UNIVERSIDADE FEDERAL DO CEARÁ

ANTÔNIO JANAEL PINHEIRO

ESCALABILIDADE DO PLANO DE CONTROLE EM REDES OPENFLOW

QUIXADÁ - CEARÁ

2013

ANTÔNIO JANAEL PINHEIRO

ESCALABILIDADE DO PLANO DE CONTROLE EM REDES OPENFLOW

Monografia apresentada no Curso de Graduação em Redes de Computadores do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do grau de Tecnólogo em Redes de Computadores.

Orientador: Jeandro de Mesquita Bezerra

QUIXADÁ - CEARÁ

2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca do Campus de Quixadá

J72e Pinheiro, Antonio Janael
Escalabilidade do plano de controle de redes openflow / Antonio Janael Pinheiro – 2013.
41 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso Superior de Tecnologia em Redes de Computadores, Quixadá, 2013.
Orientação: Prof. MSc Jeandro de Mesquita Bezerra
Área de concentração: Computação

1. Redes de computadores 2. Redes digitais de serviços integrados 3. Arquitetura de redes de computadores I. Título.

CDD 004.6

ANTÔNIO JANAEL PINHEIRO

ESCALABILIDADE DO PLANO DE CONTROLE EM REDES OPENFLOW

Monografia apresentada no Curso de Graduação em Redes de Computadores do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial para obtenção do grau de Tecnólogo.

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Me. Jeandro de Mesquita Bezerra
Universidade Federal do Ceará – UFC
Orientador

Prof. Dr. Arthur de Castro Callado
Universidade Federal do Ceará – UFC

Prof. Me. Michel Sales Bonfim
Universidade Federal do Ceará – UFC

À meus pais, Antônio Ildevan Pinheiro e Maria Jocilêda Pinheiro e
irmãs, Roberta Maria Pinheiro e Maria Vitória Pinheiro,
pelo incentivo e apoio nos momentos difíceis.

AGRADECIMENTOS

Ao professor Jeandro de Mesquita Bezerra pela orientação e oportunidades de iniciação à pesquisa.

Aos amigos que conheci na UFC ao longo destes anos, pela ajuda e amizade. Em especial, aos colegas do curso de Redes de Computadores.

Aos professores que tive durante este curso de Redes de computadores da UFC, por sua dedicação, comprometimento e compreensão.

Aos familiares e amigos que contribuíram na realização dessa conquista.

A todas as pessoas que passaram pela minha vida e contribuíram para a construção de quem sou hoje.

“No meio da dificuldade encontra-se a oportunidade.”
Albert Einstein

RESUMO

Redes definidas por *software* representam uma evolução no funcionamento das redes de computadores. A realização de experimentos em redes se torna mais simples, e abre espaço para criação de novos serviços. O foco principal das redes definidas por *software* são as aplicações executadas no plano de controle. Essas aplicações realizam o controle, e implementam serviços oferecidos na rede. O plano de controle representa o cérebro da rede, já que as decisões quanto ao funcionamento da rede são de sua responsabilidade. A escalabilidade do plano de controle é limitada por fatores como: número de conexões suportadas pelo controlador, número de entradas na tabela de fluxos e quantidade de dados transmitida entre controlador e comutador. Procuramos minimizar a quantidade de entradas na tabela de fluxos, e o número de mensagens *flow-mod* do *Openflow*, como forma de aprimorar a escalabilidade.

Palavras-Chave: *Openflow*. Escalabilidade. Controle. *Mininet*. fluxos.

ABSTRACT

Software defined networks represent an evolution in the operation of computer networks. Conducting experiments in networks becomes simpler, and makes room for creation of new services. The Main software defined networks focus are applications running on the control plane. These applications perform control, and implement services offered by the network. The control plane is the heart of the network, since decisions on the operation of the network are your responsibility. The scalability of the control plane is limited by factors such as number of connections supported by the controller, number of entries in flow table, the amount of data transported secure channel, among others. We seek to minimize the amount of entries in the table flows as a way to improve scalability.

Keywords: Openflow. Scalability. Control. Mininet. Flows.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Introdução

Redes de computadores passaram de um recurso dispensável, para essencial nas operações críticas de muitas organizações, e fazem parte do cotidiano da sociedade. Na arquitetura típica da rede a realização de experimentos é dificultado, já que o funcionamento de muitas das operações críticas das organizações estão diretamente relacionadas ao correto funcionamento da rede em produção. As redes chegaram a um alto nível de complexidade, um ponto conhecido como "ossificação"da rede (MOREIRA et al., 2009). Essa complexidade dificulta a adição de inovação na área de redes de computadores.

Outras áreas como banco de dados, sistemas operacionais, etc. possuem princípios básicos, gerenciamento e evolução simples. As redes são formadas por um grande conjunto de protocolos, são difíceis de gerenciar e evoluem muito lentamente.

Geralmente abstrações auxiliam na simplificação de um "sistema". Em redes a principal abstração é a de camadas. Abstração de camadas lida apenas com o plano de dados, não há abstrações fortes no plano de controle. Em redes definidas por *software* (SDN, *software defined network*) são definidas três abstrações, distribuição, encaminhamento e configuração. SDN oferece controle centralizado da rede, através de uma camada de distribuição conhecida como sistema operacional de rede (NOS, *Network Operation System*)(GUDE et al., 2008). Através do NOS são obtidas informações e, enviados comandos de controle para os elementos da rede. As abstrações representam um modelo simplificado da rede. Programas de controle configuram um modelo abstrato, por sua vez a camada de controle compartilhado mapeia essa configuração para os dispositivos físicos. A abstração de encaminhamento simplifica a interação com dispositivos físicos. Os *switches* são gerenciados através da *Aplication Program Interface* (API) do *Openflow*(MCKEOWN et al., 2008).

Uma das propostas defendidas pelos idealizadores da arquitetura SDN é a simplificação da rede, através de abstrações. Abstraindo a infraestrutura física da rede, de forma a permitir o gerenciamento da rede através de APIs. Operadores poderão gerenciar a rede de forma transparente. O *Openflow* é uma API para gerenciamento do plano de dados, que permite controlar *switches* com comandos padronizados. A API do *Openflow* é usada pelo controlador para gerenciar os dispositivos e se comunicar com as aplicações (VAUGHAN-NICHOLS, 2011).

Pesquisadores têm total autonomia para modificar a lógica de funcionamento dos equipamentos com suporte a *Openflow*, em contra partida os desenvolvedores de *hardware* não precisam expor detalhes sobre o funcionamento interno de seu equipamento (SHENKER et al., 2011).

Atualmente fica-se limitado a um conjunto de protocolos, e ações suportados pelos equipamentos da rede, sendo preciso muitas vezes usar "remendos" nos protocolos aceitos pelo comutador, a fim de realizar tarefas que não são suportadas por padrão. Com a flexibilidade apresentada pela arquitetura SDN, é possível adicionar funcionalidades inexistentes no comutador.

O plano de controle representa o cérebro da rede, é nele que as decisões são tomadas. Limitações quanto a escalabilidade em uma rede *Openflow* passam pela capacidade do controlador de escalar. Aspectos como quantidade de entradas nas tabelas do *switch*, número de conexões simultâneas suportadas pelo controlador, uso do *link* entre *switch* e controlador impactam na capacidade de escalar de uma rede *Openflow*(LYNCH, 2013). A tabela de fluxos é armazenada na memória *Ternary Content Addressable Memory* (TCAM)(AGRAWAL; SHERWOOD, 2006) do *switch*. Essa memória tem capacidade de armazenamento limitada, seu esgotamento pode causar a falhas na rede. A redução de entradas na tabela do *switch* reduz o uso da memoria, possibilitando adição de mais máquinas na rede. Quanto ao *link* de comunicação entre *switch* e controlador, a redução de mensagens representa diminuição na utilização do *link*.

1.2 Objetivo Geral

Identificar oportunidades para o aprimoramento da escalabilidade no plano de controle de uma rede *Openflow*.

1.3 Objetivos específicos

- Modificar o controlador para reduzir o *broadcast* de uma rede.
- Reduzir a carga de trabalho do controlador
- Modificar o *POX* para trabalhar com *Openflow* híbrido.

1.4 Trabalhos relacionados

Em (KOTANI; OKABE, 2012) os autores buscam reduzir a quantidade de consultas feitas ao controlador pelos *switches*. Eles realizam modificações no *switch* para que o mesmo seja capaz de tomar alguns decisões. A proposta deles é que o *switch* seja capaz de tomar decisões, sem precisar consultar o controlador. Apesar de bem sucedida, essa proposta apresenta alguns fatores que dificultam escalar a rede. Como é necessário modificar o *switch*, sempre que for adicionado um novo equipamento se faz necessário modifica-lo. Em uma SDN as

decisões devem ser tomadas pelo controlador. Propomos que modificações sejam realizadas no controlador, seguindo o conceito de SDN. Com modificações no controlador é mais simples escalar a rede, já que novos equipamentos devem apenas estar conectado ao controlador, que fará dinamicamente a configuração do equipamento. Em (OLIVEIRA; VERDI; VENTURINI,) os autores realizam uma comparação entre dois métodos de tratamento de mensagens *Address Resolution Protocol* (ARP) *request*. Eles buscam avaliar o impacto do *broadcast*, gerado por mensagens ARP *request*, no tempo de resposta do primeiro pacote ICMP. Eles obtém uma melhoria no tempo quando é realizado tratamento de mensagens ARP *request* em *broadcast*. No trabalho citado é analisado apenas o tempo de resposta do primeiro pacote. Pretendemos realizar testes mais aprofundados a fim de identificar o impacto do *broadcast* na construção da tabela e carga de trabalho do controlador.

2 REDES DEFINIDAS POR SOFTWARE

Ao longo de sua evolução a área de redes presenciou várias tentativas de torna-la mais simples. A maioria das propostas em adicionar programabilidade a rede não obteve sucesso.

2.1 Evolução de Tecnologias Suporte

2.1.1 Redes Ativas

Arquitetura de redes na qual os roteadores são capazes de executar programas transportados juntos com dados. Nas redes ativas *softwares* são transportados pela rede e executados nos roteadores, manipulando o tráfego da rede. Essa arquitetura surgiu ainda na década de 90. Na época de seu surgimento não houve muito interesse do mercado por sua adoção.(JR; MORAES; TEIXEIRA, 2000)

2.1.2 Protocolo ForCES

O protocolo *Forwarding and Control Element Separation* (ForCES) constitui um esforço da *Internet Engineering Task Force* (IETF) para desenvolver um padrão para comunicação entre os planos de controle e dados. A RFC 3654 define os requisitos do ForCES, e a RFC 3746 define o *framework* ForCES. Esse protocolo trabalha no modo mestre/escravo, sendo os elementos de encaminhamento escravos e elementos de controle mestres.(DORIA et al., 2007)

2.1.3 Redes Ethane

Em uma rede da arquitetura *Ethane* os switches são baseados em fluxos e o controle é centralizado. Nessa arquitetura há dois elementos principais, o controlador, responsável pela aplicação de politicas de encaminhamento ao tráfego. No controlador é definido como será gerenciado o tráfego da rede. O segundo é *switch*, sendo é responsável pelo encaminhamento dos pacotes pela rede. (CASADO et al., 2007)

2.2 Redes Definidas Por Software - SDN

Redes definidas por *software* constituem uma nova arquitetura para redes de computadores. As primeiras pesquisas relacionadas à SDN surgiram em 2007 na *Stanford university*, EUA. A arquitetura SDN propõe a divisão dos planos de dados e de controle, e a execução dos mesmos em equipamentos diferentes, para adicionar programabilidade a uma rede. Novas

características podem ser adicionadas pelo administrador, sem a necessidade de intervenção do fabricante do equipamento. Como ilustra na figura 1

> O conceito de programabilidade em redes é baseado na separação entre o plano de controle, responsável pelos protocolos e pelas tomadas de decisão que resultam nas tabelas de encaminhamento, e o plano de dados, que cuida da comutação e do repasse dos dados. Nos equipamentos legados, o plano de controle é executado no próprio equipamento a partir dos protocolos implementados no sistema operacional embarcado, impedindo qualquer tomada de decisão baseada em algoritmos que não tenham sido previstos nestes protocolos previamente instalados. (OLIVEIRA et al., 2012)

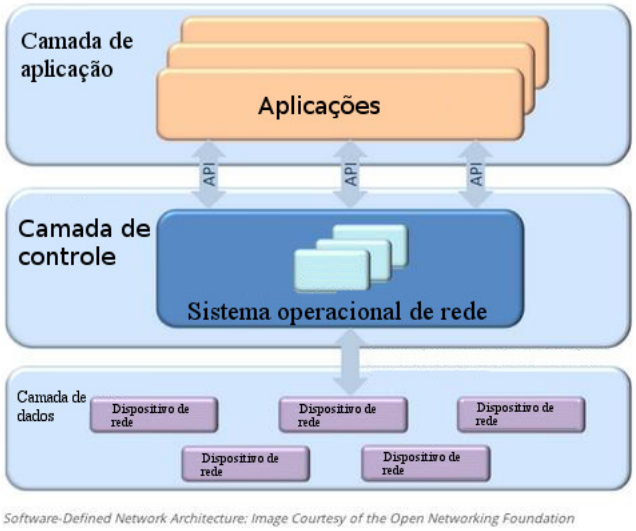

Figura 1: Camadas de uma rede definida por software. (SOFTWARE. . . , 2013)

2.2.1 *Openflow*

Openflow é um padrão aberto da *Open Networking Fundation* (ONF) para comunicação entre os planos de dados e controle em uma SDN, ilustrado na figura 2. O controlador usa a API do *Openflow* para monitorar e gerenciar equipamentos que suportem esse protocolo. Por ser uma API do tipo *southbound*, o *Openflow* não oferece um bom ambiente para desenvolvimento de aplicações de alto nível. Apresenta características como abstração de baixo nível e condição de corrida. Esse padrão apresenta três modos de funcionamento, proativo, reativo e híbrido. No modo proativo os fluxos são definidos manualmente pelo administrador. Esse modo apresenta como vantagem, a redução de requisições ao controlador, reduzindo sua carga trabalho. Uma desvantagem é a obrigação do administrador configurar todos os fluxos que serão utilizados. No modo reativo, os fluxos são criados dinamicamente, em resposta as requisições do *switch*. O modo reativo gera muitas requisições, exigindo processamento no controlador. Sua principal

vantagem é a automatização da configuração da rede. O modo híbrido possui características de funcionamento dos outros modos, sendo usado na adição de alguns fluxos proativamente e demais fluxos reativamente.

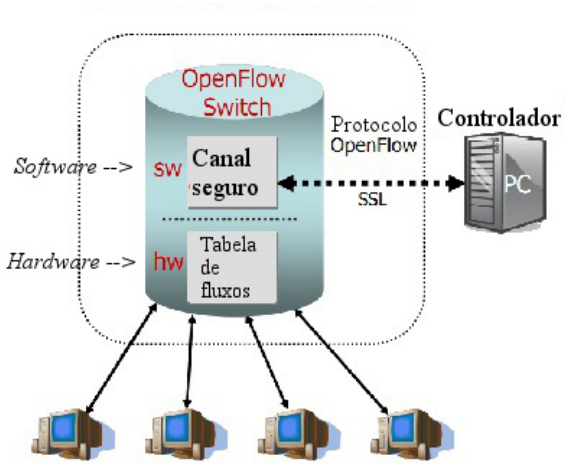

Figura 2: Arquitetura Openflow. (OPENFLOW. . . , 2013)

2.2.1.1 Componentes da arquitetura *Openflow*

- Controlador

  É o principal componente de uma rede *Openflow*. No controlador são executadas as aplicações que gerenciam a rede e, implementam serviços. Implementa o plano de dados em uma rede SDN.

- *Switch*

  É o elemento responsável pelo encaminhamento dos pacotes pela rede. É a infraestrutura de uma rede SDN. No switch é mantida uma tabela de fluxos, que mantêm informações sobre como os pacotes serão processados. Nessa tabela há três campos, são eles:

  – Regra

    O campo regra é usado para definir as características dos pacotes de cada fluxo. características como: *Internet Protocol* (IP) de origem/destino, porta de origem/-destino.

  – Ação

    Define uma ação a ser aplicada nos pacotes do fluxo. Existem três ações básicas, são elas:

    * Encaminhar
      Essa ação define que os pacotes serão roteados/encaminhados através da rede.

    * Encapsular e enviar para o controlador
      O pacote que chega ao switch é encapsulado em uma mensagem Openflow e, é enviado ao controlador. Geralmente essa ação é aplicada ao primeiro pacote de um fluxo. Essa ação pode ser usada em casos em que todos os pacotes do fluxo deverão ser processados pelo controlador.
    * Descartar
      Todos os pacotes do fluxo serão descartados. Pode ser usado na contenção de ataques de negação de serviço.
  - Estatísticas

    Mantêm registrado o número de pacotes e bytes para cada fluxo.

- Canal seguro

  Um canal criptografado, pelo qual são transmitidas mensagens entre comutador e controlador, em uma rede SDN.

2.2.1.2 Mensagens Openflow

O protocolo *Openflow* possui uma série de mensagens. As mais comuns são:

- *Flow-mod*

  Sentido Controlador –> *Switch*. São as mensagens utilizadas na gerencia da tabela de fluxos do switch. São usadas para adicionar, alterar ou remover uma entrada da tabela.

- *packetIn*

  Sentido *Switch* –> Controlador. O switch enviar mensagens *Packet-In*, para o controlador definir o que deverá ser feito com aquele tipo de pacote. São usadas em requisições ao controlador.

- *Flow-out*

  Sentido Controlador –> *Switch*. O controlador é capaz de enviar pacotes para a rede através do switch. O controlador encapsula o pacote a ser enviado em uma mensagem flow-out. Na mensagem *flow-out* é especificada a porta do switch pela o pacote será enviado para a rede.

- *Flow-stats*

  São mensagens usadas na coleta de estatisticas do *switch*. Podem ser *request*, do controlador para o switch. Ou *response*, do *switch* para o controlador.

2.2.2 Virtualização de redes

O conceito de virtualização de redes define uma infraestrutura de redes de computadores virtuais. São definidos por *software*, executando sobre máquinas físicas, de forma que toda infraestrutura virtual seja isolada da infraestrutura física, sem interferir na mesma. As redes virtuais usam recursos das máquinas físicas. Existem diversos hypervisores, programas responsáveis pelo gerenciamento das máquinas virtuais. Um dos hypervisores mais usados na criação de redes virtuais em nível de *software* é o *Xen* (FERNANDES et al., 2011). Esse programa é usado na criação de máquinas virtuais em *desktops* e servidores, e oferece a opção de criar de roteadores virtuais que podem ser utilizados na interligação de máquinas virtuais para formação de uma rede. Em SDN a construção de redes virtuais acontece em nível de hardware, através da separação do tráfego da rede física em *slices*, porções de fluxo do tráfego total. O *flowvisor*(SHERWOOD et al., 2009) possibilita virtualização em SDN.

O uso de virtualização de redes possibilita execução de experimentos distintos, sobre a mesma infraestrutura, em paralelo sem interferência entre experimentos. Possibilita a construção de redes com diferentes configurações, sob a mesma infraestrutura física. Em cada rede virtual pode-se realizar um experimento diferente.

Virtualização de redes pode ser usada para isolamento de serviços. Uma organização pode oferecer diversos serviços, com cada serviços executando em uma rede virtual diferente.

2.2.3 Emulador Mininet

Mininet é um emulador de redes definidas por *software*. Sua configuração é relativamente simples, considerando que é fornecido um *script* para configuração automática. Até o presente momento esse *script* funciona apenas no *Debian Linux* e no *Ubuntu*. Esse emulador pode ser obtido em forma de *appliance* para o *virtualbox*, com algumas ferramentas configuradas.

Por ser um sistema de emulação, o *Mininet* permite execução de aplicações reais, possibilitando execução de qualquer controlador *Openflow*.

Quando o *Mininet* é executado, ele cria um processo chamado "*ovs-openflowd*"para cada *switch* emulado. Por padrão esse processo é finalizado junto com o emulador. Em alguns casos esse processo permanece ativo, mesmo com a finalização do emulador. É importante garantir que o processo seja finalizado sempre junto com o emulador, pois pode causar interferência em um experimento. Por exemplo: o processo permanece em execução com 4 *switches*, então o *Mininet* é iniciado com mais 5 *switches*. Quando o controlador for executado irá se conectar a 9 *switches*. O experimento que deveria ser realizado com 5, será realizado com 9 *switches*.

O *Mininet* possui algumas topologias pré-definidas, são elas: *single* figura 3, *linear* figura 4, *tree* figura 5, *minimal* figura 6, *reversed* figura 7. Além dessas topologias pré-definidas, o *Mininet* possui suporte a topologias customizadas.

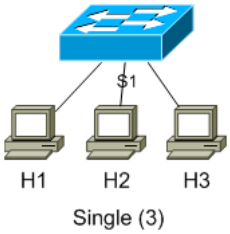

Figura 3: Exemplo de topologia single. (SINGLE, 2013)

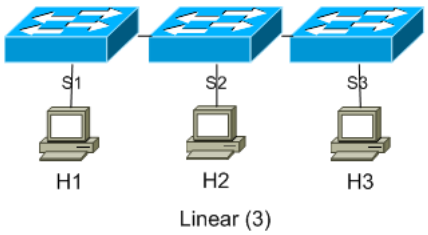

Figura 4: Exemplo de topologia linear. (LINEAR, 2013)

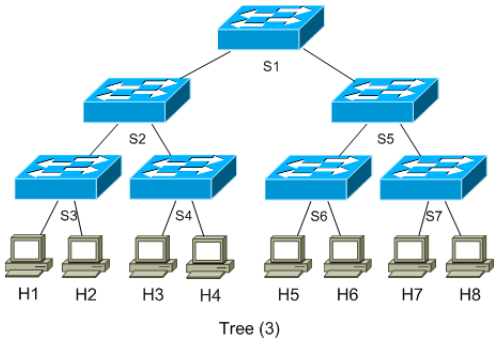

Figura 5: Exemplo de topologia tree. (TREE, 2013)

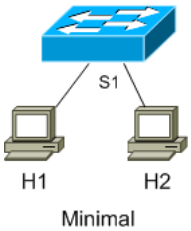

Figura 6: Exemplo de topologia minimal. (MINIMAL, 2013)

Criar uma rede no *Mininet* é relativamente simples, pode-se usar a linha de comando ou um componente chamado *miniedit.py*, figura 8, uma interface gráfica para o *Mininet*. O *miniedit.py* possui algumas limitações em relação a linha de comando. Neste trabalho foi utiliza

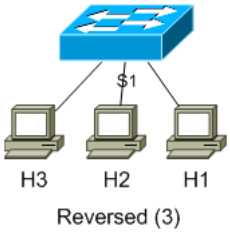

Figura 7: Exemplo de topologia reversed. (REVERSED, 2013)

a ferramenta de linha de comandos. Ao chamar o *Mininet* são passados os parâmetros sobre as características da rede. Há um grande conjunto de opções para configurar a rede. Há opções sobre topologia, número de hosts, switch, taxa de perda de pacotes, largura de banda, tipo de controlador, etc. O *switch* padrão é o *Openvswitch*(PETTIT et al., 2010). O *Openvswitch* é um *switch* virtual desenvolvido especialmente para trabalhar com o protocolo *Openflow*. Neste

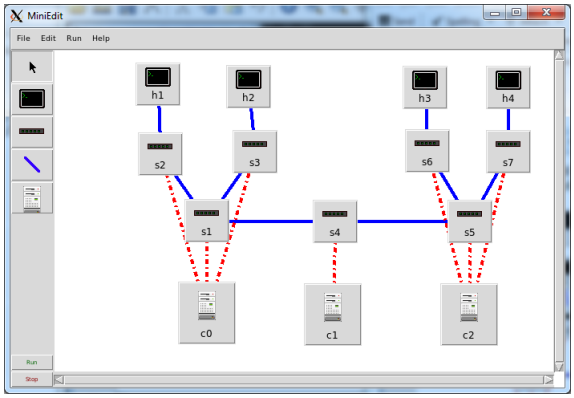

Figura 8: Ferramenta *miniedit*. (MINIEDIT, 2013)

trabalho foram usadas as topologias *minimal*, *single* e *linear*. Características das topologias usadas:

- *Minimal*

  Essa topologia é a mais simples. É formada por duas máquinas ligadas a um *switch*. É a topologia padrão.

- Single

  Uma extensão da *minimal*. Nessa topologia há apenas um *switch* ligado a um número variado de máquinas, a quantidade de máquinas é passada como parâmetro.

- *Linear*

  A topologia *linear* é formada por múltiplos *switches*, em cada *switch* há uma máquina conectada.

A topologia *minimal* é interessante na realização de pequenos testes, entre duas máquinas. A *single* é útil na emulação de uma rede com múltiplos clientes ligados a um *switch*. Para construir uma rede com múltiplos switches a *linear* é ideal. Com uma topologia linear o controlador terá que gerenciar vários switches, oferecendo uma visão global da rede para as aplicações em execução.

2.2.4 POX

O *POX* é escrito inteiramente em *Python*. Ao executar o *POX* apenas os módulos básicos serão carregados, módulos adicionais podem ser passados como parâmetros para o *pox.py*. Múltiplas aplicações podem ser executadas ao mesmo tempo. O *Openflow* não possui bons mecanismos para lidar com múltiplas aplicações, podendo causa interferência entre elas. Apesar de possível e necessário, executar aplicações simultâneas em uma rede *Openflow* pode gerar problemas no funcionamento da rede.

O *POX* é divido em vários módulos. Estes módulos implementam as características do controlador, conhecer seu funcionamento ajuda entender como o controlador funciona. Junto com os módulos básicos de funcionamento há algumas aplicações padrões. Dentre essas aplicações, há para roteamento, *spanning tree*, *NAT*, etc.

Ao criar um novo módulo é preciso coloca-lo no *PATH* do POX. A configuração do *PATH* é feita no componente *core*.py, sendo possível adicionar outros diretórios ao *PATH*. Ao baixar o controlador é criada um diretório chamado *POX*, dentro dele há uma série de subdiretórios. O subdiretório *ext* é usado especialmente para armazenar novas aplicações.

O controlador recebe os pacotes do *switch*, extrai informações do cabeçalho e, repassa essas informações para as aplicações. No *POX* informações são passadas para as aplicações como argumento do parâmetro *event*. *Launch* é a principal função de uma aplicação, e primeira a ser executada. *Core*.py é o principal componente do controlador e, é usado como base para qualquer aplicação. O componente *libopenflow_01* implementa o *Openflow* no controlador. Métodos para trabalhar com pacotes *Internet Protocol version* 4(IPv4), *ARP*, *Multi Protocol Label Switching* (MPLS), *Transmission Control Protocol* (TCP), etc. são encontrados na pasta *pox/lib/packet*. Métodos de endereçamento são encontrados no modulo *addresses*.py.

A variável *event.parsed* guarda as informações do pacote recebido, geralmente essa variável é atribuída a outra variável chamada *packet*. O valor de *event.parsed* pode ser atribuído a uma variável com outro nome, neste texto será considerado o uso do nome *packet*.

Packet.type armazena o tipo do pacote, é útil no tratamento de pacote de tipos específicos. *packet.next.protosrc* e *packet.next.srcip* guardam o endereço IP de origem da camada de enlace e rede respectivamente, já as variáveis *packet.next.dstip* e *packet.next.protodst* guardam o endereço de destino. Algumas variáveis são encontrados em apenas determinados tipos de

pacote, o *packet.type* pode ser utilizado para identificar o tipo do pacote trabalhado, evitando o erro de variável inexistente em determinado pacote.

3 SOLUÇÃO PROPOSTA

3.1 Redução do broadcast

O broadcast causa processamento extra em uma rede. Em uma rede definida por software o broadcast causa mais danos, um deles é gerar processamento no controlador. Quando um pacote é enviado em broadcast, o controlador precisa criar fluxos no switch para encaminhar esses pacotes. O ideal é minimizar a criação de fluxos, para reduzir a carga de processamento no controlador. Figura 9 e Figura 10 demonstram o funcionamento de uma conexão ARP em uma rede definida por software. Cada requisição ARP gera o mesmo processo mostrado acima.

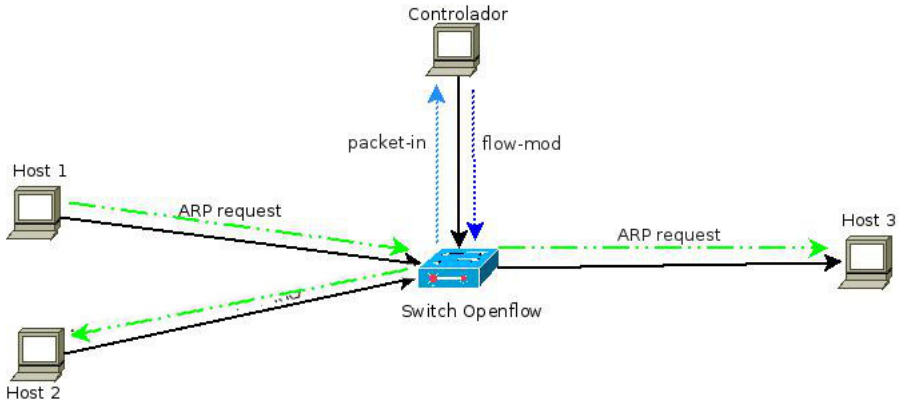

Figura 9: Requisições arp geram novos fluxos

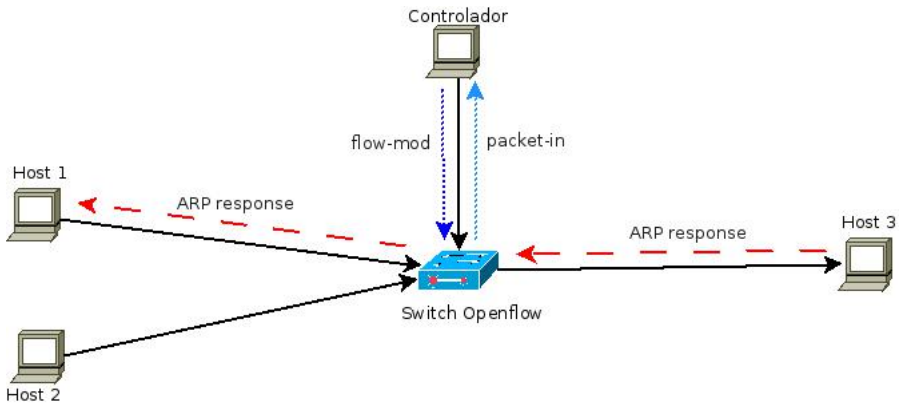

Figura 10: Respostas arp geram mais fluxos

A proposta é modificar o controlador para manter uma tabela ARP com o mapeamento IP –> MAC. A tabela é populada com informações extraídas dos pacotes da camada de rede do modelo *Open Systems Interconnectation* (OSI) que passam pelo controlador. Pacotes da camada de rede contêm endereços IP/MAC de origem/destino, usados para formar a tabela. A tabela é criada utilizando a estrutura de dados conhecida como dicionário, da linguagem *Python*. O

funcionamento do módulo proposto é bem simples. Ao chegar um pacote ARP *request*, o módulo verifica se o endereço IP de destino se encontra em sua tabela. Caso o endereço IP se encontre na tabela, significa que o MAC corresponde ao IP também está na tabela, então uma função responsável por gerar uma mensagem ARP *response* é chamada.

O controlador possui métodos para criação de pacotes ARP. O pacote ARP *response* criado possui o MAC de destino extraído da tabela. O índice do dicionário é o endereço IP e, o valor é o endereço MAC.

Quando a requisição é para um endereço inexistente na tabela, ocorre *broadcast* na rede. Uma entrada do tipo *flood* é criada na tabela do switch, para permitir que o ARP *request* seja enviado para todos na rede, com exceção do host de origem.

Sem a proposta, uma requisição *ARP* sempre gera no mínimo duas mensagens *flow-mod*. Uma mensagem adiciona um fluxo para a *ARP request*, e outra adiciona um fluxo para a ARP response. Com a solução proposta serão geradas mensagens *flow-mod* apenas quando o MAC de destino não estiver na tabela ARP mantida pelo controlador. Quando a tabela ARP do controlador contiver o MAC requisito, não será necessário adicionar entradas na tabela do switch. O controlador receberá a requisição ARP e, responderá usando uma mensagem *flow-out*, sem precisar manipular a tabela do switch. Reduzindo o número de mensagens *flow-mod*. A Figura 11 mostra o funcionamento da proposta. Código disponível no apêndice B.

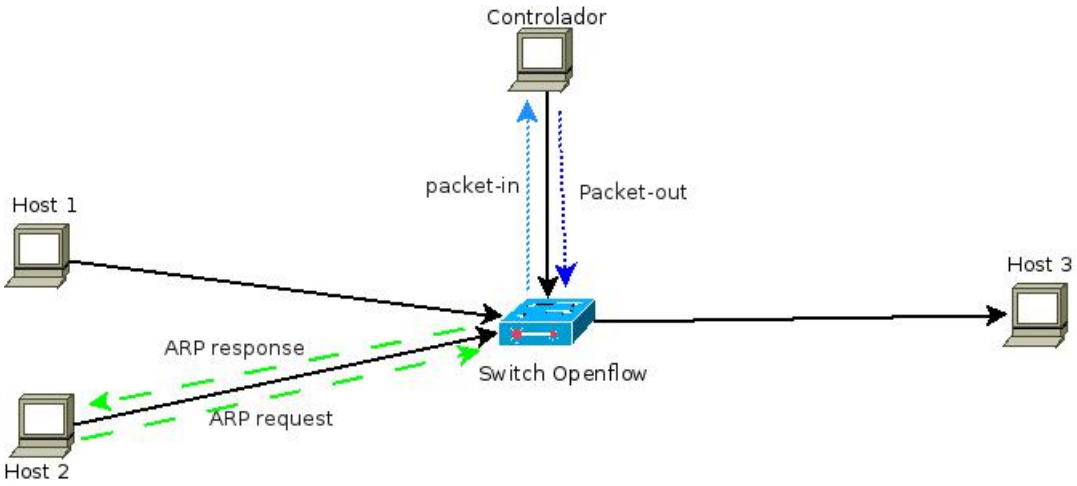

Figura 11: Controlador responde requisições ARP

4 AVALIAÇÃO

4.1 Cenário e ambientes de teste

Para validação da solução proposta foi utilizado o emulador de redes, *Mininet* (MORLING; CAIN, 1975). Foi utilizada uma rede com 5 switches, com um *hosts* conectado a cada *switch*. Foi utilizado o controlador *POX*. O POX é um sistema simples e normalmente usado na academia. Podendo ser obtido em seu repositório oficial(POXREPO, 2013) no *github*.

Neste trabalho foi usada a versão 2.0.0 em uma máquina virtual com 2GB de memória RAM, processador *Intel CORE* i5 e suporte a virtualização por *hardware*.

Depois de estudar a estrutura interna do controlador, pode-se iniciar a realização de modificações. Inicialmente apenas mudanças pequenas foram realizadas, mudanças como alterar o conteúdo de mensagens exibidas na tela.

Para facilitar a compreensão do funcionamento de uma controlador *Openflow*, é interessante conhecer profundamente seu funcionamento, (CONSORTIUM et al., 2009) é de grande ajuda nesta tarefa.

Um *software* bastante útil é o *Wireshark*(OREBAUGH; RAMIREZ; BEALE, 2006), com ele pode-se visualizar as mensagens do protocolo *Openflow* em tempo de execução.

4.2 Coletar os resultados

Switches Openflow possuem um campo em sua tabela para armazenar estatísticas sobre fluxos. O controlador é capaz de enviar requisições para o *switch*, solicitando estatísticas dos fluxos. Durante os experimentos foi utilizado um módulo *POX* responsável pela captura de estatísticas. Algumas modificações no módulo de estatísticas foram necessárias, para adequá-lo aos experimentos. Há ferramentas como o *dpctl*, que permitem visualizar a tabela do *switch*. Com o *dpctl* daria para capturar os fluxos apenas no final dos experimentos.Tabela é atualizada constantemente, o que motivou o uso do *flow-stats*, por possibilitar intervalos entre execuções, permitindo executar o *flow-stats* seguidas vezes durante o experimento. Quanto as mensagens *flow-mod*, o software usado foi o *Wireshark* (OREBAUGH; RAMIREZ; BEALE, 2006). Em sua versão 1.8.2 o *Wireshark* não possui filtro para pacotes *Openflow*, sendo necessário compilar manualmente esse filtro. O *TCPdump* (FUENTES; KAR, 2005) foi utilizado para capturar os pacotes, e o *Wireshark* para analisa-los.

4.3 Desafios da pesquisa

Durante a execução desta pesquisa, a falta de *hardware* com *Openflow* habilitado representou um grande desafio. Para contornar a falta de *switch Openflow*, foram levantadas algumas alternativas. A melhor opção seria realizar os experimentos no *Future Internet Testbed with Security* (FITS)(NUNES; PONTES; GUEDES,). No FITS seria possível realizar os testes com máquinas geograficamente distantes, como ilustra a figura 12. Na época em que os experimentos foram realizados, o FITS estava indisponível para realização dos testes. Outro obstáculo encontrado foi a versão do *Openflow* suportada pelo *POX*. O *POX* 0.1.0 possui suporte até a versão 1.0.0 do *Openflow*, e essa versão é muito limitada em relação a versão 1.3. O *Openflow* 1.3 possui mecanismos que tornam a manipulação dos pacotes mais simples.

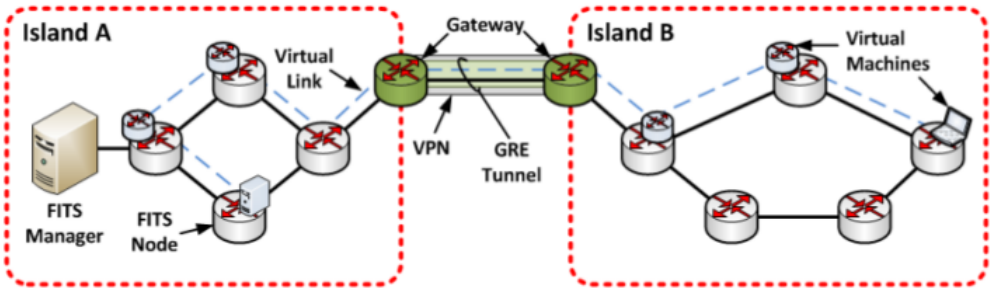

Figura 12: Testbed FITS. (TESTBED. . . , 2013)

5 RESULTADOS

As primeiras amostras de um experimento geralmente configuram a fase de estabilização, enquanto as últimas são a fase de desestabilização. Foram descartados as 5 primeiras e últimas medições de cada experimento, foi usado esse valor após a observação de que os primeiros resultados eram bastante dispersos. Foi calculada a média aritmética de cada experimento, a fim de encontrar a tendencia central de cada experimento, para compara-los. O desvio padrão foi calculado para estipular o quanto as medidas se distanciaram da média. A margem de erro e intervalo de confiança foram calculados para identificar quais valores poderiam seriam ser considerados corretos. Um nível de confiança de 95%. Os resultados obtidos são mostrados nos gráficos abaixo:

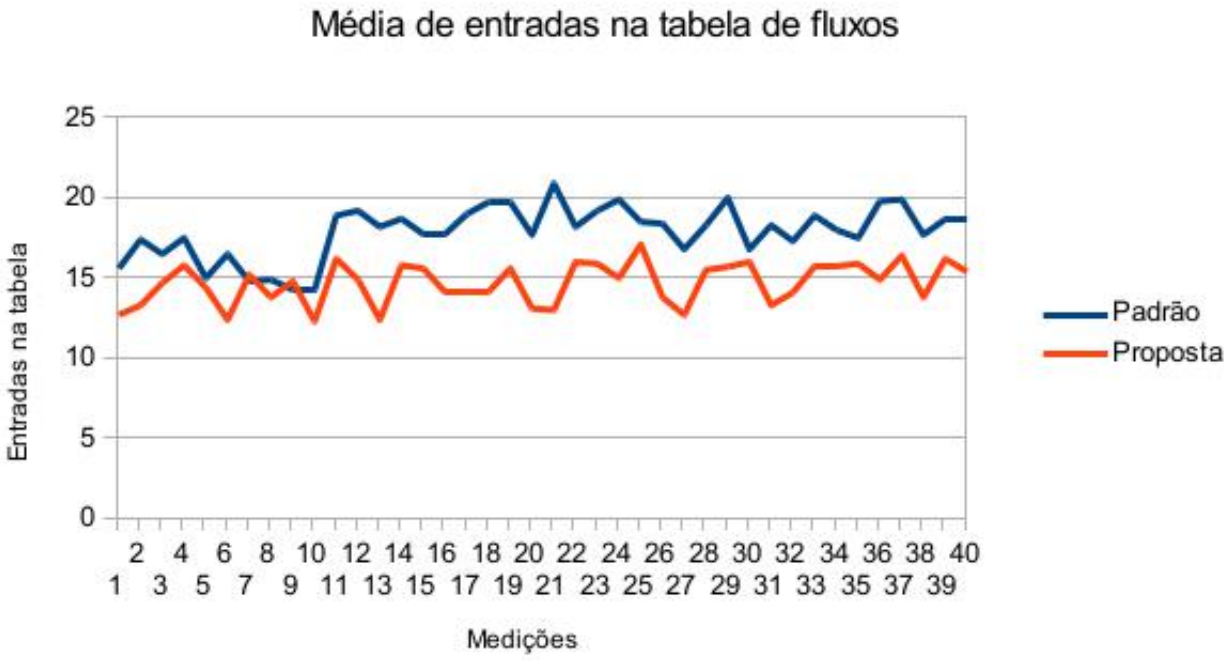

Figura 13: Média de entradas na tabela

Na execução dos experimentos foi usada uma opção do emulador especifica para realização de testes. Com o parâmetro *–test* o *Mininet* cria uma rede, executa o experimento, e para a rede. O parâmetro *–test* pode receber como valor *iperf*, *pingall*, etc. O valor do parâmetro *–test* define qual teste será realizado na rede. Testes usando *pingall* foram usados na construção das tabelas dos *switches* e para gerar mensagens *flow-mod*. Mensagens do tipo *flow-mod* são utilizadas para gerenciar a tabela do switch. Com uma mensagem *flow-mod* o controlador pode adicionar, alterar ou remover uma entrada na tabela. Com o *pingall* as máquinas pingarão umas às outras.

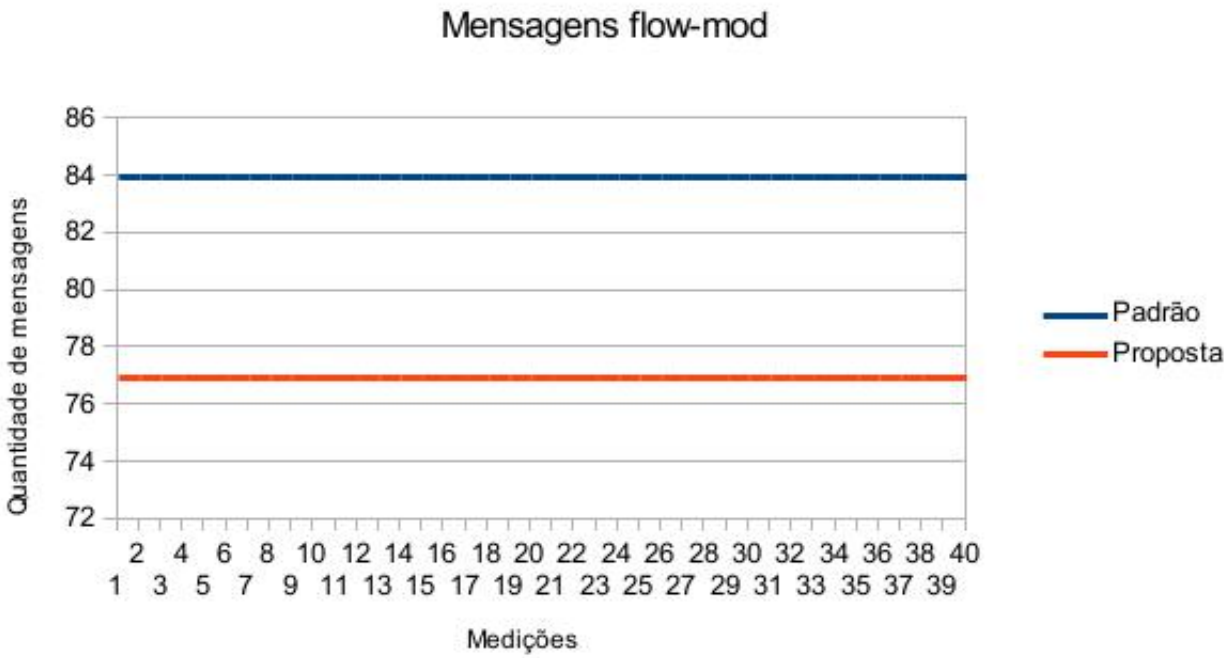

Figura 14: Média de mensagens *flow-mod*

As metas são a redução do número de entradas na tabela do switch e o número de mensagens *flow-mod* geradas. Com a redução de entradas na tabela do switch, um número maior de entradas poderão ser criadas, sem haver necessidade de adicionar mais memoria. Reduzindo o número de mensagens *flow-mod*, há uma redução da carga de trabalho do controlador. Reduzindo a carga de trabalho do controlador, abre-se espaço para adição de mais máquinas a rede.

	Padrão (quantidade)	Proposta (quantidade)
Média	17,985	14,797
Desvio-padrão	1,657	1,298
Margem-de-erro	0,513	0,402
Intervalo-de-confiança	17,472 e 18,498	14,395 e 15,199

Tabela 1: Entradas na tabela do switch

Na construção da tabela do *switch* os dois módulos apresentam variação nos valores obtidos. O módulo padrão gera uma quantidade superior. Mesmo considerando o valor mais alto do módulo proposto com o mais baixo do módulo padrão, o proposto obtêm valores menores conforme visto na Tabela 1.

Não há variação quanto a quantidade de mensagens *flow-mod*, em ambos os módulos. Não há desvio padrão, margem de erro e intervalo de confiança, a média pode observada na tabela 2.

	Padrão (quantidade)	Proposta (quantidade)
Média	84	77
Desvio-padrão	0	0
Margem-de-erro	0	0
Intervalo-de-confiança	0	0

Tabela 2: Quantidade de Mensagens *flow-mod*

6 CONCLUSÃO E SUGESTÕES DE TRABALHOS FUTUROS

Em SDN o controlador pode se tornar o gargalo da rede. Com o barateamento e aumento do poder computacional dos switches, o hardware normalmente não representa um grande problema. Nas redes tradicionais o broadcast é um problema, e se estende a SDN. Com a redução do broadcast gerado por mensagens ARP *request*, foram alcançadas redução nas mensagens geradas pelo controlador e redução da quantidade entradas na tabela do switch. Com a utilização do módulo proposto houve uma redução média de 8,33% de mensagens *flow-mod*, e 17,97% das entradas na tabela de fluxos. A redução do *broadcast* ARP em rede SDN auxilia na redução da carga de mensagens geradas no plano de controle.

O modelo de redes definidas por software vem ganhando cada vez mais espaço, tanto na academia quanto no mercado. Muitos são os desafios de pesquisa. Como sugestão para trabalhos futuros sugerimos:

- Adicionar um *timeout* a cada entrada da tabela *ARP*.

  Adicionar um tempo de permanência de cada entrada na tabela. Após o esgotamento desse tempo, a entrada deve ser removida da tabela. Atualmente, esse tempo é implementado na tabela *ARP* mantida nos hosts. Um *timeout* ajudaria a reduzir a probabilidade de uma entrada permanecer muito tempo na tabela, mesmo não sendo necessária.

- Adicionar inteligência a solução.

  Procurar explorar melhor a visão global da rede, oferecida pelo controlador. Conhecer a rede como um todo, provavelmente ajudará na manutenção da tabela. Implementar características que tornem a solução inteligente.

BIBLIOGRAFIA

AGRAWAL, Banit; SHERWOOD, Timothy. Modeling tcam power for next generation network devices. In: IEEE. *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*. [S.l.], 2006. p. 120–129.

CASADO, Martin et al. Ethane: Taking control of the enterprise. In: ACM. *ACM SIGCOMM Computer Communication Review*. [S.l.], 2007. v. 37, n. 4, p. 1–12.

CONSORTIUM, OpenFlow Switch et al. *OpenFlow Switch Specification Version 1.0. 0*. [S.l.]: December, 2009.

DORIA, Avri et al. Forces protocol specification. *EB/OL]. Dec*, 2007.

FERNANDES, NataliaC. et al. Virtual networks: isolation, performance, and trends. *annals of telecommunications - annales des télécommunications*, Springer-Verlag, v. 66, n. 5-6, p. 339–355, 2011. ISSN 0003-4347. Disponível em: <http://dx.doi.org/10.1007/s12243-010-0208-9>.

FUENTES, Felix; KAR, Dulal C. Ethereal vs. tcpdump: a comparative study on packet sniffing tools for educational purpose. *Journal of Computing Sciences in Colleges*, Consortium for Computing Sciences in Colleges, v. 20, n. 4, p. 169–176, 2005.

GUDE, Natasha et al. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, ACM, v. 38, n. 3, p. 105–110, 2008.

JR, José Helvécio Teixeira; MORAES, Luís Felipe M de; TEIXEIRA, Suzana Ramos. Uma proposta para o emprego de tecnologias de redes ativas no gerenciamento de redes. In: WTMN'00: 5TH WORKSHOP TELECOMMUNICATIONS MANAGEMENT NETWORK. [S.l.], 2000.

KOTANI, Daisuke; OKABE, Yasuo. Packet-in message control for reducing cpu load and control traffic in openflow switches. In: IEEE. *Software Defined Networking (EWSDN), 2012 European Workshop on*. [S.l.], 2012. p. 42–47.

LINEAR. 2013. Disponível em: <http://www.routereflector.com/en/2013/11/mininet-as-an-sdn-test-platform/>.

LYNCH, Bob. 2013. Disponível em: <http://www.sdncentral.com/technology/openflow-sdn/2013/06/>.

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

MINIEDIT. 2013. Disponível em: <http://gregorygee.wordpress.com/category/miniedit/>.

MINIMAL. 2013. 13/12/2013. Disponível em: <http://www.routereflector.com/en/2013/11/mininet-as-an-sdn-test-platform/>.

MOREIRA, Marcelo DD et al. Internet do futuro: Um novo horizonte. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC*, v. 2009, p. 1–59, 2009.

MORLING, RCS; CAIN, GD. Mininet: a packetswitching minicomputer network for real-time instrumentation. *Proc. AIM Int. Meet. on Minicomputers and Data Communication, Liège, Association des Ingénieurs Electriciens sortis de l'Institut Electrotechnique Montefiore, Liège (January 1975)*, 1975.

NUNES, Rogério V; PONTES, Raphael L; GUEDES, Dorgival. Uma arquitetura baseada em redes definidas por software para isolamento de redes em datacenters virtualizados.

OLIVEIRA, Renan CS de; VERDI, Fábio L; VENTURINI, Yeda R. Tratando mensagens de arp request em redes sdn's.

OLIVEIRA, Rafael Emerick Z de et al. *Parâmetros balizadores para experimentos com comutadores Openflow: avaliaçao experimental baseada em mediç oes de alta precisao*. [S.l.]: SBrT, 2012.

OPENFLOW arquitetura. 2013. Disponível em: <http://keepingitclassless.net/2013/10/introduction-to-open-vswitch/>.

OREBAUGH, Angela; RAMIREZ, Gilbert; BEALE, Jay. *Wireshark & Ethereal network protocol analyzer toolkit*. [S.l.]: Syngress, 2006.

PETTIT, Justin et al. Virtual switching in an era of advanced edges. In: *2nd Workshop on Data Center–Converged and Virtual Ethernet Switching (DC-CAVES), ITC*. [S.l.: s.n.], 2010. v. 22.

POXREPO. 2013. Disponível em: <https://github.com/noxrepo/pox>.

REVERSED. 2013. Disponível em: <http://www.routereflector.com/en/2013/11/mininet-as-an-sdn-test-platform/>.

SHENKER, Scott et al. The future of networking, and the past of protocols. *Open Networking Summit*, 2011.

SHERWOOD, Rob et al. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 2009.

SINGLE. 2013. 13/12/2013. Disponível em: <http://www.routereflector.com/en/2013/11/mininet-as-an-sdn-test-platform/>.

SOFTWARE defined networking arquiteture. 2013. Disponível em: <http://www.opennetsummit.org/why-sdn.html>.

TESTBED FITS. 2013. Disponível em: <http://www.gta.ufrj.br/novo_gta2/>.

TREE. 2013. 13/12/2013. Disponível em: <http://www.routereflector.com/en/2013/11/mininet-as-an-sdn-test-platform/>.

VAUGHAN-NICHOLS, Steven J. Openflow: The next generation of the network? *Computer*, Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, 17 th Fl New York NY 10016-5997 United States, v. 44, n. 8, p. 13–15, 2011.

APÊNDICE

APÊNDICE A – CÓDIGO DE OBTENÇÃO DAS ESTATÍSTICAS DOS FLUXOS OPENFLOW

```
#!/usr/bin/python
# Copyright 2012 William Yu
# wyu@ateneo.edu
#
# This file is part of POX.
#
# POX is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# POX is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with POX. If not, see <http://www.gnu.org/licenses/>.
#

"""
This is a demonstration file created to show how to obtain flow
and port statistics from OpenFlow 1.0-enabled switches. The flow
statistics handler contains a summary of web-only traffic.
"""

# standard includes
from pox.core import core
from pox.lib.util import dpidToStr
import pox.openflow.libopenflow_01 as of

# include as part of the betta branch
from pox.openflow.of_json import *
```

```
log = core.getLogger()

# handler for timer function that sends the requests to all the
# switches connected to the controller.
def _timer_func ():
  for connection in core.openflow._connections.values():
    connection.send(of.ofp_stats_request(body=of.ofp_flow_stats_request()))
    connection.send(of.ofp_stats_request(body=of.ofp_port_stats_request()))
  log.debug("Sent %i flow/port stats request(s)", len(core.openflow._connections))

# handler to display flow statistics received in JSON format
# structure of event.stats is defined by ofp_flow_stats()
def _handle_flowstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("FlowStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)

  # Get number of bytes/packets in flows for web traffic only
  tbytes = 0
  tflows = 0
  tpacket = 0
  for f in event.stats:
      tbytes += f.byte_count
      tpacket += f.packet_count
      tflows += 1
  print("Bytes %s -- Pacotes %s -- Fluxos %s" %(tbytes, tpacket, tflows))


# handler to display port statistics received in JSON format
def _handle_portstats_received (event):
  stats = flow_stats_to_list(event.stats)
  log.debug("PortStatsReceived from %s: %s",
    dpidToStr(event.connection.dpid), stats)

# main functiont to launch the module
def launch ():
  from pox.lib.recoco import Timer
```

```
# attach handsers to listners
core.openflow.addListenerByName("FlowStatsReceived",
  _handle_flowstats_received)
core.openflow.addListenerByName("PortStatsReceived",
  _handle_portstats_received)

# timer set to execute every one seconds
Timer(1, _timer_func, recurring=True)
```

APÊNDICE B – IMPLEMENTAÇÃO DA PROPOSTA DE REDUÇÃO DAS MENSAGENS ARP

```
from pox.core import core
import pox.openflow.libopenflow\_01 as of
from pox.lib.util import dpid\_to\_str
from pox.lib.util import str\_to\_bool
import time
from pox.lib.packet.ipv4 import ipv4
log = core.getLogger()
from pox.lib.packet.arp import arp
from pox.lib.packet import IPAddr
from pox.lib.packet import EthAddr
from pox.lib.packet import ethernet
import threading

_flood_delay = 0
#tabela
arpTable = {}
class LearningSwitch (object):
  def __init__ (self, connection, transparent):

    self.connection = connection
    self.transparent = transparent


    self.macToPort = {}

    connection.addListeners(self)


    self.hold_down_expired = _flood_delay == 0



  def _handle_PacketIn (self, event):
```

```
packet = event.parsed

# Responde requisições ARP
def RespondToARP(self, packet, match, event):

 r = arp()
 r.opcode = arp.REPLY
 r.hwdst = match.dl_src
 r.protosrc = packet.next.protodst
 r.protodst = match.nw_src
 r.hwsrc = arpTable[str(packet.next.protodst)]
 e = ethernet(type=packet.ARP_TYPE, src=r.hwsrc, dst=r.hwdst)
 e.set_payload(r)
 msg = of.ofp_packet_out()
 msg.data = e.pack()
 msg.actions.append(of.ofp_action_output(port = of.OFPP_IN_PORT))
 msg.in_port = event.port
 event.connection.send(msg)

def flood (message = None):

  msg = of.ofp_packet_out()
  if time.time() - self.connection.connect_time >= _flood_delay:

    if self.hold_down_expired is False:

      self.hold_down_expired = True


    if message is not None: log.debug(message)

    msg.actions.append(of.ofp_action_output(port = of.OFPP_FLOOD))
  else:
    pass

  msg.data = event.ofp
  msg.in_port = event.port
  self.connection.send(msg)
```

```
    def drop (duration = None):

      if duration is not None:
        if not isinstance(duration, tuple):
          duration = (duration,duration)
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet)
        msg.idle_timeout = duration[0]
        msg.hard_timeout = duration[1]
        msg.buffer_id = event.ofp.buffer_id
        self.connection.send(msg)
      elif event.ofp.buffer_id is not None:
        msg = of.ofp_packet_out()
        msg.buffer_id = event.ofp.buffer_id
        msg.in_port = event.port
        self.connection.send(msg)

    self.macToPort[packet.src] = event.port # 1
    global arpTable
    match = of.ofp_match.from_packet(packet)
    if ( match.dl_type == packet.ARP_TYPE and match.nw_proto == arp.REQUEST and str
RespondToARP(self,packet, match, event)
return
    if not self.transparent:
      if packet.type == packet.LLDP_TYPE or packet.dst.isBridgeFiltered():
        drop()
        return

    if packet.dst.is_multicast:
      flood()
    else:
      if packet.dst not in self.macToPort:
        flood("Port for %s unknown -- flooding" % (packet.dst,))
      else:
        port = self.macToPort[packet.dst]
        if port == event.port:


          drop(10)
```

```
          return
        # se for um pacote IP adiciona informações na tabela.
if packet.type == 2048:
arpTable[str(packet.next.dstip)]=packet.dst
arpTable[str(packet.next.srcip)]=packet.src

        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match.from_packet(packet, event.port)
        msg.idle_timeout = 110
        msg.hard_timeout = 130
        msg.actions.append(of.ofp_action_output(port = port))
        msg.data = event.ofp
self.connection.send(msg)


class l2_learning (object):
  def __init__ (self, transparent):
    core.openflow.addListeners(self)
    self.transparent = transparent

  def _handle_ConnectionUp (self, event):
    log.debug("Connection %s" % (event.connection,))
    LearningSwitch(event.connection, self.transparent)


def launch (transparent=False, hold_down=_flood_delay):

  try:
    global _flood_delay
    _flood_delay = int(str(hold_down), 10)
    assert _flood_delay >= 0
  except:
    raise RuntimeError("Expected hold-down to be a number")

  core.registerNew(l2_learning, str_to_bool(transparent))
```