UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
ESPECIALIZAÇÃO EM TELEINFORMÁTICA E REDES DE COMPUTADORES

FELIPE STALL RECHIA

ESTUDO DE REDES DEFINIDAS POR SOFTWARE E A SUA IMPLANTAÇÃO EM REDES CORPORATIVAS

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2014

FELIPE STALL RECHIA

ESTUDO DE REDES DEFINIDAS POR SOFTWARE E A SUA IMPLANTAÇÃO EM REDES CORPORATIVAS

Monografia apresentada ao Programa de Pós-Graduação em Tecnologia da Universidade Tecnológica Federal do Paraná, Campus Curitiba, como requisito parcial à obtenção do título de Especialista em Teleinformática e Redes de Computadores – Área de Concentração: Telemática.

Orientador: Prof. Dr. Kleber Kendy Horikawa Nabas

CURITIBA

2014

Aos meus pais pelos ensinamentos diretos e indiretos de valores e princípios morais e éticos que hoje constituem a essência do meu caráter.

Aos meus irmãos e amigos pelas discussões científicas e filosóficas que sempre serviram de estímulo para seguir valorizando e buscando conhecimento e competência.

À Fernanda pelo companheirismo, cumplicidade e amor demonstrados mesmo nos momentos de maior tensão profissional, acadêmica e pessoal.

À memória do Prof. Godoy, em cujas aulas tive a honra de presenciar disposição incrível para ensinar e preocupação sincera com o aprendizado dos alunos.

A todos os professores do curso, pela disposição e paciência em ensinar.

À Datacom e aos meus colegas de trabalho, que me proporcionaram enorme crescimento profissional e pessoal ao longo dos últimos anos.

AGRADECIMENTOS

Agradeço à Datacom pelo apoio financeiro e técnico, em especial aos meus colegas de trabalho João Strapasson e Adriano Fávaro, por estarem sempre dispostos a discutir temas relacionados a SDN. Ambos contribuíram com ideias e sugestões de pesquisa que incluí no trabalho.

Agradeço ao Prof. Dr. Nick Feamster e seus assistentes de ensino pela elaboração do curso online sobre SDN, ministrado à distância através do coursera.org, a partir do qual aprendi muito sobre SDN e seus diversos protocolos e ferramentas. A elaboração de grande parte do conteúdo deste trabalho só foi possível graças ao conhecimento adquirido nas aulas e referências bibliográficas indicadas durante o curso.

Agradeço ao Prof. MsC Márcio Luiz Ferreira Miguel pela orientação em etapa inicial em que contribuiu com ideias e sugestões de fontes de consulta sobre SDN, além de ter indicado o curso sobre SDN do coursera.org.

Agradeço ao Prof. Dr. Paulo José Abatti pelas orientações precisas sobre como elaborar a monografia, especialmente pelo esclarecimento de dúvidas sobre citações e referência bibliográficas.

Agradeço ao Prof. Dr. Kleber Kendy Horikawa Nabas pela orientação, e aos Professores Eng. Alexandre Cardoso, MBA, e Dr. Armando Rech pela disposição em revisar e sugerir melhorias para este trabalho.

Agradeço aos meus colegas e amigos Jean Manganelli e Evaldo Fortunato pelo companheirismo e pelo auxílio prestado na elaboração deste trabalho com ideias, conversas e churrascos.

The “SDN Age of Networking” is about finding Visibility, Mobility and Operability in the software layer of the network. This software layer offers the business new ways to extract productivity and profits. (FERRO, Greg, 2014)

A “Era das redes SDN” consiste em extrair visibilidade, mobilidade e operabilidade na camada de software da rede. Essa camada de software oferece ao negócio novas maneiras de extrair produtividade e lucros. (FERRO, Greg, 2014)

RESUMO

RECHIA, Felipe. Estudo de Redes Definidas por Software e a sua Implantação em Redes Corporativas. 110 f. Monografia (Especialização em Teleinformática e Redes de Computadores) – Programa de Especialização em Teleinformática e Redes de Computadores (Área de Concentração: Telemática), Universidade Tecnológica Federal do Paraná. Curitiba, 2014.

Este trabalho primeiramente apresenta uma evolução histórica de redes programáveis e algumas das tecnologias de redes definidas por software (SDN). Adota-se a convenção de representar a arquitetura SDN através de três camadas: a Interface API Norte, a Plataforma de Controle e a Interface API Sul. Alguns dos conceitos de rede mais básicos são analisados e explicados, como a comutação de quadros Ethernet e a comutação de pacotes IP. Além disso também são explicadas algumas das funcionalidades mais comuns utilizadas em redes corporativas locais, como limitação de endereços MAC por porta, listas de controle de acesso (ACLs) e a lógica do protocolo Spanning-tree (STP). Algumas dessas funcionalidades de rede são então demonstradas utilizando o protocolo SDN da interface API Sul mais disseminado: o OpenFlow, implementado utilizando o emulador de redes Mininet e o controlador POX. Em seguida uma rede corporativa arbitrária é criada e configurada utilizando-se uma topologia simulada com o Packet Tracer para fins de prova de conceito e comparação de funcionalidades de redes corporativas com funcionalidades implementadas em SDN. Baseando-se no caso da topologia apresentada, um esboço de plano de migração é traçado para implantar uma rede SDN em substituição à rede tradicional. Por fim são discutidos aspectos negativos e positivos da implantação de uma arquitetura SDN.

Palavras-chave: SDN. Funções de Rede. Redes Corporativas.

ABSTRACT

RECHIA, Felipe. A study of Software Defined Networking and its Deployment on Campus Networks. 110 f. Thesis (Certificate Program in Telecomputing and Computer Networks) – Certificate Program in Telecomputing and computer Networks (Concentration Area: Telematics), Federal University of Technology - Paraná. Curitiba, 2014.

This thesis first presents an historical evolution of programmable networks and some of the current software-defined networking (SDN) technologies. The chosen SDN convention to describe a SDN architecture consists of three layers: the northbound API, the control platform, and the southbound API. Some of the most basic networking concepts are then analyzed, such as Ethernet switching and IP routing, and some of the most common corporate area network functions are explained, such as MAC address limits per port, access control lists (ACLs) and the Spanning-tree logic (STP). Some of these network functions are then demonstrated using a well-known southbound control protocol: OpenFlow, implemented using the Mininet network emulator and the POX controller. An arbitrary corporate area network is built and configured using a Packet Tracer simulated topology, which is used as a proof-of-concept network to compare traditional corporate area network features with SDN-based ones. Based on the presented network case, a migration plan draft is created for deploying SDN as a replacement. Finally, some of the disadvantages and advantages of SDN architecture adoption are discussed.

Keywords: SDN. Network Functions. Corporate Area Networks.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

4D	Decision, Dissemination, Discovery and Data
ACL	Access Control List
ALTO	Application-Layer Traffic Optimization
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
AT&T	American Telephone and Telegraph Company
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
BGP-LS	Border Gateway Protocol – Link State, extensão do protocolo BGP
CDN	Content Delivery Network
CLI	Command Line Interface
CPqD	Centro de Pesquisa e Desenvolvimento em Telecomunicações
DCAN	Devolved Control of ATM Networks
DHCP	Dynamic Host Configuration Protocol
DLF	Destination Lookup Failure
DNS	Domain Name Server
DPI	Deep Packet Inspection
eBGP	External Border Gateway Protocol
EEPROM	Electrically-Erasable Programmable Read-Only Memory
ETSI	European Telecommunications Standards Institute
FML	Flow-based Management Language
GMPLS	Generalized Multi-Protocol Label Switching
GSMP	General Switch Management Protocol
HP	Hewlett Packard
iBGP	Internal Border Gateway Protocol
IBM	International Business Machines
ICMP	Internet Control Message Protocol
IDR	Inter-Domain Routing
IDS	Intrusion Detection System

IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IP	Internet Protocol
LAN	Local Area Network
LSP	Label Switched Path
LSR	Label Switching Router
MAC	Media Access Control
MAN	Metropolitan Area Network
MPLS	Multi-Protocol Label Switching
NaaS	Networking as a Service
NAT	Network Address Translation
NFV	Network Functions Virtualization
NLRI	Network Layer Reachability Information
NOS	Network Operating System, da Mojatatu Networks
NTP	Network Time Protocol
ONF	Open Networking Foundation
OSPF	Open Shortest Path Protocol
OVS	Open vSwitch
OVSDB	Open vSwitch Database Management Protocol
PC	Personal Computer
PCC	Path Computation Client
PCE	Path Computation Element
PCEP	Path Computation Element Protocol
RADIUS	Remote Authentication Dial In User Service
RCP	Routing Control Platform
RFC	Request for Comments (IETF)
RPC	Remote Procedure Call
RTT	Round Trip Time
SAL	Service Abstraction Layer
SDN	Software Defined Networking
SNAC	Simple Network Access Control, controlador OpenFlow baseado em NOX

SNMP	Simple Network Management Protocol
SR	Simple Router
SSL	Secure Sockets Layer
STP	Spanning Tree Protocol
TACACS	Terminal Access Controller Access-Control System
TCP	Transmission Control Protocol
TE	Traffic Engineering
UDP	User Datagram Protocol
VID	VLAN ID
VLAN	Virtual LAN
VM	Virtual Machine
WAN	Wide Area Network
WG	Working Group

LISTA DE ACRÔNIMOS

NETCONF	Network Configuration Protocol
SSH	Secure Shell
OF-Config	OpenFlow Config
ForCES	Forwarding and Control Element Separation
P2P	Peer to Peer
LTProtocol	Length-Type-Based Protocol Client/Server
RFProxy	RouteFlow Proxy
nmap	Network Mapper

SUMÁRIO

INTRODUÇÃO GERAL

Nos últimos anos o termo SDN, *Software Defined Networking*, vem se popularizando cada vez mais entre as empresas que produzem software para virtualização de servidores e computação em nuvem (KIRKPATRICK, 2013).

O termo SDN representa um conceito de arquitetura de rede na qual o controle dos fluxos de tráfego é feito por uma entidade independente dos equipamentos que encaminham os fluxos de tráfego. Ou seja, a camada de controle é separada da camada de tráfego de dados. O plano de controle, por sua vez, atua de acordo com aquilo que é definido pelas aplicações que utilizam a rede, conforme demonstrado na Figura 1.

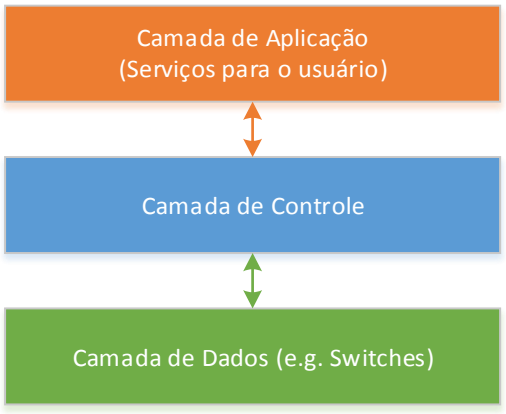

Figura 1 – Arquitetura SDN Simplificada

De acordo com (ONF, 2013a), a arquitetura SDN apresenta as seguintes características principais:

- Separação entre plano de controle e plano de dados;
- Um controlador central (e.g. servidor) gerencia os elementos de rede (e.g. switches, roteadores) e tem visão global da topologia completa;
- As interfaces dos planos de controle e dados são abertas. Ou seja, a interoperabilidade entre dispositivos de diferentes fabricantes é garantida por padrões abertos;
- É possível “programar” a rede através de aplicações externas.

Para efeito de comparação, pode-se tomar como exemplo uma rede TCP/IP tradicional utilizando o protocolo de roteamento Open Shortest Path First (OSPF). Cada elemento da rede recebe informações dos vizinhos e faz um cálculo próprio da topologia, determinando as melhores rotas para alcançar os diferentes

destinos da rede. O fato de cada elemento da rede ter que calcular suas rotas com base em toda a topologia é algo que torna o funcionamento da rede pouco eficiente. É uma duplicação de consumo de recursos de processamento e energia, segundo (LIMONCELLI, 2012).

A proposta de uma arquitetura SDN é tornar os elementos de rede responsáveis por comutar o tráfego (*switches* e roteadores) tão especializados quanto for possível. Não há necessidade de que cada elemento tenha que processar algoritmos complexos para calcular as melhores rotas para alcançar destinos em uma rede. Ao receber um novo pacote a ser encaminhado para um destino ainda não conhecido localmente, tudo que um roteador de uma SDN precisa fazer é consultar o controlador (LIMONCELLI, 2012).

Com essa arquitetura centralizada de controle da rede, basta que o controlador seja programado através de uma API (*Application Programming Interface*) para configurar a rede toda. Dessa forma pode-se realizar de maneira centralizada funções de cálculo e reserva de banda, otimização de roteamento, controle centralizado de acesso de usuários, NAT (*Network Address Translation*), balanceamento de tráfego na rede (LIMONCELLI, 2012).

Portanto, o software que interage com o controlador através da API é o software que controla a rede. Se for necessário mudar o algoritmo de roteamento ou implementar mais serviços na rede, basta trocar o software, simplificando quaisquer mudanças que sejam necessárias e tornando mais flexível a utilização de recursos (LIMONCELLI, 2012).

OBJETIVO

Comparar diferentes tecnologias e arquiteturas SDN com tecnologias e arquiteturas de redes clássicas, procurando determinar quais vantagens e desvantagens podem ser introduzidas pelas novas tecnologias. Outro aspecto a ser explorado é determinar que passos devem ser tomados para fazer a migração de uma rede clássica para uma rede SDN em um caso de uso de rede corporativa.

JUSTIFICATIVA

O objetivo de uma SDN é prover serviços de comunicação eficientes, projetada desde o princípio para atender serviços de forma dinâmica e flexível (SEZER et al, 2013) (KIRKPATRICK, 2013). A promessa das SDNs é simplificar a instalação e operação de redes, e baixar os custos de gerenciamento de redes de empresas e operadoras ao prover serviços de rede programáveis (SEZER et al, 2013).

Entretanto, a migração para redes SDN apresenta vários desafios. É necessário entendimento pleno das novas tecnologias para garantir que as SDNs atenderão requisitos de segurança, performance, escalabilidade e também interoperabilidade com redes legadas (SEZER et al, 2013).

DESCRIÇÃO DO TEXTO

A metodologia adotada para este projeto consiste em fazer uma revisão bibliográfica das diferentes tecnologias e arquiteturas SDN disponíveis. Será apresentada também uma breve revisão bibliográfica sobre o funcionamento de redes clássicas. Em seguida serão demonstrados alguns experimentos em redes virtuais utilizando-se ferramentas de simulação e emulação de redes de redes clássicas e redes SDN. Finalmente será feito um estudo de caso de uma pequena rede corporativa e traçado um plano de migração dessa rede para uma arquitetura SDN.

O capítulo 1 apresenta em mais detalhes a arquitetura de uma rede SDN e em seguida apresenta uma revisão bibliográfica das tecnologias e protocolos SDN disponíveis.

O capítulo 2 apresenta uma revisão bibliográfica de princípios de comutação e roteamento para redes LAN. São abordados: o princípio de aprendizado de endereço MAC Ethernet, para comutação, o funcionamento básico do protocolo Spanning-tree 802.1d, e princípios do roteamento IP.

O capítulo 3 aborda algumas das funcionalidades expostas no capítulo 2, ressaltando como poderiam ser implementadas em uma rede SDN emulada utilizando Mininet e POX.

O capítulo 4 apresenta o estudo de caso de uma pequena rede local corporativa que emprega as funcionalidades descritas no capítulo 2, comumente

adotadas nesse tipo de rede. Como prova de conceito, essa LAN é demonstrada com o Packet Tracer, um simulador de redes proprietário CISCO. Em seguida são apresentados alguns aspectos que podem ser utilizados para realizar um plano de migração de uma rede tradicional para uma rede SDN.

CAPÍTULO 1 – A ARQUITETURA SDN

HISTÓRICO

O termo SDN é relativamente novo, utilizado pela primeira vez em 2009, porém historicamente já existiram diversas tentativas de implementar arquiteturas em que as redes fossem mais facilmente programáveis. Grande parte das ideias e tecnologias que tornaram possível a implementação do que hoje se conhece por SDN foram concebidas há cerca de 20 anos (FEAMSTER et al, 2013).

Active Networking

Segundo Feamster et al (2013) e Nunes et al (2014), uma das primeiras tecnologias que possibilitava a programação da rede através de aplicações foi chamada de *Active Networking*. Criada em meados dos anos 90, essa tecnologia permitia que o administrador da rede tivesse acessos a recursos dos dispositivos da rede, tais como processamento, armazenamento e filas de pacotes e suportava a criação de funcionalidades personalizadas que podiam ser aplicadas aos pacotes que trafegavam pelos dispositivos.

A tecnologia de *Active Networking* não teve muito sucesso e não foi levada adiante principalmente por causa de preocupações com relação à segurança e à performance dos métodos utilizados (NUNES et al, 2014). Além disso, ambientes de *Data Centers*, que poderiam tirar vantagens mais rapidamente dessa tecnologia para implantar soluções de virtualização, ainda não estavam amplamente presentes (FEAMSTER, 2013)

Open Signaling

Em 1995 foi criado um grupo de trabalho chamado de OPENSIG, inicialmente com o objetivo de fazer o protocolo ATM (*Asynchronous Transfer Mode*), a Internet e as redes móveis abertas, extensíveis e programáveis. O principal elemento da proposta desse grupo foi de criar uma maneira de programar o hardware

dos dispositivos de rede através de uma interface aberta, que permitiria implantar novos serviços no ambiente de rede (NUNES et al, 2014).

Em seguida foi criado um grupo de trabalho do IETF, que levou à especificação de um protocolo chamado de GSMP, *General Switch Management Protocol*, que permite um controlador estabelecer e terminar conexões em um switch, adicionar ou deletar elementos em um grupo multicast e gerenciar recursos do equipamento, dentre outras funcionalidades. Esse grupo de trabalho foi oficialmente concluído e publicou o último padrão em junho de 2002, o GSMPv3 (NUNES et al, 2014)

DCAN

Ainda nos anos 90 também foi observada uma iniciativa chamada de DCAN (*Devolved Control of ATM Networks*). O objetivo desse projeto era de desenvolver uma infraestrutura de controle e gerência para redes ATM. A premissa é de que as funções de controle e gerência deveriam ser retiradas dos dispositivos da rede e colocadas em elementos externos (NUNES et al, 2014).

As arquiteturas DCAN também propunham uma solução em que era possível dividir uma única rede física em diversas redes lógicas, particionando recursos dos dispositivos da rede ATM (NUNES et al, 2014), de forma muito parecida com o que hoje é proposto com o OpenFlow e o FlowVisor (SHERWOOD et al, 2010).

RCP

Em 2004 foi criada uma plataforma de controle de roteamento, o RCP (*Routing Control Platform*), que propôs uma arquitetura em que cálculos de rotas seriam feitos de maneira logicamente centralizada por uma entidade única por um AS, ou sistema autônomo (*Autonomous System*). Os roteadores permanecem responsáveis apenas por efetivamente encaminhar os fluxos de tráfego conforme instruções recebidas do controlador RCP (FEAMSTER et al, 2004).

Conforme explicado por Feamster et al (2004), essa arquitetura propõe principalmente a substituição do eBGP e iBGP entre os elementos da rede em três etapas:

1. Substituir o iBGP “*route reflector*” pelo RCP, controlando um único AS;
2. Colocar o RCP como responsável por todas as sessões eBGP de um sistema autônomo com os sistemas autônomos vizinhos;
3. Permitir mudanças no roteamento entre domínios, fazendo com que a comunicação entre sistemas autônomos ocorra diretamente entre os RCPs responsáveis através de eBGP ou até mesmo um novo protocolo.

Embora tenha sido desenvolvida com o auxílio e adotada pela empresa americana AT&T, a arquitetura proposta pelo RCP não foi amplamente adotada.

Projeto 4D

Também em 2004, o projeto 4D propôs uma arquitetura “ficha limpa” (*clean slate*) de dividir a rede em camadas, uma camada de Decisão com visão global da rede, uma camada de Disseminação e subcamada de Descoberta responsáveis por realizar a comunicação com os elementos de rede e controlar o encaminhamento de Dados (*Decision, Dissemination, Discovery and Data*) (NUNES et al, 2014) (FEAMSTER, 2013).

Essa arquitetura inspirou diretamente o NOX, uma proposta de “sistema operacional para redes”, projetado para controlar uma rede baseada em OpenFlow (NUNES et al, 2014).

ARQUITETURA SDN

Embora existam algumas divergências quanto a nomenclaturas utilizadas para descrever uma arquitetura SDN, a maior parte das referências bibliográficas tem um ponto em comum ao descrever essa arquitetura: a separação das redes em camadas de aplicação, controle e dados, conforme ilustrado na Figura 2, adaptada e traduzida de (KIRKPATRICK, 2013) (FEAMSTER, 2013).

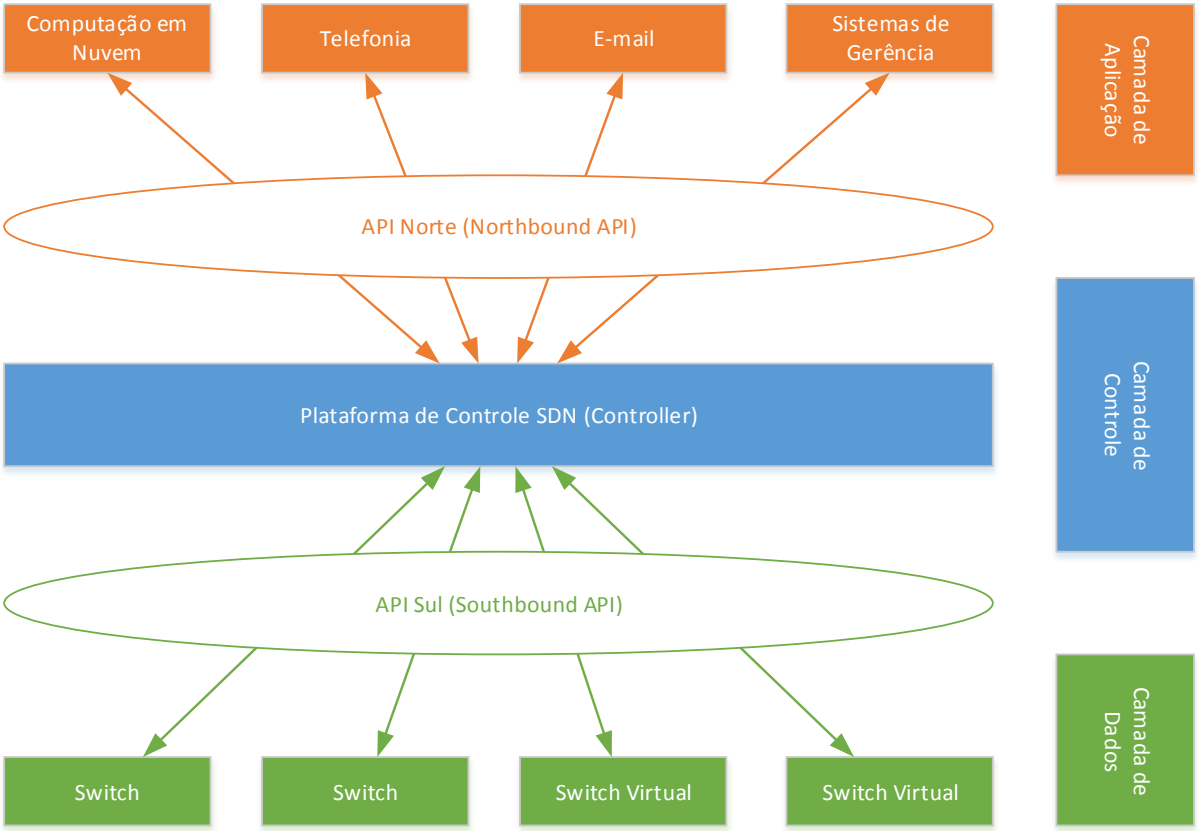

Figura 2 – Arquitetura SDN

Neste trabalho será adotada principalmente a nomenclatura proposta na Figura 2, com as seguintes interfaces entre as camadas: API Norte (*Northbound API*) e API Sul (*Southbound API*), de acordo com (KIRKPATRICK, 2013). A ONF recentemente passou a definir também as duas interfaces, porém definiu o nome *SDN Control to Data-Plane Interface* para a interface API Sul (ONF, 2013b).

A camada de aplicação é ilustrada no topo da arquitetura e representa aplicações e instruções de alto nível; a camada de controle fica no meio da arquitetura e age de acordo com as instruções recebidas das aplicações através da interface Norte, enviando instruções para a camada de dados através da interface Sul; a camada de dados apenas recebe ordens e as executa. As camadas de nível inferior também podem usar as interfaces para relatar eventos e outras informações para as camadas superiores (ONF, 2013b).

O Nome das interfaces está diretamente relacionado com a direção da comunicação, pois o elemento central de referência em uma arquitetura SDN é o controlador. Como em qualquer modelo de arquitetura de redes de computador ou sistema computacional, a interface Norte descreve a comunicação com elementos de

mais alto nível, ao passo que a interface Sul descreve a comunicação com elementos de um nível mais baixo (KIRKPATRICK, 2013).

INTERFACE API NORTE E CAMADA DE APLICAÇÃO

Esta interface permite que as aplicações programem e solicitem serviços das camadas inferiores. A camada de aplicação frequentemente contém funções de gerência, como monitorar alarmes; e funções básicas de rede, como cálculo de caminho mais curto, roteamento e segurança (KIRKPATRICK, 2013).

Não existe um padrão global aberto definido para esta interface, embora existam iniciativas para tornar isso possível, como o *OpenStack Neutron API*, anteriormente conhecido como *Quantum API* (SDNCENTRAL.COM, 2014). Esse API é responsável por fornecer a função de *Networking-as-a-Service* (NaaS), ou seja, a "rede-como-um-serviço". Serviços típicos são VPNs, reserva de banda sob demanda, dentre outros.

O *Neutron API* suporta plug-ins para integrar equipamentos SDN e controladores de diversos fabricantes, como Cisco, Juniper, Brocade, Extreme, Big Switch, entre outros (OPENSTACK.ORG, 2014a). Esta interface é parcialmente responsável por permitir serviços de computação em nuvem, como aqueles disponibilizados pelo PayPal, Mercadolibre e Cisco Webex (OPENSTACK.ORG, 2014b).

Além da iniciativa do *OpenStack Neutron*, existem também várias linguagens e sistemas que abstraem a tarefa de controle da rede através da aplicação de políticas de rede (*policy enforcement*), e podem ser consideradas interfaces do tipo API Norte:

- FML (*Flow-based Management Language*) construída utilizando o controlador NOX e proposta por Hinrichs et al (2009)
- Procera, uma linguagem proposta por Voellmy et al (2012).
- Frenetic, uma linguagem de programação de rede de alto nível e modular que atualmente é disponibilizada em OCaml e Python, neste caso chamada de Pyretic (FRENETIC-LANG.ORG, 2014).

Tanto a linguagem FML quanto a Procera são linguagens declarativas que realizam um alto nível de abstração da rede. A linguagem Procera, por exemplo,

permitiria a um administrador de rede especificar algo como “banir o dispositivo da rede assim que o seu consumo de tráfego exceder 10GB dentro do período dos últimos cinco dias”, de acordo com exemplo traduzido de Voellmy et al (2012).

De forma similar, a família de linguagens Frenetic oferece um conjunto de abstrações declarativas que permitem consultar o estado da rede, definir políticas de encaminhamento de tráfego e atualizá-las de forma consistente, conforme descrito por Foster et al (2013).

Com base nas linguagens Frenetic existe também uma plataforma de controle chamada de Resonance, ou PyResonance em sua versão baseada em Pyretic. Esta plataforma permite expressar estados da rede na forma de máquinas de estado, que podem tomar ações na rede com base em ocorrências de eventos. Essa plataforma é apresentada por Nayak et al (2009) com um modelo de rede para controle de acesso dinâmico em redes corporativas, no qual o estado dos PCs conectados à rede é rastreado. Cada endereço MAC que se conecta à rede se encaixa em estados como “Em registro”, “Autenticado”, “Operacional” ou “Em quarentena”.

PLATAFORMA DE CONTROLE SDN E INTERFACE API SUL

A plataforma de controle é considerada o núcleo de uma rede SDN. As duas funções básicas que uma plataforma de controle deve desempenhar em sua Interface API Sul são: gerência de configuração, controle de fluxos de tráfego e aplicação de políticas de rede.

Gerência de Configuração

A tarefa de provisionar os elementos de rede pode ser feita de diversas formas, como SNMP, CLI ou através de interfaces Web. No entanto, para manter o processo de configuração centralizado, a literatura de SDN menciona normalmente o uso do NETCONF (ENNS et al, 2012), um protocolo transacional que utiliza RPCs (*Remote Procedure Calls*) através de um canal de transporte seguro, como SSL e SSH, para gerenciar configurações em dispositivos remotos (DEVLIC, 2012), (NARISETTY, 2013).

A própria ONF (2013c) criou um padrão a ser utilizado em dispositivos baseados em OpenFlow chamado de OF-Config, baseado em NETCONF e YANG, este sendo apenas uma linguagem para modelar dados para NETCONF (BJORKLUND, 2010).

Controle de Fluxos de Tráfego

Embora o OpenFlow seja o padrão mais popularmente conhecido quando se fala de SDN, esse é apenas um dos padrões existentes para a Interface API Sul (FEAMSTER et al, 2013). Nesta seção serão expostas algumas informações e o histórico sobre alguns protocolos e padrões utilizados para compor arquiteturas SDN.

OpenFlow

O precursor do OpenFlow foi criado por Martin Casado, na época um estudante de PhD de Stanford. Casado et al (2007) propuseram o *Ethane*, uma arquitetura para redes corporativas. Sua arquitetura experimental foi testada através de equipamentos servindo mais de 300 hosts no Campus da universidade de Stanford. A primeira aparição do termo OpenFlow surgiu na proposta de McKeown (2008) um ano depois, e a partir desse trabalho é que surgiram as especificações do OpenFlow com o objetivo de garantir interoperabilidade no controle de *switches* na rede.

Atualmente em sua versão 1.3.4, ainda sob revisão, O OpenFlow permite ao controlador SDN acesso direto e manipulação da camada de dados de dispositivos de rede como switches e roteadores (ONF, 2014a) que também suportem OpenFlow. Os dispositivos controlados podem ser tanto físicos (e.g. switches como Cisco, HP, Dell), quanto virtuais (e.g. Open vSwitch).

Algumas das plataformas de controle OpenFlow open-source mais conhecidas são NOX/POX, Beacon, Maestro, Floodlight, SNAC e OpenDaylight. Existem também alternativas comerciais, como o Big Network Controller da Big Switch Networks. A análise de performance de controladores SDN OpenFlow é uma questão bastante discutida que vem sendo analisada em teses e artigos nos últimos anos, como exposto por Shah et al (2013) e Muntaner (2012). Nunes et al (2014) apresentam

uma lista bastante completa de controladores SDN disponíveis atualmente, principalmente para OpenFlow.

ForCES

Forwarding and Control Element Separation (ForCES) é uma arquitetura de rede criada pelo grupo de trabalho homônimo (IETF ForCES WG, 2014), e é definida por diversas RFCs e *Drafts*. É considerada uma arquitetura SDN pois também prevê a separação das camadas de controle e dados.

Apesar de conter RFCs datando desde 2003 com definição de um *framework* e uma especificação do protocolo ForCES, em 2010, o padrão não chegou ao mesmo nível de popularidade do OpenFlow. Considera-se que o ForCES possui diversas vantagens em relação ao OpenFlow pois possui vários aspectos que aumentam a extensibilidade do protocolo (HALEPLIDIS et al, 2012) (FEAMSTER, 2013).

Existem poucos relatos de testes do protocolo ForCES realizados com sucesso, e há apenas uma empresa que divulga uma implementação do protocolo como parte de seu sistema operacional de rede (*Network Operating System - NOS*), a Mojatatu Networks (HALEPLIDIS et al, 2012). No meio acadêmico já se considera a possibilidade de se integrar a popularidade do OpenFlow com a extensibilidade do ForCES (HALEPLIDIS et al, 2012).

PCE

Path Computation Element (PCE) é uma arquitetura de rede criada com o objetivo de realizar cálculo de caminhos para redes do tipo MPLS e GMPLS, criada por outro grupo de trabalho do IETF (IETF PCE WG, 2014). Da mesma forma que o proposto por outras arquiteturas SDN, o grupo de trabalho PCE afirma que o cálculo de *Label Switched Paths* (LSPs) não necessariamente é feito no próprio roteador – *Label Switching Router* (LSR).

Os primeiros documentos do grupo de trabalho PCE foram lançados em 2006, descrevendo a arquitetura de uma rede baseada em PCE (IETF PCE WG, 2014). Dentre outras padronizações, o grupo de trabalho PCE propõe o protocolo

PCEP, utilizado para comunicação entre Path Computation Clients (PCCs) e PCEs. Da mesma forma que o ForCES, a popularidade do PCE não chega a ser tão grande quanto a do OpenFlow, porém existem também estudos sobre a integração do PCE como uma aplicação implementada através de OpenFlow (MUNOZ, 2014).

BGP-LS

Outro grupo de trabalho do IETF chamado IDR, *Inter-Domain Routing*, propõe uma nova extensão ao BGP, *Border Gateway Protocol*, (HARES et al, 2006), criada com o objetivo de permitir a coleta de informações sobre o estado dos enlaces da rede (LS – *Link State*) e engenharia de tráfego (TE – *Traffic Engineering*). A extensão é comumente chamada de BGP-LS, e a divulgação de informações da rede é possível através de duas novas propostas de Gredler et al (2013):

- Um novo formato de codificação chamado de BGP NLRI (*Network Layer Reachability Information*), que descreve enlaces, nós e prefixos de rede contendo informações de estado de enlace IGP (*Interior Gateway Protocol*);
- O novo BGP *Path Attribute*, chamado de BGP-LS, responsável por transportar as informações descritas pelo NLRI;

O documento de Gredler et al (2013) ainda está em estado de draft no IETF, mas já expõe duas possibilidades de aplicação do BGP-LS:

- Interface do BGP-LS com um servidor PCE para que cálculos de caminhos otimizados sejam feitos externamente à rede pelo servidor PCE;
- Envio de informações do BGP-LS para um servidor ALTO (SEEDORF e BURGER, 2009), responsável por gerar uma abstração da topologia da rede e enviá-la para aplicações que utilizam essa informação para tomadas de decisão. Aplicações típicas são clientes ou rastreadores (*trackers*) P2P (*peer-to-peer*) ou redes do tipo CDN (*Content Delivery Network*).

OpFlex

A CISCO divulgou notícias em 2014 sobre um novo protocolo a ser utilizado na Interface API Sul, controlando switches e roteadores através de uma linguagem declarativa. O protocolo e arquitetura foram divulgados no website da CISCO como um *whitepaper* (CISCO, 2014) e publicado como um *draft* informativo no IETF (SMITH et al, 2014).

A principal característica de uma linguagem declarativa é de apenas se preocupar em determinar o que deve ser feito, e não como deve ser feito, ao contrário do OpenFlow, por exemplo. A linguagem declarativa é baseada no conceito de “teoria da promessa”, em que uma plataforma de controle pede ao elemento de rede que “prometa” que vá realizar uma determinada tarefa ou chegar a um determinado estado (CISCO, 2014).

A utilização desse conceito para controle de elementos de rede tem o objetivo de tornar o protocolo de controle mais flexível e simples, porém tem o efeito de tornar os elementos de rede controlados mais complexos, pois precisam ter a capacidade de desempenhar tarefas de maneira mais autônoma ao invés de simplesmente serem instruídos exatamente sobre como desempenhá-las. O próprio *whitepaper* (CISCO, 2014) alega que

> Systems built on declarative control can achieve high performance at scale with strong resiliency by moving complexity to edge devices, which do most of the processing.

Desse trecho entende-se que a complexidade de tomada de decisão sobre encaminhamento de fluxos é movida novamente para dispositivos de borda da rede, ou seja, se torna descentralizada. De acordo com o restante da literatura atual de SDN, esse conceito aparentemente vai de encontro a uma das principais características de centralizar o controle da rede e remover parte da complexidade dos elementos de borda (LIMONCELLI, 2013), (KIRKPATRICK, 2013), (CASADO, 2007), (NUNES et al, 2014).

Embora exista um draft público informativo descrevendo o conceito do OpFlex, a documentação ainda é escassa e não há notícias de implementação aberta ao público.

OVSDB

O OVSDB (*Open vSwitch Database Management Protocol*) é um protocolo criado para programar extensões, controlar e gerenciar *vSwitches*, *switches* virtuais, normalmente utilizados para encaminhar tráfego entre VMs (*Virtual Machines*, máquinas virtuais) e também entre VMs e a rede externa. Os *vSwitches* são comumente empregados em ambientes de virtualização de servidores (PFAFF, 2013).

O *Open vSwitch* é conhecido como o principal projeto de switch virtual e oferece implementações abertas de clientes e servidores OVSDB, além de oferecer suporte a controle através do protocolo OpenFlow (PFAFF, 2013).

VIRTUALIZAÇÃO DE FUNÇÕES DE REDE (NFV)

A virtualização de funções de rede (*Network Functions Virtualization – NFV*) é apenas um conceito de migrar funções de rede executadas normalmente por hardware especializado para um hardware genérico, como os servidores utilizados em Datacenters.

Embora seja independente do conceito de SDN, os conceitos são frequentemente confundidos. O próprio artigo da ETSI sobre NFV indica que existe a possibilidade de interação entre NFV e SDN, sendo ambos complementares, conforme indicado pela Figura 3 (ETSI NFV, 2012).

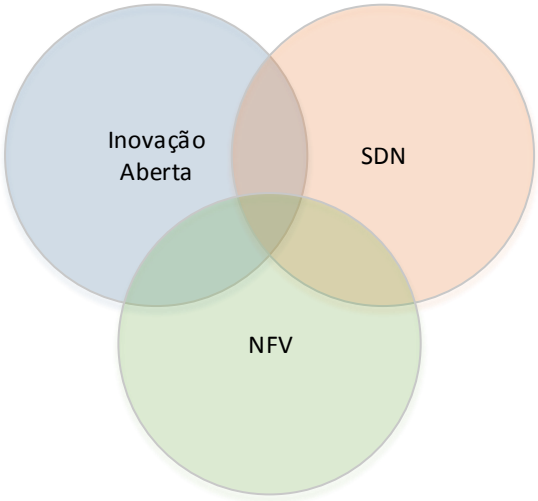

Figura 3 – Relação entre NFV e SDN

Enquanto SDN trata de uma arquitetura de rede em que o plano de controle é separado do plano de tráfego de dados, NFV é a implementação de funções

de rede em software, que pode ser executado em hardware de servidores de padrão comercial. Essas funções de rede, por serem virtualizadas, podem ser instanciadas em várias localidades de uma rede sem a necessidade de se instalar novos equipamentos (ETSI NFV, 2012).

Alguns exemplos de funções de rede que podem ser virtualizadas são:

- Inspeção da camada de aplicação de pacotes (*deep packet inspection – DPI*);
- Funções de segurança, como *Firewalls*, Sistemas de Detecção de Intrusão (*Intrusion Detection Systems - IDS*);
- Roteadores

PROJETO OPENDAYLIGHT

O projeto OpenDaylight é uma tentativa de integrar uma solução SDN completa modularizada e de código aberto. O projeto realiza o desenvolvimento de um "*framework*" que engloba as três camadas principais de uma rede SDN: a Interface API Norte; a plataforma de controle; e a Interface API Sul. Além disso enfatiza a criação de uma camada adicional entre a plataforma de controle e a Interface API Sul: a camada de abstração de serviços (SAL – Service Abstraction Layer), que seria responsável por desvincular a Interface API Norte da Interface API Sul, independente do protocolo utilizado (e.g. OpenFlow, NETCONF, OVSDB, PCE e BGP-LS).

O objetivo do projeto é possibilitar que diferentes protocolos de controle possam ser integrados como *plug-ins*, conforme explicado por Wright et al (2013). Da mesma forma, o projeto prevê a integração de diferentes Interfaces API Norte, como o OpenStack Neutron e Interfaces gráficas de gerência (OPENDAYLIGHT.ORG, 2014a).

O projeto OpenDaylight tem o apoio de diversas empresas do setor de telecomunicações, soluções de virtualização e fabricantes de hardware de servidores e switches, como Cisco, Microsoft, Brocade, Juniper, VMWare, Red Hat, IBM, Intel e vários outros (OPENDAYLIGHT.ORG, 2014b).

CONCLUSÃO

Neste capítulo foram expostos o histórico e a essência daquilo que a literatura atual define como uma arquitetura SDN. Também foram expostos os principais protocolos e projetos que podem ser utilizados para compor uma arquitetura SDN.

Observa-se que grande parte das tecnologias orientadas a SDN estão em desenvolvimento, e a quantidade e variedade de artigos sobre o tema é imensa.

CAPÍTULO 2 – FUNCIONAMENTO DE REDES CLÁSSICAS

Para um melhor entendimento das arquiteturas SDN e quais são as possíveis mudanças trazidas pelas novas tecnologias, é essencial que se tenha um bom entendimento teórico do funcionamento das redes de computadores de hoje.

As redes de computadores atuais podem ser classificadas de diversas formas, e comumente são divididas de forma hierárquica entre redes locais, ou LAN – *Local Area Networks*, redes metropolitanas, ou MAN – *Metropolitan Area Networks* e redes de longa distância, chamadas de WAN – *Wide Area Networks*, cada uma com tamanho maior. A interconexão de uma ou mais redes é chamada de *Internetwork*, cujo principal exemplo é a Internet (TANENBAUM e WETHERALL, 2010, p18).

O principal foco deste trabalho será de estudar casos de uso de redes locais, principalmente casos de uso funcionalidades comumente encontradas em redes corporativas (*enterprise networks*).

REDES LOCAIS (LANS)

É de comum conhecimento que a maior parte das redes locais são baseadas em *switches Ethernet*, os quais são ligados a computadores e servidores através de cabos e portas com padrão *Ethernet*, através de cabos óticos ou de cobre. Os *Switches* são responsáveis por realizar então o encaminhamento de quadros (*frames*) de dados entre os diversos dispositivos conectados à ele, utilizando o campo de endereço de destino do quadro para encaminhá-lo corretamente (TANENBAUM e WETHERALL, 2010, p20).

Os diversos switches podem ser interligados para expandir a rede local e acrescentar mais computadores e servidores conforme representado no cenário de três switches interligados “em cascata”, Figura 4.

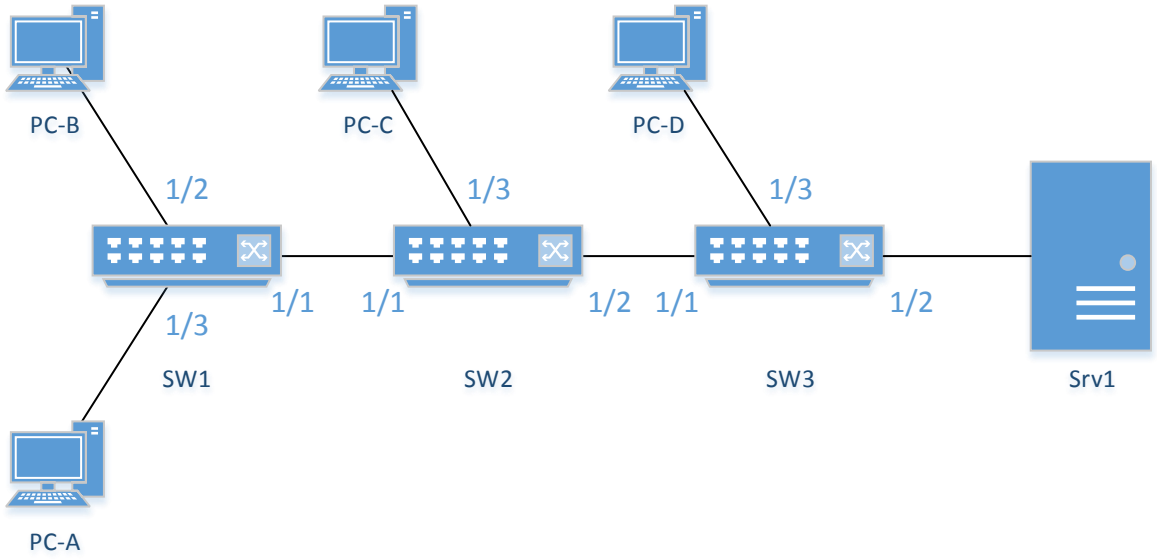

Figura 4 – Rede local “em cascata” composta por três *switches*

Funcionamento da Comutação de Pacotes em Camada de Enlace Ethernet

Historicamente as redes Ethernet utilizavam Hubs para interligação dos dispositivos de rede. Hubs encaminham quadros Ethernet recebidos em uma porta para todas as outras portas, sem verificar qual é o destino do pacote. Atuam apenas como repetidores. Essa característica os torna bastante limitados, pois toda a banda da rede é dividida entre os elementos conectados nos Hubs (TANEMBAUM e WETHERALL, 2010, p 333).

Felizmente os switches já se tornaram comuns e presentes até mesmo nas redes mais simples, e tornam o processo de comunicação em redes Ethernet muito mais eficiente. Para realizar a comutação de pacotes entre os dispositivos, os switches utilizam um algoritmo básico de aprendizado de endereços MAC (*Backward Learning*) e tabelas de endereços MAC, que são muito bem descritos por Tanenbaum e Wetherall (2010, p. 335-336). O princípio de funcionamento de um switch pode ser consolidado de maneira simplificada através das regras abaixo:

1. Ao receber um quadro numa interface, o switch grava o endereço de origem desse frame em sua tabela de endereços MAC, associando este com a porta pela qual recebeu o frame (*Backward Learning*);
2. Caso o endereço MAC de destino do quadro recebido esteja na mesma porta pela qual foi recebido, o quadro é descartado;

3. Caso o endereço MAC de destino do quadro recebido seja conhecido e esteja em porta diferente da qual foi recebido, o quadro é encaminhado apenas para esta porta;
4. Caso o endereço MAC de destino do quadro recebido seja desconhecido (DLF – *Destination Lookup Failure*), *Multicast* ou *Broadcast*, o quadro é encaminhado para todas as portas do switch, exceto pela porta de origem. Este passo é conhecido como "*flooding*";
5. Caso o endereço MAC de destino do quadro recebido seja um endereço especial (link-local), o quadro deve ser processado pelo switch ou descartado, mas não deve ser encaminhado para outras portas.

Dessa forma os switches otimizam o envio de quadros apenas para as portas que as devem receber. Para fins ilustrativos, a Figura 5 mostra como ficariam as tabelas de endereços MAC dos switches da Figura 4 caso todos os PCs da rede se comuniquem com o servidor "Srv1".

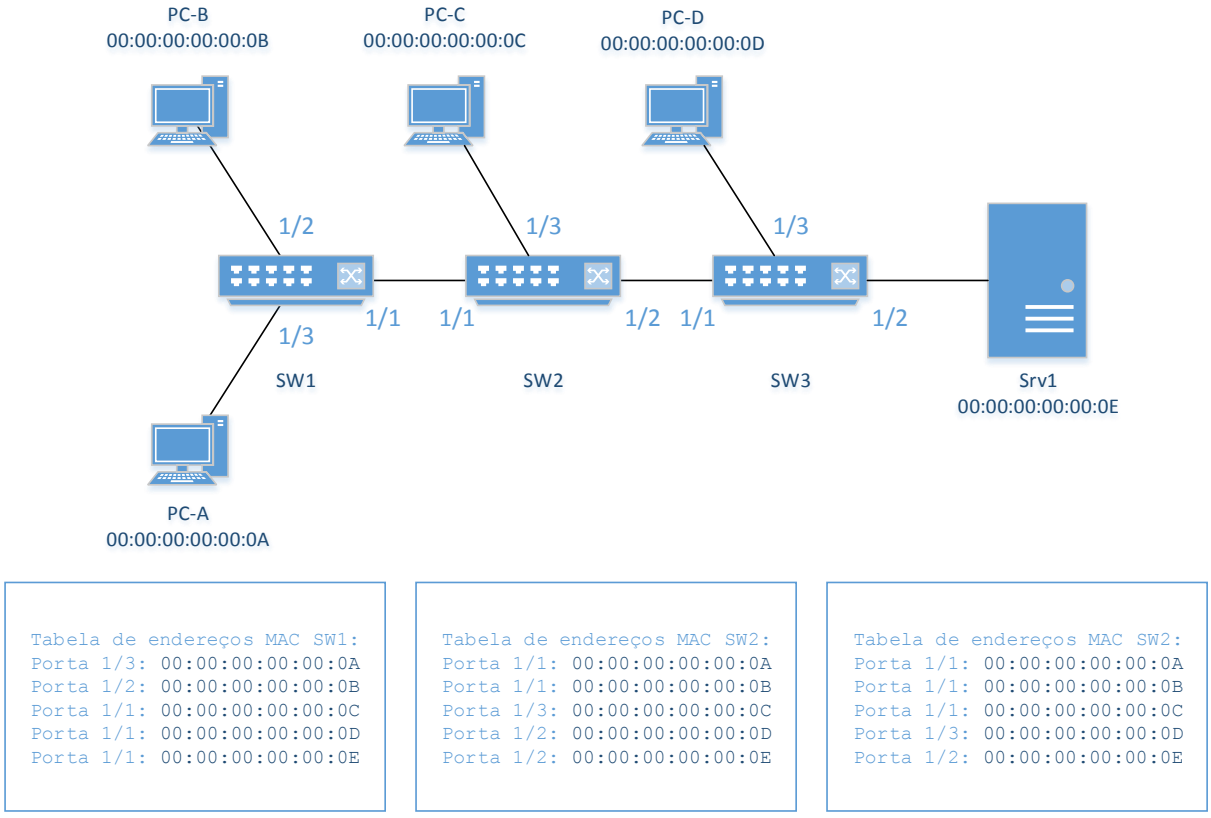

Figura 5 – Tabelas de endereços MAC preenchidas em rede Ethernet

Redes Ethernet com Enlaces Redundantes e *Spanning-tree*

Como o objetivo geral de quaisquer redes de telecomunicações é prover conectividade entre os usuários, é altamente desejável que exista pelo menos um enlace redundante entre os dispositivos da rede que mais requerem comunicação. No caso das redes Ethernet esse conceito também se aplica, porém a natureza de descoberta dos destinos através de "*flooding*" pode trazer alguns problemas.

Caso as redes Ethernet sejam utilizadas com enlaces redundantes, como representado na Figura 6 pela adição de um enlace entre os switches SW1 e SW3, qualquer quadro com destino desconhecido, *broadcast* ou *multicast* causaria um "loop" interminável na rede. A performance da rede ficaria seriamente comprometida (TANENBAUM e WETHERALL, 2010, p 337).

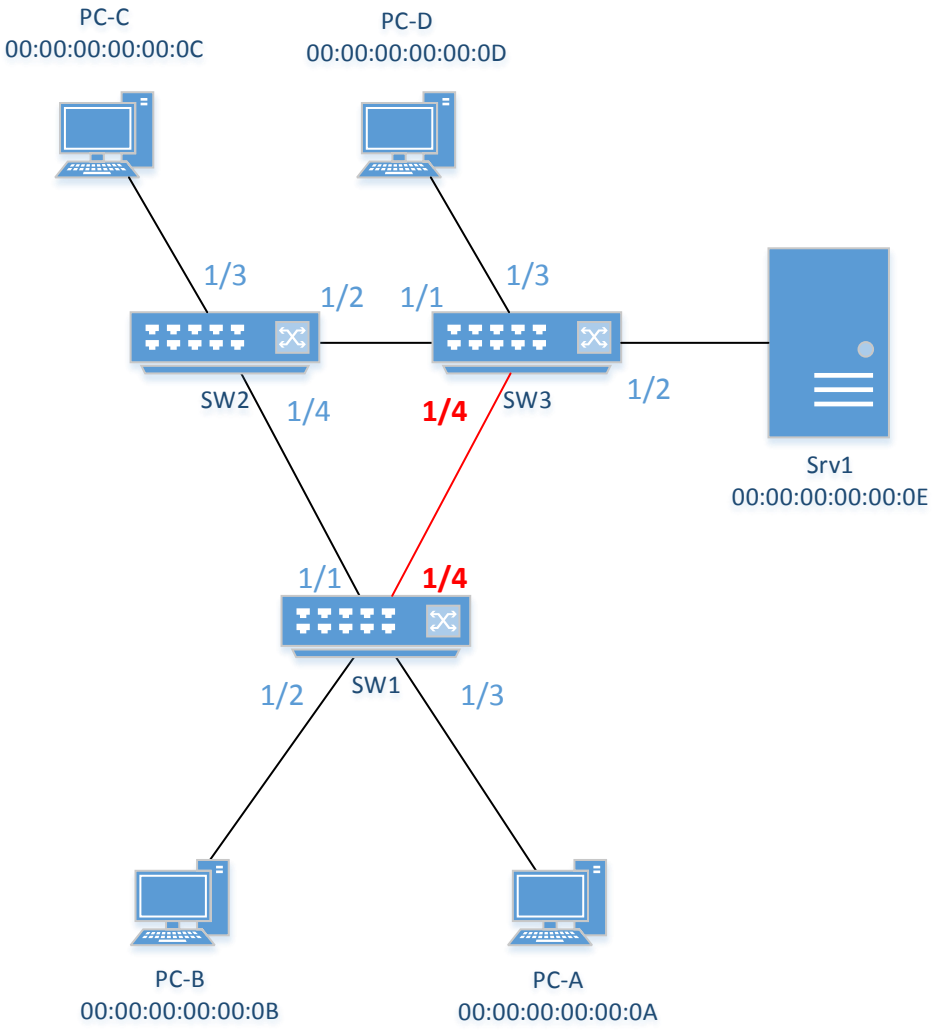

Figura 6 – Switches Ethernet conectados em topologia anel

A solução mais simples e comumente utilizada para evitar *loops* na rede utiliza um algoritmo distribuído conhecido como Spanning-tree, responsável por “descobrir” caminhos redundantes na rede e bloqueá-los caso não estejam em uso, porém garantido que toda a rede ainda tenha conectividade (TANENBAUM e WETHERALL, 2010, p 338).

Em uma rede com caminhos redundantes cada switch precisa de recursos de processamento para rodar esse algoritmo, enviando e recebendo periodicamente mensagens do protocolo spanning-tree (STP - *Spanning Tree Protocol*). Após o término de uma negociação entre todos os switches participantes do Spanning-tree, uma topologia sem *loops* é construída (TANENBAUM e WETHERALL, 2010, p 339).

Na Figura 6, por exemplo, dependendo de configurações dos switches da rede, qualquer um dos enlaces entres os switches SW1, SW2 e SW3 poderia ser bloqueado para evitar os loops. Considerando o caso do enlace entre SW1 e SW3 ser bloqueado, os outros dois enlaces seriam suficientes para encaminhar todo o tráfego necessário da rede. Nessa situação, caso qualquer outro enlace seja rompido, os switches descobririam a mudança na rede e reativariam o enlace SW1-SW3 para manter toda a rede alcançável.

Segmentação em Redes Virtuais (VLANs)

Redes corporativas de grandes empresas podem ter centenas ou até milhares de usuários, e isso torna praticamente obrigatória a segregação da rede toda em diversas LANs, o que é feito através de separações lógicas de LANs, realizadas com VLANs (virtual LANs). Segmentar a rede pode trazer diversos benefícios, como facilitar o controle de acesso a recursos e também evitar degradação de performance (TANENBAUM e WETHERALL, 2010, p344).

A segmentação em redes virtuais pode ser realizada por diversos protocolos, sendo o IEEE 802.1Q o mais amplamente utilizado, que permite a subdivisão de uma LAN em até 4094 VLANs através da inserção de um *tag* de 4 bytes no quadro Ethernet, sendo 12 bits utilizados para o VID – VLAN ID, responsável pela identificação da VLAN (TANENBAUM e WETHERALL, 2010, p348).

A Figura 7 demonstra uma pequena rede em que a LAN é separada em duas VLANs, chamadas aqui de VLAN Verde e VLAN Laranja, que podem corresponder, por exemplo, aos VIDs 10 e 20. Equipamentos coloridos em verde estão conectados à VLAN Verde e podem se comunicar apenas com outros equipamentos que estejam na mesma VLAN. Da mesma forma os equipamentos coloridos em laranja podem se comunicar apenas com outros equipamentos conectados à VLAN laranja. Por exemplo, o PC-B não consegue se comunicar com o Servidor 2 (Srv2) ou com o PC-A, nem o PC-A com o Servidor 1 (Srv1).

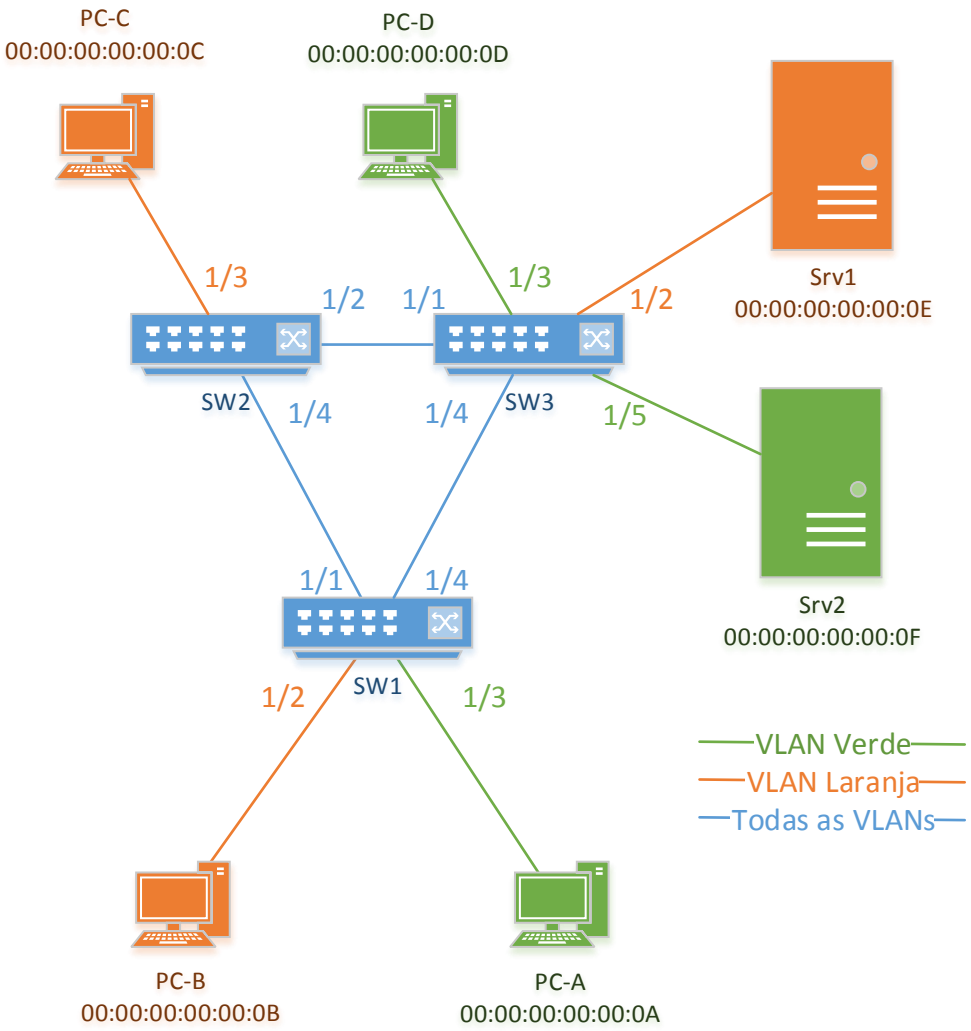

Figura 7 – Topologia em Anel com separação por VLANs

Roteamento entre Redes

Em tradicionais redes IP, cada LAN ou VLAN recebe uma faixa distinta de endereços de rede IP para que seja possível distinguir e encontrá-la na rede. Para interligar as diferentes e numerosas redes faz-se necessário então o uso de roteadores.

O conceito de roteamento é bastante amplo, porém em redes IP usualmente é empregado quando se trata de tomar decisões de encaminhamento na camada 3 entre redes (*internet*) em topologias divididas hierarquicamente em diversas redes e subredes. Tanembaum e Wetherall (2010, p 425) diferenciam a Internet mundial de outros tipos de internets através do uso da letra maiúscula “I” em Internet e neste trabalho será adotada a mesma terminologia.

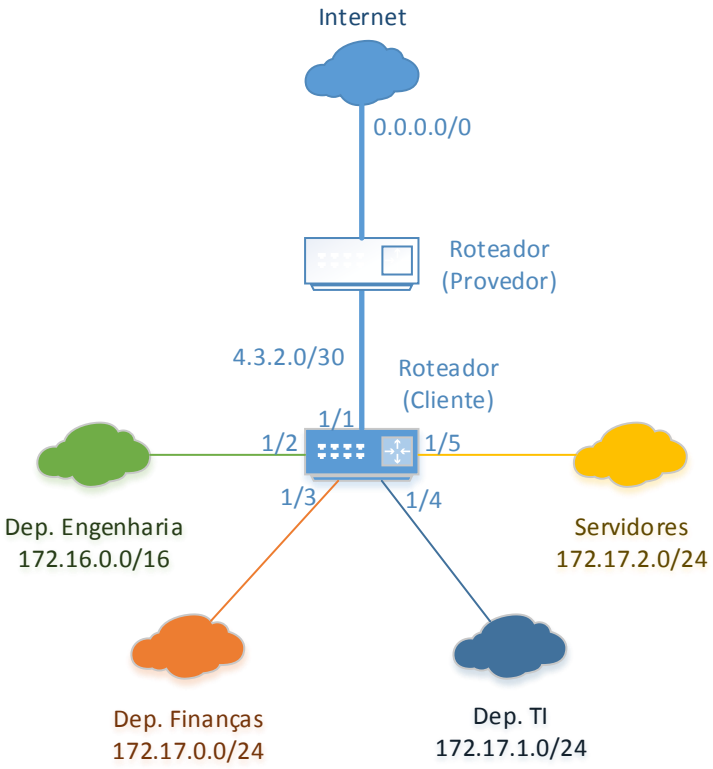

Figura 8 – Topologia simplificada de uma rede privada

A Figura 8 demonstra uma rede privada corporativa simplificada em que as redes são divididas de acordo com o departamento. Cada departamento recebe uma faixa de endereços IP privados, e o roteador (cliente) é responsável por possibilitar a comunicação entre redes. Nessa mesma Figura 8 também é demonstrada uma rede ponto-a-ponto com o roteador de um provedor de Internet, responsável por prover conectividade com esta.

Ainda no caso da Figura 8 o roteador realiza quase que exclusivamente roteamento entre redes diretamente conectadas ao roteador, não sendo necessário o uso de um protocolo de roteamento. Uma rede um pouco mais complexa poderia ser criada para fornecer conectividade a uma empresa que consista de diversas filiais localizadas em pontos geográficos distintos, como ilustrado na Figura 9. A topologia é um anel que interliga as cidades de Brasília, Rio de Janeiro, São Paulo, Curitiba e Porto Alegre. Cada um dos roteadores desta rede precisa ter em sua tabela de rotas os prefixos das redes com as quais se deseja ter comunicação.

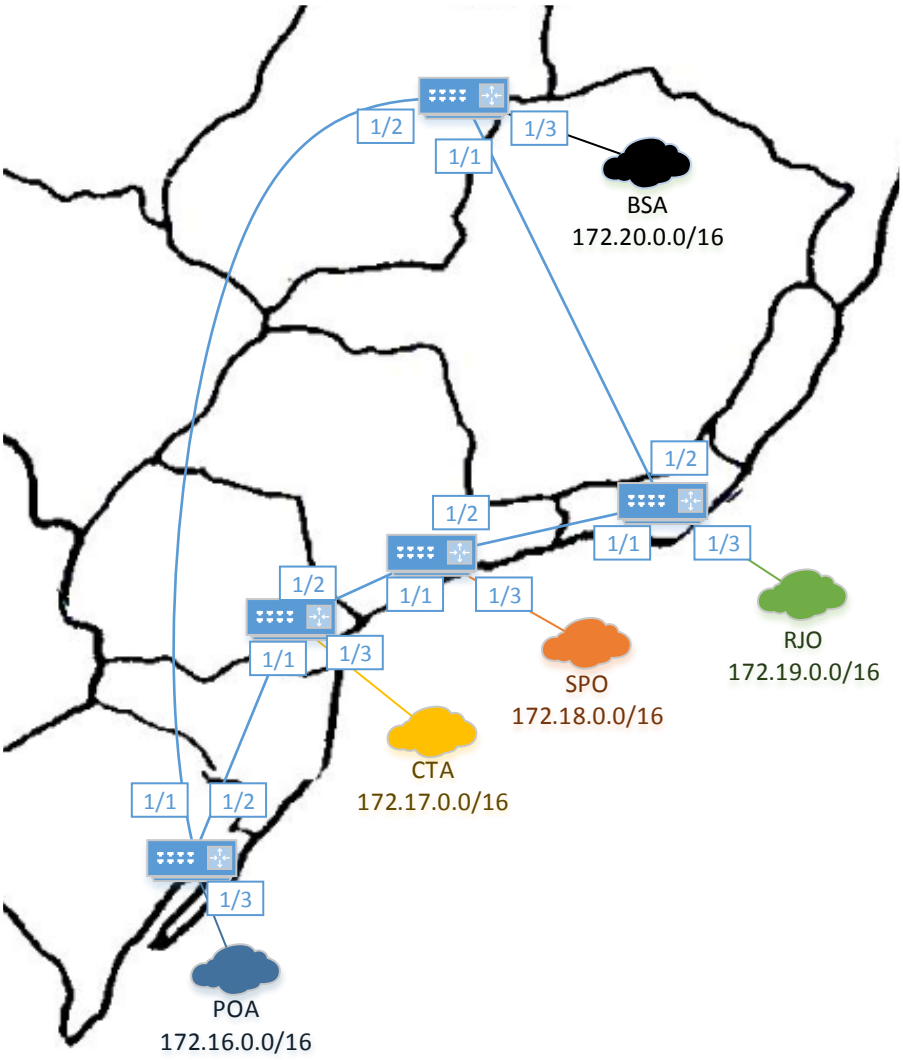

Figura 9 – Topologia de rede privada corporativa com diversas filiais

A tabela de rotas do roteador de SPO pode ser representada pela Tabela 1, evidenciando por qual interface o roteador deve enviar pacotes para que alcancem o seu destino de acordo o prefixo de rede atribuído, seguindo sempre a regra do maior prefixo que engloba o destino desejado ou *longest matching prefix* (TANENBAUM e WETHERALL, 2010, p 448).

Tabela 1 – Tabela de rotas do roteador SPO

Destino	Prefixo	Próximo	Interface de saída
POA	172.16.0.0/16	CTA	1/1
CTA	172.17.0.0/16	CTA	1/1
SPO	172.18.0.0/16	Conectada	1/3
RJO	172.19.0.0/16	RJO	1/2
BSA	172.20.0.0/16	RJO	1/2

A construção dessa tabela de roteamento pode ser feita de maneira estática, por configuração manual do administrador da rede, ou dinamicamente através do uso de protocolos de roteamento, como o OSPF (*Open Shortest Path First*), amplamente utilizado em redes de empresas (TANENBAUM e WETHERALL, 2010, p 474).

O protocolo OSPF é descrito em diversas RFCs criadas por um grupo de trabalho do IETF (IETF OSPF WG, 2014). Existem diversas especificações e extensões do protocolo descritas nas RFCs, como o OSPFv2 para IPv4 e OSPFv3 para IPv6. Um descritivo completo de todos os aspectos desse protocolo precisaria de centenas de páginas. Tanenbaum e Wetherall (2010, p 474-478) descrevem os principais aspectos do OSPF com bastante concisão, e a partir do trabalho deles pode-se resumir ainda mais aquilo que cada roteador da rede precisa fazer:

1. Cada roteador roda troca mensagens (*OSPF Hello*) com seus roteadores vizinhos para estabelecimento de adjacências;
2. Através de *flooding* de mensagens contendo informações sobre seus enlaces, cada roteador deve trocar informações com suas adjacências sobre o estado dos enlaces (*link states*) que possui e quais redes estão conectadas nesses enlaces;
3. Cada roteador constrói e mantem em memória um banco de dados de estado dos enlaces de toda a rede de sua área e computa os menores caminhos para chegar em todos os outros roteadores da área.

No exemplo dado na Figura 9, a rede tem apenas 5 roteadores, sendo que as redes de cada localidade certamente teriam mais roteadores para subdividir e hierarquizar ainda mais as faixas de rede de cada filial.

Quanto maior o número de roteadores na rede, maior a quantidade de memória que os roteadores necessitam para aprender as rotas da rede. Por esse

motivo os protocolos de roteamento dispõem de mecanismos para subdividir a rede em diversas hierarquias e permitir reduzir as tabelas de roteamento, sendo que o OSPF também dispõe dessa capacidade (TANENBAUM e WETHERALL, 2010, p 475).

SERVIÇOS DE REDES CORPORATIVAS

Nas seções anteriores foram expostos os conceitos básicos do funcionamento de uma rede corporativa do ponto de vista de conectividade entre diversas redes Ethernet. Nesta seção procura-se expor alguns dos serviços de rede utilizados por uma rede corporativa.

Configuração Dinâmica de Endereços da Rede em Hosts

Redes locais, corporativas ou caseiras, comumente utilizam equipamentos com suporte a DHCP (*Dynamic Host Configuration Protocol*) para configuração automática dos endereços IP dos dispositivos conectados à rede, os usuários finais. Além de alocação automática de endereços IP, o DHCP pode também configurar automaticamente dezenas de parâmetros automaticamente nos dispositivos que se conectam na rede, como máscara de rede, *default gateway*, serviço de DNS, NTP, etc. (TANENBAUM e WETHERALL, 2010, p 470).

Controle de Acesso à Rede

Uma das principais preocupações em grandes empresas é a confidencialidade das informações que podem ser acessadas através de sua rede. Diversas técnicas são empregadas em redes corporativas para evitar acesso não autorizado.

Uma das técnicas mais simples disponíveis em switches gerenciáveis atuais é apresentada como “port-security” ou “MAC limiting” por alguns fabricantes. Essa funcionalidade trata de limitar o número máximo de endereços MAC que uma porta de um switch pode aprender (CIAMPA, 2011, p268). É útil para bloquear ataques

de MAC *Flooding*, realizados com o objetivo de exaurir os recursos de memória de um Switch.

Outra técnica de segurança é a configuração de listas de controle de acesso (ACL – *Access Control Lists*) para bloqueio e permissão de tráfego. Essas listas podem conter uma série de regras sobre as permissões referentes a algum objeto. Quando um usuário tenta realizar uma operação sobre um objeto, por exemplo abrir uma sessão telnet da origem “A” para destino “B”, um roteador da rede pode verificar se “A” está autorizado a abrir uma sessão TCP pela porta 23 para o destino “B” e permitir o acesso. Listas de acesso tipicamente permitem configuração de regras de permissão ou negação de tráfego de pacotes com base em combinações de IP origem/destino, porta de origem/destino, protocolo utilizado, *tag* de VLAN, dentre outras possibilidades (CIAMPA, 2011, p344).

Outra alternativa é autenticação de usuários por porta. Switches gerenciáveis comumente permitem configuração de função de autenticação nas portas, de modo que portas não autorizadas só podem trafegar dados após a autenticação do usuário contra um banco de dados em servidor de autenticação, como RADIUS ou TACACS (CIAMPA, 2011, p268).

Network Address Translation – NAT

Devido às limitações de quantidade de endereços IPv4 na Internet, é praticamente onipresente o uso de NAT em redes corporativas. A ideia principal é isolar o endereçamento entre uma rede externa (pública) e uma rede interna (privada). Quando um pacote sai da rede privada para a rede pública, o elemento de rede responsável por realizar o NAT modifica o cabeçalho IP do pacote, alterando o endereço de origem interno para o endereço de origem externo, válido na Internet. O NAT realiza o mapeamento das conexões através de combinações de endereços IP interno e externo, portas de origem e destino (UDP ou TCP) do pacote, permitindo que comunicação bidirecional seja estabelecida entre dispositivos privados e públicos, “escondendo” os endereços privados na comunicação (TANENBAUM e WETHERALL, 2010, p 451-454).

Essa técnica permite que uma grande quantidade de dispositivos da rede privada utilize apenas um endereço da rede pública, reduzindo drasticamente a

quantidade de endereços IP públicos necessários. Cerca de 60 mil máquinas de uma rede privada poderiam utilizar um único IP público, conforme exposto por Tanenbaum e Wetherall (2010, p454-455).

Firewall

Firewalls funcionam como filtros de pacotes, inspecionando todos os pacotes que entram e saem de uma rede. Os critérios que um firewall utiliza são similares aos critérios de ACLs, determinando permissões de origens/destinos aceitáveis, origens/destinos que devem ser bloqueados e regras default determinando o que fazer com pacotes que não se encaixam em nenhum dos critérios (TANENBAUM e WETHERALL, 2010, p 819).

A grande diferença entre firewalls e listas de acesso configuradas para bloquear tráfego está no fato de que firewalls realizam uma inspeção mais criteriosa dos pacotes, considerando o estado das conexões, inspecionando diversos campos do cabeçalho do pacote. Por exemplo, o firewall pode determinar que um servidor Web externo pode enviar tráfego para um cliente da rede interna, porém apenas se o cliente interno iniciou essa conexão (TANENBAUM e WETHERALL, 2010, p819).

CONCLUSÃO

Este capítulo expôs parte dos principais conceitos utilizados em redes Ethernet corporativas. A partir desta introdução do funcionamento de redes pode-se discutir como seriam implementadas algumas dessas funcionalidades em uma arquitetura SDN.

CAPÍTULO 3 – FUNCIONAMENTO DE REDES SDN BASEADAS EM OPENFLOW

Neste capítulo o objetivo é demonstrar o funcionamento de uma rede SDN utilizando um emulador destinado a esse fim. Como visto no capítulo 1, o protocolo da Interface API Sul mais popularmente utilizado é o OpenFlow, e existem muitas referências bibliográficas apontando para o uso desse protocolo para experimentos de novas redes e protocolos de comunicação. É importante ressaltar que o OpenFlow foi rapidamente adotado por diversas instituições de pesquisa, como o CPqD (CPQD, 2014), e também por empresas de grande porte, como o Google (MERRITT, 2012). Além disso, essa popularidade impulsionou o rápido desenvolvimento de novos com suporte a OpenFlow pelos fabricantes de switches (NUNES et al, 2014). A DATACOM, um fabricante nacional de equipamentos de telecomunicações, também oferece suporte ao protocolo OpenFlow em alguns de seus switches (TELESINTESE.COM.BR, 2013).

O PROTOCOLO OPENFLOW

No primeiro capítulo o OpenFlow foi apresentado de um ponto de vista histórico. Nesta seção serão expostos os principais aspectos de como funciona o protocolo OpenFlow.

A Especificação do Protocolo OpenFlow

A especificação OpenFlow descreve os requisitos que um switch deve cumprir para suportar esse protocolo. A especificação determina que um switch é composto logicamente de uma ou mais tabelas de fluxos (*flow tables*) e uma tabela de grupo (*group table*), que desempenham as funções de pesquisa por regras e encaminhamento de pacotes (ONF, 2014a, pág. 8). A tabela de grupo é utilizada para que se possa realizar diversas ações num pacote. Por exemplo, é possível determinar um pacote deve ser encaminhado para um determinado grupo de portas, ou a primeira porta operacional de um grupo de portas (ONF, 2014a, pág. 20).

O switch também deve ser capaz de estabelecer um canal de comunicação com o controlador através do protocolo OpenFlow, que pode ou não ser criptografado e deve ser estabelecido com uma conexão TCP/IP (ONF, 2014a, pág. 32).

Utilizando o canal de comunicação OpenFlow, o controlador da rede pode adicionar, atualizar e deletar fluxos das tabelas do switch. Cada fluxo numa tabela é uma regra composta por campos de *correspondência* (*match*), *contadores* (*counters*) e um conjunto de *instruções* (*instructions*) a serem aplicados aos pacotes que correspondem ao fluxo instalado. Regras podem ser manipuladas nas tabelas tanto de maneira reativa quanto de maneira proativa (ONF, 2014a, pág. 8).

A manipulação de fluxos nas tabelas é denominada reativa quando o switch reage a um novo evento e pergunta ao controlador como proceder, por exemplo quando recebe um pacote que não corresponde a nenhuma regra em suas tabelas de fluxos. Essa operação é proativa quando o controlador envia comandos para configuração de regras na tabela de fluxos sem uma solicitação explícita do switch.

Funcionamento Básico de Switch Controlado por OpenFlow

Para ilustrar o funcionamento básico de um switch OpenFlow, toma-se como exemplo a sequência de eventos que ocorre até que um fluxo de tráfego chegue ao destino, sendo enviado da origem A até o destino B, conforme descrito a seguir e ilustrado na Figura 10 (SEZER et al, 2013):

1. Quando um switch OpenFlow é ligado, ele deve ser configurado para se conectar a um controlador;
2. O primeiro pacote de um fluxo de tráfego é enviado da origem A e chega no switch OpenFlow (Passo 1);
3. O *switch* consulta a sua tabela de fluxos (Passo 2). Caso exista uma entrada de um fluxo correspondente aos dados do pacote, como destino IP, a ação dessa entrada é executada e o pacote é encaminhado adiante caso permitido (Passo 5);
4. Caso não exista uma entrada correspondente ao fluxo de tráfego, o *switch* pode enviar uma amostra do pacote para o controlador utilizando o OpenFlow (Passo 3);

5. O controlador decide o que deve ser feito com o pacote e solicita ao *switch* que instale uma regra na sua tabela de fluxos (Passo 4). Por exemplo, a regra pode determinar que o pacote deve ser descartado, encaminhado adiante por determinada porta, ou até determinar que um contador deve ser incrementado;
6. O *switch* OpenFlow executa a regra recebida e a instala na tabela de fluxos. Neste exemplo o pacote é encaminhado adiante para o destino B. Da próxima vez que pacotes com as mesmas características forem recebidos, a mesma regra será executada.

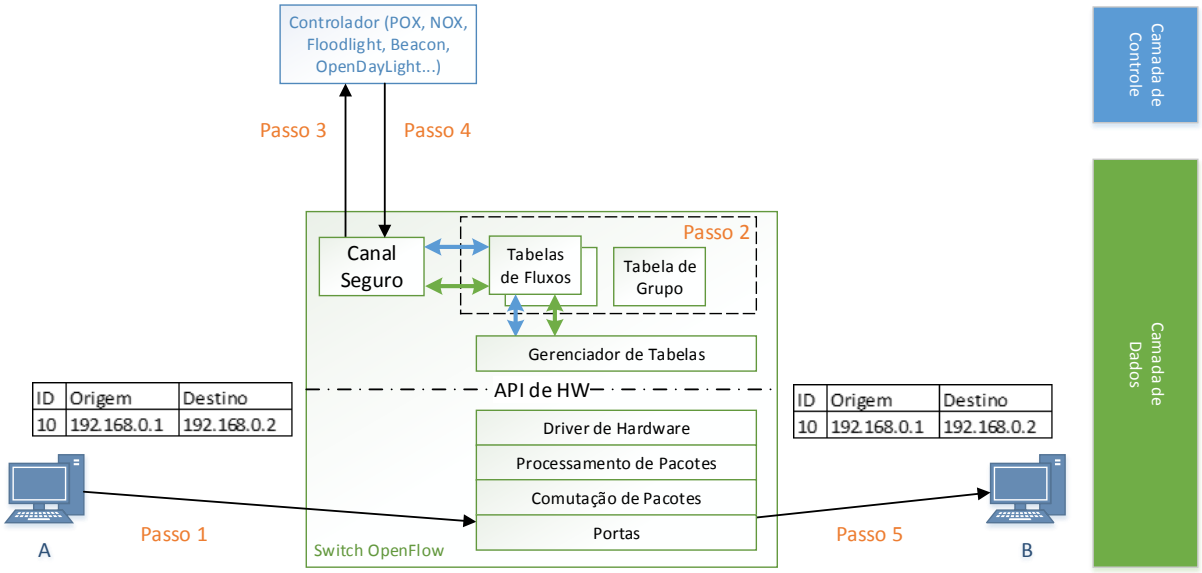

Figura 10 – Funcionamento de rede SDN com OpenFlow

A Figura 11 ilustra a troca de mensagens entre os elementos da Figura 10 no caso do PC A tentar iniciar um ping para B (*ICMP Echo Request*). A Figura 11 foi criada com base na comunicação observada entre dois PCs rodando a aplicação forwarding.l2_pairs do POX (NOXREPO, 2013), utilizando um laboratório virtual com o Mininet (MININET.ORG, 2014).

É demonstrada também na Figura 11 a conexão inicial do switch com o controlador através de OpenFlow através de uma conexão TCP na porta 6633, a porta padrão utilizada para o OpenFlow. Em seguida o switch envia a mensagem *OpenFlow Hello*, indicando a versão do OpenFlow que deseja utilizar; Essa mensagem é respondida pelo controlador, que solicita mais informações sobre as funcionalidades suportadas pelo switch através de um *OpenFlow Feature Request*; O switch, responde com um OpenFlow Feature Reply contendo informações como seu identificador DPID (*Datapath Identifier*), quantidade máxima de fluxos suportados em suas tabelas,

portas disponíveis e outras características. A partir deste ponto o switch está pronto para operar de acordo com os comandos do controlador.

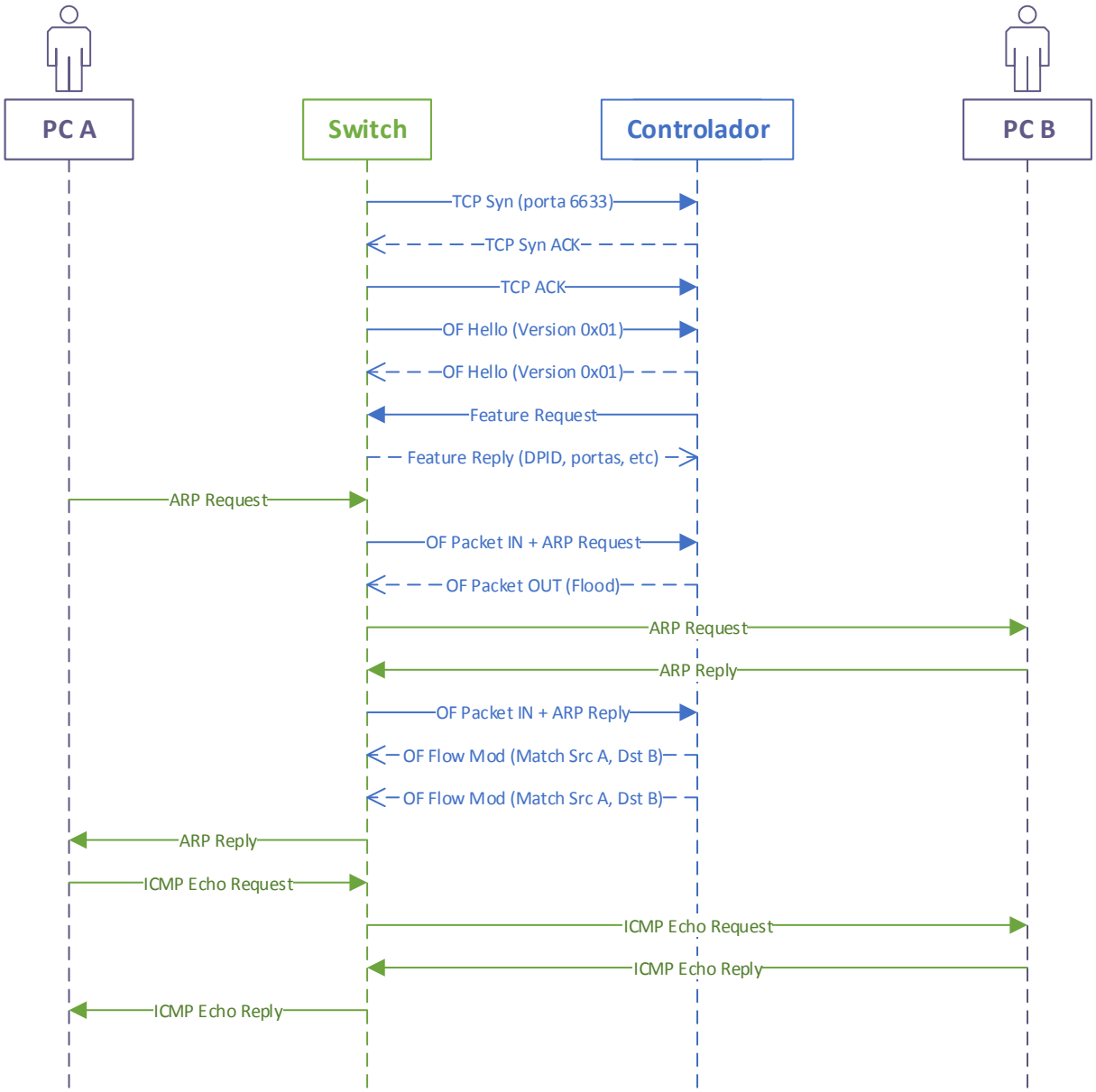

Figura 11 – Exemplo de fluxo de mensagens com programa forwarding.l2_pairs (POX)

Ainda na Figura 11, quando um pacote proveniente do PC A chega no switch, o mesmo avisa o controlador sobre a chegada de um novo pacote e envia uma cópia desse pacote para o controlador; Este observa que o endereço de destino MAC do ARP Request é Broadcast e solicita ao switch que faça *flood* do pacote para todas as portas exceto a porta de entrada; O PC B recebe o ARP Request e responde com um ARP Reply; O switch recebe o ARP Reply de B, que é um pacote unicast de B

para A, e encaminha esse pacote para o controlador; o Controlador tem agora informações suficientes para descobrir que existe uma tentativa de comunicação entre A e B pois possui o endereço dos dois no ARP Reply; O Controlador, portanto, solicita ao switch que instale duas regras na sua tabela de fluxos permitindo comunicação entre os PCs A e B, utilizando mensagens OpenFlow Mod para fazer essa solicitação. A partir deste ponto a comunicação entre os PCs A e B ocorre livremente, até o momento em que as regras da tabela de fluxos expirem por desuso ou temporização fixa dada pelo controlador.

MININET – EMULADOR DE REDES

Atualmente o emulador de redes SDN mais comumente mencionado pela literatura é o Mininet, criado por Lantz et al (2010) e hoje disponível em mininet.org (2014). Esse emulador é baseado principalmente na utilização do Open vSwitch, que suporta OpenFlow, para emular switches. Para emulação dos computadores conectados à rede, o Mininet utiliza principalmente processos rodando em diferentes "*network namespaces"* em Linux, ou seja, diferentes instâncias de interfaces de rede isoladas da rede do computador e conectadas aos switches virtuais, o que permite a simulação de centenas de nós em um laptop comum (LANTZ, 2010).

Os *vSwitches* OVS podem rodar tanto em *User Space* quanto em *Kernel Space* do Linux. Rodando em *Kernel Space* os switches podem atingir performances muito maiores, e portanto é o padrão utilizado (LANTZ, 2010).

No site mininet.org estão disponíveis instruções sobre como configurar um ambiente para testes, e é disponibilizada até mesmo uma máquina virtual pronta para ser utilizada. A instalação do Mininet e parte das principais ferramentas utilizadas para construir e avaliar o funcionamento dos cenários deste trabalho foi demonstrada apêndice A.

POX – PLATAFORMA DE CONTROLE

POX foi escolhido porque comumente é indicado pela literatura como o controlador OpenFlow mais fácil de ser utilizado no aprendizado sobre redes SDN (AL-SHABIBI et al, 2012). Sua instalação está documentada também no Apêndice A.

FUNCIONALIDADES DE REDES LOCAIS COM POX E MININET

Nesta seção serão expostos exemplos de implementações simples de aplicativos de rede implementados utilizando POX e Mininet, que mais tarde serão utilizados em conjunto para realizar o estudo de caso de migração de uma rede corporativa clássica para uma arquitetura SDN.

Topologia de Testes Simplificada

Uma topologia de testes foi criada para demonstração das principais funcionalidades, e é representada na Figura 12. Essa topologia simplesmente conecta seis hosts divididos em três redes diferentes em dois switches interligados. O controlador da rede, POX, roda em outro componente de software, porém na mesma máquina e acessível através da interface loopback.

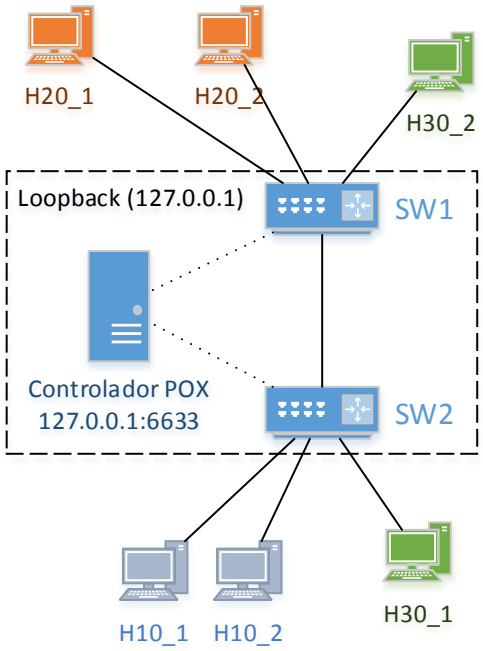

Figura 12 – Topologia L2_demo.py para testes com Mininet

Os hosts dessa topologia foram configurados com os endereços MAC e IP fixos e com rotas *default* conforme exposto na Tabela 2. O arquivo de topologia personalizada para o Mininet foi chamado de “L2_demo.py”, e está disponível no apêndice A.

Tabela 2 – Tabela de configurações por host da Figura 12

Host	Endereço MAC	Endereço IP	Default GW
H10_1	02:00:00:00:10:01	192.168.10.1	192.168.10.254
H10_2	02:00:00:00:10:02	192.168.10.1	192.168.10.254
H20_1	02:00:00:00:20:01	192.168.20.1	192.168.20.254
H20_3	02:00:00:00:20:02	192.168.20.2	192.168.20.254
H30_1	02:00:00:00:30:01	192.168.30.1	192.168.30.254
H30_2	02:00:00:00:30:02	192.168.30.2	192.168.30.254

Backward Learning com POX

Conforme já exposto anteriormente, no capítulo 2, o algoritmo mais elementar que qualquer switch Ethernet executa é o *Backward Learning*. Na instalação padrão do controlador POX diversos algoritmos que desempenha funções equivalentes já são disponibilizados.

Para demonstração, primeiramente é iniciada a rede criada no Mininet através do seguinte comando:

```
$ sudo ./L2_demo.py
*** Criando switches
*** Criando hosts e enlaces para os switches
*** Criando enlace entre os switches
*** Adicionando controlador remoto POX
*** Construindo e iniciando os elementos da rede
*** Configuring hosts
h10_1 h10_2 h20_1 h20_2 h30_1 h30_2
*** Configurando default GWs em cada host
*** Iniciando CLI
*** Starting CLI:
mininet>
```

A partir deste ponto, sem que o controlador POX seja iniciado, é feito um teste de conectividade entre todos os hosts através do comando pingall do Mininet, que trata de fazer pings de todos para todos (h10_1 pinga h10_2, h20_1, h20_2, e assim por diante):

```
mininet> pingall
*** Ping: testing ping reachability
h10_1 -> X X X X X
h10_2 -> X X X X X
h20_1 -> X X X X X
h20_2 -> X X X X X
h30_1 -> X X X X X
h30_2 -> X X X X X
*** Results: 100% dropped (0/30 received)
mininet>
```

Nota-se que todos os pings são perdidos, pois os switches da topologia simplesmente não receberam nenhuma programação do controlador, e não agem por conta própria, ao contrário de switches comuns. Neste ponto inicia-se o POX com o programa “forwarding.l2_pairs”, que nada mais é do que uma lógica de aprendizado de MACs de switches Ethernet comuns. Esse programa está disponível no repositório do POX (NOXREPO, 2014). A seguir é demonstrada a inicialização do POX com o programa forwarding.l2_pairs, realizada em um novo terminal no Ubuntu:

```
$ ./pox.py log.level --DEBUG forwarding.l2_pairs
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
INFO:forwarding.l2_pairs:Pair-Learning switch running.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-
trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
```

Dentro de alguns instantes após habilitar o POX, observa-se que os switches do Mininet se conectam ao controlador:

```
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
```

Em seguida repete-se o teste pingall no terminal do Mininet e observa-se parte dos hosts conseguem se comunicar – há comunicação apenas entre aqueles que estão na mesma rede:

```
mininet> pingall
*** Ping: testing ping reachability
h10_1 -> h10_2 X X X X
h10_2 -> h10_1 X X X X
h20_1 -> X X h20_2 X X
h20_2 -> X X h20_1 X X
h30_1 -> X X X X h30_2
h30_2 -> X X X X h30_1
*** Results: 80% dropped (6/30 received)
mininet>
```

No terminal do POX é possível ver que o controlador enviou ordens para os switches que instalassem regras de encaminhamento para os hosts da rede:

```
DEBUG:forwarding.l2_pairs:Installing 02:00:00:00:10:02 <-> 02:00:00:00:10:01
DEBUG:forwarding.l2_pairs:Installing 02:00:00:00:20:02 <-> 02:00:00:00:20:01
DEBUG:forwarding.l2_pairs:Installing 02:00:00:00:30:02 <-> 02:00:00:00:30:01
DEBUG:forwarding.l2_pairs:Installing 02:00:00:00:30:02 <->
02:00:00:00:30:01
```

Apenas os hosts que estão na mesma rede conseguem se comunicar pois não há ainda nenhum tipo de roteamento configurado. Nota-se também que duas regras são instaladas para a comunicação entre H30_1 e H30_2. Isso é explicado pelo fato de que existem dois switches entre os Hosts 30_1 e 30_2, e é necessário que ambos “aprendam” as regras de encaminhamento para realizar a encaminhamento de tráfego entre ambos.

Separação em VLANs com POX

No experimento feito na seção anterior, os hosts de redes diferentes não conseguem se comunicar devido a sua configuração de máscara de rede. No entanto, nada impede que o elemento h10_1 mude seu IP para a faixa 192.168.30.0/24 e consiga se comunicar com os hosts h30_1 e h30_2. Ou seja, não há isolamento em *Virtual LANs* no programa forwarding.l2_pairs.

Portanto ainda não ocorre isolamento adequado dos domínios L2 entre as diferentes redes. Ao pesquisar informações sobre como configurar VLANs através do controlador POX, foi observado que o mesmo não suporta nativamente todos os recursos necessários para utilização de VLANs. O controlador POX suporta apenas OpenFlow 1.0, enquanto as implementações de VLANs são suportadas do OpenFlow 1.1 em diante (MCCAULEY, 2013).

É possível utilizar VLANs com extensões da Nicira com o controlador POX, porém não há uma implementação de um programa prontamente disponível para POX – é necessário desenvolver uma aplicação por conta própria (MCCAULEY, 2013).

Outra alternativa para simular isolamento da rede de forma parecida com o uso de VLANs seria “fatiar” a rede através do uso de um controlador específico para isso, como o FlowVisor. Este controlador atua como um *proxy* entre os switches OpenFlow e múltiplos controladores OpenFlow, de forma que a rede pode ser dividida em diversas “fatias”. O FlowVisor então isola o controle dessas fatias para que apenas os controladores autorizados possam ter controle sobre elas. A divisão das fatias pode ser feita tanto por portas (camada 1) quanto por campos do cabeçalho Ethernet (camada 2) campos do cabeçalho IP (camada 3) e campos de protocolos UDP, TCP e ICMP (camada 4) (AL-SHABIBI et al, 2013).

Roteamento Nativo com o Controlador POX

Ao pesquisar uma forma de realizar roteamento em uma rede SDN simulada, foi observado que o POX não possui um programa nativo que faça roteamento.

Uma das alternativas para realizar roteamento IP é utilizar o programa forwarding.l3_learning do POX (AL-SHABIBI et al, 2012), que simula o funcionamento de gateway entre redes distintas.

Mantendo a topologia da Figura 12, foram feitos alguns testes com esse programa e foi observado que ele possibilita que as diversas redes se comuniquem entre si. Ao executar esse programa no POX, a seguinte saída pode ser vista:

```
$ ./pox.py log.level --DEBUG forwarding.l3_learning --
fakeways=192.168.10.254,192.168.20.254,192.168.30.254
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-
trusty
DEBUG:forwarding.l3_learning:Up...
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
DEBUG:forwarding.l3_learning:1 1 ARP request 192.168.10.1 => 192.168.10.2
DEBUG:forwarding.l3_learning:1 1 learned 192.168.10.1
DEBUG:forwarding.l3_learning:1 1 flooding ARP request 192.168.10.1 =>
192.168.10.2
DEBUG:forwarding.l3_learning:1 4 ARP reply 192.168.10.2 => 192.168.10.1
DEBUG:forwarding.l3_learning:1 4 learned 192.168.10.2
...
```

E no mininet o teste de pingall funciona entre todos os hosts, mesmo entre aqueles que estão em redes IP distintas:

```
mininet> pingall
*** Ping: testing ping reachability
h10_1 -> h10_2 h20_1 h20_2 h30_1 h30_2
h10_2 -> h10_1 h20_1 h20_2 h30_1 h30_2
h20_1 -> h10_1 h10_2 h20_2 h30_1 h30_2
h20_2 -> h10_1 h10_2 h20_1 h30_1 h30_2
h30_1 -> h10_1 h10_2 h20_1 h20_2 h30_2
h30_2 -> h10_1 h10_2 h20_1 h20_2 h30_1
*** Results: 0% dropped (30/30 received)
mininet>
```

No entanto, foi observado que o modo como esse programa realiza o roteamento não segue o padrão exigido em redes IP. Os “roteadores” desse programa

não segmentam as redes devidamente, e causam *flooding* indevido de conteúdo de uma rede para outra ao invés de tratar ARPs. Esse comportamento não é adequado pois não garante a segmentação de redes em diferentes domínios de *broadcast*, e vai de encontro ao comportamento que seria esperado em roteadores (TANENBAUM e WETHERALL, 2010, p. 467).

Topologia de Testes para Roteamento

Para testar uma alternativa diferente de roteamento foi criada uma nova topologia de testes, conforme exposto na Figura 13. O endereçamento dos hosts foi mantido de acordo com a Tabela 2, porém foram colocados quatro switches no total.

- Hosts da rede 192.168.10.0/24 foram colocados no SW1;
- Hosts da rede 192.168.20.0/24 no SW2;
- Hosts da rede 192.168.30.0/24 no SW3;
- SW4 é utilizado para realizar roteamento entre as redes;

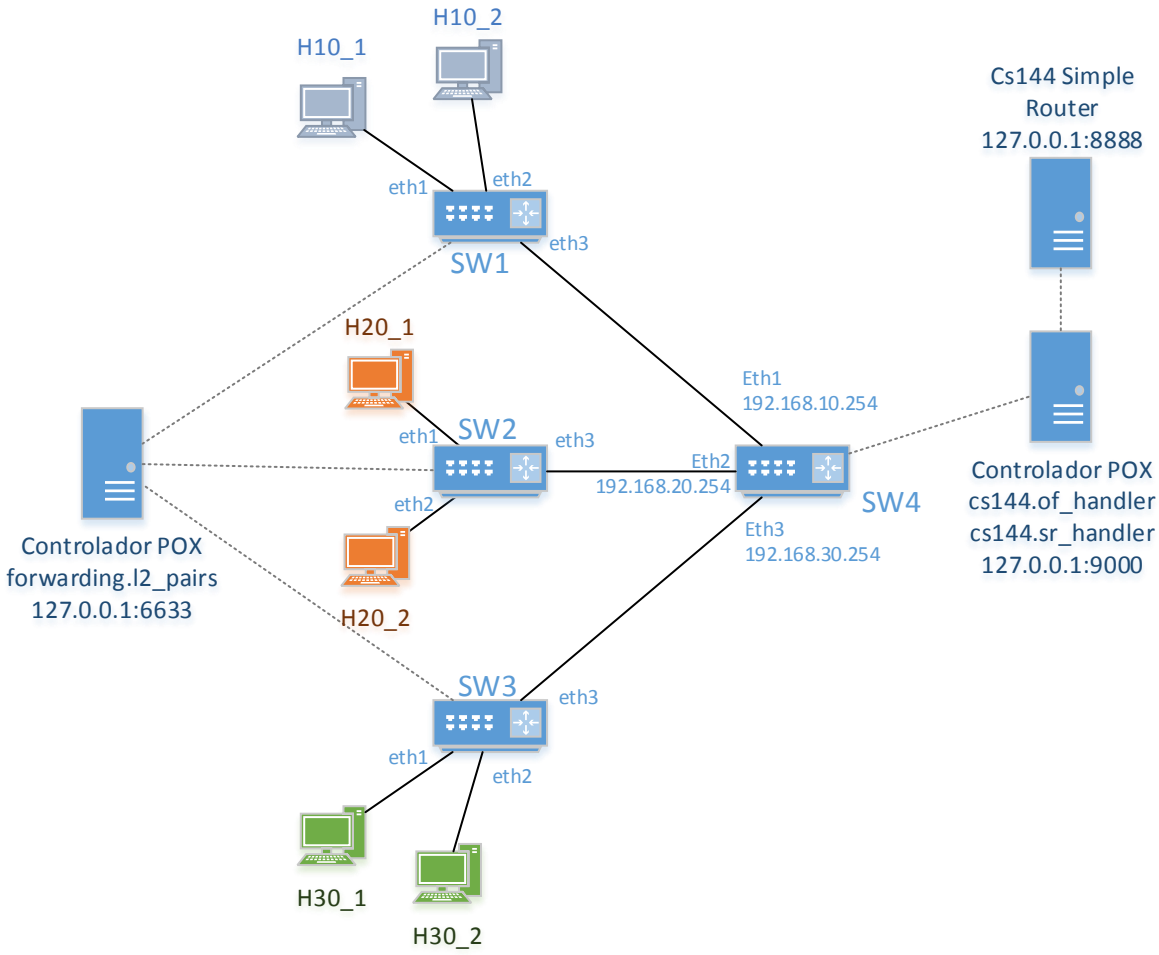

Figura 13 – Topologia L2_L3_demo_cs144_routing.py para testes com Mininet

Roteamento com POX e aplicação externa

Uma alternativa para realizar roteamento utilizando o POX, porém consultando uma aplicação externa responsável por realizar o roteamento é demonstrada também em mininet.org (2014c). O programa utilizado para roteamento é feito em linguagem C, chamado de SR (*Simple Router*). Esse roteador simples foi retirado de uma aula de laboratório de uma disciplina introdutória sobre redes de computadores da Universidade de Stanford, e seu funcionamento é explicado por Antonin (2013). O SR se comunica com o POX através da interface LTProtocol, desenvolvida na linguagem Python por Underhill (2009).

O controlador POX forwarding.l2_pairs (127.0.0.1:6633) é responsável por controlar os switches SW1, SW2 e SW3, enquanto o roteamento é feito pelo SW4

conectado a um segundo controlador POX rodando instâncias do of_handler e sr_handler (127.0.0.1:9000), que consultam a tabela de roteamento da aplicação SR (127.0.0.1:8888).

O arquivo L2_L3_demo_cs144_routing.py foi colocado no apêndice A, juntamente com outros arquivos necessários para que essa topologia funcione. A Figura 14 demonstra brevemente o funcionamento dessa solução de roteamento:

- O terminal situado no canto superior esquerdo demonstra a aplicação Simple Router rodando;
- O terminal situado no canto superior direito demonstra o Mininet rodando;
- O terminal situado no canto inferior esquerdo demonstra o controlador forwarding.l2_pairs rodando e controlando os switches L2 da topologia (SW1, 2 e 3);
- O terminal situado no canto inferior direito demonstra o controlador POX rodando as aplicações cs144.of_handler e o cs144.sr_handle, responsáveis por fazer do SW4 um roteador, de acordo com as instruções da aplicação Simple Router.

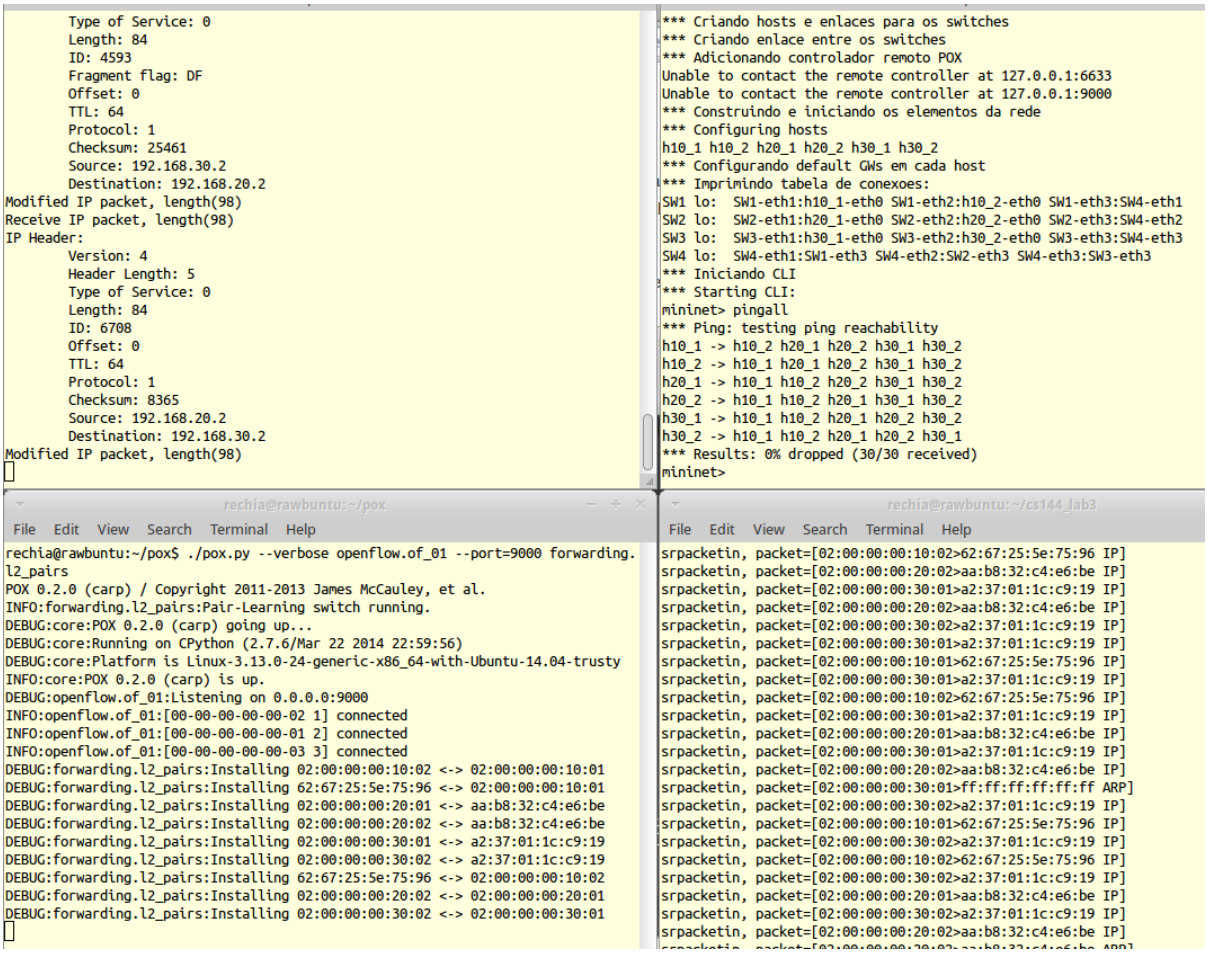

Figura 14 – Terminais abertos demonstrando a utilização do CS144 Simple Router

NAT

A função de NAT também pode ser realizada utilizando-se Mininet e POX. Um programa nativo para NAT pode ser encontrado no repositório do POX (NOXREPO, 2014), armazenado em misc/nat.py. No entanto, esse programa de NAT foi construído de maneira bastante específica, de forma que precisa ser utilizado em conjunto com um servidor DHCP.

Devido à falta de flexibilidade do NAT disponibilizado originalmente com o POX, foram realizadas algumas modificações nos programas cs144.of_handler e sr_handler originais, apenas para comprovar que o OpenFlow é capaz de realizar NAT. Com algumas linhas de código foi possível interceptar os pacotes que passavam pelo roteador e modificar campos do cabeçalho IP e realizar a função de NAT. A Figura 15 expõe o funcionamento do programa modificado, bem como a janela do Wireshark demonstrando a mudança de IP ocorrendo entre 192.168.10.1 e 192.168.30.2. O

código dessa implementação foi disponibilizado juntamente com o restante dos outros exemplos no apêndice A.

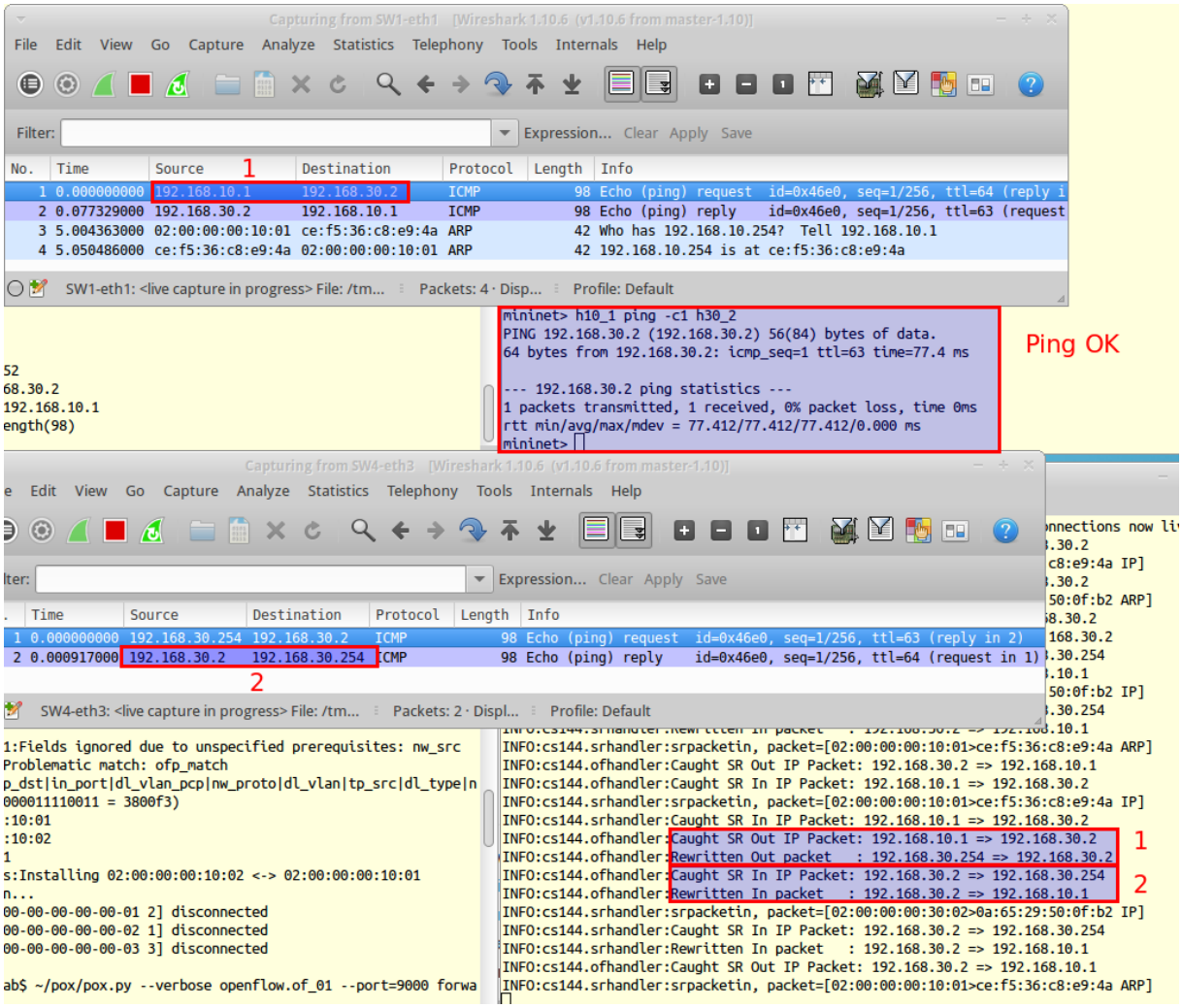

Figura 15 – Funcionamento de NAT com o Simple Router

É importante notar que a função de NAT realizada aqui foi criada apenas para demonstração, pois não mantém estado das conexões e não diferencia hosts internos, o que não está de acordo com as RFCs do NAT para TCP e ICMP (SRISURESH et al, 2008 e SRISURESH et al, 2009). Isso demonstra a flexibilidade que se tem com a programação de um switch feita através de um controlador: é possível realizar tarefas completamente personalizadas que não necessariamente estão atreladas a normas. Uma implementação mais adequada de NAT pode ser encontrada em mininet.org (2014d).

Controle de Acesso à Rede

Conforme afirmado no capítulo 2, uma das preocupações em redes corporativas é o controle de acesso para proteção dos dados acessíveis pela rede e também para proteção da integridade e robustez da própria rede. As maneiras mais simples de proteger a rede consistem em: limitar o número de endereços MAC que podem utilizar uma única porta de switch; criar listas de controle de acesso (ACL).

Para demonstrar a funcionalidade de limite de endereços MAC por porta foi criado um programa no POX que limita o número máximo de endereços dos switches. A topologia de testes Mininet e o programa para POX estão disponíveis no apêndice A (L2_L3_demo_port_security.py). A Figura 16 ilustra a proteção da porta eth4 do SW2 quando um switch adicional é colocado na porta indevidamente:

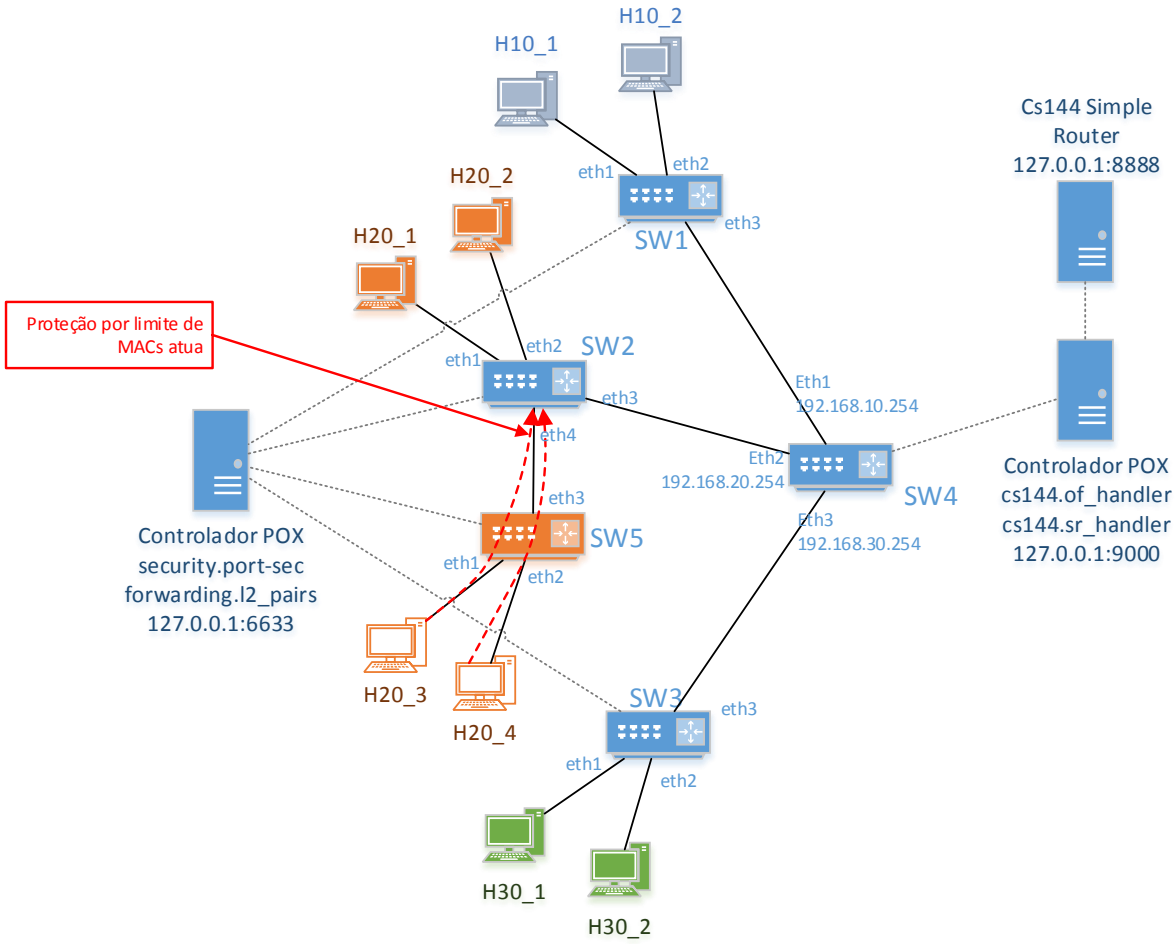

Figura 16 – Topologia para demonstração de “Port-Security” protegendo portas da rede

A Figura 17 ilustra o funcionamento do programa em POX processando os pacotes recebidos e decidindo se os pacotes serão comutados (“Adding as Trusted” e “Already Learned”) ou se serão bloqueados (“Refusing”). Ao lado direito estão

ilustradas as operações em Mininet utilizadas para testar a funcionalidade. Nota-se que h20_2 consegue pingar h10_1 e h20_3 consegue pingar h20_2. Porém quando h20_4 tenta passar pela porta eth4 do SW2, o seu tráfego é bloqueado (“Refusing”)

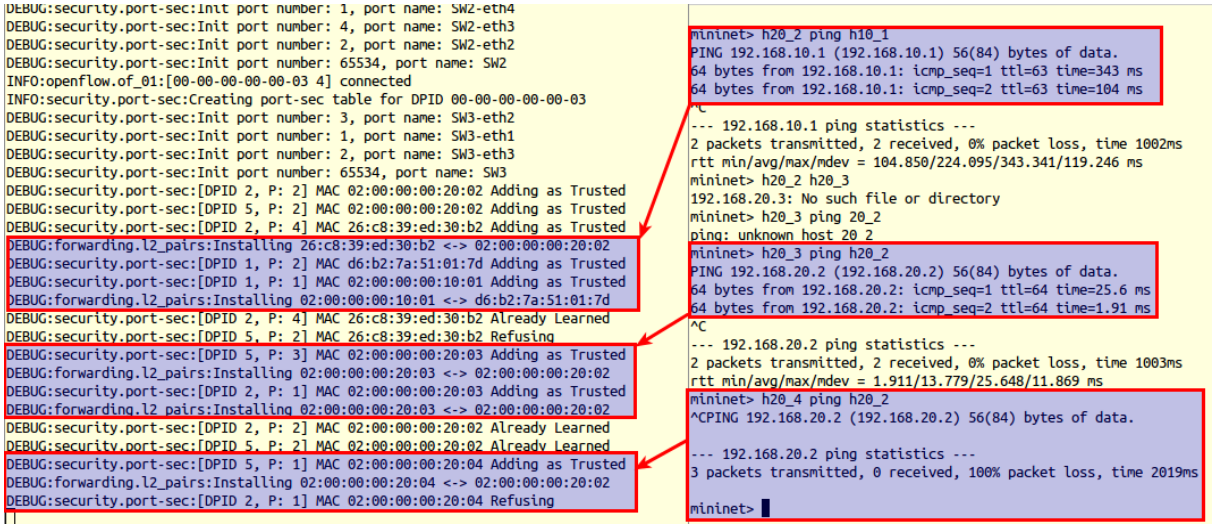

Figura 17 – Funcionamento de “Port-Security” em POX e Mininet

De forma bastante similar, é possível criar um programa que realiza a função de lista de controle de acesso com base em um arquivo de regras, como feito no programa “acl.py”, disponível no apêndice A. Em testes com a mesma topologia ilustrada na Figura 16, observa-se o funcionamento do programa nas telas mostradas na Figura 18: Comunicação entre os hosts h10_2 e h20_2 é bloqueada através de uma regra do tipo L3_IP, e a comunicação entre os hosts h30_1 e h30_2 é bloqueada através de uma regra do tipo L2.

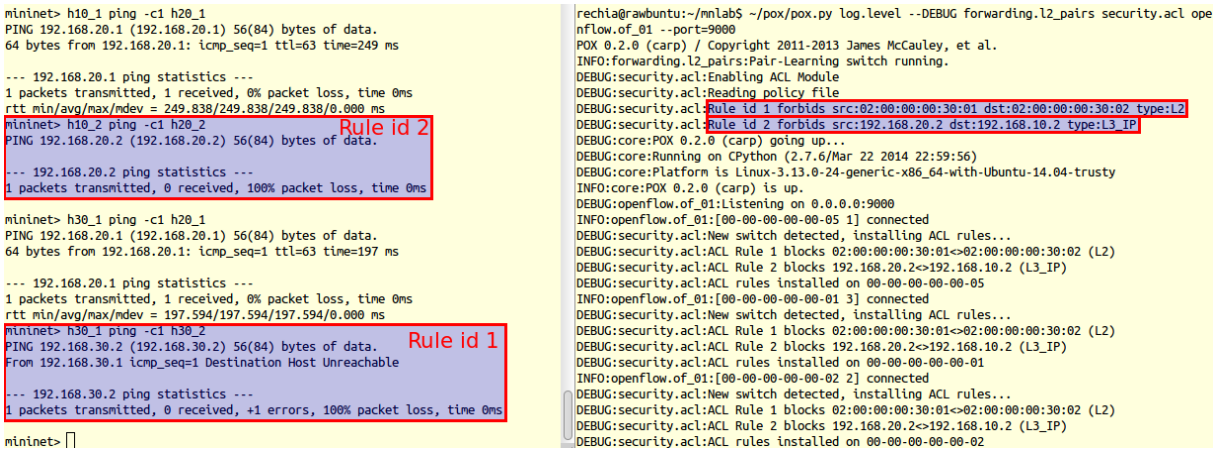

Figura 18 – Funcionamento de ACL em POX e Mininet

Roteamento com RouteFlow e POX

Ainda outra alternativa para realização de roteamento em uma rede baseada em OpenFlow é o projeto RouteFlow, do CPqD (CPQD, 2014). A ideia principal do projeto RouteFlow é criar uma topologia virtual responsável por gerar o plano de controle com um protocolo de roteamento conhecido, como OSPF ou BGP, e a partir disso interagir com um controlador OpenFlow fazendo com que os switches controlados possam atuar como roteadores. De maneira similar ao funcionamento do RCP, o RouteFlow realiza todo o cálculo de rotas na sua topologia virtual e posteriormente "instala" as rotas necessárias através do controlador RFProxy que se comunica com o Controlador Openflow, como o POX. A arquitetura de uma rede SDN utilizando o RouteFlow para roteamento é representada pela Figura 19, retirada do site do próprio projeto (CPQD, 2014a).

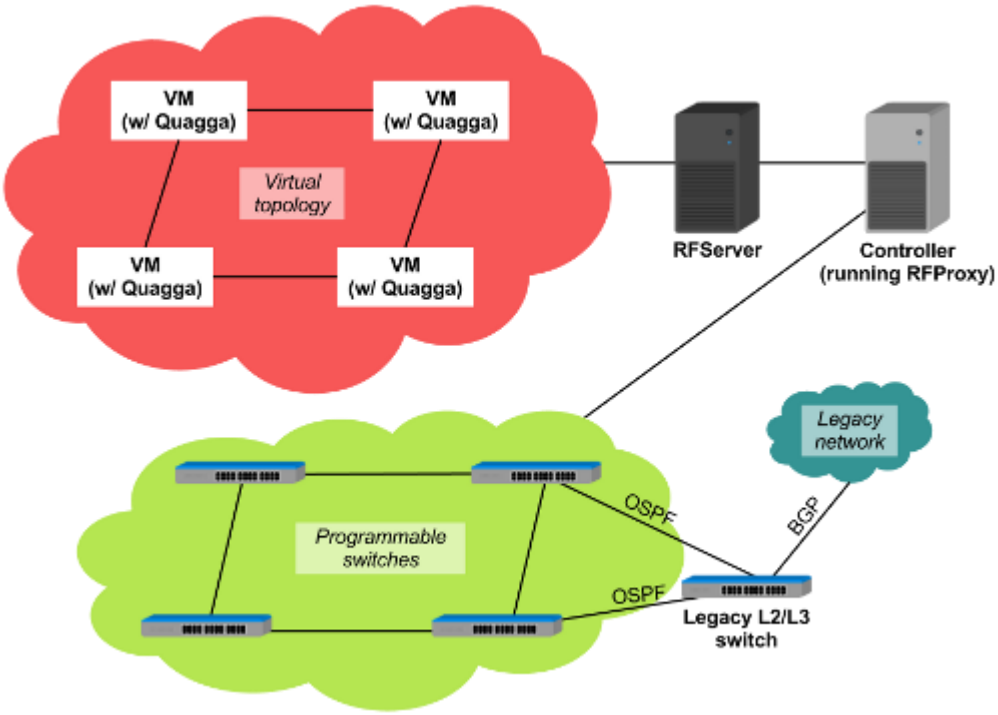

Figura 19 – Arquitetura do projeto RouteFlow

Foram feitos alguns testes do RouteFlow com o Ubuntu 14.04 com as últimas versões disponíveis do Mininet e POX, porém não se obteve sucesso na configuração de uma topologia de testes com o RouteFlow.

Com uma máquina virtual criada com Ubuntu 12.04, foi possível executar o tutorial do cenário 2 do RouteFlow exposto no repositório do projeto (CPQD, 2014b), que utiliza RouteFlow controlando uma rede Mininet através de um controlador POX.

Mesmo assim o funcionamento da rede experimental de RouteFlow com Mininet apresentou alguns problemas, e o roteamento da rede funcionou adequadamente. Frequentemente perdia-se o “sincronismo” entre os roteadores virtuais e as instâncias de Open vSwitch quando apenas um dos programas era reiniciado.

OUTROS CONTROLADORES SDN

Diversos aspectos devem ser considerados para a escolha de um controlador para uso em produção. Facilidade de uso, integração com OpenStack, linguagem de programação utilizada, suporte da comunidade de desenvolvedores, e aspectos de performance são alguns dos fatores apontados por Feamster (2014) em uma aula do curso sobre SDN no Coursera.org.

Todos os experimentos deste capítulo para prova de conceito foram realizados com o POX, pois é um controlador fácil de se utilizar. Porém para uso em ambiente de produção recomenda-se utilizar alternativas mais robustas e com melhor performance, como o OpenDayLight, NOX ou Floodlight. Feamster (2014) também mostra uma tabela com o sumário comparativo entre alguns controladores, adaptado na Tabela 3, comparando cinco diferentes controladores quanto à:

- Linguagem: linguagem de programação utilizada;
- Performance: avaliação qualitativa de performance geral, em relação aos demais controladores;
- Distribuído: se pode ser utilizado em sistemas distribuídos, como um cluster de computadores, com a carga de processamento dividida;
- Versão OpenFlow: versões de OpenFlow que o controlador suporta;
- Suporte a OpenStack: se suporta ou não interação com o OpenStack para uso em controle de redes virtualizadas;
- Facilidade de uso: Avaliação qualitativa geral da facilidade em se utilizar o controlador, incluindo aspectos como disponibilidade de documentação, interface com o usuário, suporte da comunidade de desenvolvedores, dentre outros.

Tabela 3 – Comparativo de controladores OpenFlow

	NOX	**POX**	**Ryu**	**Floodlight**	**OpenDayLight**
Linguagem	C++	Python	Python	Java	Java
Performance	Rápido	Lento	Lento	Rápido	Rápido
Distribuído	Não	Não	Sim	Sim	Sim
Versão OpenFlow	1.0*	1.0	1.0,1.1,1.3,1.4	1.0	1.0,1.3
Suporte a OpenStack	Não	Não	Sim	Sim	Sim
Facilidade de Uso	Moderado	Fácil	Moderado	Difícil	Difícil

A Tabela 3 não pretende apresentar uma lista exaustiva de controladores, apenas demonstrar alguns aspectos que podem ser utilizados para compará-los. Nunes et al (2014) apresentam uma tabela com diversos controladores SDN, e Shah et al (2013) apresentam métodos quantitativos de comparação de performance de controladores OpenFlow.

CONCLUSÃO

Com as informações apresentadas neste capítulo conclui-se que é possível reproduzir funcionalidades comumente utilizadas em redes corporativas através da programação de controladores SDN. Foi possível observar que o controlador POX pode ser utilizado para demonstrar a implementação de diversas funções de rede, porém o desenvolvimento dessas aplicações exige bons conhecimentos sobre redes Ethernet/IP e também habilidade em programação pelo menos em nível básico.

Além dos controladores apresentados neste capítulo, existem muitos outros controladores disponíveis no mercado, tanto de código aberto quanto comerciais, e cada um com sua finalidade. Diferentes aplicações precisam ser avaliadas caso a caso para se escolher um controlador SDN adequado.

CAPÍTULO 4 – MIGRAÇÃO DE REDE PARA SDN

Este capítulo apresenta o estudo de caso de uma rede corporativa utilizando alguns dos serviços expostos no capítulo 2, e traça um esboço de plano de migração dessa rede para um modelo SDN.

Para se fazer este estudo será utilizado o Packet Tracer, um simulador de redes proprietário da CISCO, pois é fácil de utilizar e também é uma boa referência de equipamentos comerciais com as funcionalidades descritas no capítulo 2.

TOPOLOGIA PROPOSTA

Utilizando o Packet Tracer foi criada uma topologia de rede representando um caso de uso de uma empresa, apenas para prova de conceito. A topologia está representada pela Figura 20 e foi configurada para incluir grande parte dos serviços descritos no capítulo 2.

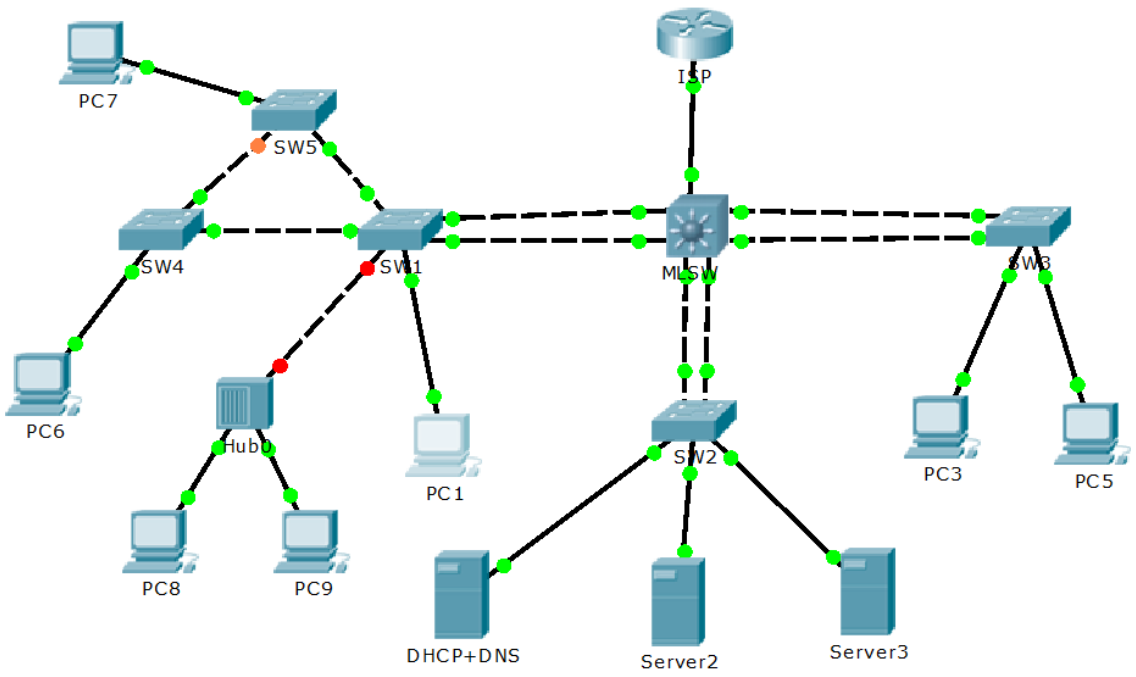

Figura 20 – Topologia de rede corporativa criada no Packet Tracer

Toda a configuração realizada para a topologia proposta foi documentada no apêndice B. Foram configuradas funcionalidades tipicamente

utilizadas em ambientes corporativos de pequeno porte: Separação em VLANs, Roteamento entre VLANs, Roteamento estático, DHCP, DNS, NAT, Limitação de endereços MAC (*Port-security*), Spanning-tree, Agregação de enlaces em camada 2 (*Port-channel*), listas de controle de acesso (ACL).

PLANO DE MIGRAÇÃO

A ONF ilustra alguns casos de migração de redes legadas para redes SDN em seu documento "*Migration Use Cases and Methods*" (ONF, 2014b). O documento demonstra exemplos de planos de migração divididos basicamente em três etapas: planejamento pré-migração, processo de migração e aceitação após migração.

Planejamento Pré-Migração

Análise de Lacunas (Gap Analysis)

O primeiro passo na abordagem de migração é a análise de lacunas (*Gap Analysis*), que consiste em analisar os serviços oferecidos pela rede atual e qual seria o impacto causado nesses serviços por uma migração. Deve-se garantir que existem alternativas viáveis para se cobrir lacunas quando for identificado que a arquitetura SDN apresentará a falta de algum serviço que existe na rede antiga (ONF, 2014b, p52).

Para a topologia proposta, a análise de lacunas é feita apenas para os switches da rede, pois inicialmente são os únicos dispositivos aos quais se aplica o uso de controlador. Considera-se uma proposta de solução SDN baseada em controlador POX, e a análise de lacunas é exposta na Tabela 4.

Nota-se que o controlador POX ainda não tem todas as implementações prontas, principalmente para realizar separação em VLANs e roteamento. No entanto, isso depende apenas de implementação em software, e não é necessário que o switch OpenFlow suporte isso explicitamente. Foi demonstrado no capítulo 3 que é possível

fazer roteamento utilizando aplicações externas, ou mesmo criar um programa específico para roteamento.

Tabela 4 – Tabela de Análise de Lacunas

Serviço de Rede	**Solução CISCO (Packet Tracer)**	**Solução SDN (Controlador POX)**
Separação em VLANs	Qualquer SW	Nova Impl. qualquer SW
Roteamento entre VLANs	Apenas MLSW	Nova Impl. qualquer SW
Roteamento estático	Apenas MLSW	Nova Impl. qualquer SW
DHCP	Qualquer SW / Servidor	Qualquer SW / Servidor
DNS	Depende de Servidor	Nova Impl. qualquer SW / Servidor
NAT	Apenas MLSW	Qualquer SW
Port-security	Qualquer SW	Qualquer SW
Spanning-tree	Qualquer SW	Qualquer SW
Port-channel	Qualquer SW	Nova Impl. qualquer SW
ACL	Qualquer SW	Qualquer SW

Na Tabela 4 pode ser observada a flexibilidade trazida por uma solução SDN. Como toda a lógica das funcionalidades é feita no controlador, qualquer SW que suporte OpenFlow 1.0 pode realizar as tarefas programadas pelo controlador. Embora possa depender de nova implementação, quaisquer funções de rede podem ser migradas para o controlador.

Todos os aspectos analisados podem ser atendidos por uma solução SDN, ainda que alguns dependam de novas implementações. Novas implementações sempre podem apresentar riscos adicionais para a migração, porém não há nenhum impedimento intrínseco à arquitetura SDN.

Listas de Verificação (Check Lists)

Sabendo quais serviços existem na rede, é necessário criar listas de verificação para determinar se estão funcionais antes e depois da migração (ONF, 2014b, p52). Por exemplo: verificação de conectividade IP entre diversos pares de origem/destino, verificar se o DHCP funciona, verificar se as ACLs estão bloqueando/permitindo tráfego corretamente, etc.

Para a topologia apresentada anteriormente neste capítulo propõe-se a lista de verificação exposta na Tabela 5, criada para ilustrar a verificação da configuração detalhada no Apêndice B.

Tabela 5 – Lista de Verificação

Serviço de Rede	**Procedimento de Teste**

Separação em VLANs	1. Em um PC conectado ao SW5, VLAN 30, alterar IP manualmente da faixa 192.168.30.0/24 para a faixa 192.168.20.0/24 (vlan errada) 2. Confirmar que não é possível pingar hosts da rede 192.168.20.0/24 3. Repetir procedimento de maneira análoga em outros switches da rede, testando o isolamento entre as VLANs
DHCP e DNS	1. Em PCs conectados a cada um dos switches da rede verificar IP adquirido através de linha de comando (Linux: ifconfig ou Windows: ipconfig) 2. O IP adquirido deve corresponder à rede da VLAN da porta. Exemplo: 192.168.30.0/24 para VLAN 30, 192.168.20.0/24 para a VLAN 20 3. Confirmar configuração de DNS e Default Gateway feitas pelo DHCP com ping para um nome da internet. Exemplo: ping www.google.com
Roteamento entre VLANs	1. Verificar que é possível um host da rede 192.168.10.0/24 (VLAN 10) pingar hosts de outras redes locais. Exemplo: 192.168.10.1: ping 192.168.20.1, ping 192.168.30.1, etc. 2. Utilizar traceroute para verificar caminho percorrido
Roteamento estático e NAT	1. Verificar que a rota estática default configurada para a Internet permite acessibilidade à Internet de qualquer ponto da rede interna. 2. Observar que NAT está funcionando pois os endereços da Internet são acessíveis e os pings enviados/recebidos para a Internet em IPv4 são traduzidos entre a rede privada/pública. 3. Utilizar traceroute para verificar caminhos percorridos
Port-security	1. Utilizando um switch simples ou hub em uma porta de acesso, observar que a comunicação dessa porta é completamente bloqueada assim que mais de um host tenta usar essa porta.
Spanning-tree	1. Verificar que a comunicação entre os hosts do anel composto pelos switches SW1, SW4 e SW5 ocorre sem perdas. Loops na camada 2 da rede causariam alta latência e perda de pacotes. 2. Utilizar ping e/ou ferramenta de tráfego (iperf, por exemplo).
Port-channel	1. Verificar interfaces agregadas em camada 2 não causam loops na rede através da passagem de tráfego por *Port-channels* por diferentes pares de origem/destino. 2. Observar que tráfego entre diferentes pares de origem/destino é balanceado entre os enlaces do *Port-channel*, comprovando funcionamento deste.
ACL	1. Observar que tráfego entre diversos pares de origem/destino é bloqueado ou permitido de acordo com as regras configuradas. 2. Utilizar ping para testes de regras que contém apenas IPs. Caso as regras sejam específicas para determinadas portas do protocolo de transporte (TCP ou UDP), utilizar nmap ou ferramenta similar que permite verificar portas liberadas.

Procedimentos de Retorno em Caso de Falha (Back-out Procedures)

Um procedimento detalhado passo a passo deve ser criado para que a rede seja migrada para a nova tecnologia. Também deve ser criado um procedimento de retorno ao estado original da rede em caso de falha. É interessante investigar a possibilidade de se automatizar o procedimento de retorno, para minimizar o tempo de indisponibilidade da rede em caso de resultados inesperados (ONF, 2014b, p52).

Praticamente todos os switches Ethernet gerenciáveis existentes no mercado possuem funções de “salvar” e “carregar” configurações diferentes em sua memória não-volátil, geralmente memória do tipo *flash* (EEPROM). O procedimento para retorno em caso de falha pode variar de acordo com o fabricante do equipamento, e consiste apenas em selecionar uma posição antiga de memória e restaurá-la.

Para a rede proposta neste trabalho supõe-se que todos os equipamentos suportam OpenFlow, pois diversos modelos mais novos de switches o suportam (NUNES et al, 2014). A seguir é demonstrado um procedimento de backup realizado em switches Cisco, apenas como exemplo.

1. Antes de iniciar o procedimento de upgrade de firmware, deve ser feito um backup das configurações. Deve-se verificar se há diferenças entre as configurações *running* e *startup*, e salvar em caso positivo.

```
Switch#show running-config
...
Switch#show startup-config
...
Switch#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...
[OK]
```

2. Em seguida verificar se há espaço livre na memória flash e salvar uma cópia de backup da configuração:

```
Switch#show flash:
Directory of flash:/

    1  -rw-     4414921          <no date>  c2960-lanbase-mz.122-25.FX.bin

64016384 bytes total (59601463 bytes free)
Switch#copy running-config flash:
Destination filename [running-config]? backup.cfg
Building configuration...
[OK]
```

```
Switch#show flash:
Directory of flash:/

    2  -rw-        1037          <no date>  backup.cfg
    1  -rw-     4414921          <no date>  c2960-lanbase-mz.122-25.FX.bin

64016384 bytes total (59600426 bytes free)
Switch#
```

a. A qualquer momento é possível restaurar a configuração inicial do switch apenas com o seguinte comando:

```
Switch#copy flash: running-config
Source filename []? backup.cfg
Destination filename [running-config]?

1037 bytes copied in 0.416 secs (2492 bytes/sec)
Switch#
%SYS-5-CONFIG_I: Configured from console by console
```

b. Se tudo estiver funcionando após restaurar a configuração para a running-config, copiar novamente a configuração para a startup-config.

```
Switch#copy running-config startup-config
Destination filename [startup-config]?
Building configuration...
[OK]
```

3. Realizar *a* atualização de *firmware* segundo documentação do fabricante, e utilizar os passos 2a e 2b para retornar à configuração original em caso de problemas

Análise de Funcionalidades (Feature-Set Analysis)

Uma análise detalhada das funcionalidades desejadas que devem rodar no controlador e switch OpenFlow, para garantir que as funcionalidades são consistentes com os requisitos da rede (ONF, 2014b, p52).

A análise de algumas das funcionalidades foi realizada no capítulo 3 de forma mais detalhada (NAT, Roteamento, Separação em VLANs). Sabe-se que diversas das funcionalidades implementadas no controlador POX precisam de implementações adicionais para que possam substituir por completo a rede atual, conforme exposto na Tabela 4.

Processo de Migração

Método de Implantação

A ONF (2014b, p43) define três tipos de dispositivos que existem no processo de migração:

1. Switch Legado (L*egacy switch*): switch que não suporta OpenFlow;
2. Switch Híbrido (*Hybrid switch*): switch que suporta OpenFlow em conjunto com o modo legado de operação;
3. Switch OpenFlow: switch que suporta apenas o modo OpenFlow de operação;

Dependendo dos tipos de dispositivos disponíveis na rede, diferentes métodos de migração podem ser adotados. A ONF (2014b, p40) propõe três métodos distintos de implantação para redes SDN:

1. **Implantação Greenfield**: neste tipo de implantação não existem restrições de interoperabilidade. A nova rede SDN é implantada a partir do zero para substituir a rede legada, ou a atualização da rede legada é feita de maneira completa, sem etapas intermediárias. Os protocolos de controle da rede (e.g. protocolos de roteamento) são inteiramente substituídos por um controlador OpenFlow ou similar (ONF, 2014b, p44);
2. **Implantação mista**: este método de implantação assume que novos dispositivos OpenFlow são implantados paralelamente aos dispositivos antigos. Os dois tipos de dispositivos deverão interoperar através de protocolos de controle da rede legada, trocando, por exemplo, informações de roteamento (ONF, 2014b, p44). Neste caso os switches OpenFlow não rodam os protocolos legados de roteamento, mas podem ser instruídos pelo controlador e enviar PDUs de protocolos de roteamento para estabelecer adjacências, como no caso do projeto RouteFlow (CPQD, 2014a);
3. **Implantação híbrida**: este método de implantação prevê a coexistência de dispositivos híbridos, dispositivos puramente OpenFlow e dispositivos legados. Os dispositivos híbridos se

> comunicam tanto com o controlador OpenFlow quanto com outros dispositivos legados de forma nativa (ONF, 2014b, p44).

Neste trabalho são utilizados ambientes de simulação/emulação diferentes para os cenários de rede legada (Packet Tracer) e SDN (Mininet com POX). Dessa forma considera-se que o método de implantação precisa ser *Greenfield,* ou seja, partindo-se da implantação do zero para substituir a rede legada.

Ferramentas de Provisionamento e Monitoração

Todas as ferramentas de gerência de rede devem estar configuradas para que seja possível monitorar o estado dos dispositivos de rede e suas interfaces, tráfego, alarmes, e outros aspectos durante e depois da migração (ONF, 2014b, p52).

Para este trabalho considera-se que todo o provisionamento da rede é feito através de CLI, utilizando comandos específicos para exibição do estado atual da rede e também através da visualização de logs presentes no equipamento e armazenados em memória flash.

Pode-se também utilizar ferramentas de gerência SNMP com monitoração de *traps*, como Zabbix ou Nagios (ZABBIX.COM, 2014) (NAGIOS.ORG, 2014). As ferramentas de gerência devem ser suportadas pelos switches da rede antes e depois da migração para OpenFlow, para possibilitar comparação do estado da rede.

Controle de Versões OpenFlow

Como existem diversas versões do OpenFlow, deve-se garantir que todos os elementos de rede apresentam a compatibilidade necessária com o controlador OpenFlow.

Neste trabalho é utilizado para testes apenas o Open vSwitch versão 1.9.3, que suporta a versão 1.0 do OpenFlow. Versões mais novas do OVS, como a 2.3, possuem suporte para OpenFlow 1.3, e em caráter experimental OpenFlow 1.5 (OPENVSWITCH, 2014).

Atualizações

Antes da migração todos os dispositivos aplicáveis devem ser atualizados para um firmware/software que suporte OpenFlow (ONF, 2014b, p52). Atualmente diversos fabricantes suportam o modo híbrido de operação do OpenFlow, o que significa que é possível fazer a atualização de um switch e continuar utilizando como um switch tradicional para encaminhamento de tráfego segundo parâmetros de camada 2 e 3 (ONF, 2014a). É possível, por exemplo, fazer com que pacotes que não correspondam com nenhuma regra da tabela de fluxos OpenFlow sejam encaminhados através da pilha de protocolos do próprio switch híbrido (NUNES et al, 2014).

Neste trabalho propõe-se apenas a utilização de switches virtuais Open vSwitch para testes de conceitos de SDN, porém em casos de uso reais devem ser consideradas questões de upgrade de firmware. É importante ressaltar que as funcionalidades SDN são implementadas no controlador, portanto depende muito pouco do hardware utilizado, desde que o suporte ao OpenFlow seja garantido pelo fabricante.

Conectividade e Resolução de Problemas.

Deve ser confirmada a conectividade entre os dispositivos OpenFlow e controlador, bem como o funcionamento de serviços de acordo com a lista de verificação criada na etapa de pré-migração (ONF, 2014b, p52).

Por exemplo, podem ser criados usuários de rede para monitoramento de disponibilidade de serviços. Ferramentas como ping e traceroute também podem auxiliar no processo de detecção e resolução de problemas de rede (ONF, 2014b, p53).

Aceitação após Migração

Após o término da migração recomenda-se continuar com a monitoração da rede por algumas semanas ou meses, analisando tráfego de teste e tráfego de usuários com ferramentas como Wireshark, ping e traceroute.

A monitoração periódica pode ser utilizada para se criar estatísticas de conectividade, performance e estabilidade da rede. O servidor ou servidores com instâncias de controladores OpenFlow devem ser monitorados constantemente para se garantir que recursos de processamento e memória não se esgotam. Alguns parâmetros interessantes a serem monitorados em controladores OpenFlow são:

- Tempo de resposta para criação de novos fluxos, ou *flow setup time* (ONF, 2014b, p37);
- Latência de ida e volta entre controlador e switches, ou RTT, *round trip time* (ONF, 2014b, p37);

CONCLUSÃO

Neste capítulo foi exposta uma configuração de rede corporativa tradicional que pode ser migrada para uma rede com arquitetura SDN. Foram expostos diversos aspectos a serem observados para que seja feita uma implantação de rede SDN com base em documentos da ONF.

A análise de lacunas realizada neste capítulo sugere que o controlador POX não é um controlador adequado para substituir completamente uma rede corporativa, sobretudo pela falta de aplicações de rede prontas.

Nota-se que o uso de uma arquitetura SDN permite uma flexibilidade maior de aplicações em qualquer ponto da rede utilizando um hardware genérico. Por exemplo, funções de roteamento e NAT podem ser realizadas em qualquer ponto da rede SDN, pois dependem muito mais da programação do controlador do que da

Aspectos de performance e robustez do controlador não foram analisados neste trabalho, porém são fatores críticos também para aplicações reais. Como sugerido no capítulo 1, outros trabalhos apresentam estudos para essas questões, como as análises de performance e arquitetura de Shah et al (2013) e Kong et al (2013). Sgambelluri et al (2013) e Sharma et al (2011) propõem melhorar a resiliência da rede através da diminuição do tempo de indisponibilidade da rede em caso de falhas. Hock et al (2013) propõem um método de posicionamento de controladores na rede para otimizar métricas de resiliência e performance considerando aspectos de balanceamento de carga e latência de resposta de controladores em casos de falhas.

CONCLUSÃO

CONSIDERAÇÕES FINAIS

Este trabalho apresentou diversas tecnologias SDN recentes que podem possibilitar rápida inovação e flexibilidade em redes Ethernet e TCP/IP. A revisão bibliográfica feita no capítulo 1 demonstra que existem muitas alternativas de arquiteturas SDN em desenvolvimento neste momento, porém poucas dessas alternativas alcançaram um nível de maturidade aceitável para que sejam prontamente adotadas por usuários finais.

Observou-se através de experimentos com o emulador de redes Mininet e o controlador POX que diversas das tecnologias de rede tradicionais expostas no capítulo 2 podem ser implementadas em arquiteturas SDN. De um ponto de vista funcional o controlador POX é bastante flexível. Porém a flexibilidade trazida pela nova tecnologia não se traduz em facilidade de implantação.

No capítulo 3 foi demonstrada a programação de algumas aplicações simples em uma rede SDN. Do ponto de vista de desenvolvimento de aplicações para redes, é notável a facilidade com que se pode implementar e testar ideias utilizando ambientes virtuais, que podem ser migrados para ambientes reais sem muitas dificuldades (LANTZ, 2010). No entanto, usuários que estavam habituados apenas em configurar redes precisariam aprender a programar para atingir os mesmos objetivos que atingiam antes apenas seguindo manuais de configuração.

As redes atuais são sistemas distribuídos de alta complexidade, que foram desenvolvidos e utilizados por décadas até alcançarem os conhecidos padrões de estabilidade, performance e confiabilidade. É perfeitamente possível substituir redes quaisquer tipos de redes legadas por arquiteturas SDN, mas ainda não é uma tarefa trivial. Foi demonstrado que o OpenFlow possibilita a programação da rede com um baixo nível de abstração, e torna possível controlar cada fluxo de tráfego da rede separadamente, porém essa tarefa exige conhecimentos sobre redes e programação.

Ainda não se tem uma interface API Norte – com mais alto nível de abstração – que esteja madura o suficiente para ser prontamente utilizada por operadores de rede e usuários finais. Conforme exposto no capítulo 4, a implantação de uma rede SDN precisa ser cuidadosamente planejada, e muitas lacunas podem

ser encontradas no momento de se fazer uma migração de rede legada para a arquitetura SDN. Essas lacunas podem ser preenchidas através do desenvolvimento de software de controle da rede. Essa é uma área promissora, pois não existem muitos profissionais no mercado prontos para fazer esse tipo de desenvolvimento, conforme apontado por Feamster e McKeown (2013) em entrevista.

Existem também alternativas promissoras em desenvolvimento para a interface API Norte, como o projeto OpenDaylight e algumas linguagens e plataformas de programação como Frenetic, Pyretic e Resonance, que oferecem níveis mais altos de abstração para controle da rede. Arquiteturas SDN vem sendo amplamente estudadas e desenvolvidas por empresas fabricantes de switches, como Cisco e Juniper, bem como por empresas de soluções de virtualização como VMWare e Microsoft (KERNER, 2014). Além disso, a comunidade acadêmica tem demonstrado grande interesse pelas possibilidades de inovação trazidas por arquiteturas SDN (KERNER, 2014). No entanto, observa-se que o mercado de redes corporativas privadas ainda não está completamente decidido em adotar tecnologias SDN tão cedo (KERNER, 2014).

O fato de grandes empresas como Google e Amazon já utilizarem SDN em suas redes comprova que existe potencial de reduzir custos de gerência e otimizar a utilização através do uso de arquiteturas SDN (MA, 2014).

TRABALHOS FUTUROS

Este trabalho explorou apenas uma fração das tecnologias SDN. Sugere-se explorar alternativas como a plataforma do OpenDaylight para resolver as lacunas encontradas na utilização do controlador POX para redes corporativas. Diversas linhas de pesquisa podem ser criadas apenas com o desenvolvimento de novas aplicações de rede inovadoras.

Limoncelli (2013) apresenta uma analogia bastante interessante de aplicações SDN com o mercado de aplicativos para telefones celulares: no passado aplicativos embarcados em telefones celulares eram controlados exclusivamente pelos fabricantes. Hoje é imensa a variedade de aplicativos que podem ser baixados e utilizados em qualquer *smartphone* atual. O conceito de SDN representa uma democratização dos aplicativos de rede similar ao que ocorreu com os aplicativos de

celulares. Ao contrário do que se observa hoje com funcionalidades de rede proprietárias, novas aplicações de rede baseadas em SDN são limitadas apenas pela imaginação.

REFERÊNCIAS BIBLIOGRÁFICAS

AHRENHOLZ, J. Comparison of CORE network emulation platforms, San Jose, CA, 2010. **MILITARY COMMUNICATIONS CONFERENCE, 2010**. p. 166–171. DOI: 10.1109/MILCOM.2010.5680218

AL-SHABIBI, A. et al. **POX Wiki - Open Networking Lab**. 02 Jul. 2012. Disponível em: <https://openflow.stanford.edu/display/ONL/POX+Wiki>. Acesso em: 29 mai. 2014.

AL-SHABIBI, A. et al. **Flowvisor**. 20 jun. 2013. Disponível em: <https://github.com/OPENNETWORKINGLAB/flowvisor/wiki>. Acesso em: 10 jul. 2014.

ANTONIN. **CS144 Lab 1**: Simple Router. 23 set. 2013. Disponível em: <http://www.stanford.edu/class/cs144/labs-fall2013/lab1/lab1_Antonin.pdf>. Acesso em: 7 jun. 2014.

BJORKLUND, M. **YANG** - A Data Modeling Language for the Network Configuration Protocol (NETCONF). out. 2010. Disponível em: <http://tools.ietf.org/html/rfc6020>. Acesso em: 20 abr. 2014.

CASADO, M. et al. Ethane: Taking Control of the Enterprise, New York, NY, USA, 2007. **Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications**. ACM. p. 1–12. DOI: 10.1145/1282380.1282382

CISCO SYSTEMS, INC. OpFlex: **An Open Policy Protocol**. [2014]. Disponível em: <http://cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html>. Acesso em: 7 mai. 2014.

CIAMPA, M. **Security+ Guide to Network Security Fundamentals**. 4. Ed. Boston, Massachusetts: Course Technology, 25 Jul. 2011.

CPQD. **RouteFlow**: Virtual IP Routing Services over OpenFlow networks. [2014]. Disponível em: <http://routeflow.github.io/RouteFlow/>. Acesso em: 1 jun. 2014.

_____. RouteFlow GitHub Repository. 19 fev. 2014. Disponível em: <https://github.com/routeflow/RouteFlow>. Acesso em: 1 jun. 2014.

DEVLIC, A. et al. A Use-Case Based Analysis of Network Management Functions in the ONF SDN Model. In: 2012 EUROPEAN WORKSHOP ON SOFTWARE DEFINED NETWORKING (EWSDN), 2012. **2012 European Workshop on Software Defined Networking (EWSDN)**. p. 85–90. DOI: 10.1109/EWSDN.2012.11

ENNS, R. et al. **NETCONF Configuration Protocol**. jun. 2012. Disponível em: <http://tools.ietf.org/html/rfc6241>. Acesso em: 20 abr. 2014.

ETSI NFV. **Network Functions Virtualization White Paper**. 22 out. 2012. Disponível em: <http://portal.etsi.org/nfv/nfv_white_paper.pdf>. Acesso em: 9 jul. 2014.

FEAMSTER, N. et al. The Road to SDN. **Queue**, v. 11, n. 12, p. 20:20–20:40. dez. 2013. DOI: 10.1145/2559899.2560327

_____. The Case for Separating Routing from Routers, New York, NY, USA, 2004. **Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture**.ACM. p. 5–12. DOI: 10.1145/1016707.1016709

_____. **SDN: Overview of SDN Controllers**. jun. 2014. Disponível em: <https://d396qusza40orc.cloudfront.net/sdn/slides/M3.2-2014.pdf>. Acesso em: 15 jun. 2014.

FEAMSTER, N. e MCKEOWN, N. **Interview with Nick McKeown - YouTube**. 31 jul. 2013. Disponível em: <https://www.youtube.com/watch?v=abXezfJsqso>. Acesso em: 10 jul. 2014.

FOSTER, N.; GUHA, A.; REITBLATT, M. et al. **Languages for software-defined networks**. IEEE Communications Magazine, v. 51, n. 2, p. 128–134. fev. 2013. DOI: 10.1109/MCOM.2013.6461197

FRENETIC-LANG.ORG **The Frenetic Project**. 2014. Disponível em: <http://frenetic-lang.org/overview.php>. Acesso em: 9 jul. 2014.

GREDLER, H. et al. **North-Bound Distribution of Link-State and TE Information using BGP**. 18 nov. 2013. Disponível em: <http://tools.ietf.org/html/draft-ietf-idr-ls-distribution-04>. Acesso em: 11 mai. 2014.

HALEPLIDIS, E. et al. Software-Defined Networking: Experimenting with the Control to Forwarding Plane Interface: . In: EUROPEAN WORKSHOP ON SOFTWARE DEFINED NETWORKING, 2012, Darmstadt. **Atas da conferência.** Darmstadt: IEEE Computer Society Conference Publishing Services (CPS), 2012. p. 91-96. DOI: 10.1109/EWSDN.2012.17

HARES, S. et al. **A Border Gateway Protocol 4 (BGP-4)**. jan. 2006. Disponível em: <http://tools.ietf.org/html/rfc4271>. Acesso em: 11 mai. 2014.

HINRICHS, T. L. et al. Practical Declarative Network Management, New York, NY, USA, 2009. **Proceedings of the 1st ACM Workshop on Research on Enterprise Networking**.ACM. p. 1–10. DOI: 10.1145/1592681.1592683

HOCK, D. et al. Pareto-optimal resilient controller placement in SDN-based core networks. In: TELETRAFFIC CONGRESS (ITC), 2013 25TH INTERNATIONAL, 2013.

Teletraffic Congress (ITC), 2013 25th International. p. 1–9. DOI: 10.1109/ITC.2013.6662939

IETF ForCES WG. **Forwarding and Control Element Separation**: Charter for Working Group. [2014]. Disponível em: <http://datatracker.ietf.org/wg/forces/charter/>. Acesso em: 20 abr. 2014.

IETF OSPF WG. **Open Shortest Path First IGP**. 2014. Disponível em: <https://datatracker.ietf.org/wg/ospf/>. Acesso em: 18 mai. 2014.

IETF PCE WG. **Path Computation Element**: Charter for Working Group [2014]. Disponível em: <http://datatracker.ietf.org/wg/pce/charter/>. Acesso em: 20 abr. 2014.

KERNER, S. M. **SDN in 2014: More Adoption and More Money for Vendors**. 2014. Disponível em: <http://www.itbusinessedge.com/slideshows/sdn-in-2014-more-adoption-and-more-money-for-vendors.html>. Acesso em: 9 jul. 2014.

KIRKPATRICK, K. Software-defined networking. **Communications of the ACM**, New York, v. 56, n. 9, p. 16-19, set. 2013. DOI: 10.1145/2500468.2500473

KONG, X. et al. Performance evaluation of software-defined networking with real-life ISP traffic. In: 2013 IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC), 2013. **2013 IEEE Symposium on Computers and Communications (ISCC)**. p. 000541–000547. DOI: 10.1109/ISCC.2013.6755002

LANTZ, B. et al. A Network in a Laptop: Rapid Prototyping for Software-defined Networks, New York, NY, USA, 2010. **Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks**.ACM. p. 19:1–19:6. DOI: 10.1145/1868447.1868466

LIMONCELLI, T. A. OpenFlow: A Radical New Idea in Networking. **Communications of the ACM**, New York, v. 55, n. 8, p. 42-47, ago. 2013. DOI: 10.1145/2240236.2240254

MA, C. **SDN secrets of Amazon and Google**. 7 mai. 2014. Disponível em: <http://www.infoworld.com/t/sdn/sdn-secrets-of-amazon-and-google-242011>. Acesso em: 9 jul. 2014.

MCCAULEY, MURPHY **[pox-dev] implementing vlans**. 6 nov. 2013. Disponível em: <https://www.mail-archive.com/pox-dev@lists.noxrepo.org/msg00840.html>. Acesso em: 1 jun. 2014.

MCKEOWN, N. et al. OpenFlow: Enabling Innovation in Campus Networks. **SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69–74. mar. 2008. DOI: 10.1145/1355734.1355746

MERRITT, R. **Google describes its OpenFlow network** | EE Times. 17 abr. 2012. Disponível em: <http://www.eetimes.com/document.asp?doc_id=1261562>. Acesso em: 14 jun. 2014.

MININET.ORG. **Mininet: An Instant Virtual Network on your Laptop (or other PC)**. 2014. Disponível em: <http://mininet.org/>. Acesso em: 24 mai. 2014.

____. **Get Started with Mininet**. 2014. Disponível em: <http://mininet.org/download/>. Acesso em: 29 mai. 2014.

____. **Simple Router**. 2014. Disponível em: <https://github.com/mininet/mininet>. Acesso em: 7 jun. 2014.

____. **Network Address Translator (NAT)**. 19 mar. 2014. Disponível em: <https://github.com/mininet/mininet>. Acesso em: 11 jun. 2014.

MUNOZ, R. et al. PCE: What is It, How Does It Work and What are Its Limitations? **Journal of Lightwave Technology**, v. 32, n. 4, p. 528–543. Fevereiro. 2014. DOI: 10.1109/JLT.2013.2276911

MUNTANER, Guillermo R. T. **Evaluation of OpenFlow Controllers**. 2012. 77 f. Dissertação (Mestrado) – KTH, School of Information and Communication Technology (ICT), 2012. Disponível em: < http://www.valleytalk.org/wp-content/uploads/2013/02/Evaluation_Of_OF_Controllers.pdf >. Acesso em: 20 abr. 2014, 18:19.

NAGIOS.ORG. **Nagios - The Industry Standard in IT Infrastructure Monitoring**. 2014. Disponível em: <http://www.nagios.org/>. Acesso em: 11 jul. 2014.

NARISETTY, R. et al. OpenFlow Configuration Protocol: Implementation for the of Management Plane. In: RESEARCH AND EDUCATIONAL EXPERIMENT WORKSHOP (GREE), 2013 SECOND GENI, 2013. **Research and Educational Experiment Workshop (GREE), 2013 Second GENI**. p. 66–67. DOI: 10.1109/GREE.2013.21

NAYAK, A. K. et al. Resonance: Dynamic Access Control for Enterprise Networks, New York, NY, USA, 2009. **Proceedings of the 1st ACM Workshop on Research on Enterprise Networking**.ACM. p. 11–18. DOI: 10.1145/1592681.1592684

NOXREPO. **The POX Controller**. 30 mai. 2013. Disponível em: < https://github.com/noxrepo/pox/blob/carp/pox/ >. Acesso em: 1 jun. 2014.

NUNES, B. et al. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. **IEEE Communications Surveys Tutorials**, v. Early Access Online. 2014. DOI: 10.1109/SURV.2014.012214.00180

ONF. **Software-Defined Networking**: The New Norm for Networks. 13 abr. 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>. Acesso em: 17 out. 2013.

____. **SDN Architecture Overview**. 12 dez. 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf>. Acesso em: 5 abr. 2014.

____. **OpenFlow Management and Configuration Protocol**: OF-Config 1.1.1. 23 mar. 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1-1-1.pdf>. Acesso em: 20 abr. 2014.

____. **OpenFlow Switch Specification**. 27 mar. 2014. Disponível em: <https://www.opennetworking.org/sdn-resources/onf-specifications/openflow>. Acesso em: 20 abr. 2014.

____. **Migration Use Cases**: Migration Use Cases. [2014]. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/use-cases/Migration-WG-Use-Cases.pdf>. Acesso em: 5 abr. 2014.

OPENDAYLIGHT.ORG. **Technical Overview | OpenDaylight**. [2014]. Disponível em: <http://www.opendaylight.org/project/technical-overview>. Acesso em: 9 jul. 2014.

____. **Members | OpenDaylight**. [2014]. Disponível em: <http://www.opendaylight.org/project/members>. Acesso em: 9 jul. 2014.

OPENSTACK.ORG. **OpenStack Networking**: Neutron. [2014]. Disponível em: <https://wiki.openstack.org/wiki/Neutron>. Acesso em: 5 abr. 2014.

____. **User Stories:** OpenStack Open Source Cloud Computing Software. [2014]. Disponível em: <http://www.openstack.org/user-stories/>. Acesso em: 5 abr. 2014.

OPENVSWITCH.ORG. **Open vSwitch FAQ**. 27 jun. 2014. Disponível em: <https://raw.githubusercontent.com/openvswitch/ovs/master/FAQ>. Acesso em: 8 jul. 2014.

PFAFF, B. e DAVIE, B. **The Open vSwitch Database Management Protocol**. dez. 2013. Disponível em: <https://tools.ietf.org/html/rfc7047>. Acesso em: 11 mai. 2014.

SDNCENTRAL.COM. **OpenStack Networking**: What is Quantum and Neutron? [S.d.]. Disponível em: <http://www.sdncentral.com/what-is-openstack-quantum-neutron/>. Acesso em: 5 abr. 2014.

SEEDORF, J. e BURGER, E. **Application-Layer Traffic Optimization (ALTO) Problem Statement**. out. 2009. Disponível em: <http://tools.ietf.org/html/rfc5693>. Acesso em: 11 mai. 2014.

SEZER, S. et al. Are We Ready for SDN? Implementation Challenges for Software-Defined Networks. **IEEE Communications Magazine**, v. 51, n. 7, p. 36-43, jul. 2013. DOI: 10.1109/MCOM.2013.6553676

SGAMBELLURI, A. et al. OpenFlow-based segment protection in Ethernet networks. **IEEE/OSA Journal of Optical Communications and Networking**, v. 5, n. 9, p. 1066–1075. set. 2013. DOI: 10.1364/JOCN.5.001066

SHAH, S. A. et al. An architectural evaluation of SDN controllers. In: 2013 IEEE INTERNATIONAL CONFERENCE ON COMMUNICATIONS (ICC), 2013. **2013 IEEE International Conference on Communications (ICC)**. p. 3504–3508. DOI: 10.1109/ICC.2013.6655093

SHARMA, S. et al. Enabling fast failure recovery in OpenFlow networks. In: DESIGN OF RELIABLE COMMUNICATION NETWORKS (DRCN), 2011 8TH INTERNATIONAL WORKSHOP ON THE, 2011. **Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the**. p. 164–171. DOI: 10.1109/DRCN.2011.6076899

SHERWOOD, R. et al. Carving Research Slices out of Your Production Networks with OpenFlow. **SIGCOMM Comput. Commun. Rev.**, v. 40, n. 1, p. 129–130. jan. 2010. DOI: 10.1145/1672308.1672333

SMITH, M. et al. IETF Internet-Draft: **OpFlex Control Protocol**. 2 abr. 2014. Disponível em: <http://tools.ietf.org/html/draft-smith-opflex-00>. Acesso em: 4 mai. 2014.

SRISURESH, P. et al. **NAT Behavioral Requirements for TCP**. 2008. Disponível em: <http://tools.ietf.org/html/rfc5382>. Acesso em: 11 jun. 2014.

SRISURESH, P. et al. **NAT Behavioral Requirements for ICMP**. 2009. Disponível em: <http://tools.ietf.org/html/rfc5508>. Acesso em: 11 jun. 2014.

TANENBAUM, A. S., WETHERALL, D. J. **Computer networks**. 5. Ed. New Jersey, Upper Saddle River: Prentice-Hall, 27 set. 2010

TELESINTESE.COM.BR. **Cpqd e Datacom Fecham Parceria para Desenvolver Equipamento para Rede IP Com Openflow**. 24 jan. 2013. Disponível em: http://www.telesintese.com.br/cpqd-e-datacom-fecham-parceria-para-desenvolver-equipamento-para-rede-ip-com-openflow/. Acesso em: 28 mai. 2014.

UNDERHILL, D. **Length-Type-Based Protocol Client/Server**. 31 mai. 2009. Disponível em: <http://dound.com/projects/ltprotocol/>. Acesso em: 7 jun. 2014.

VOELLMY, A. et al. Procera: A Language for High-level Reactive Network Control, New York, NY, USA, 2012. **Proceedings of the First Workshop on Hot Topics in Software Defined Networks**. ACM. p. 43–48. DOI: 10.1145/2342441.2342451

WRIGHT, C. et al. **OpenDaylight: An Open Source SDN for Your OpenStack Cloud**. 2013. Disponível em: <https://www.openstack.org/summit/openstack-summit-hong-kong-2013/session-videos/presentation/opendaylight-an-open-source-sdn-for-your-openstack-cloud>. Acesso em: 9 jul. 2014.

ZABBIX.COM. **Homepage of Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution**. 2014. Disponível em: <http://www.zabbix.com/>. Acesso em: 11 jul. 2014.

GLOSSÁRIO

POX	Python NOX, controlador OpenFlow escrito em Python e baseado no controlador NOX.
NOX	Nome dado a um dos primeiros controladores OpenFlow, que foi inicialmente considerado um Network Operating System, ou sistema operacional de redes. Escrito em linguagem C++.
OpenFlow	Padrão de protocolo e arquitetura de rede utilizado por plataformas de controle para controlar o plano de tráfego de dados em switches e roteadores.
FlowVisor	Controlador de uso específico, utilizado para particionar redes OpenFlow entre diversos controladores distintos.
Pyretic	Python + Frenetic, linguagem declarativa de alto nível para controle de SDN.
PyResonance	Pyretic + Resonance, plataforma de controle SDN baseada em Pyretic possibilita controlar a rede com base na ocorrência de eventos e manter máquinas de estado para tomada de decisões de controle.
Procera	Linguagem declarativa de alto nível para controle reativo de redes SDN.
Frenetic	Família de linguagens de alto nível para controle de redes SDN.
Resonance	Plataforma de controle SDN que possibilita controlar a rede com base na ocorrência de eventos e manter máquinas de estado para tomada de decisões de controle.
YANG	YANG linguagem para modelagem de dados (data modeling language), utilizada na especificação do OF-Config.
Ethane	Arquitetura de controle para redes corporativas. Precursor do OpenFlow.
Open vSwitch	Open Virtual Switch, switch virtual utilizado para executar funções de rede virtualizadas, comutação. Utilizado também pelo Mininet como switch padrão para emulação de redes. Suporta OpenFlow.
Maestro	Controlador SDN escrito em Java.
Beacon	Controlador SDN escrito em Java.
Floodlight	Controlador SDN escrito em Java.
OpenDaylight	Plataforma SDN completa, incluindo APIs Norte, Sul e plataforma de controle.
Big Network Controller	Controlador SDN comercial baseado no Floodlight.
Mininet	Emulador de redes virtuais que utiliza network namespaces e Open vSwitch para virtualização de redes.

RouteFlow	Roteador baseado em OpenFlow, projeto do CPqD.

APÊNDICE A

Este apêndice contém as instruções de instalação do emulador Mininet e do controlador de rede POX em Ubuntu 14.04 versão desktop Xubuntu. Também contém as topologias e programas utilizados nas demonstrações realizadas com Mininet e POX.

INSTALAÇÃO DO MININET E POX

A instalação do mininet pode ser feita utilizando qualquer uma das opções dadas em mininet.org. Para este trabalho escolheu-se utilizar os repositórios contendo o código mais recente do mininet e o script de instalação do mesmo, que automaticamente faz download do OpenFlow, POX e outros pacotes opcionais.

O primeiro passo é instalar no sistema pacotes essenciais, como Python, Wireshark e git:

```
$ sudo apt-get install git git-core python wireshark
```

Recomenda-se também desinstalar agentes de configuração automática de rede, que podem poluir os logs das interfaces com mensagens utilizadas na configuração automática de rede:

```
$ sudo apt-get remove network-manager avahi-daemon
```

Para configuração de rede recomenda-se utilizar configurações estáticas, através do arquivo /etc/network/interfaces. Exemplo:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet manual

auto eth2
iface eth2 inet static
    address 10.99.0.2
    netmask 255.255.255.0
```

Em seguida, no diretório *home* do usuário, fazer o git clone do repositório do mininet:

```
$ git clone git://github.com/mininet/mininet
Cloning into 'mininet'...
remote: Reusing existing pack: 5746, done.
remote: Counting objects: 18, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 5764 (delta 9), reused 8 (delta 5)
Receiving objects: 100% (5764/5764), 2.07 MiB | 806.00 KiB/s, done.
Resolving deltas: 100% (3310/3310), done.
Checking connectivity... done.
```

Executar o instalador do mininet com as seguintes opções:

```
sudo mininet/util/install.sh –nfvpw
```

- -n para instalar o mininet e suas dependências;
- -f para instalar OpenFlow;
- -v para instalar o Open vSwitch;
- -p para instalar o POX OpenFlow Controller;
- -w para instalar o dissector de OpenFlow para Wireshark

O processo de instalação pode tomar algum tempo e exibe grande quantidade de texto. Para verificar se a instalação do Mininet e POX estão funcionando adequadamente pode-se executar o procedimento abaixo para iniciar o POX em um terminal:

```
$ cd ~/pox && ./pox.py log.level --DEBUG forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-
trusty
INFO:core:POX 0.2.0 (carp) is up.
```

E em outro terminal iniciar o mininet

```
$ sudo mn --controller=remote --test=pingall
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
```

```
*** Starting 1 switches
s1
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
*** Stopping 1 switches
s1 ..
*** Stopping 2 hosts
h1 h2
*** Stopping 1 controllers
c0
*** Done
completed in 2.049 seconds
$
```

O controlador POX deve emitir alguns avisos de instalação de *flows* nos switches controlados, pois está rodando o aplicativo “l2_learning”:

```
$ cd ~/pox && ./pox.py log.level --DEBUG forwarding.l2_learning
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/Mar 22 2014 22:59:56)
DEBUG:core:Platform is Linux-3.13.0-24-generic-x86_64-with-Ubuntu-14.04-
trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
DEBUG:forwarding.l2_learning:installing flow for b2:df:fc:c5:87:93.2 ->
56:e0:e2:af:b6:7f.1
DEBUG:forwarding.l2_learning:installing flow for 56:e0:e2:af:b6:7f.1 ->
b2:df:fc:c5:87:93.2
DEBUG:forwarding.l2_learning:installing flow for b2:df:fc:c5:87:93.2 ->
56:e0:e2:af:b6:7f.1
DEBUG:forwarding.l2_learning:installing flow for b2:df:fc:c5:87:93.2 ->
56:e0:e2:af:b6:7f.1
DEBUG:forwarding.l2_learning:installing flow for 56:e0:e2:af:b6:7f.1 ->
b2:df:fc:c5:87:93.2
INFO:openflow.of_01:[00-00-00-00-00-01 2] closed
```

TOPOLOGIA DE TESTE L2_DEMO.PY

Esta topologia é demonstrada pela Figura 12. Para cria-la em Mininet é utilizado o seguinte código:

```
#!/usr/bin/python

"""
Topologia para demonstracao de funcionalidades basicas L2 e L3 com
controlador remoto POX
"""
```

```
from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel

def RemoteControllerNet():
    "Cria uma rede configurada manualmente com controlador remoto."

    net = Mininet( controller=RemoteController, switch=OVSSwitch,
build=False )

    print "*** Criando switches"
    SW1 = net.addSwitch( 'SW1' )
    SW2 = net.addSwitch( 'SW2' )

    print "*** Criando hosts e enlaces para os switches"
    hosts = {}
    for oct3 in range(10,31,10):
        for host in range(1,2+1):
            name = "h%d_%d" % (oct3,host)
            hosts[name] = net.addHost( name,
                          ip='192.168.%d.%d/24' % (oct3, host),
                          mac='02:00:00:00:%d:%d' % (oct3, host))
            if oct3 == 10:
                net.addLink(hosts[name], SW1)
            elif oct3 == 20:
                net.addLink(hosts[name], SW2)
            elif oct3 == 30 and host == 1:
                net.addLink(hosts[name], SW2)
            elif oct3 == 30 and host == 2:
                net.addLink(hosts[name], SW1)

    print "*** Criando enlace entre os switches"

    net.addLink( SW1, SW2 )

    print "*** Adicionando controlador remoto POX"
    c0 = RemoteController('POX', ip='127.0.0.1', port=6633)

    print "*** Construindo e iniciando os elementos da rede"
    net.build()
    SW1.start([c0])
    SW2.start([c0])

    print "*** Configurando default GWs em cada host"
    for oct3 in range(10,31,10):
        for host in range(1,2+1):
            name = "h%d_%d" % (oct3,host)
            hosts[name].cmd('route add default gw 192.168.%d.254' % oct3)


    print "*** Iniciando CLI"
    CLI( net )

    print "*** Parando a rede..."
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    RemoteControllerNet()
```

TOPOLOGIA DE TESTE L2_L3_DEMO_CS144_ROUTING.PY

Esta topologia é demonstrada pela Figura 13. Para cria-la em Mininet é utilizado o seguinte código:

```
#!/usr/bin/python

"""
Topologia para demonstracao de funcionalidades basicas L2 e L3 com
controlador remoto POX
"""

from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.util import dumpNodeConnections


def RemoteControllerNet():
    "Cria uma rede configurada manualmente com controlador remoto."

    net = Mininet( controller=RemoteController, switch=OVSSwitch,
build=False )

    print "*** Criando switches"
    SW1 = net.addSwitch( 'SW1' )
    SW2 = net.addSwitch( 'SW2' )
    SW3 = net.addSwitch( 'SW3' )
    SW4 = net.addSwitch( 'SW4' )

    print "*** Criando hosts e enlaces para os switches"
    hosts = {}
    for oct3 in range(10,31,10):
        for host in range(1,2+1):
            name = "h%d_%d" % (oct3,host)
            hosts[name] = net.addHost( name,
                          ip='192.168.%d.%d/24' % (oct3, host),
                          mac='02:00:00:00:%d:%d' % (oct3, host))
            if oct3 == 10:
                net.addLink(hosts[name], SW1)
            elif oct3 == 20:
                net.addLink(hosts[name], SW2)
            elif oct3 == 30:
                net.addLink(hosts[name], SW3)

    print "*** Criando enlace entre os switches"

    net.addLink( SW4, SW1 )
    net.addLink( SW4, SW2 )
    net.addLink( SW4, SW3 )

    print "*** Adicionando controlador remoto POX"
    c0 = RemoteController('POX', ip='127.0.0.1', port=6633)
    c1 = RemoteController('POX', ip='127.0.0.1', port=9000)

    print "*** Construindo e iniciando os elementos da rede"
    net.build()
```

```
    SW1.start([c1])
    SW2.start([c1])
    SW3.start([c1])
    SW4.start([c0])

    print "*** Configurando default GWs em cada host"
    for oct3 in range(10,31,10):
        for host in range(1,2+1):
            name = "h%d_%d" % (oct3,host)
            hosts[name].cmd('route add default gw 192.168.%d.254' % oct3)


    print "*** Imprimindo tabela de conexoes:"
    dumpNodeConnections(net.switches)

    print "*** Iniciando CLI"
    CLI( net )

    print "*** Parando a rede..."
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    RemoteControllerNet()
```

Para iniciar o roteamento nessa topologia é necessário seguir os procedimentos em mininet.org (2014c). Os arquivos originais baixados do site precisam ser modificados para que o roteamento funcione. A tabela de roteamento rtable utilizada é:

```
192.168.10.1 192.168.10.1 255.255.255.255 eth1
192.168.10.2 192.168.10.2 255.255.255.255 eth1
192.168.20.1 192.168.20.1 255.255.255.255 eth2
192.168.20.2 192.168.20.2 255.255.255.255 eth2
192.168.30.1 192.168.30.1 255.255.255.255 eth3
192.168.30.2 192.168.30.2 255.255.255.255 eth3
```

A tabela de endereços IP da topologia é:

```
h10_1 192.168.10.1
h10_2 192.168.10.2
h20_1 192.168.20.1
h20_2 192.168.20.2
h30_1 192.168.30.1
h30_2 192.168.30.2
SW4-eth1 192.168.10.254
SW4-eth2 192.168.20.254
SW4-eth3 192.168.30.254
```

Demais arquivos retirados de mininet.org (2014c) foram modificados e armazenados em https://github.com/fsrechia/mnlab/tree/master/pox_module/cs144, pois são muito extensos para serem apresentados no apêndice. Os arquivos desse repositório são atualizados e apresentam também as modificações para realizar NAT.

É importante observar que é necessário executar cada programa separadamente para que este cenário de testes funcione. Quatro terminais diferentes precisam ser utilizados para iniciar as aplicações necessárias para esta simulação.

TOPOLOGIA DE TESTE L2_L3_DEMO_PORT_SECURITY.PY

A topologia em Mininet para o teste do limite de MACs está abaixo (arquivo L2_L3_demo_port_security.py):

```
#!/usr/bin/python

"""
Topologia para demonstracao de funcionalidades basicas L2 e L3 com
controlador remoto POX
"""

from mininet.net import Mininet
from mininet.node import RemoteController, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel
from mininet.util import dumpNodeConnections


def RemoteControllerNet():
    "Cria uma rede configurada manualmente com controlador remoto."

    net = Mininet( controller=RemoteController, switch=OVSSwitch,
build=False )

    print "*** Criando switches"
    SW1 = net.addSwitch( 'SW1' )
    SW2 = net.addSwitch( 'SW2' )
    SW3 = net.addSwitch( 'SW3' )
    SW4 = net.addSwitch( 'SW4' )
    SW5 = net.addSwitch( 'SW5' )

    print "*** Criando hosts e enlaces para os switches"
    hosts = {}
    for oct3 in range(10,31,10):
        for host in range(1,2+1):
            name = "h%d_%d" % (oct3,host)
            hosts[name] = net.addHost( name,
                          ip='192.168.%d.%d/24' % (oct3, host),
                          mac='02:00:00:00:%d:%d' % (oct3, host))
            if oct3 == 10:
                net.addLink(hosts[name], SW1)
            elif oct3 == 20:
                net.addLink(hosts[name], SW2)
            elif oct3 == 30:
                net.addLink(hosts[name], SW3)

    print "*** Criando enlaces entre os switches"

    net.addLink( SW4, SW1 )
    net.addLink( SW4, SW2 )
```

```
    net.addLink( SW4, SW3 )

    h20_3 = net.addHost(
'h20_3',ip='192.168.20.3/24',mac='02:00:00:00:20:03')
    h20_4 = net.addHost(
'h20_4',ip='192.168.20.4/24',mac='02:00:00:00:20:04')
    net.addLink( SW5, h20_3)
    net.addLink( SW5, h20_4)
    net.addLink( SW5, SW2 )

    print "*** Adicionando controlador remoto POX"
    c0 = RemoteController('POX_L2', ip='127.0.0.1', port=6633)
    c1 = RemoteController('POX_SR', ip='127.0.0.1', port=9000)

    print "*** Construindo e iniciando os elementos da rede"
    net.build()
    SW1.start([c1])
    SW2.start([c1])
    SW3.start([c1])
    SW4.start([c0])
    SW5.start([c1])

    print "*** Configurando default GWs em cada host"
    for oct3 in range(10,31,10):
        for host in range(1,2+1):
            name = "h%d_%d" % (oct3,host)
            hosts[name].cmd('route add default gw 192.168.%d.254' % oct3)


    print "*** Imprimindo tabela de conexoes:"
    dumpNodeConnections(net.switches)
    print "*** Imprimindo DPIDs dos switches:"
    print "SW1:",SW1.dpid
    print "SW2:",SW2.dpid
    print "SW3:",SW3.dpid
    print "SW4:",SW4.dpid
    print "SW5:",SW5.dpid

    print "*** Iniciando CLI"
    CLI( net )

    print "*** Parando a rede..."
    net.stop()

if __name__ == '__main__':
    setLogLevel( 'info' )  # for CLI output
    RemoteControllerNet()
```

LIMITAÇÃO DE ENDEREÇOS MAC POR PORTA

E aplicação para POX que realiza a função de limitação de endereços MAC por porta, port-sec.py, é colocada abaixo. Esta aplicação apenas intercepta os pacotes que chegam no switch, mantendo uma tabela de MACs confiáveis, que são os primeiros MACs a se conectarem ao switch. O programa decide se os pacotes

recebidos pelo switch devem ser encaminhados para a funcionalidade forwarding.l2_pairs para comutação normal ou se os mesmos serão bloqueados através de regra de bloqueio de endereço MAC de origem:

```
"""
An application that limits the number of mac addresses learned per port.
Similar to features usually called MAC Limiting or Port Security.

Hardcoded to use forwarding.l2_pairs as the learning mechanism.

Author: Felipe Stall Rechia
"""

# These next two imports are common POX convention
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr
from collections import namedtuple
import pox.forwarding.l2_pairs as l2p
import os


# Even a simple usage of the logger is much nicer than print!
log = core.getLogger()
class PortSecurity(EventMixin):

    def __init__ (self,max_macs=1):
        self.listenTo(core.openflow)
        self.max_macs = max_macs
        self.table = {}
        log.info("Enabling Port Security Module")
        log.info("Maximum MAC Addresses per port: %d" % self.max_macs)

    def _handle_ConnectionUp (self, event):
        log.info("Creating port-sec table for DPID %s",
                 dpidToStr(event.dpid))
        self.table[event.dpid] = {}
        # check all the switch ports and initialize MAC address
        # table to zero
        for port in event.ofp.ports:
            log.debug("Init port number: %d, port name: %s"
                      % (port.port_no,port.name))
            self.table[event.dpid][port.port_no] = []


    def _handle_PacketIn (self, event):
        packet = event.parsed
        if packet.src in self.table[event.dpid][event.port]:
            log.debug("[DPID %d, P: %d] MAC %s Already Learned"
                      % (event.dpid,event.port,packet.src))
            l2p._handle_PacketIn(event)
        elif len(self.table[event.dpid][event.port]) < self.max_macs:
            log.debug("[DPID %d, P: %d] MAC %s Adding as Trusted"
                      % (event.dpid,event.port,packet.src))
            self.table[event.dpid][event.port].append(packet.src)
            l2p._handle_PacketIn(event)
        else:
```

```
            log.debug("[DPID %d, P: %d] MAC %s Refusing"
                      % (event.dpid,event.port,packet.src))
            block_rule = of.ofp_flow_mod()
            block_rule.match.dl_src = packet.src
            block_rule.match.port = event.port
            block_rule.priority = 65535
            event.connection.send(block_rule.pack())

def launch ():
    core.registerNew(PortSecurity)
```

Tanto a topologia em Mininet quanto o programa em POX estão também disponíveis no repositório https://github.com/fsrechia/mnlab.

LISTA DE CONTROLE DE ACESSO

Para realizar a função de lista de controle de acesso foi criado o programa a seguir, para POX. Também está disponível online em https://github.com/fsrechia/mnlab:

```
"""
An application which consults a CSV file and installs rules to
block traffic between pairs of hosts.

Author: Felipe Stall Rechia
"""

from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.revent import *
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr, IPAddr
from collections import namedtuple
import os
import csv

log = core.getLogger()
policyFile = "%s/pox/pox/security/acl.csv" % os.environ[ 'HOME' ]

class ACL (EventMixin):

    def __init__ (self):
        self.listenTo(core.openflow)
        log.debug("Enabling ACL Module")
        self.policies = {}
        log.debug("Reading policy file")
        with open(policyFile,'r') as csvfile:
            next(csvfile)
            policies = csv.reader(csvfile)
            for row in policies:
                log.debug("Rule id %s forbids src:%s dst:%s type:%s"
                          % (row[0],row[1],row[2],row[3]))
```

```
                self.policies[row[0]] = {'addr0':row[1],
                                         'addr1':row[2],
                                         'type':row[3]}
                #print self.policies

    def _handle_ConnectionUp (self, event):
        ''' Add your logic here ... '''
        log.debug("New switch detected, installing ACL rules...")
        for id,rule in self.policies.iteritems():
            log.debug('ACL Rule %s blocks %s<>%s (%s)'
                      % (id,rule['addr0'],rule['addr1'],rule['type']))
            if rule['type']== "L2":
                # create rule addr0 -> addr1
                msg1 = of.ofp_flow_mod()
                msg1.match.dl_src = EthAddr(rule['addr0'])
                msg1.match.dl_dst = EthAddr(rule['addr1'])
                msg1.priority = 65535
                # create rule addr1 -> addr0
                msg2 = of.ofp_flow_mod()
                msg2.match.dl_src = EthAddr(rule['addr1'])
                msg2.match.dl_dst = EthAddr(rule['addr0'])
                msg1.priority = 65535
                # send both rules
                event.connection.send(msg1.pack())
                event.connection.send(msg2.pack())
            elif rule['type']== "L3_IP":
                # create rule addr0 -> addr1
                msg1 = of.ofp_flow_mod()
                msg1.match.dl_type = 0x0800 # Ethertype IP
                msg1.match.nw_src = IPAddr(rule['addr0'])
                msg1.match.nw_dst = IPAddr(rule['addr1'])
                msg1.priority = 65535
                # create rule addr1 -> addr0
                msg2 = of.ofp_flow_mod()
                msg2.match.dl_type = 0x0800 # Ethertype IP
                msg2.match.nw_src = IPAddr(rule['addr1'])
                msg2.match.nw_dst = IPAddr(rule['addr0'])
                msg2.priority = 65535
                # send both rules
                event.connection.send(msg1.pack())
                event.connection.send(msg2.pack())

        log.debug("ACL rules installed on %s", dpidToStr(event.dpid))

def launch ():
    core.registerNew(ACL)
```

APÊNDICE B

Este apêndice contém as configurações dos dispositivos de rede da Figura 20, utilizadas para prova de conceito do caso de uso de rede corporativa. As configurações utilizadas e o arquivo de *packet tracer* estão disponíveis online em https://github.com/fsrechia/mnlab/tree/master/pkt_tracer.

CONFIGURAÇÃO DO ELEMENTO SW5

O elemento SW5 compõe um anel Spanning-tree juntamente com os switches SW1 e SW4, e serve para prestar acesso à rede para o PC7 com a VLAN 30. Suas interfaces de uplink estão prontas para tráfego das VLANs 1, 10, 20 e 30, caso necessário.

```
!
hostname SW5
!
no ip domain-lookup
!
spanning-tree mode rapid-pvst
!
interface FastEthernet0/1
 switchport trunk allowed vlan 1,10,20,30
 switchport mode trunk
!
interface FastEthernet0/2
 switchport trunk allowed vlan 1,10,20,30
 switchport mode trunk
!
interface FastEthernet0/3
 switchport access vlan 30
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
! Configurações para permitir gerência apenas de 192.168.50.0/24
! com usuario/senha = “usuario/senha”
!
enable password senha
!
username usuario privilege 15 secret 5 $1$mERr$62fSvvBAj69RpPvTBk3e8.
!
access-list 10 permit 192.168.50.0 0.0.0.255
!
line vty 0 4
 login local
 transport input telnet
 access-class 10 in
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
```

```
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO ELEMENTO SW4

O elemento SW4 compõe um anel Spanning-tree juntamente com os switches SW1 e SW5, e serve para prestar acesso à rede para o PC6 com a VLAN 20. Suas interfaces de uplink estão prontas para tráfego das VLANs 1, 10, 20 e 30, caso necessário.

```
!
hostname SW4
!
no ip domain-lookup
!
spanning-tree mode rapid-pvst
spanning-tree vlan 1,10,20,30 priority 28672
!
interface FastEthernet0/1
 switchport trunk allowed vlan 1,10,20,30
 switchport mode trunk
!
interface FastEthernet0/2
 switchport trunk allowed vlan 1,10,20,30
 switchport mode trunk
!
interface FastEthernet0/3
 switchport access vlan 20
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
! Configurações para permitir gerência apenas de 192.168.50.0/24
! com usuario/senha = "usuario/senha"
!
enable password senha
!
username usuario privilege 15 secret 5 $1$mERr$62fSvvBAj69RpPvTBk3e8.
!
access-list 10 permit 192.168.50.0 0.0.0.255
!
line vty 0 4
 login local
 transport input telnet
 access-class 10 in
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO ELEMENTO SW1

O elemento SW1 compõe um anel Spanning-tree juntamente com os switches SW4 e SW5, e serve para prestar acesso à rede para o PC1 com a VLAN 10. Suas interfaces de uplink estão prontas para tráfego das VLANs 1, 10, 20 e 30, e possui um enlace com redundância por agregação de dois enlaces FastEthernet (port-channel) conectados ao MLSW. Possui um enlace com o Hub0 para demonstrar o funcionamento do Port-security, funcionalidade que bloqueia o enlace quando detecta mais de um endereço MAC (PC8 e PC9), de acordo com a configuração.

```
!
hostname SW1
!
!
!
no ip domain-lookup
!
spanning-tree mode rapid-pvst
spanning-tree vlan 1,10,20,30 priority 24576
!
interface FastEthernet0/1
 switchport access vlan 10
 switchport trunk allowed vlan 1,10,20,30
 switchport mode trunk
 spanning-tree portfast
!
interface FastEthernet0/2
 switchport access vlan 10
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/3
 channel-group 1 mode on
 switchport mode trunk
!
interface FastEthernet0/4
 channel-group 1 mode on
 switchport mode trunk
!
interface FastEthernet0/5
 switchport trunk allowed vlan 1,10,20,30
 switchport mode trunk
!
interface FastEthernet0/6
 switchport access vlan 10
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface Port-channel 1
```

```
 switchport mode trunk
!
! Configurações para permitir gerência apenas de 192.168.50.0/24
! com usuario/senha = "usuario/senha"
!
enable password senha
!
username usuario privilege 15 secret 5 $1$mERr$62fSvvBAj69RpPvTBk3e8.
!
access-list 10 permit 192.168.50.0 0.0.0.255
!
line vty 0 4
 login local
 transport input telnet
 access-class 10 in
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO ELEMENTO MLSW

O elemento de rede MLSW é um switch multicamada (*multilayer switch*) e realiza o roteamento entre todas as redes distribuídas das VLANs 10, 20, 30, 40 e 50. Também disponibiliza acesso à Internet através de um enlace com o núcleo da rede corporativa – representado pelo roteador ISP – e realizado através de rota *default*. Desempenha a função de NAT, escondendo os endereços da rede privada ao enviá-los para a Internet, e também faz o serviço de DHCP Relay para toda a rede (ip helper-address).

```
hostname MLSW
!
ip routing
!
no ip domain-lookup
!
spanning-tree mode pvst
spanning-tree vlan 10,20,30,40,50 priority 0
!
interface FastEthernet0/1
 channel-group 1 mode on
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface FastEthernet0/2
 channel-group 1 mode on
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface FastEthernet0/3
 no switchport
```

```
 ip address 200.200.200.2 255.255.255.252
 ip nat outside
 duplex auto
 speed auto
!
interface FastEthernet0/5
 channel-group 3 mode on
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface FastEthernet0/6
 channel-group 3 mode on
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface GigabitEthernet0/1
 channel-group 2 mode on
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface GigabitEthernet0/2
 channel-group 2 mode on
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface Port-channel 1
 switchport trunk allowed vlan 1,10,20,30
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface Port-channel 2
 switchport trunk allowed vlan 1,40
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface Port-channel 3
 switchport trunk allowed vlan 1,50
 switchport trunk encapsulation dot1q
 switchport mode trunk
!
interface Vlan1
 no ip address
 shutdown
!
interface Vlan10
 ip address 192.168.10.254 255.255.255.0
 ip helper-address 192.168.40.1
 ip nat inside
!
interface Vlan20
 ip address 192.168.20.254 255.255.255.0
 ip helper-address 192.168.40.1
 ip nat inside
!
interface Vlan30
 ip address 192.168.30.254 255.255.255.0
 ip helper-address 192.168.40.1
 ip nat inside
!
interface Vlan40
 ip address 192.168.40.254 255.255.255.0
```

```
 ip nat inside
!
interface Vlan50
 ip address 192.168.50.254 255.255.255.0
 ip helper-address 192.168.40.1
 ip nat inside
!
ip nat pool INTERNET 200.200.200.2 200.200.200.2 netmask 255.255.255.252
ip nat inside source list 1 pool INTERNET overload
ip classless
ip route 0.0.0.0 0.0.0.0 FastEthernet0/3
!
!
access-list 1 permit 192.168.0.0 0.0.255.255
!
! Configurações para permitir gerência apenas de 192.168.50.0/24
! com usuario/senha = "usuario/senha"
!
enable password senha
!
username usuario privilege 15 secret 5 $1$mERr$62fSvvBAj69RpPvTBk3e8.
!
access-list 10 permit 192.168.50.0 0.0.0.255
!
line vty 0 4
 login local
 transport input telnet
 access-class 10 in
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO ELEMENTO ISP

O elemento ISP representa o provedor de acesso à Internet. Possui um enlace com o MLSW e apenas foi configurado para simular alguns endereços da internet, como por exemplo google.com, humblebundle.com e utfpr.edu.br.

```
hostname ISP
!
license udi pid CISCO1941/K9 sn FTX1524L03I
!
no ip domain-lookup
!
no spanning-tree vlan 1
spanning-tree mode pvst
!
interface Loopback1
 description google.com
 ip address 74.125.196.106 255.255.255.255
!
interface Loopback2
```

```
 description humblebundle.com
 ip address 198.41.187.33 255.255.255.255
!
interface Loopback3
 description ufpr.edu.br
 ip address 200.19.73.151 255.255.255.255
!
interface GigabitEthernet0/0
 ip address 200.200.200.1 255.255.255.252
 duplex auto
 speed auto
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO ELEMENTO SW2

O elemento SW2 é responsável pelas conexões com os servidores da rede. Possui um enlace através de port-channel de interfaces GigabitEthernet com o MLSW e leva a VLAN 40 até os servidores.

```
hostname SW2
!
no ip domain-lookup
!
spanning-tree mode rapid-pvst
spanning-tree vlan 1,40 priority 40960
!
interface FastEthernet0/1
 switchport access vlan 40
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/2
 switchport access vlan 40
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/3
 switchport access vlan 40
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface GigabitEthernet1/1
 channel-group 2 mode on
 switchport mode trunk
!
interface GigabitEthernet1/2
```

```
 channel-group 2 mode on
 switchport mode trunk
!
interface Port-channel 2
 switchport trunk allowed vlan 1,40
 switchport mode trunk
!
! Configurações para permitir gerência apenas de 192.168.50.0/24
! com usuario/senha = “usuario/senha”
!
enable password senha
!
username usuario privilege 15 secret 5 $1$mERr$62fSvvBAj69RpPvTBk3e8.
!
access-list 10 permit 192.168.50.0 0.0.0.255
!
line vty 0 4
 login local
 transport input telnet
 access-class 10 in
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO ELEMENTO SW3

O elemento SW3 é responsável por levar a VLAN 50 para os acessos dos PC3 e PC5. Neste elemento foi configurado port-channel FastEthernet conectado ao MLSW.

```
hostname SW3
!
no ip domain-lookup
!
spanning-tree mode rapid-pvst
spanning-tree vlan 1,50 priority 40960
!
interface FastEthernet0/1
 switchport access vlan 50
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/2
 switchport access vlan 50
 switchport mode access
 switchport port-security
 spanning-tree portfast
 spanning-tree bpduguard enable
!
interface FastEthernet0/3
```

```
 channel-group 3 mode on
 switchport mode trunk
!
interface Port-channel 3
 switchport trunk allowed vlan 1,50
 switchport mode trunk
!
! Configurações para permitir gerência apenas de 192.168.50.0/24
! com usuario/senha = "usuario/senha"
!
enable password senha
!
username usuario privilege 15 secret 5 $1$mERr$62fSvvBAj69RpPvTBk3e8.
!
access-list 10 permit 192.168.50.0 0.0.0.255
!
line vty 0 4
 login local
 transport input telnet
 access-class 10 in
!
! omitidas as configurações das interfaces não utilizadas (em shutdown)
! e as configurações padrão de fábrica do switch
!
```

CONFIGURAÇÃO DO SERVIDOR DHCP+DNS

O servidor DHCP+DNS foi criado para facilitar as configurações das interfaces de todos os PCs da rede. A Figura 21 e a Figura 22 demonstram a configuração realizada para DHCP e DNS, sendo o servidor 192.168.40.1 responsável pelos dois serviços.

Para facilitar o teste de ping à Internet foram criados registros DNS estáticos no servidor DNS para humblebundle.com, gooogle.com, e utfpr.edu.br. Esses endereços foram configurados em *loopbacks* do roteador ISP apenas para simular a conectividade com a Internet.

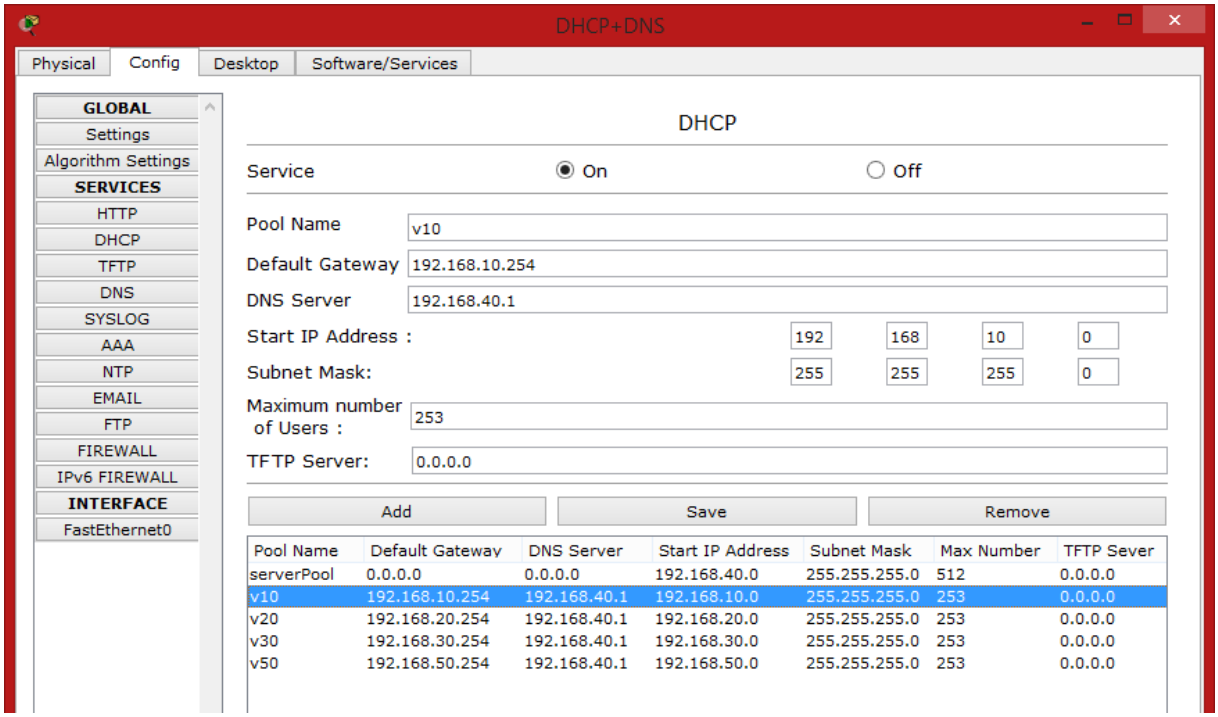

Figura 21 – Configuração do Servidor DHCP no Packet Tracer

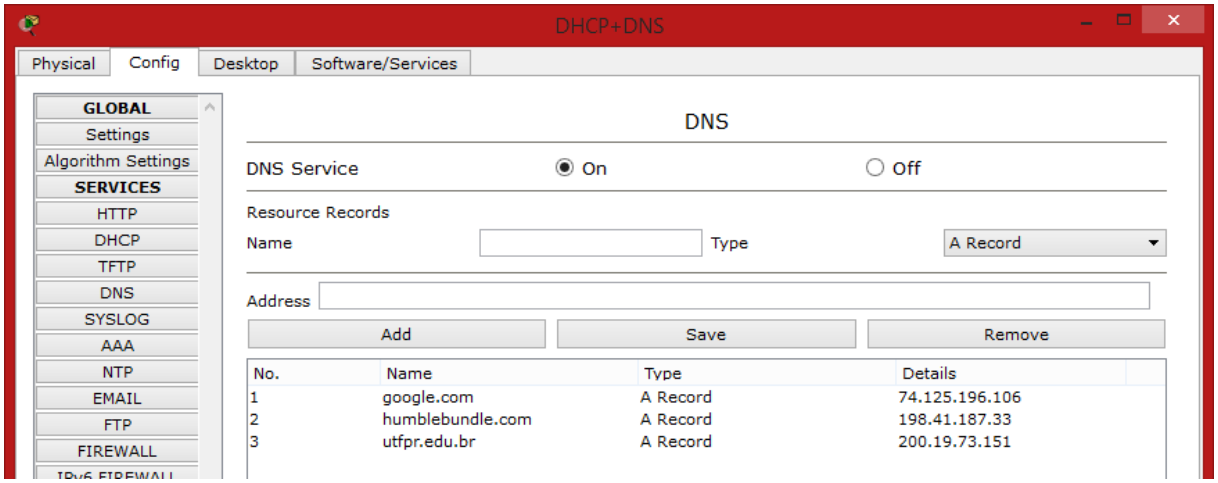

Figura 22 – Configuração do Servidor DNS no Packet Tracer

CONFIGURAÇÃO DOS PCS

Todos os computadores foram configurados para configurar as interfaces de rede automaticamente. O teste básico realizado nos testes de conectividade consiste em pingar alguns elementos da rede e servidores, conforme ilustrado pelo trecho abaixo retirado do “Command Prompt” do PC1.

```
PC>ipconfig /renew
```

```
IP Address.....................: 192.168.10.1
Subnet Mask....................: 255.255.255.0
Default Gateway................: 192.168.10.254
DNS Server.....................: 192.168.40.1

PC>ping humblebundle.com

Pinging 198.41.187.33 with 32 bytes of data:

Request timed out.
Reply from 198.41.187.33: bytes=32 time=0ms TTL=254

Ping statistics for 198.41.187.33:
    Packets: Sent = 2, Received = 1, Lost = 1 (50% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

Control-C
^C
PC>ping 192.168.50.1

Pinging 192.168.50.1 with 32 bytes of data:

Request timed out.
Reply from 192.168.50.1: bytes=32 time=0ms TTL=127

Ping statistics for 192.168.50.1:
    Packets: Sent = 2, Received = 1, Lost = 1 (50% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

Control-C
^C
PC>
PC>ping utfpr.edu.br

Pinging 200.19.73.151 with 32 bytes of data:

Request timed out.
Reply from 200.19.73.151: bytes=32 time=4ms TTL=254
Reply from 200.19.73.151: bytes=32 time=0ms TTL=254

Ping statistics for 200.19.73.151:
    Packets: Sent = 3, Received = 2, Lost = 1 (34% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 4ms, Average = 2ms

Control-C
^C
PC>ping google.com

Pinging 74.125.196.106 with 32 bytes of data:

Request timed out.
Reply from 74.125.196.106: bytes=32 time=0ms TTL=254

Ping statistics for 74.125.196.106:
    Packets: Sent = 2, Received = 1, Lost = 1 (50% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

```
Control-C
^C
PC>
```