UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
TECNÓLOGO EM REDES DE COMPUTADORES

FRANCISCO WENDEL DE LIMA MACIEL

UMA SOLUÇÃO DE ENGENHARIA DE TRÁFEGO PARA REDES COM MÚLTIPLAS CONEXÕES WAN, BASEADA EM SDN.

QUIXADÁ-CE
2016

FRANCISCO WENDEL DE LIMA MACIEL

UMA SOLUÇÃO DE ENGENHARIA DE TRÁFEGO PARA REDES COM MÚLTIPLAS CONEXÕES WAN, BASEADA EM SDN.

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Tecnólogo em Redes de Computadores da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Tecnólogo.

Área de concentração: Computação.

Orientador: Prof. Me. Michel Sales Bonfim

QUIXADÁ-CE
2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo (a) autor (a)

M138s Maciel, Francisco Wendel de Lima.
Uma solução de Engenharia de tráfego para redes com múltiplas conexões WAN, baseada em SDN. / Francisco Wendel de Lima Maciel. – 2016.
47 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2016.
Orientação: Prof. Me. Michel Sales Bonfim.

1. Software-defined networking. 2. Redes definidas por software. 3. OpenFlow. 4. Engenharia de Tráfego. I. Título.

CDD 004.6

FRANCISCO WENDEL DE LIMA MACIEL

UMA SOLUÇÃO DE ENGENHARIA DE TRÁFEGO PARA REDES COM MÚLTIPLAS CONEXÕES WAN, BASEADA EM SDN.

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Tecnólogo em Redes de Computadores da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Tecnólogo.

Área de concentração: Computação.

Aprovado em: _____ / Julho / 2016.

BANCA EXAMINADORA

_____
Prof. Me. Michel Sales Bonfim (Orientador)
Universidade Federal do Ceará-UFC

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

_____
Prof. Me. Marcos Dantas Ortiz
Universidade Federal do Ceará-UFC

Aos meus pais, familiares, amigos e ao meu orientador, por toda ajuda prestada no desenvolvimento desse projeto.

AGRADECIMENTOS

Agradeço aos meus pais por todo apoio no decorrer desse processo e todos os ensinamentos que me fizeram chegar até aqui.

Ao meu orientador Michel Sales Bonfim, por todo apoio prestado no desenvolvimento desse projeto.

A todos os professores que de alguma forma contribuíram para a minha formação, tanto pessoal como acadêmica.

E por fim, a todos os meus amigos.

"Information is power. But like all power, there are those who want to keep it for themselves.”
(Aaron Swartz)

RESUMO

A Engenharia de Tráfego tem como finalidade otimizar o desempenho das redes. Para alcançar um melhor desempenho, ela utiliza técnicas que buscam a diminuição dos congestionamentos na rede, melhorando sua performance. As técnicas de engenharia de tráfego têm sido amplamente utilizadas nas redes atuais e legadas. Entretanto, novos requisitos de rede têm surgido, exigindo o surgimento de novos paradigmas de rede e a criação de novas tecnologias de engenharia de tráfego. Nesse contexto, surge um novo paradigma de rede conhecido como *Software Defined Network (SDN)* ou Redes Definidas por *Software*. SDN é um paradigma de rede que tem como finalidade a separação entre o plano de controle e o plano de dados. Consiste em tornar a rede programável, repassando a decisão para equipamentos externos da forma de encaminhamento dos fluxos dos pacotes. Este trabalho propõe uma solução SDN de engenharia de tráfego para redes com múltiplas conexões externas, permitindo a orientação do tráfego através da definição de políticas de rede em alto nível. Um estudo de caso foi realizado para validar a nossa solução.

Palavras chave: Redes Definidas por Software, *OpenFlow* e Engenharia de Tráfego.

ABSTRACT

The Traffic Engineering aims to optimize network performance. To achieve better performance, it uses techniques that seek the reduction of congestion on the network, improving their performance. Traffic engineering techniques have been widely used in current and legacy networks. However, new network requirements have emerged, requiring the emergence of new network paradigms and creating new traffic engineering technologies. In this context, a new network paradigm known as Software Defined Network (SDN). SDN is a networking paradigm that has the purpose of separating the control plane from the data plane. It makes the network programmable, transferring the routing decision to external devices. This paper proposes an SDN solution of traffic engineering for networks with multiple external connections, allowing the direction of traffic through the network and policy-making at a high level. A case study was performed to validate our solution.

Keywords: *Software Defined Network (SDN), OpenFlow and Traffic Engineering.*

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Introdução

A Engenharia de Tráfego (ET) tem como objetivo otimizar o desempenho das redes. De acordo com Duarte (2009), para alcançar essa otimização, a engenharia de tráfego avalia, modela, caracteriza o tráfego, monitora e controla. Sempre visando evitar congestionamentos que venham a diminuir a performance na rede. As técnicas de ET têm sido amplamente utilizadas nas redes atuais e legadas, incluídas em soluções como ATM e MPLS. Contudo, novos requisitos de rede têm surgido, exigindo o surgimento de novos paradigmas de rede e a criação de novas tecnologias de ET.

Segundo Akyildiz et al., (2014), as soluções que existiam de ET (ATM e MPLS) são desfavoráveis para serem aplicadas em novos modelos de redes por duas razões principais. Primeiro, as atuais aplicações de rede exigem uma arquitetura que responda as solicitações em tempo real e sejam escaláveis, permitindo que sejam feitas diversas requisições. Nessa arquitetura deve existir a capacidade de classificar diversos tipos de tráfegos de aplicações distintas em um curto período de tempo. Segundo, com o aumento da utilização da computação em nuvem, faz-se necessário, além de uma rede escalável, um gerenciamento apropriado da rede, que seja capaz de melhorar a utilização dos recursos, melhorando assim o desempenho do sistema. Para resolver esses problemas, surgiram nos últimos anos algumas propostas para criação de redes do futuro, dentre as quais destaca-se o novo paradigma de redes conhecido como as Redes Definidas por *Software* (*Software Defined Network - SDN*).

SDN é um novo paradigma de rede que tem como objetivo separar o plano de controle do plano de dados. Segundo Costa (2013), o plano de controle é responsável pelos protocolos e pelas tomadas de decisão, gerando assim uma tabela de encaminhamento de pacotes. O plano de dados é responsável pela comutação e repasse dos pacotes, baseado nesta tabela.

De acordo com Costa (2013), o objetivo das SDNs consiste em tornar a rede programável, decidindo em equipamentos externos a forma de encaminhamento dos fluxos de pacotes. Nesses equipamentos localiza-se o “sistema operacional de rede” ou controlador SDN. Este decidirá que tratamento o pacote deve ter dentro do equipamento de rede, qual a melhor porta de saída, dentre outras ações.

No paradigma das SDN, a plataforma mais conhecida e utilizada é o *OpenFlow*. Segundo Mckeown et al., (2008), devido às dificuldades de se realizar testes em uma rede que

está em produção, foi desenvolvido o *OpenFlow*. Uma de suas vantagens é que ele consegue manter os testes em uma rede real sem atrapalhar o tráfego de produção. Segundo Costa (2013), o *OpenFlow* é um padrão aberto, composto por uma interface de programação que dá a possibilidade ao administrador da rede de controlar diretamente os dispositivos de encaminhamento de pacotes que estão nos elementos da rede.

O *OpenFlow* é bastante flexível em relação a programação de cada pacote da rede, como ele deve ser tratado e por onde ele deve ser encaminhado. Ele consegue implementar como deve ser definido um fluxo, quais as ações que são permitidas tomar em relação a ele e determina qual o protocolo de comunicação que será utilizado entre o controlador e os dispositivos de encaminhamento.

A proposta do nosso trabalho consiste em criar uma solução capaz de encaminhar pacotes para redes com múltiplas interfaces externas, baseadas em políticas bem definidas. A solução proposta é baseada no conceito de SDN, utilizando o *OpenFlow* para criar uma interface programável no roteador de borda de uma rede com múltiplas interfaces WAN. Este projeto consiste em um trabalho tecnológico.

Para tanto, uma aplicação em *Python* foi desenvolvida sob o controlador *OpenDayLight* [1], que ficará responsável por determinar como os pacotes serão encaminhados, podendo realizar ações de balanceamento de carga, de acordo com políticas bem definidas. Tais políticas serão definidas em arquivos XML e lidas e executadas pela aplicação. Um protótipo foi implantado e validado com estudos de caso.

É esperado que nossa solução seja adotada por usuários e pesquisadores da área, a fim de auxiliá-los nas suas pesquisas que estejam relacionadas à engenharia de tráfego em redes definidas por *software*.

Este trabalho está organizado da seguinte forma: Na seção 2 estão os objetivos, gerais e específicos. A seção 3 descreve os trabalhos relacionados. Na seção 4 está a fundamentação teórica, onde estão os conceitos chaves do projeto. A seção 5 é referente a explicação da proposta do projeto. Na seção 6 são apresentados os processos de validação da solução, e por fim, na seção 7 estão as considerações finais.

[1] https://www.opendaylight.org/

2 OBJETIVOS

2.1 Objetivo Geral

Desenvolver uma solução SDN de engenharia de tráfego para redes com múltiplas conexões WAN, para realizar ações de balanceamento de carga, de acordo com políticas bem definidas.

2.2 Objetivos Específicos

- Identificar técnicas de engenharia de tráfego, utilizando tecnologia SDN.
- Definir um modelo de engenharia de tráfego para redes de múltiplas conexões com WAN, utilizando SDN.
- Validar a solução utilizando estudo de caso.

3 TRABALHOS RELACIONADOS

O trabalho de Agarwal, Kodialam e Lakshman (2013), traz uma proposta de uso da tecnologia SDN em engenharia de tráfego quando SDN está sendo introduzida em uma rede já existente. A ideia é demonstrar que o controle centralizado da tecnologia pode melhorar a utilização da rede, bem como reduzir a perda de pacotes e atrasos.

O problema que o trabalho dos pesquisadores busca atacar está relacionado, como mencionado anteriormente, à implantação do paradigma SDN em redes já existentes, pois em uma rede que já existe, não fica todo tráfego controlado por um único controlador SDN, e sim vários controladores distribuídos na rede, podendo até utilizar o roteamento já existente.

O trabalho mencionado anteriormente, assemelha-se com este projeto de pesquisa no que diz respeito a ideia de um controlador central na rede que busca definir para onde devem ser encaminhados os pacotes de acordo com um modelo de engenharia de tráfego baseado em políticas bem definidas de encaminhamento, buscando diminuir atrasos e perda de pacotes na rede, aumentando assim o desempenho da rede.

A proposta do nosso projeto difere do artigo selecionado como trabalho relacionado em relação ao local onde a solução foi implantada. Os pesquisadores desenvolveram sua solução baseado em redes já existentes na qual a inclusão da tecnologia SDN seria feita de maneira gradativa. Na proposta do nosso projeto, a rede já será estabelecida no modelo SDN, com o *OpenFlow* no roteador de borda e um controlador central decidindo, baseado em políticas, por onde os pacotes devem trafegar.

No trabalho realizado por Caria, Jukan e Hoffmann (2013), é proposta uma análise da migração para SDN a partir da perspectiva de engenharia de tráfego. De acordo com uma topologia existente, surge a questão de quais roteadores em meio a rede devem migrar para SDN. É proposto um algoritmo para o melhor caminho com a intenção de escolher quais roteadores serão substituídos, mostrando que quando os nós SDN estão estrategicamente localizados, fornecem diversas alternativas para o trajeto do pacote, reduzindo a necessidade de modificações futuras na rede.

Tanto o nosso projeto como o artigo publicado pelos autores, trabalham a ideia de engenharia de tráfego em redes definidas por *software*, buscando aumento de desempenho da rede. Após os autores realizarem testes utilizando a simulação e aumento de tráfego de acordo com a topologia escolhida, foi apresentada uma melhoria de 16% em desempenho ainda em fases iniciais de migração utilizando o algoritmo proposto pelos pesquisadores, diferente dos 9% quando a escolha dos roteadores a serem modificados é feita de maneira aleatória.

Apesar da semelhança em relação a ideia de engenharia de tráfego, ambos os trabalhos diferem no que diz respeito ao modelo de rede implantado, uma vez que os autores trabalham com um algoritmo que escolhe, em meio aos outros roteadores da rede, quais devem ser migrados para SDN de maneira gradual analisando possíveis melhorias em relação ao modelo de escolha aleatória. No nosso trabalho, a ideia central se baseia em um modelo de rede com múltiplas interfaces de saída na qual já está implantada a tecnologia SDN em que o encaminhamento de pacotes ficará a cargo do controlador central que determinará por onde o pacote vai trafegar de acordo com políticas.

No trabalho de Zhang et al., (2015), os autores apresentam uma abordagem conhecida como roteamento híbrido com o objetivo de alcançar balanceamento de carga para múltiplas matrizes de tráfego com baixa complexidade e boa escalabilidade. A ideia é complementar as abordagens para alcançar o balanceamento de carga para múltiplas matrizes de tráfego, conhecidas como roteamento baseado em destino e encaminhamento explícito para tirar proveito de ambas abordagens. Roteamento baseado em destino é o modelo comum de encaminhamento, onde roteadores tomam decisões de encaminhamento com base nos endereços de destino especificados nos cabeçalhos do pacote. Assim, cada roteador fica responsável de encaminhar os pacotes destinados para o mesmo destino da mesma maneira independentemente dos endereços de fonte. Roteamento explícito oferece uma granularidade no controle de distribuição do tráfego, ele suporta roteamento flexível para cada fluxo individual. Foi mostrado que o desempenho do roteamento explícito supera o roteamento

baseado em destino, quando a realização da otimização do roteamento para múltiplas matrizes de tráfego.

Roteamento híbrido reduz o número de encaminhamento de entradas em comparação com o roteamento explícito puro. Uma heurística foi desenvolvida para obter a configuração de roteamento híbrido perto do ideal. De acordo com os resultados obtidos no trabalho de Zhang, Junjie.,et al.(2015), o roteamento híbrido atinge carga perto do ideal. Em particular, o roteamento híbrido economiza pelo menos 84,6% dos recursos. Nosso trabalho assemelha-se ao proposto por Zhang, Junjie et al. (2015), no que diz respeito a busca por balanceamento de carga em tecnologia SDN para alcançar um desempenho melhor da rede, porém não focando na ideia das abordagens para múltiplas matrizes de tráfego, roteamento baseado em destino e encaminhamento explícito.

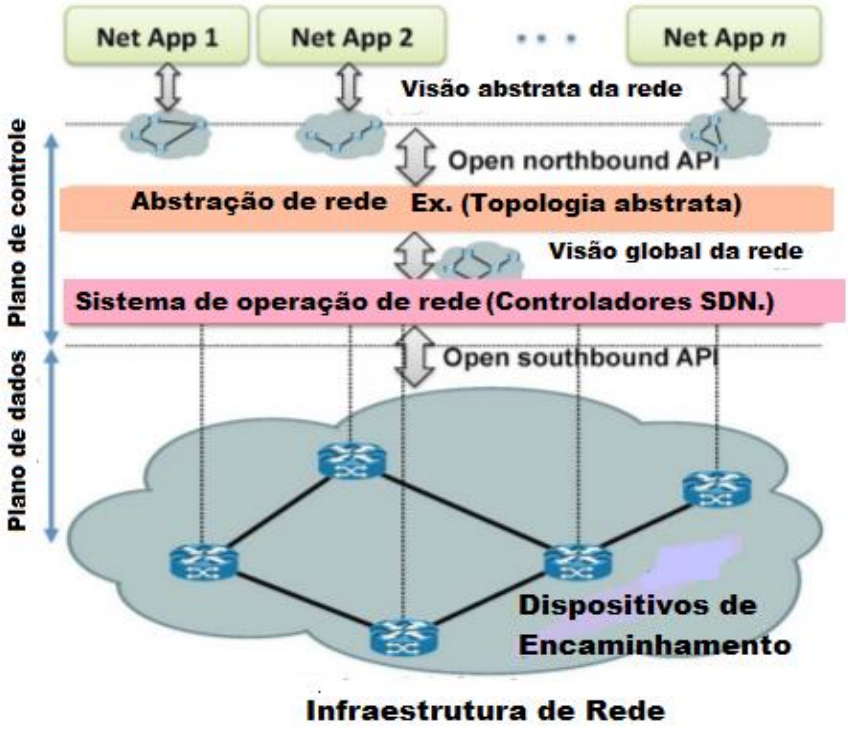
4 FUNDAMENTAÇÃO TEÓRICA

Essa seção apresenta os conceitos chaves abordados nesse projeto. Nas subseções 4.1, 4.2 e 4.3 e 4.4 são apresentados os conceitos relacionados a SDN – *Software Defined Network, OpenFlow* e Engenharia de tráfego (ET) e ET em SDN, respectivamente.

4.1 SDN – *SOFTWARE DEFINED NETWORK*

Redes Definidas por *Software* consistem em um novo paradigma de rede que tem como objetivo a flexibilização da rede através da separação entre o plano de controle e o plano de dados. De acordo com Costa (2013), o plano de controle é responsável pelos protocolos e pelas tomadas de decisão confeccionando assim uma tabela de roteamento, já o plano de dados é responsável pela comutação e pelo repasse dos pacotes. A figura 1 ilustra o modelo da arquitetura da tecnologia SDN focando na apresentação de seus componentes principais: Plano de dados, controlador e plano de controle.

Figura 1 – Arquitetura SDN.



Fonte: Modificado pelo autor.

Plano de dados: Responsável por encaminhar o tráfego de acordo com a lógica implementada no plano de controle.

Controlador: Nesses dispositivos fica concentrada toda a comunicação entre os elementos programáveis da rede obtendo uma visão geral da rede.

Plano de controle: É onde fica planejada a lógica de controle do comportamento do encaminhamento dos pacotes.

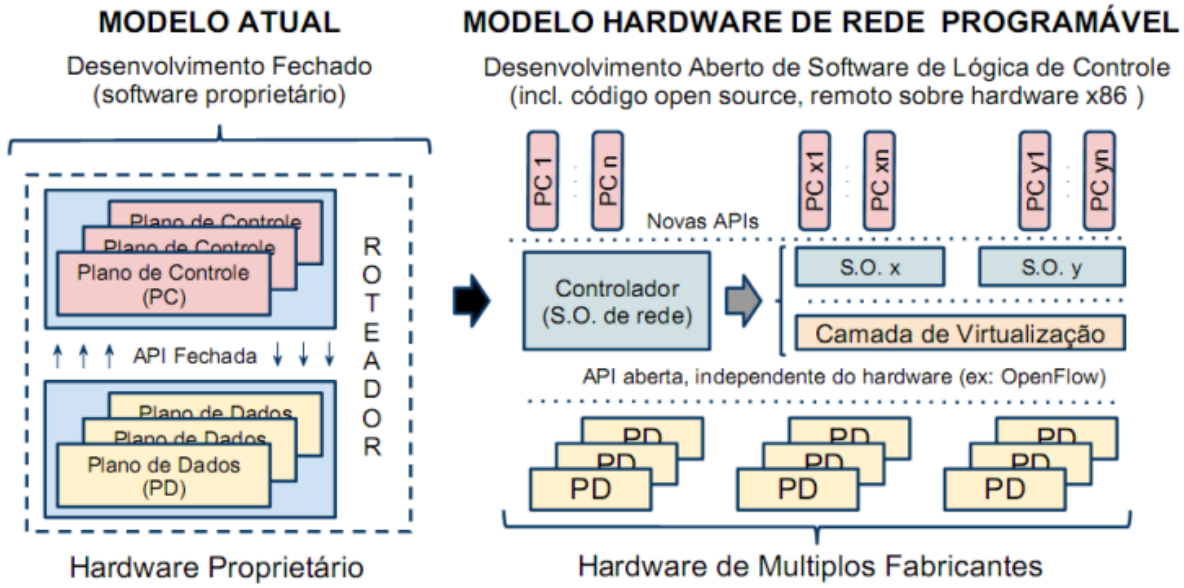
Nas redes tradicionais, os equipamentos além de serem responsáveis pelo encaminhamento de pacotes também possuem um *software* de controle que fica no próprio equipamento, realizando as tomadas de decisão e gerando assim a tabela de roteamento. Por ter esse *software* implementado nos próprios equipamentos proprietários, existe a impossibilidade de executar qualquer tomada de decisão que não tenha sido estabelecida nos mesmos.

Segundo Costa (2013), essa nova arquitetura de rede de computadores baseia-se em redes que sejam capazes de serem programadas sob demanda, ou seja, redes que consigam ser programadas e flexíveis para poderem lidar com necessidades e problemas atuais ou que venham a surgir no futuro. A figura 2, apresenta um comparativo entre as redes do modelo

atual e redes programáveis. Mostrando suas diferenças principalmente em relação a programação da rede. No modelo atual, temos uma arquitetura de programação fechada, com *software* proprietário. Em redes programáveis, o desenvolvimento é aberto, podendo ser alterado de acordo com a necessidade do programador.

Na figura 2, é exibida uma comparação entre as arquiteturas de roteadores. Demonstrando que no modelo atual, o desenvolvimento é baseado em *APIs* proprietárias o que vem a dificultar qualquer tipo de alteração na lógica das redes, diferente do que acontece com o modelo programável, no qual se baseia o paradigma de SDN. No modelo programável é possível observar que o desenvolvimento de aplicações é realizado através de *APIs* abertas, como exemplo o *OpenFlow*, onde o administrador consegue alterar de acordo com suas necessidades a lógica implementada na rede.

Figura 2 – Arquitetura de roteadores: Modelo Atual x Modelo Programável.



Fonte: ROTHENBERG, Christian Esteve et al., (2010).

De acordo com Kreutz et al., (2015), as redes definidas por *software* possuem quatro fatores que ajudam aos pesquisadores a adotar essa solução, são esses:

- Existe a clara separação entre o plano de dados e o plano de controle. A funcionalidade do plano de controle é retirada dos equipamentos, que se tornarão apenas elementos de encaminhamento.
- As tomadas de decisão de encaminhamento são baseadas em fluxos. Um fluxo é definido por um conjunto de valores de campo de pacotes atuando como filtro e um conjunto de ações. Na tecnologia SDN um fluxo é uma sequência de pacotes entre uma fonte e um destino. A

abstração dos fluxos permite unificar o comportamento dos dispositivos de rede, incluindo roteadores, *switches, firewalls e middleboxes*.

- A lógica do plano de controle é movida para entidades externas, que são conhecidas como controladores SDN ou NOS - Sistema Operacional de Rede. O controlador SDN é uma plataforma de software que é executada em um servidor externo.
- A lógica da rede é programável através de uma aplicação de software que interage com os dispositivos do plano de dados.

A infraestrutura SDN é composta por um conjunto de dispositivos (*switch, roteadores, middleboxes*) assim como no modelo de redes de computadores tradicionais, porém esses dispositivos agora ficam apenas como simples encaminhadores de pacotes, sem um software implantado no dispositivo para controle das tomadas de decisão. Em SDN o plano de controle é movido para uma entidade externa, conhecida como controlador da rede ou NOS (*Network Operational System*) - Sistema operacional de rede.

De acordo com Costa (2013), as redes SDN permitem o desenvolvimento de aplicações de rede que realizam lógicas de monitoramento de tráfego mais sofisticadas e mais completo do que as implementadas hoje em dia. É importante a apresentação dos elementos principais da tecnologia SDN, abaixo estão listados os elementos junto a seus conceitos de acordo com Kreutz, et al., (2015).

1. ***Interface Southbound***

Interfaces *Southbound* ou (*Southbound API*) definem os protocolos de comunicação entre os dispositivos de encaminhamento e os elementos do plano de controle. Este protocolo define como o os elementos do plano de controle e do plano de dados interagem.

2. ***Interfaces Northbound***

Uma interface *Northbound* resume os conjuntos de instruções de baixo nível usadas pela interface *Southbound* para programar os dispositivos de encaminhamento.

3. **Plano de Gestão**

Conjunto de aplicações que favorecem o desenvolvimento das funções oferecidas pela interface *Northbound* para implementar o controle e operações lógicas da rede. Isso inclui aplicações como encaminhamento, *firewall*, balanceadores de carga e monitoramento. Normalmente um software de gerenciamento define as políticas para o *Southbound* específico, para programar o comportamento do encaminhamento dos dispositivos.

4. **Dispositivos de encaminhamento**

Os dispositivos de encaminhamento tem um conjunto de instruções bem definidas, por exemplo, regras de fluxo usadas para tirar ações nos pacotes de entrada; estas instruções são definidas pela interface *Southbound* e são instalados nos dispositivos de encaminhamento pelos controladores SDN implementando os protocolos *Southbound*.

No paradigma SDN, podemos destacar alguns benefícios da sua utilização, segundo Kreutz, et al., (2015):

- A programação das aplicações tornam-se mais fácil pelo fato da abstração fornecida pela plataforma de controle permitir que ela seja compartilhada.
- Todas as aplicações podem tirar proveito das mesmas informações de rede, existe uma visão geral da rede, enquanto reutiliza o módulo de *software* do plano de controle.
- Estas aplicações podem reconfigurar os dispositivos de encaminhamento a partir de qualquer localização na rede. Nesse sentido, não existe a necessidade de se elaborar uma estratégia precisa sobre o lugar onde estão implantadas as novas funcionalidades.
- A integração de diferentes aplicações torna-se mais simples. Aplicativos de balanceamento de carga e de roteamento podem ser combinados, com as decisões de balanceamento de carga tendo prioridades sobre políticas de roteamento.

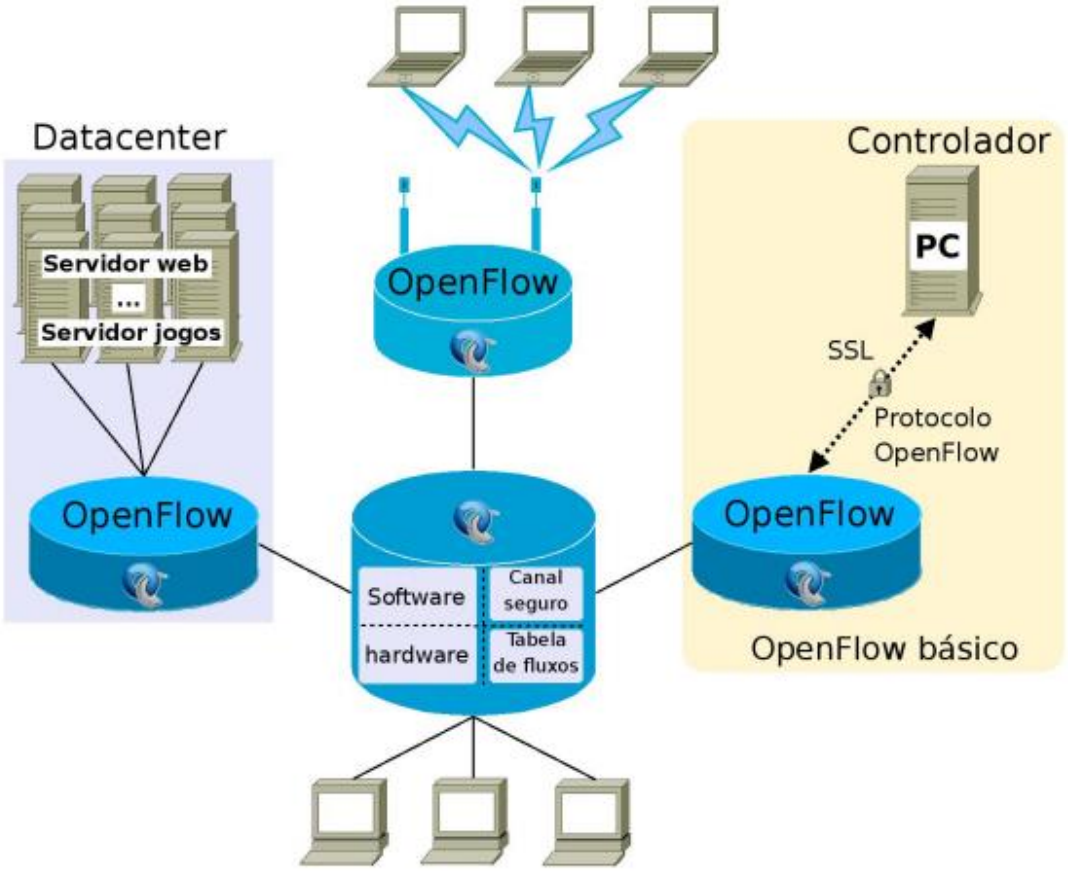
O nosso projeto de pesquisa é uma solução de engenharia de tráfego para redes com múltiplas conexões WAN, baseada em SDN, ou seja, todo o trabalho está concentrado nesse novo paradigma de rede. Como mencionado anteriormente, o componente de rede SDN conhecido como controlador SDN é parte fundamental do nosso projeto, pois nele ficarão as tomadas de decisão de encaminhamento, para qual melhor saída os pacotes devem ser encaminhados, baseadas em políticas bem definidas.

4.2 OPENFLOW

O *OpenFlow* foi proposto pela Universidade de Stanford em 2007. De acordo com Rothenberg, et al., (2010), essa tecnologia veio com o objetivo de atender as demandas de aprovação de novas arquiteturas e protocolos de redes sobre equipamentos comerciais, podendo assim, possibilitar a inovação da rede, realizando a execução das redes de teste, sem comprometer a rede de produção. *OpenFlow* também consiste em uma forma de os fabricantes poderem inserir essa tecnologia nos seus equipamentos sem a necessidade de expor o projeto desses comutadores.

Segundo Costa (2013), o *OpenFlow* é um padrão aberto, composto por uma interface de programação que dá a possibilidade ao administrador da rede de controlar diretamente os dispositivos de encaminhamento de pacotes que estão nos elementos da rede. É muito importante que esses equipamentos tenham custo baixo e consigam manter o desempenho dos equipamentos que estão sendo utilizados na rede de produção para que seja possível a troca desses equipamentos pelo que estão sendo utilizados na rede. A figura 3 ilustra uma rede com o padrão *OpenFlow*, onde como pode ser observado, o padrão *OpenFlow* trabalha como uma interface de comunicação (protocolo *OpenFlow*), entre o controlador e os outros elementos programáveis da rede.

Figura 3 – Exemplo de uma rede com *OpenFlow*.



Fonte: ROTHENBERG, Christian Esteve et al., (2010).

De acordo com Costa (2013), o padrão *OpenFlow* também tem como objetivo atender aos seguintes requisitos:

- Ser adotado em implementações de baixo custo e alto desempenho.
- Possibilitar o suporte a uma vasta gama de pesquisas científicas.
- Permitir o isolamento do tráfego de produção do tráfego experimental.
- Não ter a necessidade de expor o projeto dos fabricantes em suas plataformas.

Para a implantação de uma rede *OpenFlow*, se faz necessário que em sua arquitetura estejam associados quatro componentes, são eles: Tabela de fluxos, protocolo *OpenFlow*, controlador e canal seguro. Os componentes são apresentados a seguir, segundo Rothenberg, et al (2010):

- **Tabela de fluxo**

As entradas da tabela de fluxos do *hardware* de rede baseiam-se em regras, ações e contadores. As regras são formadas com base no valor de um ou mais campos do cabeçalho do pacote, já as ações são associadas a ela para definir como os pacotes devem ser processados e para onde devem ser encaminhados. Os contadores têm o objetivo de manter estatísticas de utilização e para remover fluxos inativos.

- **Canal Seguro**

Para evitar que a rede venha a sofrer ataques de elementos mal-intencionados, o canal seguro garante a confiabilidade na troca de informações entre o *switch* e o controlador.

- **Protocolo *OpenFlow***

O Protocolo aberto utilizado para comunicação que usa uma interface externa é definido pelo *OpenFlow* para a troca de mensagens entre os equipamentos de rede e os controladores. Atua como uma interface, que possibilita configuração e o gerenciamento dos diferentes equipamentos de rede.

- **Controlador**

É o *Software* responsável pela tomada de decisões, além de ser possível adicionar e remover as entradas na tabela de roteamento. O controlador atua como um sistema operacional para monitoramento e controle das redes.

De acordo com Rothenberg et al., (2010), na tecnologia *OpenFlow* existem quatro fatores que contribuem para sua aceitação, são esses:

- Através de uma atualização no *firmware*, o *OpenFlow* pode ser incorporado em equipamentos de rede como *switches* e roteadores que estejam em operação, sem a necessidade de se modificar o *hardware*.
- O protocolo *OpenFlow* separa o plano de controle do plano de dados. Essa separação possibilita que sejam utilizados controladores remotos. O *software* de controle é responsável por determinar como os fluxos de pacotes são trafegados na rede. Isso permite que o controle sobre o plano de dados seja repassado aos programadores. Assim, o controle da rede não fica embarcado nos equipamentos, que eram limitados por implementações e padrões de fabricantes.

- Uma rede *OpenFlow* permite a definição de setores de rede, que garante o isolamento entre os diferentes controladores que estão executando sobre a rede, permitindo que o tráfego operacional e o tráfego experimental sejam executados de modo simultâneo.
- O *OpenFlow* é compatível com a Internet atual, onde em mais de um setor da rede o tráfego pode estar em execução.

O projeto também vai utilizar o padrão *OpenFlow* através de um *switch* virtual como o *Open vSwitch*[2]. De acordo com PFAFF et al., (2009), uma das maneiras de se implementar um comutador *OpenFlow* em computadores pessoais é com o *Open vSwitch*. O funcionamento do *Switch OpenFlow* se encontra em um módulo do Linux que programa o encaminhamento de pacotes diretamente no *Kernel* do sistema, já seu plano de controle é atingido a partir do espaço de usuário.

O controlador escolhido no projeto é o *OpenDayLight*. A escolha pelo *OpenDayLight* em nosso projeto se deu pelo fato de ser um controlador *Open Source*, programado em Java e que conta com uma interface web, que no início do desenvolvimento do projeto foi essencial para entender a dinâmica na criação dos fluxos. A tabela 1 ilustra as principais diferenças entre os controladores SDN *Open Sources* no que diz respeito a linguagem e a versão do *OpenFlow* utilizada.

Tabela 1 – Controladores SDN.

Controlador	**Linguagem**	***OpenFlow***	**Criado por:**
Nox	C++	*Openflow 1.3*	Niciria Networks
Maestro	JAVA	*Openflow 1.3*	Stanford University
Beacon	JAVA	*Openflow 1.0*	Rice University
Floodlight	JAVA	*Openflow 1.3*	Big Switch Networks
Trema	RUBY e C	*Openflow 1.3*	NEC
Node-Flow	JAVA SCRIPT	*Openflow 1.3*	DreamersLab
OpenDayLight	JAVA	*Openflow 1.3*	*Cisco e OpenDayLight*

Fonte: KHATTAK, Zuhran Khan; AWAIS, Muhammad; IQBAL, Adnan. (2014).

[2] ***openvswitch***.org/

4.2.1 OPENDAYLIGHT

Nosso projeto irá utilizar como controlador o *OpenDayLight*[3]. O *OpenDayLight* é um *software* baseado em Java que pode ser executado em qualquer Sistema Operacional. Ele é uma implementação para ser utilizada em SDN. É uma ferramenta de código aberto que é financiada por diversas empresas de tecnologia, tais como: *Cisco, IBM e VMware*. As lógicas de negócio dos algoritmos residem nas aplicações. Esses aplicativos utilizam o controlador para reunir informações sobre a inteligência das redes, executando seu algoritmo para fazer análises e em seguida, determinar as novas regras de encaminhamento de pacotes em toda a rede. A arquitetura de *software* do controlador *OpenDayLight* define a criação de aplicativos de interação e serviços de aplicações, como encaminhamentos de mensagens e configuração dos dados. O controlador possui uma interface web para facilitar a visualização dos hosts que estão conectados ao *Switch*. Para tornar possível o acesso à interface web é necessária a instalação de alguns componentes para que o acesso seja liberado. Esses componentes são conhecidos como *features* que são utilizados para habilitar diversos módulos. Na tabela 2 estão algumas *features* que devem ser instaladas após a instalação do *OpenDayLight*:

Tabela 2 – Lista de *Features OpenDayLight* utilizadas.

NOME	**DESCRIÇÃO**
feature:install odl-l2switch-switch-ui	O recurso *L2Switch* permite internamente a correção dos dados MD-SAL e o serviço plug-in *openFlow*. *L2Switch* também garante que, em uma topologia, os hosts fiquem visíveis junto com *switches*.
feature:install odl-dlux-all	Dlux fornece uma interface gráfica de usuário moderna e intuitiva para o *OpenDayLight* com base no AngularJS.
feature:install odl-restconf-all	REST como protocolo funciona sobre HTTP para acessar dados definidos na API YANG. É utilizada para que seja realizada

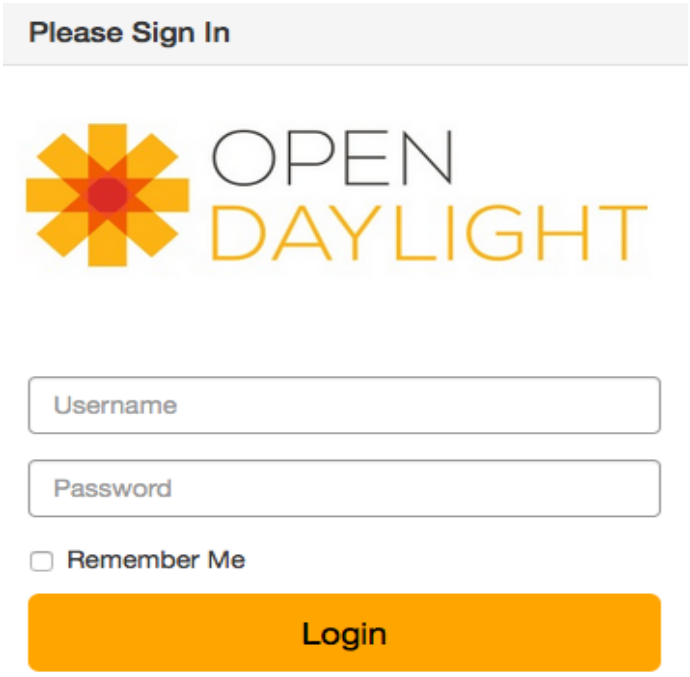
[3] https://www.opendaylight.org/

	autenticação na interface WEB.
feature:install odl-aaa-authn	Recomendado que sempre que possível, AAA seja usado para implementar a autenticação.
feature:install odl-mdsal-apidocs	*Model-Driven Service Abstraction Layer* (MD-SAL), a abordagem *Model-Driven* apresenta uma oportunidade para unificar as duas APIs *Northbound* e uma *Southbound* e as estruturas de dados utilizadas nos diferentes serviços e componentes de um controlador de SDN.
feature:install odl-openflowplugin-all	Um *plugin* para suportar implementações de especificações *OpenFlow* como ele se desenvolve e evolui. Ele pode ser estendido para adicionar suporte para especificações *OpenFlow* subsequentes.
feature:install odl-netconf-connector-all	É um protocolo para configuração de elementos de redes.

Fonte: Elaborada pelo autor.

Após da instalação dessas *features*, já é possível obter o acesso via interface WEB no endereço: http://localhost:8181/index.html. A figura 4 apresenta a tela de login para que seja realizada a autenticação no controlador. O *OpenDayLight* vem configurado com o seu login e senha como “admin”.

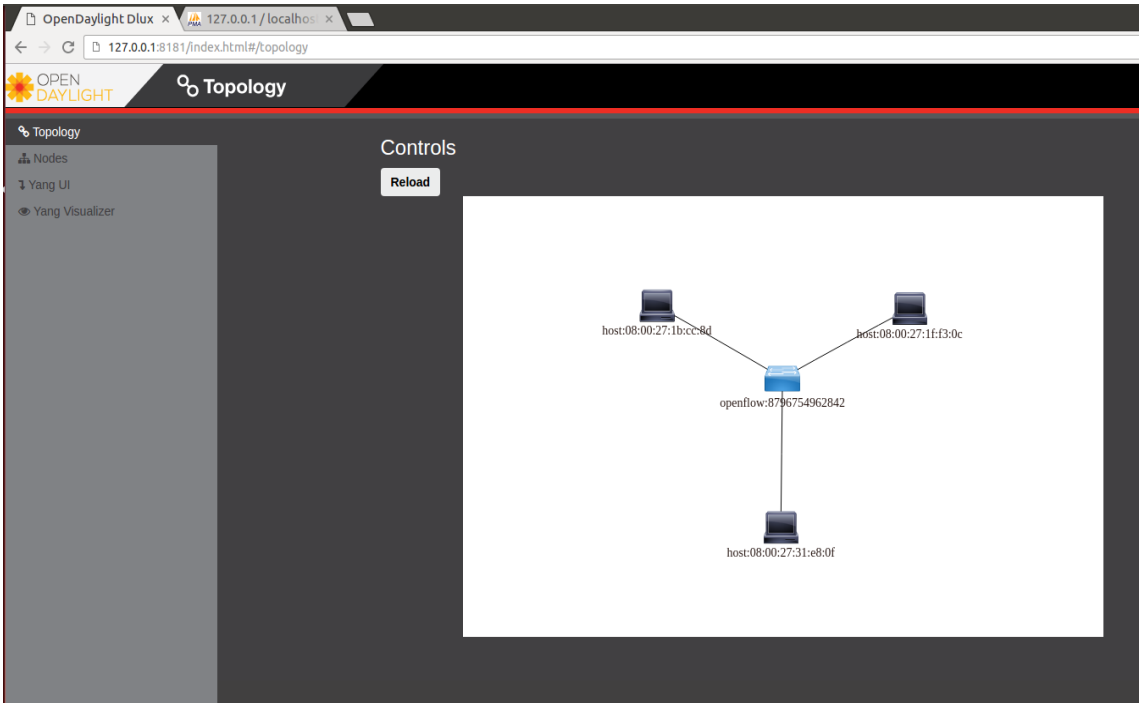
Figura 4 – Tela de autenticação do *OpenDayLight*.



Fonte: Opendaylight.org

Após o login, uma tela com os nós conectados ao *switch* deve aparecer, como mostra a figura 5:

Figura 5 – Tela após a autenticação.



Fonte: Elaborado pelo autor.

4.3 ENGENHARIA DE TRÁFEGO

A engenharia de tráfego tem como objetivo otimizar o desempenho das redes. Para alcançar essa otimização, de acordo com Duarte (2009), a engenharia de tráfego avalia, modela, caracteriza o tráfego, monitora e o controla, visando sempre evitar congestionamentos na rede. Segundo Carvalho (2013), a engenharia de tráfego possui alguns objetivos específicos, são esses:

- Manter o nível de QoS que for solicitado pelas aplicações;
- Melhorar a eficiência na utilização dos recursos da rede;
- Elevar a confiabilidade da rede, diminuindo possíveis erros, vulnerabilidades e falhas que venham a atingir a rede;

Para alcançar esses objetivos a engenharia de tráfego precisa de um gerenciamento da capacidade de rede e dos fluxos que estão sendo trafegados. Esse gerenciamento de capacidade são ações como planejamento de capacidade de rede, controle de roteamento e o gerenciamento de recursos. De acordo com Carvalho (2013), os métodos de controle podem ser reativos e proativos. Nos métodos reativos, as informações coletadas são usadas para corrigir e adaptar a rede às mudanças que estão acontecendo. No método proativo, são utilizadas ações preventivas a fim de impedir que mudanças indesejadas ocorram.

No final da década de 1980 foi desenvolvida a tecnologia de redes ATM (*Asynchronous Transfer Mode*). Nessa época, o principal desafio da engenharia de tráfego era prover o controle de congestionamento de tráfego para atender a requisitos de desempenho de multimídia que vinha crescendo com o passar do tempo. A ATM consiste de um meio para enviar informações em modo assíncrono, através de pacotes de tamanho fixo, conhecidos como células. Uma célula contém um endereço que é utilizado pelos equipamentos para informar o seu destino. Toda comunicação ATM é orientada a conexão, ou seja, uma conexão deve ser estabelecida antes das células serem enviadas.

Em Redes ATM, é fundamental a implementação do controle de congestionamento para atender a requisitos de QoS de aplicações de dados, vídeos e áudios. No controle de congestionamento, existem dois métodos principais, o controle reativo e o controle preventivo. No controle reativo, os nós de origem param o fluxo de tráfego no início do congestionamento para dar informações sobre esse acontecimento, porém isso traz desvantagens, pois as informações só são repassadas após o congestionamento ter ocorrido. No controle preventivo, os nós de origem tentam impedir que a rede chegue a passar por congestionamento, ao invés de esperar ele acontecer.

Redes ATM possuem limitação no transporte de pacotes de acordo com a banda disponível. Através do aumento contínuo do tráfego, a banda pode chegar a um limite onde a rede fica congestionada, ocasionando a perda das células que estão sendo enviadas. Nesse momento, acontece o processo de reenvio de pacotes que estão sendo perdidos, que começam a disputar a rede com o novo tráfego que vem chegando, o que deixa a rede sem condições de responder às requisições com êxito.

As Redes MPLS (*Multi-Protocol Label Switching*) utilizam rótulos de tamanho fixo que são utilizados para determinar o próximo salto do pacote. Os roteadores vizinhos devem ter a possibilidade de se comunicarem e entenderem os detalhes implantados nesses rótulos. Para realizar esses procedimentos se faz necessária a utilização de um roteador compatível com a tecnologia MPLS, que roteadores IPs comuns não são capazes de realizar. Engenharia de tráfego em MPLS traz vantagens por conseguir implementar a ideia de roteamento explícito entre a origem e o destino. Os rótulos MPLS são distribuídos pelo protocolo de distribuição de rótulos LDP (*Label Distribution Protocol*).

Os roteadores intermediários, conhecidos como LSR (*Label Switching Router*), diferente dos roteadores tradicionais, não analisam sua tabela de roteamento, nem o IP de destino do pacote. Esses roteadores encaminham o pacote de acordo com o rótulo MPLS. Para realizar esse procedimento, antes de encaminhar os pacotes, os roteadores LSR removem o rótulo do pacote e adicionam um novo que mostrará ao roteador seguinte por onde o pacote deve ser trafegado. Como esses rótulos não existem em uma rede IP convencional, para que uma rede MPLS funcione junto de uma rede IP tradicional é necessário a utilização de um roteador de borda, chamado de: LER (*Label Edge Router*). Esses roteadores são responsáveis por rotular o pacote IP que entra em uma rede MPLS e também são responsáveis por retirar esse rótulo quando o pacote deixa de fazer parte da rede. A sequência de LSRs que o pacote vai passar até sair da rede MPLS é chamado de LSP (*Label Switched Path*).

4.4 ENGENHARIA DE TRÁFEGO EM SDN

Nos equipamentos de rede tradicionais, as tomadas de decisão ocorrem no próprio dispositivo, ou seja, o roteamento dos pacotes de rede é realizado por algoritmos desenvolvidos previamente pelos fabricantes o que torna a modificação na maioria das vezes impossível de ser realizada geralmente por serem fechados, onde o equipamento toma as decisões para onde encaminhar o pacote. Para contornar essa situação surgiu a tecnologia SDN – *Software Defined Network*, em uma ação conjunta entre a indústria e a academia.

Redes definidas por software são uma abordagem que visa a flexibilidade da rede. Nesta abordagem as tomadas de decisão que antes eram implementadas no próprio dispositivo, passam agora a serem realizadas em uma entidade externa, dando a possibilidade da rede ser controlada a partir de aplicações. Essas entidades são conhecidas como controladores de rede ou sistemas operacionais de rede que são responsáveis por controlar aplicações específicas para a finalidade de cada rede (COSTA;2013). Aplicar a tecnologia SDN implementado como um sistema centralizado em engenharia de tráfego, leva uma grande vantagem ao modelo convencional. Algumas dessas vantagens são, segundo Akyildiz et al., (2014):

- A tecnologia SDN inclui uma visão centralizada e global das informações da rede. As limitações de recursos de rede e alteração do status da rede são obtidas de forma dinâmica, além de informações de aplicações globais, como por exemplo, requisitos de QoS são informações que podem ser coletadas.
- A possibilidade de programar a rede sem ter que lidar com cada elemento individualmente, ou seja, dispositivos dos planos de dados podem ser programados de forma dinâmica pelo controlador centralizado, para alocar os recursos da rede com o objetivo de evitar congestionamento da rede e melhorar o desempenho de QoS.
- Os elementos do plano de dados, independente dos fornecedores, possuem uma interface unificada e aberta, permitindo ao controlador a programação do plano de dados e do estado de coleta da rede.
- Vários caminhos de saída na tabela dos *switches* podem fazer uma gestão dos fluxos mais flexível e eficiente. Desde o surgimento da tecnologia SDN, tem sido aplicada essa técnica a uma variedade de ambientes de rede, ou seja, redes de empresas, redes de centros de dados em grande escala, etc.

Figura 6 – Mecanismos atuais de engenharia de tráfego em SDN.

Fonte: Traduzida pelo autor. (2014).

Como mostrado na figura 6, os mecanismos atuais de engenharia de tráfego concentram-se principalmente em quatro pilares: Gestão de fluxo, tolerância a falhas, atualização de topologia e análise de tráfego/caracterização. Desses pilares, nosso projeto vai trabalhar principalmente na Gestão de fluxos através da determinação do caminho que o pacote irá trafegar de acordo com decisões tomadas pelo controlador e na análise de tráfego, para que seja possível programar no controlador qual o tipo de pacote correto a ser encaminhado para determinado caminho. Nosso projeto consiste em prover balanceamento de carga baseado em políticas bem definidas. Entretanto, elas não são autônomas, ou seja, não alteram o comportamento do tráfego de acordo com o estado da rede.

5 PROPOSTA

A nossa proposta baseia-se no desenvolvimento de uma solução de engenharia de tráfego para uma rede múltiplas conexões com a Internet, baseado em SDN. Os *scripts* criados no desenvolvimento do projeto estão disponíveis no repositório *GitHub*[4].

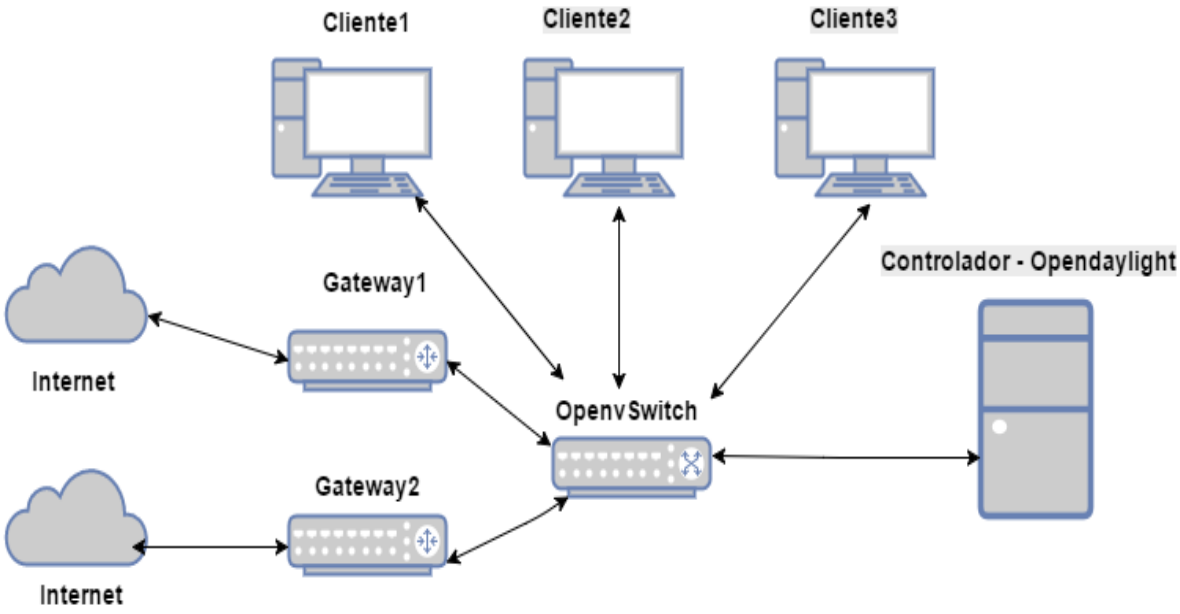
Na figura 7 é possível ter uma visão geral do que o projeto propõe. Nela pode-se observar um servidor, onde está instalado o controlador SDN *OpenDayLight*, que possui uma aplicação desenvolvida para encaminhamento dos pacotes. O servidor armazena os *scripts* responsáveis por criar os fluxos de acordo com políticas lidas em arquivos .XML.

[4] https://github.com/wendelmcl/EtOpendaylight

Além do controlador SDN, o servidor também possui instalado o *Switch OpenFlow Open vSwitch*, onde ficarão definidos os fluxos para encaminhamento de pacotes desenvolvidos pelo controlador.

Dois *Gateways* foram utilizados para a conexão externa com a Internet. Esses *gateways* recebem solicitações para encaminhar pacotes dos *hosts* de acordo com o fluxo definido no *Switch*. Esses fluxos, como mencionado anteriormente, são criados baseados em políticas de alto nível criadas em arquivo .XML. Essas políticas são definidas de acordo com tipo de aplicação do pacote.

Figura 7 – Visão geral da solução proposta.



Fonte: Elaborada pelo autor.

5.1 Solução proposta

O *Script* principal, desenvolvido em *Python*, deve criar todos os fluxos que estão escritos no arquivo de políticas de alto nível automaticamente, sempre que for executado, podendo também ter os fluxos criados de maneira manual, preenchendo os campos referentes ao clientes e *gateways*, além do protocolo que ele quer que esteja relacionado ao fluxo que ele deseja criar. A figura 8 ilustra o momento em que o script é executado e os fluxos são criados de maneira automática.

Figura 8 - Criação dos fluxos.

```
wendel@wendel-S460-L-BG24P1:~/Documentos$ ./final.py
Criando fluxos...
Porta: 80
MAC do Cliente: 08:00:27:39:65:79
MAC do Gateway: 08:00:27:25:23:78
Status: 200 OK
Status: 0.064813
Porta: 20
MAC do Cliente: 08:00:27:12:34:85
MAC do Gateway: 08:00:27:89:58:34
Status: 200 OK
Status: 0.004855
Porta: 22
MAC do Cliente: 08:00:27:25:02:53
MAC do Gateway: 08:00:27:25:23:78
Status: 200 OK
Status: 0.004684
```

Fonte: Elaborada pelo autor.

Nesse *script*, também são lidas políticas de alto nível que estão em um arquivo .XML, chamado de políticas.xml. Essas políticas foram determinadas através dos dados referentes aos clientes, *gateways* e protocolos. Onde estão selecionados como filtros, clientes e protocolos e o *gateway* de destino definido como a ação. As políticas estão implementadas como mostra a figura 9:

Figura 9 – Implementação das políticas de alto nível.

politicas.xml

```
1 <?xml version="1.0" ?>
2 <policies>
3         <policy id="1">
4                 <cliente>cliente1</cliente>
5                 <protocolo>http</protocolo>
6                 <gateway>gateway1</gateway>
7         </policy>
```

Fonte: Elaborada pelo autor.

Nesse exemplo o *script* ao ler o ID da política, consegue atribuir ao arquivo .XML responsável pelas regras, onde todo o tráfego do cliente1 referente ao protocolo HTTP será repassado ao gateway1, que possui uma conexão externa a internet. O protocolo em questão terá seu nome convertido na sua porta, no exemplo seria utilizada a porta 80.

Após a leitura do arquivo .XML das políticas de alto nível, o *script* fica responsável pela criação de um outro .XML, chamado de regras.xml. Esse arquivo de regras segue um

padrão que é o mesmo utilizado pela *API YANG-UI* com as regras que correspondem aos campos de regras do *OpenDayLight*, assim como nos campos *opendaylight-inventory*. Essas regras de encaminhamento possibilitam a conexão ao controlador *OpenDayLight*. No arquivo das regras, seguindo o modelo que é utilizado pelo *Opendaylight*, além dos campos referentes aos clientes, *gateways* e protocolos, também existem outros campos como o número da tabela onde o fluxo será inserido, o identificador do fluxo e a prioridade que o fluxo vai receber.

Pelo padrão de regras do *OpenDayLight*, os campos de origem, destino e porta devem ser inseridos de acordo com o endereço MAC do cliente, dos *gateways* e o número da porta do protocolo, nesse caso é necessário que após a leitura do arquivo de políticas, o *script* principal traduza os valores onde estão o nome dos clientes e *gateways* pelos seus respectivos endereços MAC que o *script* obtém através de um banco de dados e repasse esses valores ao arquivo de regras, assim, ao conectar o arquivo de regras ao *OpenDayLight* será possível gerar o fluxo, pois o padrão foi seguido corretamente.

Para a criação de novas políticas, o administrador da rede deve adicionar no arquivo os campos referentes à configuração das mesmas. O primeiro campo é referente ao ID da política, deve ser passado de forma sequencial para uma melhor leitura do *Script* principal, sempre seguindo o padrão do arquivo. Após a inclusão do ID, o administrador deve selecionar o cliente que vai ser utilizado, todos os clientes estão armazenados em um banco de dados *MySQL*, caso o administrador escreva um cliente que não esteja no banco de dados é gerado um erro referente a cliente inválido. Da mesma forma que acontece com os clientes, os *gateways* também precisam ser identificados na política, todos *gateways* estão armazenados no mesmo banco de dados e assim como ocorre com os clientes, caso seja escolhido um *gateway* que não está armazenado no banco é gerado um erro de *gateway* inválido.

Para que seja possível a geração desses fluxos é necessário que o *script* principal esteja responsável por se conectar a um *script* desenvolvido em *ShellScript* que possui um comando chamado *cURL*[5] que oferece uma ferramenta de linha de comando que torna possível a transferência de dados usando diversos protocolos. *cURL* utiliza *libcurl*, que suporta vários protocolos de internet, que incluem: http, https, ftp, ftps, telnet, o file, imap, pop3, smtp e diversos outros.

No *script* desenvolvido em *ShellScript*, o comando utilizado para geração dos fluxos pode ser observado como no exemplo a seguir:

[5] https://curl.haxx.se/

curl –X PUT –d @regras.xml –H “Content-Type: application/xml” –H “Accept: application/xml” —user “admin”:”admin” http://127.0.0.1:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/flow-node-inventory:table/1/flow/1.

Onde o arquivo regras.xml que foi criado pelo *script* principal estabelece as regras que são enviadas via *API REST* para o controlador. O usuário e senha estão configurados como “admin”. O campo correspondente a “openflow:1” é referente ao *switch* que está conectado ao controlador. Após a passagem desse comando, o *script* principal estabelece uma resposta vinda do *API REST*, 200, caso o fluxo tenha sido recebido com sucesso pelo *switch*, ou outro valor caso o fluxo não tenha sido aplicado com sucesso. Após a criação dos fluxos automáticos pelo *Script*, é iniciado um menu com as opções de configurações ao administrador da rede, como demostra a figura 10:

Figura 10 – Tela Principal.

```
Porta: 25
MAC do Cliente: 08:00:27:25:02:53
MAC do Gateway: 08:00:27:25:23:78
Status: 200 OK
Status: 0.007971

Escolha sua opcao:
 1 - Cliente
 2 - Gateway
 3 - Inserir fluxo
 4 - Listar fluxos
 5 - Excluir fluxos
 6 - Sair
```

Fonte: Elaborada pelo autor.

Existe também a possibilidade da leitura e execução de fluxos *defaults* que são criados sem especificação do protocolo a ser passado, são criados apenas caso o administrador da rede queira conexão externa aos clientes, independente dos protocolos a serem processados. O *Script* também possibilita a exclusão de um fluxo através do ID do mesmo, que é obtido através de um valor padrão, sendo ele o mesmo valor do ID da tabela onde ele está sendo criado, como é ilustrado no menu da figura 10 no campo 5.

No *script* principal também foram criadas funções que possibilitam ao administrador da rede adicionar um cliente, listar os que já foram criados, além de atualizar suas informações e excluí-los. Essas mesmas funções também são atribuídas em relação aos *gateways*, podendo adicionar, listar, atualizar as informações e excluir o *gateway* desejado. A

figura 11 apresenta o menu referente aos clientes, como mencionado anteriormente. O mesmo modelo também é implementado para os *gateways*.

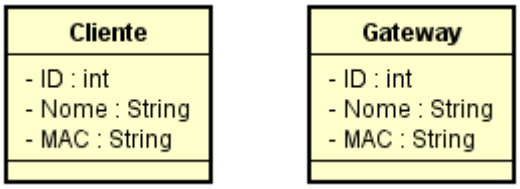
Figura 11 – Menu do cliente.

```
Escolha sua opção
 1 - Listar os Clientes
 2 - Adicionar um Cliente
 3 - Atualizar Cliente
 4 - Deletar Cliente
 5 - Sair:
```

Fonte: Elaborada pelo autor.

Todas essas informações estão armazenadas em um banco de dados *MySQL*. Esse banco está com o nome de "projeto" possuindo duas tabelas chamadas de "cliente" e "*gateway*", cada uma dessas tabelas possuem os campos Id, Nome e Mac para retornar esses valores ao *script*. A figura 12 apresenta as tabelas criadas no banco, junto de seus atributos.

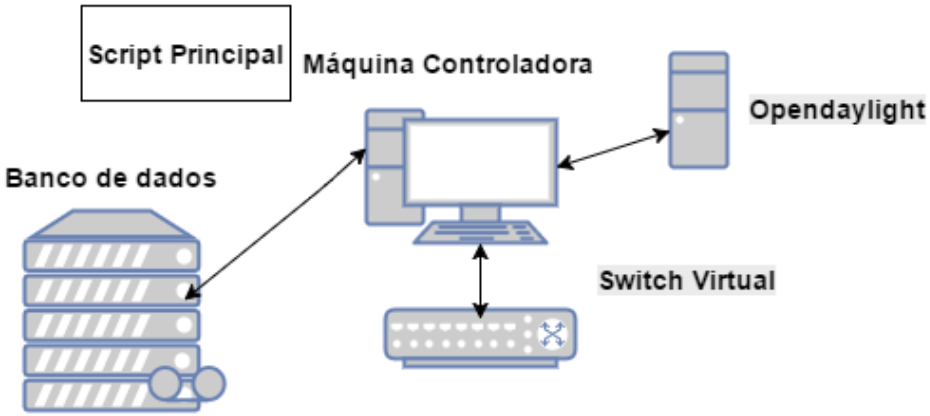
Figura 12 – Tabelas criadas no Banco de dados.



Fonte: Elaborada pelo autor.

A máquina controladora, ou seja, a máquina onde estão instaladas as ferramentas do controlador *OpenDayLight* e do *Open vSwitch* está configurada com o sistema operacional Ubuntu Desktop 14.04 com 4gb de memória RAM, nela além das ferramentas já citadas, foi criado um banco de dados para enviar os dados dos clientes e dos *gateways* para aplicação. A figura 13 ilustra os componentes que estão instalados na máquina principal.

Figura 13 – Componentes instalados na máquina principal.



Fonte: Elaborada pelo autor.

6 VALIDAÇÃO DA SOLUÇÃO

Nesta seção são apresentados tanto os cenários utilizados no projeto, como a instalação das ferramentas utilizadas e os resultados da solução proposta nesse projeto.

Foram realizados os testes em uma máquina do Laboratório de Redes da Universidade Federal do Ceará, Campus Quixadá, através de máquinas virtuais criadas e *scripts* desenvolvidos com o objetivo de auxiliar os testes para validar a solução.

6.1 Instalação do controlador SDN *OpenDayLight*

O controlador reúne informações sobre a inteligência das redes, executando seu algoritmo para fazer análises e em seguida, determinar as novas regras de encaminhamento de pacotes em toda rede. É um controlador desenvolvido em Java que pode ser executado a partir de qualquer sistema operacional, desde que ele tenha suporte a Java. O *OpenDayLight* utiliza algumas ferramentas para sua execução, são essas: *Maven,* OSGI, JAVA interfaces, *APIs REST*.

O controlador utiliza a ferramenta *Maven* para facilitar a compilação. OSGI é utilizado para o carregamento de arquivos de pacotes JAR. A Interface JAVA é usada para monitorar eventos. *APIs REST* são usadas como um gerente de topologia, um programador de fluxos e como ferramenta para roteamento estático.

A versão do *OpenDayLight* utilizada no nosso projeto é a *Lithium-SR4*[6] de março de 2016. A instalação do controlador foi feita em uma máquina virtual, via *VirtualBox*. A máquina está configurada da seguinte forma: 4gb de memória RAM, 20gb de HD, com o sistema operacional Ubuntu 14.04 LTS 64 bits do laboratório de Redes e possuindo uma interface de rede *HostOnly (VBoxNet).* Após a instalação da ferramenta, é necessária a instalação de alguns componentes conhecidos como *"features"*. Alguns exemplos de *features* são: *(feature:install odl-dlux-all)* para que seja possível acessar o controlador via interface web e *(odl-restconf-all)* para que seja realizada a autenticação no controlador em sua interface web. Depois de feita a instalação dos componentes, aguarda-se em torno de cinco minutos para poder acessar a interface web do controlador.

Para a criação dos fluxos na interface web, a *API Yang UI* é responsável pela criação dos fluxos no campo *opendaylight-inventory*, onde você pode criar suas regras de encaminhamento. Após criar os fluxos na *API Yang UI* é gerado tanto um arquivo .XML, como um arquivo *JSON*. Nosso projeto não utiliza a interface web do controlador, todo o

[6] https://nexus.opendaylight.org/content/repositories/opendaylight.release/org/opendaylight/integration/distribution-karaf/0.3.4-Lithium-SR4/distribution-karaf-0.3.4-Lithium-SR4.tar.gz

fluxo e conexões são feitos através de *scripts* desenvolvidos em *Python* que se conectam ao controlador. Caso seja utilizada a interface web do controlador a tela de autenticação pode ser acessada através do endereço: http://localhost:8181/index.html com o login e senha como “admin”.

No nosso projeto, o controlador é responsável por criar os fluxos para encaminhar os pacotes através de políticas de alto nível geradas em arquivos .XML. Nessa máquina, são criados os scripts em *Python* onde baseadas nessas políticas de alto nível se cria um outro arquivo .XML com as regras de encaminhamento que são enviadas para o *switch* virtual, onde é gerado o fluxo e assim os pacotes serem encaminhados para o destino correto.

6.2 Instalação do *Open vSwitch*

De acordo com PFAFF et al., (2009), uma das maneiras de se implementar um comutador *OpenFlow* em computadores pessoais é com o *Open vSwitch*. No nosso projeto, o *Switch* virtual foi instalado na mesma máquina em que o controlador se encontra. Após a instalação do *Open vSwitch*, é necessária a criação de uma *bridge* para que seja possível conectá-la ao controlador, a criação da *bridge* é feita através do comando: sudo ovs-vsctl add-br ‘nome_da_bridge’.

Logo após a criação da bridge, você já consegue conectar o *switch virtual* ao controlador utilizando o comando: sudo ovs-vsctl set-controller ‘nome_da_bridge’ tcp:127.0.0.1:6633, além de ter a possibilidade de conectar o *switch* com o gerente através do comando: sudo ovs-vsctl set-manager tcp:127.0.0.1:6640. Após se conectar ao controlador, aguarde alguns minutos até que o controlador reconheça a *bridge* criada. Com o comando sudo ovs-vsctl show você pode observar se a conexão do *switch* com o controlador já está ativa, além de outros detalhes, como o nome da *bridge*, as interfaces ativas e o número de portas da *bridge*. A figura 14 exemplifica a como resposta ao comando:

Figura 14 – Dados da Bridge

```
wendel@wendel-S460-L-BG24P1:~$ sudo ovs-vsctl show
e9717259-81fa-44a9-a716-28041fb66ac6
    Manager "tcp:127.0.0.1:6640"
    Bridge "br0"
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        Port "eth0"
            Interface "eth0"
        Port "br0"
            Interface "br0"
                type: internal
    Bridge br-int
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
```

Fonte: Elaborada pelo autor.

Agora, já é possível acessar o controlador, para além de realizar a criação dos fluxos no *Switch* Virtual, realizar o encaminhamento dos pacotes.

6.3 Cenário

Para os experimentos do projeto, foi necessária a criação de seis máquinas virtuais no Laboratório de Redes da Universidade Federal do Ceará – Campus Quixadá. As máquinas foram configuradas como ilustra a tabela 3:

Tabela 3 – Configuração das máquinas virtuais.

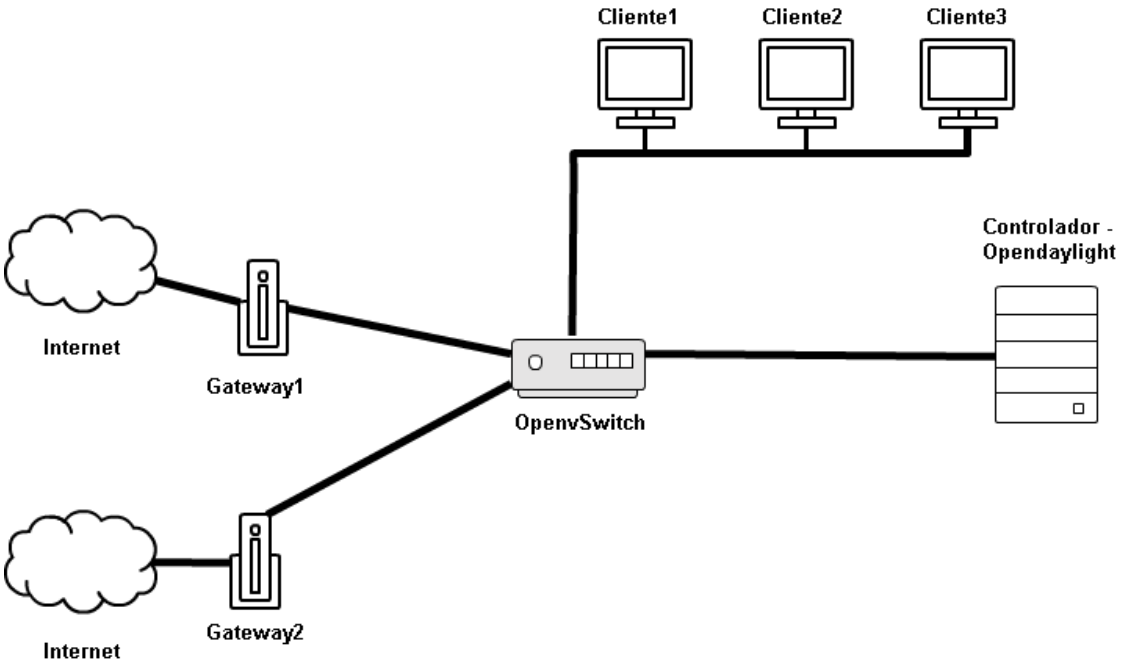
Máquina	**Sistema Operacional**	**Memória RAM**	**Interface de Rede**	**IP**
Controlador/Switch_Virtual	Ubuntu_Desktop14.04 64GB	4GB	HostOnly	192.168.1.2/24
Gateway1	Ubuntu_Server14.04 64GB	512MB	HostOnly/ NAT	192.168.1.3/24 10.0.2.15/24
Gateway2	Ubuntu_Server14.04 64GB	512MB	HostOnly/ NAT	192.168.1.4/24 10.0.2.16/24
Cliente1	Ubuntu_Desktop14.04 64GB	1 GB	HostOnly	192.168.1.5/24
Cliente2	Ubuntu_Desktop14.04 64GB	1GB	HostOnly	192.168.1.6/24
Cliente3	Ubuntu_Desktop14.04 64GB	1GB	HostOnly	192.168.1.7/24

Fonte: Elaborada pelo autor.

Todas as máquinas como pode ser visto na tabela 3, estão configuradas com a interface de rede *HostOnly*. Essa interface foi adicionada para que todas as máquinas consigam se comunicar através de uma rede local. As máquinas que possuem a função de *gateway*, além de uma rede local, estão também configuradas com uma interface *NAT* para se conectarem à rede externa e assim encaminhar os fluxos vindos dos clientes.

No cenário utilizado, todo o fluxo dos clientes é programado através de *Scripts* que ficam armazenados na máquina onde estão localizados o controlador e o *Switch* virtual. Através da rede interna, todas as máquinas conseguem se comunicarem entre si, porém, sem a possiblidade dos clientes acessarem uma rede externa. Com a criação das políticas de alto nível para o desenvolvimento do projeto, cada *gateway* ficou responsável por encaminhar o tráfego dos clientes de acordo com o protocolo da aplicação. Na figura 15 é possível ilustrar como os hosts estavam distribuídos no cenário. Com o *Switch* virtual no meio para repassar aos *gateways* as solicitações vindas dos clientes após a aplicação de políticas e posteriormente a criação de regras enviadas ao controlador que é uma entidade externa, criando assim o fluxo no *Switch* virtual.

Figura 15 – Cenário utilizado.



Fonte: Elaborada pelo autor.

De acordo com esse cenário, diversas políticas foram implementadas para validar a solução. Seguindo critérios referentes aos protocolos das aplicações, todo tráfego foi programado no controlador através de *Scripts*, como explicado na seção 5.1.

6.4 Validação da Solução

Para que fosse possível através do estudo de caso validar a solução, foi necessária a utilização de uma ferramenta chamada *Zabbix*[7]. O *Zabbix* é uma ferramenta de monitoramento de dispositivos através da rede. Diversas informações podem ser coletadas por ele, como por exemplo, CPU, memória, disco, além de dados sobre as interfaces de rede.

No nosso projeto foi utilizado o monitoramento de interfaces de rede dos clientes e dos *gateways* para demostrar a mudança do fluxo após a alteração da política aplicada. Na máquina hospedeira foi instalada a ferramenta de monitoramento para que o gerente *Zabbix* conseguisse habilitar as máquinas que seriam monitoradas. Todos os hosts estão na mesma rede, inclusive a máquina hospedeira. Esta adiciona uma interface *Vboxnet0* quando ela é configurada no *VirtualBox*. E assim todas as máquinas que estavam configuradas no experimento conseguiram se comunicar. Em cada máquina utilizada para análise, foi instalado um agente *Zabbix* para terem a possibilidade de serem monitorados pelo gerente instalado na máquina hospedeira.

Tanto os clientes quantos os *gateways* foram adicionados à ferramenta. Foram realizados testes utilizando tráfego HTTP e uma aplicação UDP na porta 5001. O Cliente através do comando *"wget"* faz diversas solicitações HTTP, que estão sendo processadas pelo Gateway1. Além disso, também foram feitas requisições UDP pelo Cliente com a utilização da ferramenta Iperf[8] para geração desse tipo de tráfego. Como pode ser visto na figura 16, a política referente ao tráfego UDP enviado pelo Cliente está configurado para ser processado pelo Gateway1.

Figura 16 – Política referente ao tráfego UDP.

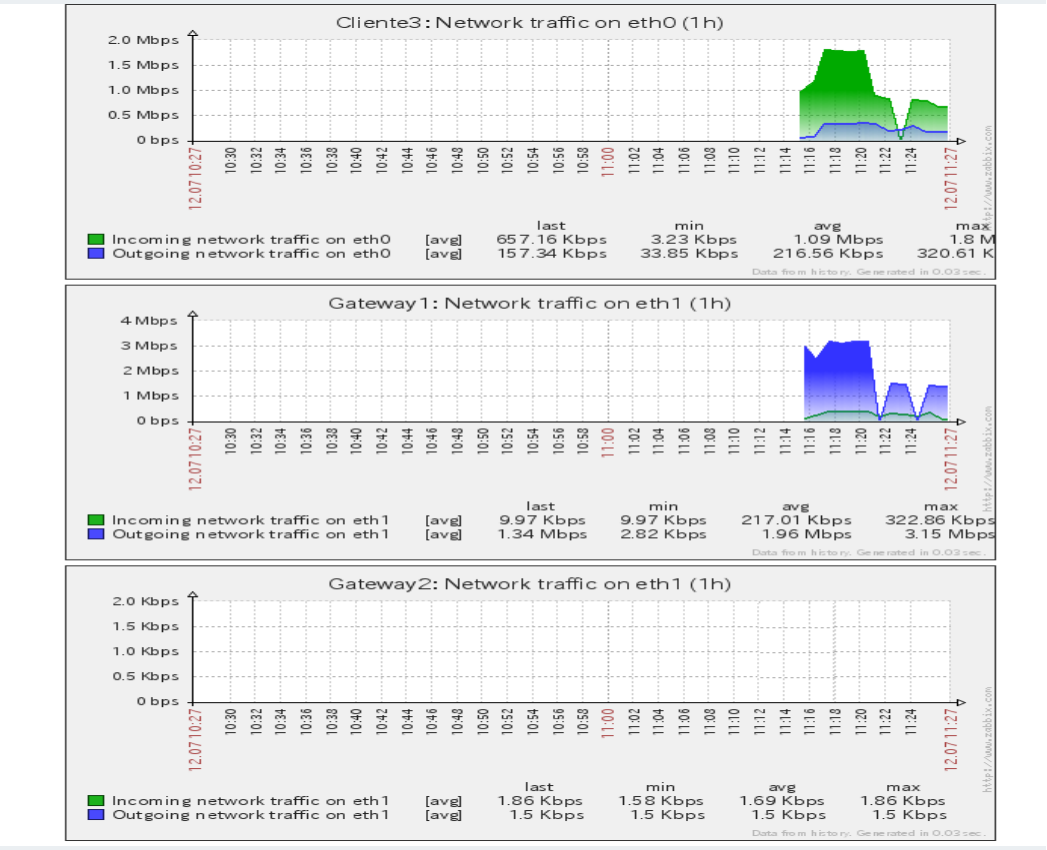
```
<policy id="4">
        <cliente>cliente3</cliente>
        <protocolo>udp</protocolo>
        <gateway>gateway1</gateway>
</policy>
```

Fonte: Elaborada pelo autor.

[7] http://www.zabbix.com
[8] https://iperf.fr/

Na figura 17 podemos observar que no início do nosso experimento, todos os tráfegos (HTTP e UDP) estavam sendo enviados para o Gateway1.

Figura 17 – Tráfego inicial.



Fonte: Elaborada pelo autor.

Em seguida, uma nova política foi criada, estabelecendo que o tráfego UDP, que saia do Cliente, fosse enviado agora pelo Gateway2. A figura 18 ilustra a mudança no arquivo de políticas de alto nível referente a alteração do *Gateway* responsável por processar o tráfego UDP vindo do Cliente.

Figura 18 – Mudança de política do tráfego UDP.

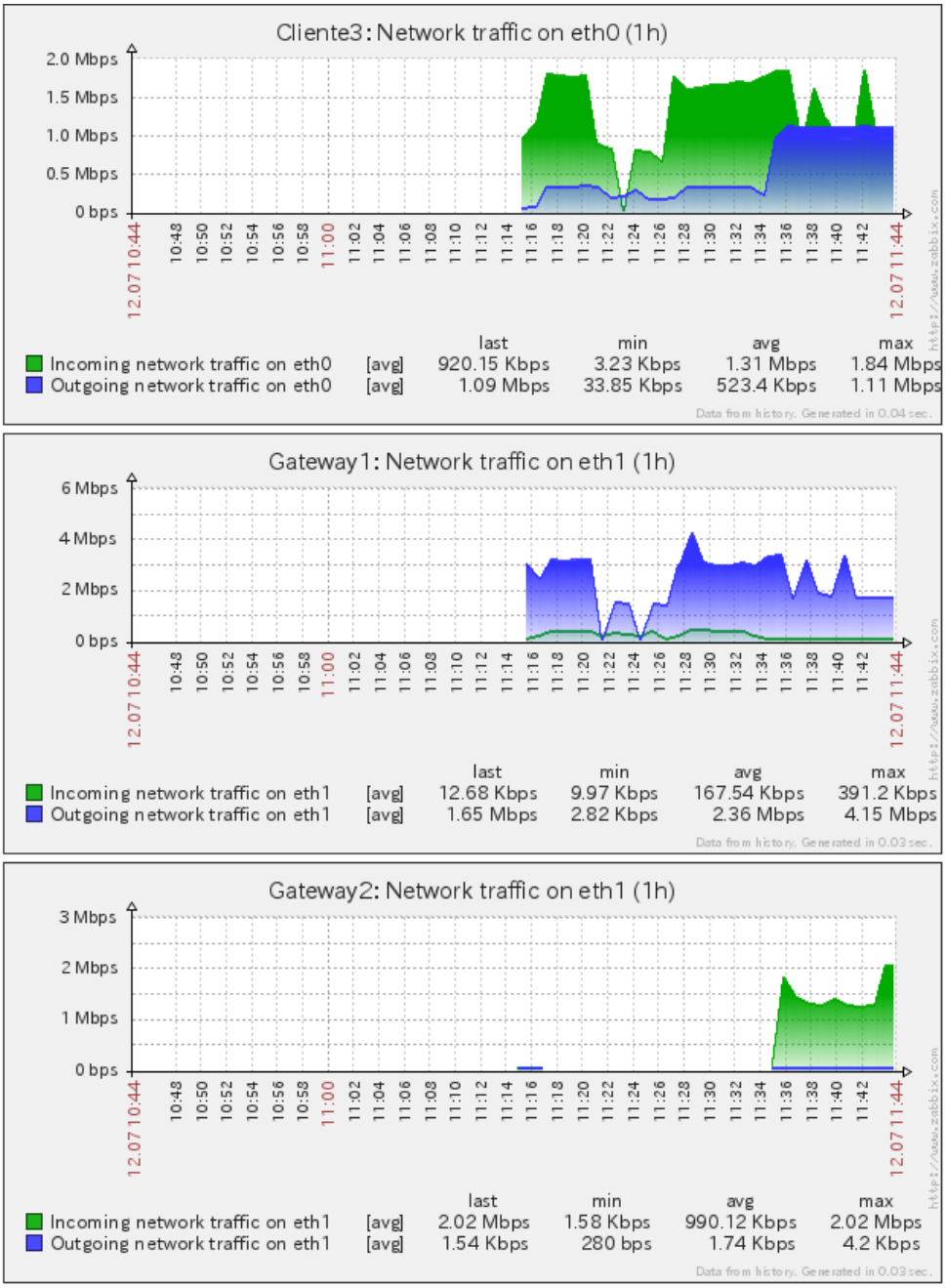
```
<policy id="5">
        <cliente>cliente3</cliente>
        <protocolo>udp</protocolo>
        <gateway>gateway2</gateway>
</policy>
```

Fonte: Elaborada pelo autor.

Na figura 19, podemos visualizar o momento em que o Gateway1 diminui a carga e para de processar o tráfego UDP do Cliente, sendo este agora processado pelo Gateway2.

No exato momento em que é aplicada uma nova política para o tráfego UDP, que antes era processado pelo Gateway1, o tráfego que chegava ao Gateway1 sofre uma queda até parar de ser processado. Já no Gateway2, o tráfego começa a ser executado. É possível observar também que o Cliente ainda continua a enviar solicitações HTTP que continuam a ser processadas pelo Gateway1, a mudança que ocorre e que foi utilizada para a validação é referente ao tráfego udp que altera o *Gateway* onde o processamento estava ocorrendo.

Figura 19 – Tráfego após mudança de política.



Fonte: Elaborada pelo autor.

7 CONSIDERAÇÕES FINAIS

Neste trabalho, foi possível notar a importância da tecnologia SDN (*Software Defined Network*) para o futuro das técnicas de engenharia de tráfego em Redes de Computadores. Com a possibilidade de programar o fluxo da rede em equipamentos externos, deixando os equipamentos tradicionais do plano de dados apenas como comutadores e repassadores de pacotes, obtém-se uma rede bem mais flexível e dinâmica, o que é essencial se tratando de redes do futuro, tendo em vista o aumento de tráfego gerado.

Para tanto, foi desenvolvida uma solução de engenharia de tráfego para múltiplas conexões externas, baseadas em SDN. No projeto, para que fossem atingidos os objetivos, foram desenvolvidos *Scripts* que se comunicavam com o controlador para que nele fosse gerado o tráfego e assim fosse encaminhado os pacotes de acordo com políticas de alto nível definidas em arquivos .XML. Além da leitura desses arquivos de políticas, o *Script* gerava outro arquivo .XML, agora com as regras a serem aplicadas pelo controlador.

No decorrer do projeto foram encontradas algumas dificuldades no que diz respeito a forma como poderiam ser criados os fluxos no controlador bem como eles seriam repassados ao *Switch* Virtual. Inicialmente foi realizado um estudo das ferramentas para compreensão do seu funcionamento. Além da dificuldade com o funcionamento das ferramentas, outro percalço foi a instalação das ferramentas, onde diversos tutoriais que foram encontrados não executavam de maneira satisfatória.

É esperado que nossa solução seja adotada por usuários e pesquisadores da área, afim de auxiliá-los nas suas pesquisas que estejam relacionadas à engenharia de tráfego em redes definidas por *software*. Além disso, nossa ferramenta pode substituir ferramentas legadas de balanceamento de carga, aplicadas em plataformas de nuvem, como o *HAProxy*[9] no *OpenStack*[10], permitindo uma criação mais flexível de regras de balanceamento e em tempo real.

Para o desenvolvimento de trabalhos futuros, outros fatores podem ser adicionados à decisão do encaminhamento dos pacotes, como a disponibilidade do *Switch* e a carga do enlace, também podem ser incluídos ações de *failover*, além de soluções de qualidade de serviço, disponibilizadas pelo *OpenFlow*, como a Tabela Meter, ficando a cargo do desenvolvedor quais métricas ele deseja que sejam aplicadas.

[9] http://www.haproxy.org/
[10] https://www.openstack.org/

REFERÊNCIAS

AGARWAL, Sankalp; KODIALAM, Murali; LAKSHMAN, T. V. Traffic engineering in software defined networks. In: **INFOCOM, 2013 Proceedings IEEE**. IEEE, 2013. p. 2211-2219. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6567024>

Acesso em: 28 de Outubro de 2015.

AKYILDIZ, Ian F. et al. A roadmap for traffic engineering in SDN-OpenFlow networks. **Computer Networks**, v. 71, p. 1-30, 2014. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128614002254>

Acesso em: 28 de Outubro de 2015.

AHMED, Shamim et al. Orchestration algorithms for network-assisted virtual machine migrations using OpenDayLight Controller. In: **Electrical Information and Communication Technology (EICT), 2015 2nd International Conference on**. IEEE, 2015. p. 106-111. Disponível em : <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7391931>

Acesso em 12 de Março de 2016.

BRAUN, Wolfgang; MENTH, Michael. Software-Defined Networking using OpenFlow: Protocols, applications and architectural design choices. **Future Internet**, v. 6, n. 2, p. 302-336, 2014. Disponível em: < http://www.mdpi.com/1999-5903/6/2/302/htm >

Acesso em 08 de Fevereiro de 2016

CARIA, Marcel; JUKAN, Admela; HOFFMANN, Marco. A performance study of network migration to SDN-enabled traffic engineering. In: **Global Communications Conference (GLOBECOM), 2013 IEEE**. IEEE, 2013. p. 1391-1396. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6831268>

Acesso em: 28 de Outubro de 2015.

COSTA, Lucas Rodrigues. OpenFlow e o paradigma de redes definidas por software. 2013. Disponível em: <http://bdm.unb.br/handle/10483/5674>

Acesso em: 15 de Março de 2015.

KHATTAK, Zuhran Khan; AWAIS, Muhammad; IQBAL, Adnan. Performance Evaluation of OpenDaylight SDN Controller. Disponível em: < https://www.namal.edu.pk/wp-content/uploads/2015/02/14ICPADS_ODLBenchmarking.pdf>

Acesso em: 29 de Outubro de 2015.

KREUTZ, Diego et al. Software-defined networking: A comprehensive survey. **proceedings of the IEEE**, v. 103, n. 1, p. 14-76, 2015. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6994333>

Acesso em: 15 de Março de 2015.

LARA, Adrian; KOLASANI, Anisha; RAMAMURTHY, Byrav. Network innovation using openflow: A survey. **Communications Surveys & Tutorials, IEEE**, v. 16, n. 1, p. 493-512, 2014. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6587999> Acesso em: 29 de Outubro de 2015.

MAYORAL, A. et al. Experimental validation of automatic lightpath establishment integrating OpenDayLight SDN controller and Active Stateful PCE within the ADRENALINE testbed. In: **2014 16th International Conference on Transparent Optical Networks (ICTON)**. IEEE, 2014. p. 1-4. Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6876334>

Acesso em 08 de Fevereiro de 2016.

MCKEOWN, Nick et al. OpenFlow: enabling innovation in campus networks. **ACM SIGCOMM Computer Communication Review**, v. 38, n. 2, p. 69-74, 2008. Disponível em: <http://dl.acm.org/citation.cfm?id=1355746>

Acesso em: 30 de Outubro de 2015.

MEDVED, Jan et al. Opendaylight: Towards a model-driven sdn controller architecture. In: **2014 IEEE 15th International Symposium on**. IEEE, 2014. p. 1-6. Disponível em: < http://www.computer.org/csdl/proceedings/wowmom/2014/4786/00/06918985-abs.html>

Acesso em: 30 de Outubro de 2015.

PFAFF, Ben et al. Extending Networking into the Virtualization Layer. In: **Hotnets**. 2009. Disponível em: <http://openvswitch.github.io/papers/hotnets2009.pdf>

Acesso em 30 de Outubro de 2015.

ROTHENBERG, Christian Esteve et al. OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. **Cad. CPqD Tecnologia, Campinas**, v. 7, n. 1, p. 65-76, 2010. Disponível em: <http://www.researchgate.net/profile/Christian_Esteve_Rothenberg/publication/266292305_OpenFlow_e_redes_definidas_por_software_um_novo_paradigma_de_controle_e_inovao_em_redes_de_pacotes/links/542fec440cf27e39fa99b9a7.pdf>
Acesso em: 15 de Março de 2015.

ZHANG, Junjie et al. Load balancing for multiple traffic matrices using sdn hybrid routing. In: **High Performance Switching and Routing (HPSR), 2014 IEEE 15th International Conference on**. IEEE, 2014. p. 44-49. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6900880>

Acesso em: 28 de Outubro de 2015.