Universidade Federal do ABC
Graduação em Engenharia de Informação

Nathalia Levorato

REDES DEFINIDAS POR SOFTWARE:
Ambiente Multiusuário

Santo André – SP
2015

Nathalia Levorato

REDES DEFINIDAS POR SOFTWARE:
Ambiente Multiusuário

Dissertação apresentada ao Curso de Graduação Universidade Federal do ABC, como requisito parcial para obtenção do grau de Bacharelado em Engenharia de Informação.

Orientador: Prof. Dr. João Henrique Kleinschmidt

Santo André – SP
2015

Ficha Catalográfica

Levorato, Nathalia. REDES DEFINIDAS POR SOFTWARE: Ambiente Multiusuário. – Santo André, SP: UFABC, 2015. 33 p

Nathalia Levorato

REDES DEFINIDAS POR SOFTWARE:
Ambiente Multi-usuário

Essa dissertação foi julgada e aprovada para a obtenção do grau de Bacharelado em Engenharia de Informação no curso de Engenharia de Informação da Universidade Federal do ABC.

Santo André – SP, 21 de julho de 2015

_____
Prof. Dr. Ricardo Suyama
Coordenador do Curso

BANCA EXAMINADORA

_____
Prof. Dr. João H. Kleinschmidt
Orientador

_____
Prof. Dr. Amaury Kruel Budri
UFABC

_____
Prof. Dr. Nunzio Marco Torrisi
UFABC

A meus pais, Paulo e Izilda, e ao meu
Noivo, Alexandre, pelo apoio constante.

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Comparando a sociedade dos anos 90 com a dos anos 2000 nota-se uma grande mudança na comunicação entre homens, até os anos 90 predominava-se uma comunicação direta, direta no sentido que para se comunicar era necessário estar próximo do interlocutor, o uso de elementos intermediários como telefone, cartas, mensagens e outros, não predominava no dia-a-dia das pessoas. [1]

A partir dos anos 2000 com a Internet amplamente presente na vida das pessoas a comunicação indireta começou a surgir fortemente e com o passar do tempo foi se tornando cada vez mais frequente.

A princípio essa comunicação indireta foi acelerada com o uso de telefones, a redução dos custos de ligação e a evolução das redes telefonia fixa e móvel permitiram o aumento da comunicação entre pessoas fisicamente distintas de forma rápida.

Posteriormente, o uso de e-mails e comunicadores instantâneos (ICQ, MSN e outros), que possuíam como pré-requisito ter um computador com acesso à internet, passaram a fazer parte do cotidiano, fazendo com que assuntos que eram tratados pessoalmente ou através do telefone passassem a ser tratados de forma totalmente virtual.

Nos dias atuais, com a evolução da internet móvel, e-mails e comunicadores instantâneos passaram a ser usados em celulares e as pessoas passaram a se comunicar, em boa parte do tempo, através destas tecnologias, além de continuarem com o uso de computadores.

Não somente a comunicação entre pessoas passaram a fazer uso de tecnologia, mas relações comerciais, financeiras e diversos tipos de relação passaram a funcionar através da internet.

Para atender a toda essa demanda se faz necessário o avanço constante das redes de computadores, desde o aumento desta malha, priorização do tráfego que passa por ela, até uma reformulação da arquitetura atual que se apresenta pouco flexível.

Como alternativa a arquitetura atual tem se proposto redes programáveis ou redes definidas por software, um novo paradigma de redes que abordaremos neste trabalho.

1.1 Redes Definidas por Software

As redes definidas por software possuem como grande aliado o OpenFlow, iniciativa mais bem-sucedida na área, essa tecnologia atribui a inteligência dos switches a um servidor OpenFlow que define a forma de trabalho e consequentemente o protocolo de funcionamento do switch. Nas redes definidas por software o processo de encaminhamento continua sendo tarefa do hardware, porém o modo como ele deve ser encaminhado é definida por camadas superiores.

1.2 Motivação

Os grandes fabricantes de ativos de rede vêm falando de Redes definidas por software em ambientes de data center, elas aprimoram os benefícios da virtualização, aumentando a flexibilidade e a utilização de recursos e reduzindo custos e sobrecargas de infraestrutura.

O gerenciamento centralizado permite a automatização da configuração da estrutura, definição de políticas de acesso e configuração de forma rápida.

Essa necessidade de mudança de arquitetura de redes e vantagens que vem junto como despertou grande interesse e motivação para simulação de um ambiente que se aplique a data center multiusuário.

1.3 Objetivo

O objetivo deste trabalho é simular um ambiente aplicável a datacenter, onde usuários diferentes se conectam a um mesmo switch Openflow e se comunicam com redes externas através de roteamento estático.

A simulação será realizada através do Mininet, para construção da estrutura, utilizará um controlador POX, como controlador SDN, e visa garantir e testar a conectividade entre hosts de diferentes estruturas, analisando o tráfego gerado nela.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Internet

A Internet surgiu nos últimos anos da década de 1960, chamada de ARPANET, era uma rede de longa distância experimental que tinha como objetivo permitir aos fornecedores do governo compartilhar recursos computacionais. O uso colaborativo da rede vem desde essa época com o compartilhamento de arquivos e e-mail. [2]

O protocolo que padronizou a ARPANET foi a pilha TCP/IP (*Transmission Control Protocol/Internet Protocol*) criado no início da década de 1980, baseado nesta pilha de protocolos a ARPANET original tornou-se o backbone de redes locais e regionais chamada de Internet. Posteriormente esse backbone foi substituído pela rede NSFNET e, nos dias atuais, é composto de múltiplos backbones comerciais.

O Padrão TCP/IP é conhecido como Modelo de Referência TCP/IP composto de quatro camadas e foi adotado pela indústria de rede na fabricação de equipamentos. Já o Modelo de Referência OSI (*Open Systems Interconnection*), foi adotado pelas instituições acadêmicas, estabelecido em sete camadas, criado pela ISO na década de 1970 para eliminar a incompatibilidade entre fabricantes.

Dado que existe uma equivalência entre os modelos, será analisado o modelo OSI e posteriormente será estabelecida uma equivalência entre este e o modelo TCP/IP.

2.2 Modelo OSI

O modelo OSI, também chamado de Modelo de Referência ISO OSI possui sete camadas. As camadas foram criadas pela necessidade de um nível de abstração adicional, com uma função bem definida a partir de protocolos estabelecidos internacionalmente. Os limites entre elas são definidos a fim de minimizar o fluxo de informações pelas interfaces e a quantidade de forma que funções distintas não tenham que ficar na mesma camada e que a arquitetura não se torne complexa de controlar. [3]

Assim, as camadas serão apresentadas a partir da inferior até a superior, especificando a função de cada uma.

2.2.1 Camada Física

Nesta camada ocorre a transmissão de bits brutos através de um canal de comunicação, garantindo a recepção do bit correto. Nesta camada estão envolvidos os níveis de tensão e a duração de cada bit, a forma de transmissão, modo como a conexão será estabelecida e encerrada. Também se define os diferentes canais de comunicação, cabo coaxial, cabo UTP, fibra óptica e outros.

2.2.2 Camada de Enlace de Dados

Enquanto a camada física trata da transmissão de bits, nesta camada ocorre a transmissão de quadros, eliminando os erros não detectados na camada anterior. Os quadros devem ser transmitidos sequencialmente e em caso de serviço confiável o receptor enviará um quadro de confirmação ao receptor. Também é realizado o controle de fluxo para que um transmissor rápido envie dados excessivos a um receptor lento.

2.2.3 Camada de Rede

Nesta camada os dados apresentam-se na forma de pacotes a fim de determinar como ocorrerá o roteamento entre origem e destino. A determinação de rotas pode ocorrer de forma estática ou dinâmica.

O conjunto de redes sob uma mesma administração que segue uma política de roteamento é chamado de SA (Sistema Autônomo). [4]

Nesta camada operam os protocolos de roteamento internos e externo que permite a comunicação entre redes distintas. Os protocolos de roteamento interno, como RIP (*Routing Information Protocol*) e OSPF (*Open Shortest Path First*), são utilizados dentro de um SA a fim de simplificar a fronteira entre SAs e reutilização de códigos. Já o externo, BGP (*Border Gateway Protocol)*, realiza a comunicação entre redes pertencentes a diferentes SAs. [5]

Esses protocolos estão presentes em toda a Internet e permitem que ocorra a comunicação entre todos os equipamentos existentes na rede.

Questões como o controle de congestionamento na rede são tratadas na camada de rede (qualidade de serviço). Permite a interconexão de redes heterogêneas.

Além destas atividades e protocolos nesta camada opera o protocolo IP (*Internet Protocol*), ICMP (*Internet Control Message Protocol*), ARP (*Address Resolution Protocol*) e outros. Detalharemos o ARP um pouco mais, por se tratar de um protocolo importante, pois mapeia o endereço lógico (IP) com o endereço físico (MAC).

Todas as vezes que um host deseja conversar com outro é enviado um pacote ARP de consulta, que contém os endereços físico e lógico do remetente e o, lógico do destinatário. Esse pacote é enviado para todos os destinos da rede e somente o host com o endereço de destino envia a resposta ARP. Este processo pode ser verificado na Figura 1. [6]

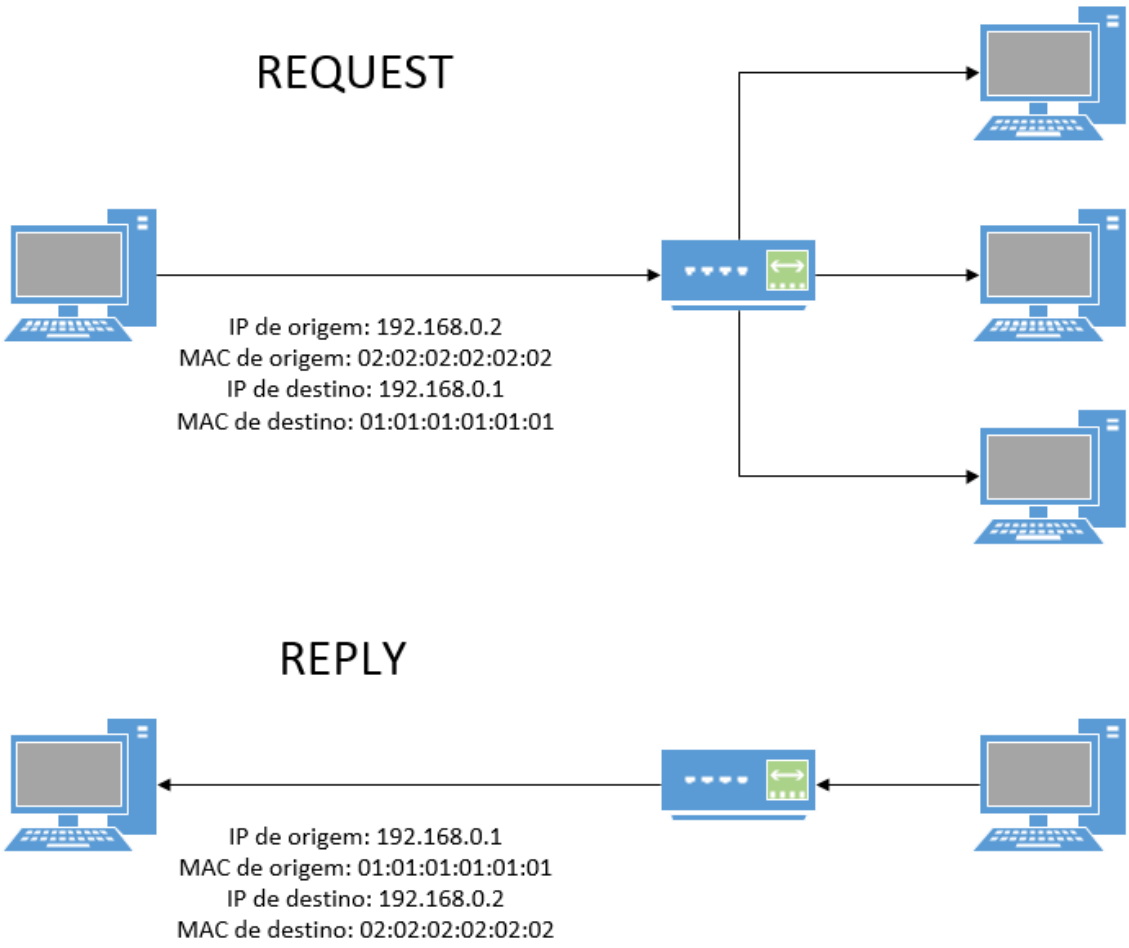


Figura 1 Processo de comunicação APR.

2.2.4 Camada de Transporte

Os dados nesta camada são chamados de segmentos, que são os dados da camada superior (sessão) divididos ou não em unidades menores repassados para camada de rede garantindo a entrega para a outra extremidade. Permite que a tecnologia das camadas inferiores a esta sejam transparentes para as superiores. Também o tipo de serviço que deve ser fornecido a camada de sessão é definido nesta camada ao estabelecer conexão. Estabelece a conexão fim a fim para troca de informações.

2.2.5 Camada de sessão

Permite que usuários de diferentes máquinas estabeleçam sessões entre eles, cada sessão oferece diversos serviços como controle de diálogo, gerenciamento de token e sincronização.

2.2.6 Camada de Apresentação

Relacionada à sintaxe e semântica das informações transmitidas. Ela gerencia estruturas de dados abstratas e permite a definição e o intercâmbio de estrutura de dados de nível mais alto.

2.2.7 Camada de Aplicação

Nesta camada encontra-se uma diversidade de protocolos necessários para o usuário executar tarefas como acesso à Web, transferência de arquivos, correio eletrônico e transmissão de notícias pela rede.

O modelo TCP/IP e o modelo OSI possuem algumas semelhanças, na Figura 2 verifica-se uma equivalência entre as camadas de cada modelo, onde as camadas 1 e 2 do OSI se resumem em uma no TCP/IP e as camadas 5 e 6 não estão presentes, demais camadas são equivalentes.

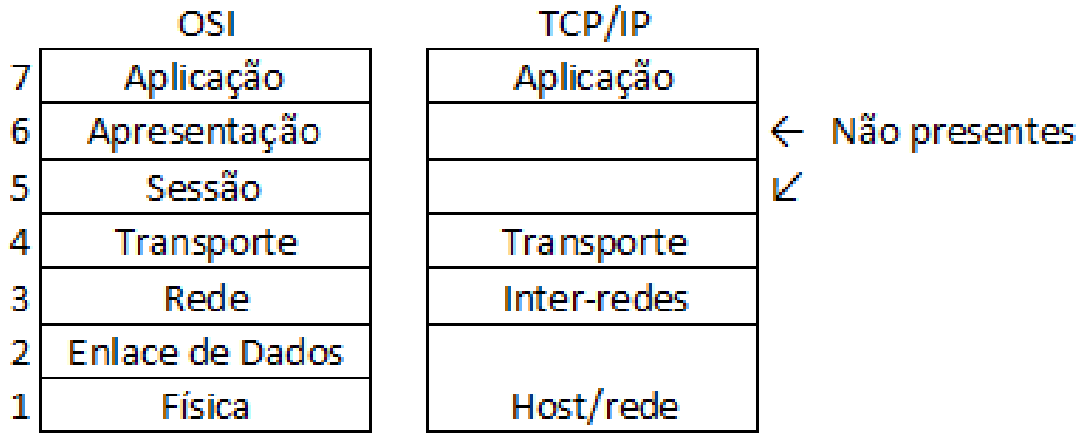


Figura 2 Modelo de referência OSI x TCP/IP

Baseado nesses modelos as redes e a Internet foram se desenvolvendo e surgindo a necessidade de ambientes de alta disponibilidade, acessíveis de qualquer lugar, garantindo a continuidade do negócio de diversas organizações, dessa forma será discutido na seção 2.3 sobre o papel dos data centers.

2.3 Data Center

"O Data Center (Centro de Dados) é um ambiente projetado para abrigar servidores e outros componentes como sistemas de armazenamento de dados (*storages*) e ativos de rede (switches, roteadores). O objetivo principal de um Data Center é garantir a disponibilidade de equipamentos que rodam sistemas cruciais para o negócio de uma organização, tal como o ERP (*Enterprise Resource Planning*) ou CRM (*Customer Relationship Management*), garantindo assim a continuidade do negócio". [7]

Os principais componentes do datacenter são: infraestrutura de rede, segurança física, combate e prevenção contra incêndios, refrigeração e energia. [7]

A infraestrutura de rede que garante o acesso aos sistemas hospedados no Data Center, contando com redundância de equipamentos e links afim de evitar a ocorrência de falhas na estrutura.

O projeto de um Data Center é, geralmente, baseado em uma abordagem em camadas para melhorar a escalabilidade, performance, flexibilidade, resiliência e manutenção da rede. Existem três camadas para o projeto:

- Core: realiza comutação de pacotes de alta velocidade para todos os fluxos que entram e saem do datacenter. Na camada core operam protocolos de roteamento interno e balanceamento de tráfego entre Campus Core e Camada de agregação.
- Agregação: Define os domínios de colisão, processamento do STP (*Spanning tree Protocol* – Protocolo gerenciador de uplinks que provê caminhos redundantes e, ao mesmo tempo, previne loops indesejáveis na rede [8]) e redundância do gateway padrão. Pode ou não operar protocolos de roteamento e/ou *spanning tree*.
- Acesso: Fornece conectividade física dos servidores a rede através de switches. Opera *spanning tree*.

Quando se fala em protocolo de roteamento refere-se a um protocolo de camada 3 do modelo OSI, geralmente utiliza-se o protocolo OSPF (*Open Shortest Path First*). Já o spanning tree, refere-se a camada 2.

Os equipamentos de camada 3, devem possuir alta capacidade para manipulação da tabela de roteamento que dependendo do tamanho da rede consome bastante capacidade de processamento.

Os maiores problemas relacionados a camada 2 refere-se a loops gerados na estrutura, dada a necessidade de uplinks redundantes, que se mal definidos ocasionam este tipo de problema.

A utilização de VLAN (*Virtual Local Area Network*) é muito comum para a segmentação de ambientes, dividindo a rede em diferentes domínios de broadcast, evitando excesso de tráfego devido a comunicação ocorrida em camada 2. [9]

Assim, as redes baseadas em software, um novo conceito de redes, podem contribuir com ambientes de data center descentralizando as atividades realizadas na camada CORE e realizando uma distribuição interna. Os equipamentos de borda que necessitavam ser de grande porte requeriam um alto investimento podem ser substituídos por switches de porte médio com interfaces OpenFlow, que teriam as informações de roteamento conforme demanda detectada pelo controlador, minimizando os custos.

Já em relação aos problemas de loop na camada 2 ocasionado por VLAN, a substituição de VLAN por switches virtuais permite a interligação dos equipamentos

do cliente, sendo o controlador responsável pela definição das portas dos diversos comutadores que compõe o comutador virtual. Mesmo que hajam equipamentos do cliente conectados a diferentes comutadores, será possível liga-los em um único comutador virtual.

Assim, pode-se verificar na seção 4 um pouco mais sobre as redes definidas por software.

2.4 Redes Definidas por Software

Redes definidas por software (Do inglês *Software Defined Networks* - SDN) representam um novo paradigma para redes de computadores que vem superar os limites das tecnologias de redes de computadores desenvolvidas tradicionalmente, e permitir que as redes sejam programáveis. [3 10]

Uma das propostas para redes programáveis veio com as redes ativas, que necessita de substituição dos elementos de rede, representando um alto investimento. Como as redes de computadores tradicionais estão engessadas, pesquisas estão sendo desenvolvidas para que as redes programáveis sejam implementadas gradativamente sobre estruturas de redes tradicionais, viabilizando a implementação de redes programáveis em etapas, distribuindo também o investimento necessário para implementação.

A finalidade de uma SDN é desvincular a comutação de pacotes através do hardware e seus protocolos de roteamento e vinculá-la a aplicações, além da possibilidade de desenvolvimento de aplicações que controlem a comutação da rede física.

A rede definida é composta basicamente por comutador, divisores de recursos/visões, controlador e aplicações de rede, estes quatro elementos serão descritos abaixo:

2.4.1 Comutador - Elemento de comutação programável

Realiza o encaminhamento de pacotes baseado no conceito de fluxos, cada pacote recebido no comutador é analisado conforme interface de entrada e encaminhado para a interface de destino conforme tabela de encaminhamento

(definida através dos recursos oferecidos pela interface de programação, como o OpenFlow), atravessando as interconexões internas do comutador.

2.4.2 Divisores de Recursos/Visões

Permite o uso de diferentes controladores para atender os fluxos existentes na rede. Cada fluxo responderá a um padrão de comportamento existente na rede. Por exemplo, o tráfego real de uma rede poderá ter seu comportamento conforme as regras dos fabricantes de hardware ou roteadores legados, sendo considerado o fluxo padrão. Já os diferentes fluxos de pesquisa e testes tratarão ambientes de simulação sobre a rede existente. Um fluxo não impactará nos demais.

O FlowVisor faz esta função realizando a divisão direta dos recursos OpenFlow (interface de programação que controla comutadores), dividindo o espaço de endereçamento disponíveis. A definição do fluxo é realizada pelo rótulo dos pacotes que através dos diferentes campos determinam a política de processamento.

2.4.3 Controlador

Opera como um sistema operacional da rede, contendo as tarefas de manipulação direta dos dispositivos da rede (visão unificada da rede) e trazendo um nível de abstração de mais alto nível para o desenvolvedor.

A visão unificada da rede é uma visão lógica que permite uma análise detalhada que facilita as decisões operacionais sobre a operação do sistema.

Existem diversos tipos de controladores disponíveis que podem ser classificados quanto ao tipo de interface de programação que disponibilizam. Os tipos de interface podem ser imperativa (Ambientes de tempo de execução), funcional ou declarativa (Detecta conflitos ou realiza depuração automatizada da rede).

A forma de distribuição mais simples é a centralizada, porém acarreta problemas de escalabilidade, mas existem outras formas de distribuição para ambientes de grande porte, como datacenters, garantindo a escalabilidade e disponibilidade do sistema.

2.4.4 Aplicações de rede

Composta pelo software desenvolvido para ser executado sobre a rede física que implementa novas funcionalidades. O desenvolvimento é realizado tendo como base a visão lógica desenvolvida da rede. Exemplos de aplicações podem ser encontrados no controle de acesso (uma política de firewall que determinará o fluxo), gerenciamento de redes (configuração simplificada), comutador virtual distribuído (facilita a configuração e monitoramento de máquinas virtuais), roteador expansível de alta capacidade (substituição de switches de grande porte por switches de médio com interface openflow) e redes de grande porte (virtualização de switches para serviços diferenciados).

2.4.5 Relação entre os elementos

Na Figura 3 observa-se as camadas encontradas em uma estrutura OpenFlow onde cada camada é composta pelos elementos descritos anteriormente.

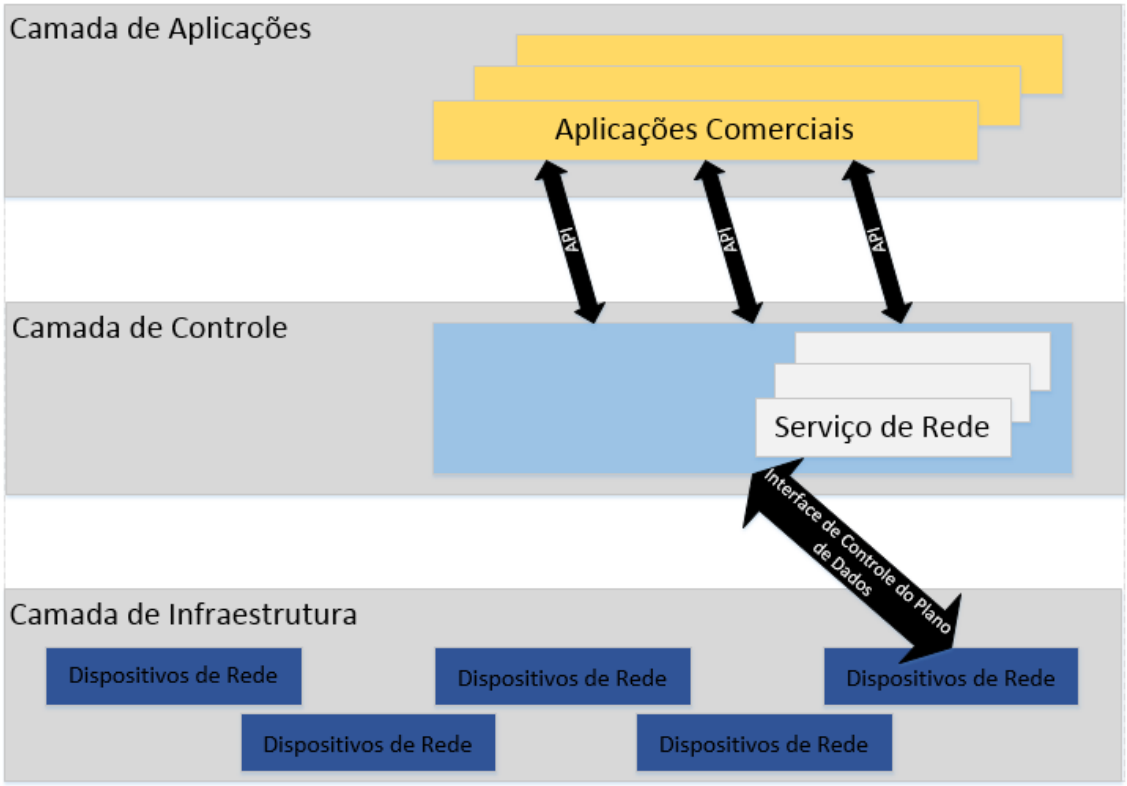


Figura 3 Camadas da estrutura OpenFlow.

A camada de aplicação é composta pelas aplicações de redes que se comunica com o controlador e divisores de recursos/visões (Camada de controle - intermediária) através de API's, a camada de aplicação permite que elementos como firewalls transmitam informações para que a camada de controle estabeleça fluxos baseados nas regras definidas por este.

Já a comunicação entre camada de controle e infraestrutura, sendo a segunda composta pelos comutadores, se comunicam através de uma interface de controle do plano de dados. Os comutadores estabelecem os fluxos conforme tabela definida pelos controladores.

Na Figura 4 observa-se uma visão global de uma estrutura de rede usando SDN, onde um servidor armazena as informações da rede e se comunica com o comutador para que este saiba qual o fluxo de encaminhamento dos pacotes.

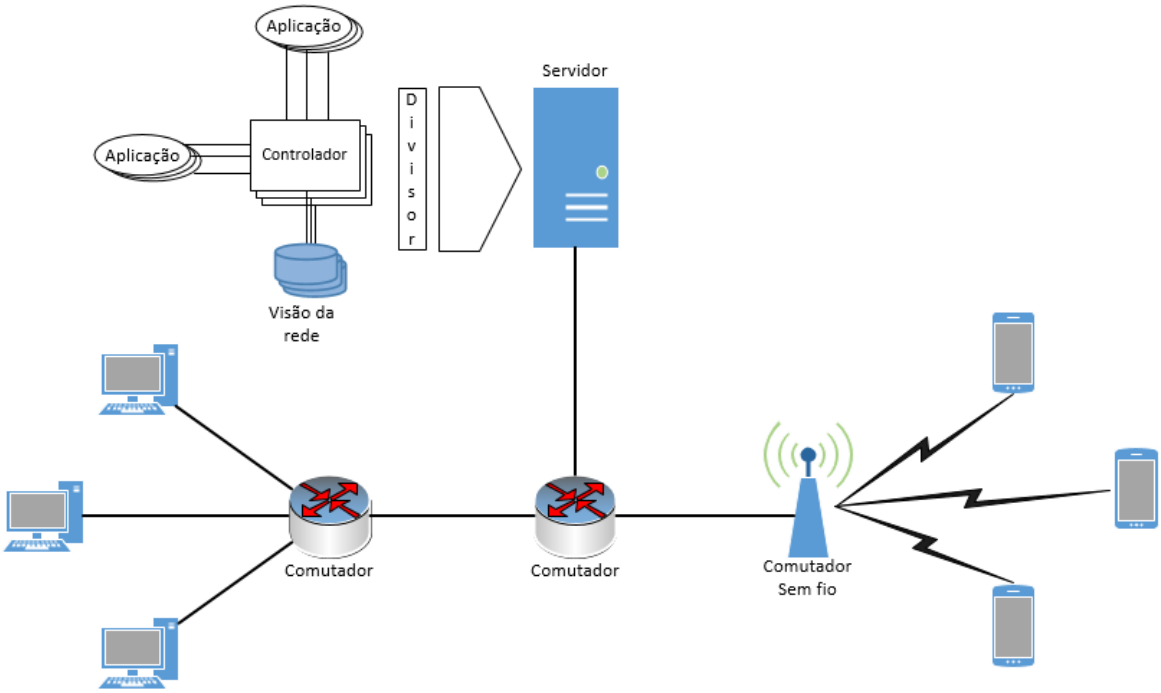


Figura 4 Estrutura geral de uma SDN.

2.5 OpenFlow

O OpenFlow foi proposto pela Universidade de Stanford com o objetivo de testar as novas estruturas de rede lógica em desenvolvimento, permitindo sua operação sobre equipamentos comerciais, atualmente continua a ser desenvolvido pela Open Network Foundation (ONF). É um padrão aberto que provê uma interface

de programação para controle dos elementos de encaminhamento de pacotes dos dispositivos. [11]

A partir da estrutura operando com o OpenFlow torna-se possível a operação de redes de produção e teste em uma mesma estrutura, tornando-se atrativo para os pesquisadores que tem a possibilidade de realizar testes das pesquisas em uma estrutura de rede real com o tráfego isolado do tráfego de produção.

O OpenFlow possui uma característica de separação entre plano de dados e de controle. No plano de dados ocorre o encaminhamento de pacotes conforme entradas da tabela de encaminhamento do comutador, já no de controle, permite programar a entrada da tabela pelo controlador SDN que detectam os fluxos de interesse e as regras deste.

A rede OpenFlow possui quatro componentes: tabela de fluxos, protocolo OpenFlow, Controlador e Canal seguro que serão descritos abaixo.

2.5.1 Tabela de Fluxos

A tabela de fluxos define os fluxos conforme regras que definem as ações a serem tomadas. A tabela de fluxo é composta pelos campos abaixo que podem ser visualizados na figura 5:

- Regras: define como os campos do cabeçalho determinam o fluxo
- Ações: indica o processamento dos pacotes
- Controle de estatística: mantém a estatística de utilização e remove fluxos inativos

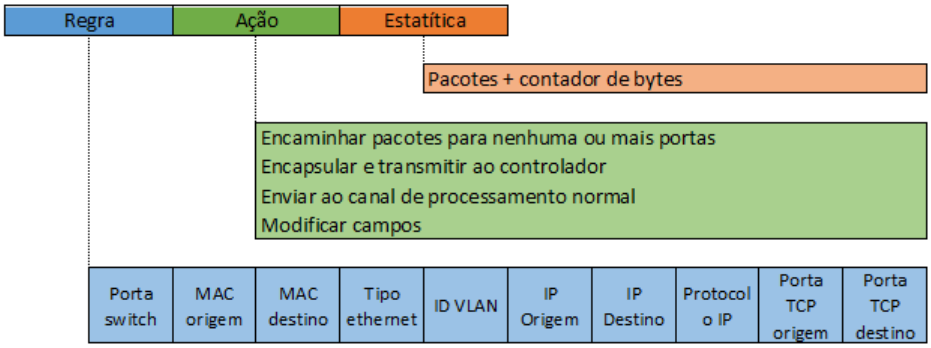


Figura 5 Definição do fluxo OpenFlow.

2.5.2 Protocolo OpenFlow

Realiza a comunicação entre os controladores de rede e OpenFlow.

2.5.3 Controlador

Decide qual a política de encaminhamento a ser implementada ou alterada na tabela de fluxos.

2.5.4 Canal Seguro

O canal seguro é o canal de comunicação entre comutadores e controladores confiável e com baixa taxa de erros, impedindo ataques de elementos mal-intencionados.

2.5.5 Mensagens OpenFlow

A comunicação entre o controlador OpenFlow e os elementos encaminhadores ocorrem através de mensagens de diferentes tipos. Abaixo segue as mensagens trocadas pelo controlador e suas características:

- Característica: após estabelecimento do canal seguro, solicita ao comutador a conexão e encaminha as características específicas e capacidades.
- Configurações: originada pelo controlador, solicita/adiciona parâmetros de configuração ao comutador, este responde a solicitação.
- Modificações de estado: controlador envia regras para tabela de fluxos dos comutadores.
- Leitura de estado: enviada pelo controlador para obter status e estatísticas do controlador.
- Packet-out: envio de pacotes completos para o comutador.
- Barreira: garante cumprimento da dependência de mensagens e recebe notificação de operações concluídas.

2.5.6 Considerações Finais sobre o OpenFlow

Conforme visto o OpenFlow permite o avanço tecnológico das redes, propiciando um ambiente de testes sobre uma infraestrutura real. A cada nova versão apresenta maior flexibilidade, com menos restrições, melhor especificada, suporte a fibra, outros.

Para incentivar o desenvolvimento/adoção de normas para redes SDN+OpenFlow a Indiana University junto com a ONF lançou o SDN InteroperabilityLab.

2.6 Controladores

Nesta etapa serão abordadas as principais características de diversos controladores, buscando compreender a finalidade deles. Com enfoque nos mais populares e os que possuem maior relação com ambientes de datacenter, outros serão apenas citados.

2.6.1 NOX

O NOX tem como função principal o desenvolvimento de controladores em C++, sua operação baseia-se no conceito de fluxo de dados.

Um sistema operacional simples que provê algumas primitivas para gerenciamento de eventos como: packet_in, ocorre quando um pacote recebido de uma porta física e é encaminhado para o controlador; stats_in, resposta ao pedido das estatísticas associadas às regras contidas no padrão pattern, retornando os contadores de pacotes e bytes; flow_removed, remove regra pattern da tabela de fluxos por exceder o tempo limite; switch_join, entrada de um switch na rede; switch_exit, Saída de um switch na rede; port_changes, alteração de status do enlace de uma determinada porta física.

Além dos eventos, ele possui funções para envio de mensagens ao switch: install, insere uma regra com os parâmetros dado padrão, prioridade, tempo limite e ações na tabela de fluxos; unistall, remove a regra; send, envia um pacote que executará alguma ação no switch; e, query_stats, solicita uma regra padrão no switch, retornando um identificador de requisição xid (mapeia resposta assíncrona do switch).

Por possuir duas interfaces, C++ e Phyton, permitindo atingir ambientes de alto desempenho com o C++ e agilidade/simplificação no desenvolvimento com o Phyton.

2.6.2 POX

O POX vem substituir o NOX em casos que o desempenho não é um item crítico. É uma plataforma de desenvolvimento e prototipagem rápida de aplicações para SDN.

Mais estável e com uma interface mais elegante que seu antecessor, tornando-se mais moderno e simples. [12]

O POX é uma versão do NOX em Python, mantendo o NOX em C++ e desenvolvendo uma plataforma separada para Python. [13]

Algumas vantagens do NOX: [14]

- Interface OpenFlow em Python.
- Possui componentes de amostra reutilizáveis para seleção de caminho, descoberta de topologia e outros.
- Pode ser implementado utilizando o interpretador Pypy que utiliza JIT (*Just in time compilation*) pois converte o código diretamente em código de máquina.
- Funciona em múltiplas plataformas, Linux / MAC OS / Windows.
- Suporta a mesma GUI (*Guide user inferface*) e ferramentas de virtualização que o NOX.
- Melhor performance que aplicações NOX escritas em Python.
- Permite a criação de componentes próprios e possui alguns nativos.
- Popular para o ensino e pesquisa sobre SDN e aplicações de redes programáveis.

Alguns dos componentes nativos disponíveis são:

- Py: Inicia um interpretador Python interativo para realização de debug.
- Forwarding.hub: Instala apenas regras de inundação.

- Forwarding.l2_learning: Faz com que um switch OpenFlow atue com switch L2 (*Layer 2*).

Para o desenvolvimento de componentes estão disponíveis diversas API, dentre elas:

- Pox core: Provê interoperabilidade entre componentes.
- Pox lib packet: Biblioteca de análise e criação de pacotes.
- Pox lib revent: Sistema de eventos.
- Pox openflow libopenflow_01: Aplica ações a pacotes conforme regras e cria caminhos.

2.6.3 Maestro

O Maestro é uma plataforma de controlador escalável para os switches OpenFlow, orquestrando aplicações de controle no modelo SDN.

Suas interfaces permitem a introdução de novas funções através da adição componentes de controle modular, manutenção do estado da rede em nome dos componentes de controle e composição dos componentes de controle (sequência de execução e estado dos componentes).

Para melhorar o desempenho do sistema utiliza-se do paralelismo na máquina, buscando facilitar a programação para a melhor administração da paralelização. [15]

2.6.4 Onix

O Onix tem por objetivo controlar de forma confiável redes de grande porte, desenvolvido pela Nicira, Gloogle e NEC, tratando-se de um produto proprietário e fechado. A confiabilidade e garantia de escalabilidade se dá na abstração para distribuição do estado da rede entre diversos controladores.

O Onix mantém uma visão geral da rede chamada Network Information Base – NIB, todo o controle da rede é realizado pela consulta a essa base. A NIB é particionada e replicada e diversos servidores a fim de garantir a escalabilidade

através de sua estrutura hierárquica, com concentradores e diferentes níveis de detalhes para os recursos do sistema.

2.6.5 NEC ProgrammableFlow

O NEC ProgrammableFlow é uma família que disponibiliza componentes de software e hardware. Através do software ProgrammableFlow Management Console é possível o monitoramento da rede de forma centralizada. Já o ProgrammableFlowController possibilita a virtualização em nível de rede, possibilitando o monitoramento e controle de redes com níveis de gerencia variados. Indicado para uso em data centers.

2.6.6 Outros

Abaixo, pode-se verificar outros controladores não citados anteriormente, focados em outras necessidades ou características:

- **Trema**: Possui como escopo facilitar o desenvolvimento na criação de controladores customizados.
- **Beacon**: Suporta operações baseada em eventos e threads, possui uma estrutura modular a atualização em tempo real do controlador.
- **FML**: Permite especificar políticas de gerência e segurança em redes OpenFlow.
- **Frenetic**: Permite a programação da rede como um todo, sem necessidade de configurar cada equipamento.
- **Floodlight**: Para redes corporativas e baseado na linguagem Java. [16 17]

3 SOFTWARE DE SIMULAÇÃO

3.1 Mininet

O Mininet é um ambiente para simulação de redes que roda em sistemas Linux. Ele oferece dispositivos de rede como hosts finais (se comportam como hosts reais), switches, roteadores e links. [18]

Com o Mininet é possível criar redes que se assemelham a uma rede real.

Algumas vantagens do Mininet são:

- Agilidade: a inicialização de redes leva segundos, agilizando o processo de execução, edição e debug.
- Ambientes customizados: através do Mininet é possível criar as mais diversas topologias, variando de estruturas simples com um único switch até estruturas mais complexas, por exemplo um datacenter.
- Execução de programas reais: todos os sistemas que rodam em Linux estão disponíveis para execução.
- Customização do encaminhamento de pacotes: Redes SDN podem ser simuladas facilmente, dado que os switches são programáveis e utilizam protocolo OpenFlow.
- Facilidade de execução: O Mininet é compatível com qualquer sistema pois pode ser rodado em uma VM ou cloud.
- Projeto open source: seu código fonte pode ser analisado, modificado, corrigido bugs, etc.
- Desenvolvimento constante.

Limitações:

- Ao rodar em um único sistema os recursos são balanceados e compartilhados entre os hosts virtuais e switches.
- Por usar um sistema com kernel Linux não pode executar softwares para BSD, Windows e outros que sejam incompatíveis.
- O Mininet não escreve controladores OpenFlow.

- Por padrão, o Mininet é isolado da LAN e internet.
- Por padrão todos os hosts Mininet compartilham o sistema de arquivo host e espaço PID.
- O tempo das medições são baseadas em tempo real, dificultando a simulação de tempos menores que o tempo real.

Características:

- Criação de topologia parametrizada realizada através de códigos em Python e fazendo o uso de classes, métodos, funções e variáveis.
- Além de redes de comportamento básico, Mininet fornece desempenho e isolamento limitando recursos, através das classes CPULimitedHost e tclink.
- Execução de programas no host final, simples comandos como pingAll e iperf são fornecidos por padrão pelo Mininet.
- O esquema de atribuição de nomes segue o padrão: para hosts são h1 ... hN; switches, s1 ... sN e; interfaces, são nomeadas começando com o nome do nó seguido da interface, por exemplo h1-eth0.
- Uma das características mais poderosas e úteis do Mininet é que ele usa Software Defined Networking. Usando o protocolo OpenFlow e ferramentas relacionadas, você pode programar chaves para fazer quase qualquer coisa que você quiser com os pacotes que eles entram.

O Mininet é composto por um conjunto de funcionalidades integradas no Linux que permitem que um único sistema seja dividido em um monte de "containers" menores, cada um com uma quota fixa do poder de processamento, combinado com o código de link virtual que permite links com atrasos e velocidades precisos.

Internamente, ele emprega recursos de virtualização leves no kernel do Linux, incluindo os grupos de processo, isolamento largura de banda de CPU e namespaces de rede, e os combinam com link de programadores e links Ethernet virtuais. Esses recursos produzem um sistema que inicia mais rápido e escalável para mais hosts de emuladores que usam máquinas virtuais completas.

4 METODOLOGIA

Em um computador com processador de 5 núcleos, 4GB de RAM, 10GB de espaço livre no disco rígido, com sistema operacional Windows 7 Home Premium foi executada a VM Mininet através do VirtualBox e utilizado o controlador POX na estrutura. O processo de instalação pode ser visto no Anexo B.

4.1 Topologia

Redes locais distintas necessitam, tradicionalmente, de um roteador para realizar a comunicação entre elas, permitindo a comunicação entre hosts pertencentes a cada uma dessas redes.

O lado A da Figura 6 representa duas redes locais onde todo o tráfego gerado entre elas passará pelo switch-roteador-switch para atingir o host desejado. O lado B da figura representa o roteador de borda, que é o gateway de saída de toda a estrutura abaixo dele. O gateway da estrutura pode ser um roteador de protocolo de roteamento exterior que recebe um ou mais links de dados e se comunica através de roteamento externo com outros SA (Sistema autônomo) utilizando protocolo BGP. O uso do protocolo BGP trata-se de um exemplo, outros protocolos de roteamento poderiam ser utilizados, como roteamento estático, RIP e OSPF, que são protocolos de roteamento internos.

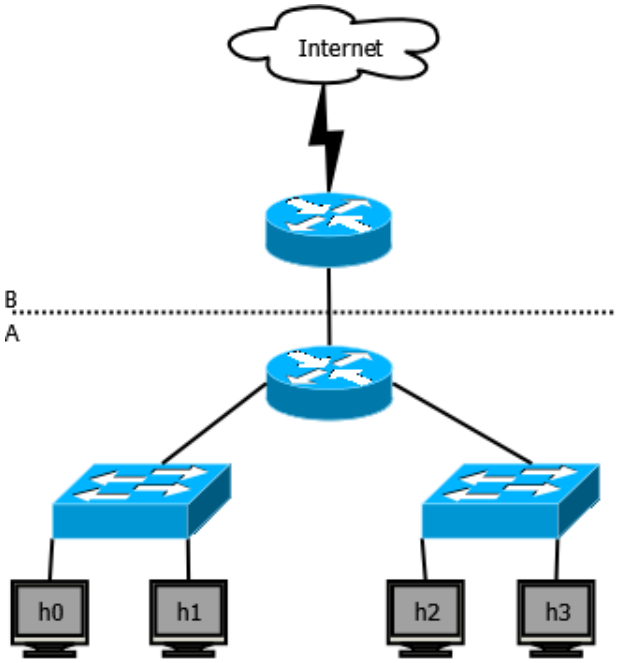


Figura 6 Exemplo de aplicação

A implementação visa substituir a parte A da Figura 6 por uma rede SDN, onde todos os equipamentos se conectarão em um switch OpenFlow, substituindo o roteador e os dois switches. Na topologia com SDN o host h0 exerce função híbrida, operando como roteador de borda e como o próprio host h0, membro de uma das redes. O host h4 representa a Internet com diversas redes configuradas nele. (Ver Figura 7)

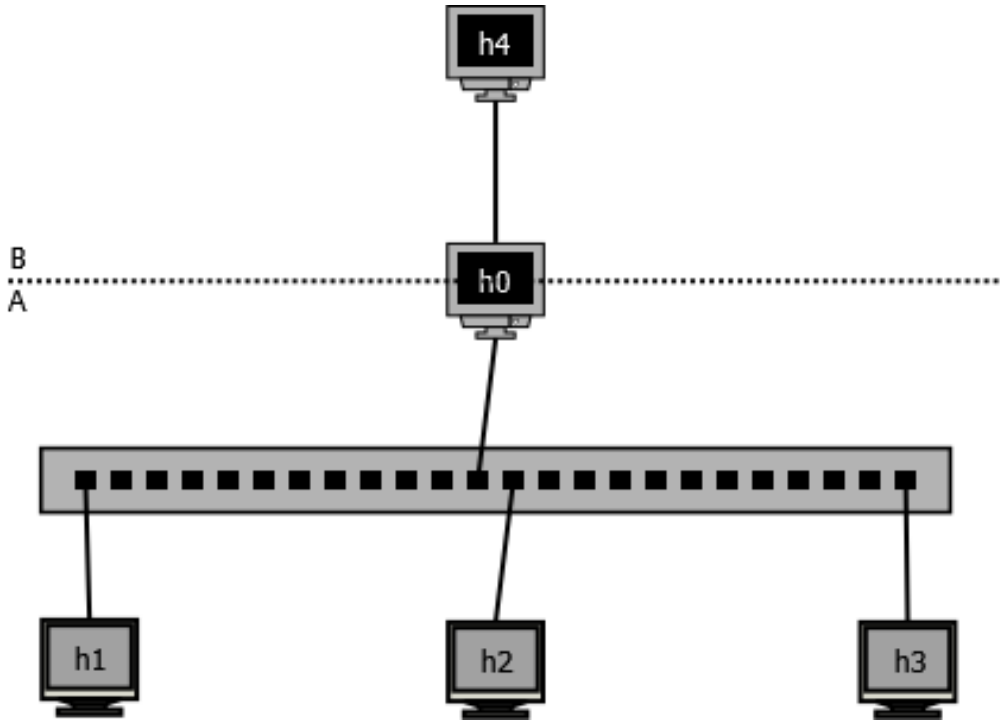


Figura 7 Topologia com switch OpenFlow

4.1.1 Implementação

A simulação implementa uma topologia com duas redes, A e B, compostas de dois hosts cada uma, onde todos hosts destas se conectam a um mesmo switch OpenFlow. As redes utilizadas são 10.0.1.0/24 e 10.0.2.0/24, sendo, respectivamente, rede A e B.

O primeiro host da rede A, h0, é o único equipamento que possui conexão externa, ou seja, fora do switch OpenFlow, com o equipamento h4 da rede C (172.16.0.0/32), que possui conexão com diversas redes (172.16.1.0/24, 172.16.2.0/24 e 172.16.3.0/24).

Em resumo, a topologia é composta de cinco hosts, três redes e um switch OpenFlow distribuídos conforme Figura 8.

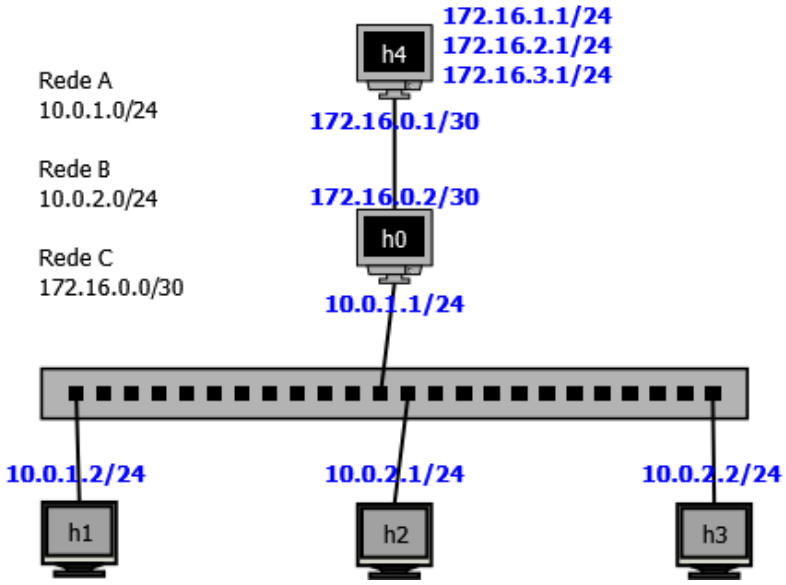


Figura 8 Topologia Simulada

Na plataforma de simulação (MiniNet) a topologia é criada, em grande parte, através de uma API em Python, que pode ser consultada no Anexo A na seção A.1. As configurações complementares são realizadas através da própria CLI do MiniNet.

A API define todos os hosts da estrutura com endereço de MAC e IP das interfaces principais e default gateway dos hosts h1, h2 e h3, o switch s0 e seu MAC, além de todas as conexões.

Na CLI do MiniNet é configurado o endereço IP da segunda interface do host h0, a rota estática para as redes 10.0.0.0/8 nos hosts h0 e h4, a rota default e o encaminhamento de pacotes IPv4 no host h0,

Todos os comandos podem ser visualizados no Anexo A na seção A.2.

4.2 Controlador

O controlador utilizado no projeto foi um controlador POX desenvolvido, não foi utilizado nenhum controlador nativo.

O controlador implementou um roteador e seu código foi baseado no exemplo fornecido na referência 5, seu código completo pode ser visualizado no Anexo A3 e abaixo encontra-se o fluxograma deste.

Na Figura 9 verifica-se o fluxograma da chegada de um pacote dentro do switch OpenFlow, inicialmente é verificada a integridade do pacote, descartando o pacote

caso não haja cabeçalho. Posteriormente, o tipo de pacote é verificado e para cada tipo é realizada uma ação:

- LLDP: pacote descartado.
- ARP: Insere a origem na tabela IP e analisa o destino, atualizando seu estado caso necessário e executa o fluxo de resposta ARP (Figura 10).
- IP: Executa o fluxo de roteamento (Figura 11).

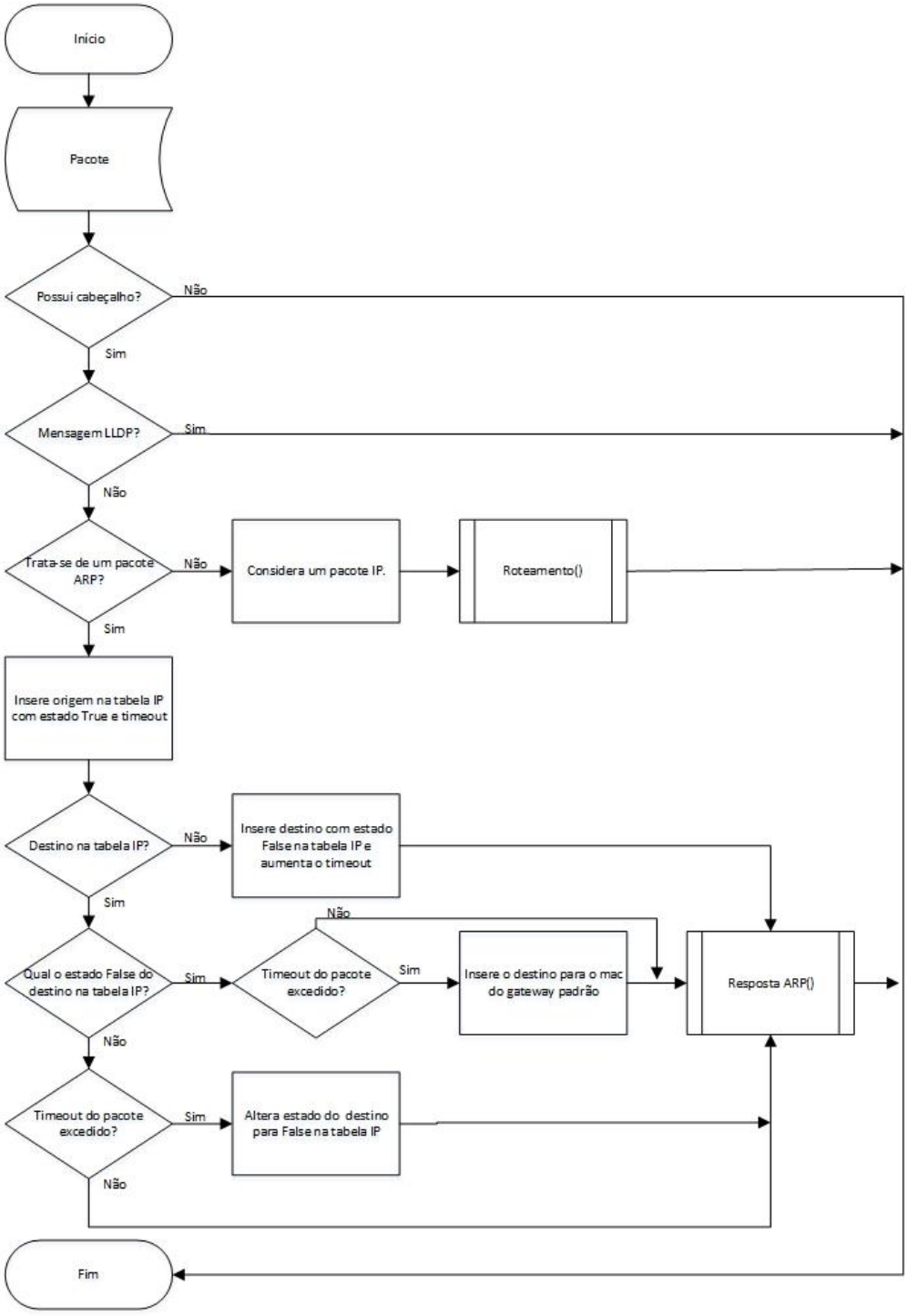


Figura 9 Entrada de pacote no switch OpenFlow

Na resposta ARP é verificada a integridade do datagrama, encerrando o fluxo caso não esteja completo. Posteriormente a resposta ARP é gerada somente para ARP request cujo destino encontra-se na tabela ARP sem timeout expirado, demais casos realizam a verificação para inundação de pacotes que só não é realizada quando timeout do destino na tabela ARP está excedido. O fluxo pode ser observado na Figura 10.

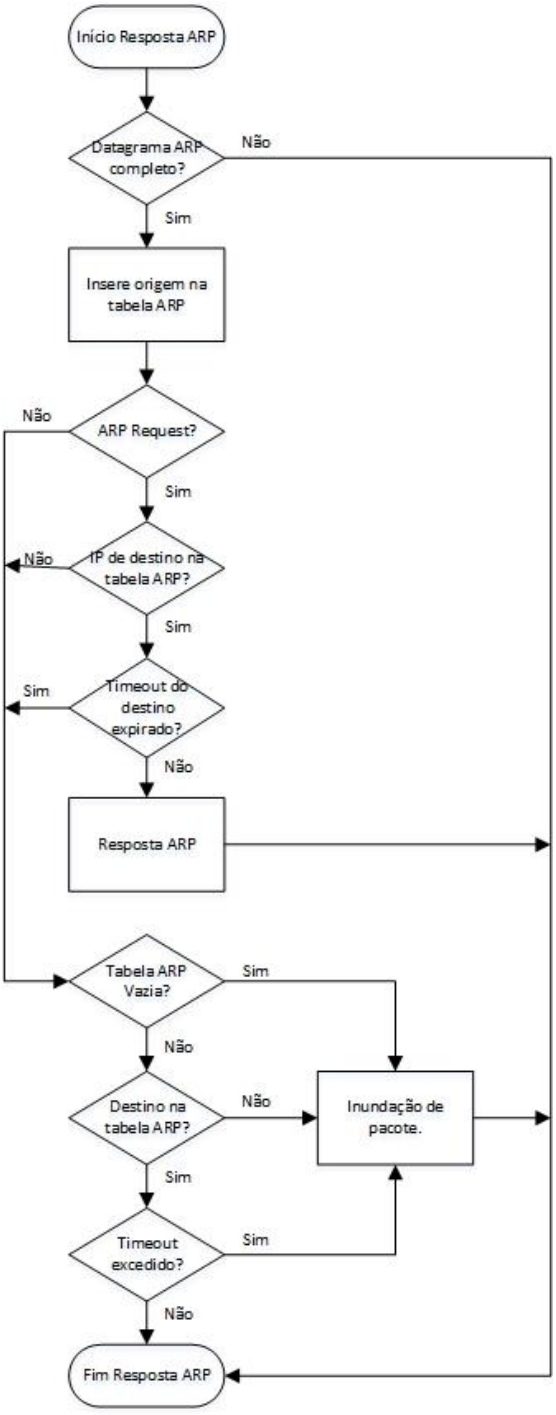


Figura 10 Fluxo Resposta ARP

No roteamento (Figura 11) verifica-se se a tabela ARP está vazia, encerrando caso esteja, caso contrário é atualizado o timeout da origem na tabela. Três cenários podem ocorre:

1. Destino não está na tabela: encerra.
2. Timeout excedido: remove destino na tabela e encerra.
3. Timeout não excedido: cria fluxo entre origem e destino e encerra.

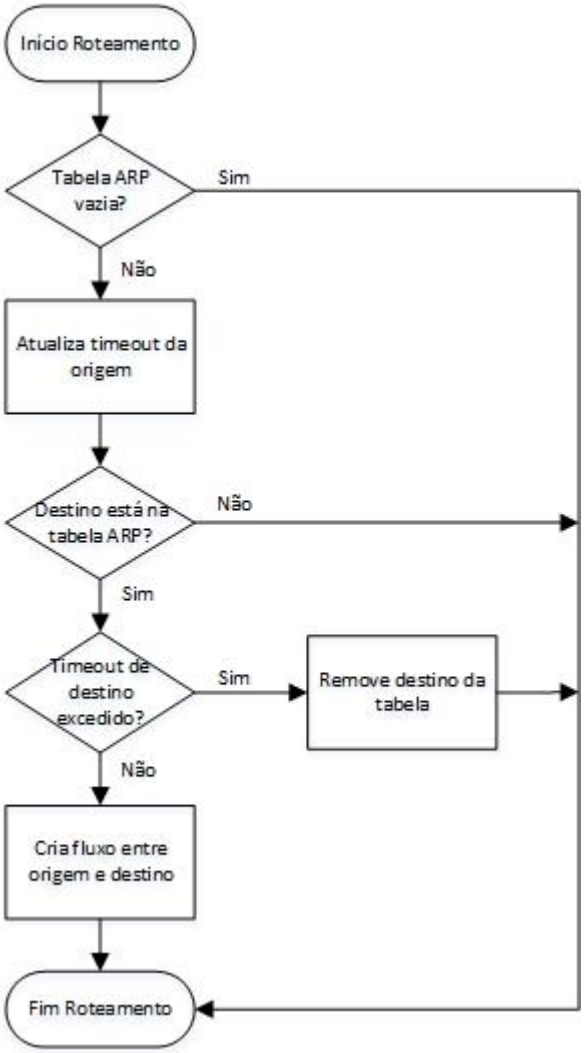


Figura 11 Fluxo roteamento

5 RESULTADO

No decorrer dos testes foi verificada a conectividade entre os hosts da estrutura conforme Figura 12, onde todos os hosts da estrutura respondem a ping para os demais, mostrando conectividade fim-a-fim entre todos os equipamentos.

Figura 12 Resultado do pingall no Mininet

Além da análise de conectividade foram coletados os logs de tráfego durante testes de ping podendo observar os pacotes trocados entre origem e destino para diferentes casos.

Os casos verificados tiveram como origem um host cuja interface de origem do ping estava diretamente conectada ao switch OpenFlow e casos variados de destino, um host com interface na mesma rede, um host de rede distinta conectado ao switch OpenFlow, um host remoto diretamente conectado a um host conectado ao switch OpenFlow.

Neste último caso foram analisadas duas situações onde o primeiro teste foi realizado para o IP da rede de enlace com o host conectado ao switch OpenFlow e o outro para outra rede configurada neste host. Segue abaixo cada um dos casos detalhados.

1. h2 para h3

Neste primeiro caso a origem é 10.0.2.1 e o destino, 10.0.2.2, ambos da rede 10.0.2.0/24. A Figura 13 mostra o resultado do ping onde, dos 4 pacotes enviados, 100% dos pacotes foram recebidos pelo destino. Em destaque é possível verificar os quadros de ARP request e replay, procurando o endereço de MAC do IP de destino, o reply é enviado para a origem com o endereço e desta forma é iniciada a comunicação IP, visualizando-se os quatro ICMP (*Internet Control Message Protocol*) request e reply no host h3.

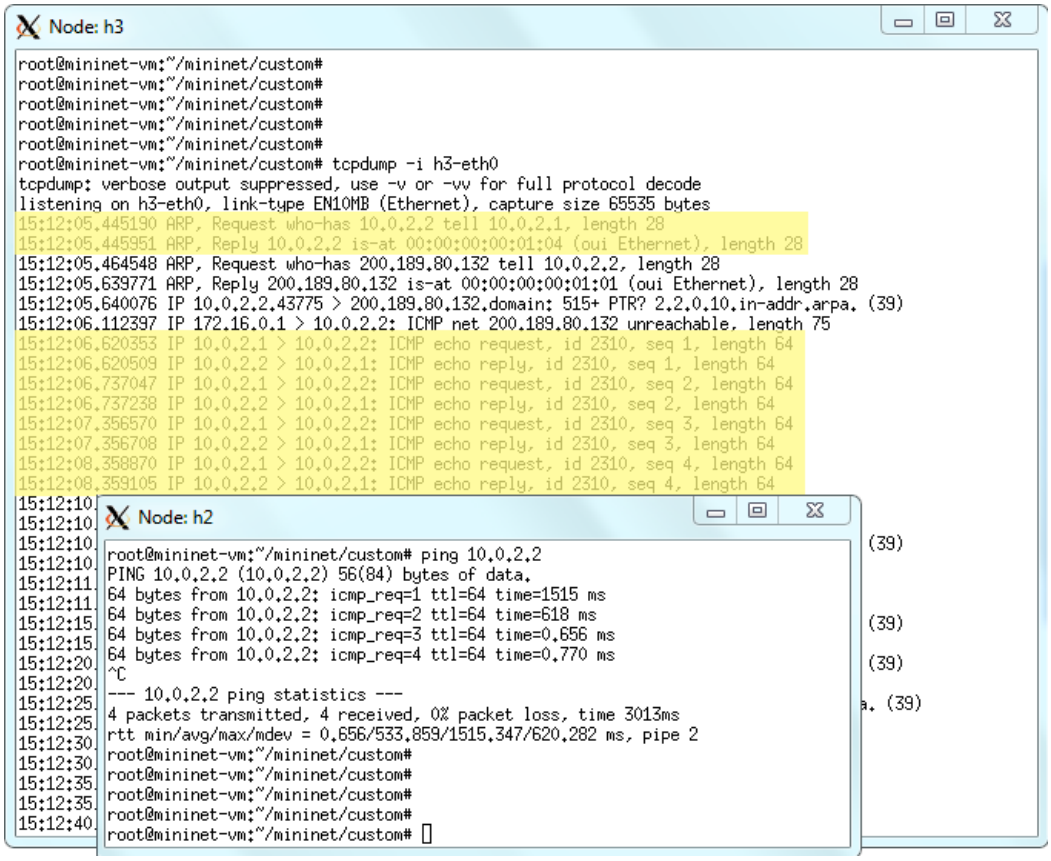


Figura 13 Node: h3 - Monitoramento de tráfego da interface h3-eth0 / Node: h2 – Ping

Na Figura 14 observa-se o tráfego gerado no switch OpenFlow, através do wireshark monitorando as interfaces cujos hosts h2 e h3 estão conectados.

Todo o tráfego é duplicado pois é a coleta é realizada nas duas interfaces.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:01:03	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.1
2	0.099179000	00:00:00_00:01:03	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.1
3	0.100034000	00:00:00_00:01:04	00:00:00_00:01:03	ARP	42	10.0.2.2 is at 00:00:00:00:01:04
8	1.002498000	00:00:00_00:01:03	Broadcast	ARP	42	Who has 10.0.2.2? Tell 10.0.2.1
9	1.136660000	00:00:00_00:01:04	00:00:00_00:01:03	ARP	42	10.0.2.2 is at 00:00:00:00:01:04
10	1.136946000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=1/256, ttl=64
11	1.136987000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=2/512, ttl=64
12	1.274326000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=1/256, ttl=64
13	1.274566000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=1/256, ttl=64
14	1.391006000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=2/512, ttl=64
15	1.391317000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=2/512, ttl=64
16	1.514542000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=1/256, ttl=64
17	1.626819000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=2/512, ttl=64
18	2.010404000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=3/768, ttl=64
19	2.010864000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=3/768, ttl=64
20	2.010542000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=3/768, ttl=64
21	2.010801000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=3/768, ttl=64
22	3.012742000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=4/1024, ttl=64
23	3.012863000	10.0.2.1	10.0.2.2	ICMP	98	Echo (ping) request id=0x0906, seq=4/1024, ttl=64
24	3.013271000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=4/1024, ttl=64
25	3.013350000	10.0.2.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0906, seq=4/1024, ttl=64

Figura 14 Comunicação entre h2 e h3 coletadas via Wireshark no switch OpenFlow

2. h2 ping h1

Neste caso a origem é 10.0.2.1 e o destino, 10.0.1.2, de redes distintas, porém ambos conectados ao mesmo switch OpenFlow. A Figura 15 mostra o resultado do ping onde, dos 4 pacotes enviados, 100% foram recebidos pelo destino. Em destaque é possível verificar os quadros de ARP e ICMP (request e replay).No ARP request requisita-se o endereço de MAC do IP de destino, o reply é enviado para a origem com o MAC address e desta forma é iniciada a comunicação IP através do protocolo ICMP.

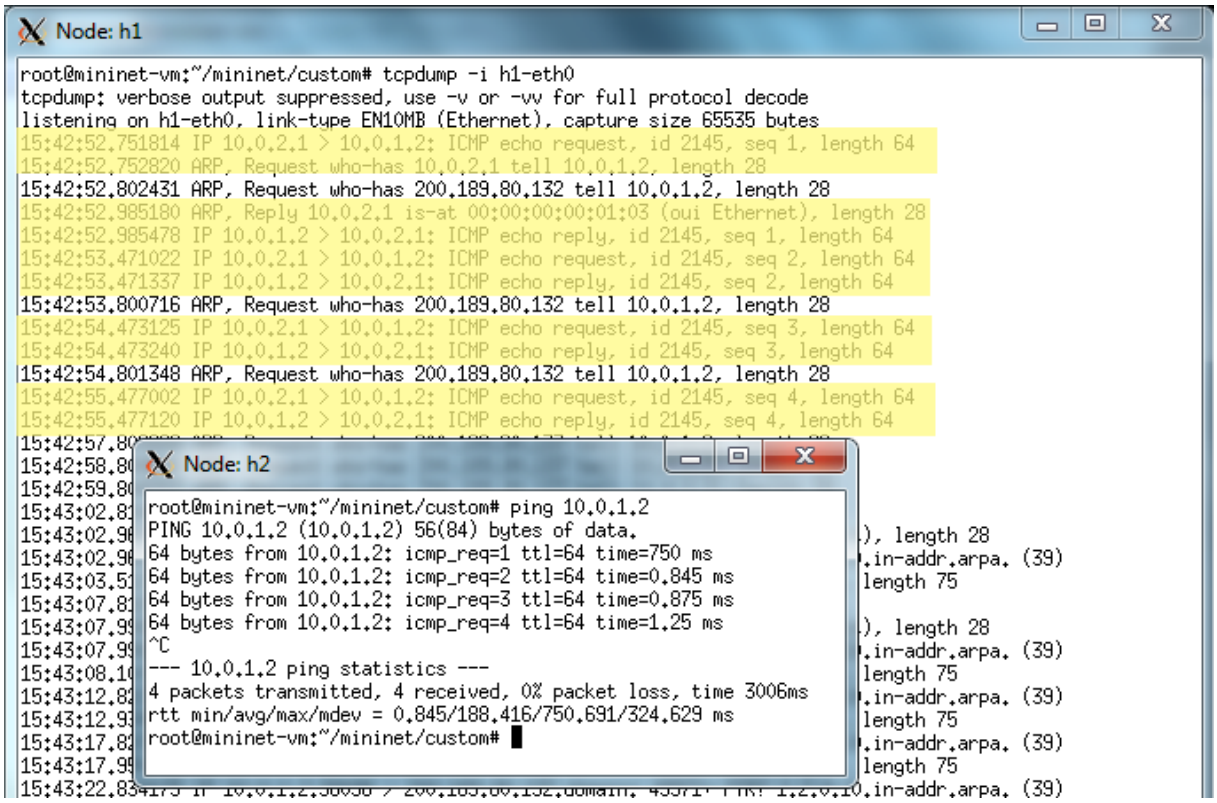


Figura 15 Node: h1 - Monitoramento de tráfego da interface h1-eth0 / Node: h2 – Ping

Na Figura 16 observa-se o tráfego gerado no switch OpenFlow, através do wireshark monitorando as interfaces cujos hosts h1 e h2 estão conectados, todo o tráfego é duplicado devido a coleta nas duas interfaces.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:01:03	Broadcast	ARP	42	Who has 10.0.1.2? Tell 10.0.2.1
2	0.157557000	00:00:00_00:01:02	00:00:00_00:01:03	ARP	42	10.0.1.2 is at 00:00:00:00:01:02
3	0.157764000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=1/256, ttl=64
4	0.281440000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=1/256, ttl=64
5	0.282585000	00:00:00_00:01:02	Broadcast	ARP	42	Who has 10.0.2.1? Tell 10.0.1.2
7	0.514801000	00:00:00_00:01:03	00:00:00_00:01:02	ARP	42	10.0.2.1 is at 00:00:00:00:01:03
8	0.515219000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=1/256, ttl=64
10	0.750393000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=1/256, ttl=64
11	1.000521000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=2/512, ttl=64
12	1.001160000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=2/512, ttl=64
13	1.000659000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=2/512, ttl=64
14	1.001093000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=2/512, ttl=64
17	2.002549000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=3/768, ttl=64
18	2.003267000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=3/768, ttl=64
19	2.002757000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=3/768, ttl=64
20	2.003068000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=3/768, ttl=64
23	3.005826000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=4/1024, ttl=64
24	3.006907000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=4/1024, ttl=64
25	3.006401000	10.0.2.1	10.0.1.2	ICMP	98	Echo (ping) request id=0x0861, seq=4/1024, ttl=64
26	3.006859000	10.0.1.2	10.0.2.1	ICMP	98	Echo (ping) reply id=0x0861, seq=4/1024, ttl=64

Figura 16 Comunicação entre h2 e h1 coletadas via Wireshark no switch OpenFlow

3. h2 ping h4

Neste caso a origem é 10.0.2.1 e o destino, 172.16.0.1, redes distintas, a primeira diretamente conectada ao switch OpenFlow e a segunda acessível através do host h0, por uma rede diretamente conectada ao h0 na interface não conectada ao switch OpenFlow. A Figura 17 mostra o resultado do ping onde, dos 4 pacotes enviados, 100% dos pacotes foram recebidos pelo destino. Em destaque é possível verificar os quadros de ARP request e replay, procurando o endereço de MAC do IP de destino, o reply é enviado para a origem com o endereço e desta forma é iniciada a comunicação IP, visualizando-se os quatro ICMP request e reply no host h4.

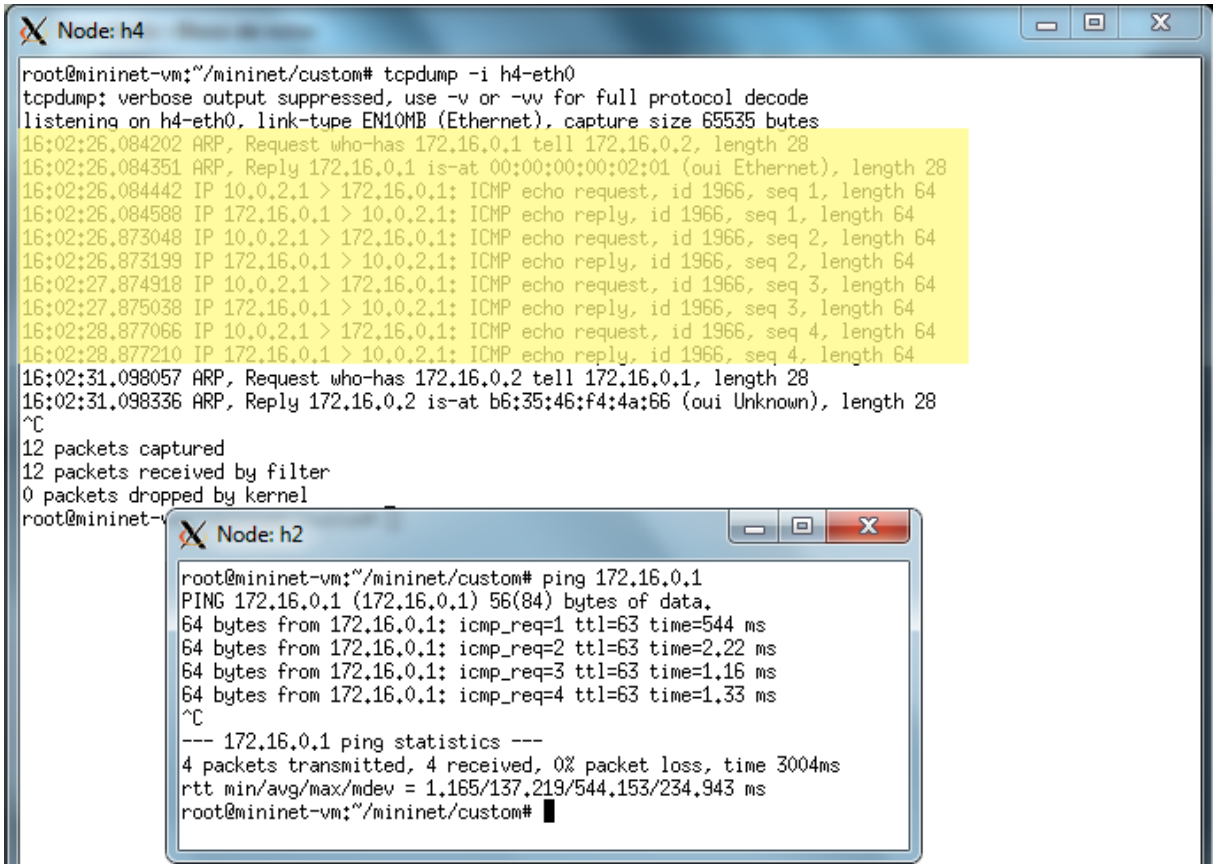


Figura 17 Node: h4 - Monitoramento de tráfego da interface h4-eth0 / Node: h2 – Ping

Na Figura 18 observa-se o tráfego gerado no switch OpenFlow, através do wireshark monitorando as interfaces cujos hosts h2 e h0 estão conectados. Todo o tráfego é duplicado devido coleta em duas interfaces distintas.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:01:03	Broadcast	ARP	42	Who has 172.16.1.1? Tell 10.0.2.1
2	0.131339000	00:00:00_00:01:01	00:00:00_00:01:03	ARP	42	172.16.1.1 is at 00:00:00:00:01:01
3	0.131617000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=1/256, ttl=64
4	0.329862000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=1/256, ttl=64
5	0.330740000	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.2.1? Tell 10.0.1.1
6	0.663676000	00:00:00_00:01:03	00:00:00_00:01:01	ARP	42	10.0.2.1 is at 00:00:00:00:01:03
7	0.664068000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=1/256, ttl=63
8	0.891452000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=1/256, ttl=63
9	1.000189000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=2/512, ttl=64
10	1.000943000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=2/512, ttl=63
11	1.000281000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=2/512, ttl=64
12	1.000870000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=2/512, ttl=63
13	2.003459000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=3/768, ttl=64
14	2.004466000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=3/768, ttl=63
15	2.003582000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=3/768, ttl=64
16	2.004395000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=3/768, ttl=63
17	3.006400000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=4/1024, ttl=64
18	3.007693000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=4/1024, ttl=63
19	3.006611000	10.0.2.1	172.16.1.1	ICMP	98	Echo (ping) request id=0x07f7, seq=4/1024, ttl=64
20	3.007619000	172.16.1.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07f7, seq=4/1024, ttl=63

Figura 18 Comunicação entre h2 e h0 coletadas via Wireshark no switch OpenFlow

4. h2 ping h4

Neste caso a origem é 10.0.2.1 e o destino, 172.16.1.1, redes distintas, a primeira diretamente conectada ao switch OpenFlow e a segunda acessível através do host h0 que está conectado a redes externas através de rotas estáticas. A Figura 19 mostra o resultado do ping onde, dos 4 pacotes enviados, 100% dos pacotes foram recebidos pelo destino. Em destaque é possível verificar os quadros de ARP request e replay, procurando o endereço de MAC do IP de destino, o reply é enviado para a origem com o endereço e desta forma é iniciada a comunicação IP, visualizando-se os quatro ICMP request e reply no host h4.

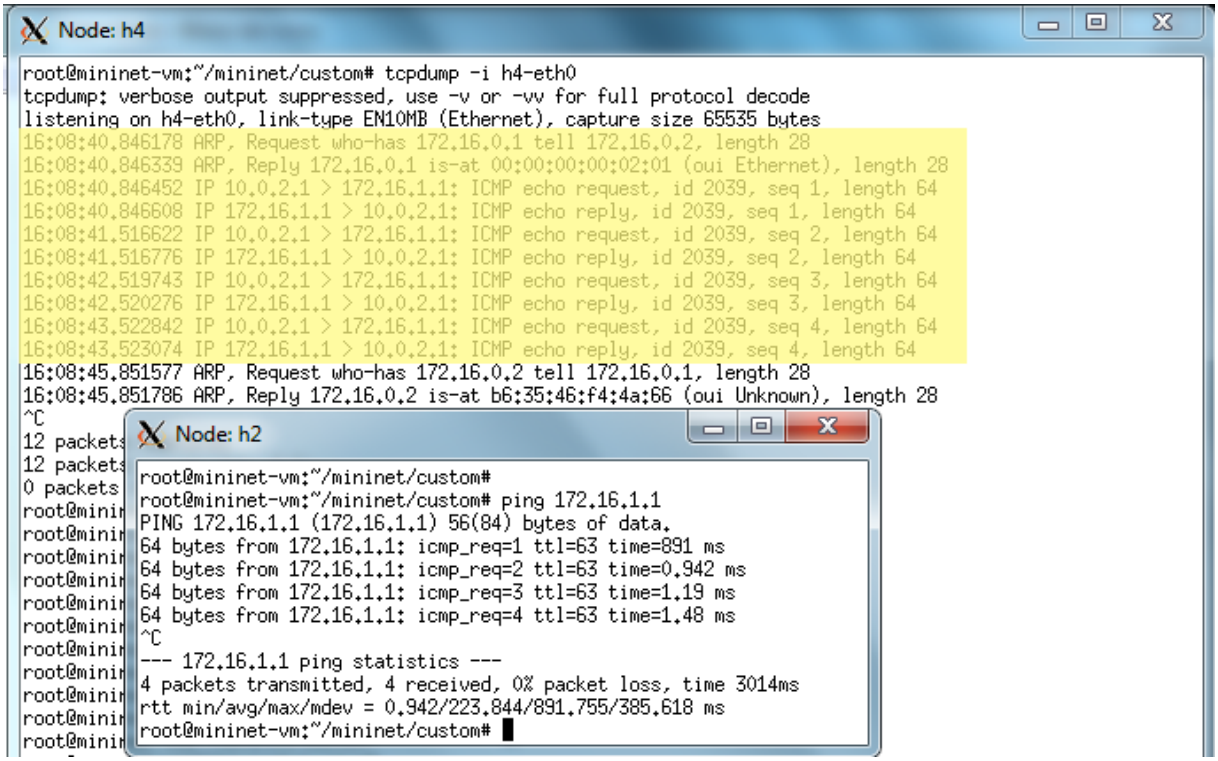
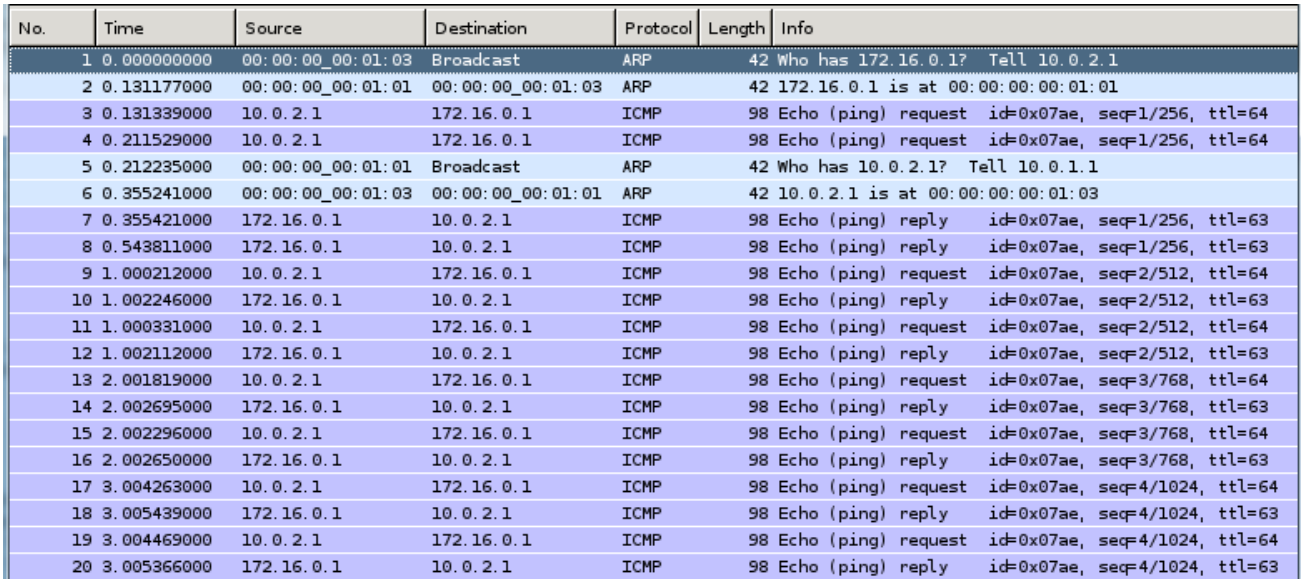


Figura 19 Node: h4 - Monitoramento de tráfego da interface h3-eth0 / Node: h2 – Ping

Na Figura 20 observa-se o tráfego gerado no switch OpenFlow, através do wireshark monitorando as interfaces cujos hosts h2 e h0 estão conectados, todo o tráfego é duplicado devido a coleta em duas interfaces.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:01:03	Broadcast	ARP	42	Who has 172.16.0.1? Tell 10.0.2.1
2	0.131177000	00:00:00_00:01:01	00:00:00_00:01:03	ARP	42	172.16.0.1 is at 00:00:00:00:01:01
3	0.131339000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=1/256, ttl=64
4	0.211529000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=1/256, ttl=64
5	0.212235000	00:00:00_00:01:01	Broadcast	ARP	42	Who has 10.0.2.1? Tell 10.0.1.1
6	0.355241000	00:00:00_00:01:03	00:00:00_00:01:01	ARP	42	10.0.2.1 is at 00:00:00:00:01:03
7	0.355421000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=1/256, ttl=63
8	0.543811000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=1/256, ttl=63
9	1.000212000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=2/512, ttl=64
10	1.002246000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=2/512, ttl=63
11	1.000331000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=2/512, ttl=64
12	1.002112000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=2/512, ttl=63
13	2.001819000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=3/768, ttl=64
14	2.002695000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=3/768, ttl=63
15	2.002296000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=3/768, ttl=64
16	2.002650000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=3/768, ttl=63
17	3.004263000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=4/1024, ttl=64
18	3.005439000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=4/1024, ttl=63
19	3.004469000	10.0.2.1	172.16.0.1	ICMP	98	Echo (ping) request id=0x07ae, seq=4/1024, ttl=64
20	3.005366000	172.16.0.1	10.0.2.1	ICMP	98	Echo (ping) reply id=0x07ae, seq=4/1024, ttl=63

Figura 20 Comunicação entre h2 e h0 coletadas via Wireshark no switch OpenFlow

Nas Figuras 13, 15, 17 e 19, onde se encontra a execução do ping, observa-se em todos os casos que o tempo de resposta do primeiro pacote foi maior que os seguintes, variando entre 0,5 e 1,5 s, enquanto os demais foram na ordem de 0,6 e 2,2 ms. Esse processo se deve a construção da tabela ARP, pois o destino desejado não se encontrava na tabela, sendo necessária a busca do endereço MAC na rede.

Somente no primeiro caso os 2 primeiros pacotes apresentaram tempo de resposta alto, pois o tempo de resposta ARP foi mais elevado, demais casos somente o primeiro pacote teve tempo de resposta elevado.

Nos demais tempos de resposta para comunicação com a mesma rede o tempo foi menor que 1 ms; rede diferente diretamente conectada ao switch OpenFlow, aproximadamente 1 ms; rede diferente não conectada diretamente ao switch OpenFlow, em sua maioria superior a 1 ms.

Na Figura 21 estão os logs do controlador do processo de construção da tabela ARP descrito abaixo:

1. Chega um ARP request no switch OpenFlow.
2. O endereço de MAC e IP da origem (10.0.2.1) são inseridos na tabela ARP.
3. Como o destino (10.0.2.2) não estava na tabela ARP foi gerado o ARP request e realizado o broadcast na rede.
4. Destino responde ao request.
5. O endereço de MAC e IP de destino (10.0.2.2) são inseridos na tabela ARP.

6. Chega outro ARP da origem e como o destino já se encontra na tabela ARP inicia-se a comunicação IP.

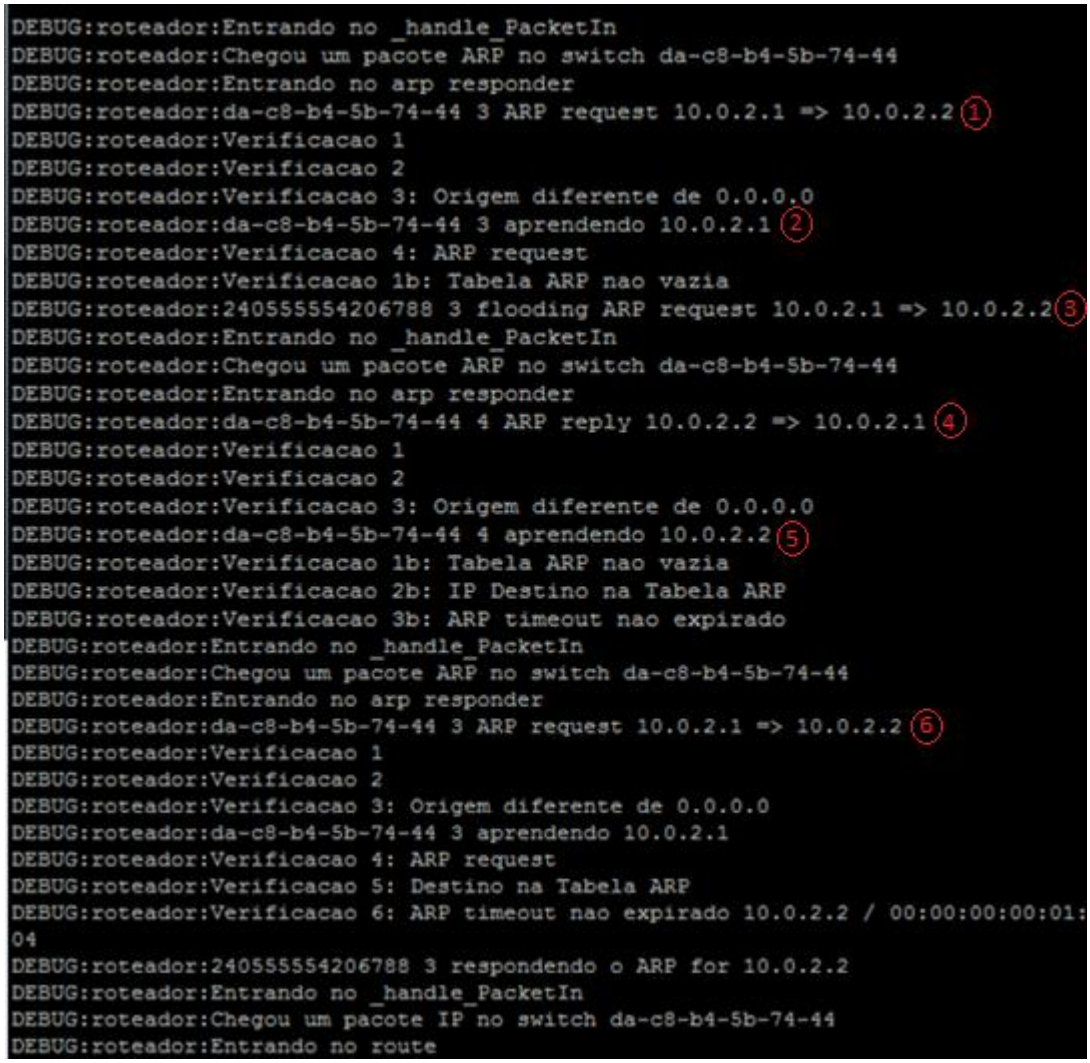


Figura 21 Construção da tabela ARP

Na Figura 22 estão os logs resultantes da comunicação IP, que passam pelas seguintes etapas:

1. Os endereços de MAC da origem e destino da requisição do ICMP são identificados.
2. A porta de origem e destino são identificadas.
3. É estabelecido o fluxo entre origem e destino e a requisição é transmitida.
4. Os endereços de MAC da origem e destino da resposta do ICMP são identificados.

5. A porta de origem e destino são identificadas.
6. É estabelecido o fluxo entre origem e destino e a resposta é transmitida.

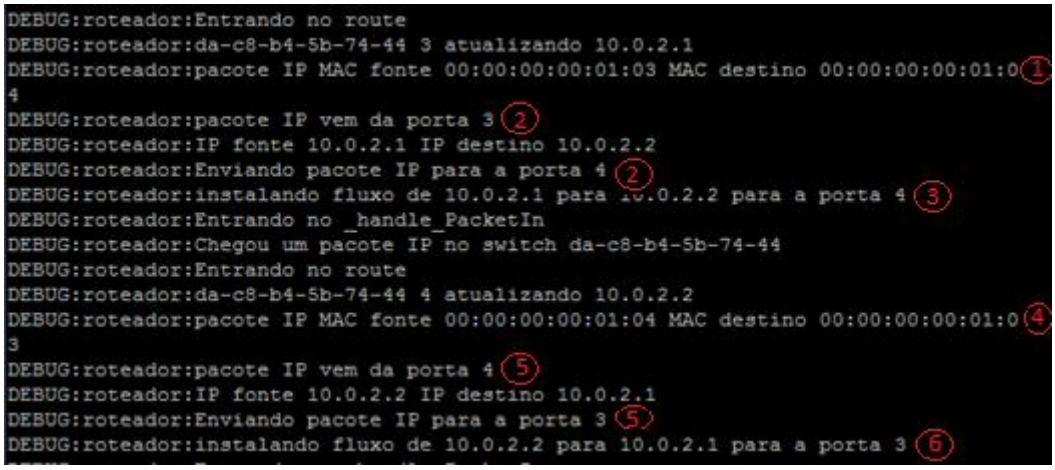


Figura 22 Comunicação IP

A topologia simulada permitiu visualizar um outro paradigma para comunicação em redes, permitindo uma flexibilidade de configuração muito maior do que as redes tradicionais. Essa flexibilidade ao mesmo tempo se apresenta como uma vantagem pode ser uma desvantagem em relação à autonomia do programador da rede, que necessita de uma atenção muito maior para não gerar indisponibilidades nos ambientes.

Em ambientes de datacenter multiusuário onde são necessários diversos equipamentos independentes para composição da estrutura, o conceito de administração centralizada presente nas SDNs apresenta-se como uma grande vantagem.

6 CONCLUSÃO

Este trabalho apresenta uma rede baseada em software com um controlador que simula um roteador, desenvolvido com o intuito de oferecer uma solução para data centers multiusuários.

A estrutura foi implementada usando o simulador mininet e o controlador de rede utilizado foi o POX. O mininet permitiu a criação do ambiente desejado, com duas redes pertencentes a diferentes usuários, que se comunicam com redes externas a estrutura SDN;

O experimento realizado demonstrou que é possível criar uma rede programável para ambientes que só eram estruturados como redes tradicionais, dando maior autonomia para o administrador da rede, que pode programar a rede com as necessidades especificas do ambiente.

Além disso a possibilidade de criação de um ambiente hibrido, com rede tradicional e SDN, é essencial para redes que atualmente são totalmente tradicionais possam implementar SDN aos poucos, pois o processo de mudança de tecnologia precisa ser feito paralelamente, pois a criticidade dos ambientes de rede não permitem grandes períodos de indisponibilidade.

Também em ambientes reais de data centers multiusuários, o uso de SDN é de interesse devido o ambiente com gerenciamento centralizado, minimizando a quantidade de configuração e esforço necessário para realiza-la, pois trata-se de uma rede complexa com muitos ativos envolvidos.

Em trabalhos futuros outros tipos de estruturas e controladores podem ser simulados a fim de permitir uma comparação entre eles e definição de qual o mais adequado para os diferentes cenários.

Os diferentes cenários a serem simulados podem ser:

- Duas redes SDN distintas conectadas através de uma rede tradicional, com rotas estáticas para comunicação entre as estruturas.
- Ampliar a rede SDN, aumentando o número de usuários e controladores interligados de maneiras distintas.
- A composição das estruturas acima, interligar duas redes SDN complexas com o uso de roteamento estático para comunicação entre elas.

REFERÊNCIAS

[1] JUNIOR, C. C. **Informática, Internet e Aplicativos**. Curitiba: Ibpex, 2007.

[2] MENDES, D.R. In: Redes de Computadores – Teoria e Prática: **Introdução às Redes de Computadores**. São Paulo: Novatec, 2007.p.17-38.

[3] TANENBAUM, A.S. **Redes de computadores**. Rio de Janeiro: Elsevier, 2003.

[4] PERES, A.; LOUREIRO, C. A. H.; SCHIMITT, M .A. R. **Redes de computadores II: níveis de transporte e rede**. Porto Alegre: Bookman, 2014.

[5] TANENBAUM, A.S. **5. A Camada de Rede.** In: TANENBAUM, A.S. Redes de computadores. Rio de Janeiro: Elsevier, 2003.

[6] FOROUZAN, B. A. **Protocolo TCP/IP.** Porto Alegre: AMGH, 2010.

[7] **Data Center**. Disponível em: <http://www.telecorp.com.br/glossario/data-center/>. Acesso em: 11 mai. 2014.

[8] **Network Protocols Handbook**. Saratoga: Javvin Technologies, 2005.

[9] **Cisco data Center infrastructure 2.5. Design Guide**. Disponível em: <http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCI_SRNDa.pdf>. Acesso em: 11 mai. 2014.

[10] NADEAU, T.D.; GRAY,K. SDN: **Software Defined Networks**. O'Reilly, 2013.

[11] ROTHENBERG, C. E.; NASCIMENTO, M. R.; SALVADOR, M. R.; MAGALHÃES; M. F. **OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes.**

[12] **NOX**. Disponível em: <http://www.noxrepo.org/pox/about-pox/>. Acesso em: 11 mai. 2015.

[13] NADEAU, T. D.; GRAY, K. **SDN Software Defined Network.** Sebastopol: O'Reilly, 2013.

[14] **OPENFLOW - POX** WIKI. Disponível em: <https://openflow.stanford.edu/display/ONL/POX+Wiki#POXWiki-StockComponents>. Acesso em: 11 jul. 2015.

[15] **Maestro: maestro-platform**. Disponível em:<https://code.google.com/p/maestro-platform/>. Acesso em: 11 mai. 2015.

[16] COSTA, L.R. **OpenFlow e o Paradigma de Redes Definidas por Software**. Brasília: UnB, 2013. 305 p.

[17] GUEDES, D.; VIEIRA, L.F.M.; VIEIRA, M.M.; RODRIGUES, H.; NUNES, R.V. Livro Texto dos Minicursos do XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos: **Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento das pesquisas em Redes de Computadores**. Ouro Preto: Sociedade Brasileira de Computação, 2012.

[18] **Mininet Overview**. Disponível em: <http://mininet.org/overview/>. Acesso em: 05 jun. 2015.

ANEXO A

A.1 Código para criação da Topologia

A seguir é apresentado o código para criação da topologia simulada com comentários:

```
##########################################
#                                        #
# Arquivo: topologia.py                  #
# Elaborado por: Nathalia Levorato       #
# Trabalho de Graduacao          #
#                                        #
##########################################

fromminine t.topo import Topo

class MyTopo( Topo ):

def __init__( self ):

# Inicializando a topologia
Topo.__init__( self )

    # Configurandoos hosts - nome,mac-address,ip,gateway
    host0 = self.addHost( 'h0', mac='00:00:00:00:01:01', ip='10.0.1.1/24' )
    host1 = self.addHost( 'h1', mac='00:00:00:00:01:02', ip='10.0.1.2/24', defaultRoute='h1-eth0' )
    host2 = self.addHost( 'h2', mac='00:00:00:00:01:03', ip='10.0.2.1/24', defaultRoute='h2-eth0' )
    host3 = self.addHost( 'h3', mac='00:00:00:00:01:04', ip='10.0.2.2/24', defaultRoute='h3-eth0' )
    host4 = self.addHost( 'h4', mac='00:00:00:00:02:01', ip='172.16.0.1/30' )

    # Configurando o switch OpenFlow - nome,mac-address
    switch0 = self.addSwitch( 's0', mac='00:00:00:00:00:01')

# Estabelecendo a conexao entre os componentes
self.addLink( host0, switch0 )
self.addLink( host1, switch0 )
self.addLink( host2, switch0 )
self.addLink( host3, switch0 )
self.addLink( host0, host4 )

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

A.2 Comandos para criação da Topologia

A seguir são apresentados os comandos para complementar a topologia simulada com comentários:

```
# Configurando o segundo IP no host h0
h0 ifconfig h0-eth1 172.16.0.2 netmask 255.255.255.252
# Adicionando rota para a rede 10.0.0.0/8 no host h0com destino para h0-eth0
h0 route add -net 10.0.0.0 netmask 255.0.0.0 h0-eth0
# Adicionando rota default no host h0 com destino para 172.16.0.1
h0 route add default gw 172.16.0.1
# Adicionando rota para a rede 10.0.0.0/8 no host h4 com destino para 172.16.0.2
h4 route add -net 10.0.0.0 netmask 255.0.0.0 gw 172.16.0.2
# Habilitando encaminhamento de pacotes no host h0 – comportamento de roteador
h0sysctl -w net.ipv4.ip_forward=1
# Configurando o segundo IP no host h4
```

```
h4 ifconfig h4-eth0:1 172.16.1.1 netmask 255.255.255.0
# Configurando o segundo IP no host h4
h4 ifconfig h4-eth0:2 172.16.2.1 netmask 255.255.255.0
# Configurando o segundo IP no host h4
h4 ifconfig h4-eth0:3 172.16.3.1 netmask 255.255.255.0
```

A.3 Código de configuração do controlador

A seguir é apresentada a programação do controlador:

```
##########################################
#                                        #
# Arquivo:roteador.py                    #
# Elaborado por: Nathalia Levorato       #
# Trabalho de Graduacao          #
#                                        #
##########################################


frompox.core import core
frompox.lib.util import dpidToStr
frompox.lib.revent import *
import pox.openflow.libopenflow_01 as of
importpox.lib.packet as pkt
import pox
frompox.lib.packet.ethernet import ethernet
from pox.lib.packet.ipv4 import ipv4, IP_ANY
frompox.lib.packet.arp import arp
frompox.lib.revent import *
frompox.lib.addresses import *
import time

FLOW_IDLE_TIMEOUT = 10

ARP_TIMEOUT = 60 * 2

log = core.getLogger()

class Router(object):
def __init__ (self, connection):
log.debug("Iniciando o switch")
self.connection = connection
connection.addListeners(self)
self.arp_table = {}
self.table_ip = {}

defarp_responder (self, packet_in, packet, dpid):
log.debug("Entrando no arp responder")
packet_arp = packet.next
log.debug("%s %i ARP %s %s => %s ", dpidToStr(dpid), packet_in.in_port,
{arp.REQUEST:"request",arp.REPLY:"reply"}.get(packet_arp.opcode, 'op:%i' %(packet_arp.opcode,)),
str(packet_arp.protosrc), str(packet_arp.protodst))
ifpacket_arp.prototype == arp.PROTO_TYPE_IP:
log.debug("Verificacao 1")
ifpacket_arp.hwtype == arp.HW_TYPE_ETHERNET:
log.debug("Verificacao 2")
ifpacket_arp.protosrc != IP_ANY:
log.debug("Verificacao 3: Origem diferente de 0.0.0.0")
```

```
log.debug("%s %iaprendendo %s ", dpidToStr(dpid), packet_in.in_port, packet_arp.protosrc)
self.arp_table[packet_arp.protosrc] = (packet_in.in_port, packet.src, time.time() + ARP_TIMEOUT)
ifpacket_arp.opcode == arp.REQUEST:
log.debug("Verificacao 4: ARP request")
ifpacket_arp.protodst in self.arp_table:
log.debug("Verificacao 5: Destino na Tabela ARP")
if not time.time() >self.arp_table[packet_arp.protodst][2]:
log.debug("Verificacao 6: ARP timeout naoexpirado %s / %s", str(packet_arp.protodst),
str(self.arp_table[packet_arp.protodst][1]))
reply = arp()
reply.hwtype = packet_arp.hwtype
reply.prototype = packet_arp.prototype
reply.hwlen = packet_arp.hwlen
reply.protolen = packet_arp.protolen
reply.opcode = arp.REPLY
reply.hwdst = packet_arp.hwsrc
reply.protodst = packet_arp.protosrc
reply.protosrc = packet_arp.protodst
reply.hwsrc = self.arp_table[packet_arp.protodst][1]
ether = ethernet(type=packet.type, src=reply.hwsrc, dst=packet_arp.hwsrc)
ether.set_payload(reply)
log.debug("%i %irespondendo o ARP for %s" %(dpid, packet_in.in_port, str(reply.protosrc)))
msg = of.ofp_packet_out()
msg.data = ether.pack()
msg.actions.append(of.ofp_action_output(port = of.OFPP_IN_PORT))
msg.in_port = packet_in.in_port
self.connection.send(msg)
return
iflen(self.arp_table.keys()) != 0:
log.debug("Verificacao 1b: Tabela ARP nao vazia")
ifpacket_arp.protodst in self.arp_table:
log.debug("Verificacao 2b: IP Destino na Tabela ARP")
if not time.time() >self.arp_table[packet_arp.protodst][2]:
log.debug("Verificacao 3b: ARP timeout naoexpirado")
return

log.debug("%i %i flooding ARP %s %s => %s" %(dpid, packet_in.in_port,
   {arp.REQUEST:"request",arp.REPLY:"reply"}.get(packet_arp.opcode,                    'op:%i'
%(packet_arp.opcode,)), str(packet_arp.protosrc), str(packet_arp.protodst)))
msg = of.ofp_packet_out()
msg.in_port = packet_in.in_port
action = of.ofp_action_output(port = of.OFPP_FLOOD)
msg.actions.append(action)
ifpacket_in.buffer_id != -1 and packet_in.buffer_id is not None:
msg.buffer_id = packet_in.buffer_id
else:
ifpacket_in.data is None:
return
msg.data = packet_in.data
self.connection.send(msg)
return

def route(self, packet_in, packet, dpid):
log.debug("Entrando no route")
packet_ip = packet.next
iflen(self.arp_table.keys()) == 0:
log.warning("Tabela ARP vazia, pacote ignorado")
return
log.debug("%s %iatualizando %s", dpidToStr(dpid), packet_in.in_port, packet_ip.srcip)
self.arp_table[packet_ip.srcip] = (packet_in.in_port, packet.src, time.time() + ARP_TIMEOUT)
```

```
ifpacket_ip.dstip in self.arp_table:
if not time.time() >self.arp_table[packet_ip.dstip][2]:
port = self.arp_table[packet_ip.dstip][0]
log.debug("pacote IP MAC fonte %s MAC destino %s" %(packet.src, packet.dst))
log.debug("pacote IP vem da porta %i" %(packet_in.in_port))
log.debug("IP fonte %s IP destino %s" %(packet_ip.srcip,packet_ip.dstip))
log.debug("Enviando pacote IP para a porta %i" %(port))
msg = of.ofp_flow_mod()
msg.match = of.ofp_match.from_packet(packet, packet_in.in_port)
log.debug("instalando fluxo de %s para %s para a porta %i" %(packet_ip.srcip, packet_ip.dstip, port))
msg.idle_timeout = FLOW_IDLE_TIMEOUT
   msg. hard_timeout = 30
ifpacket_in.buffer_id != -1 and packet_in.buffer_id is not None:
msg.buffer_id = packet_in.buffer_id
else:
ifpacket_in.data is None:
return
msg.data = packet_in.data
msg.actions.append(of.ofp_action_output(port = port))
self.connection.send(msg)
return
else:
log.warning("Time do ARP expirado!!!")
delself.arp_table[packet_ip.dstip]
return
else:
log.warning("ARP naoencontradonatabela!!!")
def _handle_PacketIn (self,event):
log.debug("Entrando no _handle_PacketIn")
packet_eth = event.parsed
packet_in = event.ofp
dpid = event.dpid
if not packet_eth.parsed:
log.warning("Pacote foi ignorado pois nao ha cabecalhos")
return
ifpacket_eth.type == ethernet.LLDP_TYPE:
return

ifpacket_eth.type == pkt.ethernet.ARP_TYPE:
log.debug("Chegou um pacote ARP no switch %s", dpidToStr(event.dpid))
self.table_ip[packet_eth.next.protosrc] = (True, time.time() + ARP_TIMEOUT)
ifpacket_eth.next.protodst in self.table_ip:
if not self.table_ip[packet_eth.next.protodst][0]:
iftime.time() >self.table_ip[packet_eth.next.protodst][1]:
self.arp_table[packet_eth.next.protodst] = (1, EthAddr("00:00:00:00:01:01"), time.time() +
ARP_TIMEOUT)
else:
iftime.time() >self.table_ip[packet_eth.next.protodst][1]:
self.table_ip[packet_eth.next.protodst] = (False,self.table_ip[packet_eth.next.protodst][1])
else:
self.table_ip[packet_eth.next.protodst] = (False,time.time() + 5)
self.arp_responder(packet_in, packet_eth, dpid)
else:
log.debug("Chegou um pacote IP no switch %s", dpidToStr(event.dpid))
self.route(packet_in, packet_eth, dpid)
return
def launch ():
log.debug("entrando launch")
defstart_switch (event):
log.debug("entrandostart_switch")
```

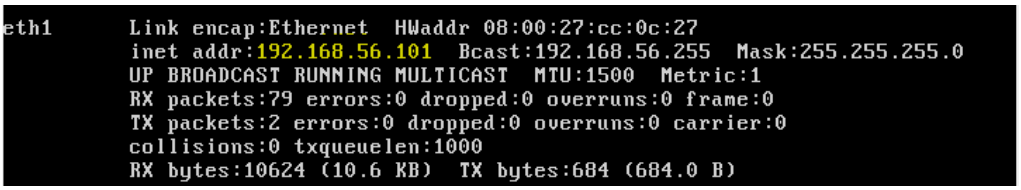
```
Router(event.connection)
core.openflow.addListenerByName("ConnectionUp", start_switch)
```

ANEXO B

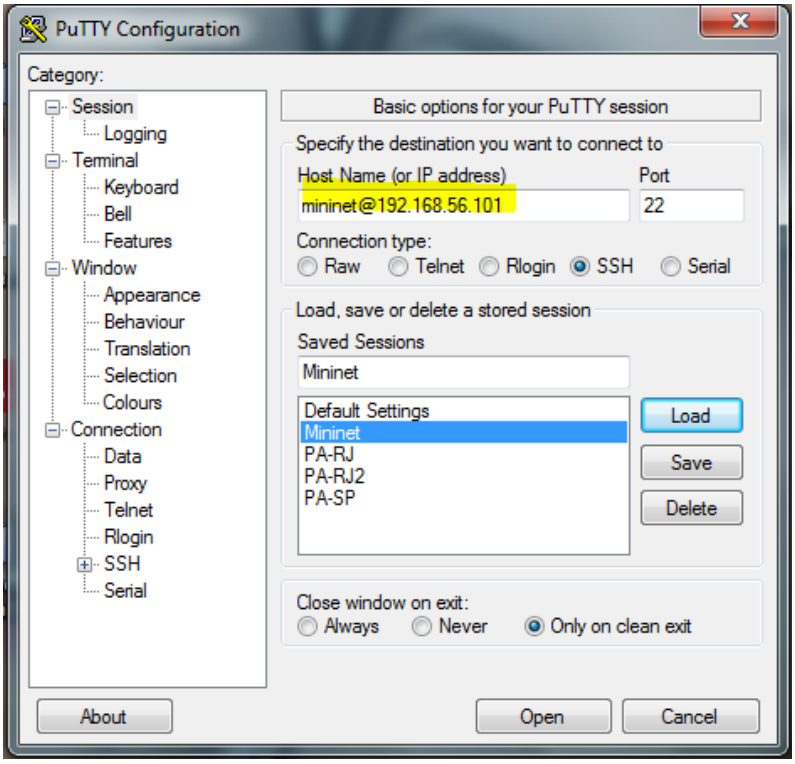
B.1 Instalando e configurando o Mininet

1. Baixada a imagem da VM mininet (https://github.com/mininet/mininet/wiki/Mininet-VM-Images).
2. Baixado e instalado o VirtualBox. (https://www.virtualbox.org/wiki/Downloads).
3. Baixado e instalado o servidor Xming X para Windows (http://sourceforge.net/projects/xming/files/latest/download).
4. Baixado o putty (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html).

   No putty ativada a opção Enable X11 forwarding em Connection --> SSH --> X11.
5. Conectado na VM com o usuário mininet, senha mininet.
6. Verificado o IP da VM com o comando ifconfig.



7. Acessada a VM através do putty.



8. Criado o arquivo roteador.py do controlador no diretório /home/mininet/pox/ext/, com o conteúdo do anexo A3. Executado o controlador (./pox/pox.py log.level --DEBUG roteador).
9. Acessado a VM através do putty.
10. Executado o comando xterm e na nova janela. Acessado o diretório (cd /home/mininet/mininet/custom/).
11. Criado o arquivo topologia.py com o conteúdo do anexo A1. Executada a topologia (sudo mn --custom topologia.py --topo mytopo --mac --switch ovsk --controller remote).
12. Realizado os testes no ambiente.