André Felipe Klauberg

Isolamento das redes virtuais (VLAN) em uma infraestrutura em nuvem usando uma abordagem SDN/OpenFlow

São José – SC

Agosto / 2016

André Felipe Klauberg

Isolamento das redes virtuais (VLAN) em uma infraestrutura em nuvem usando uma abordagem SDN/OpenFlow

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Marcelo Maia Sobral, Dr.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Agosto / 2016

Monografia sob o título "*Isolamento das redes virtuais (VLAN) em uma infraestrutura em nuvem usando uma abordagem SDN/OpenFlow*", defendida por André Felipe Klauberg e aprovada em 23 de agosto de 2016, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Marcelo Maia Sobral, Dr.
Orientador

Prof. Ederson Torresini, M.Sc.
IFSC

Profa. Simara Sonaglio, Eng.
IFSC

Tudo o que temos de decidir é o que fazer com o tempo que nos é dado.

J. R. R. Tolkien

Resumo

A implantação de redes locais virtuais sobre uma infraestrutura de rede física é uma necessidade comum em projetos de redes de computadores. Isso facilita a organização da rede como um todo, a manutenção e modificação de sua topologia, e a alocação dos recursos físicos (switches e enlaces). A abordagem usual para implantar redes locais virtuais se baseia no padrão IEEE 802.1Q, que usa uma etiqueta inserida em quadros ethernet para identificar a que rede virtual ele pertence. O modelo SDN/OpenFlow oferece a possibilidade de implantar redes locais virtuais a partir de uma visão global da rede, por meio de um controlador centralizado que programa dinamicamente os switches com regras de comutação baseadas nos cabeçalhos de protocolos no cabeçalhos dos quadros recebidos, entre outras possibilidades. Este trabalho propõe a implementação de redes virtuais baseadas na codificação do endereço Ethernet de cada host, substituindo o endereço real da interface de rede por um endereço virtual que contenha a informação da VLAN a qual o dispositivo pertence. O trabalho demonstra o funcionamento do SDN/OpenFlow através de um protótipo criado em uma rede simulada, contendo os elementos convencionais de uma rede real.

Abstract

The implementation of virtual LANs on a physical network infrastructure is a common need for projects of computer networks. This facilitates the organization of the network as a whole, maintenance and modification of its topology, and the allocation of physical resources (switches and links). The usual approach to implant virtual local area networks is based on the IEEE 802.1Q standard, which uses a label in the ethernet frames to identify which virtual network it belongs. The SDN/OpenFlow model offers the ability to implant virtual local area networks from a global view of the network through a centralized controller that program dynamically switches with switching rules based on protocol headers in the headers of the received frames, among other possibilities. This paper proposes the implementation of virtual networks based on coding of the Ethernet address of each host, replacing the actual address of the network interface for a virtual address that contains the information of the VLAN which the device belongs. This paper demonstrates the operation of the SDN/OpenFlow through a prototype created in a simulated network, containing conventional elements of a real network.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

Redes de área local, chamadas de LAN (*Local Area Network*), são redes privadas que permitem a transferência de dados entre os elementos da rede e também o compartilhamento de recursos, como por exemplo, o uso de impressoras, porteiros e câmeras de segurança (TANENBAUM, 2003). LANs são usadas para conectar computadores pessoais, estações de trabalho, servidores e instalações industriais.

A tecnologia dominante atualmente tem origem no padrão IEEE 802.3 [IEEE 802.3, 2012], conhecido popularmente como *Ethernet*. Essas redes implementam enlaces multiponto, o que torna necessário um esquema de endereçamento em nível de enlace para identificar os equipamentos terminais dentro de uma mesma rede local. Originalmente essas redes tinham uma topologia física em barramento, sendo compostas por cabos compartilhados entre os equipamentos terminais. Atualmente seu projeto e implantação tem como principal equipamento o *switch* ethernet, com o qual se criam topologias em malha tendo os switches como concentradores de enlaces. Com esse equipamento a capacidade da rede é melhor utilizada, pois quadros ethernet são comutados e enviados somente pelo enlace onde se encontra o equipamento terminal de destino, de acordo com a topologia da rede.

A topologia da rede corresponde a como os equipamentos estão interligados entre si. Em um modelo ideal, a topologia é formada pelo administrador de rede de forma que seja possível atender todos os equipamentos terminais e que a rede possa prover uma alta capacidade de transmissão. Com essa infraestrutura montada e funcionando, diferentes redes locais podem ser implantadas. Cada LAN pode ser utilizada para uma finalidade diferente e por usuários diferentes. De acordo com Kurose e Ross (2006) podem ser identificadas três desvantagens nesse modelo de rede: (a) a falta de isolamento do tráfego, (b) uso ineficiente dos switches e (c) mau gerenciamento dos usuários. Todas as desvantagens podem ser solucionadas utilizando um switch com suporte a virtualização, dividindo a rede local em redes virtuais isoladas.

Atualmente, uma técnica de virtualização de redes locais é conhecida como rede local virtual (VLAN) IEEE 802.1Q [IEEE802.1q, 2005]. Esse padrão foi aprovado em 1996, e é co-

nhecido também como *VLAN Tagging*. Com ele se confere aos switches ethernet a capacidade de segmentarem logicamente a rede em redes locais virtuais. Com a abordagem da VLAN a rede local é configurada por software ao invés do meio físico, criando uma divisão lógica na estrutura da LAN. Cada VLAN é identificada por um número único, escrito na etiqueta (*Tag*) de identificação.

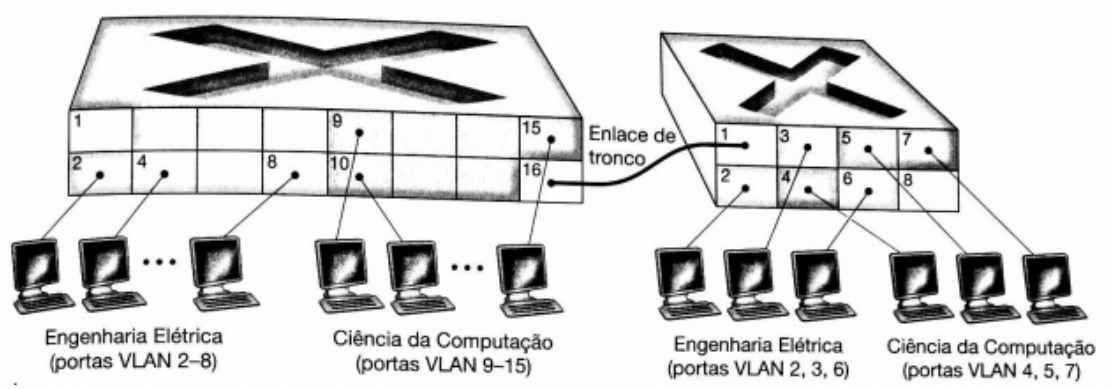

Figura 1.1: Exemplo de cenário com diferentes VLANs (KUROSE; ROSS, 2006).

Além de prover a divisão dos equipamentos em diferentes redes lógicas, de acordo com a necessidade de cada um, a virtualização se torna muito útil no caso de algum elemento da rede necessite mudar de rede local. Caso não houvesse nenhuma técnica de virtualização de redes sendo aplicada, a infraestrutura da rede precisaria ser alterada para que um equipamento terminal migrasse de uma rede local para outra. Com o uso de VLAN, apenas a configuração dos switches precisa ser alterada. Com essa alteração lógica, o equipamento terminal pode ser transferido para a rede local desejada sem alterações no ponto de vista físico.

Uma abordagem alternativa que pode ser utilizada para virtualização de redes é a Rede Definida por Software, ou *SDN* (*Software-Defined Networking*). SDN é um modelo em que o funcionamento de cada equipamento de rede, e da rede como um todo, pode ser programado de forma centralizada por meio de uma interface bem definida. Uma proposta para SDN chamada OpenFlow, idealizada na Universidade de Stanford na Califórnia, propõe um modelo em que switches (ou qualquer outro equipamento capaz de encaminhar pacotes) consultam um controlador central para saber como comutar fluxos de pacotes. Essa comunicação se faz por meio de um protocolo bem definido. Assim, o controle da rede e suas decisões ficam separados do hardware e implementados de forma centralizada (DÜRR, 2012). Esse controlador possui, portanto, uma visão global da rede e do estado de seus equipamentos, o que cria a possibilidade do administrador da rede criar regras de encaminhamento ou descarte dos pacotes, integrando ou isolando os elementos da rede da forma que for necessário.

Este trabalho propõe implantar redes locais virtuais usando OpenFlow. Para isso, pretende-

se estruturar os endereços ethernet usados dentro de uma rede local, de forma a neles embutir a identificação da rede local virtual a que pertence cada equipamento, além de sua localização dentro da topologia física. Com base nessa estruturação, regras OpenFlow podem ser definidas nos switches de forma a comutarem corretamente quadros ethernet dentro de suas redes locais virtuais.

1.1 Motivação

A implantação de redes locais virtuais sobre uma infraestrutura de rede física é uma necessidade comum em projetos de redes de computadores. Isso facilita a organização da rede como um todo, a manutenção e modificação de sua topologia, e a alocação dos recursos físicos (switches e enlaces). Em alguns cenários, tais como *datacenters* geridos como infraestruturas em nuvem, o uso de redes virtuais se torna massivo, pois centenas e mesmo milhares de clientes do datacenter precisam ter suas próprias redes exclusivas. A abordagem usual para implantar redes locais virtuais se baseia no padrão IEEE 802.1Q, que usa uma etiqueta inserida em quadros ethernet para identificar a que rede virtual ele pertence. No entanto, o modelo SDN/OpenFlow apresenta a possibilidade de implantar redes locais virtuais usando as informações contidas em cabeçalhos de protocolos das camadas de enlace e de rede, além das portas de switches por onde quadros são recebidos. Além disso, no modelo SDN/OpenFlow a rede pode ser programada a partir de uma visão global de sua estrutura, o que possibilita implantar as redes locais virtuais de forma a otimizar o uso dos recursos físicos através de uma interface centralizada e utilizar critérios diferenciados (endereço Ethernet, endereço IP, tipo de quadro) para definir qual dispositivo pertence à cada VLAN.

1.2 Objetivos

Este trabalho tem como objetivo implantar redes locais virtuais utilizando o protocolo OpenFlow. São objetivos específicos:

- Propor uma abordagem para estruturação de endereços ethernet a qual inclua a identificação da rede local virtual dentro dos endereços Ethernet.
- Criar um controlador OpenFlow que implante redes locais virtuais usando a abordagem de estruturação de endereços escolhida.

1.3 Organização do texto

O texto está organizado da seguinte forma: No capítulo 2 é apresentada a fundamentação teórica que dá suporte ao desenvolvimento do trabalho. No capítulo 3 é abordada a proposta deste trabalho. As conclusões estão dispostas no capítulo 4, contendo algumas propostas para trabalhos futuros. Os códigos fonte para os Controladores e ambiente de rede simulado Mininet estão disponíveis como anexos.

2 *Fundamentação Teórica*

Este capítulo aborda algumas características do funcionamento das redes ethernet convencionais. O assunto é importante e já foi abordado por diversos autores. Além disso, é apresentado com detalhes o método SDN e o protocolo OpenFlow, que são fundamentais para o entendimento dos cenários abordados no Capítulo 3.

2.1 Redes locais

De acordo com Tanenbaum (2003) redes de área local são redes privadas usadas para conectar computadores e que permitem a transferência de dados entre os elementos da rede e também o compartilhamento de recursos, como por exemplo, o uso de impressoras, porteiros e câmeras de segurança.

2.1.1 Endereço Ethernet

Endereço Ethernet, também conhecido por endereço MAC (*Media Access Control*), identifica uma estação em uma LAN. Padronizada pelo IEEE em 1985 [IEEE 802.3, 2012] a associação do endereço é realizada de forma que não possa ser alterado pelo usuário, normalmente gravado na memória ROM da interface de rede do equipamento, e portanto é chamado de endereço físico ou endereço de hardware.

Estes endereços possuem 48 bits de comprimento, divididos igualmente entre *OUI - Organisationally Unique Identifier* e *NIC - Network Interface Controller*, conforme a Figura 2.1 ilustra.

Os 24 bits mais significativos (OUI) são definidos pelo IEEE e utilizados para identificar o fabricante do equipamento. Os últimos 24 bits (NIC) do endereço são definidos pelo próprio fabricante, que se responsabiliza por não vender equipamentos com mesmo endereço Ethernet para as mesmas regiões (ou países). Essa preocupação diz respeito à garantia de unicidade

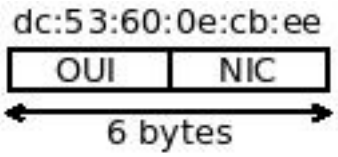

Figura 2.1: Exemplo de endereço Ethernet.

de endereços Ethernet dentro de uma mesma rede local. Isso acontece para garantir o correto encaminhamento de quadros dentro de uma rede local, do ponto de vista da camada de enlace, porque é suficiente que os endereços Ethernet sejam únicos.

2.1.2 Comutação de quadros em redes locais

Redes de computadores permitem a transferência de dados entre os dispositivos que a formam. Dentro de uma rede local privada (LAN) o equipamento mais comum na interligação dos elementos da rede é o switch ethernet, que cria o enlace físico e mapeia todos os terminais de acordo com seus respectivos endereços Ethernet. Em uma LAN a informação é transmitida através de quadros ethernet, contendo, entre outras informações, o endereço Ethernet do equipamento de origem do quadro e o endereço Ethernet para onde o quadro se destina.

Um switch aprende automaticamente que endereços ethernet são acessíveis por meio de cada porta. O aprendizado, chamado de *learning bridge* e padronizado originalmente pela IEEE em 1990 [IEEE 802.1d, 2004], acontece quando um quadro é recebido pelo switch. O endereço ethernet de origem contido no quadro é associado à porta por onde foi recebido, e memorizado pelo switch em uma tabela de endereços. Desta forma, quando um switch precisar encaminhar um quadro, seu endereço ethernet de destino é consultado na tabela de endereços. Se o endereço procurado existir na tabela, o quadro é transmitido pela porta a ele associada. Caso contrário, o quadro é transmitido por todas as portas com exceção daquela por onde foi recebido.

2.2 Virtualização de redes locais

Segundo Fernandes (2011) a virtualização de redes é capaz de dividir a rede física em várias parcelas, chamadas redes virtuais, com suas características próprias. Alguns aspectos principais devem ser garantidos a essas redes virtuais, como o isolamento, de modo que uma rede não gere interferência para as demais, o desempenho no encaminhamento de pacotes e a qualidade de serviço (*QoS*). Este trabalho tem objetivo de tratar do isolamento das redes virtuais, não abordando os quesitos de desempenho e qualidade de serviço.

2.2.1 Virtualização com redes locais virtuais (VLAN)

Em 1996 foi aprovado o padrão 802.1Q, conhecido como *VLAN Tagging*, que possibilita o uso de redes locais virtuais em equipamentos de diferentes fabricantes. O padrão abriu caminho para padronizações adicionais em questões relacionadas com VLANs.

Uma VLAN pode ser definida como uma rede local implantada e configurada por software sobre a estrutura da rede local física. Para um dispositivo pertencer à LAN é necessário que ele esteja conectado fisicamente a ela, pois o critério de participação é geográfico (FOROUZAN, 2006). A proposta da técnica de VLAN é a divisão lógica de uma LAN, criando os segmentos chamados de VLANs, com seu próprio domínio de colisão broadcast. Se um equipamento pertencia a uma VLAN for transferido para outra apenas a parte lógica será alterada, por software, a parte física permanecerá intacta.

Em redes ethernet, o padrão 802.1Q confere aos switches ethernet a capacidade de segmentarem logicamente a rede em VLANs. Essencialmente, VLANs são definidas nos switches ethernet, o que significa que nenhuma configuração adicional é necessária nos equipamentos terminais como computadores, servidores, câmeras, telefones IP e todos os demais equipamentos que se comunicam pela rede. Quadros que trafegam entre switches podem possuir um cabeçalho adicional, chamado de etiqueta (*tag*), nos quatro bytes seguintes ao campo de endereço de origem (SZCZEPANEK, 2002).

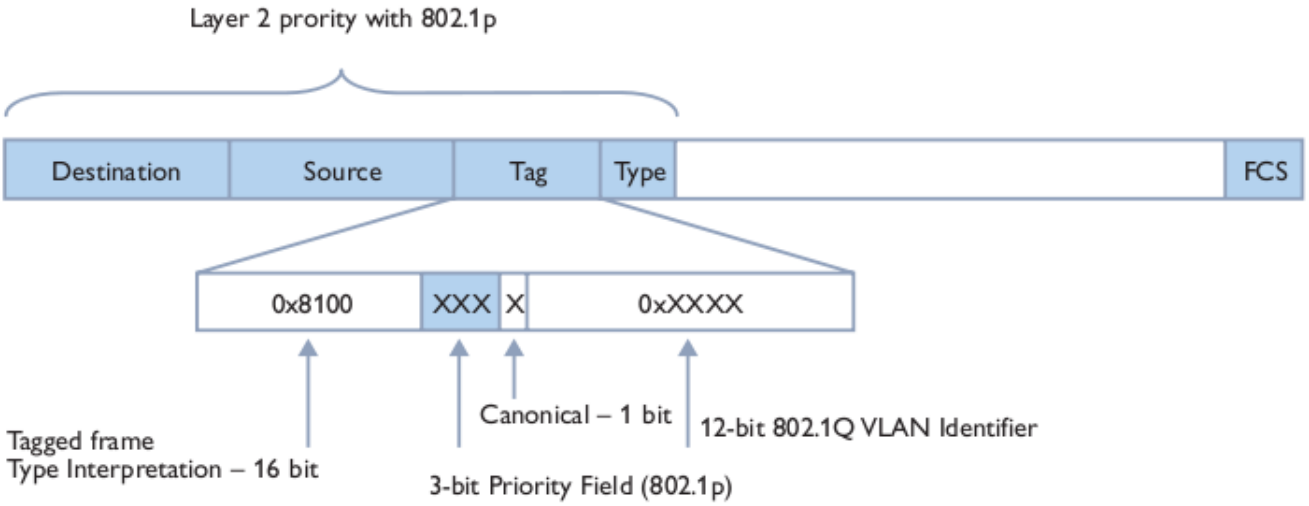

Figura 2.2: Tag inserida em um quadro Ethernet

Os dois primeiros bytes identificam o próprio cabeçalho da etiqueta e são chamados de Rótulo de Identificação de Protocolo (*Tag Protocol Identifier - TPID*) contendo o valor 0x8100h. Os outros dois bytes contêm a informação de controle da etiqueta (*Tag Control Information - TCI*). (KUROSE; ROSS, 2006)

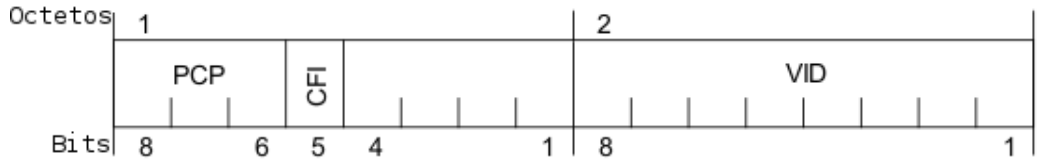

Figura 2.3: Tag Control Information (TCI) (IEEE; NETWORKS, 2005).

Em redes em que se implantam VLANs com o padrão IEEE 802.1Q, uma VLAN é reconhecida e diferenciada por um número: o identificador de VLAN (*Vlan Identifier - VID*). Essa informação está contida dentro do TCI, e é inserida nos quadros ethernet para indicar a qual VLAN cada quadro pertence, conforme a Figura 2.3.

Portas do switch que recebem e transmitem quadros com o cabeçalho Tag operam em modo *tagged*, do contrário operam em modo *untagged* e não alteram ou adicionam os cabeçalhos dos quadros com as informações sobre VLAN. Isso é necessário somente quando os quadros trafegam entre equipamentos usados para definir a topologia da VLAN (essencialmente switches ethernet). Assim, entre equipamentos terminais usualmente não é necessário esse cabeçalho adicional, mas entre switches sim.

Como se pode observar na Figura 2.3, o cabeçalho TCI reserva 12 bits para o número identificador de VLAN (VID). Portanto, é possível configurar até 4096 VLANs simultaneamente em uma mesma rede local física. A quantidade de identificadores pode parecer grande, mas é um ponto de fragilidade do padrão. Existe a possibilidade de que a quantidade de VLANs possa ser insuficiente e impossibilite que o cenário seja implementado com o uso da técnica. Normalmente LANs de pequeno e médio porte não utilizam toda a capacidade dos identificadores de VLAN, mas se considerarmos um grande *datacenter*, essa quantidade pode não ser suficiente, inviabilizando a construção do cenário.

A criação de VLANs em uma rede local envolve a participação ativa do administrador de rede, que deve configurar as portas de switches apropriadamente. Tal tarefa é trabalhosa e sujeita a erros, e a manutenção ou modificação de VLANs deve ser feita com cuidado para não romper as comunicações entre os dispositivos da rede. Existe um protocolo para configuração automática, chamado GVRP (HEUVEN et al., 2003), que auxilia a configuração das portas de switch. Com ele, basta definir as VLANs nas portas dos switches de borda (onde se conectam os equipamentos terminais), sendo que as portas entre switches têm suas VLANs determinadas automaticamente.

2.3 Software Defined Networking - SDN

Atualmente infraestruturas de redes são compostas por equipamentos (roteadores, switches, pontos de acesso) que funcionam de forma autônoma, possivelmente usando protocolos especiais para intercambiarem informações que os ajudem a decidir como encaminhar pacotes entre si. Isso pode ser exemplificado tanto em redes locais quanto em redes WAN.

Em redes locais, switches comutam quadros ethernet de forma independente, baseando-se exclusivamente em seus endereços MAC de destino. Para saber em que porta se encontra cada MAC em sua rede local, um switch mantém uma tabela que associa os endereços MAC conhecidos às portas de onde foram recebidos seus respectivos quadros ethernet. Além disso, switches ethernet usam o protocolo STP para definirem uma topologia lógica isenta de caminhos fechados, sem a qual a comutação de quadros causaria anomalias na rede, com quadros enviados para broadcast sendo transmitidos indefinidamente ao longo de caminhos fechados. Por fim, um switch comuta quadros somente entre portas de uma mesma VLAN, sendo que a configuração de VLAN é definida manualmente pelo administrador de rede (podendo ser auxiliada pelo protocolo GVRP). Assim, cada switch opera de forma autônoma, com algumas funcionalidades dependendo de trocas de informações com outros switches para se obter um funcionamento consistente da rede local.

No caso de redes WAN, roteadores usam suas tabelas de rotas para encaminhar datagramas IP com base em seus endereços de destino. Tais tabelas são atualizadas por meio de rotas adicionadas manualmente pelo administrador de rede ou por meio de protocolos de roteamento capazes de descobrir rotas dentro da rede. Assim, com essa abordagem tradicional, modificações na estrutura ou no funcionamento da rede (no que diz respeito ao encaminhamento de pacotes) são tomadas localmente por cada equipamento de rede e a forma como esses equipamentos funcionam e tomam decisões de encaminhamento é opaca e não pode ser modificada. A Figura 2.4 mostra como funciona essa organização.

A proposta SDN (*Software-Defined Networking*) é justamente possibilitar que os equipamentos de rede possam ser programados no que diz respeito à forma como encaminham pacotes. Isso abre a possibilidade de controlar a rede globalmente (STEIN, 2012), por ser um modelo que permite estruturar a rede logicamente de forma programável. Desta forma, a consistência no funcionamento da rede pode ser definida de forma global, atualizando os estados dos equipamentos de forma centralizada, o que difere da forma com que esses equipamentos funcionam atualmente.

SDN é uma maneira de se construir redes de computadores, separando e abstraindo os

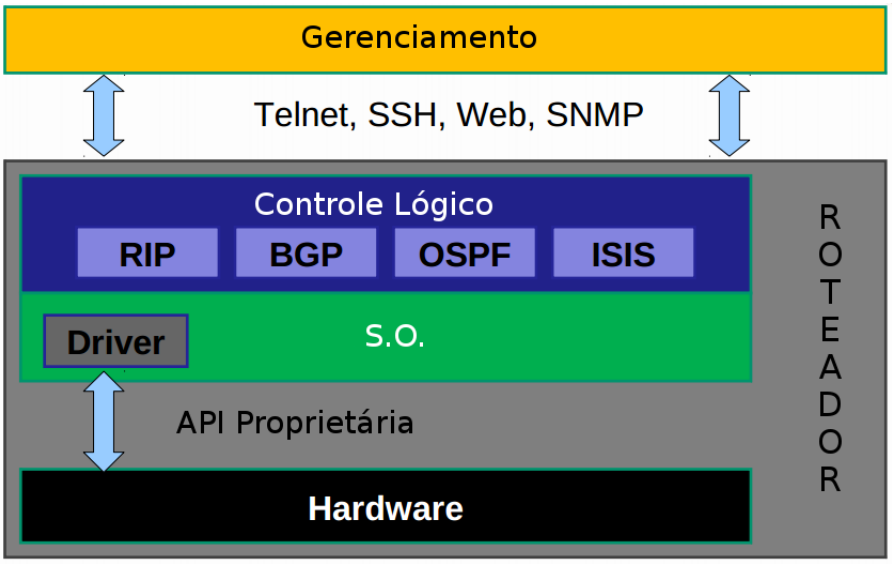

Figura 2.4: Modelo atual do roteador (ROTHENBERG, 2012).

elementos desse sistema. O controle da rede será retirado do hardware e dado ao Controlador, uma aplicação de software.

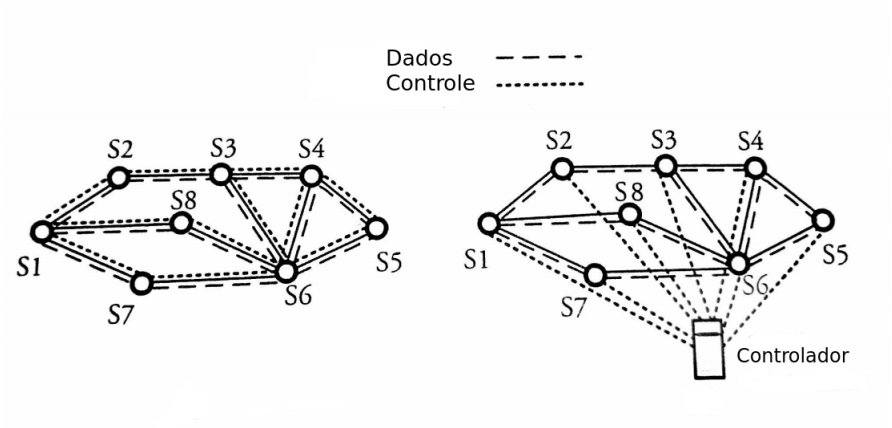

Figura 2.5: Comparação entre modelos de rede (MIR, 2006).

O administrador da rede tem o poder de modelar o tráfego da sua rede, com o uso de um Controlador central, sem precisar modificar a configuração de cada switch separadamente (MCKEOWN et al., 2008). Podem-se trocar as regras da rede, como modificar a prioridade de um certo fluxo ou bloquear o tráfego de certo serviço, de um modo detalhista e específico. Isso é de grande ajuda em uma arquitetura de computação em nuvem, por exemplo, pois possibilita o administrador gerenciar as cargas de tráfego de maneira flexível e eficiente e de forma imprevista pelos padrões de rede existentes. Um modelo dessa abordagem é exemplificado na Figura 2.6

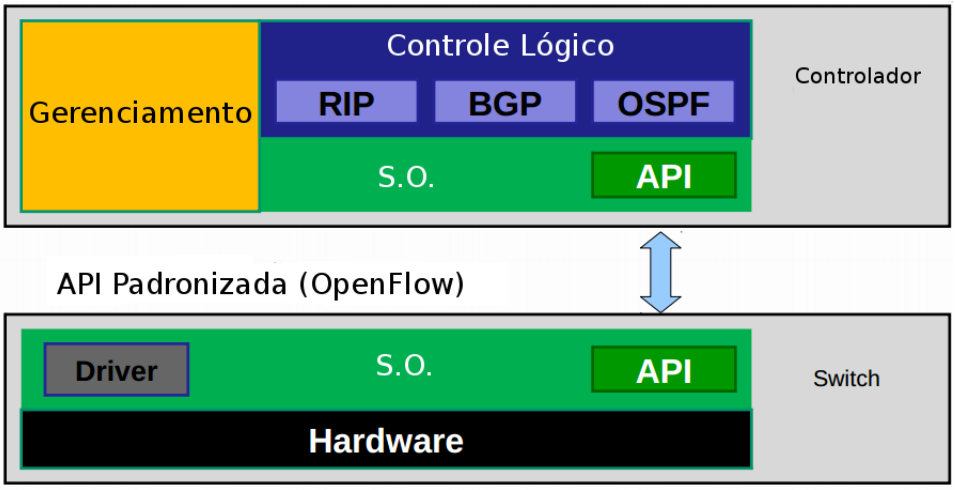

Figura 2.6: Modelo de roteador com SDN (ROTHENBERG, 2012).

2.3.1 OpenFlow

Novas tecnologias mudaram e complicaram a natureza das redes, como por exemplo, a necessidade de grandes *datacenters* e do uso de computação em núvem (*cloud computing*), tornando mais complexo o modo como as redes são utilizadas (VAUGHAN-NICHOLS, 2011). Para suprir essas demandas, os administradores esperam que suas redes sejam mais inteligentes e melhor utilizadas, além de poder controlar e gerenciar suas redes. Essa necessidade fez o interesse em SDN/OpenFlow aumentar.

OpenFlow é um protocolo aberto para programar a *tabela de fluxo*, tabela que contém as regras de processamento de quadros, em diferentes roteadores e switches (MCKEOWN et al., 2008). Criado em 2008, o projeto é abrigado na Universidade de Stanford - Califórnia e a especificação do OpenFlow Switching tem o objetivo de apoiar o OpenFlow (SPECIFICATION, 2011).

Segundo (MCKEOWN et al., 2008), o OpenFlow pode ser dividido em, no mínimo, três partes:

- A tabela de fluxo, criada pelo Controlador e descreve ao switch como processar certo fluxo;
- Um canal seguro que conecta o switch com o Controlador;
- Protocolo OpenFlow, que provê um método aberto e padronizado de comunicação com o switch.

A Figura 2.7 ilustra que o switch OpenFlow se comunica com o Controlador através do

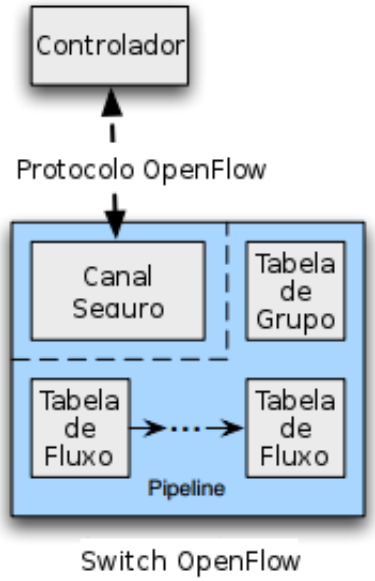

Figura 2.7: Modelo de switch OpenFlow (SPECIFICATION, 2011).

Protocolo OpenFlow, em um canal seguro. Também indica os componentes de um switch: uma ou mais tabelas de fluxo e uma tabela de grupos, que são responsáveis pela classificação (*lookup*) dos pacotes, e um canal OpenFlow para conectar-se a um Controlador externo. O Controlador gerencia o switch, sendo capaz de adicionar, alterar e deletar as regras (*flow entries*) (SPECIFICATION, 2011). Cada tabela de fluxo contém um conjunto de regras. Cada regra consiste em um campo para comparação e classificação (*match field*), contadores e o conjunto de aplicações e instruções para cada pacote classificado.

A comparação começa na primeira tabela de fluxo e pode continuar para as tabelas seguintes, se necessário. As regras, cujos campos de comparação estão contidos na Tabela 2.1, funcionam de forma hierárquica, procurando encontrar uma regra na primeira tabela. Caso uma regra seja encontrada (i.e. seu campo *match field* avalie para verdadeiro), as instruções associadas com esta regra específica são executadas; isso pode envolver busca por regras em outras tabelas.

Caso não seja encontrada nenhuma regra para o pacote na tabela, o próximo passo depende da configuração do switch: o pacote pode ser encaminhado pelo protocolo OpenFlow ao Controlador (via canal OpenFlow), descartado ou pode continuar para a próxima tabela de fluxo, buscando uma regra onde se adeque.

Cada regra das tabelas de fluxo contém ou um conjunto de ações ou modificam o processamento *pipeline*. As ações são instruções que modificam o pacote, descrevem seu encaminhamento e seu processamento com a tabela de grupo. As instruções de pipeline permitem que os

Campos de Comparação (*match field*)
Porta de entrada
Metadados
Ethernet de Origem
Ethernet de Destino
Tipo de Ethernet
Identificação de VLAN
Prioridade de VLAN
Etiqueta MPLS
Classe de Tráfego MPLS
Origem IPv4
Destino IPv4
IPv4 com ARP
Bits do Tipo de Serviço no IPv4
Porta de Origem do TCP/UDP/SCTP do ICMP
Porta de Destino do TCP/UDP/SCTP do ICMP

Tabela 2.1: Campos do pacote utilizados para comparação (SPECIFICATION, 2011)

pacotes sejam enviados às próximas tabelas para uma nova verificação de *match* e novas ações a serem realizadas com o pacote. O processamento pipeline termina quando o conjunto de ações da regra não define uma próxima tabela. Provavelmente o pacote já foi alterado o suficiente para que possa ser encaminhado.

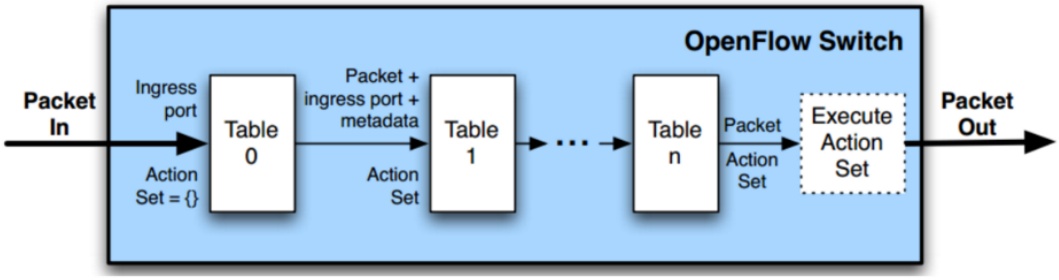

Figura 2.8: Modelo de processamento pipeline OpenFlow (SPECIFICATION, 2011).

As regras de encaminhamento de pacotes devem especificar por qual porta do switch a ação será realizada. De modo geral, é uma porta física do equipamento, mas pode ser uma porta lógica definida pelo switch ou uma porta reservada pela especificação do OpenFlow. Portas reservadas são utilizadas para fazer um encaminhamento mais genérico dos pacotes como enviá-los ao Controlador, realizar a transmissão em broadcast (*flooding*) ou encaminhar sem métodos OpenFlow (como um switch comum). As portas lógicas definidas pelo switch podem indicar grupos de agregação de links, túneis ou interfaces de loopback.

As ações associadas nas regras de encaminhamento podem enviar o pacote para um grupo,

que define processamento adicional. Grupos geralmente são utilizados para regras de *flooding* ou para regras de encaminhamento de vários fluxos para um mesmo destino, como um próximo salto na rede. Essa abstração permite que as regras de encaminhamento das tabelas sejam alteradas de forma eficiente.

A tabela de grupos contém regras de grupos. Cada uma dessas regras de grupo contém uma sequência de ações para modificar o pacote e o seu fluxo antes de encaminhá-lo às portas participantes do grupo.

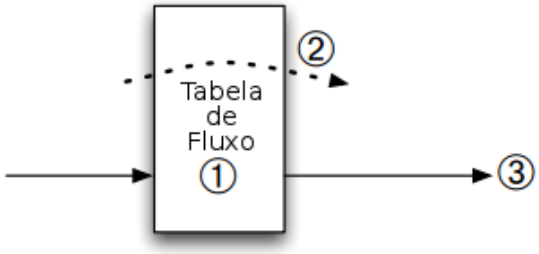

Figura 2.9: Modelo de tratamento OpenFlow (SPECIFICATION, 2011).

A Figura 2.9 exemplifica o processo de encaminhamento de um quadro. Primeiramente é feita a procura da regra, com a maior prioridade, que seja válida para o pacote (item 1). Após encontrar um *match*, as ações (*actions*) são aplicadas (item 2) e, por fim, o pacote é encaminhado (item 3).

Um switch OpenFlow não necessariamente precisa suportar todas as possíveis ações do OpenFlow. Existe um conjunto de ações obrigatórias e outro conjunto de ações opcionais. A seguinte lista traz algumas possibilidades de ações que uma regra pode conter.

- *Output*: A ação Output encaminha um pacote para uma porta especificada. Os switches OpenFlow devem ser capazes de suportar o encaminhamento para portas físicas e lógicas.
- *Drop*: Não existe uma ação explícita para o descarte. Na verdade, os pacotes que não contém ação nenhuma serão descartados.
- *Group*: Processa o pacote através do grupo específico.
- *SetQueue* (opcional): Ação que modifica o ID de fila do pacote. Quando um pacote é encaminhado com a ação Output, o ID de fila determina em qual fila da porta de saída o pacote será enviado.
- *PushTag/PopTag* (opcional): Ação que ajuda a integração com as redes já existentes, pois

pode facilmente modificar os campos dos cabeçalhos dos pacotes. Por exemplo, adicionar ou retirar um cabeçalho VLAN ou MPLS.

O canal OpenFlow é a interface que conecta o switch OpenFlow com o Controlador (SPECIFICATION, 2011). É através dessa interface que se realiza o gerenciamento e a configuração do switch e também o recebimento de eventos. O canal OpenFlow utiliza o protocolo de transporte TCP para fazer as transmissões.

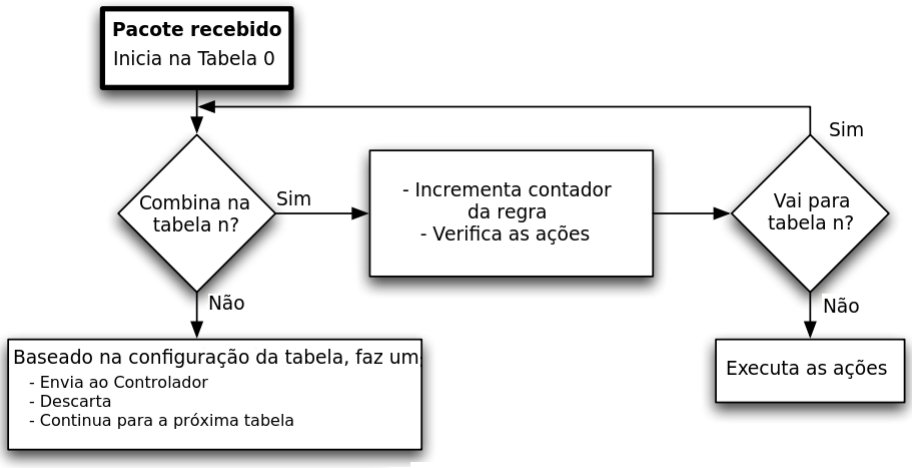

Figura 2.10: Fluxograma do processamento do pacote OpenFlow (SPECIFICATION, 2011).

A Figura 2.10 ilustra um exemplo de processamento OpenFlow. Um pacote é recebido e a tabela inicial de regras é a tabela 0. Dentro da tabela será buscada uma combinação (match) dos campos do quadro recebido entre as regras instaladas nessa tabela específica. Caso não combine com nenhuma regra, o pacote poderá ser encaminhado ao Controlador, ser descartado ou ir para a próxima tabela de regras, de acordo com a configuração. Quando uma combinação for verdadeira, incrementa-se o contador da regra e verifica-se as instruções da regra. Caso exista uma ação encaminhando o pacote para uma outra tabela, o pacote será novamente verificado em busca de combinações, caso contrário as ações da regra são executadas.

O protocolo abre oportunidades para estudos e inovações nas redes já existentes, pois, por exemplo, um gerente de rede pode particionar o fluxo entre produção e pesquisa. Dessa forma, os pesquisadores podem controlar seus fluxos, escolhendo rotas e receptores, testando novos protocolos de roteamento, segurança e esquemas de endereçamento na mesma rede que a produção utiliza, sem nenhum dano.

2.3.2 Controlador

O Controlador é o elementro central das decisões de processamento de fluxos em uma rede que implementa SDN/OpenFlow, pois é através dele que o administrador da rede define como se comportam certos encaminhamentos ou descartes de dados em cenários específicos. As tabelas de fluxo, que determinam as decisões de um switch, são moldadas de acordo com as regras programadas no Controlador e atualizadas sempre que necessário.

Existem diferentes maneiras de se construir um Controlador OpenFlow, algumas plataformas estão disponíveis para servirem de base para programar e testar o protocolo através de cenários experimentais. Utilizando a linguagem Java, Beacon e Floodlight são as possíveis opções. A linguagem Python conta com o POX e Ryu, e a linguagem Ruby com o Trema. Todas as plataformas foram projetadas para pesquisa e educação, e a escolha entre eles deve ser tomada de acordo com a linguagem de programação que o administrador é mais familiarizado. Neste trabalho foi utilizado o framework Ryu para o desenvolvimento do Controlador OpenFlow, que permite trabalhar com diferentes versões do protocolo (1.0, 1.2, 1.3 e 1.4) e tem uma boa documentação[1].

[1] Disponível em: https://osrg.github.io/ryu/

3 *Proposta e Resultados*

Este trabalho propõe implantar redes locais virtuais (VLANs) usando OpenFlow para prover o isolamento da rede. A definição de VLAN na abordagem proposta se assemelha àquela contida no padrão IEEE 802.1q: cada VLAN se compõe de um conjunto de portas de switch, em que podem estar conectados equipamentos terminais ou outros switches. A diferença se encontra na forma com que essa estrutura é implementada. Ao invés de identificar VLANs por números (chamados de VID no padrão IEEE 802.1q) e usá-los para rotular portas de switch e quadros Ethernet que trafegam na rede, propõe-se usar regras OpenFlow para limitar o encaminhamento de quadros às VLANs a que pertencem. Essas regras devem explorar os endereços Ethernet contidos nos quadros, de forma que sejam encaminhados somente entre switches e equipamentos que sejam parte da mesma VLAN dos equipamentos de onde se originaram.

Endereços Ethernet não possuem uma estrutura no que diz respeito à rede local em que são usados. Equipamentos que formam uma rede local possuem cada um seu próprio endereço Ethernet, porém não há nenhuma informação nesses endereços que corresponda à essa rede local. Assim, em princípio não há como switches identificarem a qual rede local (ou VLAN) pertence um quadro Ethernet em trânsito apenas com base nos endereços Ethernet nele contidos. Se a informação sobre VLAN puder ser codificada nesses endereços, isso pode ser usado por switches para comutarem quadros eficientemente dentro de suas VLANs e gerar um aproveitamento melhor dos 96 bits reservados apenas ao endereçamento de origem e destino de cada quadro.

A implantação de redes locais virtuais proposta neste trabalho se baseia na codificação em seu endereço Ethernet da localização física de um equipamento dentro de uma rede local. Essa codificação deve ser realizada em qualquer switch onde estejam conectados equipamentos terminais. Quando um desses switches receber um quadro vindo de um equipamento terminal, seu endereço Ethernet deve ser substituído por um endereço que codifique as informações sobre sua localização física, representada por (a) o switch em que está conectado, (b) a qual VLAN o equipamento pertence, e (c) o número do equipamento no escopo desse switch (porta). O identificador da VLAN depende da topologia implantada, sendo definido pelo gerente de rede.

A modificação de endereços Ethernet é uma das ações possíveis de serem realizadas com regras OpenFlow. Endereços assim codificados podem em seguida ser usados pelos switches para o correto encaminhamento para seus destinos, mediante regras OpenFlow que avaliem esses endereços.

Os endereços Ethernet modificados pelos switches podem ser entendidos como endereços Ethernet virtuais. Esse conceito foi explorado em outros trabalhos, porém no contexto de grandes *datacenters*, como (MYSORE et al., 2009) e (MATIAS et al., 2011). De forma análoga a esses trabalhos, a estrutura proposta para esses endereços busca facilitar a criação de regras OpenFlow para que os switches possam encaminhar quadros sem precisar consultar frequentemente o controlador OpenFlow. Além disso, pretende-se com essa abordagem reduzir a quantidade de regras necessárias nos switches. No entanto, para que isso funcione, cada quadro Ethernet enviado por um equipamento terminal deve ser destinado a um desses endereços virtuais. No caso de redes TCP/IP, isso implica respostas de requisições ARP[1] conterem um endereço virtual ao invés do endereço Ethernet real do equipamento procurado. Esse requisito também foi identificado no trabalho PortLand (MYSORE et al., 2009), e pretende-se adotar a solução ali proposta. Desta forma, switches devem usar regras OpenFlow para interceptar respostas a requisições ARP, e substituir endereços Ethernet nelas informados pelos respectivos endereços virtuais.

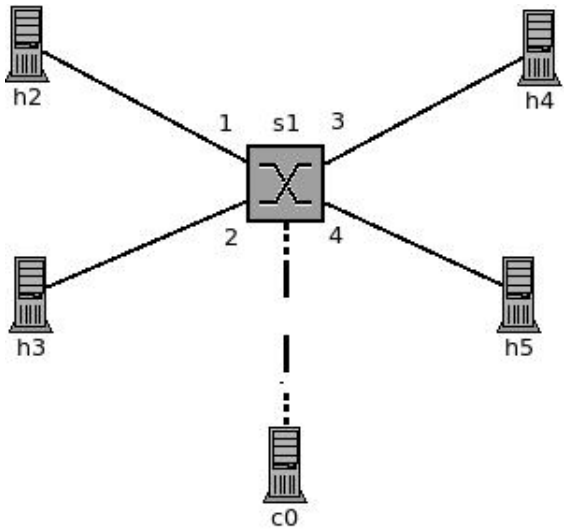

Figura 3.1: Exemplo de topologia proposta

A Figura 3.1 demonstra um exemplo de topologia possível para a implementação das redes virtuais baseadas em SDN/OpenFlow, com todos os elementos necessários: um switch ethernet (s1) com suporte à OpenFlow, quatro hosts (h2, h3, h4 e h5) conectados em suas portas e o

[1] *Address Resolution Protocol*, usado para descobrir que endereço Ethernet está associado a determinado endereço IP, e especificado pela RFC 826

Controlador (c0), que conecta-se diretamente ao switch utilizando o Protocolo OpenFlow.

3.1 Ambiente de desenvolvimento

Para desenvolver este trabalho foi utilizada a seguinte plataforma: switches do projeto OpenFlow 1.2 Software Switch, desenvolvido pelo CPqD. Esse é projeto de código aberto para prover serviços de roteamento virtualizado com OpenFlow[2] em conjunto com máquinas virtuais fornecidas pela ferramenta de simulação de rede Mininet para formar a rede. O sistema operacional utilizado foi o Ubuntu 11.10. Dentre as possíveis opções de linguagens para programar o Controlador (Java, Python, C++ e Ruby), foi escolhida a plataforma Ryu para desenvolver o Controlador, que se baseia na linguagem Python.

Uma premissa adotada nos cenários experimentais foi a topologia da rede ser fixa e previamente conhecida pelo Controlador. Caso alguma alteração na rede seja necessária, a programação das regras do Controlador precisa ser revisada e atualizada para se ajustar a nova topologia.

3.2 Cenário introdutório

O primeiro cenário construído e estudado foi com um switch e quatro hosts, e pode ser facilmente ajustado para qualquer quantidade de hosts. A ferramenta Mininet oferece a criação do cenário simulado com um único switch e *n* hosts por padrão, não sendo necessário nenhuma configuração adicional para o ambiente de testes.

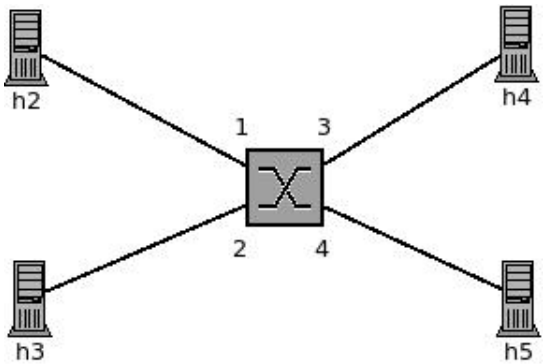

Figura 3.2: Topologia física da rede no Cenário 1

A segmentação da LAN foi idealizada pelo número da porta do switch, dividindo as portas pares das portas ímpares. O Controlador foi desenvolvido para interceptar as mensagens ARP e substituir o endereço Ethernet contido no cabeçalho e conteúdo do pacote. O endereço Ethernet

[2] Acessível em https://github.com/CPqD/of12softswitch

virtual deste cenário segue o seguinte padrão: (a) número identificador da VLAN, (b) número do switch e (c) número da porta do switch onde o host se conecta.

No primeiro cenário temos duas VLANs configuradas, portanto. O número de identificação da VLAN está composto por 4 bytes, utilizando os 32 bits mais significativos do endereço. Na sequência temos um byte dedicado a identificar o switch da rede, nesse caso sempre com o mesmo número de identificação. O último byte, com os 8 bits menos significativos do endereço, é destinado a identificar qual porta do switch o quadro foi recebido. Substituindo assim os endereços físicos associados a cada host componente da rede por uma informação visualmente mais estruturada e isolando o tráfego entre as VLANs. Nenhuma configuração é necessária nos hosts.

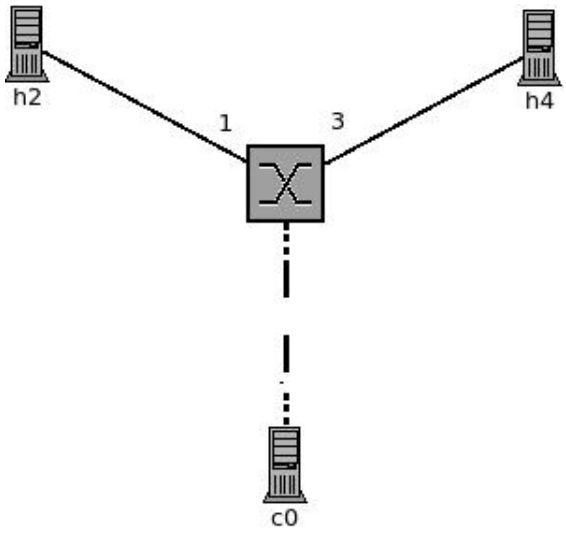

Figura 3.3: Topologia da VLAN 1 no Cenário 1

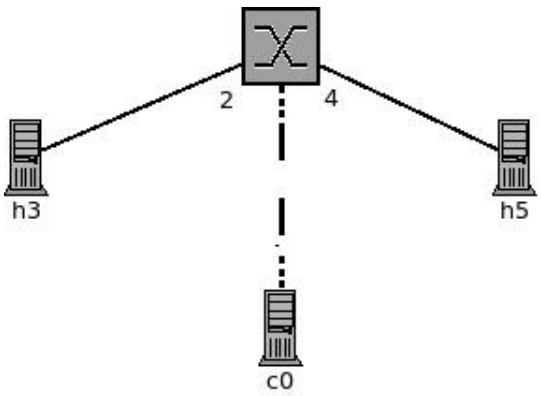

Figura 3.4: Topologia da VLAN 2 no Cenário 1

A identificação da VLAN formada por hosts conectados em portas pares é 01:01:01:11 e a VLAN formada pelas portas ímpares é 02:02:02:22. Com esse formato possuindo 32 bits de identificador, são 4.294.967.296 possibilidades de VLANs diferentes no mesmo cenário. Dessa

forma, podemos observar a flexibilidade oferecida pelo OpenFlow e uma definitiva superioridade em relação ao total de 4096 VLANs suportadas pelo padrão 802.3Q, sem abrir mão de nenhuma qualidade ou vantagem da virtualização.

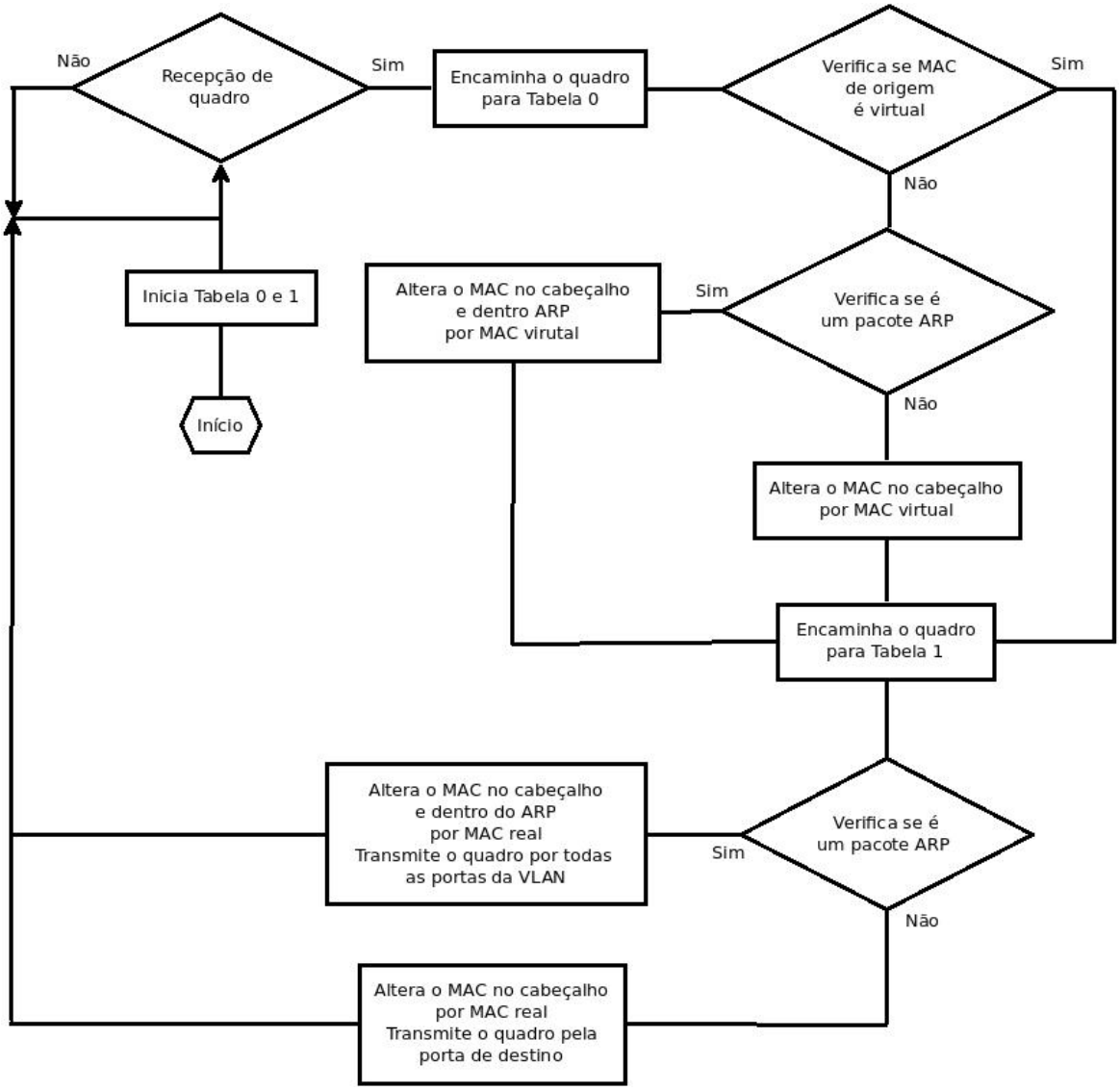

Figura 3.5: Fluxograma de processamento das regras OpenFlow no switch

O fluxograma demonstrado na Figura 3.6 descreve as decisões e ações do Controlador ao receber um novo quadro. É importante destacar o uso das duas tabelas de regras deste cenário, numeradas como tabela 0 e tabela 1. A tabela 0 contém as regras que tratam pacotes recebidos pelo switch que ainda apresentam o endereço Ethernet real do host em seu cabeçalho. Caso um pacote tenha o endereço virtual em seu cabeçalho, será direcionado imediatamente à tabela 1, responsável por fazer o encaminhamento do pacote dentro da VLAN.

Ao confirmar o recebimento de um pacote com endereço real, verifica-se por qual porta (*in_port*) foi recebido para decidir à qual VLAN o host pertence e determina-se qual seu endereço

virtual. O Controlador também verifica se o pacote é uma mensagem do procolo ARP. Caso positivo, altera-se o endereço real (*eth_src*) contido no cabeçalho e no conteúdo do pacote para o endereço virtual. Caso contrário, apenas o endereço do cabeçalho será alterado. Após essa(s) alteração(ões) o pacote é direcionado à tabela 1.

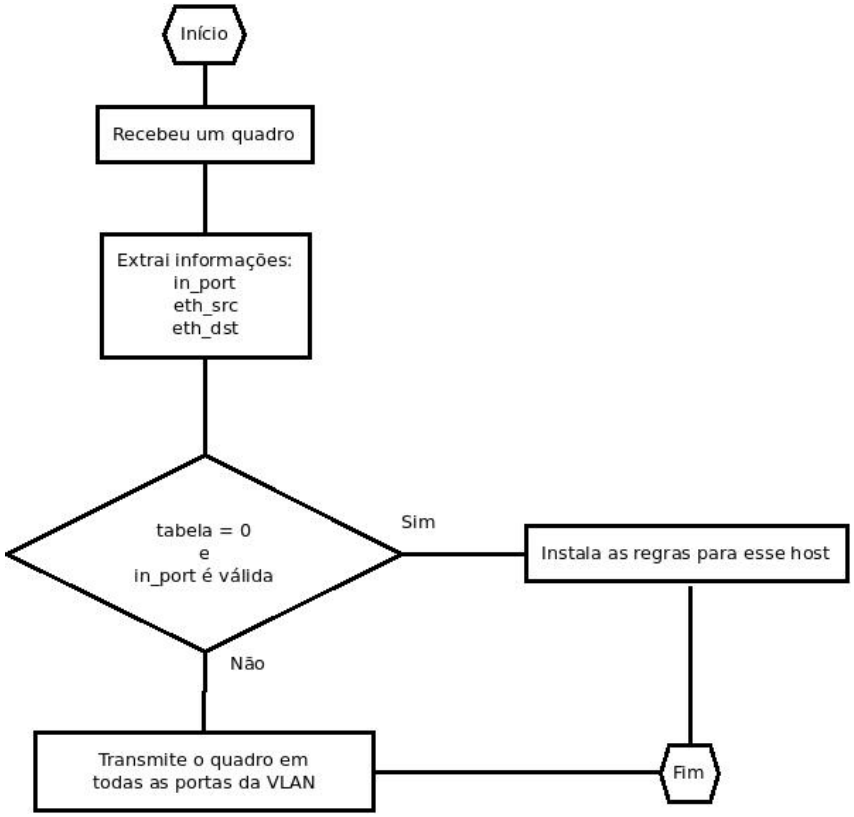

Figura 3.6: Fluxograma de processamento ao receber um quadro

A tabela 1 contém apenas regras de encaminhamento dos pacotes. O host destinatário deve receber o pacote com o endereço real do host de origem, portanto, verifica-se novamente se é uma mensagem ARP para fazer a substituição dos endereços. Caso seja um pacote ARP, ambos os endereços são modificados para o endereço real do host antes de encaminhar o pacote para todas as portas da VLAN. Caso seja um pacote qualquer, modifica-se o endereço Ethernet do pacote para o endereço real e o pacote é direcionado apenas ao host de destino (*eth_dst*).

Host	MAC Real	MAC Virual	VLAN
h2	fe:23:d0:13:bb:2d	02:02:02:22:01:01	02
h3	5e:5a:8a:08:b5:b8	01:01:01:11:01:02	01
h4	fe:81:43:e2:ca:18	02:02:02:22:01:03	02
h5	2e:c9:1b:f2:01:30	01:01:01:11:01:04	01

Tabela 3.1: Relação entre endereços reais e virtuais do Cenário 1

A Tabela 3.1 apresenta a comparação entre os endereços reais e virtuais do cenário introdutório, além de sinalizar a VLAN que o host pertence.

3.3 Cenário final

O segundo cenário do projeto foi construído com três switches conectados entre si e com três hosts conectados em cada switch, somando nove hosts. A Figura 3.7 ilustra como o cenário está distrubuído e conectado físicamente. O Mininet não fornece um ambiente simulado padrão para essa topologia, portanto foi necessário criar um arquivo customizado conforme a necessidade do projeto, conforme descrito no anexo B. Todos os elementos da rede precisam ser referenciados e todas as conexões são individualmente descritas. Uma vez criado o cenário para a simulação é possível verificar nos switches quais dispositivos estão conectados em cada porta.

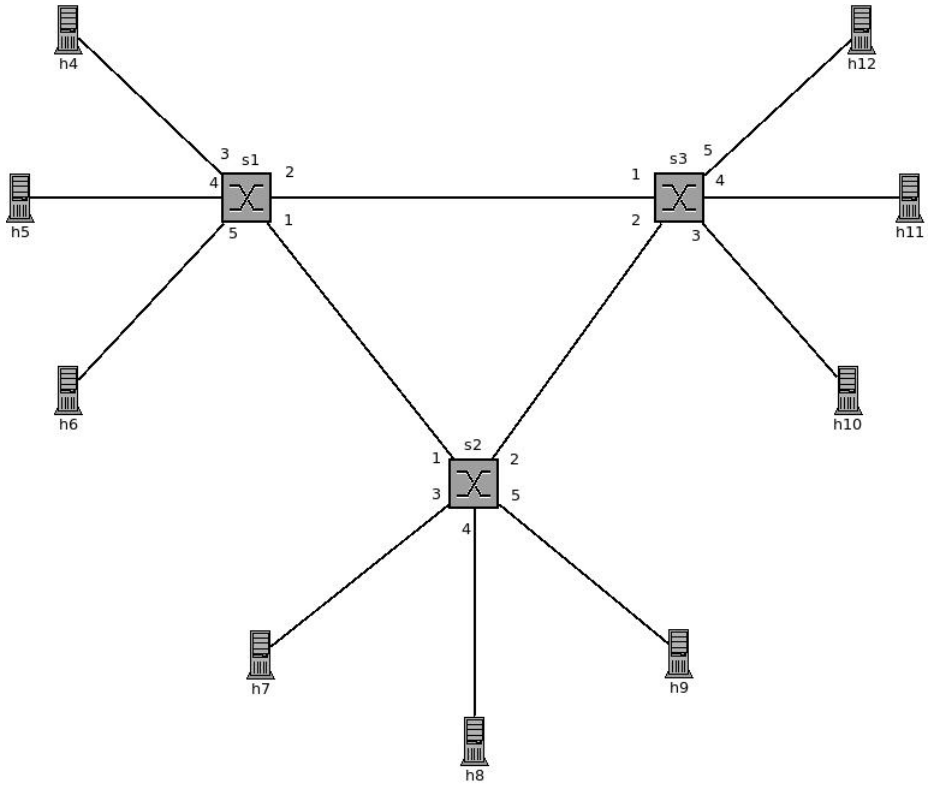

Figura 3.7: Topologia física da rede no Cenário 2

Para a segmentação dessa rede três VLANs foram utilizadas. O método de ingresso na VLAN foi aleatório, limitando um host participante por switch e apenas uma VLAN por host, forçando que os hosts de cada VLAN utilizem o núcleo da rede para comunicarem-se entre si. Como os switches formam conexões com caminhos fechados e nenhum protocolo do tipo spanning-tree é utilizado, cada VLAN estabelece uma topologia diferente de conexões no núcleo da rede, evitando anomalias causadas por loops e criando um cenário diferente em cada caso.

O mapeamento das conexões entre switches para cada VLAN do cenário deve ser descrito de forma estática e ficar disponível para que o Controlador consulte as três topologias virtuais sobre a estrutura física do cenário. Portanto, caso seja necessário alterar algum caminho entre switches para uma determinada VLAN, a descrição da topologia virtual precisa ser ajustada afim de satisfazer a nova configuração.

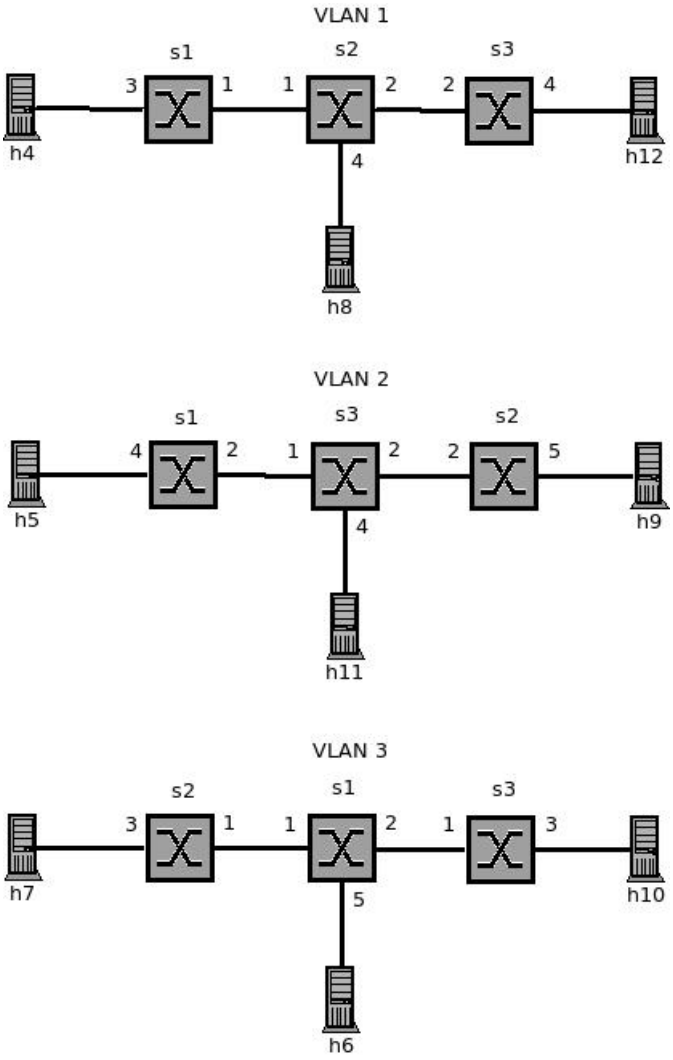

Figura 3.8: Topologias das VLANs no Cenário Final

O Controlador desenvolvido para esse experimento não propaga quadros com requisições ARP recebidas de outro switch, portanto torna-se necessário que o administrador da rede faça o mapeamento dos endereços manualmente após iniciar o ambiente simulado Mininet e a aplicação do Ryu, relacionando o endereço IP do host de destino com o seu endereço ethernet virtual. A Figura 3.9 exemplifica como fica o comando para manipular a tabela ARP do host.

```
$ arp -s 10.0.0.4 12:34:56:01:01:03
```

Figura 3.9: Configuração manual do ARP

Como já estabelecemos um experimento com um número de VLANs disponível muito superior ao padrão 802.3Q, nesse cenário foi adotado um prefixo comum à todos os endereços virtuais, foi utilizado o formato de acordo o seguinte padrão: (a) prefixo, (b) número identificador da VLAN, (c) número do switch e (d) número da porta do switch onde o host se conecta.

O prefixo dos endereços será 12:34:56, utilizando os 24 bits mais significativos de forma bem similiar ao identificador de fabricante do padrão 802.3Q. O byte seguinte vai trazer a informação de qual VLAN o host pertence, seguido pela identificação do switch e sendo finalizado com o número da porta, ambos também com um byte de comprimento.

Host	MAC Real	MAC Virual	VLAN
h4	36:e7:d8:ea:9a:45	12:34:56:01:01:03	01
h5	f2:a6:58:44:7f:1f	12:34:56:02:01:04	02
h6	4a:1d:94:b8:98:68	12:34:56:03:01:05	03
h7	d2:e1:a0:21:5a:9c	12:34:56:03:02:03	03
h8	3a:36:c4:39:e9:13	12:34:56:01:02:04	01
h9	92:ea:b0:61:26:dd	12:34:56:02:02:05	02
h10	86:71:0d:e1:07:57	12:34:56:03:03:03	03
h11	2e:ed:58:05:c4:f6	12:34:56:02:03:04	02
h12	0a:62:53:4d:ba:d0	12:34:56:01:03:05	01

Tabela 3.2: Relação entre endereços Ethernet reais e virtuais do Cenário 2

A Tabela 3.2 apresenta a comparação entre os endereços reais e virtuais relativos ao cenário final, também informa a VLAN que o host pertence.

O Controlador do cenário utiliza duas tabelas de regras, nomeadas como tabela 0 e 1. Quando um pacote é recebido em uma das portas do switch, é executada uma busca nas topologias virtuais pra verificar qual a VLAN o pacote pertence. Se o quadro ainda possui seu endereço real, é sinal de que o host está conectado diretamente neste switch e é necessário enviá-lo à tabela 0. A tabela 0 destina-se a tratar os pacotes que contém o endereço ethernet real do host de origem, instalando as regras que definem o endereço virtual desse dispositivo baseado nas informações da topologia.

Quando o pacote já chega com o seu endereço virtual, significa que chegou de outro switch da topologia e será processado na tabela 1. A tabela 1 contém as regras de encaminhamento dos pacotes, sendo responsável por direcioná-los ao próximo switch da rede utilizando o endereço virtual ou ao host de destino, ajustando o cabeçalho para utilizar o endereço real. O fluxograma apresentado na Figura 3.6 ilustra esse processamento.

4 Conclusões

Este trabalho propôs a implementação de redes virtuais utilizando SDN/OpenFlow através de um protótipo executado dentro de um ambiente simulado. O objetivo proposto busca substituir os endereços Ethernet dos hosts da rede por endereços virtuais, que codificam e apresentam a identificação da VLAN a qual o host pertence, entre outras informações possíveis.

Utilizando o ambiente de trabalho descrito no item 3.1 foi construído e testado um *Controlador*, elemento centralizado que programa o comportamento dos switches, para interceptar os quadros que trafegam na rede e realizar a ações que o protocolo OpenFlow proporciona para a segmentação da topologia física em redes virtuais e estruturação dos endereços dos elementos dentro da rede.

Foram utilizados dois cenários com topologias diferentes para verificar o funcionamento do OpenFlow no desenvolvimento do trabalho. Os endereços Ethernet dos elementos foram manipulados de forma que embutissem diferentes dados, fornecendo prefixos, identificadores de VLAN, identificadores de switch e identificadores de portas, tornando o endereço Ethernet visualmente mais estruturado e isolando o tráfego entre as redes virtuais.

No desenvolvimento do cenário com mais de um switch no núcleo da rede, encontrou-se a dificuldade de propagar quadros contendo mensagens ARP vindos de outros switches. Essa questão foi resolvida com o mapeamento estático da associação de endereços dentro do ambiente simulado, porém pode ser implementada em um trabalho futuro.

Através dos testes realizados nos cenários deste trabalho foi possível verificar que o SDN/OpenFlow é um modelo eficaz para configurar e administrar redes de computadores, oferecendo novas possibilidades e ferramentas para definir os fluxos de comunicações entre os elementos, satisfazendo de forma plena os objetivos propostos no item 1.2.

4.1 Sugestões de implementações futuras

Outras possibilidades podem ser exploradas a partir deste trabalho. Todo o material utilizado é de software livre e código aberto, facilitando a reprodução do ambiente e dos cenários estudados. Dentre essas possibilidades de implementação, destacam-se:

- Descoberta automática da topologia física.
- Encaminhamento de mensagens ARP recebidas por outro switch.
- Desenvolvimento de interface gráfica para configuração de regras.

ANEXO A – Cenário introdutório

```
# coding=utf-8
from ryu.base.app_manager import RyuApp
from ryu.controller.ofp_event import EventOFPSwitchFeatures
from ryu.controller.ofp_event import EventOFPPacketIn
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.ofproto.ofproto_v1_2 import OFPG_ANY
from ryu.ofproto.ofproto_v1_2 import OFP_VERSION
from ryu.lib.mac import haddr_to_bin
from ofutil import OFutil
import string


ARP_TYPE=0x806

class L2Switch(RyuApp):
    OFP_VERSIONS = [OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
        # O dicionario dp contem os datapath indexados por seus
            datapath_id
        self.dp = {}
        # O objeto rule_gen serve para facilutar a criacao de
           regras Openflow
```

```
def mac2str(self, mac):
  r = ''
  for c in mac:
    r += '%02X:' % ord(c)
  return r[:-1]

@set_ev_cls(EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    """Handle switch features reply to install table miss
       flow entries."""
    datapath = ev.msg.datapath
    self.dp[datapath.id]=OFutil(datapath)
    [self.dp[datapath.id].install_table_miss(n) for n in
       [0, 1]]
    #print '>>>', dir(datapath)
    print '>>>', datapath.id

@set_ev_cls(EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    """Handle packet in events."""
    msg = ev.msg
    table_id = msg.table_id
    rgen = self.dp[msg.datapath.id]

    # Extract fields
    res = rgen.getFields(msg)
    in_port = res['in_port']
    eth_src = res['eth_src']
    eth_dst = res['eth_dst']

    print "chegou quadro de %s para %s pelo port %d E COM
       ETH_TYPE %d" % (msg.match['eth_src'], msg.match['
       eth_dst'], in_port, msg.match['eth_type'])

    # Install flow entries
```

```
        if table_id == 0 and rgen.valid_port(in_port):
            self.install_src_entry(rgen, in_port, eth_src,
                eth_dst)
            self.install_dst_entry(rgen, in_port, eth_src, msg
                )

        # Flood em portas das vlan
        if in_port % 2:
            ports = [1,3,5,7]
        else:
            ports = [2,4,6,8]
        rgen.floodBroadcast(msg.data, in_port, ports)

    def virtual_mac_check(self, eth_src, mac):
        """Check if the mac is virtual already"""
        if eth_src[:4] == mac[:4]:
            print 'MAC %s jah eh virtual.' % self.mac2str(
                eth_src)
            smac = eth_src[:5]+chr(0)
            smask = chr(255)*5+chr(0)

            #packet in
            match = rgen.create_match(in_port=in_port, eth_src
                =(smac,smask))
            actions = rgen.gotoTable(1)
            rgen.installRule(0, match, actions)

            #packet out
            match = rgen.create_match(in_port=in_port, eth_dst
                =(smac,smask))
            actions = rgen.packetOut(in_port)
            rgen.installRule(1, match, actions)
            return False
        else:
            return True
```

```
def install_src_entry(self, rgen, in_port, eth_src, eth_dst
    ):
    """Create two VLANs based on the in_port value"""
    bmac = self.mac2str(string.join(map(chr, [0xff, 0xff, 0
        xff, 0xff, 0xff, 0xff]),''))
    #bmac = string.join(map(chr, [0xff, 0xff, 0xff, 0xff, 0
        xff, 0xff]),'')
    # VLAN 01 with even ports and VLAN 02 with odd ports
    if in_port % 2:
        lmac = map(chr, [0x02,0x02,0x02,0x22,rgen.datapath.
            id,in_port])
        vlan = 2
    else:
        lmac = map(chr, [0x01,0x01,0x01,0x11,rgen.datapath.
            id,in_port])
        vlan = 1

    mac = string.join(lmac,'')

    if self.virtual_mac_check(eth_src, mac):
        print 'pacote pertence a VLAN 0%d, in_port eh %d. 
            mac original %s e mac virtual %s' % (vlan,
            in_port, self.mac2str(eth_src), self.mac2str(mac)
            )
        #ARP: changes the ARPs source MAC for a virtual MAC
        match = rgen.create_match(eth_type=ARP_TYPE, in_port
            =in_port, eth_src=self.mac2str(eth_src))
        actions = rgen.changeSrc(mac)
        actions = rgen.changeArp(actions, eth_src=mac)
        actions = rgen.gotoTable(1, actions)
        rgen.installRule(0, match, actions)

        #changes the real MAC for a virtual MAC
```

```
            match = rgen.create_match(in_port=in_port, eth_src=
                self.mac2str(eth_src))
            actions = rgen.changeSrc(mac)
            actions = rgen.gotoTable(1, actions)
            rgen.installRule(0, match, actions)

    def install_dst_entry(self, rgen, in_port, eth_src, msg):
        """Forwards the packet to local port or another switch
            """
        #bmac = self.mac2str(string.join(map(chr, [0xff, 0xff,
            0xff, 0xff, 0xff, 0xff]),''))
        bmac = string.join(map(chr, [0xff, 0xff, 0xff, 0xff, 0
            xff, 0xff]),'')
        # VLAN 01 with even ports and VLAN 02 with odd ports
        if in_port % 2:
            lmac = map(chr, [0x02,0x02,0x02,0x22,rgen.datapath.
                id,in_port])
            ports = [1,3,5,7]
        else:
            lmac = map(chr, [0x01,0x01,0x01,0x11,rgen.datapath.
                id,in_port])
            ports = [2,4,6,8]

        mac = string.join(lmac,'')
        rmac = list(eth_src)

        if self.virtual_mac_check(eth_src, mac):
            mac = self.mac2str(mac)

            #ARP: changes the ARP source MAC for host real MAC
            match = rgen.create_match(eth_type=ARP_TYPE, eth_dst
                =mac)
            actions = rgen.changeDest(eth_src)
            actions = rgen.changeArp(actions, eth_dst=mac)
```

```
rgen.floodBroadcast(msg.data, in_port, ports,
    actions)

#changes virtual MAC for the real MAC, forwards the
    packet
match = rgen.create_match(eth_dst=mac)
actions = rgen.changeDest(eth_src)
actions = rgen.packetOut(in_port, actions)
rgen.installRule(1, match, actions)
```

ANEXO B – Ambiente customizado no Mininet

```
from mininet.topo import Topo, Node

class MyTopo(Topo):

    def __init__(self, enable_all = True):
        "Create custom topo."

        # Add default members to class.
        super(MyTopo, self).__init__()

        # Set Node IDs for hosts and switches
        #sw1
        leftSwitch = 1
        leftHost1 = 4
        leftHost2 = 5
        leftHost3 = 6

        #sw2
        centerSwitch = 2
        centerHost1 = 7
        centerHost2 = 8
        centerHost3 = 9

        #sw3
        rightSwitch = 3
        rightHost1 = 10
        rightHost2 = 11
```

```
        rightHost3 = 12

        # Add nodes
        self.add_node(leftSwitch, Node(is_switch=True))
        self.add_node(rightSwitch, Node(is_switch=True))
        self.add_node(centerSwitch, Node(is_switch=True))
        self.add_node(leftHost1, Node(is_switch=False))
        self.add_node(leftHost2, Node(is_switch=False))
        self.add_node(leftHost3, Node(is_switch=False))
        self.add_node(centerHost1, Node(is_switch=False))
        self.add_node(centerHost2, Node(is_switch=False))
        self.add_node(centerHost3, Node(is_switch=False))
        self.add_node(rightHost1, Node(is_switch=False))
        self.add_node(rightHost2, Node(is_switch=False))
        self.add_node(rightHost3, Node(is_switch=False))

        # Add edges
        self.add_edge(leftSwitch, centerSwitch)
        self.add_edge(leftSwitch, rightSwitch)
        self.add_edge(leftSwitch, leftHost1)
        self.add_edge(leftSwitch, leftHost2)
        self.add_edge(leftSwitch, leftHost3)
        self.add_edge(centerSwitch, rightSwitch)
        self.add_edge(centerSwitch, centerHost1)
        self.add_edge(centerSwitch, centerHost2)
        self.add_edge(centerSwitch, centerHost3)
        self.add_edge(rightSwitch, rightHost1)
        self.add_edge(rightSwitch, rightHost2)
        self.add_edge(rightSwitch, rightHost3)

        # Consider all switches and hosts 'on'
        self.enable_all()


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

ANEXO C – Cenário final

```
from ryu.base.app_manager import RyuApp
from ryu.controller.ofp_event import EventOFPSwitchFeatures
from ryu.controller.ofp_event import EventOFPPacketIn
from ryu.controller.handler import set_ev_cls
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.ofproto.ofproto_v1_2 import OFPG_ANY
from ryu.ofproto.ofproto_v1_2 import OFP_VERSION
from ryu.lib.mac import haddr_to_bin
from ryu.lib.packet.packet import Packet
from ofutil import OFutil
from topo import Topologia,Switch
from vlan import rede
import ryu.lib.packet.ethernet
import string


ARP_TYPE=0x806

class L2Switch(RyuApp):
    OFP_VERSIONS = [OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L2Switch, self).__init__(*args, **kwargs)
        # O dicionario dp contem os datapath indexados por seus
            datapath_id
        self.dp = {}
```

```
def mac2str(self, mac):
    r = ''
    for c in mac:
        r += '%02X:' % ord(c)
    return r[:-1]

def __mac_and_mask__(self, mac, length):
    n = 6 - length
    vmac = self.mac2str(mac[:length]+chr(0)*n)
    vmask = self.mac2str(chr(255)*length+chr(0)*n)
    return (vmac, vmask)

def install_initial_rules(self, rgen):
    lmac = map(chr, [0x12,0x34,0x56,0,0,0])
    mac = string.join(lmac,'')
    vmac, vmask = self.__mac_and_mask__(mac, 3)
    #encaminha o pacote pra tabela 1 se o MAC de origem for
        virtual
    match = rgen.create_match(eth_src=(vmac,vmask))
    actions = rgen.gotoTable(1)
    #prioridade baixa, avalia a regra apenas em ultimo caso
    rgen.installRule(0, match, actions, priority=100)

    for vlan in rede.vlans:
        # switch da vlan que corresponde ao datapath
        switch = vlan.get_switch(int(rgen.datapath.id))
        # para cada outro switch da vlan
        for osw in vlan.switches:
            if osw == rgen.datapath.id: continue #ignora se
                for o proprio sw
            print "VLAN %d - switch %d" % (vlan.nome,osw)
            mac = chr(0x12)+chr(0x34)+chr(0x56)+chr(vlan.
                nome)+chr(osw)+chr(0)
```

```
                smac = self.mac2str(chr(0x12)+chr(0x34)+chr(0
                    x56)+chr(0)+chr(osw)+chr(0))
                smask = self.mac2str(chr(255)*3+chr(0)+chr(255)
                    +chr(0))
                topomac, topomask = self.__mac_and_mask__(mac,
                    4)

                # Tabela 1: cria a regra de encaminhamento pro
                    switch de destino "osw"
                port, gw = vlan.get_path(switch, vlan.
                    get_switch(osw))
                match = rgen.create_match(eth_src=(topomac,
                    topomask), eth_dst=(smac, smask))
                actions = rgen.packetOut(port)
                rgen.installRule(1, match, actions, priority
                    =100)

    @set_ev_cls(EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        """Handle switch features reply to install table miss
            flow entries."""
        datapath = ev.msg.datapath
        rgen = OFutil(datapath)
        self.dp[datapath.id] = rgen
        for n in [0,1]:
            [self.dp[datapath.id].install_table_miss(n)]
        #print '>>>', dir(datapath)
        print '>>>', datapath.id, dir(datapath.ports), datapath
            .ports.__class__, datapath.ports
        #depois de criar as tabelas
        self.install_initial_rules(rgen)

    @set_ev_cls(EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        """Handle packet in events."""
```

```
msg = ev.msg
table_id = msg.table_id
rgen = self.dp[msg.datapath.id]

# Extract fields
res = rgen.getFields(msg)
in_port = res['in_port']
eth_src = res['eth_src']
eth_dst = res['eth_dst']
pkt = Packet(msg.buf)

vid = 0
actions = []

for vlan in rede.vlans:
    sw = vlan.get_switch(msg.datapath.id)
    if sw.isPort(in_port):
        # eh esta topologia ,,, vlan.nome
        vid = vlan.nome
        break

if vid > 0:
    print "chegou quadro de %s para %s in_port %d do sw
      %s. com ETH_TYPE %d e vid %d" % (msg.match['
       eth_src'], msg.match['eth_dst'], in_port, sw,
       msg.match['eth_type'], vid)
    # Install flow entries
    if table_id == 0 and rgen.valid_port(in_port):
        actions = self.install_src_entry(rgen, in_port,
            eth_src, vid, pkt)
        self.install_dst_entry(rgen, in_port, eth_src,
           pkt)

    # Flood em portas das vlan
```

```
            print 'Enviando pelas portas', sw.allPorts(), 'do
                switch', msg.datapath.id
            rgen.floodBroadcast(msg.data, in_port, sw.allPorts
                (), actions)

    def install_src_entry(self, rgen, in_port, eth_src, vid,
        pkt):
        """Create two VLANs based on the in_port value"""
        lmac = map(chr, [0x12,0x34,0x56,vid,rgen.datapath.id,
            in_port])
        mac = string.join(lmac,'')

        frame = pkt.get_protocols(ryu.lib.packet.ethernet.
            ethernet)
        frame = frame[0]

        print 'pacote pertence a VLAN 0%d, in_port eh %d. mac
            original %s e mac virtual %s' % (vid, in_port, self.
            mac2str(eth_src), self.mac2str(mac))
        #ARP: changes the ARPs source MAC for a virtual MAC
        match = rgen.create_match(eth_type=ARP_TYPE, in_port=
            in_port, eth_src=self.mac2str(eth_src))
        actions1 = rgen.changeSrc(mac)
        actions1 = rgen.changeArp(actions1, eth_src=mac)
        actions1 = rgen.gotoTable(1, actions1)
        rgen.installRule(0, match, actions1)

        #changes the real MAC for a virtual MAC
        match = rgen.create_match(in_port=in_port, eth_src=self
            .mac2str(eth_src))
        actions2 = rgen.changeSrc(mac)
        actions2 = rgen.gotoTable(1, actions2)
        rgen.installRule(0, match, actions2)

        if frame.ethertype == ARP_TYPE: # ARP
```

```
            return actions1[:-1]
        return actions2[:-1]


    def install_dst_entry(self, rgen, in_port, eth_src, pkt):
        """Forwards the packet to local port or another switch
           """
        lmac = map(chr, [0x12,0x34,0x56,0,rgen.datapath.id,
           in_port])
        mac = self.mac2str(string.join(lmac,''))
        mask = self.mac2str(chr(255)*3 + chr(0) + chr(255)*2)

        frame = pkt.get_protocols(ryu.lib.packet.ethernet.
           ethernet)
        frame = frame[0]

        #ARP: changes the ARP source MAC for host real MAC
        match = rgen.create_match(eth_type=ARP_TYPE, eth_dst=(
           mac, mask))
        actions1 = rgen.changeDest(eth_src)
        actions1 = rgen.changeArp(actions1, eth_dst=eth_src)
        actions1 = rgen.packetOut(in_port, actions1)
        rgen.installRule(1, match, actions1)

        #changes virtual MAC for the real MAC, forwards the
           packet
        match = rgen.create_match(eth_dst=(mac, mask))
        actions2 = rgen.changeDest(eth_src)
        actions2 = rgen.packetOut(in_port, actions2)
        rgen.installRule(1, match, actions2)

        if frame.ethertype == ARP_TYPE: # ARP
            return actions1[:-1]
        return actions2[:-1]
```

Referências Bibliográficas

DÜRR, F. Software-defined networking. Universität Stuttgart, 2012.

FOROUZAN, B. A. *Comunicação de dados e redes de computadores*. [S.l.]: Bookman, 2006.

HEUVEN, P. V. et al. Vlans and gvrp on linux: quickly from specification to prototype using the click router platform. In: *Linux Kongress*. [S.l.: s.n.], 2003.

IEEE, C. S. I. S. f. L.; NETWORKS metropolitan area. Virtual bridged local area networks. 2005.

KUROSE, J.; ROSS, K. *Redes de Computadores e Internet*. [S.l.: s.n.], 2006.

MATIAS, J. et al. An openflow based network virtualization framework for the cloud. In: . IEEE Computer Society, 2011. Disponível em: <http://dx.doi.org/10.1109/CloudCom.2011.104>.

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

MIR, N. F. *Computer and Communication Networks*. [S.l.]: Prentice Hall PTR, 2006.

MYSORE, R. N. et al. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, ACM, 2009. Disponível em: <http://doi.acm.org/10.1145/1594977.1592575>.

ROTHENBERG, C. E. Routeflow: Virtualized ip routing services in openflow networks. 2012.

SPECIFICATION, O. S. Openflow switch specification - version 1.1.0 implemented. 2011. Disponível em: <http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>. Acesso em: 04/03/2013.

STEIN, Y. Sdn: A backward step forward. 2012. Disponível em: <www.rad.com/12/25827>. Acesso em: 20/02/2013.

SZCZEPANEK, A. *VLAN tag transport within a switch*. [S.l.]: Google Patents, 2002.

TANENBAUM, A. S. *Computer Networks*. [S.l.]: Editora Campus, 2003.

VAUGHAN-NICHOLS, S. J. Openflow: The next generation of the network? IEEE, Computer Society, 2011.