Kalvim Scotti

Balanceamento de Tráfego em LANs Ethernet usando uma abordagem SDN/Openflow

São José – SC

Julho / 2014

Kalvim Scotti

Balanceamento de Tráfego em LANs Ethernet 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

Julho / 2014

Resumo

Neste documento será apresentado como as redes locais são compostas, assim como os problemas que ocorrem quando se conecta equipamentos de rede como pontes formando caminhos fechados. Descreve como os protocolos *Spanning Tree* funcionam para resolver este problema e introduz a ideia de protocolos desenvolvidos sobre a abordagem de Rede Definida por Programa para realizar balanceamento de tráfego de uma forma mais eficaz que os protocolos atuais, resolvendo também o problema com caminhos fechados. Será proposto o desenvolvimento de um controlador *Openflow* para realizar o balanceamento de tráfego com a plataforma *Ryu*, assim como será apresentado o modelo do controlador desenvolvido e os resultados obtidos.

Abstract

This document presents how local networks are composed and the problems that occur when network devices are connected as bridges, resulting in closed paths. It describes how the Spanning Tree protocols deal to solve this problem and introduces the idea of protocols developed under the Network Defined by Program approach to perform traffic balancing in a more effective way than the usual protocols, also solving the closed paths issue. It propposes to develop a controller Openflow to perform the traffic balancing throught the Ryu platform, presenting the developed controller model and the obtained results. Describes how the *Spanning Tree* protocols working to resolv this problem and introduce the idea of protocols developed over Software Defined Networking to making traffic balance with a way more effective that the standards protocols resolving too the problems witch the closed ways. Did proposed the development of the Openflow controller to make the traffic balance with the Ryu framework, and did presents the controller model developed and achieved results.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

A necessidade de fazer com que dados fossem compartilhados remotamente, fez com que computadores fossem interconectados, criando assim as redes de computadores. As redes locais são redes privadas formadas por computadores interconectados por meio de um barramento (*Ethernet*) ou conectados entre eles formando um anel (*Token Ring*).

A tecnologia predominante nas redes locais é a *Ethernet*, na qual assume uma topologia em forma de barramento. Nesse modelo, o sinal é transmitido no barramento, que é um meio físico como um fio de cobre formando um único domínio de colisão, dessa forma ficando disponível para todos os computadores. No entanto, apenas o computador que possui o adaptador de rede com o endereço físico contido no cabeçalho do quadro (PDU de enlace) irá extrair o datagrama e entregar à camada de rede. Porém, essa estrutura se torna saturada com o aumento de hosts e um tráfego de rede muito alto, tornando assim, o acesso ao meio de comunicação muito disputado, gerando muitas colisões.

Com a evolução das redes e a necessidade de topologias mais flexíveis e escaláveis, foram desenvolvidos equipamentos como bridges e *switches*. Estes equipamentos comutadores eliminam o domínio único de colisão isolando suas portas, formando assim, um domínio de colisão para cada interface. Para isso, o quadro recebido será processado pelo equipamento e encaminhado à porta na qual o host de destino se encontra, conforme sua tabela de comutação que é formada de forma automática e dinâmica. Então, se um endereço de destino não consta na tabela de comutação, o *Switch* repassa o quadro em todas as suas interfaces, dessa forma, para cada quadro recebido será armazenado na tabela de comutação a porta em que o endereço físico do remetente se encontra.

Comutadores se tornaram muito importantes para desempenho da rede, sendo eles a conexão central da rede local. Notamos então, que em redes de médio a grande porte se faz necessário a utilização de muitos *switches*. Nestes casos é possível organizar estes equipamentos em diferentes topologias, afim de ter caminhos redundantes para evitar falhas, expandir a rede, ter alta disponibilidade, realizar isolamentos na rede e fazendo tudo isso sempre com auto

desempenho. Porém, ao formar caminhos redundantes irá formar *loops* entre *switches*, que são indesejados pois geram problemas como alocação de todos os enlaces e o travamento da rede. Estes problemas ocorrem pelo próprio funcionamento padrão dos equipamentos e dos protocolos de rede, que mandam mensagens em *broadcast* gerando encaminhamentos das cópias do frame, passando de um *switch* para outro em um *loop*. Dessa forma, os quadros percorrerão os *switches* de forma cíclica, multiplicando-se de forma exponencial.

Para evitar o problema de caminhos redundantes entre *switches*, utiliza-se protocolos do tipo *Spanning Tree* (STP), que atuam controlando os *switches* determinando a porta a ser desabilitada e assim evitar *loops*. Para isso o protocolo STP envia BPDUs (*Bridge* PDUs) entre os *switches*, que de acordo com o *Bridge ID* escolhe quem será o raiz, determina uma topologia de acordo com o ID dos demais e o custo dos enlaces, de modo que não formem nenhum *loop* entre eles. Apesar da utilização destes protocolos resolver o problema de quadros circulando indefinidamente pelos *switches*, para resolver o problema acabam diminuindo a otimização dos recursos físicos.

Uma abordagem interessante para tratar estes tipos cenários é o *Software-Defined Networking* (SDN). Com SDN é possível criar um controlador para equipamentos de rede e programar todo o seu funcionamento, para isso, se abstrai a camada de *hardware* e faz o controle da rede de forma centralizada na camada lógica de controle do SDN (FOUNDATION, 2012). A camada de controle interage diretamente com as camadas de aplicação e de *hardware*. Por esse motivo, pode-se criar ambientes de rede de forma customizada, atendendo especificamente a necessidade do administrador de rede. Tendo essa possibilidade de se programar o funcionamento da rede na camada de controle, pode-se criar regras de encaminhamento para que o fluxo de dados assuma algum determinado caminho, afim de realizar balanceamento de tráfego e evitar o problema de *loops* em caminhos redundantes.

Um modelo para SDN é o *Openflow*, que foi proposto na Universidade de *Stanford* nos Estados Unidos (MCKEOWN N. ANDERSON et al., 2008). Ele foi a primeira proposta de interface de comunicação entre as camadas de controle e de *hardware*. *Openflow* usa informações contidas em cabeçalhos de protocolos de pacotes recebidos para identificar fluxos de pacotes, e assim aplicar regras de encaminhamento definidas estaticamente ou dinamicamente por um controlador. O controlador pode residir em um outro equipamento, possibilitando o controle centralizado da rede. Com isso o administrador de rede pode definir como os determinados fluxos de pacotes devem ser encaminhados pelos equipamentos de rede. Esse modelo possibilita realizar balanceamento de tráfego e tratar o problema com enlaces redundantes de uma só vez. Para isso, precisam-se identificar os determinados fluxos de pacotes e assumir diferentes cami-

nhos pelos *switches*, de forma a utilizar todos os recursos disponíveis e evitar encaminhamentos cíclicos.

1.1 Objetivo Geral

O objetivo deste trabalho é desenvolver um controlador *Openflow* capaz de programar os *switches* de uma rede local, afim de balancear o tráfego de pacotes, de forma que:

- Distribua os fluxos de pacotes por todos os enlaces de forma dinâmica;
- Trate problemas de caminhos redundantes;
- Utilize todos os enlaces.

1.2 Justificativa

Atualmente, com o crescimento das redes locais, necessita-se de um controle da rede muito mais eficaz e seguro. Portanto, nos cenários de médio a grande porte, a realização de balanceamento de tráfego é imprescindível. Dessa forma, tirando os gargalos da rede e aproveitando os recursos físicos, com certeza resulta em um impacto muito positivo para o desempenho da rede e consequentemente na produtividade da instituição.

De fato, o que nos impulsiona a tratar estas questões é ter acesso a um protocolo que nos possibilita programar o funcionamento dos *Switches*. Sendo que, sem essa possibilidade estamos limitados aos mecanismos disponibilizados pelos fabricantes dos equipamentos, não havendo muito poder de controle e customização. Sendo assim, *Openflow*/SDN possibilita desenvolver uma implementação de controle de rede para fazer o devido balanceamento de tráfego e otimização de recursos físicos.

2 *Fundamentação Teórica*

2.1 Conectando LANs

Algumas topologias de nível de enlace são formadas com intenções de tornar a rede tolerante a falhas, realizar balanceamento de carga, e evitar gargalos. No entanto, ao configurar essas topologias com switches geralmente se formarão caminhos fechados (loops), necessitando a utilização de mecanismos que evitam os problemas de broadcast resultantes dos loops entre switches e bridges.

2.1.1 Caminhos Redundantes

Caminhos redundantes são topologias que possuem diferentes caminhos para chegar em um mesmo destino. Estes caminhos irão formar os caminhos fechados, que são formados pela conexão dos equipamentos em uma forma de anel. Geralmente estes caminhos redundantes são formados intencionalmente, conforme figura 2.1-A, mas frequentemente acontecem de forma acidental como na figura 2.1-B, quando alguém sem conhecimento conecta dois pontos de rede com o mesmo cabo, sendo que são pontos do mesmo switch (DONAHUE, 2007).

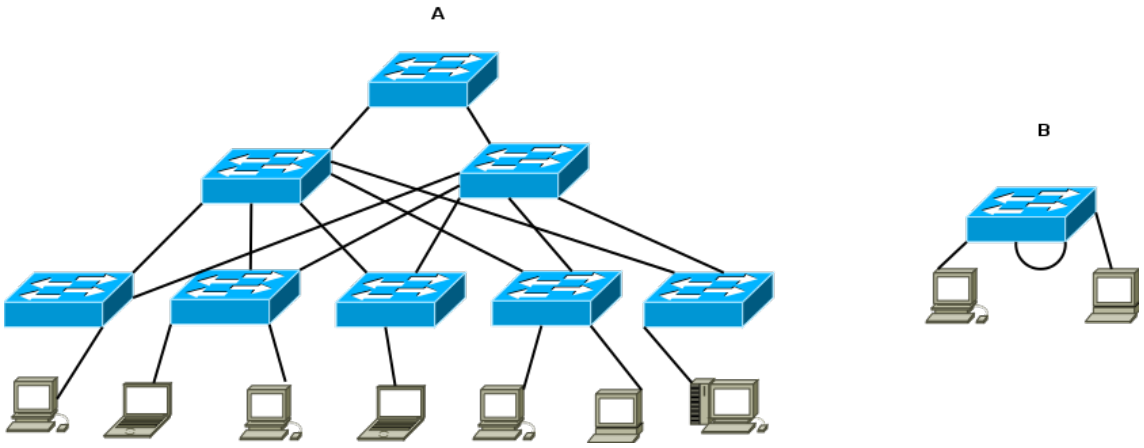

Figura 2.1: A:Topologia com caminhos redundantes em forma de espinha. B: Loop entre portas do mesmo switch ocasionado acidentalmente

No entanto, os loops são indesejados, eles geram problemas como alocação de todos os enlaces, ocasionando o travamento da rede. Este é um problema que acontece devido ao próprio funcionamento dos bridges, switches e dos protocolos de rede em si, que ao encaminhar mandam mensagaens em broadcast geram cópias dos quadros de um swtich para outro em um loop (PERLMAN, 2000). Dessa forma, os quadros percorrerão os switches de forma cíclica, multiplicando-se de maneira indefinida.

Um exemplo de transmissão de quadro em broadcast se verifica em mensagens ARP. Em uma rede local, esse protocolo tem por finalidade identificar o endereço MAC da interface de rede de um equipamento que possui um determinado endereço IP. Ele é necessário para que um host em uma rede local descubra o endereço MAC de outro host com que precisa se comunicar e cujo endereço IP é conhecido. Assim, o host de origem transmite uma mensagem ARP em broadcast, perguntando qual é o endereço MAC do host que possui o endereço IP em questão.

Para compreender como acontece o esgotamento dos enlaces da rede nos caminhos fechados, vamos analisar um cenário simples como o da figura 2.2.

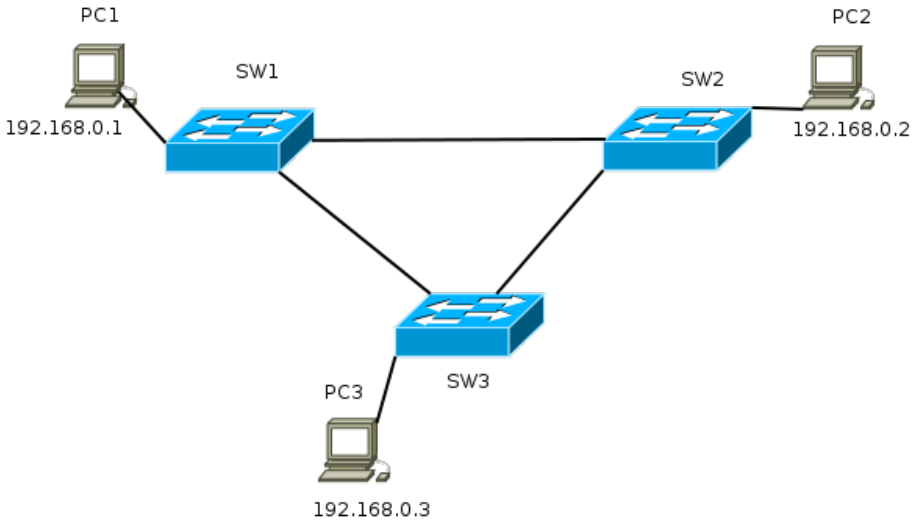

Figura 2.2: Cenário com caminhos redundantes, ocasionando um loop na rede

Supomos que o PC1 irá encaminhar uma mensagem para o PC3 e não conhece a informação do endereço físico do mesmo. Então antes de qualquer coisa, ele encaminhará uma mensagem ARP em broadcast perguntando *who as 192.168.0.3? Tell 192.168.0.1*. O SW1 vai receber a mensagem e irá encaminhar para todos os nós, pois é uma mensagem broadcast. Da mesma forma irá acontecer com SW2 e SW3. Como os switches estão interligados, a mensagem ARP será replicada continuamente de um para o outro, gerando um tráfego indefinido entre os switches e dessa forma até os enlaces ficarem totalmente ocupados por este tráfego, ocorrendo o travamento da rede.

Para evitar este problema, foram desenvolvidos protocolos que alteram a topologia da rede de forma lógica evitando os caminhos redundantes (FOROUZAN, 2007). Os protocolos do tipo *Spanning Tree (STP, RSTP, MSTP)* são os protocolos padrões para tratar este problema.

2.1.2 Spanning Tree Protocol (STP, RSTP, MSTP)

O STP é um protocolo especificado pela norma IEEE 802.1d - 1998. O princípio do protocolo é formar uma topologia lógica sem loops. Portanto, é necessário que todos os equipamentos Bridges e Switches estejam com o mecanismo ativado para que o protocolo realize a comunicação entre os equipamentos e defina uma topologia livre de caminhos fechados. Para isso o protocolo STP envia suas mensagens de configuração entre os switches, que de acordo com o ID do switch escolhe quem será o switch raiz (PERLMAN RADIA,2000). A partir disso, será determinada uma topologia de acordo com o ID dos demais e o custo dos enlaces, de modo que não formem nenhum loop entre eles. Apesar da utilização destes protocolos resolver o problema de quadros sendo encaminhados indefinidamente pelos switches, acaba desperdiçando recursos físicos obrigando-os a estarem desabilitados. Outro fator negativo é de não prover uma convergência muito rápida caso haja alguma alteração na topologia física, podendo demorar de 30 até 50 segundos.

O Bridge ID é o parâmetro identificador do switch que pode ser definido pelo administrador. Ele é composto pela prioridade e o MAC address. Caso não seja definida a prioridade, o endereço MAC será o determinante.

As etapas do STP são as seguintes:

- O switch com menor ID se torna o Root;
- A porta de cada switch com menor custo até o Root será determinada como porta raiz (RP);
- Em cada segmento será determinado como portas designadas (DP) as portas com menor custo para chegar ao Root. Caso o custo seja o mesmo, a porta designada será a porta do switch que tenha o menor ID;
- As portas designadas e raizes serão marcadas como portas de encaminhamento, as demais portas serão portas bloqueadas (NDP).

Cada equipamento conterá a informação de switch raiz pelo seu bridge ID conforme figura 2.3. As taxas de transmissão dos enlaces também são utilizadas como forma de priorizar os melhores caminhos.

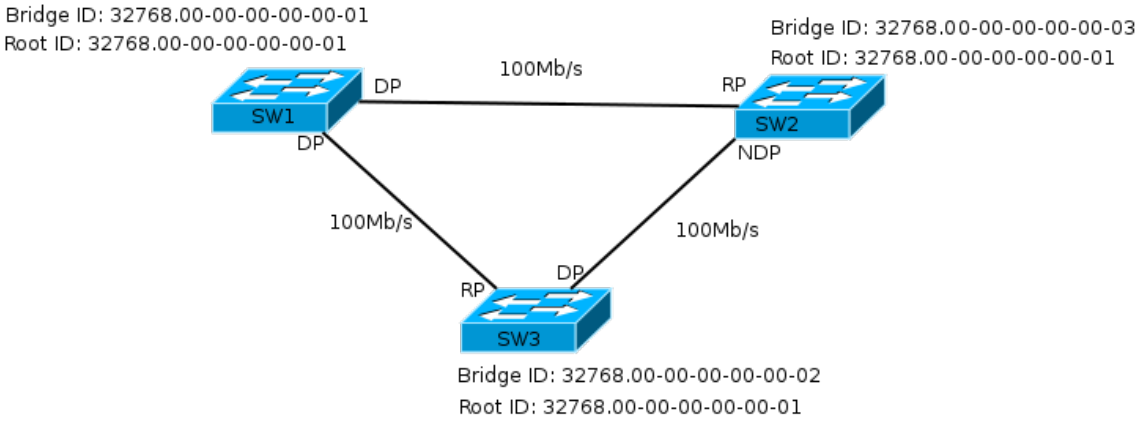

Figura 2.3: STP configurando a topologia lógica em árvore

A PDU do protocolo STP é chamada de BPDU (Bridge Protocol Data Unit). Para o funcionamento do protocolo, o STP utiliza o endereço multicast, enviandos os BPDUs para o endereço 01:80:C2:00:00:00 (802.1D, 2004). Por default os BPDUs são enviados a cada 2 segundos. São definidos 3 tipos de BPDUs:

- **Configuration cBPDU (CBPDU)** – usado para a execução do STP;
- **Topology Change Notification (TCN) BPDU** – anuncia mudança na topologia da rede;
- **Topology Change Notification Acknowledgment (TCA)** – confirmação do switch raiz ao receber um TCN de algum switch.

Estas PDUs definem o funcionamento do protocolo desde as determinações de funções das portas, assim gerando a topologia da rede, até realizando as alterações de topologias quando ocorrem mudanças físicas. Portanto, quando outro switch é conectado na rede, a porta pode permanecer em estado *blocking* se for verificado que formaria um *loop* na rede.

As BPDUs de notificação de alteração da topologia (TCN) são usadas para informar outros switches das mudanças ocorridas. TCNs são enviadas na rede por um switch e encaminhadas até o switch raiz. Após a recepção da TCN, o switch raiz irá definir uma *flag* de mudança de topologia em suas BPDUs. Esta flag é propagada para todos os outros switches aprenderem a nova topologia e atualizarem suas entradas da tabela de encaminhamento.

No STP são definidos 6 tipos de estados para as portas dos switches. São eles:

- ***Blocking*** – é a porta que formaria o *loop* se não estivesse bloqueada. Não encaminha quadros ou recebe quadros, mas pode mudar seu estado para forwarding se ocorrer alguma falha em algum outro ponto da rede;

- ***Listening*** – processa BPDUs e aguarda novas informações que possam fazer seu estado voltar para *blocking*. Não alimenta a tabela MAC e não encaminha quadros;
- ***Learning*** – alimenta a tabela MAC e não encaminha quadros;
- ***Forwarding*** – opera normalmente. Encaminha e recebe quadros;
- ***Initializing*** – apenas inicializa a porta, podendou deixar ela ativa ou desativada pelo administrador;
- ***Disabled*** – não encaminha quadros e nem participa da configuração do *spanning tree*.

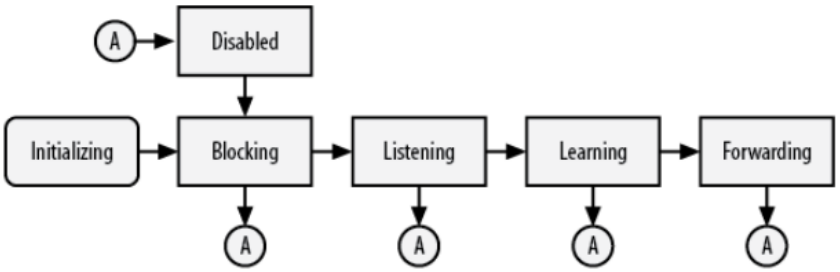

Figura 2.4: Estados das portas (DONAHUE, 2007)

De acordo com a figura 2.4 as portas que executam o STP irão assumir uma sequência de estados até finalmente assumir o estado estabelecido pelo protocolo. O estado em que a porta assume está identificado pelo círculo com a letra ”A”. A porta inicialmente está desativada para o STP, inicializando a interface do switch ela entrará no estado *blocking*, se não se manter nesse estado mudará para o estado *listening*, podendo mudar para o estado anterior ou assumir o estado *learning* e por fim poderá assumir o estado que se manterá em funcionamento normal, *forwarding*.

O RSTP *(Rapid Spanning Tree Protocol)* é uma alteração do protocolo STP definido pela norma IEEE 802.1w – 2001. Em 2004 o RSTP foi incorporado ao IEEE 802.1D e o STP original se tonou obsoleto (802.1D, 2004). Essa nova versão proporciona uma convergência mais rápida quando há uma alteração na topologia física. Para isso são introduzidas no protocolo novas formas de convergência e novas funções para as portas. O STP demorava até 50 segundos para responder a uma alteração física, pois o tinha que aguardar expirar o *max age* time (20s), passar para o estado *listening* (15s) e *learning* (15s), para então estar em *forwarding*, conforme (WIKIPEDIA, 2014). No entanto, o RSTP demora até 6 segundos, correspondentes a 3 *Hello-Times* de 2 segundos.

A tabela 2.1 mostra as informações de custo utilizadas pelo STP e o RSTP para as determinadas taxas de transmissão dos enlaces. Quanto maior a taxa de transmissão, menor será o custo do enlace e portanto, maior será a prioridade.

Data rate	STP Cost (802.1D-1998)	RSTP Cost (802.1W-2001)
4 Mbit/s	250	5,000,000
10 Mbit/s	100	2,000,000
16 Mbit/s	62	1,250,000
100 Mbit/s	19	200,000
1 Gbit/s	4	20,000
2 Gbit/s	3	10,000
10 Gbit/s	2	2,000

Tabela 2.1: Tabela de custo no STP e RSTP (WIKIPEDIA, 2014)

O problema também deve ser tratado em redes virtuais IEEE 802.1Q, em que os cenários ficam ainda mais complicados, sendo necessário topologias diferentes para tratar as diferentes Vlans. Para isso, foi desenvolvido o MSTP (*Multiple Spanning Tree Protocol*) definida pelo padrão IEEE 802.1s. Essa nova versão estende o protocolo RSTP, para realizar diferentes Spanning Trees para grupos de Vlans ou para cada Vlan.

2.2 SDN - Software-Defined Networking

Devido à evolução das aplicações de redes e o aumento da demanda por alto desempenho, as redes se tornaram muito complexas, e assim, devido a falta de escalabilidade nestes casos, tornando-se inadequadas para e o atendimento de empresas provedoras de serviços de internet e sua larga escala de usuários [5].

Para atender estes cenários que necessitam de escalabilidade, poder de controle e flexibilidade na rede, foi proposto o conceito de SDN *(Software-Defined Networking)*, que permite configurar a rede de forma centralizada e personalizada (FOUNDATION, 2012), sendo que o próprio administrador define a programação para o seu funcionamento.

2.2.1 Conceituação e comparação com abordagem usual

Os equipamentos utilizados em redes locais como roteadores, switches e access points, funcionam de forma independente e autônoma. Eles não possuem nenhum controle centralizado que comande suas operações, mas para que a rede funcione é preciso que cada equipamento funcione conforme os protocolos pre-estabelecidos.

Switches encaminham os quadros de acordo com endereçamento MAC dos hosts contido no cabeçalho do quadro. Eles também devem obedecer o protocolo STP que define uma topologia lógica sem caminhos fechados, sem a qual, a comutação de quadros causaria anomalias na rede, com quadros enviados em broadcast indefinidamente entre os caminhos fechados.

Roteadores encaminham datagramas IP de acordo com a sua tabela de roteamento e os endereços IP contidos nos cabeçalhos dos datagramas IP. Além disso, precisam seguir um protocolo de roteamento para descobrir dinamicamente rotas dentro da rede. Essas rotas podem se alterar conforme mudanças na topologia da rede.

Percebe-se então que o funcionamento dos equipamentos convencionais depende de protocolos padronizados. Além disso, os mecanismos de encaminhamento desses equipamentos não são abertos, no sentido de que não apresentam uma interface de programação que possibilite estendê-los. O fato de os protocolos serem fechados inibe a experimentação de novas técnicas de encaminhamento, pois impossibilita que sejam testados em equipamentos de rede reais. Essa limitação motivou o surgimento do conceito de Rede Definida por Software (SDN – Software Defined Network).

Em uma rede definida por software, os equipamentos de rede apresentam interfaces de programação que tornam possível adicionar ou modificar funcionalidade à rede, o que possibilita inclusive a programação do seu funcionamento de forma centralizada, separando conceitualmente a estrutura dos equipamentos em uma camada de controle e outra camada de encaminhamento de dados. Na camada de controle podem ser definidas todas as regras para o tratamento dos fluxos de pacotes, ficando a camada de encaminhamento responsável por comutar pacotes eficientemente. Com isso, podem-se desacoplar as funcionalidades de rede embutidas na camada de controle, tais como protocolos e técnicas de encaminhamento, do *hardware* do equipamento.

Como outra consequência desse modelo, potencializa-se e flexibiliza-se a administração da rede, pois o controle e a configuração de cada equipamento podem ser feitos de forma centralizada e sob medida para as aplicações que usarão a rede. Como vemos na Figura 6, a arquitetura SDN é dividida em 3 camadas. Na camada inferior reside o *hardware* de rede, que opera um

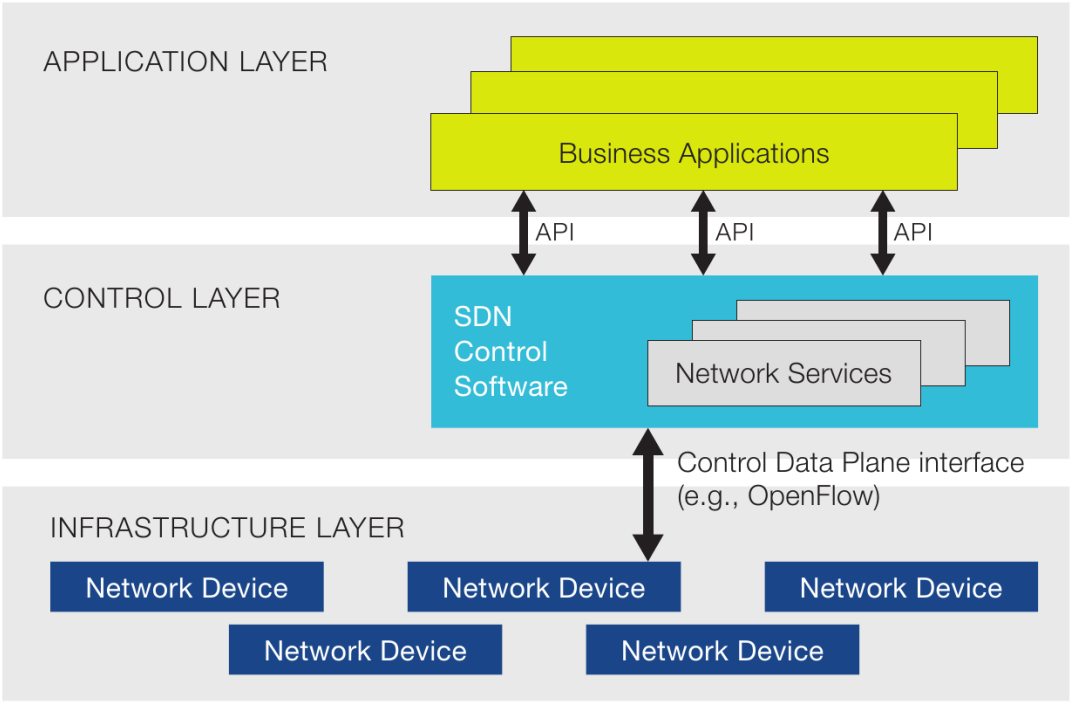

Figura 2.5: Arquitetura SDN em camadas (FOUNDATION, 2012)

módulo SDN para receber as "ordens" da camada imediatamente superior e determinar seu funcionamento. Esta camada intermediária é a parte de controle da arquitetura, que neste modelo é abstraída do hardware e pode até mesmo estar em outro equipamento. Nessa camada está toda a inteligência da rede. O controlador se comunica com a camada de aplicação através de APIs de comunicação, de forma transparente. A camada de aplicação é uma interface de gerenciamento do controlador, que em conjunto com as APIs de comunicação permite acesso os parâmetros de configuração da camada controle (Open Networking Foundation, 2012).

2.2.2 OpenFlow

Openflow é um protocolo proposto para implementar SDN. Ele foi criado em 2008 na Universidade de Stanford, onde seu desenvolvimento continua até hoje. A proposta inicial dos pesquisadores de Stanford era criar a possibilidade de pesquisadores executarem seus experimentos de procolos de redes (MCKEOWN N. ANDERSON et al., 2008). No entanto, o projeto tomou proporções bem maiores, indo dos arredores das universidades aos *data-centers* de grandes empresas.

O Openflow é um protocolo que possibilita a implementação de SDN em diferentes roteadores e switches. Ele atua tanto na camada de controle (*control path*) como na camada física do switch (*datapath*), fazendo a interface de comunicação entre eles. Define fluxos de mensagens

de acordo com as regras definidas por alguns parâmetros tais como os campos de cabeçalhos dos pacotes, como edereço MAC, endereço IP, portas TCP/UDP. Para estes fluxos são definidos ações como *forward, drop, rewrite,* ou "enviar para o controlador" (WANG et al., 2011).

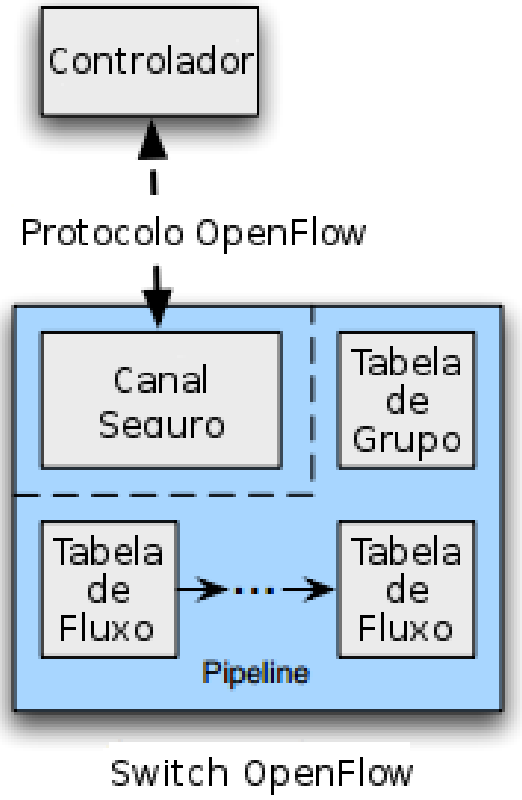

Figura 2.6: Modelo Openflow Switch (SPECIFICATION, 2011)

O *datapath* do Openflow Switch possuí uma tabela de encaminhamento de fluxos com os campos que definem os fluxos dos pacotes e uma ação para cada entrada. Essas regras são carregadas no *datapath* de acordo com as instruções enviadas pelo canal de comunicação Openflow a partir do controlador. Se o switch receber algum pacote de um padrão ainda não recebido, qual não possui nenhuma entrada na sua tabela, nesse caso, enviará o pacote para o controlador. O controlador definirá o que deve ser feito com o pacote. Ele pode não fazer nada, ou pode enviar instruções de entrada na tabela de fluxo do switch, para que trate o novo fluxos.

De acordo com a definição do artigo publicado por Mckeown (MCKEOWN N. ANDERSON et al., 2008), o Openflow Switch pode ser separado em pelo menos três partes conforme figura 2.6:

- Uma tabela de fluxos, que instrui o switch como proceder para cada entrada de fluxo;
- Um canal seguro que conecta o controlador remoto com o switch;

- Protocolo Openflow, que provê um método aberto e padronizado de comunicação com o switch.

2.2.3 Tabelas OpenFlow

O Openflow Switch pode ter uma ou mais tabelas de fluxos e uma tabela de grupo. O controlador pode adicionar, remover ou atualizar as entradas nas tabelas de fluxos dos switches.

Na tabela de fluxo existem regras que definem as fluxos. Cada entrada da tabela de fluxo contém o campo de comparação *match fields*, contadores e um conjunto de instruções para aplicar aos fluxos classificados conforme tabela abaixo.

Match fields	Contadores	Instruções

Tabela 2.2: Principais campos de uma entrada de fluxo na tabela de fluxo (SPECIFICATION, 2011)

- ***Match fields*:** para comparar os pacotes e classificá-los. Será verificado a porta de entrada e os cabeçalhos do pacote. Também pode ser comparado algum metadado de uma tabela anterior;
- **Contadores:** para atualizar os pacotes determinados;
- **Instruções:** para modificar um conjunto de ações ou o processo de *pipeline*.

As comparações de campos nos fluxos podem percorrer todas as tabelas do switch se for necessário. Caso não seja classificado em nenhuma tabela, será encaminhada ao controlador, para então o controlador poder passar alguma nova entrada para a tabela de fluxos do switch ou não fazer nada.

Openflow *pipeline* é composto pelo conjunto de tabelas de fluxos do controlador, como mostra a figura 2.7. Os processos de *pipeline* definem como os pacotes interagem com as tabelas de fluxos. Caso for desejado, pode ser definida apenas uma tabela de fluxo. Nesse caso o processo de *pipeline* se torna bem simplificado.

As tabelas são sequencialmente numeradas, começando em 0. O processo de *pipeline* sempre inicia com a tabela 0 e pode ser encaminhadas para outras tabelas dependendo da decisão da primeira tabela. Portanto algumas entradas nas tabelas podem apenas encaminhar os pacotes para outra tabela. Desse modo, os pacotes podem ser diretamente encaminhados por diferentes tabelas, sem a necessidade de ter alguma ação, sendo processado quando chegar na última

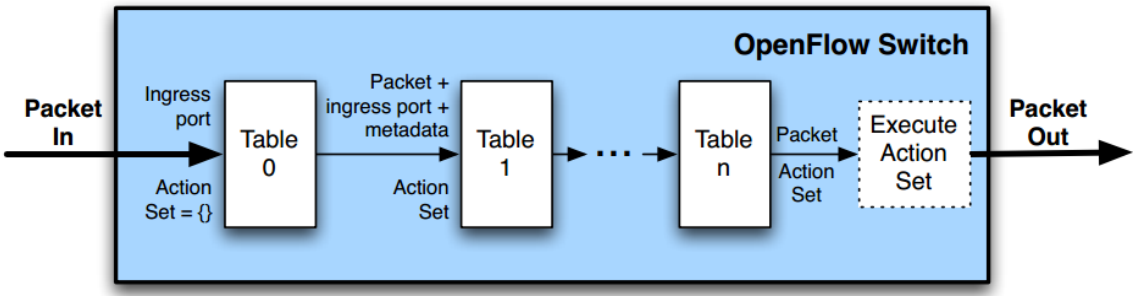

Figura 2.7: Pacotes são comparados e processados por múltiplas tabelas de fluxos (SPECIFICATION, 2011)

tabela. Caso o pacote não for classificado em nenhuma entrada da tabela, será encaminhado ao controlador pelo canal de comunicação. A tabela pode determinar que o pacote deve continuar mesmo sem ter nenhuma entrada para o mesmo, quando isso acontecer, o pacote será encaminhado para a próxima tabela.

As tabelas de grupos são definidos por conjuntos de entradas, em que os determinados fluxos serão processados. São definidos quatro tipos de grupos:

- ***All:*** Executa todas as entradas do grupo. Geralmente usado nos encaminhamentos *broadcast* e *multicast*.
- ***Select:*** Executa uma entrada do grupo. O pacote é enviado para apenas uma entrada do grupo de acordo com um algorítimo de seleção externo. Se ocorre alguma falha em uma porta específica da entrada selecionada, o switch vai encaminhar o fluxo pelas outras portas do grupo. Reduzindo problemas de quedas de links.
- ***Indirect:*** Executa uma entrada definida para esse grupo. Permite que múltiplos fluxos ou grupos apontem para um mesmo identificador de grupo, permitindo uma convergência mais rápida e eficiente.
- ***Fast Failover:*** Executa a primeira entrada ativa. Permite que o switch altere a rota de encaminhamento sem necessitar passar pelo controlador.

Toda entrada das tabelas contém algum valor, ou vários, para ser comparado com as informações dos pacotes. Além dos cabeçalhos dos pacotes, metadados podem ser usados para passar informações de uma tabela para a outra. Assim, quando um pacote é processado por uma tabela, ela pode inserir os metadados para serem comparados com as entradas da tabela 2.3.

Quando o switch Openflow recebe um pacote, ele vai realizar as funções apresentadas no fluxograma da figura 2.8. O pacote inicia sendo processado pela tabela de fluxos 0. Se não

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 ToS IPv4
Porta de Origem do TCP/UDP/SCTP do ICMP
Porta de Destino do TCP/UDP/SCTP do ICMP

Tabela 2.3: Campos do pacote que pode ser comparadado com o *match field* da entrada de fluxo (SPECIFICATION, 2011)

tiver nenhuma entrada correspondente irá realizar as seguintes determinações de acordo com a configuração da tabela: envia para o controlador; descarta; ou encaminha para a próxima tabela. Mas se combinar com alguma entrada da tabela, executa as instruções determinadas pela entrada,que podem ser: atualizar o conjunto de ações; atualizar o os campos de comparação do pacote; atualizar os metadados. Nesse ponto o contador correspondente é atualizado. Após executar as instruções, se não tiver nenhum encaminhamento para outra tabela serão executadas as ações determinadas pela tabela de fluxo. Mas caso for encaminhado para uma outra tabela, então o pacote realizará todo o processo novamente até não tiver mais nenhum encaminhamento para outra tabela e então executará as suas ações.

Toda entrada possui instruções a serem executadas quando o pacote coincide com seus campos de comparação. As instruções podem realizar alterações nos pacotes, em suas ações e processo *pipeline*. As instruções suportadas são as seguintes:

- ***Apply-Actions action(s):*** Aplica as ações imediatamente, sem alterar o conjunto de ações. Pode ser usada para modificar um pacote entre duas tabelas ou executar múltiplas ações do mesmo tipo;
- ***Clear-Actions:*** Remove todo o conjunto de ações imediatamente;
- ***Write-Actions action(s):*** Insere ações ao conjunto de ações existente. Caso já exista a ação inserida, ela será sobrescrevida;

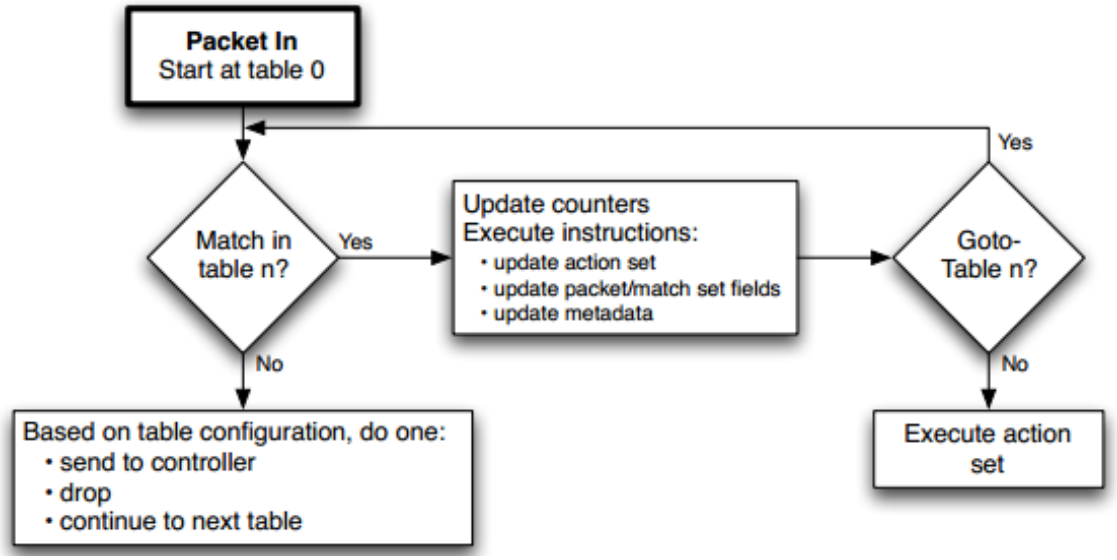

Figura 2.8: Fluxograma demonstrando o fluxo dos pacotes pelo switch Openflow (SPECIFICATION, 2011)

- ***Write-Metadata metadata/mask****:* Insere a máscara de metadado no campo do correspondente;
- ***Go-Table next-table-id****:* Informa a próxima tabela no processo pipeline. O id da tabela deve ser maior que o id corrente.

O switch pode rejeitar alguma entrada de fluxo caso não suporte a instrução associada. Nesse caso o switch deve retornar um erro de fluxo não suportado. As tabelas de fluxo podem não suportar todas as comparações e todas as instruções.

O conjunto de ações está associado a cada pacote. As ações são definidas ao passar pelas tabelas de fluxos recebendo as instruções *Write-actions* e *Clear-actions*. As ações serão executadas quando o processo *pipeline* chegar ao fim, para isso, a entrada de fluxo não deve possuir a instrução *Go-Table*. O máximo de ações que pode haver no conjunto é uma de cada tipo. Porém quando é necessário executar mais de uma ação do mesmo tipo é necessário utilizar a instrução *Apply-Action*.

O switch não precisa suportar todas as ações, mas algumas ações são necessárias e existem outras que são opcionais, são elas:

- ***Output:*** Encaminha um pacote para uma porta especifica. Os switches OpenFlow devem ser capazes de suportar o encaminhamento para portas físicas e virtuais;
- ***Drop:*** Não existe uma ação específica para descartar os pacotes. Eles serão descartados quando não houver nenhuma ação para ser executada;

- ***Group:*** Encaminha um pacote através de um grupo específico;
- ***Set-Queue (opcional):*** 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;
- ***Push-Tag/Pop-Tag (opcional):*** Permite remover e insirir tags nos cabeçalhos dos pacotes. Dessa forma, ajuda na integração com outros protocolos existentes na rede;
- ***Set-Field (opcional):*** Permite modificar os campos dos cabeçalhos do pacote.

2.2.4 Canal Openflow

O canal Openflow é a interface de comunicação que conecta o switch Openflow ao Controlador. Através desse canal de comunicação o controlador configura e gerencia o switch, recebe eventos do switch e encaminha pacotes.

A interface entre o datapath e o canal Openflow é uma implementação específica, portanto todas as mensagens entre o canal Openflow deve ser formatadas de acordo com o protocolo Openflow. O canal de comunicação é seguro, pois pode ser usado com criptografia TLS (*Transport Layer Security*). No entanto, pode ser usado diretamente sobre TCP (*Transmission Control Protocol*).

O protocolo Openflow suporta três tipos de mensagens, *controller-to-switch, asynchronous,* e *symmetric*, cada uma delas tendo diversas subdivisões (SPECIFICATION, 2011).

As mensagens do tipo *controller-to-switch*, são iniciadas pelo controlador para configurar e verificar o estada do do switch, sendo que não necessitam de resposta do switch.

Já as mensagens do tipo *asynchronous* são iniciadas pelo switch sem nenhuma solicitação do controlador. Elas servem para informar o controlador sobre os eventos da rede como chegada de um pacote, algum erro ou alterações no estado do switch.

As mensagens *symmetric* são enviadas tanto pelo controlador quanto pelo switch e não possuem solicitações, são apenas mensagens de controle.

2.2.5 Controlador Openflow

O controlador Openflow está disponível em diferentes plataformas. As quatro mais populares são, Beacon (Java), Maestro (Java), NOX (C++/Python) e Trema (C/Ruby). Estas plataformas foram desenvolvidas em projetos para pesquisa e educação.

Além das plataformas citadas acima, existem outras surgindo no mesmo seguimento. A plataforma que conheci que chamou atenção é o framework Ryu. Ele permite trabalhar com diferentes protocolos SDN, como, Netconf, OF-config, Openflow (versões 1.0, 1.2, 1.3 e 1.4). É deselvolvido na linguagem Python, fator pelo qual tive interesse em trabalhar com o mesmo.

3 *Proposta e resultados*

A proposta deste trabalho é desenvolver um controlador Openflow/SDN para realizar balanceamento de tráfego em uma rede local. Criando um controlador SDN se torna possível definir o funcionamento dos equipamentos de rede de forma bem específica, e com isso balancear o tráfego da rede de forma a aproveitar todos os recursos da rede. O cenário foi desenvolvido e testato em um laboratório virtual a partir da ferramenta de simulação de rede Mininet.

O balanceamento de tráfego foi desenvolvido identificando os padrões dos fluxos de encaminhamentos de pacotes e definindo os caminhos que os diferentes fluxos percorrerão na rede. Os caminhos dos fluxos serão estabelecidos de forma a utilizar de forma equânime os enlaces no núcleo da rede local, dessa forma, aproveitando todos os enlaces da rede.

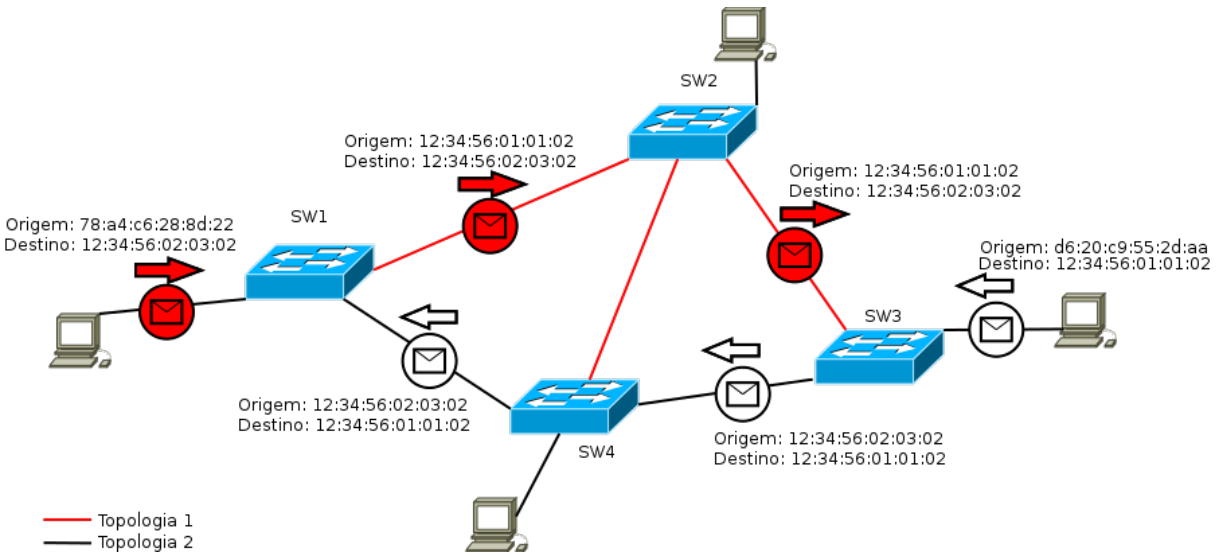

Figura 3.1: Os switches manipulam os MAC address nos quadros que trafegam na rede local para determinarem os caminhos lógicos

Conforme a figura 3.1 os fluxos são identificados de acordo com os endereços MAC de origem e destino, sendo encaminhados por caminhos diferentes na rede para cada fluxo de transmissão entre uma origem e destino. Os quadros que estão no núcleo da rede possuem o endereço MAC manipulado para que se determine o caminho que percorrerão. Para cada nova transmissão será sorteada uma topologia que interligará os dois destinos envolvidos, sendo que

a transmissão a partir de um host poderá percorrer uma topologia diferente da transmissão de resposta do outro *host*.

Este trabalho pretende também ser uma alternativa ao protocolo STP e suas variações, pois o balanceamento tráfego desenvolvido define os caminhos dos fluxos de quadros independentemente da existência de caminhos fechados entre os switches, de forma que logicamente não exista caminhos fechados, sendo desnecessário a utilização de *spanning-tree*.

Não foi desenvolvido nenhuma descoberta de rede automática nesta implementação. O desenvolvimento do controlador Openflow foi realizada de acordo com o *layout* físico da rede, sendo que para qualquer alteração física na rede se faz necessária a reconfiguração do controlador Openflow.

3.1 Modelo Openflow

O controlador baseado em Openflow irá funcionar assumindo os seguintes pressupostos:

- O controlador deve conhecer exatamente como a rede está interconectada, incluindo os switches e as portas que interligam uns aos outros;
- O controlador predefinirá todas as topologias lógicas possíveis, isentas de caminhos fechados, de acordo com a *layout* físico da rede, assim incrementando a utilização dos enlaces da rede;
- A escolha por uma topologia virtual predefinida é feita de forma aleatória pelo controlador quando um quadro é encaminhado pela primeira vez a um switch;
- A comunicação entre o controlador e os switches se faz a partir de uma vlan específica;
- São utilizados endereços MAC virtuais entre os switches com o objetivo de encaminhar e tratar os quadros conforme a topologia utilizada.

A estrutura do endereço MAC virtual possui informações para o tratamento do quadro entre os switches da rede. De acordo com informações contidas nos três primeiros octetos do MAC address o quadro poderá ser identidicado e encaminhado por um caminho predefinido.

```
MAC virtual:  12:34:56 : 01 : 02    :     03
                |  |  |    |    |          |
```

```
prefix:topo_id:datapath_id:port_sw
Legenda: Corresponde a topologia 1, originário de um host conectado ao
         switch 2, em sua interface nº 3.
```

O primeiro octeto, contando da direita para esquerda, indica a porta do switch qual o host se encontra conectado; o segundo octeto identifica o switch em que o host está conectado; o terceiro octeto representa a topologia que o pacote irá trafegar dentro da rede; os últimos 3 octetos são apenas um prefixo sem nenhuma relevância.

O Openflow utiliza tabelas e regras para poder definir a configuração e funcionamento da rede. Uma tabela é composta por uma ou diversas regras. Uma regra possui três campos, o *match field*, contadores e instruções. As instruções são compostas por ações a serem executadas afim de tratar o quadro. O *match field* identifica selecionando o quadro para que receba suas instruções compostas de ações definidas pelo administrador da rede. Podem ser utilizadas diversas tabelas em conjunto, chegando a utilizar grupos de tabelas se for o caso. Uma regra pode identificar um quadro e encaminhá-lo a outra tabela e assim receber instruções de diferentes tabelas.

3.1.1 O controlador

No caso do controlador desenvolvido foram utilizadas duas tabelas. A primeira é a tabela 0, que contém regras que determinam a origem de quadros entrantes no switch. Já a tabela 1 contém regras que encaminham os quadros para os determinados destinos. Algumas regras da tabela 0 encaminham o quadro para a tabela 1 para poder determinar o destino.

Conforme mostra o fluxograma da figura 3.2, primeiramente o controlador cria as duas tabelas quando o switch inicia, logo após, cria as regras iniciais que definiem as topologias virtuais predefinidas. Estas regras são inseridas na tabela 1 e definem o encaminhamento de cada switch para cada destino de acordo com todas as topologias lógicas.

O controlador receberá algum quadro quando este não corresponder com nenhuma regra das tabelas, neste caso, o controlador insere uma nova regra nas tabelas referente a este novo encaminhamento. Caso o quadro recebido possuir um endereço MAC real, o controlador escolherá uma topologia lógica aleatóriamente e definirá um MAC virtual com base na topologia escolhida, no switch de acesso e na porta do switch em que está conectado. Criará uma regra na tabela 0 pare fazer a alteração do MAC de origem para o virtual e criará outra regra na tabela 1 para alterar o MAC virtual para o real e encaminhar o quadro pela interface de encaminhamento.

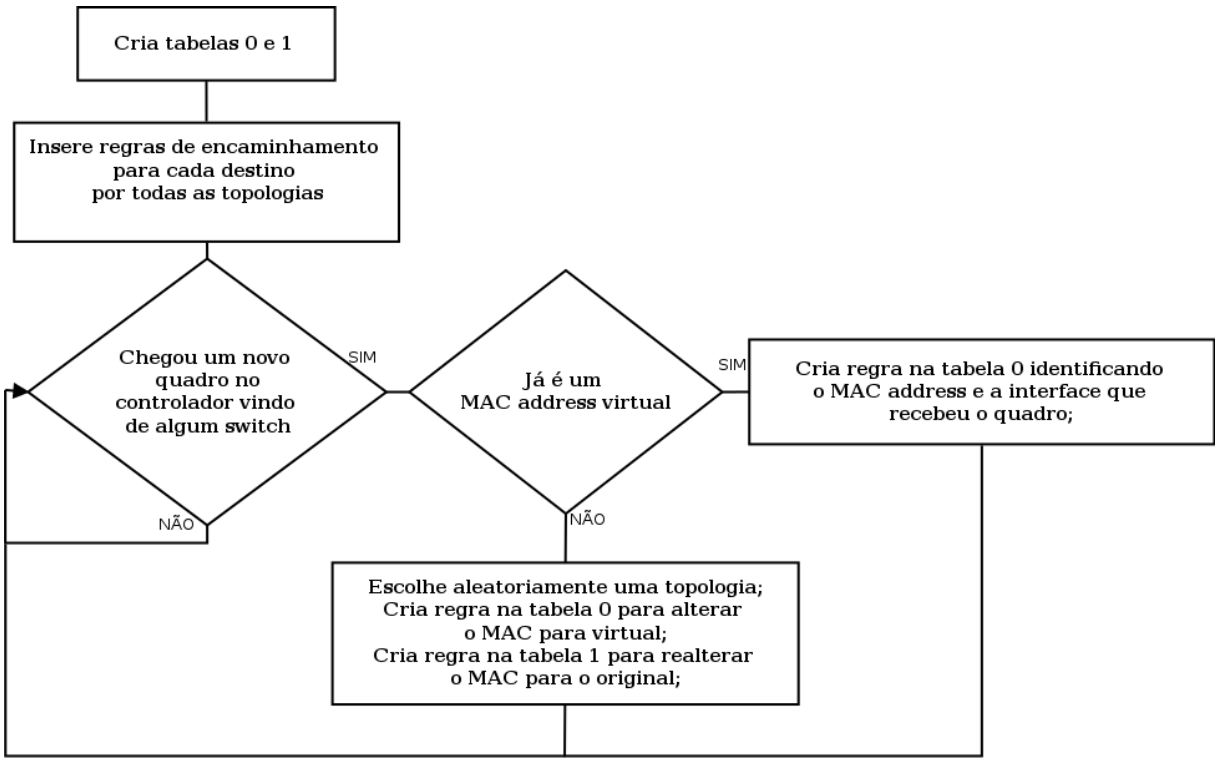

Figura 3.2: Fluxograma de ações do controlador desenvolvido

Caso o quadro já possuir um endereço MAC virtual de origem, o controlador insere uma regra na tabela 0 para encaminhar futuros quadros esta mesma identificação para serem encaminhados pela tabela 1 e insere uma regra na tabela 1 com encaminhamento pela porta de entrada para quadros em que o destino corresponde a este MAC address virtual. Se nenhuma regra encaminhar o quadro, o controlador envia um comando ao switch para encaminhar o quadro por todas suas interfaces, com exceção da porta por qual o quadro foi recebido.

Nas mensagens ARP também foi necessário alterar a informação do MAC address contido dentro do pacote para que o protocolo ARP funcionasse corretamente.

3.1.2 O switch

O switch, quando iniciado, se conectará ao controlador e receberá as configurações das suas tabelas Openflow. Neste momento serão definidas todas as regras de encaminhamento correspondentes as diferentes topologias lógicas por quais se realizará o balanceamento de tráfego. Quando receber algum quadro que não case com nenhuma regra de suas tabelas, envia este ao controlador para que defina o encaminhamento e atualize suas tabelas.

As regras que determinam as topologias, ou seja, o funcionamento da rede entre os switches, são predefinidas pelo controlador. Já as regras que envolvem os hosts fazendo o mapeamento

pelos MAC address virtuais, são criadas por demanda. Sempre que houver um novo host com MAC address diferente, qual não possui nenhuma tratativa correspondente nas tabelas, os switches encaminharão o pacote ao controlador que tratará de inserir uma nova regra nas tabelas do switch para determinar os encaminhamentos correspondente ao novo host.

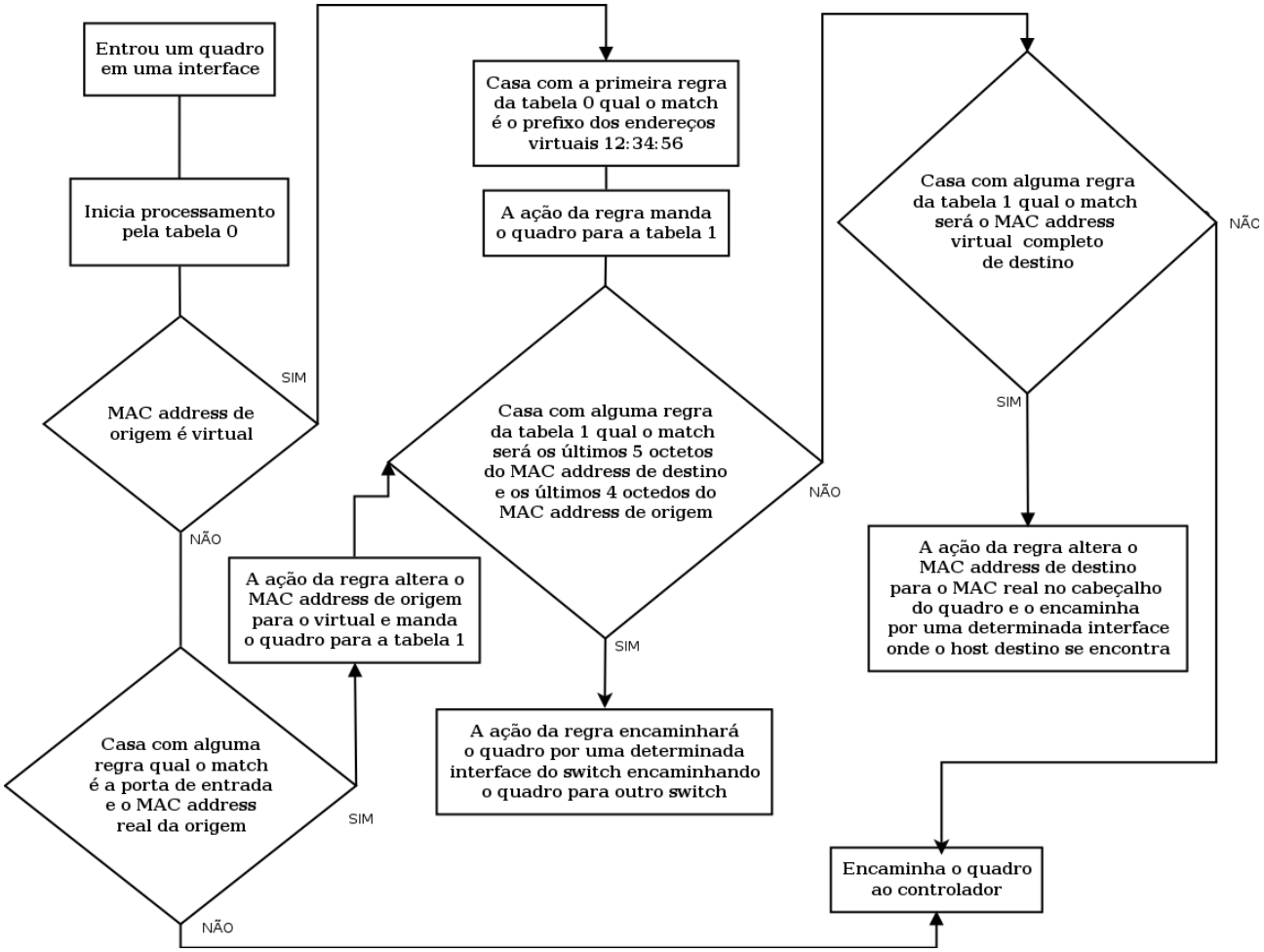

Figura 3.3: Fluxograma de ações do switch openflow

O fluxo de um quadro que é processado por um switch está descrito no fluxograma acima. Ele percorrerá as tabelas do switch, que dependendo da localização da origem e destino e da topologia, irá definir diferentes ações para o determinado quadro. No próximo item será descrito como as regras das tabelas openflow do switch irão definir o mapeamento dos quadros de acordo com os MAC address de origem e destino.

3.1.3 Mapeamento com MAC address virtual

O mapeamento por endereços MAC é feito utilizando as regras das tabelas Openflow. Os endereços MAC dos hosts são manipulados quando o quadro entra na rede. Esta manipulação é feita com a finalidade de o endereço MAC conter a informação de que porta do switch o host

se encontra, em qual switch ele está conectado e por qual topologia lógica o quadro poderá ser encaminhado.

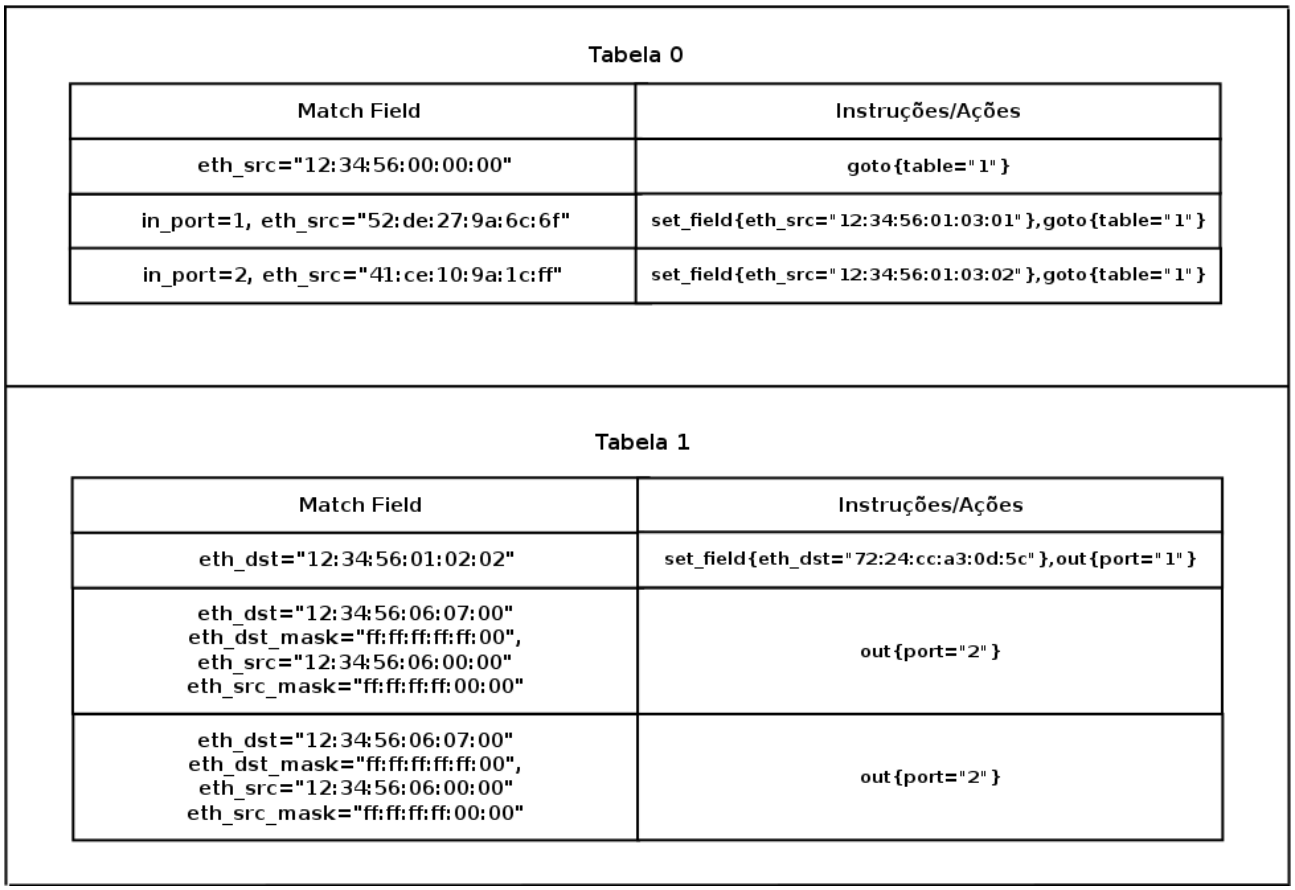

Tabela 0

Match Field	Instruções/Ações
eth_src="12:34:56:00:00:00"	goto{table="1"}
in_port=1, eth_src="52:de:27:9a:6c:6f"	set_field{eth_src="12:34:56:01:03:01"},goto{table="1"}
in_port=2, eth_src="41:ce:10:9a:1c:ff"	set_field{eth_src="12:34:56:01:03:02"},goto{table="1"}

Tabela 1

Match Field	Instruções/Ações
eth_dst="12:34:56:01:02:02"	set_field{eth_dst="72:24:cc:a3:0d:5c"},out{port="1"}
eth_dst="12:34:56:06:07:00" eth_dst_mask="ff:ff:ff:ff:ff:00", eth_src="12:34:56:06:00:00" eth_src_mask="ff:ff:ff:ff:00:00"	out{port="2"}
eth_dst="12:34:56:06:07:00" eth_dst_mask="ff:ff:ff:ff:ff:00", eth_src="12:34:56:06:00:00" eth_src_mask="ff:ff:ff:ff:00:00"	out{port="2"}

Figura 3.4: Exemplo de utilização das tabelas 0 e 1.

As regras das tabelas possuem em seu campo match máscaras de MAC address virtuais, conforme mostra a figura 3.4. Uma máscara de origem utilizada por exemplo, pode ser 12:34:56:01:00:00. Então, se um quadro que contém um MAC de origem igual a 12:34:56:01:03:03 for processado pela tabela, irá casar com a regra e lhe será atribuído uma ação para ser encaminhado pela tabela 1. A tabela 1 conterá regras com *matches* que irão classificar de acordo com o endereço MAC de destino, por exemplo, o match conterá uma máscara 12:34:56:02:00:00 e instrução com ação para encaminhar pela porta 04 do switch. Portanto, se o mesmo quadro possuir um endereço MAC no campo de destino igual a 12:34:56:02:01:04 ele será pŕocessado pela regra e encaminhado pela porta 04 do switch.

Na tabela 0 são inseridas regras para o quadro sofrer a manipulação do endereço MAC sempre quando chegar no switch a partir do mesmo host de origem, esta regra irá conter no campo match a informação do MAC real e a porta do switch qual o host está conectado. Ao classificar o quadro pelo match, as instruções da regra conterá ações a serem executadas. A primeira ação será para alterar o MAC dentro do cabeçalho do quadro para o MAC virtual

que foi definido de acordo com a topologia escolhida pelo controlador. E a próxima ação será mandar o quadro ser processado também pela tabela 1, com a finalidade de encontrar alguma regra de encaminhamento com base no MAC de destino.

Os encaminhamentos dos quadros são feitos pelas regras que estão na tabela 1. Toda regra da tabela 1 contém a ação de encaminhar o quadro por uma determinada interface. Algumas regras, que determinam o encaminhamento para um determinado *host*, possuem a ação de alteração do MAC virtual para o MAC real da interface de rede do *host*.

3.2 Detalhes da implentação

O controlador escolhido para realizar o desenvolvimento do projeto foi o Ryu, o qual pode ser trabalhado na linguagem Python. Em conjunto, foi utilizado o simulador de rede Mininet, que executa máquinas virtuais, interligadas por *softswitches* desenvolvidos pelo CPqD da Telebras no âmbito projeto RouteFlow.

Para esta implementação foram instalados em um servidor Ubuntu, o *framework* Ryu e o simulador Mininet.

3.2.1 Criando o laboratório com Mininet

Para montar o cenário virtual é preciso configurar um arquivo texto do mininet. Esta configuração é específicamente para definir uma topologia na rede, tendo em vista que o mininet já possui algumas topologias padrões para executar diretamente. No caso deste trabalho, como necessita-se customizar o cenário com frequência, definir a topologia em um arquivo é a melhor opção.

Para criar uma rede customizada no mininet deve-se especificar cada *host* e switch, além de criar todas as interconecções entre eles.

```
def __init__( self, enable_all = True ):
    super( MyTopo, self ).__init__()
    # Set Node IDs for hosts and switches
    leftHost = 1
    leftSwitch = 2
    rightSwitch = 3
    rightHost = 4
    # Add nodes
```

```
        self.add_node( leftSwitch, Node( is_switch=True ) )
        self.add_node( rightSwitch, Node( is_switch=True ) )
        self.add_node( leftHost, Node( is_switch=False ) )
        self.add_node( rightHost, Node( is_switch=False ) )
        # Add edges
        self.add_edge( leftHost, leftSwitch )# h1 em eth0 de s2
        self.add_edge( leftSwitch, rightSwitch )# eth1 de s2 em eth0 de s3
        self.add_edge( rightSwitch, rightHost )# eth0 de s3 em eth1 de h4
        self.enable_all()
```

3.2.2 Desenvolvendo o controlador Openflow com framework Ryu

O *framework* Ryu possibilita desenvolver um controlador utilizando Openflow na linguagem Python. Com algumas funções básicas do Ryu foi criada a primeira versão do protocolo utilizado no primeiro cenário com dois switches.

O controlador possui dois métodos fundamentais que são acionados quando ocorrem os eventos em que o switch e controlador interagem. O primeiro é o *switch_features_handler()* que é o evento correspondente ao momento em que quando o switch inicia e se comunica com o controlador. Este método irá criar as duas tabelas Openflow e configurar as regras iniciais correspondente as topologias lógicas.

```
@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
   # cria as duas tabelas
   for n in [0,1]:
     self.dp[datapath.id].install_table_miss(n)
   print '>>>', datapath.id
   # configura as regras iniciais correspondente as topologias logicas
   self.install_initial_rules(rgen)
```

Outro evento importante é quando o controlador recebe algum novo quadro para que o switch não encontrou nenhuma regra de encaminhamento. Esse evento é tratado pelo método *_packet_in_handler()*. Ele basicamente irá criar novas regras de mapeamento de MAC virtual

referente o novo quadro recebido, o que inclui escolher aleatóriamente uma topologia lógica predefinida por onde transitar esse quadro. Isso é feito somente se for um quadro que terá endereço MAC real. Por fim, irá encaminhar o quadro por todas as portas do switch se não for encaminhado por nenhuma regra.

```
@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 eh um objeto OFutil especifico para este datapath
    rgen = self.dp[msg.datapath.id]

    # Obtem os campos de interesse relativos ao pacote
    res = rgen.getFields(msg)
    in_port = res['in_port']
    eth_src = res['eth_src']
    eth_dst = res['eth_dst']

    str_mac_topo = ord(msg.match['eth_src'][10]) - 48

    # Escolhe topologia aletóriamente
    # se estiver na tabela 0
    if table_id == 1 and str_mac_topo != 0:
        str_mac_topo -= 1
        topo = self.topologias[str_mac_topo]
    else:
        topo = random.choice(self.topologias)

    # Instala regras de entrada e de encaminhamento
    actions = []
    if table_id == 0 and rgen.valid_port(in_port):
        print "installing new source mac received from port", in_port
        pkt = Packet(msg.buf)
        actions = self.install_src_entry(rgen, in_port, topo.nome, eth_src, pkt)
        self.install_dst_entry(rgen, in_port, topo.nome, eth_src, pkt)
    # Se tabelas nao encaminham o pacote, entao faz um flood
    switch = topo.get_switch(rgen.datapath.id)
```

```
rgen.floodBroadcast(msg.data, in_port, switch.allPorts(), actions)
```

A baixo um exemplo de como foi feito para utilizar as funções do Ryu para o Openflow definir as ações que os switches devem realizar para alterar o endereço MAC do pacote:

```
def changeSrc(self, new_mac, actions=None):
  if not actions: actions = []
  set_src = self.parser.OFPMatchField.make(self.ofproto.OXM_OF_ETH_SRC, new_mac)
  actions.append(self.parser.OFPActionSetField(set_src))
  return actions

def changeDest(self, new_mac, actions=None):
  if not actions: actions = []
  set_dst = self.parser.OFPMatchField.make(self.ofproto.OXM_OF_ETH_DST, new_mac)
  actions.append(self.parser.OFPActionSetField(set_dst))
  return actions
```

3.2.3 1º Cenário

Durante o aprendizado das ferramentas e do *framework*, o modelo de laboratório foi se alterando conforme a evolução. De início foi montado um laboratório com dois switches tendo cada um mais um *host* conectado.

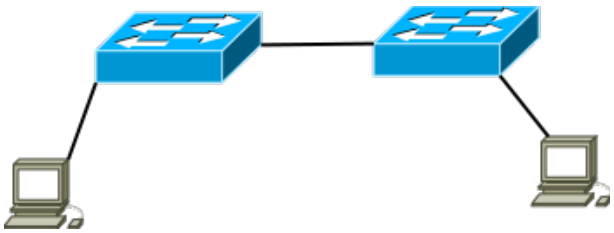

Figura 3.5: primeiro cenário

No primeiro cenário foi criado um laboratório com dois switches a fim de ter o primeiro contato com as plataformas utilizadas e também realizar o estudo da linguagem Python em conjunto com o *framework* Ryu para a utilização das funções do Openflow.

O desafio nesse momento foi definir como os switches iriam se comportar para não gerarem tráfego indefinido entre eles quando interconectados, tendo em vista que não utilizei os protocolos do tipo *spanning-tree* para tratar esse comportamento.

O problema com tráfego indefinido não deveria ocorrer em um cenário com dois switches, mas isto ocorreu devido ao funcionamento do simulador de rede Mininet. Então foi verificado a

existência de *bugs* tanto do mininet quanto do controlador, que demandaram um bastante tempo até achar uma solução.

Neste cenário, com apenas dois switches, temos apenas uma topologia, então resta apenas manipular os endereços MAC. São criadas regras que fazem os switches substituir o endereço MAC de origem, no cabeçalho do quadro, para um MAC virtual com um prefixo predefinido. É também inserida uma regra na tabela 1 gravando o MAC address real para o switch fazer a tradução novamente do MAC virtual quando for entregar um pacote a este host, isso se faz necessário pois a rede local não irá conhecer o seu endereço real mas apenas o seu MAC virtual. O mesmo acontece para mensagens do protocolo ARP.

Então basicamente os switches recebem os pacotes e alteram o endereço MAC para transmitir ao destino, sendo que no núcleo da rede apenas trafega mensagens com endereços virtuais e quando o switch entrega o quadro ao host ele realtera o endereço MAC para o endereço original.

Conforme as capturas dos quadros a baixo, realizadas com a ferramente de rede *tcpdump*, o funcionamento da rede ficou de acordo com o esperado para este cenário.

```
 root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth3
tcpdump: WARNING: s3-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
20:04:37.572808 12:34:56:01:04:04 > 9a:6f:c9:29:63:0a, ethertype IPv4 (0x0800),
   length 98: 10.0.0.6 > 10.0.0.5: ICMP echo request, id 6667, seq 1, length 64
20:04:37.572837 9a:6f:c9:29:63:0a > 12:34:56:01:04:04, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.6: ICMP echo reply, id 6667, seq 1, length 64
```

Na captura dos dois quadros acima, realizada pela ferramenta *tcpdump*, notamos um *ping* sendo recebido interface do switch que conecta o *host* na rede. De acordo com as informações do cabeçalho do quadro, o endereço MAC da origem está no formato virtual para encaminhamento dentro da rede. Já o endereço MAC do destino, o qual está conectado na interface do switch que está sendo capturado o *ping*, está no formado original.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth2
tcpdump: WARNING: s3-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
20:04:37.570593 12:34:56:01:04:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
```

```
    length 98: 10.0.0.6 > 10.0.0.5: ICMP echo request, id 6667, seq 1, length 64
20:04:37.576914 12:34:56:01:03:03 > 12:34:56:01:04:04, ethertype IPv4 (0x0800),
    length 98: 10.0.0.5 > 10.0.0.6: ICMP echo reply, id 6667, seq 1, length 64
```

Já na interface do switch por onde estão chegando os quadros, notamos que os endereços MAC da origem e destino estão com o formato de MAC virtual, conforme mostra a captura acima.

A baixo está a captura do *ping* na interface que conecta o *host* que gerou a requisição. Percebe-se que o quadro ainda está com a informação do MAC address original.

```
 root@openflowvm:~/of12softswitch# tcpdump -lne -i s4-eth4
tcpdump: WARNING: s4-eth4: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s4-eth4, link-type EN10MB (Ethernet), capture size 65535 bytes
20:04:37.570169 ee:f5:cf:1c:b7:85 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.6 > 10.0.0.5: ICMP echo request, id 6667, seq 1, length 64
20:04:37.579890 12:34:56:01:03:03 > ee:f5:cf:1c:b7:85, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.6: ICMP echo reply, id 6667, seq 1, length 64
```

3.2.4 2º Cenário

No segundo cenário foi inserido um switch entre os outros dois já existentes ligando-os em uma linha. Nesse cenário o *layout* ainda continua sem *loop*, apenas para garantir o funcionamento da rede com mapeamento com endereços MAC virtuais.

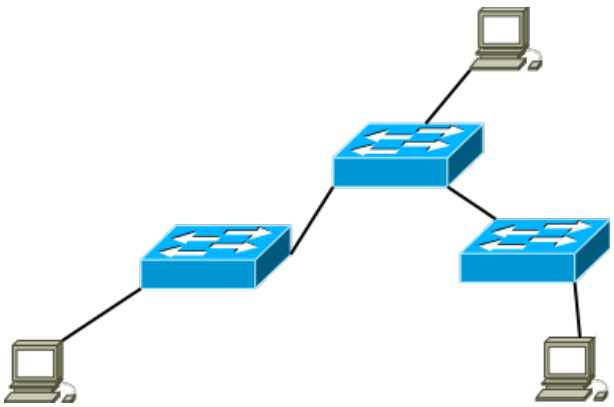

Figura 3.6: segundo cenário

O funcionamento se manteve igual ao cenário anterior, pois é possível apenas uma topologia lógica. Sendo, que apenas foi inserido mais um switch e a rede de manteve em funcionamento.

3.2.5 3º Cenário

Com três switches interligados em *loop* formando um *layout* em que existe a problemática dos caminhos fechados entre switches.

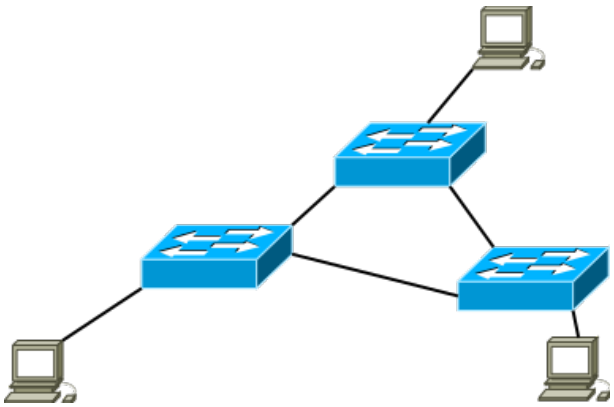

Figura 3.7: terceiro cenário

A solução para esse cenário foi criar as regras predefinidas no controlador que definem as topologias funcionando como uma tabela de roteamento.

Com os três switches em *loop* foram criadas três topologias lógicas como se eles estivessem conectados em uma linha. Para configurar estas 3 topologias lógicas foram identificadas as interfaces dos switches que são *uplinks* e para cada topologia foram definidas as interfaces que poderiam ser utilizadas. Portanto, um switch estará nas três topologias ao mesmo tempo mas em posições diferentes.

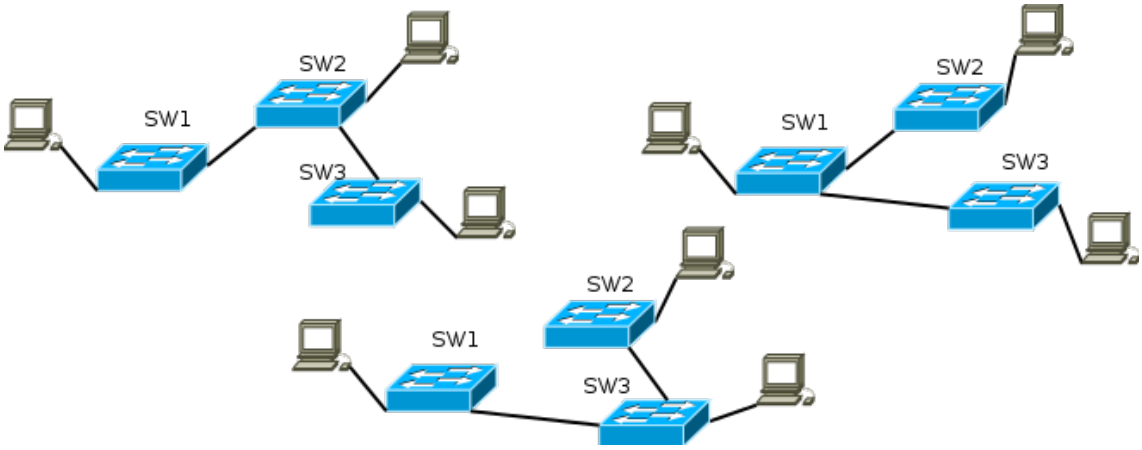

Figura 3.8: Topologias Vituais

Para exemplificar pode-se dizer que na topologia 1 o switch estará em uma ponta transmitindo apenas pelo *uplink* da interface 01, na topologia 2 estará na outra ponta transmitindo também por apenas 1 *uplink*, já na topologia 3 o switch estará no centro pelo outro caminho transmitindo pelas duas interfaces de *uplink*.

Foram realizados testes adicionando 3 *hosts* em cada switch. Analisando os enlaces entre os switches foram realizados testes de *pings* entre todos os *hosts* da rede. Verificou-se a utilização de todos os enlaces. Não ocorreu nenhuma perda de pacote e se observou o funcionamento perfeito das topologias lógicas utilizando MAC address virtuais.

Abaixo está as capturas de um *ping* realizado neste cenário do *host* 10.0.0.8 para o 10.0.0.5:

```
 root@openflowvm:~/of12softswitch# tcpdump -lne -i s7-eth4
tcpdump: WARNING: s7-eth4: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s7-eth4, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.818669 d2:22:d2:30:07:8a > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 1, length 64
20:33:52.833348 12:34:56:01:03:03 > d2:22:d2:30:07:8a, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.8: ICMP echo reply, id 7039, seq 1, length 64
```

De acordo com a captura acima, o *host* que está requisitanto o *ping* está conectado na interface eth4 do switch s7 e seu endereço MAC original é d2:22:d2:30:07:8a. Abaixo verifica-se que o mesmo quadro está saindo do switch pela interface eth3 com o endereço MAC do *host* alterado para 12:34:56:03:07:04. E na interface eth2 do switch verifica-se que a resposta de 10.0.0.5 está chegando para ser entregue ao *host*, confirmando que a requisção do *ping* está indo por um caminho e resposta está voltando por outro.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s7-eth3
tcpdump: WARNING: s7-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s7-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.819449 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 1, length 64
20:33:53.826064 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s7-eth2
tcpdump: WARNING: s7-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s7-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.828546 12:34:56:01:03:03 > 12:34:56:03:07:04, ethertype IPv4 (0x0800),
```

```
   length 98: 10.0.0.5 > 10.0.0.8: ICMP echo reply, id 7039, seq 1, length 64
20:33:53.840059 12:34:56:01:03:03 > 12:34:56:03:07:04, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.8: ICMP echo reply, id 7039, seq 2, length 64
```

O switch s4 recebe o quadro de requisição pela interface eth3 e o encaminha pela interface eth2 ao switch s3. Nota-se que neste switch trafega apenas a requisição não contendo nenhum quadro da resposta do ping.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s4-eth3
tcpdump: WARNING: s4-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s4-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.819461 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 1, length 64
20:33:53.826084 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s4-eth2
tcpdump: WARNING: s4-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s4-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.820104 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 1, length 64
20:33:53.829275 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 2, length 64
```

De acordo com as capturas a baixo, o switch s3 recebe o quadro de requisição do ping pela interface eth2 e entrega o quadro pela interface eth3 ao host com o MAC address original 9a:6f:c9:29:63:0a. A resposta, sendo o tráfego no sentido 10.0.0.5 - 10.0.0.8, sai pela interface eth1 do switch.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth3
tcpdump: WARNING: s3-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.823156 12:34:56:03:07:04 > 9a:6f:c9:29:63:0a, ethertype IPv4 (0x0800),
```

```
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 1, length 64
20:33:52.823187 9a:6f:c9:29:63:0a > 12:34:56:03:07:04, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.8: ICMP echo reply, id 7039, seq 1, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth1
listening on s3-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.828534 12:34:56:01:03:03 > 12:34:56:03:07:04, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.8: ICMP echo reply, id 7039, seq 1, length 64
20:33:53.840036 12:34:56:01:03:03 > 12:34:56:03:07:04, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.8: ICMP echo reply, id 7039, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth2
listening on s3-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
20:33:52.820116 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 1, length 64
20:33:53.829295 12:34:56:03:07:04 > 12:34:56:01:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.8 > 10.0.0.5: ICMP echo request, id 7039, seq 2, length 64
```

3.2.6 4º Cenário

Neste cenário foi adicionado o quarto switch no centro da rede dividindo-a formando três caminhos fechados. Com esse *layout* físico foram definidas 6 topologias virtuais.

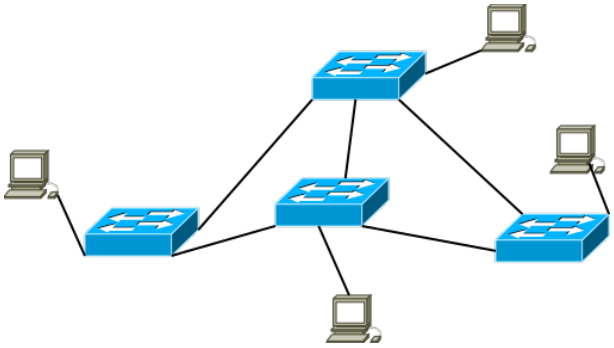

Figura 3.9: Quarto cenário

O controlador funcionou corretamente assumindo este novo cenário. Verificou-se a utilização de todas as topologias lógicas aproveitando todos recursos de enlaces presente na rede.

Utilizando o analisador de tráfego *tcpdump*, pode-se verificar o correto funcionamento das manipulações de endereços MAC para utilização das topologias lógicas.

Conforme captura abaixo, a requisição do *ping* é encaminhado pelo *host* e entra no switch

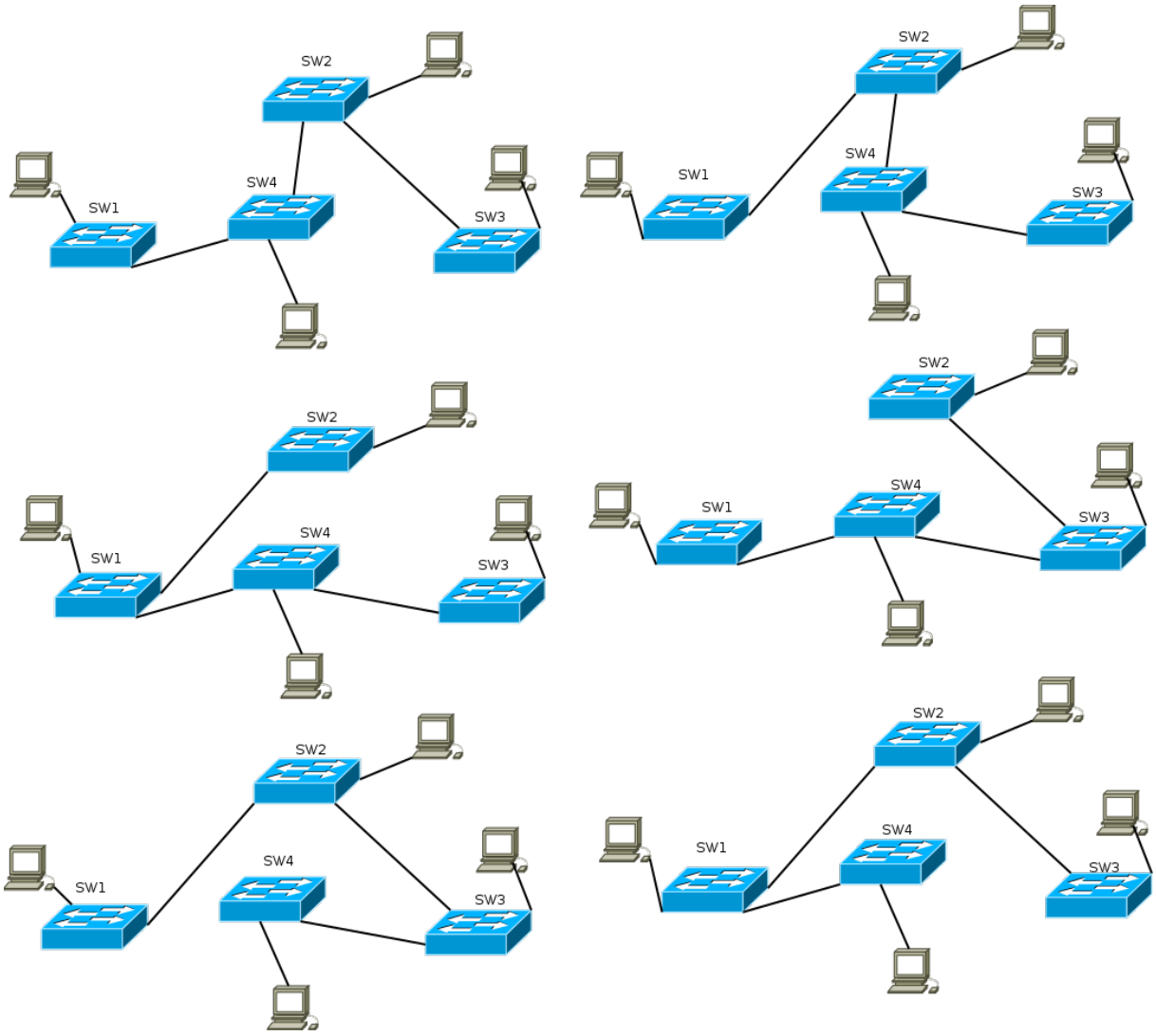

Figura 3.10: Tologias lógicas

s2 pela interface eth1 com o MAC verdadeiro igual a 36:ac:0e:d9:35:a6. Ao sair pela interface eth3 do switch, o MAC já foi alterado no cabeçalho do quadro para 12:34:56:01:02:01.

```
 root@openflowvm:~/of12softswitch# tcpdump -lne -i s2-eth1
tcpdump: WARNING: s2-eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.529516 36:ac:0e:d9:35:a6 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:26.540321 12:34:56:02:03:03 > 36:ac:0e:d9:35:a6, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s2-eth3
tcpdump: WARNING: s2-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.531510 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:27.535659 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s2-eth2
tcpdump: WARNING: s2-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s2-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.538015 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64
22:51:27.546486 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 2, length 64
```

No s2 a requisição do ping está saindo pela interface eth3 e a resposta do destino está voltando pela interface eth2.

Verifica-se que na interface eth3 está o host destino do ping analisado. A requisição chega no switch s3 pela interface eth1 e é encaminhada ao próximo pela interface eth3 ao host de IP 10.0.0.5.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth1
```

```
tcpdump: WARNING: s3-eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.533661 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:27.540968 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth3
tcpdump: WARNING: s3-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.535859 12:34:56:01:02:01 > 2a:e7:46:2a:0b:0a, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:26.535901 2a:e7:46:2a:0b:0a > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s3-eth2
tcpdump: WARNING: s3-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s3-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.536928 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64
22:51:27.544911 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 2, length 64
```

Conforme a captura acima, a reposta do ping está chegando pela interface eth3 do host com MAC address 2a:e7:46:2a:0b:0a e é encaminhada pela interface eth2 com MAC virtual 12:34:56:02:03:03 ao próximo switch.

Em s4 a requisição chega pela interface eth1 e é encaminhada ao s7 pela interface eth3. Já a resposta entra pela interface eth2 e é encaminhada também pela interface eth3.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s4-eth1
tcpdump: WARNING: s4-eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s4-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.531567 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
```

```
22:51:27.535684 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s4-eth3
tcpdump: WARNING: s4-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s4-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.532944 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:26.537535 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s4-eth2
tcpdump: WARNING: s4-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s4-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.536940 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64
22:51:27.544924 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 2, length 64
```

No switch s7 a requisição entra pela interface eth3 e é encaminhada pela eth2. A resposta também entra pela interface eth3 e é encaminhada pela interface eth1.

```
root@openflowvm:~/of12softswitch# tcpdump -lne -i s7-eth2
tcpdump: WARNING: s7-eth2: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s7-eth2, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.533647 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:27.540949 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s7-eth1
tcpdump: WARNING: s7-eth1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s7-eth1, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.538004 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
```

```
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64
22:51:27.546476 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 2, length 64

root@openflowvm:~/of12softswitch# tcpdump -lne -i s7-eth3
tcpdump: WARNING: s7-eth3: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on s7-eth3, link-type EN10MB (Ethernet), capture size 65535 bytes
22:51:26.532958 12:34:56:01:02:01 > 12:34:56:02:03:03, ethertype IPv4 (0x0800),
   length 98: 10.0.0.1 > 10.0.0.5: ICMP echo request, id 2304, seq 1, length 64
22:51:26.537546 12:34:56:02:03:03 > 12:34:56:01:02:01, ethertype IPv4 (0x0800),
   length 98: 10.0.0.5 > 10.0.0.1: ICMP echo reply, id 2304, seq 1, length 64
```

De acordo as capturas realizadas acima, pode-se perceber que o tráfego do host de IP 10.0.0.1 com sentido ao *host* de IP 10.0.0.5 percorreu um trajeto e o fluxo contrário passou por outro caminho.

O caminho traçado pela fluxo da requisição percorreu a topologia lógica descrita na figura 3.11.

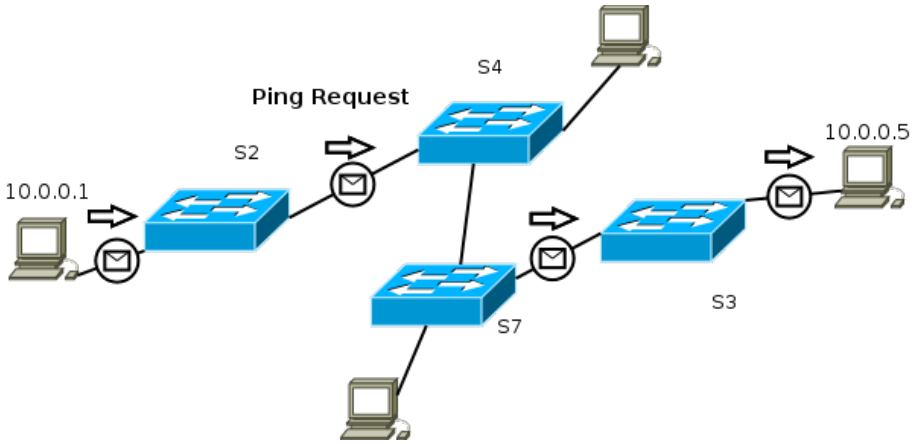

Figura 3.11: Caminho percorrido pelo *ping request*.

Já a topologia escolhida para a resposta do *ping* ficou conforme mostra a figura 3.12.

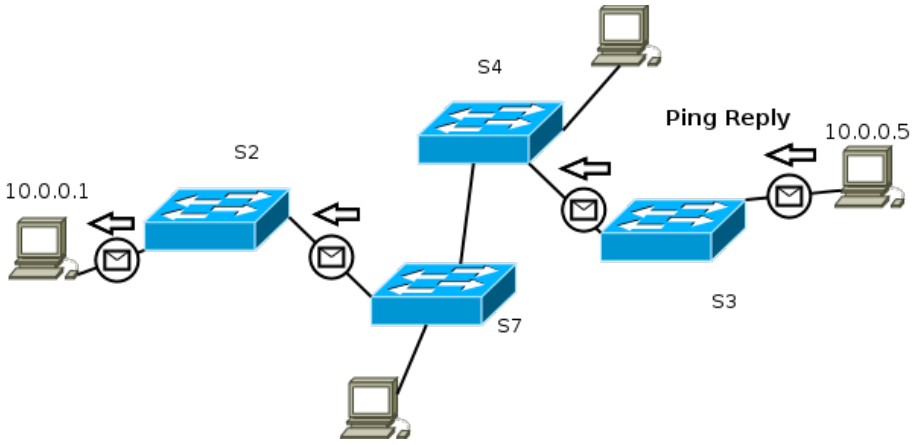

Figura 3.12: Caminho percorrido pelo ping reply.

4 Conclusão

Este trabalho foi um aprendizado de uma nova ferramente para estudos e desenvolvimento na área de redes. O maior esforço foi dedicado a entender o funcionamento do Openflow e do Ryu, além do bom tempo gasto paraa corrigir bugs do softswitch. Mas obviamente foi percebido a gama de possibilidades em utilizar SDN para definir o funcionamento da rede, sendo uma ótima ferramenta de estudo para novos métodos de comunicação entre equipamentos de rede.

O objetivo do trabalho foi concluído. O estudo de como criar uma rede definida por software foi realizado e com base nisso foi possível implementar uma rede local com switches realizando balanceamento de tráfego. A ideia inicial de realizar o balanceamento em uma rede local com caminhos fechados sem ter que utilizar protocolos Spanning-tree e aproveitando todos os enlaces disponíveis foi efetuado com sucesso. Portanto foi verificado a eficácia do controlador Ryu utilizando Openflow/SDN para realizar a proposta sugerida.

Obviamente, muitas melhorias podem ser feitas com base no trabalho que foi realizado. Fica agora algumas sugestões de futuros trabalhos para melhorar e deixar o funcionamento da rede o mais inteligente possível.

- Descoberta automática das topologias lógicas;
- Descoberta automática dos switches e da topologia física;
- Escolha das topologias em função da carga nos switches e por número de saltos;
- Escolha de novas topologias caso o enlace cair.

Referências Bibliográficas

802.1D, I. *IEEE Standard for Local and metropolitan area networks Media Access Control (MAC) Bridges*. [S.l.: s.n.], 2004.

DONAHUE, G. A. *Network Warrior*. [S.l.: s.n.], 2007.

FOROUZAN, B. A. *Data Communications and Networking. Fourth Edition*. [S.l.: s.n.], 2007.

FOUNDATION, O. N. *Software-Defined Networking: The New Norm for Networks.* [s.n.], 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

MCKEOWN N. ANDERSON, T. et al. *OpenFlow: Enabling Innovation in Campus Networks.* [s.n.], 2008. Disponível em: <http://www.openflow.org/documents/openflow-wp-latest.pdf>.

PERLMAN, R. *Interconnections: bridges, routers, switches, and internetworking protocols. 2ª Edition*. [S.l.: s.n.], 2000.

SPECIFICATION, O. S. *Openflow Switch Specification - version 1.1.0*. [s.n.], 2011. Disponível em: <http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf>.

WIKIPEDIA. *Spanning Tree Protocol*. [s.n.], 2014. Disponível em: <http://en.wikipedia.org/wiki/Spanning Tree Protocol>.