que estejam conectadas à rede.

Utilizando o protocolo ICMP (Postel, 1981; Gont e Pignataro, 2013), o FPING envia pacotes de informação a um *host* especifico e aguarda um pacote de retorno. Ao pacote

enviado chamamos de ICMP e*cho-request* e à resposta recebida pela origem caso o *host* de destino esteja acessível chamamos de ICMP e*cho-reply*. Se após um determinado tempo limite o *host* não responder, ele será considerado inalcançável.

Diferentemente do Ping tradicional, com o FPING é possível fornecer vários endereços de *hosts* em uma só linha de comando ou um arquivo texto, com uma lista de *hosts* a serem sondados. Em vez de enviar pacotes a um único destinatário e aguardar que o tempo de resposta se esgote ou que uma resposta seja recebida, o FPING envia um pacote e segue para o próximo destino em um método do tipo *round-robin*. Os *hosts* que respondem às solicitações são marcados em uma lista como ativos e não são mais checados. Os que não respondem, após um determinado período, são marcados como inalcançáveis (Schemers, 2007).

Ao contrário do Ping o FPING foi criado para ser usado em *scripts*, logo sua saída foi projetada para ser fácil de ser analisada (Schemers, 2007). Além disso, o FPING permite a alteração do intervalo entre pacotes ICMP e do *timeout* dos pacotes de maneira simples, característica essa que motivou a sua utilização.

10.3 *tcpdump*

O *tcpdump* é um *sniffer* de pacotes desenvolvido no início dos anos de 1990 pelo *Lawrence Berkeley National Laboratory* (Liebeherr e Zarki, 2003). O *tcpdump* é uma ferramenta amplamente utilizada por administradores de rede e desenvolvida para sistemas operacionais tipo UNIX em modo CLI. Pode ser utilizada para coletar dados de uma rede, decifrar os cabeçalhos dos pacotes e exibi-los de uma forma de mais simples compreensão.

Em vez de capturar todo o tráfego e buscar por toda a saída do comando pelas informações de interesse, o *tcpdump* permite limitar o tráfego a ser capturado especificado uma expressão de filtragem, como limitar a captura somente de pacote ARP ou ICMP. Além disso, é possível também realizar o salvamento das capturas de tráfego para posterior análise dos dados por ferramentas como *Wiresha*r*k*, por exemplo, ou atém mesmo pelo próprio *tcpdump*.

Essa ferramenta foi adotada principalmente pela sua simplicidade de execução através de uma única linha de comando e pela familiaridade com sua utilização.

10.4 Mininet

O Mininet é um emulador utilizado para criar redes de experimentais completas em um PC com *hosts*, *switches*, controladores e enlaces virtuais. Os *hosts* emulados rodam os *softwares* de rede Linux padrão disponíveis na máquina onde o emulador é executado, assim como o *kernel* Linux real e a toda a pilha de protocolos TCP/IP. Por padrão, os switches virtuais são executados por meio da aplicação *Open vSwitch* (OvS) (Linux Foundation, 2016), que permite o suporte ao protocolo OpenFlow para desenvolvimento de SDN.

Como principais características o Mininet: a. provê uma plataforma de testes de rede simples e sem custos para desenvolvimento de aplicações SDN/OpenFlow; b. permite a

criação de topologias de teste complexas sem a necessidade de se criar uma rede com dispositivos fisicamente conectados; c. possui uma CLI para a depuração de problemas ou execução de testes em toda a rede; d. provê o suporte a topologias customizadas através de uma API Python; e. inclui um conjunto de topologias básicas parametrizadas, o que simplifica muito a criação de topologias, e permite a rápida criação de topologias complexas (Mininet Team, 2016).

O suporte a API-Python para criação de topologias customizadas, a possibilidade de criação de topologias de teste complexas através de comandos via CLI de modo direto e pelo Mininet rodar código real nos *hosts* e *switches* virtuais, minimizando a necessidade de alteração de código para implementação em um sistema real são os fatores que levaram a sua adoção para prototipagem e avaliação de desempenho.

11 APÊNDICE D – MODIFICAÇÕES DO ARCABOUÇO DO CONTROLADOR RYU PARA DESCOBERTA DE TOPOLOGIA EM REDES SDN/OPENFLOW HÍBRIDAS

Segue abaixo as modificações realizadas no arcabouço do controlador Ryu para permitir o suporte a descoberta de topologia em redes SDN/OpenFlow híbridas, conforme descrito na Seção 3.4. Descoberta de Topologia em Redes SDN/OpenFlow Híbridas. As necessárias foram todas realizadas na biblioteca "*switches.py*", disponibilizada por padrão em: `/usr/local/lib/python2.7/dist-packages/ryu/topology/switches.py`

```
root@mininet-vm:/usr/local/lib/python2.7/dist-packages/ryu/topology# diff
switches_bkp.py switches.py
35,36c35,36
< from ryu.lib.packet import arp, ipv4, ipv6
< from ryu.ofproto.ether import ETH_TYPE_LLDP
---
> from ryu.lib.packet import arp, vlan, ipv4, ipv6
> from ryu.ofproto.ether import ETH_TYPE_LLDP, ETH_TYPE_8021Q
58a59,62
> PCP_NETWORK_CONTROL = 7
> CFI_LLDP_PKT = 0
> VID_LLDP_PKT = 0
>
436,437c443,446
<         ethertype = ETH_TYPE_LLDP
<         eth_pkt = ethernet.ethernet(dst, src, ethertype)
---
>         #ethertype = ETH_TYPE_LLDP # Modificado em 30/05/2016
>         ethertype_vlan = ETH_TYPE_8021Q # Adicionado em 30/05/2016
>         #eth_pkt = ethernet.ethernet(dst, src, ethertype) # Modificado
em 30/05/2016
>         eth_pkt = ethernet.ethernet(dst, src, ethertype_vlan) #
Adicionado em 30/05/2016
439a449,455
>       pcp = PCP_NETWORK_CONTROL # Adicionado em 30/05/2016
>       cfi = CFI_LLDP_PKT # Adicionado em 30/05/2016
>       vid = VID_LLDP_PKT # Adicionado em 30/05/2016
>       ethertype_lldp = ETH_TYPE_LLDP # Adicionado em 30/05/2016
>       vlan_pkt = vlan.vlan(pcp, cfi, vid, ethertype_lldp) # Adicionado
em 30/05/2016
>       pkt.add_protocol(vlan_pkt) # Adicionado em 30/05/2016
>
466a484,486
>         vlan_pkt = six.next(i) # Adicionado 31/05/2016
>         assert type(vlan_pkt) == vlan.vlan # Adicionado 31/05/2016
>
636,638c656,666
```

```
<                       match = ofproto_parser.OFPMatch(
<                           eth_type=ETH_TYPE_LLDP,
<                           eth_dst=lldp.LLDP_MAC_NEAREST_BRIDGE)
---
>                      #match = ofproto_parser.OFPMatch(
>                      #    eth_type=ETH_TYPE_LLDP,
>                      #    eth_dst=lldp.LLDP_MAC_NEAREST_BRIDGE) #
Modificado em 31/05/2016
>                      ## OFPCML_NO_BUFFER is set so that the LLDP is not
>                      ## buffered on switch
>                      #parser = ofproto_parser
>                      #actions =
[parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
>                      #
ofproto.OFPCML_NO_BUFFER
>                      #                                    )]
>                      #inst = [parser.OFPInstructionActions(
>                      #        ofproto.OFPIT_APPLY_ACTIONS, actions)]
>                      #mod = parser.OFPFlowMod(datapath=dp, match=match,
>                      #                        idle_timeout=0,
hard_timeout=0,
>                      #                        instructions=inst,
>                      #                        priority=0xFFFF)
>                      #dp.send_msg(mod)
```

Serviço de L2VPN SDN em Redes de *Backbone*: estudo de caso da REDECOMEP-Rio

Pedro Henrique Diniz da Silva, Natália Castro Fernandes, Nilton Alves Jr. e Márcio Portes de Albuquerque

***Resumo*— As redes OpenFlow introduzem o conceito de controladores centralizados para gerenciamento do comportamento do encaminhamento dos elementos de rede. Esse conceito permite que diversos serviços em redes de *backbone*, atualmente implementados através de soluções proprietárias, distribuídas pelos diversos elementos e de grande complexidade de operação, sejam simplificados facilitando a operação de rede. Esse trabalho propõe uma nova aplicação para provimento de conexões de Redes Privadas Virtuais de Camada 2 (L2VPN, do inglês *Layer 2 Virtual Private Network*) em redes de *backbone*, para caso de uso do *backbone* acadêmico, de pesquisa e de governo da cidade do Rio de Janeiro (REDECOMEP-Rio). São apresentados os algoritmos implementados através do protocolo OpenFlow e controlador Ryu. A solução proposta é avaliada através de cenários emulados por meio do emulador Mininet e de um estudo de caso executado na rede de produção da REDECOMEP-Rio.**

***Palavras-Chave*— L2VPN; Redes Definidas por Software; OpenFlow; Controlador Ryu.**

***Abstract*— OpenFlow networks introduce the concept of centralized controllers for managing the forwarding behavior of network elements. This concept allows multiple services in backbone networks, currently implemented through proprietary solutions, distributed by the various elements, and of highly complex operation, to be simplified to facilitate network operation. This paper proposes a new application for provision of Layer 2 Virtual Private Networks (L2VPN) in backbone networks for use case of academic, research, and government backbone network of the city of Rio de Janeiro (REDECOMEP-Rio). We present the algorithms implemented using the OpenFlow protocol and Ryu controller. We also evaluate the proposed solution through emulated scenarios using Mininet emulator and a case study implemented in the production network of Redecomep-Rio.**

***Keywords*— L2VPN; Software Defined Networks; OpenFlow; Ryu Controller.**

I. INTRODUÇÃO

A RedeRio Metropolitana (REDECOMEP-Rio) [1] é uma infraestrutura de fibras óticas próprias que formam uma rede de alta velocidade para as instituições de ensino, ciência, tecnologia, inovação e governo na cidade do Rio de Janeiro. Apesar de ser um *backbone* que atende instituições acadêmicas, a REDECOMEP-Rio, que opera como uma rede de núcleo IP tradicional puramente roteada, também atende a diversos outros tipos de instituições, as quais incluem empresas ligadas à prefeitura, ao estado e ao governo federal. Além de serviços de conectividade de alta velocidade à Internet e de alta confiabilidade, essas instituições também carecem de serviços que simulem redes privadas em cima dessa rede de núcleo compartilhada entre os diversos afiliados. Esse tipo de rede privada é também conhecido como Rede Privada Virtual (VPN, do inglês *Virtual Private Network*).

As VPNs mais comuns são as de camada 3 do modelo de referência TCP/IP (L3VPN) [2], ou seja, fazem com que dois elementos de rede pareçam estar diretamente conectados via uma rede roteada, ainda que existam diversos elementos de rede (roteadores) no meio do caminho. Entretanto, existem também as VPNs de camada 2. Esse tipo de rede virtual são conexões ponto-a-ponto que simulam um circuito físico de camada 2. Sua principal vantagem é a transparência do enlace virtual formado ponta-a-ponta, permitindo que qualquer protocolo, e não somente o IP, possa trafegar.

Apesar de cada uma das tecnologias apresentadas oferecer diversas vantagens e desvantagens uma em relação a outra, lidar com a variedade de protocolos para a criação de VPNs torna a gerência e operação da infraestrutura de rede mais complexa para os seus operadores. Essas tecnologias são difíceis de implementar, ou operacionalmente quase impossíveis de realizar em larga escala em redes IP puramente roteadas [3], como o caso da REDECOMEP-Rio, devido principalmente a complexidade de configuração dos equipamentos envolvidos. Alguns trabalhos, como [4] demonstram a utilização do protocolo OpenFlow para provimento de serviços de VPNs em redes MPLS (do inglês, *Multi Protocol Label Switching*), porém não é de conhecimento soluções de VPNs sobre redes IP tradicionais.

Com o enfoque em simplificar e otimizar o processo de operação de uma rede de núcleo puramente roteada, propõe-se uma solução de estabelecimento de circuitos L2VPN que evita a complexidade de ter protocolos de encapsulamento adicionais para o estabelecimento das conexões ponto-a-ponto, sem necessidade de configurações adicionais nos equipamentos envolvidos e centraliza a configuração das conexões em um ponto único da rede. Essa solução escala naturalmente ao passo que a mesma depende, basicamente, do processo de descoberta de topologia do protocolo OpenFlow.

A aplicação desenvolvida para o estabelecimento de L2VPNs instala regras de fluxos OpenFlow reativamente acada circuito que passa pelo *backbone*, baseado em um arquivo de configuração centralizado. Essa aplicação visa ser implementada na operação da REDECOMEP-Rio como um serviço na rede de produção.

Na próxima seção, é apresentada a arquitetura proposta de estabelecimento de L2VPNs. Apresenta-se, também, na Seção III, uma avaliação do protótipo e um estudo de caso

Pedro Henrique Diniz da Silva, Nilton Alves Jr. e Márcio Portes de Albuquerque¸ Coordenação de Atividades Técnicas, Centro Brasileiro de Pesquisas Físicas, Rio de Janeiro-RJ, Brasil, E-mails: phds@cbpf.br, naj@cbpf.br, mpa@cbpf.br. Natália Castro Fernandes¸ Departamento de Engenharia de Telecomunicações, Universidade Federal Fluminense, Niterói-RJ, Brasil, E-mail: nataliacf@id.uff.br.

na REDECOMEP-Rio. O artigo é concluído na Seção IV.

II. Características Principais da Aplicação Proposta

Uma rede de *backbone* IP consiste de múltiplos roteadores do tipo *Provider Edge (PE)*, que conectam o roteador de borda *Customer Edge* (CE) à rede de núcleo do provedor, e roteadores do tipo *Provider* (P), os quais funcionam como um roteador de trânsito na rede de núcleo entre os PE, conforme mostrado na Fig. 1.

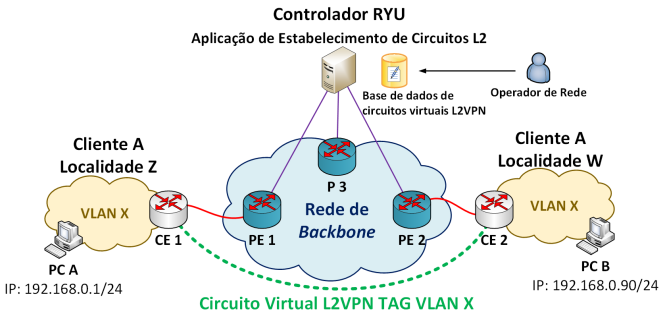


Fig. 1. Modelo básico da visão da aplicação de estabelecimento de circuitos L2.

Na solução proposta, os administradores de rede do cliente informam aos operadores de rede do provedor as *tags* de VLAN 802.1q [5] utilizadas em cada localidade nas quais desejam estabelecer um circuito L2VPN, e os operadores somente precisam armazenar as informações do circuito (como, por exemplo, *tag* de VLAN e roteadores PE envolvidos) em uma base de dados. A aplicação, então, é responsável por direcionar o tráfego de maneira adequada através da rede de núcleo, baseando-se nas informações obtidas através da base de dados e utilizando o algoritmo de escolha de menor caminho *Shortest Path First* (SPF) [6].

Nesta seção, primeiramente, são descritas as características principais do protocolo OpenFlow utilizadas nessa solução. Em seguida, é descrito como o operador de rede configura sua base de dados de estabelecimento de circuitos L2VPN. Em sequência, é descrito a utilização do algoritmo de menor caminho SPF para estabelecimento entre os roteadores PE.

A. Características Principais do Protocolo OpenFlow

O OpenFlow é um protocolo que permite que tabelas de fluxos em *switches* e roteadores sejam remotamente gerenciadas por um controlador. O protocolo define um fluxo como uma tupla com valores tais como: porta física de entrada, endereço MAC de origem e destino, campo tipo do cabeçalho *Ethernet*, identificador de VLAN, endereços IP de origem e de destino, campo protocolo do cabeçalho IP, porta de origem e destino do protocolo de camada de transporte. O *pipeline* de tabelas de fluxos em um *switch* OpenFlow mapeia a definição de um fluxo através dessa tupla em uma ação a ser tomada nos pacotes que pertencem a esse fluxo. Algumas dessas ações podem ser descartar os pacotes, encaminhá-los em uma porta física específica ou em um conjunto delas, ou enviar os pacotes para o controlador. Em caso de um pacote não corresponder a nenhuma das entradas da tabela de fluxos, o *switch* pode ser configurado para armazenar em um *buffer*, encapsular e enviar o pacote ao controlador para inspeção. Quando o controlador toma a decisão referente ao que fazer com todos os pacotes com aquela característica descrita pela tupla de campos do cabeçalho, ele adiciona uma entrada para esse fluxo na tabela de fluxos para armazenar essa decisão. Além disso, existe também a opção do controlador instalar entradas pró-ativamente para que esse processo não seja novamente repetido. Essas entradas são também conhecidas como regras.

O OpenFlow também implementa um conjunto de mensagens de sinalização e controle trocadas entre o controlador e o *switch*. Essas mensagens são responsáveis por realizar diversas ações como: verificação de características, configuração, modificação de estados, leitura de estados, envio de pacotes e mensagens de barreira. Dentre os tipos de mensagens mais relevantes estão o *packet_in*, o *packet_out* e o *flow_mod*. O *packet_in* é uma mensagem assíncrona enviada do *switch* para o controlador para notificar a chegada de um fluxo não classificado. O *packet_out* é uma mensagem enviada do controlador para o *switch*, em resposta a um *packet_in*, indicando qual ação deve ser tomada para aquele pacote. Já o *flow_mod* é também enviado do controlador para o *switch* para modificar o estado do mesmo, podendo realizar diversos comandos como adição e modificação de entradas na tabela de fluxos.

Em nossa solução de L2VPN, o controlador realiza algumas funções principais como: o procedimento de descoberta de topologia; verificação dos *endpoints* (*switches* PE) do circuito virtual L2VPN a ser estabelecido; escolha do menor caminho entre todos os *switches* na rede e encaminhamento do pacote na porta do *switch* associada ao caminho entre os *endpoints*; realização do VLAN *stitching*, garantindo a consistência do circuito fim-a-fim. Essas funções serão descritas em mais detalhes adiante.

B. Verificação dos Endpoints e VLAN Stitching

A aplicação proposta se baseia nos dados armazenados em duas bases de dados: (i) gerenciada pelo operador de rede para armazenar a configuração do circuito virtual; (ii) utilizada para o processo de VLAN *stitching* no núcleo da rede. No caso da base de configuração a cada cinco minutos, cada entrada de configuração é checada em sequência em busca de modificações.

Na base de dados de configuração, cada entrada é representada por uma tupla de seis valores: porta *switch* PE 1, *switch* PE 1, VLAN ID de acesso no *switch* PE 1, porta *switch* PE 2, *switch* PE 2 e VLAN ID de acesso no *switch* PE 2.

Para lidar com as situações em que um mesmo cliente utiliza *tags* de VLANs distintas nos locais em que é atendido, e para decisão de qual *tag* é utilizada no núcleo da rede sem que haja sobreposição, outra base de identificação é utilizada. Nela, cada circuito passa a ser identificado por uma tupla de quatro valores: *switch* PE 1, VLAN ID de acesso no *switch* PE 1, *switch* PE 2, VLAN ID de acesso no *switch* PE 2. A partir desses dados, seleciona-se sequencialmente em função das *tags* já em uso no núcleo, qual a próxima a ser utilizada e

armazena-se essa informação na base, conforme ilustrado na Fig. 2.

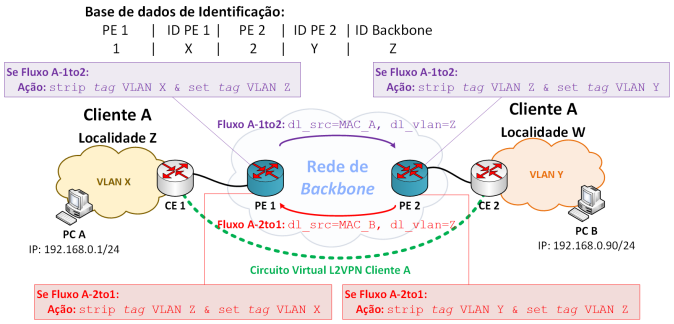


Fig. 2. Modelo básico da visão da aplicação de estabelecimento de circuitos L2 para processo de VLAN *stitching*.

C. Escolha do Menor Caminho e Estabelecimento do Circuito

A cada pacote do tipo *packet_in*, o controlador analisa as informações dos cabeçalhos do pacote, verificando para cada entrada de configuração se: (i) o *switch* de entrada do pacote é um dos *endpoints* do circuito; (ii) a porta de entrada do *switch* é referente a um dos circuitos; (iii) a *tag* de VLAN do pacote é igual ao VLAN ID de entrada de algum circuito. Caso essas informações se confirmem, o *switch* realiza as ações de associar o endereço MAC de origem do pacote ao *endpoint*, de substituir a *tag* de entrada pela ID de *backbone* do circuito e de encaminhar o pacote. Ao chegar no *endpoint* de destino a operação inversa é realizada. Caso o *switch* que recebe o pacote não seja um *endpoint*, mas a *tag* seja associada a um ID de circuito no *backbone* e o MAC de origem esteja vinculado a um dos *endpoints*, o pacote é encaminhado para o outro *endpoint*. Então, para estabelecer o caminho entre o *switch* que envia a mensagem de *packet_in* e o *switch* PE de destino do circuito, a aplicação utiliza o algoritmo de *Shortest Path First* [6]. Dessa forma, a aplicação é capaz de estabelecer o circuito L2VPN fim-a-fim reativamente a cada mensagem de *packet_in*, encaminhando o fluxo à porta de saída de cada *switch*, obtida em função do algoritmo SPF em conjunto com as bases de dados de configuração e de identificação de circuitos. Para evitar que ocorram *loops* na topologia utilizou-se também o protocolo *Spanning Tree* (STP).

A aplicação proposta instala uma regra com *soft-timeout* de 60 segundos para poder lidar com o processo de alteração da configuração de um circuito. Então, após uma mudança por parte do operador de rede, há a necessidade de aguardar: (a) um período de 60 segundos de inatividade para que a entrada da tabela de fluxo de cada *switch* no caminho seja deletada; (b) um período de 5 minutos para que a configuração possa ser verificada novamente e o novo circuito possa ser estabelecido a partir de novas mensagens de *packet_in* - podendo ambos os intervalos (a) e (b) ocorrerem concorrentemente.

III. Implementação e Avaliação do Funcionamento da Aplicação

Nesta seção, buscou-se avaliar o tempo de convergência da aplicação visto pelo *switch* CE, definido como o intervalo de tempo entre o início do funcionamento da aplicação e o momento em que os protocolos STP e SPF já convergiram, ou seja, quando os *switches* PE OpenFlow iniciam o encaminhamento de pacotes baseados nas mensagens de *packet_in* e *packet_out*. Para tal, os resultados foram subdivididos em dois cenários: topologias emuladas e estudo de caso da implantação no *Backbone* REDECOMEP-Rio. A avaliação de desempenho ilustra como o procedimento para o estabelecimento do circuito é eficaz e como o sistema escala adequadamente, aparentando ter pouco *overhead* em função da complexidade da rede. A complexidade, nesse caso, é considerada como o número total de nós, isto é, o número total de *switches* mais *hosts*.

Para validar a proposta nos cenários de topologias emuladas, construiu-se um protótipo utilizando o OpenVswitch (um *switch* OpenFlow via *software*) e o controlador OpenFlow Ryu, por meio do emulador de redes Mininet [7]. A aplicação Ryu instala as regras nos *switches* do caminho do circuito L2VPN por meio do protocolo OpenFlow v1.3. Nesse protótipo, todos os procedimentos de verificação do circuito e de escolha de menor caminho foram desenvolvidos conforme descritos nas Seções II-B e II-C.

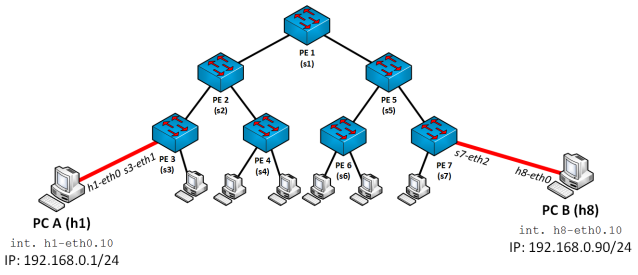
Para avaliar o funcionamento da aplicação, foi medido o tempo de convergência na rede, considerando o uso do algoritmo STP, disponibilizado por padrão no controlador Ryu, com o algoritmo SPF, desenvolvido para a aplicação, e com algoritmo proposto de estabelecimento de conexões L2VPN. Emulou-se o processo de estabelecimento de uma conexão entre dois *hosts* de um mesmo cliente, assumindo que o circuito já estava previamente descrito no arquivo de configuração. Foi medido o intervalo de tempo entre o primeiro pacote ARP *Request* enviado e sem resposta e o primeiro pacote ARP *Request* que recebe resposta. Os resultados são apresentados com um intervalo de confiança de 95%.

A. Cenários Emulados

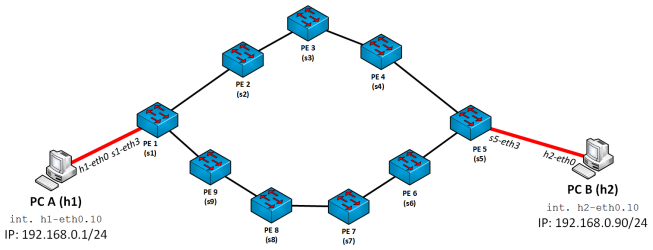
Para realizar a avaliação do funcionamento da aplicação, emulou-se três diferentes topologias. Utilizou-se duas topologias padrão do Mininet e uma customizada, conforme descritas em detalhes adiante. Para geração dos pacotes para estabelecimento de conexão e geração dos pacotes ARP, utilizamos a ferramenta FPING [8], com intervalos entre mensagens ICMP *Echo Request* de 10 ms. Os intervalos entre os pacotes ARP foram checados através da ferramenta *tcpdump* [9], capturando pacotes na interface do *host* de origem.

1. **Topologia Linear com 2 *Switches*:** Como avaliação inicial, utiliza-se a topologia mais simples disponível no emulador com dois *switches*. Os *switches* são conectados diretamente e dois *hosts* são conectados cada um a um dos *switches*. ***2.*** **Topologia *Fat-tree* com Três Níveis e Sete *Switches*:** A topologia *fat-tree* é um tipo de topologia voltada para *data centers*. Realizou-se a avaliação de funcionamento em uma topologia de três níveis com sete *switches* e oito *hosts* conectados de acordo com a Fig. 3(a). ***3.*** **Topologia Emulada do *Backbone* REDECOMEP-Rio:** Como a aplicação visa ser implementada na rede em produção da REDECOMEP-Rio, optou-se por realizar uma emulação em uma infraestrutura

similar à sua topologia real. A topologia de *backbone* da REDECOMEP-Rio, atualmente, é composta por nove Pontos de Presença (PoP, do inglês *Point-of-Presence*) espalhados por uma topologia em anel com múltiplas redundâncias. A Fig.3(b) exibe o *layout* da topologia da rede em produção simplificada. Foram conectados nove *switches* em anel com dois *hosts*. Nesse cenário, cada *host* faz o papel dos *switches* CE que devem ser conectados aos *switches* PE de *backbone*. Vale ressaltar que como a topologia é em anel, ocorrem *loops* do *Spanning Tree*.



(a) Fat-tree com sete *switches* e oito *hosts*.



(b) Emulação da REDECOMEP-Rio com 9 *switches* e 2 *hosts*.

Fig. 3. *Layout* das topologias emuladas.

Os resultados referentes aos cenários emulados podem ser visualizados na Fig. 4(a). O tempo de convergência é em grande parte dominado pelo tempo de convergência do STP, que para todos os casos apresentados é de 30s. Isso ocorre de acordo com o apresentado em [10]. A partir do instante em que o STP converge a aplicação demora somente 1,5s para iniciar o encaminhamento de pacotes. Assim como os casos em que não ocorrem *loops*, o tempo de convergência é em grande parte dominado pelo tempo de convergência do STP, cerca de 30s. Porém, após uma porta entrar em estado *BLOCKED* pelo STP, é necessário mais 13,5s para que o LLDP, implementado por padrão no Ryu, identifique que o enlace está bloqueado e a aplicação, a partir de então, possa recalcular os caminhos. Desde o momento em que o enlace passa a estar bloqueado, a aplicação leva cerca de 1s para encaminhar os pacotes.

Cabe ainda ressaltar que a partir do momento em que a aplicação já convergiu, o estabelecimento do circuito depende somente da latência entre os *switches* e do tempo em que as mensagens de *packet_out* e *flow_mod* demoram para serem processadas e enviadas pelo controlador. Como pode ser observado na Fig. 4(b), a função de distribuição acumulada (CDF) dos tempos de processamento dessas mensagens demonstra que esse intervalo representa cerca de 0,017% do tempo total de convergência para o pior dos casos de processamento de mensagens *flow_mod*, ou seja, a topologia emulada *fat-tree*. Logo, isso mostra que a partir da convergência da aplicação, o tempo para estabelecimento do circuito é muito pequeno.

B. Estudo de Caso da Implantação Inicial no Backbone REDECOMEP-Rio

Apresenta-se, a seguir, a avaliação do sistema protótipo na REDECOMEP-Rio. O estudo de caso foi executado em uma configuração com equipamentos híbridos, ou seja, realizando o encaminhamento dos pacotes da rede em produção no mesmo enlace que os pacotes SDN/OpenFlow. O cenário descrito foi avaliado utilizando os equipamentos: 2 (PE) x Cisco ASR9000 rodando IOS XR 5.1.3 equipados com pelo menos 6 GB DRAM; 1 (CE) x Cisco Catalyst WS-C3560G-24TS rodando IOS 12.2(50)SE5; 1 (CE) x Cisco Catalyst WS-C3750G-24TS-1U rodando 12.2(25)SEE2. A associação entre os roteadores Cisco ASR9000 e o controlador ocorre *in-band*. Todos os dispositivos são conectados com enlaces de 1 Gbps de capacidade, conforme mostrado na Fig. 5. A conexão entre o *switch* CE e o roteador PE ocorre através de uma interface do tipo tronco com permissão de tráfego das VLANs 802.1q 10 e 100.

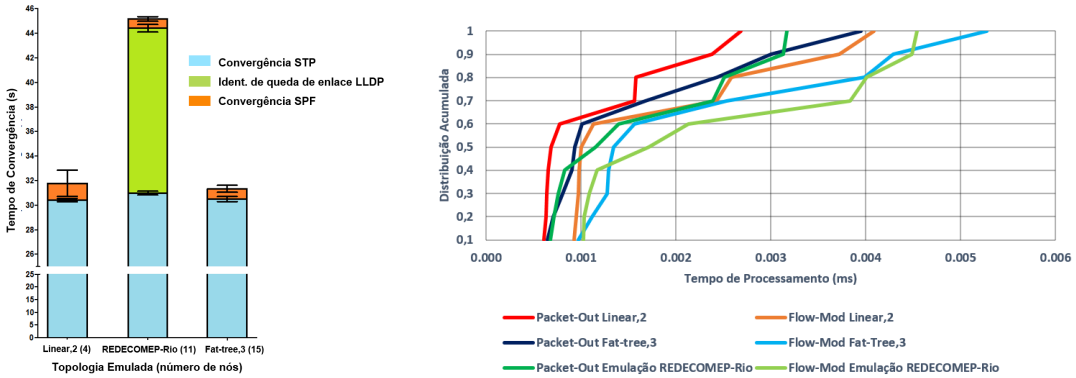
De acordo com o que pode ser observado na Fig. 5, fez-se necessário o estabelecimento de dois enlaces virtuais entre os roteadores PE para o correto funcionamento da aplicação. Além do transporte dos pacotes com *tags* de VLAN 802.1q para o estabelecimento de circuitos, também é necessário o transporte dos pacotes LLDP para o mapeamento da topologia. Desse modo, para cada tipo de enlace virtual foi permitido um determinado tipo de encapsulamento de pacotes. Isto é, para o transporte dos pacotes dos circuitos com encapsulamento dot1q configurou-se uma interface com *encapsulation dot1q any*, em vermelho na Fig. 5, enquanto que para os pacotes de controle LLDP sem encapsulamento foi configurado outro tipo de interface com *encapsulation default*, em laranja na Fig. 5.

Assim como o exposto na Seção III-A, o tempo de convergência da aplicação também foi avaliado, *i.e.*, o intervalo de tempo entre o pacotes ARP *Request* inicial e o que recebe resposta foi determinado. Entretanto, em vez de utilizar a ferramenta FPING foi utilizado o utilitário *ping* disponível por padrão nos *switches*, também com intervalo de 10ms entre ICMP *Echo Request*, e para análise dos resultados a ferramenta *tcpdump*. As requisições foram enviadas a partir do *switch* denominado Catalyst 3560 na Fig. 5.

A Fig.6 exibe o resultado comparativo entre as CDFs dos tempos de convergência do cenário emulado de topologia linear descrito na Seção III-A e do estudo de caso descrito na presente seção. Pode-se observar uma diferença de até 8s entre o menor valor para o cenário emulado e o maior valor do estudo de caso. Essa diferença ocorre, principalmente, devido ao intervalo entre tentativas de associação entre controlador e roteador ASR9000 serem fixadas em 8s, enquanto que o intervalo referente ao OpenVswitch ser de apenas 1s.

IV. Conclusão

Redes de *backbone* têm como uma de suas principais demandas o provimento de serviços de estabelecimento de redes virtuais privadas. Um dos principais tipos de redes são as VPNs de camada 2 ou L2VPNs. A solução proposta de



(a) Tempo total de convergência em relação ao número total de nós na rede. (b) CDF do tempo de processamento das mensagens de *packet_out* e *flow_mod*.

Fig. 4. Resultados obtidos para o cenários emulado.

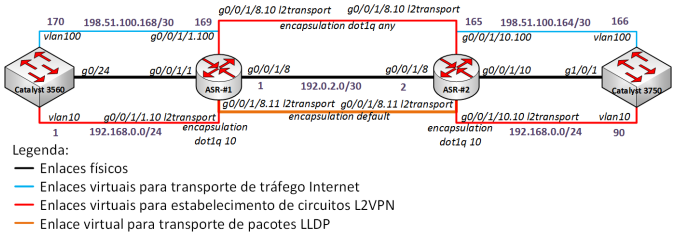


Fig. 5. Cenário de estudo de caso da avaliação do funcionamento da aplicação de estabelecimento de circuitos L2VPN no *backbone* da REDECOMEP-Rio.

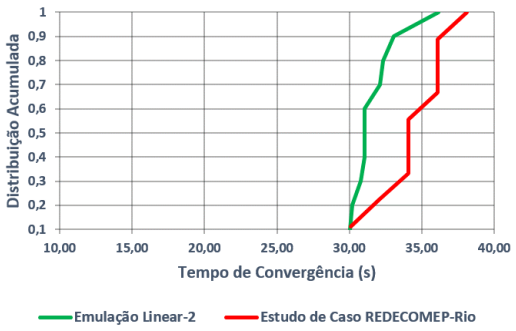


Fig. 6. CDF do tempo total de convergência da aplicação para os cenários emulado linear e de estudo de caso.

estabelecimento desse tipo de circuitos instala regras reativamente nos *switches* da rede utilizando o controlador OpenFlow Ryu. Essa solução se difere, principalmente, das disponíveis no mercado atualmente, pois não é necessário em nenhum momento que o operador tenha conhecimento do funcionamento do protocolo, nem sequer tenha que alterar as configurações dos equipamentos. As avaliações mostram que o sistema escala bem para redes com até quinze nós de *backbone*, dependendo em grande parte do tempo de convergência do protocolo *Spanning Tree*, e se há ou não *loops* na topologia. Esse efeito ocorre devido ao fato de que o protocolo LLDP leva cerca de 13s após a convergência do STP para detectar o bloqueio de um enlace. Além disso, a diferença encontrada entre o cenário emulado e o implantado, mostra que o tempo de associação entre o controlador e o *switch* OpenFlow tem um pequeno impacto no tempo total de convergência da aplicação. Ademais, o impacto dos atrasos entre os *switches*, e entre eles e o controlador no tempo de estabelecimento de circuito é pequeno, devido ao fato dos tempos de processamento de mensagens de *packet_out* e *flow_mod* somados a esses atrasos serem muito pequenos em relação ao tempo total de convergência. Assim, após o período de inicialização da rede, o tempo para estabelecimento automático da L2VPN é inferior a 10 ms, o que é um excelente tempo quando comparado ao processo tradicional para a configuração e estabelecimento desse tipo de circuito.

Portanto, em função dos resultados obtidos, a implementação da aplicação em todo o *backbone* em produção da REDECOMEP-Rio está em fase de finalização, onde o impacto de sua utilização nos usuários da rede deve ser avaliado, posteriormente.

REFERÊNCIAS

[1] V. Zepeda, "Instituicoes cientificas do estado ganham internet de altissima velocidade," *Arquivo de Noticias, FAPERJ*, 05 jun. 2014, Accessed: 2016-03-29. [Online]. Available: http://www.faperj.br/?id=2691.2.2

[2] P. Knight and C. Lewis, "Layer 2 and 3 virtual private networks: taxonomy, technology, and standardization efforts," *Communications Magazine, IEEE*, vol. 42, no. 6, pp. 124–131, 2004.

[3] E. D. Osborne and A. Simha, *Traffic engineering with MPLS*. Cisco Press, 2002.

[4] A. R. Sharafat, S. Das, G. Parulkar, and N. McKeown, "MPLS-TE and MPLS VPNs with OpenFlow," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 452–453.

[5] *IEEE 802: Local and Metropolitan Area Network Standards*, IEEE Standard 802.1q, 2014. [Online]. Available: http://standards.ieee.org/getieee802/download/802-1Q-2014_mibs.zip

[6] T. J. Misa and P. L. Frana, "An Interview with Edsger W. Dijkstra," *Commun. ACM*, vol. 53, no. 8, pp. 41–47, Aug. 2010. [Online]. Available: http://doi.acm.org/10.1145/1787234.1787249

[7] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX. New York, NY, USA: ACM, 2010, pp. 19:1–19:6. [Online]. Available: http://doi.acm.org/10.1145/1868447.1868466

[8] D. Schweikert. (2011) Fping. [Online]. Available: http://fping.org/

[9] F. Fuentes and D. C. Kar, "Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose," *Journal of Computing Sciences in Colleges*, vol. 20, no. 4, pp. 169–176, 2005.

[10] P. T. RYU. (2014) Ryubook. [Online]. Available: http://osrg.github.io/ryu-book/en/Ryubook.pdf