Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Avaliação de Balanceamento de Carga Web em Redes Definidas por Software

Cristiane Pinto Rodrigues

JUIZ DE FORA
DEZEMBRO, 2014

Avaliação de Balanceamento de Carga Web em Redes Definidas por Software

Cristiane Pinto Rodrigues

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Alex Borges Vieira

JUIZ DE FORA

DEZEMBRO, 2014

Avaliação de Balanceamento de Carga Web em Redes Definidas por Software

Cristiane Pinto Rodrigues

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Alex Borges Vieira
D. Sc.

Luciano Jerez Chaves
M. Sc.

Francisco Henrique Cerdeira Ferreira
M. Sc.

JUIZ DE FORA

10 DE DEZEMBRO, 2014

A Deus, por ter me dado força e coragem durante toda esta longa caminhada.

À minha família por sempre acreditar em mim.

Ao Leonardo por estar sempre ao meu lado.

Resumo

O paradigma denominado Redes Definidas por Software (SDN) trouxe um novo conceito sobre como gerenciar e projetar as redes de computadores com a inserção de elementos programáveis. Com SDN, tornou-se mais simples a realização de experimentos nas redes, abrindo espaço para pesquisas de problemas ainda não totalmente resolvidos na área e que possuem um importante papel em redes, como por exemplo, o balanceamento de carga. Para suportar os grandes tráfegos da Internet e manter um tempo de resposta suficientemente rápido para os clientes, é necessário que os recursos de uma rede sejam bem utilizados. Sendo assim, o balanceamento de carga trouxe uma forma de processar as requisições de forma que todos os recursos possam ser bem aproveitados, não sobrecarregando equipamentos específicos. No presente trabalho foi criada uma arquitetura SDN, utilizando o padrão OpenFlow e o controlador POX, na qual foram utilizados diferentes algoritmos para realizar o balanceamento de carga em servidores *Web*. A partir desse contexto foram realizados testes e avaliações através de métricas de desempenho, com o objetivo de comparar os algoritmos propostos e apontar qual deles é o melhor para a realidade da rede em três cenários distintos.

Palavras-chave: Balanceamento de carga, Redes Definidas por Software, avaliação.

Abstract

The paradigm called Software Defined Networks (SDN) brought a new concept about managing and designing computer networks with the inclusion of programmable elements. With SDN has become simpler to perform experiments on networks, making way for research problems not yet fully solved in the field and that have an important role in networks, such as load balancing. To support large Internet traffic and maintain a sufficiently fast response time to customers, it is necessary that the resources of a network are better used. Thus, load balancing brought a way to process requests so that all resources can be better used, not overloading specific equipment. In the present work an SDN architecture was created using the OpenFlow standard and the POX controller, in which we used different algorithms to perform load balancing across Web servers. From this context, tests and reviews were performed through performance metrics, in order to compare the proposed algorithms and point out which one is the best for the network in three different scenarios.

Keywords: Load balancing, Software Defined Network, evaluation.

Agradecimentos

Agradeço primeiramente a Deus, por ter permitido que tudo isso acontecesse, iluminando todo meu caminho. Ele é a base de tudo. Jamais me deixou desistir, me ajudando a superar cada dificuldade e nos momentos que eu mais preciso Ele está comigo. Obrigada meu Deus por me ajudar a levantar em cada tombo, por nunca me deixar desistir. Eu não sou ninguém sem o seu amor.

Agradeço as minhas irmãs Camila e Carina por sempre acreditarem em mim, a minha mãe Cleusa por toda preocupação, ao meu pai João e a minha vó Dina por todas as orações, por me ensinarem sempre que o mais importante é não perder a fé. Meu pai que tantas e tantas vezes me tranquilizou com todo o seu carinho, me dando sempre a certeza que as coisas iriam melhorar. Agradeço a minha família por todo carinho, toda compreensão, que mesmo não podendo me ajudar financeiramente, me ajudou infinitamente com seu amor, me dando a força necessária para continuar lutando. Eu amo vocês!

Agradeço ao meu namorado Leonardo, por todo carinho, amizade, compreensão, por estar ao meu lado sempre nesses últimos anos. Seu amor, sua amizade foram essenciais para que eu pudesse passar por esse obstáculo. Não tenho dúvidas de que sem você esse trabalho não teria sido possível. Essa vitória também é sua amor. Obrigada por tudo. Você é meu presente de Deus!

Agradeço aos meus sogros Bernadete e Emanoel, por todo carinho, por todo acolhimento, por toda ajuda, por me fazer desde sempre parte da família. Isso foi fundamental para essa conquista.

Agradeço aos meus amigos e colegas de faculdade. À minha querida amiga Lenita, por toda sua amizade. Sentirei sempre saudades das nossas conversas.

Agradeço ao orientador Alex pela ajuda, atenção, orientação e por passar toda tranquilidade de que o trabalho ficaria pronto.

Agradeço a todos os professores pelo aprendizado e dedicação demonstrado ao longo do curso.

Agradeço à UFJF pela oportunidade de formação e pela ajuda financeira que foi muito importante para que eu conseguisse concluir meus estudos.

Agradeço a todos que, mesmo não estando citados aqui, tanto contribuíram para a conclusão desta etapa.

"Veja,

Não diga que a canção está perdida

Tenha fé em Deus, tenha fé na vida

Tente outra vez!"

Raul Seixas (Tente outra vez)

Sumário

Lista de Figuras

Lista de Abreviações

SDN	Redes definidas por Software (Software Defined Network)
SSL	Secure Socket Layer
HTTP	Hypertext Transfer Protocol

1 Introdução

Atualmente, muitas das atividades cotidianas da sociedade estão relacionadas a algum tipo de rede de computadores, como a Internet. A grande utilização dos serviços que passam por essas redes as obriga a serem estáveis. Em outras palavras, serviços não podem parar. Assim, realizar tarefas como configurar ou dar manutenção a uma rede de computadores tornou-se um problema bastante complicado. Isso ocorre pois é necessário lidar com muitos equipamentos e sistemas operacionais diferentes.

Dado esse contexto, há novas propostas que visam resolver esse problema, de certa forma. Uma dessas soluções foi a proposição de um novo paradigma denominado *Software Defined Network* ou Rede Definida por Software (SDN), que vem mudando a forma de como modificar a rede, mudando a forma de gerenciá-la. SDN é um paradigma interessante que permite a inovação na forma de como projetar e gerenciar a rede (Feamster et al, 2014).

A pesquisa em Redes Definidas por Software, por se tratar de uma área nova, vem crescendo nos últimos anos. SDN vem ganhando a atenção de grande parte da comunidade acadêmica e da indústria da área. Muita da atenção até o momento tem sido voltada para o padrão OpenFlow (Guedes et al, 2012). Com a padronização do OpenFlow, vendedores e operadores de rede começaram a dar uma importância maior a ele. Em 2011, empresas como Facebook, Yahoo, Google e Microsoft formaram a Open Networking Foundation (ONF). A ONF é uma organização sem fins lucrativos que tem como objetivo promover a tecnologia OpenFlow, trazendo para o mercado normas e soluções SDN (ONF, 2014).

Recentemente, surgiram alguns trabalhos relacionados ao tema em conferências da área (Guedes et al, 2012). Essas novas pesquisas, aliadas ao interessante tema, despertaram a vontade de se pesquisar alguns problemas ainda não totalmente resolvidos na área, como, por exemplo, o balanceamento de carga.

Balanceamento de carga é um tema de pesquisa atrativo, uma vez que um balanceador específico pode-se tornar um ponto de falha e congestionamento, sendo necessário mantê-lo em uma rede com bastante tráfego (Wang et al, 2011). Com o balanceamento de carga na rede será possível dividir o tráfego, não sobrecarregando equipamentos es-

pecíficos.

Com o intuito de encontrar melhores soluções para as técnicas de balanceamento de carga existentes nas redes atuais, uma série de trabalhos vêm destacando o crescimento e a importância de pesquisas relacionadas a esse tema utilizando Redes Definidas por Software. Alguns trabalhos que podem ser citados são Uppal and Brandon (2010); Sherwood et al (2009); Handigol et al (2009); Wang et al (2011); Zhou et al (2014) e Ragalatha et al (2013). Todos eles realizam o balanceamento de carga através da construção de uma arquitetura SDN com o padrão OpenFlow, porém simulam redes e ambientes específicos. Ao avaliar os trabalhos relacionados ao tema, não foram encontradas referências que utilizassem o controlador POX para a realização do balanceamento de carga. O POX se mostra um controlador de fácil utilização para pesquisas e estudos.

O objetivo do presente trabalho é avaliar o desempenho do balanceamento de carga em uma Rede Definida por Software, utilizando o padrão OpenFlow e o controlador POX. Esse balanceamento é feito através de três algoritmos, que são o Aleatório, o *Round Robin* e o Baseado na carga. Para verificar qual dos algoritmos possui o melhor desempenho, eles são comparados em três cenários distintos. Através dessa comparação pode-se verificar qual o algoritmo proporciona o melhor desempenho para a rede em cada cenário.

Os resultados mostram que os algoritmos apresentam desempenhos semelhantes para taxas de chegada mais baixas, nos três cenários. Além de semelhanças, cada cenário apresenta também seus resultados específicos. O algoritmo *Round Robin* teve um desempenho razoável em todos os cenários e o Baseado na carga mostrou-se um bom algoritmo para um cenário em específico.

O restante deste trabalho está assim organizado: o Capítulo 2 apresenta os principais conceitos e componentes das Redes Definidas por Software; o Capítulo 3 apresenta como será realizado o trabalho, descrevendo as propostas para o balanceamento de carga; o Capítulo 4 descreve a metodologia para geração de carga, e os resultados avaliados; o Capítulo 5 traz um apanhado geral dos trabalhos relacionados ao tema de balanceamento de carga em Redes Definidas por Software; e por fim, o Capítulo 6 indica as conclusões deste trabalho e aponta possíveis direções para pesquisas futuras.

2 Redes Definidas por Software

2.1 Introdução

Redes de computadores vem se tornando um recurso essencial para o cotidiano da sociedade. Elas são formadas por um grande conjunto de protocolos, difíceis de gerenciar e evoluem muito lentamente (Feamster et al, 2014). São gerenciadas por meio da configuração de baixo nível dos componentes individuais, o que torna complicado qualquer tipo de modificação (Feamster et al, 2014). Se houver a necessidade de inserção de novas funcionalidades nos equipamentos de comutação de pacotes, o que pode acontecer é um grande gasto de tempo já que o desenvolvimento e os testes ficam restritos ao fabricante. Esses equipamentos costumam ser proprietários, fechados e de alto custo, dificultando qualquer tipo de customização e configuração particular da rede.

O que se vê é uma grande necessidade de mudança de paradigma, onde a rede possa se tornar mais programável, facilitando assim o gerenciamento da mesma, podendo realizar mais facilmente pesquisas e inovações. Uma solução foi a proposição de um novo paradigma denominado Redes Definidas por Software (SDN) (Guedes et al, 2012).

O paradigma SDN propõe que toda a lógica da rede, que antes era feita dentro dos comutadores, passe a ser feita por um elemento lógico centralizado programável, fornecendo assim uma interface de programação para as redes (Levin et al, 2012). A rede passa então a ser controlada através de software, que pode ser programado independentemente do hardware, tornando a rede uma ferramenta programável.

Com Redes Definidas por Software há formas de melhorar a rede sem ter que pará-la ou prejudicar o seu desempenho. Também simplifica o gerenciamento das redes, que passa a ser centralizado, oferecendo a visualização de toda a rede. Com SDN passam a existir aplicações inteligentes como a programação da comutação dos pacotes por parte do gerente da rede, que pode agora automatizar esse encaminhamento, sem redução do desempenho dessa operação (Guedes et al, 2012).

Na arquitetura SDN, o plano de controle é separado do plano de dados. O pri-

meiro é o responsável por toda a lógica de encaminhamento da rede e o segundo encaminha todo tráfego de acordo com a lógica do plano de controle. O software controlador poderá estar localizado em qualquer ponto da rede, podendo evoluir independente do hardware comutador (Sezer et al, 2013). Com ele é possível a criação de programas de alto nível, obtendo mais facilmente o controle sobre a lógica de decisão, adequando assim o comportamento da rede de acordo com a realidade da mesma. Essa separação permite que a rede possa ser mais eficiente e barata, já que passa a não haver dependência do fornecedor, que possui o software de controle e o hardware agregados no mesmo equipamento.

Ao longo do desenvolvimento da rede de computadores, foram surgindo algumas tentativas de transformá-la em algo menos complicado, através da adição de programabilidade. Um exemplo disso são as Redes Ativas. Apesar do seu potencial, tiveram pouca aceitação pela necessidade de alteração dos elementos de rede para permitir que se tornassem programáveis (Guedes et al, 2012). No panorama histórico, SDN surgiu da definição da arquitetura de redes Ethane. Com ela é proposto um mecanismo de controle de acesso distribuído para a rede, porém com o controle centralizado em um nó, responsável pela supervisão (Casado et al, 2009). Na identificação de um novo fluxo, o elemento supervisor realiza consulta para a verificação das políticas de acesso e instala uma nova regra na tabela de fluxo do switch através de uma interface de programação ou descarta os pacotes desse fluxo. Esse padrão foi então ampliado e estabelecido, dando origem ao OpenFlow (McKeown et al, 2008).

OpenFlow é um padrão de código aberto, baseado em Redes Definidas por Software, que tem como base a ideia de se controlar o modo como o tráfego flui dentro da rede (Wang et al, 2011). É possível através dele permitir aos usuários definir políticas de tráfego e determinar quais caminhos os fluxos de dados devem percorrer independente do hardware em si.

A arquitetura de uma SDN é composta, além do padrão OpenFlow, por um outro componente básico denominado controlador. Esse último é o componente de software que faz parte do plano de controle da rede e é o responsável por tomar as decisões quanto ao encaminhamento dos fluxos de pacotes (Sezer et al, 2013). Com o controlador é possível gerenciar as entradas de fluxo na rede, de forma que se torne mais fácil a criação de

aplicações e serviços.

2.2 OpenFlow

Nas redes tradicionais, existem poucas maneiras práticas de se experimentar novos protocolos (por exemplo, protocolos de roteamento) em ambientes suficientemente reais. Isso torna complicada a obtenção da confiança necessária para a implantação desses protocolos (McKeown et al, 2008). Com o surgimento do padrão OpenFlow tornou-se possível obter um domínio maior das redes. Pesquisadores passaram a ter total autonomia para realizar pesquisas, testes e experimentos, modificando a lógica de funcionamento dos equipamentos de rede, sem ter que interferir no funcionamento normal do tráfego de uma rede em produção.

O OpenFlow é um padrão de código aberto, que leva a uma programação simplificada dos dispositivos de rede através de uma interface padronizada ou Aplication Program Interface (API). OpenFlow tem papel importante na arquitetura das Redes Definidas por Software. Ao atuar como uma interface entre as camadas de controle e dados, ele implementa as regras de encaminhamento que tornam o conceito de SDN possível. Em outras palavras, OpenFlow foi proposto para padronizar a comunicação entre os switches (ou outros equipamentos comutadores) e o controlador na arquitetura SDN (Lara et al, 2014).

Além disso, o padrão OpenFlow determina como um fluxo de pacotes que chega ao switch pode ser definido, as ações que podem ser realizadas para cada pacote pertencente a um fluxo e o protocolo de comunicação entre comutador e controlador (Guedes et al, 2012). Dependendo das políticas de tráfego criadas por uma aplicação controladora, um switch que rode OpenFlow pode se comportar como um roteador, switch normal, firewall ou algum tradutor de endereço de rede (Feamster et al, 2014).

Os switches comerciais podem habilitar o OpenFlow como recurso. Vários fabricantes já oferecem produtos com a interface OpenFlow: Juniper, NEC, HP, Dell, OpenvSwitch, Cisco, Ciena, entre outros (Lara et al, 2014). Isso pode ser feito através da atualização do firmware desses equipamentos (Ragalatha et al, 2013). Através dessa ativação, serão adicionados no switch a tabela de fluxo, o canal seguro e o protocolo

OpenFlow.

Apesar do OpenFlow ser o foco principal dos ambientes em Redes Definidas por Software, o paradigma SDN não se limita a ele, pois existem outras possibilidades de desenvolvimento de uma interface de programação que atenda os objetivos do paradigma (Guedes et al, 2012).

2.3 Controlador

Um elemento de grande importância para o SDN é o controlador. Controladores de rede, também denominados Sistemas Operacionais de Rede ou Network Hypervisors (Guedes et al, 2012), são os responsáveis por tomar decisões, adicionar, atualizar e remover as entradas na tabela de fluxos, de acordo com o objetivo desejado. O controlador permite aos desenvolvedores escrever programas de controle da rede em linguagens de programação de alto nível, de acordo com sua necessidade, obtendo assim, um controle mais refinado sobre o tráfego da rede. Com os estudos na área de SDN e OpenFlow se intensificando cada vez mais, novas pesquisas com controladores vão surgindo. Segundo Shalimov et al (2013), no momento, existem mais de 30 controladores OpenFlow diferentes criados para diversas finalidades, escritos em diferentes linguagens por vendedores, universidades, grupos de pesquisa, entre outros.

O controlador centraliza a comunicação com os elementos programáveis, oferecendo uma visão unificada da rede. A existência da visão global simplifica a representação dos problemas e a tomada de decisão, facilitando melhor a gestão, a segurança e outros recursos. Com essa visão única pode-se também trabalhar de forma distribuída, através da divisão dos elementos de visão ou através de algoritmos distribuídos (Guedes et al, 2012). O controlador não necessita estar localizado em um único ambiente físico, já que essa visão unificada da rede é lógica.

Alguns dos principais controladores que podemos citar são: o POX (POX, 2014), NOX (NOX, 2014), Beacon (Beacon, 2014), FloodLight (Floodlight, 2014), Ryu (Ryu, 2014) e Pyretic (Pyretic, 2014). O controlador POX será utilizado para a realização do presente trabalho. A seguir, uma breve descrição desses controladores.

1. **NOX:** É o primeiro controlador OpenFlow, é de código aberto e tem como principal função o desenvolvimento de controladores eficientes em C++ (NOX, 2014). Ele também possui a versão em Python. O controlador gerencia o conjunto de regras instaladas nos comutadores reagindo a eventos de rede (Guedes et al, 2012). Esses eventos podem ser um fluxo novo chegando, fluxo saindo, etc. Muitos dos trabalhos na área são desenvolvidos com o NOX. Nos trabalhos de Uppal and Brandon (2010) e Korner and Kao (2012), o NOX foi utilizado como controlador para realizar o balanceamento de carga.

2. **Beacon:** Beacon (Beacon, 2014) é um controlador baseado em Java que suporta tanto a operação baseada em eventos quanto em threads. Ele é de código aberto e é multi-plataforma, podendo ser executado em servidores de alto desempenho Linux e em celulares Android. O projeto vem sendo desenvolvido desde 2010 e é considerado estável, segundo seus desenvolvedores.

3. **FloodLight:** FloodLight (Floodlight, 2014) é um controlador de código aberto, de uso comercial, distribuído segundo a licença Apache, baseado totalmente em Java. Foi projetado para trabalhar com um grande número de switches, roteadores, switches virtuais e pontos de acesso que suportam o padrão OpenFlow. Floodlight administra a entrada e saída dos switches OpenFlow, rastreia a localização de hosts finais na rede e fornece suporte para integração com redes não-OpenFlow (Ragalatha et al, 2013).

4. **Ryu:** O controlador Ryu (Ryu, 2014) é escrito na linguagem Python, distribuído segundo a licença Apache 2.0. Ryu é uma estrutura de rede definida por software baseada em componentes. Ele suporta vários protocolos para gerenciamento de dispositivos de rede, como OpenFlow, Netconf, DE-config, etc.

5. **Pyretic:** Pyretic (Pyretic, 2014) é um software de código aberto, baseado em linguagem Python. Ele é utilizado por desenvolvedores comerciais e de pesquisa. Com ele é possível modificar as políticas dinamicamente e construir módulos independentes.

6. **POX:** O POX foi o controlador escolhido para realização do trabalho. Ele é muito utilizado em pesquisa e ensino, por ser de fácil uso e por ser desenvolvido em uma linguagem mais simples, o Python. O POX foi criado com base no modelo NOX com o objetivo de substituir o controlador NOX nos casos em que o desempenho não seja um requisito crítico (Guedes et al, 2012), além de oferecer melhor desempenho do que a versão do NOX desenvolvida em Python.

   As funcionalidades do controlador POX (POX, 2014) são providas por entidades denominadas componentes (ou módulos), com as quais torna-se possível desenvolver o que se deseja. Em outras palavras, componente é o nome dado às aplicações que são programadas para executar sobre o POX. Para iniciar o POX, o módulo pox.py deverá ser executado, recebendo como argumentos a lista de nomes de componentes que serão instanciados durante a execução. Esses módulos também podem possuir argumentos.

   A inicialização do POX é feita através de linha de comando. Para executar, por exemplo, um módulo de switch de aprendizagem, um comando típico seria:

   ./pox.py forwarding.l2_learning

   Pode-se também incluir nessa linha de comando o componente log.level. Este permite que seja configurado qual o nível de detalhe de log que se deseja obter. O código fonte do POX está dividido em um conjunto de diretórios que compreende tanto módulos essenciais para seu funcionamento básico quanto componentes adicionais que oferecem funcionalidades adicionais úteis (Guedes et al, 2012). Ele possui também diversas bibliotecas para o desenvolvimento dos módulos, como por exemplo a biblioteca packet, que acessa detalhes dos pacotes que estão sendo manipulados.

2.4 Switch com suporte OpenFlow

O switch OpenFlow é formado pela tabela de fluxos, o canal seguro e o protocolo de comutação OpenFlow entre o switch e o controlador (McKeown et al, 2008). Este último

é utilizado para a comunicação e troca de mensagens entre o switch e o controlador. O canal seguro (McKeown et al, 2008) é o local por onde são transmitidas as mensagens entre o comutador e o controlador, ele é criptografado e utiliza configurações e procedimentos da conexão do protocolo Secure Socket Layer (SSL). O modelo do switch OpenFlow é mostrado na figura 2.1[1] .

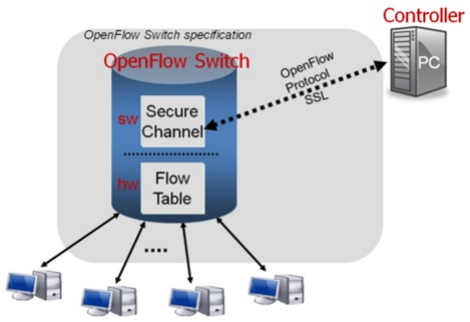

Figura 2.1: Modelo do switch OpenFlow.

O switch OpenFlow funciona como um switch padrão com uma tabela de fluxo, realizando pesquisa de pacotes e encaminhamento, contudo a diferença está na forma como as regras de fluxo são inseridas e atualizadas dentro da tabela de fluxo do switch (Uppal and Brandon, 2010). A união de uma definição de fluxo e um conjunto de ações forma uma entrada da tabela de fluxos OpenFlow (McKeown et al, 2008).

Quando o remetente envia o primeiro pacote de um novo fluxo ao switch OpenFlow, é verificado se o cabeçalho desse pacote tem correspondência na tabela de fluxo do switch. Cada entrada na tabela possui uma ação correspondente. Na figura 2.2[2] é mostrada uma visão geral da entrada da tabela de fluxo, onde possui o cabeçalho que será comparado ao do pacote. Se uma entrada for encontrada, as instruções associadas a esse fluxo são executadas. Se nenhuma correspondência for encontrada na tabela, o pacote será encaminhado para o controlador, que irá processá-lo. O pacote é enviado ao controlador através de um canal seguro (Sezer et al, 2013). O controlador poderá adicionar, atualizar ou excluir entradas na tabela de fluxo. As ações associadas aos fluxos poderão ser: encaminhar o pacote para uma determinada porta de saída (ou portas), modificar os campos do cabeçalho, ou, como medida de segurança, descartar os dados, entre ou-

[1]Disponível em: http://yuba.stanford.edu/cs244wiki/index.php/Overview
[2]Disponível em: https://www.sdncentral.com/resources/sdn/what-is-openflow/

tras. Além das ações, a arquitetura OpenFlow prevê manutenção de três contadores por fluxo: pacotes, bytes trafegados e duração do fluxo. Esses contadores são implementados para cada entrada da tabela de fluxos e podem ser acessados pelo controlador através do protocolo (Guedes et al, 2012).

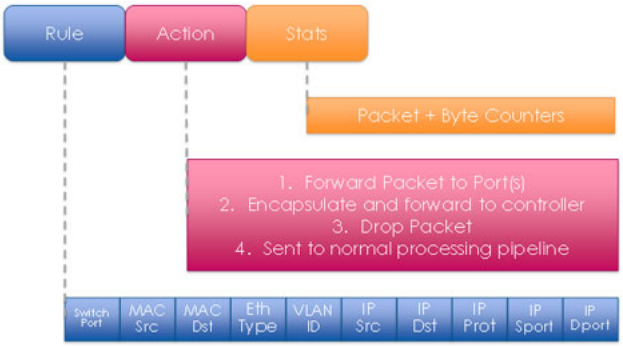

Figura 2.2: Visão geral da entrada da tabela de fluxo OpenFlow 1.0.

3 Balanceamento de carga

O balanceamento de carga se tornou um mecanismo muito importante para uma rede de computadores. Com a inclusão de novas tecnologias, a Internet vem se tornando algo complexo, na qual uma rede deve ser capaz de lidar com milhares de requisições simultaneamente. Para suportar os grandes tráfegos da Internet e manter um tempo de resposta suficientemente rápido para os clientes é necessário que os recursos de uma rede sejam bem utilizados. O ideal é que cada equipamento ou servidor processe uma quantidade de trabalho compatível com sua capacidade. Sendo assim, o balanceamento de carga trouxe uma forma de processar as requisições de forma que todos os recursos possam ser bem aproveitados, maximizando o rendimento. O balanceamento em redes de computadores é uma técnica usada para espalhar a carga de trabalho entre vários links ou computadores (Uppal and Brandon, 2010).

Com balanceamento de carga é possível dividir a quantidade de trabalho que um servidor poderia realizar entre mais servidores. Desta forma, mais tarefas poderão ser realizadas na mesma quantidade de tempo e, em geral, todas as requisições poderão ser respondidas em menos tempo. Além disso, o balanceamento de carga traz vantagens como o aumento da escalabilidade, pois poderá existir flexibilidade em adicionar mais recursos de forma rápida e transparente aos utilizadores finais; o aumento do desempenho através da utilização dos recursos de forma inteligente; e uma melhor disponibilidade, já que se um dos servidores falhar, a carga poderá ser redistribuída aos outros servidores existentes sem comprometer o processamento das requisições.

Os balanceadores de carga especializados já existentes no mercado são caros, executam software personalizado e as opções de políticas oferecidas são rígidas, não sendo possível implementar políticas arbitrárias (Uppal and Brandon, 2010). Com as Redes Definidas por Software pode-se criar um balanceador de carga que apresente total flexibilidade na criação das regras de balanceamento, o que torna o balanceador uma ferramenta programável. Com SDN é possível gerenciar toda a rede através de um controlador centralizado, com o qual poderá se distribuir o tráfego de forma mais eficiente, com um custo

financeiro menor quando comparado a um balanceador comercial e um maior desempenho.

Existem alguns trabalhos relacionados ao tema, nos quais foram construídos balanceadores de carga através de uma arquitetura SDN e o padrão OpenFlow. Esses trabalhos utilizam controladores diversos. Alguns como Uppal and Brandon (2010) e Korner and Kao (2012) adotam o NOX como controlador. Para realizar o balanceamento de carga, algumas soluções utilizam algoritmos conhecidos, como *Round Robin* e Aleatório, e em outras foram desenvolvidos algoritmos próprios, como é o caso de Handigol et al (2009), que criou um algoritmo que otimiza o balanceamento de carga, verificando a realidade da rede. Em Ragalatha et al (2013) foi implementado um balanceador de carga dinâmico, onde a carga de trabalho é calculada e distribuída entre os servidores em tempo de execução e não no início da execução do serviço. A maioria dos trabalhos avaliou a eficácia dos algoritmos e provou a capacidade de desempenho dos protótipos desenvolvidos. Verificou-se que em nenhum destes trabalhos se adotou o POX como controlador.

A proposta do presente trabalho é a criação e avaliação do balanceamento de carga em uma rede SDN com OpenFlow. A rede foi criada para prover serviço *Web*, através de 2 servidores que recebem requisições HTTP do cliente e processam esses pedidos. O cliente envia sua requisição para o servidor *Web* através do endereço IP de serviço. Esse IP representa o endereço de acesso à rede. Todas as requisições para esse endereço são balanceadas, ou seja, na chegada do pedido será escolhido qual o servidor irá executar a tarefa e, logo em seguida, a requisição será repassada para esse servidor. Para que cada servidor receba essas requisições, o controlador cria uma ação para que o switch OpenFlow modifique o cabeçalho dos pacotes, reescrevendo o endereço MAC e IP de destino do pacote para o endereço do servidor. Quando o pacote retorna ao cliente, seu cabeçalho é novamente reescrito. Dessa forma, o cliente não tem a informação de qual servidor processou sua solicitação.

Em resumo, foi criada uma arquitetura SDN utilizando o padrão OpenFlow e o controlador POX. Além disso, foram utilizados três algoritmos para realizar o balanceamento dessas requisições, a saber: *Round Robin*, Aleatório e Baseado na carga. A arquitetura da rede é assim composta:

- 2 Servidores virtuais, IP 192.168.56.3 e 192.168.56.4, com sistema Operacional Linux

Ubuntu Server 12.04. Esses equipamentos rodam o servidor *Web* Apache.

- 1 Switch OpenFlow.
- 1 Controlador POX.
- 1 máquina virtual cliente, IP 192.168.56.5, com sistema operacional Linux Ubuntu 14.04. O cliente envia as requisições para o IP de serviço 192.168.56.6.

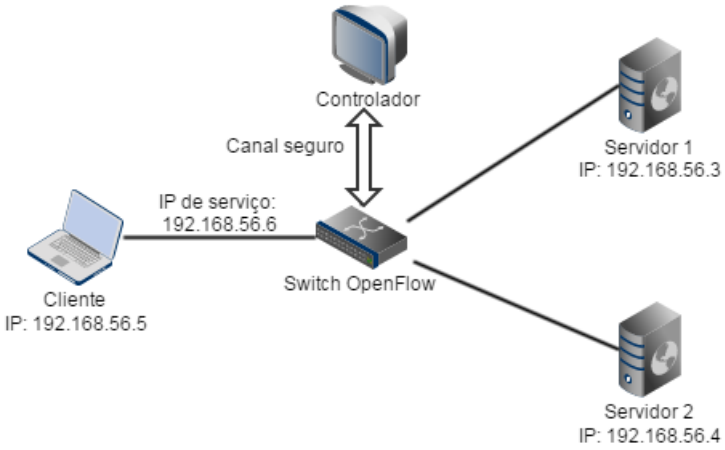

Figura 3.1: Arquitetura da rede proposta

Os algoritmos foram implementados na linguagem Python, específica para o controlador POX. Quando o primeiro fluxo do cliente chega ao switch OpenFlow, ele o encaminha para o controlador que decide qual dentre os dois servidores irá receber a requisição, de acordo com as políticas desenvolvidas. As políticas implementadas para o balanceamento de carga, já anteriormente citadas, são detalhadas a seguir:

1. **Aleatório:** O controlador escolhe aleatoriamente para qual servidor enviar a solicitação.
2. **Round Robin:** Para cada entrada de fluxo, um servidor é escolhido alternativamente para receber a solicitação do cliente. Assim, as requisições são divididas uniformemente entre todos os servidores.
3. **Baseado na carga:** Essa política envia a solicitação para o servidor com a menor carga média de CPU. Essa carga média é calculada pelo sistema operacional através

da média do último minuto de uso da CPU. A carga é obtida no arquivo de sistema que se encontra em /proc/loadavg, de cada servidor Linux. Os servidores enviam as suas cargas para a máquina onde se encontra o controlador, através de comunicação estabelecida via Sockets Python, por uma rede externa à SDN. Essa rede externa é criada através de uma segunda placa de rede contida na máquina física, tornando possível a comunicação com os servidores, externamente à rede SDN. Dessa forma, é possível obter dados dos servidores sem violar o princípio básico de separação dos planos de controle e dados existentes na rede SDN.

O switch OpenFlow utilizado na arquitetura proposta foi criado através do Open vSwitch. Open vSwitch (Pfaff et al, 2009) é uma implementação de software open source de um switch virtual multicamadas com suporte OpenFlow, construído para funcionar em ambientes virtualizados. Ele suporta a criação de VLANS e GRE tunneling, além de fornecer conectividade entre as máquinas virtuais e as interfaces físicas. Através da interface física, é criada uma ponte ou bridge para que possam ser criadas novas interfaces virtuais para a rede. A figura 3.2 mostra como funciona essa virtualização. Na máquina onde está instalado o Open vSwitch é criada a bridge sw1, através da interface física eth0. É a partir dessa bridge que são criadas as interfaces virtuais que se conectarão aos servidores virtuais. Na arquitetura proposta, o Open vSwitch foi instalado em uma máquina física com sistema operacional Linux Ubuntu 14.04. Nessa máquina física está contida toda a arquitetura SDN.

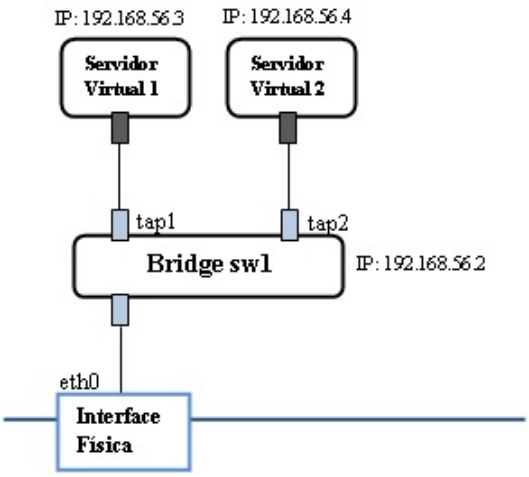

Figura 3.2: Modelo do Open vSwitch

4 Avaliação

4.1 Metodologia

Nessa parte do trabalho, serão apresentadas a contrução do ambiente de simulação, com a criação da arquitetura SDN, a elaboração e realização dos testes e a posterior avaliação dos resultados obtidos.

A arquitetura SDN foi criada através da instalação do Open vSwitch - um switch virtual com suporte OpenFlow -, da configuração das máquinas virtuais e da instalação do controlador POX. Primeiramente, foi projetada uma rede interna que fosse capaz de suportar o número de componentes definidos na arquitetura. Foram alocados IPs fixos para as máquinas e o switch. Em seguida, foi instalado o Open vSwitch em uma máquina física. Vale salientar que é nessa máquina física que está contida toda arquitetura SDN. O Open vSwitch utiliza a placa de rede desta máquina para a criação do switch virtual OpenFlow. No switch são criadas interfaces virtuais que se conectarão às máquinas virtuais.

Com o switch OpenFlow instalado, foi definida a configuração das máquinas virtuais. São três hosts virtuais, dos quais dois são os servidores e o outro é o cliente. Essas máquinas estão em execução dentro da máquina física onde se encontra o switch OpenFlow. Elas são acessadas através do software VirtualBox.

Os servidores possuem sistema operacional Linux Ubuntu Server 12.04, com 1 GB de memória. Em cada servidor foi instalado o servidor *Web* Apache versão 2.2. O Apache, que é o servidor *Web* mais utilizado atualmente, foi escolhido por ser de fácil uso e por atender bem aos requisitos da arquitetura. Não houve necessidade de configurações adicionais no Apache. Já a máquina cliente possui sistema operacional Ubuntu 14.04, com 512 MB de memória. Nela foi instalada a ferramenta que foi responsável por enviar as requisições ou cargas, o Httperf. Mais detalhes do Httperf serão apresentados mais adiante nesta seção.

O controlador POX foi instalado no equipamento onde se encontra o switch Open-

Flow. No controlador foram desenvolvidos os três algoritmos do presente trabalho. Eles representam as políticas para a realização do balanceamento de carga. Foram todos desenvolvidos na linguagem Python, que é a mesma utilizada no POX. O comando padrão para executar cada algoritmo é mostrado a seguir:

./pox.py log.level forwarding.ip_loadbalancer_nomepolítica –ip=192.168.56.6 –servers=192.168.56.3, 192.168.56.4

O campo *ip_loadbalancer_nomepolítica* refere-se ao algoritmo que se deseja utilizar para realizar o balanceamento. Os campos *ip* e *servers* são os argumentos desse algoritmo, onde o *ip* é o IP de serviço da rede e os *servers* são os endereços IP dos servidores. Na forma que foram implementados os códigos desses algoritmos, é possível acrescentar mais servidores como parâmetros para realizar o balanceamento de carga.

Quando o cliente envia uma requisição para o endereço IP de serviço, essa requisição é encaminhada para o switch OpenFlow. O switch verifica o cabeçalho do primeiro pacote pertencente a requisição e se não encontrar uma entrada correspondente na sua tabela de fluxo, o mesmo envia o pacote para o controlador, que decide para qual servidor será enviada essa requisição. O controlador poderá escolher alternativamente entre os servidores através da política de *Round Robin*. Poderá também escolher aleatoriamente com o algoritmo Aleatório ou poderá escolher baseado na menor carga dentre os servidores, segundo o algoritmo Baseado na carga. Feito essa escolha, o controlador instala a regra na tabela de fluxo do switch OpenFlow.

Quando o primeiro pacote chega ao controlador, seu cabeçalho aponta o IP de serviço como endereço de destino. Então, o controlador, após a escolha de qual servidor irá atender a requisição, cria uma ação para que o switch OpenFlow reescreva o campo de destino do cabeçalho com endereços IP e MAC do servidor escolhido, em substituição ao endereço de serviço. Da mesma forma ocorre no retorno do pacote para o cliente, onde há novamente a reescrita do cabeçalho, com a diferença que dessa vez o endereço de origem do pacote passa a ser o endereço de serviço em substituição ao endereço do servidor que recebeu a requisição. A operação de reescrita é transparente para o cliente, que não sabe para qual servidor foi enviada sua requisição e de qual servidor ele obteve resposta.

Para enviar as cargas, a partir da máquina cliente, foi utilizada a ferramenta

Httperf. O Httperf é uma ferramenta que tem como propósito medir o desempenho de um servidor *Web* (Mosberger and Jin , 1998). Com ele é possível gerar diversas cargas de trabalho HTTP, a partir de parâmetros pré-definidos. Ao final da execução do teste, o Httperf gera logs com estatísticas gerais de desempenho. Um comando típico do Httperf é mostrado a seguir.

httperf –server 192.168.56.106 –port 80 –uri /webpage.html –rate 100 –num-conn 1000 –num-call 1 –timeout 60

Cada um dos parâmetros do comando anterior são brevemente explicados a seguir.

- *server*: endereço para o qual a requisição será enviada;
- *port*: porta por onde ocorrerá a comunicação;
- *uri*: arquivo que será carregado a partir do servidor *Web*;
- *rate*: taxa de chegada de requisições no servidor, em requisições/segundo;
- *num-conn*: número total de conexões TCP;
- *num-call*: número de requisições para cada conexão;
- *timeout*: tempo de expiração da requisição.

Os testes foram realizados em três cenários distintos, sendo que em cada um deles foram utilizados os três algoritmos de balanceamento de carga descritos no capítulo 3. O objetivo dos testes é obter os tempos médios de conexão para diferentes taxas de chegada de requisições por segundo, para a posterior avaliação de desempenho dos algoritmos em cada situação. Para cada taxa, o comando Httperf foi executado 10 vezes.

No primeiro cenário foi testado um pequeno arquivo HTML de 10KB, realizando 1000 conexões com 20 requisições cada. As taxas de chegada para uma melhor análise desse cenário foram: 100, 200, 225, 250, 300, 350, 400, 500, 750 e 1000. A seguir um exemplo de um comando do Httperf com taxa 100, para esse cenário. Ele foi executado por 10 vezes, assim como para cada uma das taxas.

httperf –server 192.168.56.106 –port 80 –uri /webpage.html –rate 100 –num-conn 1000 –num-call 20 –timeout 120

No comando mostrado anteriormente, o número de requisições é 20, por se tratar de uma página html, que possui diversos objetos que são carregados na mesma conexão.

No segundo cenário foi testado um arquivo PDF de 1MB, realizando 1000 conexões com 1 requisição cada. As taxas de chegada para uma melhor análise desse cenário foram: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 500 e 1000. A seguir um exemplo de um comando do Httperf com taxa 10, para esse cenário. Ele foi executado por 10 vezes, assim como para cada uma das taxas.

httperf –server 192.168.56.106 –port 80 –uri /file.pdf –rate 10 –num-conn 1000 –num-call 1 –timeout 120

E por fim, no terceiro cenário foi testado um arquivo de vídeo de 100MB, realizando 100 conexões com 1 requisição cada. As taxas de chegada para uma melhor análise desse cenário foram: 1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90 e 100. A seguir um exemplo de um comando do Httperf com taxa 1, para esse cenário. Ele foi executado por 10 vezes, assim como para cada uma das taxas.

httperf –server 192.168.56.106 –port 80 –uri /vídeo.mov –rate 100 –num-conn 100 –num-call 1 –timeout 500

O número de conexões é menor para o cenário com o vídeo, por se tratar de um arquivo grande, podendo dificultar a realização dos testes em termos de tempo e recurso computacional para processar uma grande quantidade de requisições. O timeout é maior que nos outros cenários, pois o tempo de resposta para essa requisição é maior, devido ao tamanho do arquivo.

Cada vez que o comando Httperf é executado, é gerado um arquivo de log, com as estatísticas de desempenho. Como esse comando é executado por 10 vezes para cada taxa, os referentes resultados de cada execução são escritos sequencialmente no mesmo arquivo, para posterior análise. É salvo um arquivo para cada taxa, de cada cenário. A seguir, na figura 4.1, um exemplo de um log típico Httperf, com as estatísticas obtidas.

```
Maximum connect burst length: 1

Total: connections 100 requests 100 replies 100 test-duration 9.902 s

Connection rate: 10.1 conn/s (99.0 ms/conn, <=1 concurrent connections)
Connection time [ms]: min 1.8 avg 2.4 max 18.7 median 2.5 stddev 1.7
Connection time [ms]: connect 0.2
Connection length [replies/conn]: 1.000

Request rate: 10.1 req/s (99.0 ms/req)
Request size [B]: 67.0

Reply rate [replies/s]: min 10.0 avg 10.0 max 10.0 stddev 0.0 (1 samples)
Reply time [ms]: response 1.0 transfer 1.2
Reply size [B]: header 186.0 content 1158.0 footer 0.0 (total 1344.0)
Reply status: 1xx=0 2xx=100 3xx=0 4xx=0 5xx=0

CPU time [s]: user 2.31 system 7.59 (user 23.4% system 76.7% total 100.1%)
Net I/O: 13.9 KB/s (0.1*10^6 bps)

Errors: total 0 client-timo 0 socket-timo 0 connrefused 0 connreset 0
Errors: fd-unavail 0 addrunavail 0 ftab-full 0 other 0
```

Figura 4.1: Log típico de um comando Httperf

Após a realização de todos os testes e a obtenção de todos os arquivos com os logs, foi desenvolvido um script em Python que extrai os dados necessários para a realização de cálculos de interesse. Os campos obtidos dos logs foram o tempo médio de conexão para cada taxa e o número total de erros de conexão. Com esses valores foi possível calcular as médias gerais de tempo de conexão para as 10 execuções, o desvio padrão através dessa média geral e os intervalos de confiança. Esses intervalos de confiança são de 95%, calculados através de distribuição “t”.

Com esses dados calculados, foi possível gerar os gráficos do tempo médio de conexão pela taxa, para os três algoritmos de balanceamento, em cada um dos cenários descritos.

4.2 Resultados

Nesta parte do trabalho serão mostrados os resultados dos testes realizados para cada um dos cenários que foram citados na seção anterior. De modo gráfico, será apresentado o desempenho de cada um dos algoritmos para cada cenário, com o objetivo de comparar esses algoritmos e apontar qual deles obteve o melhor desempenho em cada caso.

Os gráficos apresentam a relação entre o tempo médio de duração das conexões

e o número de requisições enviadas pelo cliente para a rede por segundo (taxa), com respectivo intervalo de confiança para cada ponto nas curvas do gráfico. O propósito de analisar essa relação é verificar o impacto do aumento da taxa sobre o tempo médio de duração das conexões. O valor dessa taxa influencia no tempo de decisão do controlador. Quanto maior o valor da taxa, mais rapidamente essas requisições chegam ao controlador, que decide para qual servidor encaminhá-las. Por exemplo, se há 1000 requisições com uma taxa de 100 requisições por segundo, o controlador demora aproximadamente 10 segundos para tratar todas elas.

4.2.1 Primeiro cenário

No primeiro cenário, os testes foram realizados com um arquivo HTML, onde o cliente envia 1000 conexões com 20 requisições cada. Na figura 4.2, é mostrado o gráfico para essa situação.

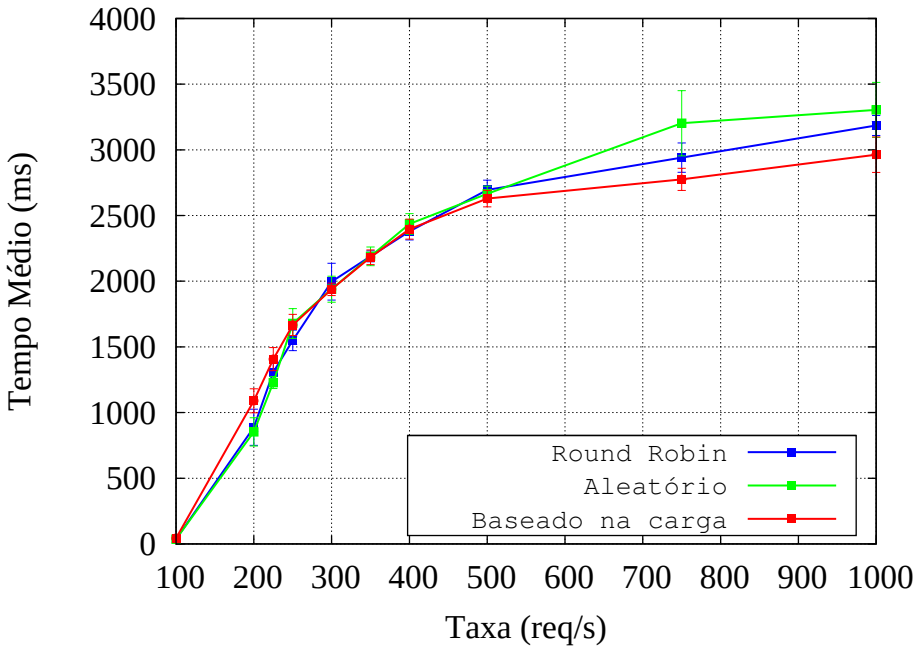

Figura 4.2: Gráfico do tempo médio por taxa para o arquivo HTML

Analisando a figura 4.2, pode-se observar que entre as taxas de 100 a 500, os três algoritmos apresentam comportamentos semelhantes, não sendo possível apontar qual o melhor algoritmo. Isso pode ser confirmado através da interseção dos intervalos de confiança entre os algoritmos, observados no gráfico. Nessa situação, qualquer algoritmo

pode ser escolhido, pois, mesmo havendo muitas requisições, o arquivo possui um tamanho pequeno, e devido a isso, o servidor consegue atender a maior parte dessas requisições.

Pelo fato de haver muitas requisições por conexão, ocorreram alguns erros de conexão com o servidor. Esses erros geralmente ocorrem porque forma-se uma grande fila de requisições nos servidores, e os mesmos, por não conseguirem atender em tempo hábil a todas as requisições, acabam por recusar algumas delas. Mas em um panorama geral são poucos erros, em torno de 1% para as taxas de 200 a 500.

Entre as taxas de 500 a 750, a diferença de desempenho entre os algoritmos aumentou um pouco. Considerando o intervalo de confiança e a taxa de erros, os algoritmos Baseado na carga e *Round Robin* podem ser considerados semelhantes. De acordo com o gráfico, o algoritmo Baseado na carga tende a possuir tempo médio um pouco melhor. Contudo, se forem considerados os erros de conexão, por exemplo na taxa 750, o Baseado na carga obteve mais erros que o *Round Robin*, 1,81% para o primeiro e 0,67% para o último. Então, a leve vantagem do Baseado na carga em relação ao tempo médio é compensado por uma maior porcentagem de erro, o que torna os dois algoritmos, de certa forma, semelhantes no desempenho. Vale ressaltar que, para o caso de 500 a 750, o algoritmo Aleatório tende a ser o pior, pois seu tempo médio e sua taxa de erro são maiores do que o dos outros algoritmos.

Na taxa 1000, o algoritmo Baseado na carga enfrenta o problema de que a grande maioria das requisições vai para o mesmo servidor, devido a escolha rápida do controlador. Como o número total de conexões é igual a 1000 e a taxa de chegada também é de 1000, o controlador leva cerca de um segundo para decidir e encaminhar uma grande quantidade de requisições para um servidor específico. Devido a esse curto espaço de tempo para a escolha do controlador, não houve tempo suficiente para a atualização da carga média do servidor, que não ocorre instantaneamente. Dessa forma, um servidor específico recebe muitas conexões e não consegue tratá-las, o que ocasiona uma taxa maior de erros, em torno de 7,5%, enquanto o algoritmo *Round Robin* possui 0,65% e o Aleatório possui 1,87%. Se o gráfico for observado, a curva do algoritmo Baseado em Carga parece ser a de melhor desempenho, contudo, considerando a taxa de erros, o seu desempenho pode ser comparável ao dos outros algoritmos.

Concluindo, em termos gerais, os algoritmos apresentam desempenhos semelhantes para valores de taxas mais baixos, e considerando as taxas de erro, também possuem desempenhos semelhantes para taxas de chegada mais altas.

4.2.2 Segundo cenário

No segundo cenário, os testes foram realizados com um arquivo PDF, onde o cliente envia 1000 conexões com 1 requisição cada. Na figura 4.3, é mostrado o gráfico para essa situação.

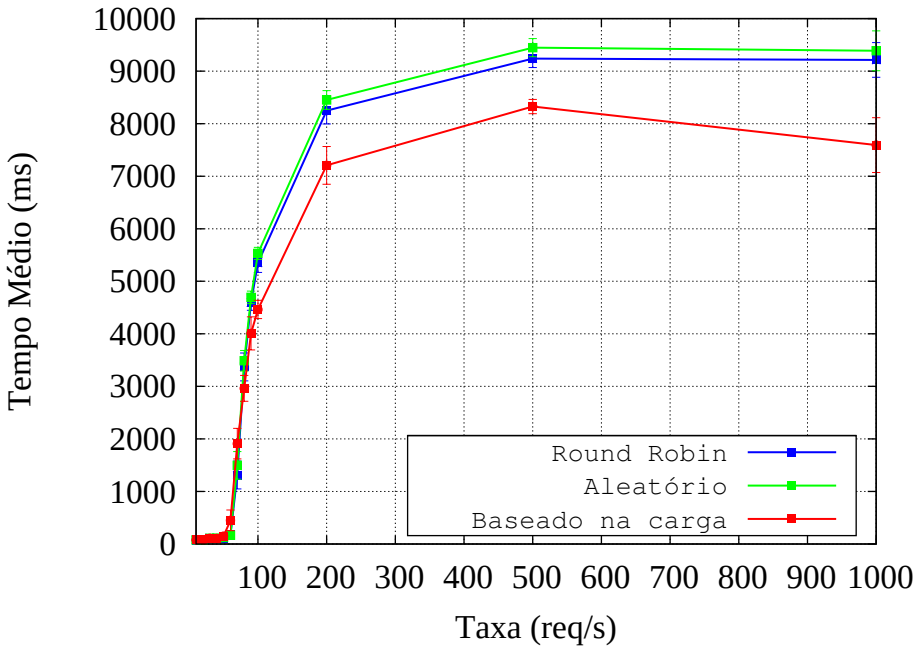

Figura 4.3: Gráfico do tempo médio por taxa para o arquivo PDF

A análise do gráfico da figura 4.3, pode ser dividida em duas. Na primeira parte serão analisadas as taxas que variam entre 10 e 80. Já na segunda parte serão analisadas as taxas que variam de 90 a 1000 requisições por segundo. Para facilitar a visualização, a figura 4.4 mostra um detalhamento das curvas para as menores taxas de chegada.

Observando a figura 4.4 pode-se constatar que entre as taxas de 10 e 50, os algoritmos possuem desempenhos praticamente idênticos, pois as requisições são enviadas mais lentamente para os servidores, e como o arquivo é de tamanho médio, os servidores conseguem atendê-los com tempo de resposta semelhante. Entre 60 e 80, os algoritmos continuam próximos em desempenho, se verificarmos os intervalos de confiança, que estão

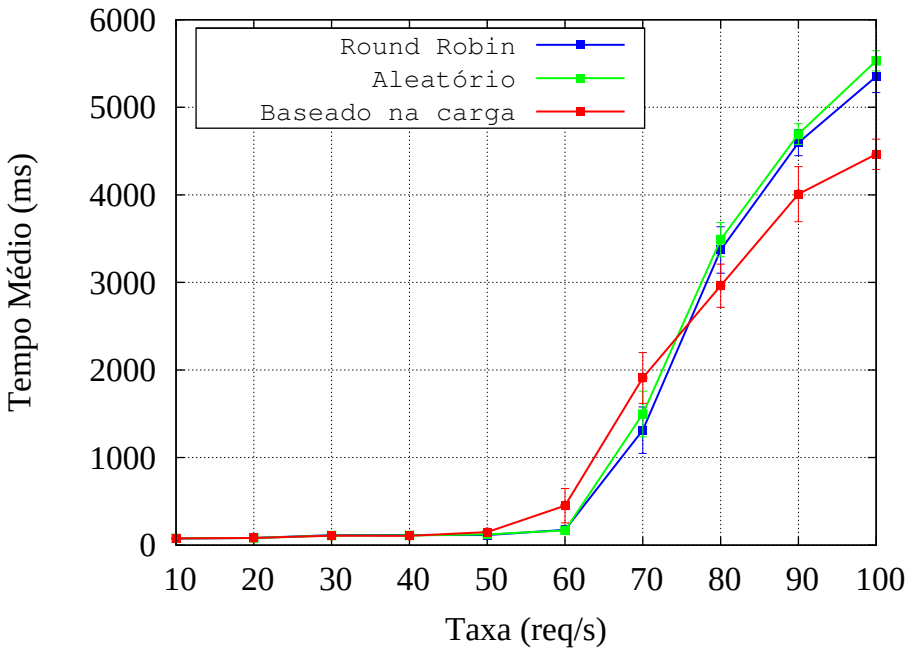

Figura 4.4: Detalhamento das curvas para menores taxas de chegada no arquivo PDF

se interceptando.

Já a partir da taxa 90, o algoritmo Baseado na carga passa a ter o melhor desempenho de todos os algoritmos, inclusive considerando os intervalos de confiança. Da taxa 100 até a taxa 500, pode-se afirmar que o algoritmo Baseado na carga é o melhor algoritmo para esse cenário, sendo aproximadamente 1 segundo mais rápido do que os outros algoritmos em tempo de conexão.

Já para a taxa de 1000 requisições por segundo, não se pode afirmar qual é o melhor algoritmo, pois todos eles apresentaram taxas de erros significativas. As taxas de erros ocorridas nos algoritmos *Round Robin*, Aleatório e Baseado na carga foram de 5,5%, 6,83% e 13,16%, respectivamente. O erro é maior para o algoritmo Baseado na carga pelo mesmo fato do cenário anterior. Devido a escolha rápida do controlador, apenas um servidor recebe a maioria das requisições. Não conseguindo tratá-las em tempo hábil, algumas das requisições são recusadas, o que aumenta a porcentagem de erros quando este algoritmo é comparado com os demais.

Para toda a execução dos testes neste cenário, os algoritmos *Round Robin* e Aleatório apresentaram desempenhos semelhantes.

Concluindo, em geral, o algoritmo Baseado na carga pode ser considerado o me-

lhor algoritmo de balanceamento para este cenário. Vale ressaltar que praticamente não houve nenhum erro de conexão para taxas até 500 requisições por segundo.

4.2.3 Terceiro cenário

No terceiro cenário, os testes foram realizados com um arquivo de vídeo, onde o cliente envia 100 conexões com 1 requisição cada. Na figura 4.5, é mostrado o gráfico para essa situação.

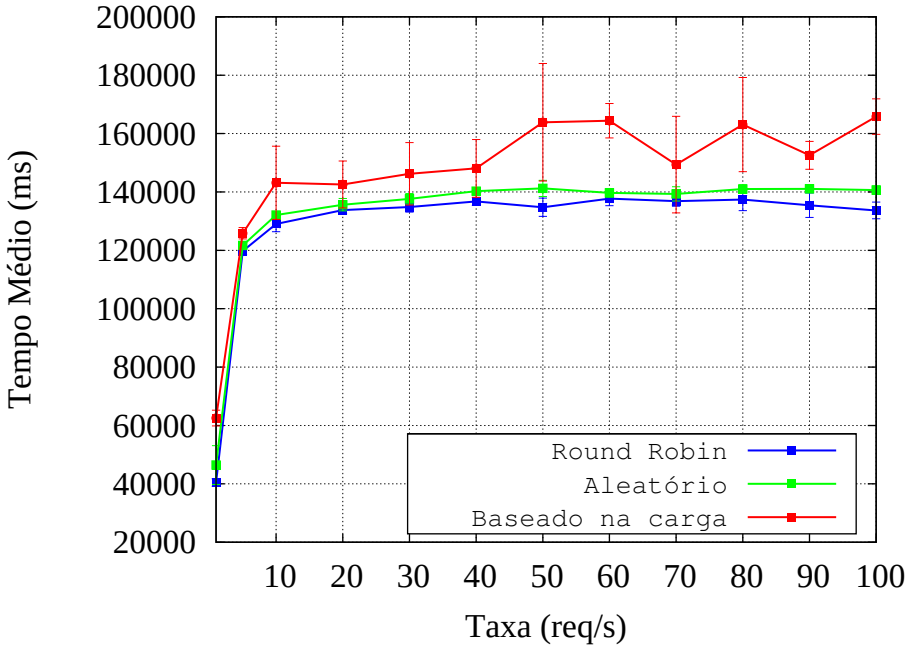

Figura 4.5: Gráfico do tempo médio por taxa para o arquivo de vídeo

Observando o gráfico da figura 4.5, as curvas mostram dois tipos de comportamentos, uma alta taxa de crescimento entre 1 e 10, e valores mais constantes com a taxa variando entre 10 e 100. Pode-se verificar que os algoritmos Aleatório e *Round Robin* possuem comportamentos semelhantes ao longo de suas curvas. Contudo, em casos extremos, o algoritmo de *Round Robin* pode ter um desempenho superior ao do Aleatório, podendo por exemplo, para taxa de 50, existir uma diferença de tempo médio de conexão de aproximadamente 7 segundos entre os dois. Essa diferença pode acontecer pois, dependendo da escolha do algoritmo Aleatório, pode-se sobrecarregar algum servidor em específico, o que irá acarretar um tempo maior de conexão em relação ao *Round Robin*. O último, nesse caso, se mostra um pouco melhor já que divide a carga igualmente para

os servidores.

Já o algoritmo Baseado na carga é nitidamente o pior. Para taxas pequenas (de 1 a 10) pôde-se verificar, durante a realização dos testes, que houve uma má distribuição da carga, pois o valor da carga do servidor demora para ser atualizado, o que não mostra a realidade do estado do servidor no momento, lembrando que a informação de carga de trabalho que o servidor envia para o controlador é a média do último minuto. Para as taxas grandes ocorre o mesmo problema, só que neste caso, o controlador escolhe mais rapidamente qual servidor irá executar a requisição. Cria-se rajadas de requisições ao mesmo servidor, aumentando a fila dele e, por consequência, os tempos de conexão das próprias requisições.

O comportamento da curva do algoritmo Baseado na carga apresenta-se instável a partir da taxa 50, pois depende-se fortemente do desempenho dos servidores. Quanto maior a taxa de chegada, maior a probabilidade de somente 1 servidor receber todas as 100 requisições. Uma possível solução seria a de manter servidores com melhores configurações para suportar uma demanda maior de requisições por arquivos grandes. Vale frisar que neste cenário, o tamanho do arquivo acaba influenciando também no aumento dos tempos de conexão.

Concluindo, deve-se ressaltar que, apesar da sobrecarga dos servidores, não houve registro de erros de conexão. Nesse cenário, pode-se considerar que o *Round Robin* teve um desempenho levemente superior. O algoritmo Baseado na carga não obteve um desempenho muito bom, devido ao fato dele depender fortemente do desempenho do servidor.

5 Trabalhos Relacionados

Em Uppal and Brandon (2010) foi avaliada uma arquitetura de balanceamento de carga alternativa através de um switch OpenFlow. Essa arquitetura de balanceamento de carga consiste em um switch OpenFlow com um controlador NOX e múltiplos servidores conectados às portas de saída do switch. Clientes enviam requisições a esses servidores pela Internet, onde o switch OpenFlow usa uma interface para conectar-se à Internet. Esses servidores, que possuem IP estático, rodam um emulador de servidor *Web*. Esse trabalho descreve três políticas de balanceamento de carga, são elas: Aleatório, Roud Robin e Baseado na carga. São implementadas por módulos em C++, que são executados pelo controlador NOX. Com a definição dessas políticas serão acrescentadas novas regras para a tabela de fluxo do switch. Todos os pedidos do cliente são destinados para o mesmo endereço IP, o módulo executado pelo controlador NOX acrescenta regras para cada fluxo e modifica o endereço MAC e IP de destino do pacote para o endereço do servidor que irá receber as solicitações. Quando o pacote retorna ao cliente, seu cabeçalho é novamente reescrito. Dessa forma, o cliente sempre recebe os pacotes do mesmo endereço IP. Eles avaliaram os componentes da rede e mediram o desempenho global do sistema. Foi comparado o desempenho do switch para pacotes encaminhados normalmente e para aqueles nos quais o cabeçalho é reescrito.

Em Korner and Kao (2012) também utiliza-se o controlador NOX, mas aqui a forma de enxergar a rede é diferente. O NOX é utilizado juntamente com o FlowVisor. Segundo Sherwood et al (2009), FlowVisor é uma aplicação que funciona como uma política baseada em Proxy. Ela cria a possibilidade de dividir os recursos de rede e utilizar um controlador diferente para cada divisão. Nesse trabalho, os testes são feitos dividindo a rede e as funcionalidades através de vários switches. Com essa divisão da rede, atribui-se a cada uma controladores diferentes, dessa forma, um controlador especial poderia ser implementado para cada tipo de tráfego de rede. Com isso, a arquitetura é baseada em vários controladores, no qual existe um controlador para cada rede de serviços. Com a arquitetura distribuída, se houver falha em um dos controladores, poderia-se implementar

facilmente a funcionalidade de que o outro controlador assumisse a gestão de dois ou mais serviços de rede. O FlowVisor, dependendo das informações de cabeçalho do pacote, decide as políticas e para qual divisão e controlador o pacote deverá ser encaminhado. Os testes são feitos em uma rede física, onde são avaliados os elementos da rede e a viabilidade do conceito proposto.

No projeto de Handigol et al (2009) foi criado um próprio balanceador de carga, que eles colocam como o controlador, o Plug-n-Serve, que é baseado no NOX. Ele equilibra as cargas através de redes não estruturadas, tentando minimizar o tempo de resposta médio para as solicitações. Esse controlador possui um algoritmo chamado LOBUS (Load-Balancing over UnStructured networks), no qual otimiza o balanceamento de carga, verificando a realidade da rede. O LOBUS é comparado com os algoritmos tradicionais de balanceamento de carga. Os testes foram realizados fisicamente com servidores *web* e com switches comerciais que possuem o OpenFlow. Eles testaram na rede do prédio de Ciência da Computação da Universidade de Stanford, onde a rede é não estruturada, sendo adicionado mais recursos para verificar o desempenho da resposta às solicitações.

O Plug-n-Serve possui três unidades funcionais: o Flow Manager, que é o módulo controlador, no qual é implementado o algoritmo LOBUS; o Net Manager que é responsável pelo monitoramento da rede e o Host Manager que monitora o estado e a carga dos servidores. No Plug-n-Serve o primeiro pacote de cada solicitação do cliente é interceptado, e é instalada uma regra de encaminhamento no switch para o restante da conexão. Nessa solução pode haver um certa limitação de escalabilidade, já que em cada conexão de um cliente, o controlador será envolvido, podendo tornar mais lenta essa conexão, além de haver muitas regras instaladas no switch. No trabalho Wang et al (2011), que cita o problema relacionado acima, é criado um balanceador de carga que instala regras curingas nos switches, para pedidos diretos de grupos de clientes, sem envolver o controlador. O propósito é determinar globalmente as regras curingas ótimas. O controlador utilizado é o NOX.

No trabalho Zhou et al (2014), o balanceamento de carga foi implementado com o controlador PyResonance. PyResonance é implementado com Pyretic. Utilizam mininet para simular a infra-estrutura de rede. É preservado o modelo de máquina de Estado

Finito para definir uma política de rede, onde estados representam ações diferentes ou comportamentos, e as transições entre estados representam reações a eventos dinâmicos ocorridos. Foram desenvolvidos quatro módulos para o balanceamento de carga, onde há o módulo que faz o monitoramento, o que recebe as mensagens e aciona a mudança de estado, para que outro módulo realize a transição desse estado. No último módulo está implementada a política padrão e a política correspondente de cada estado. As políticas definem a regra de encaminhamento dos fluxos na rede.

No projeto de Ragalatha et al (2013) é utilizado o conceito de balanceamento de carga dinâmico, no qual a carga de trabalho é calculada e distribuída entre os servidores em tempo de execução e não no início da execução do serviço, como ocorre com o balanceamento estático. No projeto proposto, o cliente envia a solicitação ao controlador através da rede sem saber qual o endereço IP do servidor real, esse pedido é enviado para um endereço IP virtual desses servidores e enviado ao controlador, que decide para qual servidor encaminhar o pacote. Isso ocorre no primeiro pedido do cliente. Os servidores registram no controlador e reportam suas cargas atuais, bem como outros atributos, para que se possa simular o desempenho e a utilização dos recursos através de algoritmos de roteamento diferentes. A carga dos servidores e a utilização de diferentes caminhos até esses servidores são monitorados continuamente, pois para realizar o balanceamento de carga é usado a combinação entre a carga do servidor e a utilização do caminho até ele. Esse algoritmo de balanceamento de carga baseia-se nos métodos de ligações mínimas e menos ponderadas, para seleção de um nó. Quando os servidores estão em um ambiente com capacidades semelhantes utiliza-se o primeiro método, se eles possuem capacidades variadas, utilizam a estratégia de menos ponderadas, onde as decisões são tomadas a partir da capacidade ou do peso atribuído ao servidor. O algoritmo criado recebe como entrada a topologia de rede com os pesos dos servidores e devolve como saída o caminho ideal do balanceador de carga, no qual o pedido do cliente é atendido pelo servidor com menor carga e no melhor caminho encontrado. Para simular a rede, utilizam o Mininet e o MiniEdit que é um editor de rede baseado em GUI, onde se pode editar a rede, adicionando os componentes como switches, hosts. É utilizado o controlador FloodLight.

6 Conclusão

A consolidação do paradigma de Redes Definidas por Software vem mudando a forma de gerenciar e projetar as redes de computadores, permitindo que elas possam evoluir e melhorar mais rapidamente em relação ao que ocorre atualmente. Dessa forma, com SDN foi possível criar um ambiente de teste para estudar um tema que tem um importante papel nas redes de computadores: o balanceamento de carga. Portanto, no presente trabalho, foi desenvolvido uma arquitetura SDN com o objetivo de se estudar e avaliar o balanceamento, utilizando diferentes algoritmos em diferentes cenários.

No primeiro cenário, os algoritmos apresentam desempenhos semelhantes para valores de taxas mais baixos, como pode ser confirmado através da interseção dos intervalos de confiança no gráfico. Apesar dos testes terem sido realizados com um arquivo pequeno, houve alguns erros de conexão com o servidor. Para que fosse possível avaliar o desempenho de cada algoritmo, em valores de taxas entre 500 e 1000, foi necessária a inclusão de erros na análise. Entre as taxas de 500 a 750, os algoritmos Baseado na carga e *Round Robin* foram considerados semelhantes, considerando o intervalo de confiança e a taxa de erros, já o Aleatório foi o pior para esse caso, pois seu tempo médio de conexão e seus erros foram maiores que os outros dois algoritmos. Na taxa de chegada 1000, os três apresentaram desempenho semelhantes, considerando novamente as taxas de erros. Neste cenário não foi possível apresentar qual o melhor algoritmo, portanto, qualquer um que for escolhido consegue atender bem a maior parte das requisições.

No segundo cenário, pode-se constatar que para taxas de chegada mais baixas, entre 10 e 50, os algoritmos possuem desempenhos praticamente idênticos; e entre 60 e 80, eles são próximos em desempenho, levando em conta os intervalos de confiança. Entre as taxas de 90 e 500, pode-se afirmar que o melhor algoritmo para o balanceamento é o Baseado na carga, podendo ser, em média, 1 segundo mais rápido em alguns casos. Acredita-se que, neste caso, esse algoritmo é o melhor porque ele está dividindo a carga de forma que os recursos dos servidores sejam bem utilizados. Já para a taxa de 1000 requisições por segundo, não se pode afirmar qual o melhor algoritmo, pois todos apre-

sentam taxas de erro significativas. Em um panorama mais geral, podemos afirmar que, para esse cenário, o algoritmo Baseado na carga pode ser considerado o melhor algoritmo de balanceamento.

No terceiro cenário, pode-se constatar que os algoritmos *Round Robin* e Aleatório possuem comportamentos semelhantes ao longo de suas curvas, com uma leve vantagem de desempenho para o *Round Robin* em alguns casos, podendo chegar, por exemplo, a 7 segundos de diferença entre os dois, na taxa 50. O algoritmo Baseado na carga, por sua vez, foi o que apresentou o pior desempenho. O principal problema do algoritmo, neste caso, é o fato dele depender fortemente do desempenho dos servidores, que ao receberem grande quantidade de requisições, não conseguem atendê-las em um tempo hábil e comparável aos outros algoritmos. Em um panorama geral, pode-se considerar que o algoritmo *Round Robin* teve um desempenho levemente superior. Acredita-se que, para esse caso, o balanceamento igualitário das cargas contribuiu para um melhor desempenho.

Em resumo, os resultados encontrados para os cenários possuem como características em comum semelhanças entre os desempenhos dos algoritmos para taxas mais baixas. O algoritmo *Round Robin* pode ser considerado um algoritmo razoável para ser utilizado em qualquer cenário. Já o algoritmo Baseado na carga mostrou-se ser bom nos dois primeiros cenários, e teve um pior desempenho no terceiro. Para que ele pudesse ser considerado o melhor algoritmo em todos os cenários, acreditamos que algumas modificações em elementos que o influenciam sejam necessárias. Por exemplo, estudar uma melhor forma de obter a carga de trabalho instantânea dos servidores; obter recurso suficiente para fazer um balanceamento de carga com muitas requisições, através da melhoria das configurações dos servidores; e verificar a possibilidade de aumento do número de servidores, se necessário for.

O próximo passo para dar continuidade a essa pesquisa de balanceamento de carga é a criação da arquitetura SDN em uma rede física. Além disso, como propostas de trabalhos futuros, poderão ser criados algoritmos mais dinâmicos, que consigam enxergar melhor a realidade da rede e tomar melhores decisões de balanceamento; como também pode ser pesquisado o balanceamento de carga que vise um melhor aproveitamento dos recursos, economizando energia.

Referências Bibliográficas

Beacon. **What is Beacon?** Disponível em: https://openflow.stanford.edu /display/Beacon/Home. Acessado em: Novembro de 2014.

Casado, M.; Freedman, M. J.; Pettit, J.; Luo, J.; Gude, N.; McKeown, N. ; Shenker, S. **Rethinking enterprise network control**. In: IEEE/ACM Trans. Netw., volume 17, p. 1270–1283, Piscataway, NJ, USA, 2009.

Feamster, N.; Rexford, J. ; Zegura, E. **The road to SDN: an intellectual history of programmable networks**. In: ACM SIGCOMM Computer Communication Review, volume 44, p. 87–98. ACM New York, NY, USA, 2014.

Floodlight. **Floodlight is an open SDN controller**. Disponível em: http://www.projectfloodlight.org/floodlight. Acessado em: Novembro de 2014.

Guedes, D.; Vieira, L. F. M.; Vieira, M. M.; Rodrigues, H. ; Nunes, R. V. **Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores**. In: Minicurso do XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2012, Ouro Preto, Brasil, 2012.

Handigol, N.; Seetharaman, S.; Flajslik, M.; McKeown, N. ; Johari, R. **Plug-n-serve: load-balancing web traffic using openflow**. In: Proceedings of demo at ACM SIGCOMM, 2009.

Korner, M.; Kao, O. **Multiple service load-balancing with openflow**. In: Proceedings of the IEEE 13th Conference on High Performance Switching and Routing. IEEE Publishers, Belgrado, Sérvia, 2012.

Lara, A.; Kolasani, A. ; Ramamurthy, B. **Network innovation using openflow: A survey**. In: Communications Surveys & Tutorials, IEEE, volume 16, p. 493–512, 2014.

Levin, D.; Wundsam, A.; Heller, B.; Handigol, N. ; Feldmann, A. **Logically centralized? state distribution trade-offs in software defined networks**. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, p. 1–6, New York, NY, USA, 2012.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. ; Turner, J. **Openflow: Enabling innovation in campus networks**. In: SIGCOMM Comput. Commun. Rev., volume 38, p. 69–74, New York, NY, USA, 2008.

Mosberger, D.; Jin, T. **httperf: A tool for measuring web server performance**. In: Proceedings of the 1998 Internet Server Performance Workshop, volume 26, p. 31–37, 1998.

NOX. **About nox**. Disponível em: http://www.noxrepo.org/nox/about-nox/. Acessado em: Novembro de 2014.

ONF. **What is onf?** Disponível em: https://www.opennetworking.org /images/stories/downloads/about/onf-what-why.pdf. Acessado em: Novembro de 2014.

POX. **Pox wiki**. Disponível em: https://openflow.stanford.edu/display/ONL/POX+Wiki. Acessado em: Novembro de 2014.

Pfaff, B.; Pettit, J.; Koponen, T.; Amidon, K.; Casado, M. ; Shenker, S. **Extending networking into the virtualization layer**. In: 8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII). New York, NY, USA, 2009.

Pyretic. **Pyretic**. Disponível em: http://www.frenetic-lang.org/pyretic/. Acessado em: Novembro de 2014.

Ragalatha, P.; Challa, M. ; K, S. K. **Design and implementation of dynamic load balancer on openflow enabled sdns**. In: Research Paper in International Organization of Scientific Research Journal of Engineering (IOSRJEN), ISSN: 2250-3021, volume 3, Issue 8, p. 32–41, 2013.

Ryu. **What's Ryu?** Disponível em: http://osrg.github.io/ryu/. Acessado em: Novembro de 2014.

Sezer, S.; Scott-Hayward, S.; Chouhan, P.-K.; Fraser, B.; Lake, D.; Finnegan, J.; Viljoen, N.; Miller, M. ; Rao, N. **Are we ready for SDN? Implementation challenges for software-defined networks**. In: IEEE Communications Magazine, volume 51, p. 36–43, 2013.

Shalimov, A.; Zuikov, D.; Zimarina, D.; Pashkov, V. ; Smeliansky, R. **Advanced study of SDN/Openflow controllers**. In: Proceedings of the 9th Central Eastern European Software Engineering Conference in Russia. ACM, 2013.

Sherwood, R.; Gibb, G.; Yap, K.-K.; Appenzeller, G.; Casado, M.; McKeown, N. ; Parulkar, G. **Flowvisor: A network virtualization layer**. In: Technical Report Openflow-tr-2009-1, OpenFlow. Stanford University, 2009.

Uppal, H.; Brandon, D. **Openflow based load balancing**. In: Proceedings of CSE561: Networking Project Report. University of Washington, Spring, 2010.

Wang, R.; Butnariu, D. ; Rexford, J. **Openflow-based server load balancing gone wild**. In: Hot-ICE'11 Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services. USENIX Association Berkeley, CA, USA, 2011.

Zhou, Y.; Ruan, L.; Xiao, L. ; Liu, R. **A method for load balancing based on software defined network**. In: Advanced Science and Technology Letters, volume 45, p. 43–48, 2014.