Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Leonardo de Carvalho Freitas Padilha Aguilar

Análise do uso de SDN para balanceamento de carga

São Paulo
Dezembro de 2018

Análise do uso de SDN para balanceamento de carga

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Supervisor: Prof. Dr. Daniel Macêdo Batista

São Paulo
Dezembro de 2018

Resumo

Rede definida por software (ou *software defined networking*, SDN) é um recurso recente que permite a criação, controle e personalização da rede com o uso de software, diferentemente do modelo tradicional onde a rede era fixa e definida por *hardware*. Essa mudança de paradigma é realizada com a existência de uma entidade central na rede, chamada de controlador, que é responsável por enviar para os *switches* as regras de repasse, modificação e/ou bloqueio de fluxos de pacotes. O objetivo desse trabalho é analisar o uso de SDN para balanceamento de carga através do desenvolvimento de um balanceador que possa executar três algoritmos diferentes, dando ao usuário a possibilidade de escolher e alterar (em tempo de execução) qual será utilizado, bem como seus parâmetros.
Palavras-chave: SDN, balanceador de carga, algoritmos de balanceamento.

Abstract

Software defined networking (SDN) is a recent feature that allows the creation, control and customization of the network through the use of software, unlike the traditional model where the network was fixed and defined by hardware. This paradigm change is accomplished by the existence of a central entity in the network, called controller, which is responsible for sending the rules of forwarding, modifying and/or dropping of packet flows to the switches. The purpose of this work is to analyze the use of SDN for load balancing by developing a balancer that can execute three different algorithms, giving to the user the possibility to choose and change (at run time) which will be used, as well as their parameters.
Keywords: SDN, load balancer, load balancing algorithms.

Conteúdo

Lista de Abreviaturas

ADC	Controlador de Fornecimento de Aplicativos (*Application Delivery Controller*)
ADN	Rede de Entrega de Aplicação (*Application Delivery Network*)
API	Interface de Programação de Aplicativos (*Application Programming Interface*)
ARP	Protocolo de Resolução de Endereços (*Address Resolution Protocol*)
CPU	Unidade Central de Processamento (*Central Processing Unit*)
DDoS	Negação de Serviço Distribuído (*Distributed Denial of Service*)
DNS	Sistema de Nome de Domínios (*Domain Name System*)
HLD	Dispositivo de Balanceamento de Carga (*Hardware Load Balancer Device*)
HTML	Linguagem de Marcação de Hipertexto (*Hypertext Markup Language*)
HTTP	Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*)
IP	Protocolo Internet (*Internet Protocol*)
OF	OpenFlow
OSI	Interconexão de Sistemas Abertos (*Open System Interconnection*)
PHP	PHP: Preprocessador Hipertexto (*PHP: Hypertext Preprocessor*)
QoE	Qualidade de Experiência (*Quality of Experience*)
RAM	Memória de Acesso Aleatório (*Random Access Memory*)
SDN	Rede Definida por Software (*Software defined networking*)
SSL	Camada Segura de *Sockets* (*Secure Sockets Layer*)
TCP	Protocolo de Controle de Transmissão (*Transmission Control Protocol*)
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)

Lista de Códigos

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Contextualização

O balanceamento de carga em redes de computadores, conforme Moharana (2013), é uma técnica importante que busca garantir a otimização dos recursos existentes para uma melhor distribuição dos pacotes de dados, evitando a sobrecarga dos equipamentos e diminuindo o tempo de resposta das aplicações. A técnica se baseia em escolher o servidor mais adequado para responder a uma determinada requisição, podendo ser realizado de diversas maneiras, sendo atualmente uma das mais comuns, segundo Gandhi *et al.* (2014), a implementação de um *hardware* específico que tem como objetivo reencaminhar o pacote de dados para o servidor em "melhor condição"(essa seleção é definida por um algoritmo de distribuição que pode variar de dispositivo para dispositivo). Porém, essa implementação pode causar diversos tipos de limitações, como a falta de flexibilidade quanto à escolha do algoritmo ou a quantidade de servidores suportados na rede, além do custo para a aplicação dela ser bastante elevada, podendo chegar à casa dos U$ 2000[1].

Entretanto, com a crescente disseminação das redes definidas por software (ou SDN, do inglês *software defined networking*), o estabelecimento do comportamento e do tráfego de dados dentro da rede não precisa depender apenas do *hardware*, de acordo com Esteve Rothenberg *et al.* (2018). Isso garante uma maior adaptação e customização que variam de acordo com as condições da rede (por exemplo os servidores disponíveis, meta-dados do pacote, etc.) diferente do modelo rígido anterior. A utilização dessa técnica recente se dá em diversos aspectos, podendo ir desde o âmbito acadêmico, para o desenvolvimento e testes de novos protocolos de rede, como também no âmbito comercial, para diminuição dos custos e maior controle sobre a rede.

Com base nisso, é possível supor que a aplicação da técnica de SDN possa ser uma alternativa mais flexível e com custo de implementação menor quando comparado ao uso de *hardware* para balanceamento de carga.

1.2 Objetivos e contribuição

O objetivo desse trabalho será analisar o desempenho (com enfoque no tempo de resposta) e o custo do uso de SDN e o de *hardware* para o balanceamento de carga.

Para realizar essa análise, pretende-se desenvolver uma aplicação de SDN que contenha

[1]Foi escolhido como parâmetro de preço o ADC de entrada da Barracuda, modelo Barracuda Load Balancer ADC 240. Disponível em <http://www.zones.com/site/product/index.html?id=100713342>. Acessado em 22/04/2018

três algoritmos diferentes para o balanceamento e permita ao administrador da rede escolher qual deverá ser executado, bem como os seus devidos parâmetros. Isso permitirá um maior controle de todo o processo de balanceamento ao usuário, algo que não é permitido com os dispositivos físicos comumente utilizados para essa tarefa.

Como contribuição, a aplicação para arquitetura SDN que implementa o balanceamento de carga será disponibilizada como software livre para que outras pessoas possam usá-la em estudos sobre o assunto e eventualmente melhorá-la adicionando novas funcionalidades.

1.3 Estruturação da monografia

No Capítulo 2, serão apresentados os conceitos base que envolvem esse trabalho, ou seja, o balanceamento de carga, as redes definidas por software e a emulação de rede, que será usada para a execução da análise de desempenho.

O Capítulo 3 é focado no balanceamento de carga utilizando *hardware*, expondo como é executada essa tarefa, além de listar os diferentes tipos de dispositivos. Por fim, será apresentado um desempenho geral e o custo de aplicação dos mesmos.

No Capítulo 4 é explicado como foi desenvolvido o balanceador de carga utilizando a técnica de SDN, apresentando também a aplicação do balanceador e os algoritmos de balanceamento que foram inseridos.

O Capítulo 5 é direcionado à execução e resultados obtidos da análise de desempenho do balanceador usando SDN.

Por fim, o Capítulo 6 apresentará as considerações finais acerca do trabalho desenvolvido e pontuará algumas formas de aprofundar o tema de pesquisa trabalhado nessa monografia.

Capítulo 2

Fundamentação teórica

2.1 Balanceamento de carga

De acordo com Moharana (2013), balanceamento de carga é um mecanismo muito utilizado para melhor aproveitar todos os recursos disponíveis, otimizando o uso dos mesmos e o tempo de execução das tarefas. Sua aplicação se dá tanto para distribuir a carga de trabalho entre recursos específicos de um computador, como discos rígidos, quanto para servidores de rede em geral, a fim de garantir que a melhor máquina (ou seja, aquela com maior disponibilidade) atenda a determinada requisição.

Com a popularização da internet de alta velocidade, a demanda para uma melhor qualidade de resposta para os serviços disponíveis aumentou drasticamente e, para que isso fosse possível, as aplicações começaram a não mais investir em melhorar o seu único servidor (aumentando a memória disponível, por exemplo) mas sim em adicionar novas máquinas, dando origem, assim, ao conceito de fazenda de servidores (do inglês *server-farm*), ou seja, vários computadores com uma só tarefa: atender as requisições dos clientes para uma dada aplicação, proporcionando uma alternativa mais econômica quando comparado ao investimento em melhoria de um único servidor.

Nesse sentido, o balanceamento de carga é uma técnica importante pois é a responsável por levar essa requisição até um dos servidores correspondentes, podendo, assim, utilizar toda a capacidade de processamento disponível.

Vários métodos podem ser utilizados para garantir esse balanceamento, como será exposto nos tópicos seguintes.

2.1.1 No lado do cliente

Para esse tipo de balanceamento, entrega-se uma lista com os endereços IP[1] disponíveis para o cliente e ele se encarrega de selecionar o servidor com o qual irá se comunicar. Isso traz diversas implicações de segurança pois o cliente terá acesso direto aos servidores disponíveis, além de problemas de otimização, pois a aplicação não consegue garantir que o cliente sempre escolherá o melhor servidor naquele instante. Por exemplo, se grande parte dos clientes decidirem fazer o pedido para o servidor número 1, isso irá gerar sobrecarga no mesmo e, consequentemente, diminuir o tempo de resposta.

[1] Dispositivos conectados na Internet são identificados na camada de rede pelo seu endereço IP.

2.1.2 Utilizando DNS

Nesse método, faz-se uso do Sistema de Nomes de Domínios (ou DNS, do inglês *Domain Name System*), associando a um domínio a lista de endereços IP que estão disponíveis na fazenda de servidores. O DNS, então, seleciona o próximo servidor utilizando o algoritmo *round-robin* (associa-se os endereços a uma fila e quando houver requisição, utiliza-se aquele que está no começo da fila, removendo-o e reposicionando no final). Um exemplo de como funciona a topologia para esse tipo de balanceamento pode ser visto na Figura 2.1. O cliente requisita ao servidor DNS a resolução do domínio da aplicação (Fluxo 1 na Figura 2.1) e esse é responsável por escolher um dos servidores disponíveis, com o qual o cliente se comunicará e enviará os dados requisitados (Fluxo 2 na Figura 2.1).

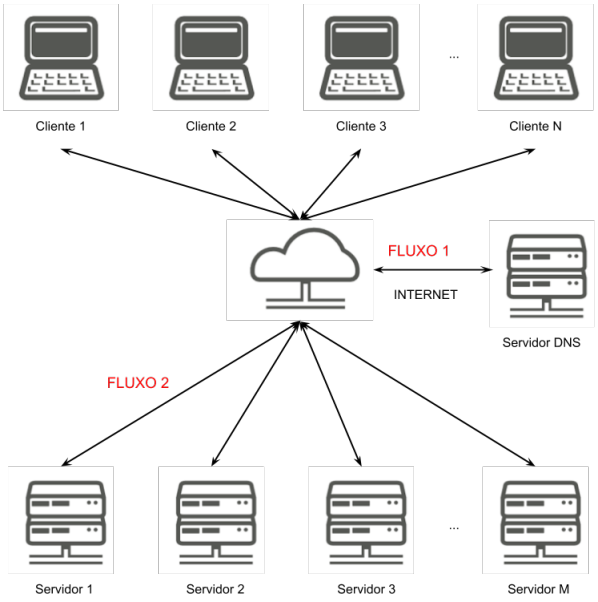

Figura 2.1: *Funcionamento de um balanceador de carga utilizando DNS. O cliente requisita ao servidor DNS a resolução do domínio da aplicação (Fluxo 1). O servidor DNS, portanto, escolhe um dos servidores disponíveis, com o qual o cliente se comunicará e enviará os dados requisitados (Fluxo 2).*

2.1.3 Utilizando balanceadores de carga nos servidores

Esse é o método mais utilizado atualmente. A aplicação possui um *hardware* (o balanceador) que será responsável por escutar os clientes e, quando uma requisição chegar, é de responsabilidade dele redirecionar para um dos servidores da fazenda.

Esse balanceador é o ponto único de contato dos clientes com a aplicação (veja Figura 2.2), garantindo, então que os clientes não saibam da divisão de funcionalidade dos servidores e não consigam contactá-los diretamente, o que é uma boa solução em termos de segurança.

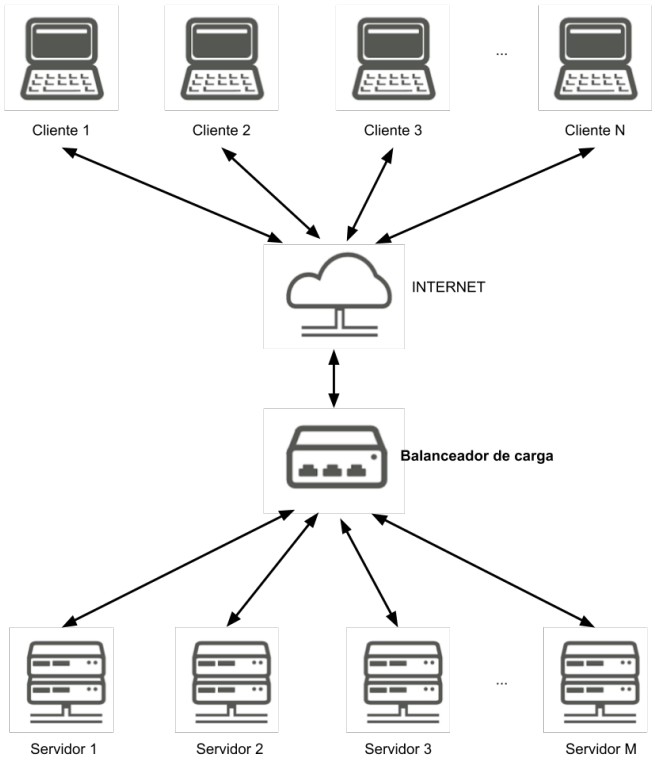

Figura 2.2: *Funcionamento de um balanceador de carga no lado do servidor. O cliente irá se comunicar diretamente com o balanceador de carga e este será responsável por direcionar os dados para um dos servidores da aplicação.*

2.2 Algoritmos de balanceamento

Para garantir uma melhor qualidade na divisão de carga entre os servidores, é muito comum que os balanceadores utilizem-se de algoritmos que vão além da simples escolha aleatória para selecionar o próximo servidor que irá atender a próxima requisição.

Não existe uma técnica que pode ser considerada preferível, pois cada algoritmo possui sua peculiaridade, podendo agir melhor em cenários diferentes. Por esse motivo, alguns fabricantes chegam a utilizar em seus dispositivos uma combinação de mais de um algoritmo, aumentando os cenários nos quais a distribuição é a mais eficiente.

Serão apresentados, a seguir, os três algoritmos que serão tratados ao longo do trabalho.

2.2.1 Aleatório

Uma das técnicas mais simples de balancear carga consiste em escolher um dos servidores aleatoriamente para cada requisição do cliente.

O algoritmo aleatório pode permitir uma boa distribuição de carga para servidores com mesmas características pois ela garante que os servidores recebam requisições de maneira equiprovável (desde que o gerador de números aleatórios utilizado siga uma distribuição uniforme).

2.2.2 *Round-robin*

Como acontece no balanceamento utilizando DNS, que foi visto na seção anterior, o algoritmo *round-robin* faz uso de uma fila de endereços IP e, sempre que um cliente faz uma requisição, seleciona-se o servidor cujo endereço está no início da fila, removendo o mesmo dessa posição e inserindo no fim. A Figura 2.3 exemplifica um cenário com a utilização do algoritmo *round-robin*. Nesse exemplo, pode-se ver um balanceador com três servidores diferentes. A primeira requisição chega e é enviada para o primeiro servidor (de IP 10.0.0.1), a segunda para o segundo servidor (IP 10.0.0.2) e a terceira para o último servidor (IP 10.0.0.3). A próxima requisição volta, então, a ser enviada para o servidor de IP 10.0.0.1, pois agora ele é o primeiro da fila.

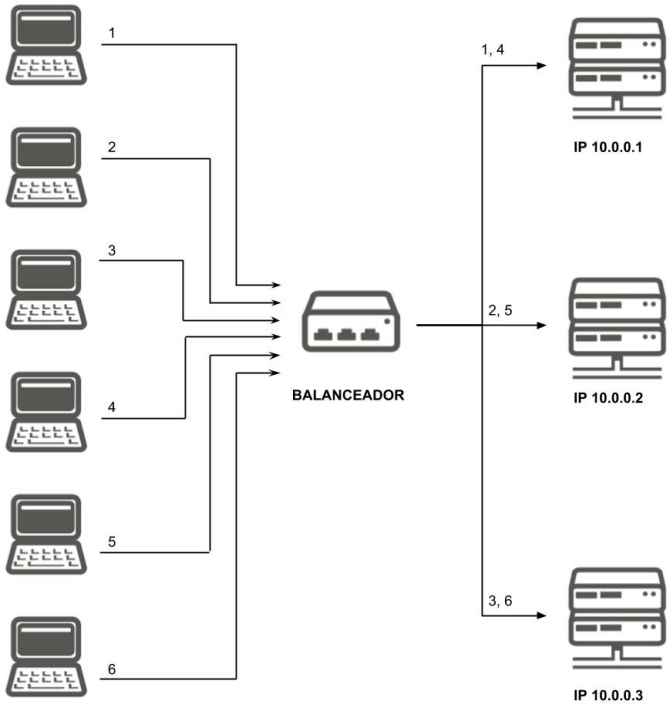

Figura 2.3: *Exemplo do algoritmo* round-robin *para balanceamento de carga. A primeira requisição é enviada para o primeiro servidor (de IP 10.0.0.1), a segunda para o segundo servidor (IP 10.0.0.2) e a terceira para o último servidor (IP 10.0.0.3). A próxima requisição volta, então, a ser enviada para o servidor de IP 10.0.0.1, pois agora ele é o primeiro da fila.*

Essa técnica é interessante para fazenda de servidores com computadores que possuem, basicamente, as mesmas especificações (por exemplo a mesma quantidade de memória RAM, mesma CPU, etc.) pois ela distribui a carga de forma equivalente, ignorando essas características.

Para casos em que os servidores possuem características diferentes, pode-se adicionar pesos aos servidores e, dessa forma, máquinas com pesos maiores receberão mais carga do que máquinas com pesos menores, permitindo, assim, que máquinas com mais capacidade de atendimento recebam mais requisições do que as outras. A Figura 2.4 exemplifica um cenário com a utilização do algoritmo *round-robin* com pesos. Nele, pode-se ver que o primeiro servidor possui peso 3 (logo, recebe as três primeiras requisições), o segundo possui peso 1 (recebendo a quarta requisição) e o terceiro peso 2 (recebendo as últimas 2 requisições).

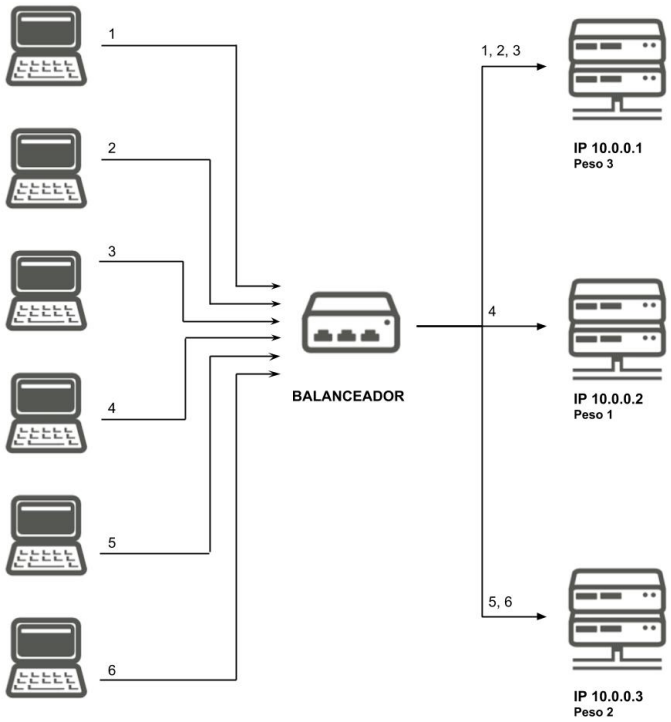

Figura 2.4: *Exemplo do algoritmo* round-robin *com pesos. O primeiro servidor possui peso 3 (logo, recebe as três primeiras requisições), o segundo possui peso 1 (recebendo a quarta requisição) e o terceiro peso 2 (recebendo as últimas 2 requisições).*

2.2.3 *Least-bandwidth*

O algoritmo de *least-bandwidth* (menor largura de banda, em tradução livre) seleciona o servidor com menor consumo de tráfego de rede para atender a próxima requisição. O balanceador analisa e guarda o número de bytes que são transmitidos para cada um dos servidores e, com isso, consegue determinar a próxima máquina (aquela que transmitiu menos dados até então). A Figura 2.5 exemplifica um cenário com a utilização do algoritmo *least-bandwidth*, os servidores possuem tráfego inicial de 4 MB para o servidor de IP 10.0.0.1, 2 MB para o servidor de IP 10.0.0.2 e 1 MB para o servidor 10.0.0.3. Quando a requisição é levada para algum servidor, é adicionado no tráfego total daquela máquina a quantidade de bytes relacionadas àquele pacote.

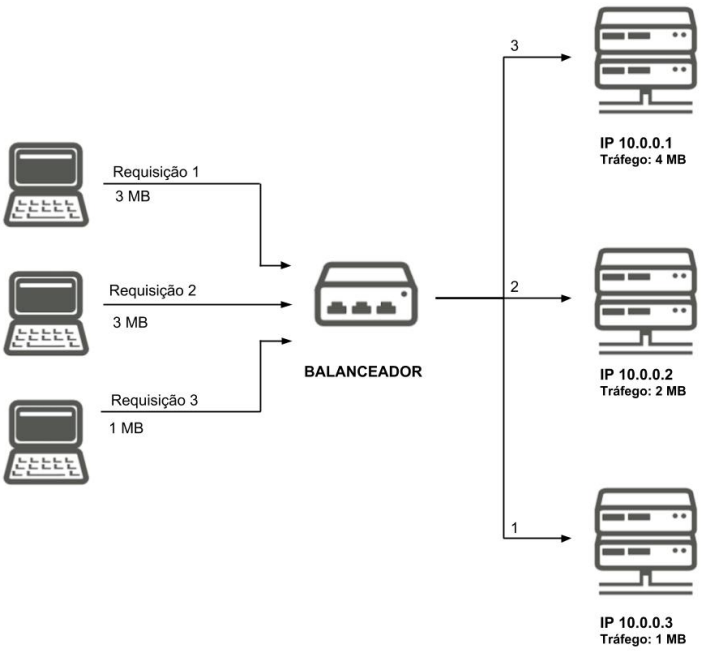

Figura 2.5: *Exemplo do algoritmo* least-bandwidth.

Assim como o método *round-robin*, o *least-bandwidth* também pode aceitar pesos nos servidores. Sendo w_i o peso do servidor i e b_i o consumo de tráfego de rede do mesmo, seleciona-se então a máquina com a menor razão $\frac{b_i}{w_i}$ para atender a próxima requisição, dessa forma, máquinas com pesos maiores recebem mais carga do que máquinas com pesos menores.

2.3 Redes definidas por software

As redes definidas por software (SDN, do inglês *Software Defined Networking*) são um conceito de desacoplação do plano de dados (a camada que transporta os dados do usuário) e do plano de controle (a camada responsável por encaminhar os dados) das redes tradicionais de tal forma que se consiga definir o encaminhamento dos dados utilizando software.

Durante anos a arquitetura das redes era definida apenas através de dispositivos físicos onde estes planos eram indivisíveis, ou seja, o encaminhamento de pacotes dos roteadores e *switches*[2] era gerenciado por uma camada de controle criada pelo fabricante, sem a possibilidade de alteração pelos administradores de rede. Esse plano de controle estava embutido no próprio equipamento. Assim, uma rede com n equipamentos de interconexão poderia possuir n diferentes regras de planos de controle, uma para cada dispositivo, não necessariamente compatíveis entre si.

Com a computação cada vez mais dinâmica, surgiu a necessidade de separar essas duas camadas, dando mais poder ao administrador e permitindo que a rede fique mais adaptável e centralizada no plano de controle.

[2] O *switch* é um dispositivo de rede capaz de conectar computadores, permitindo a comunicação entre eles.

Com SDN, então, permite-se que a definição dos encaminhamentos fique em um servidor dedicado com alta capacidade de processamento e este é o responsável por adicionar regras de encaminhamento de pacotes em cada dispositivo de rede (como permitir o fluxo de determinado endereço IP, bloquear pacotes de determinada porta, etc.) . Isso tem grande valor não só na área de pesquisa, onde consegue-se utilizar SDN como ferramenta para fazer testes de novos protocolos sem que o fabricante exponha o funcionamento interno dos equipamentos (Duque *et al.*, 2012), mas também no âmbito comercial, diminuindo a complexidade da rede interna das aplicações.

A arquitetura de SDN divide as redes em três camadas distintas, sobrepostas conceitualmente uma acima da outra, que possuem tarefas específicas e que se comunicam apenas com sua camada adjacente, como pode-se ver na Figura 2.6.

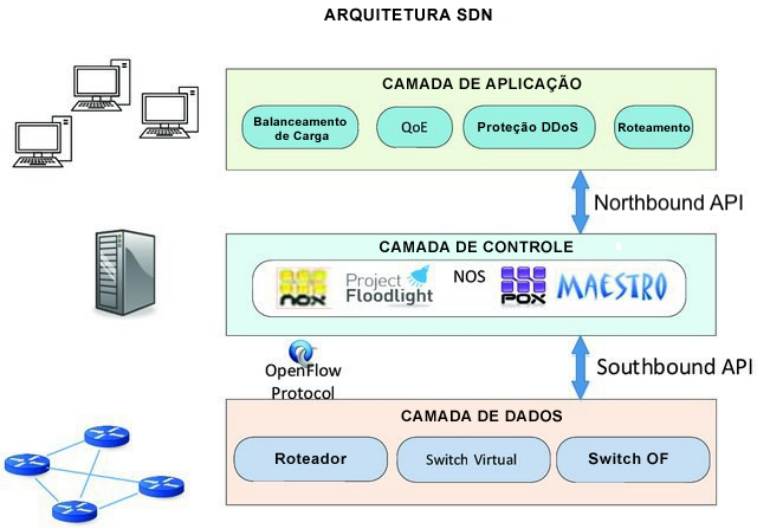

Figura 2.6: *Arquitetura SDN.(Fernández et al., 2018, p.5, tradução nossa)*

A primeira camada, que fica no nível mais acima, é a camada de **aplicação**. As aplicações são softwares que conseguem se comunicar com a camada mais abaixo com o objetivo de definir o comportamento dos fluxos da rede. O balanceador a ser definido ao longo desse trabalho irá atuar nessa camada.

A camada intermediária é a chamada camada de controle (comumente referida como o **controlador** da SDN). Ela é caracterizada por se comunicar tanto com o nível abaixo quanto com o nível acima, proporcionando à camada superior uma abstração dos dispositivos de encaminhamento e permitindo que os fluxos deles sejam modificados por ela.

Já a camada mais abaixo é a camada de **dados** (ou camada dos dispositivos de rede), que tem como função apenas receber os pacotes de dados, realizar alguma ação com eles e atualizar seus contadores internos (estatísticas gerais como largura de banda utilizada por determinado fluxo).

A comunicação entre a camada de aplicação e o controlador (ou *Northbound API*), segundo a Open Network Foundation (s/d), normalmente fornece visões abstratas da rede e permitem a expressão direta do comportamento e dos requisitos de rede. Já a comunicação entre as camadas de mais baixo nível com o controlador (também conhecida como *Southbound API*) é feita através de um protocolo como o **OpenFlow**, que permite o controle dos fluxos e da tabela de fluxo dos dispositivos.

2.3.1 Arquitetura OpenFlow

A arquitetura OpenFlow é uma das mais difundidas arquiteturas focadas em redes definidas por software. Ela define uma série de padrões que permitem o controle das chamadas **tabelas de fluxos** de cada dispositivo de encaminhamento.

As tabelas de fluxos são responsáveis por guardar as informações do fluxo da rede, ou seja, são tabelas cujas entradas definem informações como regras de encaminhamento (algum valor específico do cabeçalho do pacote), ação (o que fazer caso a regra seja atendida) e estatísticas gerais sobre o fluxo, como na Figura 2.7.

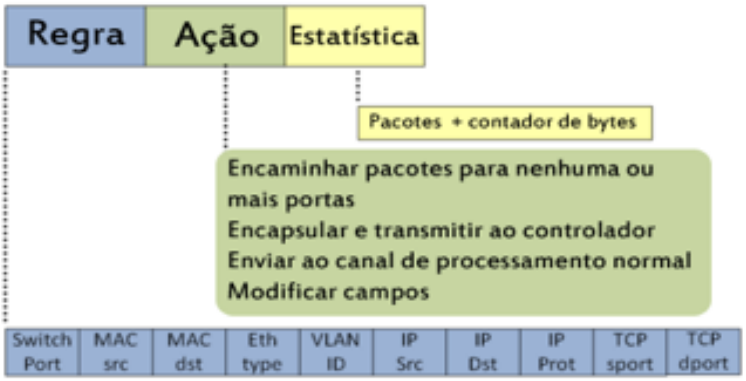

Figura 2.7: *Representação de um fluxo em uma tabela de fluxo. (Duque et al., 2012)*

Além disso, a arquitetura OpenFlow também conta com um dispositivo chamado **controlador** (que é, como visto anteriormente, o responsável por permitir uma abstração dos fluxos às aplicações de rede), além de um canal seguro onde ocorre a comunicação entre o controlador e os dispositivos e um protocolo bem definido para essa comunicação, o protocolo OpenFlow.

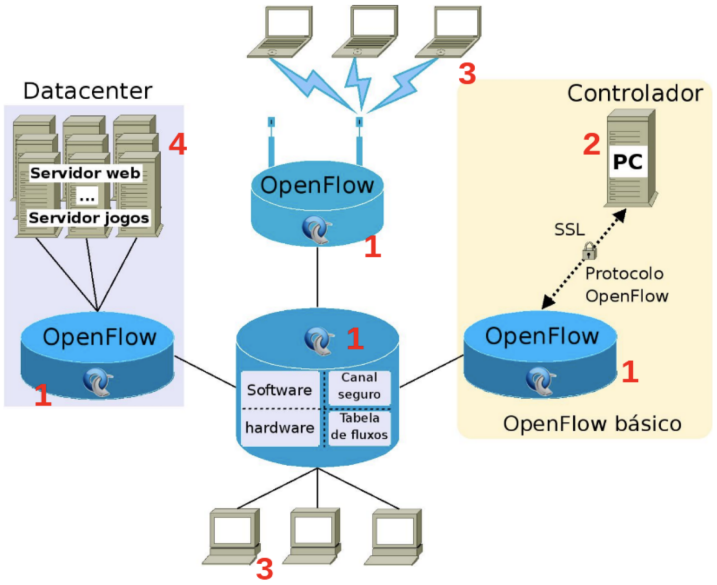

Figura 2.8: *Arquitetura básica de Openflow. (Esteve Rothenberg et al., 2018)*

No exemplo da Figura 2.8, os itens identificados por 1 são *switches* OpenFlow, dispositivos responsáveis pela comunicação da rede. Neles estão incluídos as tabelas de fluxos e o canal seguro, juntamente com o plano de dados e o plano de controle. O controlador, que está

identificado pelo item 2, é quem gerencia, remotamente, os encaminhamentos de pacotes. A comunicação entre controlador e *switches* é feita pelo canal seguro através do protocolo OpenFlow. Já os itens 3 e 4 identificam os usuários finais da rede (Silva, 2016).

O pacote chega no dispositivo de encaminhamento e este verifica se existe algum fluxo cuja regra se encaixa com o pacote. Em caso afirmativo, a ação é executada com esse pacote, caso negativo o pacote é enviado para o controlador e esse permite que as aplicações de rede decidam o que fazer com esse pacote, adicionando um pequeno *overhead* na transmissão de dados. Quando essa nova decisão é tomada, ela é armazenada nos dispositivos de encaminhamento para evitar um tráfego intenso com o controlador e lá permanece por um determinado intervalo de tempo.

2.3.2 Controladores

Como dito anteriormente, os controladores são o centro do plano de controle das redes definidas por software, agindo como verdadeiros sistemas operacionais da rede, onde gerenciam as tabelas de fluxos dos dispositivos de encaminhamento (Esteve Rothenberg *et al.*, 2018) e abstraem esse conceito para as aplicações, tornando mais fácil o desenvolvimento de novas características para as redes.

Existem diversos tipos de controladores, em diversos tipos de linguagens, como o NOX, desenvolvido em C++ e Python ou o Beacon, desenvolvido em Java e que permite a utilização na plataforma Android. Entretanto este trabalho terá como foco um terceiro controlador chamado POX, escrito em Python, por conta da sua simplicidade e melhor desempenho quando comparado ao NOX.

2.4 Emulação de rede

Um grande problema quando se trata de pesquisa na área de redes são as metodologias utilizadas para execução de testes e experimentos, pois é necessário uma infraestrutura muito grande, com equipamentos físicos interconectados a fim de criar um ambiente que consiga reproduzir a realidade. Nesse sentido, surge o conceito de emulação de rede, que basicamente permite a criação de um ambiente virtual que emule as características de uma rede real, com roteadores, *switches* e *links* entre eles.

A ferramenta de emulação a ser utilizada neste trabalho é a Mininet, que funciona como uma coleção de dispositivos físicos conectados em um único *kernel* Linux (Silva, 2016). Ela também permite a criação de topologias de rede com facilidade e com total suporte a tecnologia OpenFlow de tal forma que o código desenvolvido para o Mininet pode ser implementado no mundo real com mudanças mínimas (Mininet, s/d).

Capítulo 3

Balanceamento de carga por *hardware*

A distribuição de carga por *hardware* (também conhecida como HLD, do inglês *hardware load balancer device*) se dá através de um dispositivo físico, o balanceador, que se conecta fisicamente com os servidores disponíveis, posicionando-se geralmente entre a rede externa e a rede interna da organização que fornece o serviço a ser balanceado. Esse dispositivo conta com um processador próprio, memórias e interfaces de rede para receber e distribuir os pacotes da melhor maneira possível.

Esse era o único método utilizado na década de 1990 conforme Sekar (2017), já que a utilização de um meio que envolvesse software era muito difícil e lenta pois os *hardwares* da época não eram tão evoluídos (segundo a Figura 3.1, em 1995, a memória RAM tinha capacidade média de aproximadamente 100Mb, o que causa impacto negativo no desempenho dos programas) sendo necessário um dispositivo com enfoque em distribuição de carga para otimização dessa tarefa.

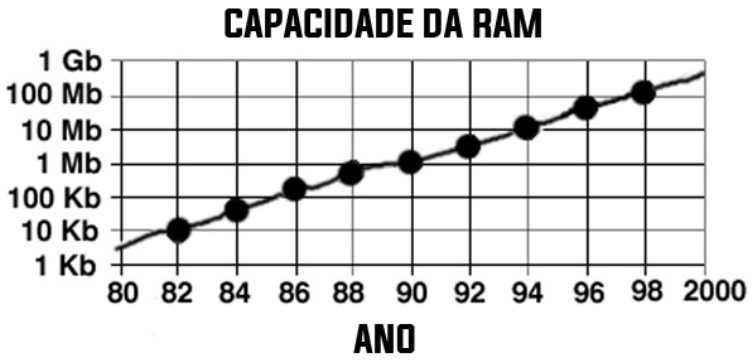

Figura 3.1: *Gráfico da capacidade da memória RAM por ano. (Pagano, 2012, p.55, tradução nossa)*

O uso de sistemas operacionais focados em distribuição de carga, *firmwares* especializados em rede e também algoritmos de distribuição otimizados para determinado *hardware* são características que permitem uma maior eficiência para a execução do balanceamento. Mas nem todos os balanceadores possuem todas essas características. Alguns, como o TP-Link TL-R480+, são dispositivos que possuem apenas uma interface de rede com suporte a diversos protocolos e uma política de roteamento simples, sem a necessidade de um suporte através de software. [1] Isso mostra o quanto esse mercado é variado, possuindo diversos tipos de balanceadores que atendem a grupos variados de usuários.

[1] As informações aqui apresentadas sobre o balanceador TP-Link TL-R480+ estão disponíveis em <https://www.tp-link.com/br/products/details/cat-4910_TL-R480T+.html>. Acessado em 16/08/2018.

Como eles podem atuar na camada 4 (transporte) e na camada 7 (aplicação) do modelo OSI (acrônimo do inglês *Open System Interconnection*, padrão que define uma caracterização das redes de computadores em camadas), é comum que se classifique os roteadores em L4 ou L4/L7, dependendo de sua camada de atuação. Os roteadores que atuam na camada 7 conseguem ter acesso ao conteúdo da mensagem (URL e *cookies*, por exemplo) pois o protocolo HTTP é parte da camada de aplicação e, por isso, esses roteadores formam uma rede de entrega de aplicação (ADN, do inglês *application delivery network*) onde a entrega é baseada no conteúdo dessa camada, por exemplo, *scripts* PHP para um conjunto de servidores específicos e páginas HTML para um outro conjunto. Por esse conjunto de recursos disponíveis, os roteadores L4/L7 são em geral muito mais caros do que os L4, que atuam apenas na camada de transporte, realizando o encaminhamento dos pacotes.

A utilização de *hardware* para o balanceamento também dá à organização uma maior segurança pois o dispositivo não pode ser acessado fisicamente por pessoas que não tenham acesso às dependências do local onde o mesmo é mantido.

Além da questão da segurança, os aparelhos também tem a capacidade de permitir a aceleração do protocolo de camada de *sockets* segura (ou SSL, do inglês *Secure Sockets Layer*, o padrão de estabelecimento de conexões seguras entre o servidor e o cliente), dessa forma, a decodificação de dados aconteceria no balanceador antes de passar a requisição para frente. Isso permite que os servidores não gastem tempo de computação extra nessa desencriptação, melhorando assim o seu desempenho.

Porém, existem diversos problemas da implementação de dispositivo físico para essa tarefa. O primeiro deles e o mais significativo é o custo muito elevado, tanto para adquirir um balanceador de última geração quanto para fazer a manutenção da rede física.

Além disso, os dispositivos possuem um limite de servidores que podem ser conectados a eles, o que torna difícil a escalabilidade do sistema pois caso seja necessário adicionar mais máquinas que o limite ao *server-farm*, precisa-se adicionar mais um balanceador ou trocá-lo por um que tenha um limite maior, tornando o uso de *hardware* para essa tarefa pouco flexível a mudanças.

3.1 Custo dos dispositivos

O custo dos dispositivos é bem variado, como dito anteriormente. Tudo depende das suas funcionalidades, da quantidade de tráfego que estima-se e do número de servidores disponíveis. A Tabela 3.1 lista alguns balanceadores de carga, suas funcionalidades e seus respectivos preços.

Dispositivo	**Aceleração SSL**	**Taxa de Transferência (Gbps)**	**Capacidade máxima de servidores**	**Preço (USD)**
KEMP LM-2200	Sim	0,95	4	100
KEMP LM-X3	Sim	3,4	8	4.000
Citrix Netscaler MPX 7500	Sim	5	8	17.300
F5 LTM-2000S	Sim	5	8	17.300
Citrix MPX-8015	Sim	Não Publicado	12	48.000
KEMP LM-8020M	Sim	30	8	60.000
F5-BIG-BT-i7800	Sim	40	12	144.900

Tabela 3.1: *Tabela de características de alguns dispositivos físicos.(KEMP Technologies, s/d, tradução nossa)*

Capítulo 4

Balanceamento de carga usando SDN

4.1 SDN como balanceador de carga

Como visto no Capítulo 3, a utilização de um dispositivo físico para a execução do balanceamento de carga tem um custo de implementação muito alto (o preço de bons dispositivos varia na casa dos milhares de dólares). Para resolver este problema, será proposto uma aplicação de SDN como balanceador de carga, diminuindo o custo e permitindo ao administrador maior flexibilidade quando comparado ao modelo fixo determinado pelos fabricantes dos balanceadores.

A aplicação de SDN para resolver esse problema é bastante clara, já que a principal característica desses dispositivos é a compreensão do estado atual da rede interna e de como os servidores estão se comportando, exatamente o que a técnica de rede definida por software permite.

Será feito, então, proveito tanto da flexibilidade quanto da facilidade que a SDN permite para criar um balanceador de carga que execute 3 algoritmos diferentes para balanceamento de carga (aleatório, *round-robin* e *least-bandwidth*) e que permita ao administrador selecionar não só o algoritmo, mas também a proporção de carga que cada servidor irá receber, tudo isso em tempo de execução utilizando o terminal.

Dessa forma, quem será o responsável por receber e redirecionar os pacotes para os servidores será o *switch* OpenFlow. O controlador será responsável por receber as informações do administrador e aplicá-las no *switch*, alterando a configuração do balanceador, além de selecionar qual servidor deve receber a próxima requisição, como pode-se ver na Figura 4.1.

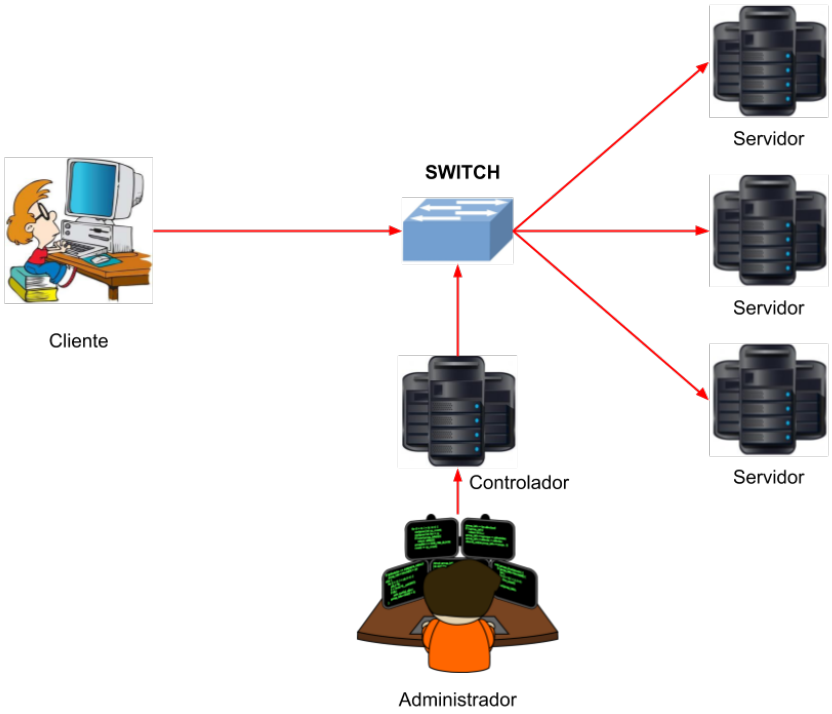

Figura 4.1: *Exemplificação do uso de SDN para balanceamento de carga.*

Para isso, será utilizado o controlador POX, um controlador escrito em Python, que permite um desenvolvimento rápido e fácil, bastante utilizado para prototipação de projetos em SDN.

A execução do POX também é bastante simples, como podemos ver no Código 4.1. Basta executar o comando pox.py, passando como argumento opcional as opções do controlador e como argumento obrigatório pelo menos um nome do componente (um módulo Python) que será executado, mais especificamente a aplicação SDN que irá se comunicar com o controlador.

```
1    ./pox.py [POX options] [C1 [C1 options]] [C2 [C2 options]] ...
```

Código 4.1: *Uso do controlador POX, onde C1, C2, etc. são nomes dos controladores (módulos Python).*

Será preciso, portanto, criar um componente POX que será o balanceador de carga. Convenientemente, o controlador já possui um exemplo de componente que executa essa tarefa, mas não contém todas as características que foi definido anteriormente. Sendo assim, será utilizado esse módulo como base para o desenvolvimento do balanceador.

4.2 Módulo de balanceamento

O módulo de balanceamento do POX se chama *ip_loadbalancer* e pode ser encontrado no diretório *misc* do controlador. Nele, existem duas classes: a MemoryEntry, uma abstração dos fluxos que estão nos *switches* e também a classe ipbl, contendo a lógica do balanceador em si. Essa classe armazena os endereços IP dos servidores, o endereço IP do controlador, os fluxos que estão ativos e os métodos para seleção de servidores e para tratamento dos pacotes que chegam ao controlador.

O componente atualmente recebe, pela linha de comando (como pode-se ver no Código 4.2), o endereço IP no qual o balanceador será utilizado (no exemplo vemos que esse

endereço foi definido como 10.0.1.1) e a lista de endereços dos servidores (no código 4.2, essa lista é definida por 10.0.0.1, 10.0.0.2 e 10.0.0.3). O controlador, então, usa essas informações dos servidores para, de tempos em tempos, enviar uma mensagem ARP para esses endereços e assim conseguir descobrir se houve alguma queda em um servidor (a máquina que não responder a mensagem por um longo período provavelmente caiu e deve ser removida da lista), realizando, assim, o *health checking* dos servidores.

```
1    ./pox.py misc.ip_loadbalancer \\
2    ip=10.0.1.1 \\
3    servers=10.0.0.1,10.0.0.2,10.0.0.3
```

Código 4.2: *Exemplo de execução do componente ip_loadbalancer.*

O balanceamento desse módulo é feito pelos protocolos TCP/IP, ou seja, apenas pacotes desse protocolo irão ser balanceados entre os servidores (que são os protocolos mais utilizados em aplicações *web*). Quando um pacote desse protocolo é recebido no endereço IP do balanceador, o controlador irá se encarregar de enviar para um dos endereços IP da lista de servidores ativos usando o algoritmo aleatório e, quando se faz isso, é criada uma cópia do fluxo no controlador por um período de tempo.

A Figura 4.2 exemplifica como é feito o balanceamento utilizando esse componente. No exemplo, os servidores conectados ao *switch* possuem os endereços IP 10.0.0.1, 10.0.0.2 e 10.0.0.3 e o balanceador em si possui endereço IP externo 10.0.1.1 e está conectado diretamente com a internet. Os pacotes que chegam no balanceador são enviados para o controlador e ele decidirá para qual servidor o *switch* deve redirecionar.

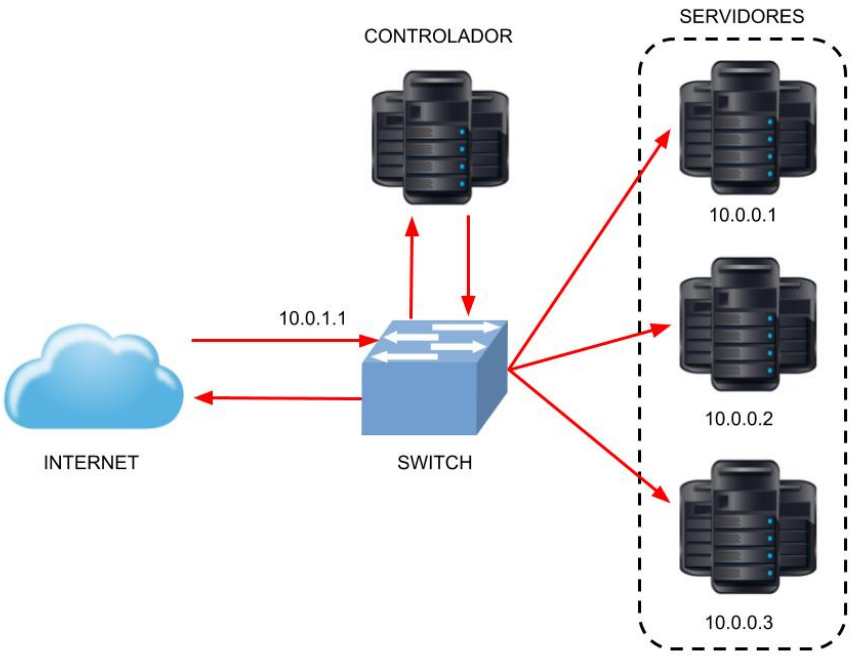

Figura 4.2: *Exemplificação do componente ip_loadbalancer.*

Será adicionado a esse módulo mais 2 algoritmos de balanceamento (*round-robin* e *least-bandwidth*), ambos com suporte aos pesos por servidores, além de possibilitar ao administrador que consiga alterar os algoritmos em tempo de execução, com um terminal no próprio controlador.

4.3 Desenvolvimento do balanceador

As funcionalidades do balanceador foram elencadas a seguir, por ordem de dependência:

1. Adição dos algoritmos (*round-robin* e *least-bandwidth*);
2. Suporte a pesos nos algoritmos;
3. Adição de interação com o usuário.

4.3.1 Algoritmo *round-robin*

Todos os novos algoritmos devem ser adicionados no método _pick_server, que tem o objetivo de selecionar o próximo servidor a atender a próxima requisição.

Para a adição especificamente do *round-robin*, primeiramente, será necessário a lista de servidores. A classe do balanceador (*iplb*) já tem essa estrutura através da variável live_servers que possui todos os endereços IP dos servidores ativos. Dessa forma, será preciso apenas adicionar um contador para saber qual o índice dessa lista está o servidor que atenderá a próxima requisição.

Na construção do balanceador, esse índice (que é uma variável de instância da classe iplb) se inicia com 0. A cada nova chamada do _pick_server, o endereço IP daquela posição será recuperado para atender a requisição e, por fim, será adicionado 1 ao índice.

Caso o valor da variável ultrapasse o tamanho da lista, será atribuído 0 a ela. Mantendo, assim, a circularidade do algoritmo sem adicionar novas estruturas de dados ao balanceador.

Por fim, será permitido que o usuário consiga escolher entre os dois algoritmos de balanceamento possíveis. Para isso, será adicionado o suporte a uma *flag* de linha de comando chamada *algorithm* que poderá ser *random* (caso se queira o algoritmo aleatório, que é o valor padrão da *flag*), *round-robin* ou *least-bandwidth*. Esse valor será passado para o balanceador e ele irá selecionar o algoritmo baseado nisso.

4.3.2 Algoritmo *least-bandwidth*

Para esse algoritmo, além da lista de endereços IP, também será preciso ter acesso à quantidade de tráfego transferido para os servidores e, assim, selecionar aquele com o menor valor.

Com a técnica de SDN, tendo como base tudo que foi exposto até esse ponto, isso é bastante simples de ser feito, pois as informações de tráfego podem ser obtidas analisando as estatísticas de cada fluxo que estão nos *switches*. O OpenFlow permite que o controlador peça para os *switches* essas estatísticas através de uma requisição específica (a *StatsRequest*), dessa forma, o que o controlador precisa é pedir esses dados de forma contínua, para atualizar os valores de tráfego para cada servidor.

A classe do balanceador, então, terá um dicionário onde as chaves são os endereços IP dos servidores e o valor é a quantidade de tráfego para cada servidor nos últimos 14 segundos (um valor arbitrário de tal forma que o controlador não gaste tanto tempo se encarregando de tratar as requisições de estatísticas dos *switches*). Dessa forma, será implementado um *timer* para que a cada 14 segundos o controlador envie os StatsRequests para cada um dos *switches* e assim, quando a resposta for recebida, os valores do dicionário serão atualizados para cada um dos servidores envolvidos nos fluxos.

4.3.3 Taxa de carga por servidor

Supondo que a classe do balanceador receba em seu construtor de classe um dicionário onde a chave é um endereço IP e o valor um número inteiro que representa a proporção de carga a qual o servidor com aquele endereço IP deve receber, é possível alterar os algoritmos vistos anteriormente para suportar pesos (ou taxas de carga) por servidores.

No *round-robin*, se o peso do servidor é x, então será necessário o envio de x requisições antes de passar para o próximo elemento da lista. Sendo assim, basta a criação de mais um contador que representará quantas requisições aquele servidor atendeu com aquele índice. Assim que esse valor for igual ao peso daquele servidor, o índice poderá avançar para a próxima posição.

Já para o *least-bandwidth*, será utilizado a fórmula baseada no algoritmo de menor conexão com pesos (Linux Virtual Server Knowledge Base, s/d), supondo um conjunto de servidores $S_1, S_2, ..., S_n$, supomos também que $W(S_i)$ e $B(S_i)$ são o peso do servidor S_i e o tráfego consumido por ele, respectivamente. Seja B_t a quantidade total de tráfego consumido por todos os servidores, dessa forma, $B_t = \sum_{i=1}^{n} B(S_i)$. Logo, o servidor a responder à próxima requisição é o servidor m cujo valor $\frac{\frac{B(S_m)}{B_t}}{W(S_m)}$ é o menor dentre todos.

Como B_t é uma constante, pode-se omitir e, dessa forma, basta achar o servidor m onde $\frac{B(S_i)}{W(S_i)} > \frac{B(S_m)}{W(S_m)}$ para todo i tal que $1 <= i < n$. Como divisão consome muito mais ciclos de processamento quando comparado à multiplicação, é possível otimizar essa fórmula para $B(S_i) * W(S_m) > B(S_i) * W(S_m)$. Será percorrido, então, toda a lista de endereços IP de servidores apenas uma vez, como visto no Código 4.3, para encontrar o servidor que satisfaça essa condição. Dessa forma, o tempo consumido pelo algoritmo é $O(n)$.

```
1     for i in range(n):
2         if weights[servers[i]] > 0:
3             best_server = servers[i]
4
5             for ii in range(i + 1, n):
6                 if data_transferred[best_server]*weights[servers[ii]] > \
7                  data_transferred[servers[ii]]*weights[best_server]:
8                     best_server = servers[ii]
9             return best_server
```

Código 4.3: *Código do algoritmo* least-bandwidth *com suporte a pesos.*

Por fim, será permitido que o usuário escolha os pesos dos servidores através de um argumento da linha de comando chamado weights_val, que é um vetor contendo os pesos de todos os servidores na mesma ordem da lista de endereços IP, ou seja, se o servidor de endereço IP 10.0.0.3 foi o terceiro a ser adicionado na lista de servidores da linha de comando, então o seu peso é representado pelo terceiro valor no argumento weights_val, como pode-se ver no Código 4.4, onde o peso do servidor de endereço 10.0.0.3 é 2.

```
1    ./pox.py misc.ip_loadbalancer \\
2    ip=10.0.1.1 \\
3    servers=10.0.0.1,10.0.0.2,10.0.0.3 \\
4    algorithm=round-robin \\
5    weights_val=1,2,2
```

Código 4.4: *Exemplo de execução do balanceador de carga com pesos nos servidores.*

Caso nenhum valor de peso seja especificado, será utilizado o peso 1 para todos os servidores. Caso também a quantidade de valores especificados nesse argumento seja diferente da

quantidade de servidores, será retornado um erro para o usuário e execução do controlador será cancelada.

4.3.4 Interação com o usuário

A interação com o usuário será feita através de um terminal situado no controlador. Nele, o usuário poderá alterar os pesos dos servidores e o algoritmo de balanceamento.

O controlador POX possui um meio de adicionar interatividade no controlador através do objeto Interactive, que permite que outras aplicações (no caso, o nosso módulo de balanceamento) interajam com o interpretador Python que está rodando o controlador. Assim, basta relacionar o nome da variável a qual o usuário irá digitar no interpretador com a função que deseja-se que seja executada. O usuário também precisará expressar na linha de comando, quando for executar o controlador, que gostaria de acessar o interpretador através da *flag* py, como pode-se ver no Código 4.5.

```
1    ./pox.py misc.ip_loadbalancer \\
2    ip=10.0.1.1 \\
3    servers=10.0.0.1,10.0.0.2,10.0.0.3 \\
4    algorithm=round-robin \\
5    weights_val=1,2,3 \\
6    py
```

Código 4.5: *Exemplo de execução do balanceador de carga com a flag do interpretador.*

Cada uma dessas funções estará definida como um método da classe do balanceador. Para a alteração do algoritmo, o usuário precisará digitar no interpretador o comando change_algorithm e passar como argumento o nome do novo algoritmo (podendo ser *random*, *round-robin* ou *least-bandwidth*), como pode-se ver no Código 4.6.

```
1    > change_algorithm("random")
```

Código 4.6: *Exemplo de mudança de algoritmo do balanceador para o aleatório.*

Já para alteração dos pesos dos servidores, o usuário precisará digitar a função change_weights e passar como argumento um dicionário, onde a chave é o endereço IP de cada servidor e o valor o seu respectivo peso (como exemplificado no Código 4.7).

```
1    > change_weights({"10.0.0.1": 1, "10.0.0.2": 4, "10.0.0.3": 3 })
```

Código 4.7: *Exemplo de mudança de pesos de 3 servidores com endereços IP 10.0.0.1, 10.0.0.2, 10.0.0.3 para 1, 4 e 3, respectivamente.*

Capítulo 5

Análise de desempenho

5.1 Metodologia

Para testar a eficiência do balanceador de carga, ele será submetido a um teste de carga e será avaliado a taxa de resposta do mesmo para cada um dos tipos de algoritmos implementados e para uma quantidade variada de servidores conectados ao balanceador.

O software utilizado para os testes será o Apache Benchmark, um programa de linha de comando disponibilizado pela Apache Software Foundation que permite uma avaliação comparativa de servidores HTTP, avaliando quantas requisições por segundo os servidores conseguem servir de maneira ótima. Ele foi criado originalmente para testar apenas servidores HTTP Apache, porém ele é genérico suficiente para testar qualquer tipo de servidor (Apache HTTP Server Version 2.4, s/d).

A ferramenta Apache Benchmark é gratuita e distribuída sob os termos da Licença Apache, podendo ser obtido, em um ambiente Linux baseado na distribuição Debian, através do comando `apt-get install ab`. Após a instalação, o usuário conseguirá executar a ferramenta com o comando `ab`. A versão utilizada nos experimentos foi a 2.3.

O Apache Benchmark permite a execução de requisições concorrentemente, ou seja, mais de uma requisição ao servidor pode ser feita simultaneamente. Isso se assemelha a um cenário real onde mais de um usuário poderá fazer o pedido de algum serviço do servidor. Dessa forma, será utilizado esse recurso da ferramenta para simular um cenário onde 10 usuários tentam acessar alguma página da nossa aplicação. A simulação também irá fazer 1000 requisições a uma mesma URL para assim analisar se o balanceador distribui de forma eficiente e mantém a taxa de resposta média aceitável.

Assim, o teste se baseia em executar o comando exibido no Código 5.1 50 vezes e calcular a média aritmética do tempo de resposta de cada um dos testes.

```
1    ab -n 1000 -c 10 http://10.0.1.1
```

Código 5.1: *Execução do Apache Benchmark para os testes, supondo que o endereço IP do balanceador é 10.0.1.1.*

Como não temos *hardware* disponível para a execução dos testes com diversos servidores, será utilizado o emulador Mininet, conforme explicado na Seção 2.4, emulando uma rede com a topologia padrão do Mininet, ou seja, um *switch* (que será o balanceador) conectado a outros *hosts*, que poderão ser tanto servidores quanto clientes. Logo, será criado uma rede que terá $n+1$ *hosts*, onde n representa a quantidade de servidores que se quer testar, restando 1 *host* para servir apenas como cliente da aplicação, onde serão rodados os testes.

O *host* número 1 (cujo IP definido pelo Mininet é sempre 10.0.0.1) sempre será o cliente, enquanto os outros (cujos IPs variam de 10.0.0.2 até 10.0.0.n) serão os servidores da

aplicação.

A Figura 5.1 exemplifica como será a topologia da rede para um teste com 5 servidores e como serão separadas cada uma das funções. O comando a ser executado para a criação de tal rede pode ser visto no Código 5.2. O controlador será de natureza remota pois será executado simultaneamente, utilizando o comando do Código 5.3, o módulo de balanceamento que foi apresentado na Seção 4, que aguardará por conexões na porta de número 6633.

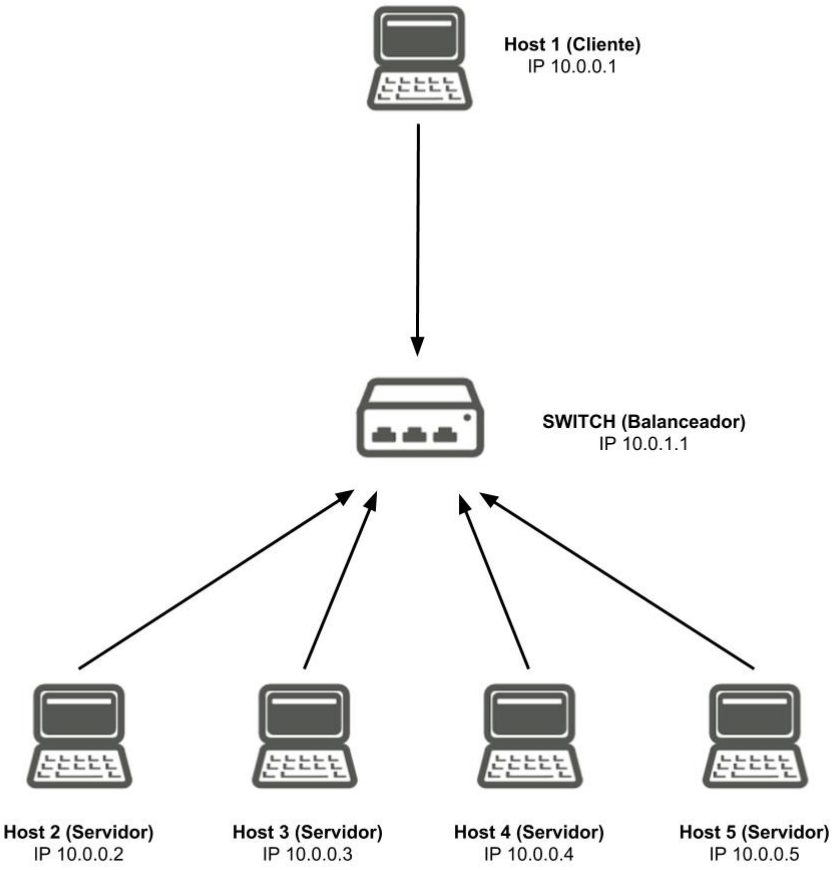

Figura 5.1: *Exemplificação da rede emulada para os testes com 5 servidores.*

```
1    sudo mn --topo single,5 --controller=remote,port=6633
```

Código 5.2: *Exemplificação do comando para a criação de uma rede com 5 servidores para os testes de carga do balanceador.*

```
1    ./pox.py misc.ip_loadbalancer \\
2    --ip=10.0.1.1 \\
3    --servers=10.0.0.1,10.0.0.2,10.0.0.3,10.0.0.4,10.0.0.5 \\
4    --algorithm=random
```

Código 5.3: *Exemplo do comando utilizado para executar o controlador que fará o balanceamento de carga para 5 servidores utilizando o algoritmo aleatório.*

Os testes serão divididos em duas partes. Na primeira, será avaliado o desempenho do balanceador (através do tempo de resposta) para 2, 3, 4, 5, 6, 7 e 8 servidores, onde cada um deles entregará uma página *web* com tamanho aleatório (esse tamanho varia de 100 kilobytes

a 50 megabytes) de forma semelhante ao que acontece em um servidor real, já que as páginas que os cliente pedem nem sempre possuem o mesmo tamanho, podendo variar de requisição para requisição.

A segunda parte tem como objetivo avaliar como o balanceador se comporta quando os servidores entregam uma única página de tamanho definido. Para tal teste, será utilizado um *server-farm* com 8 servidores e executaremos os testes com páginas de tamanho de 100KB, 250KB, 500KB, 1MB, 2.5MB, 5MB, 10MB, 25MB e 50MB e será analisado o tempo de resposta para cada um desses cenários.

Por fim, os testes foram realizados em uma máquina virtual Ubuntu 14.04.4 com 4096MB de memória RAM reservadas rodando em um Macbook Pro com processador Intel Core i5 de 2,3 GHz, memória RAM de 8 GB e sistema operacional macOS High Sierra v10.13.6. Foi utilizado também o Python 2.7.6, o Mininet 2.2.2 e o POX 0.5.0 (eel).

5.2 Resultados obtidos

Os testes com quantidade de servidores diferentes, cujos resultados podem ser vistos na Figura 5.2, mostraram que o controlador se comporta como o esperado, ou seja, ao adicionarmos mais servidores, o tempo de resposta tende a cair. Além disso, o tempo de resposta médio para os testes foram bastante aceitáveis (o máximo alcançado foi de 28 milissegundos o que, segundo Nielsen (1993), está dentro do que o usuário espera como resposta instantânea). Uma outra consideração que pode-se fazer com relação aos resultados é que o tempo de resposta cai pela metade quando comparamos um *server-farm* com 2 servidores e um com 3 servidores, entretanto, o resultado se mantém quase que constante a partir dessa quantidade de servidores.

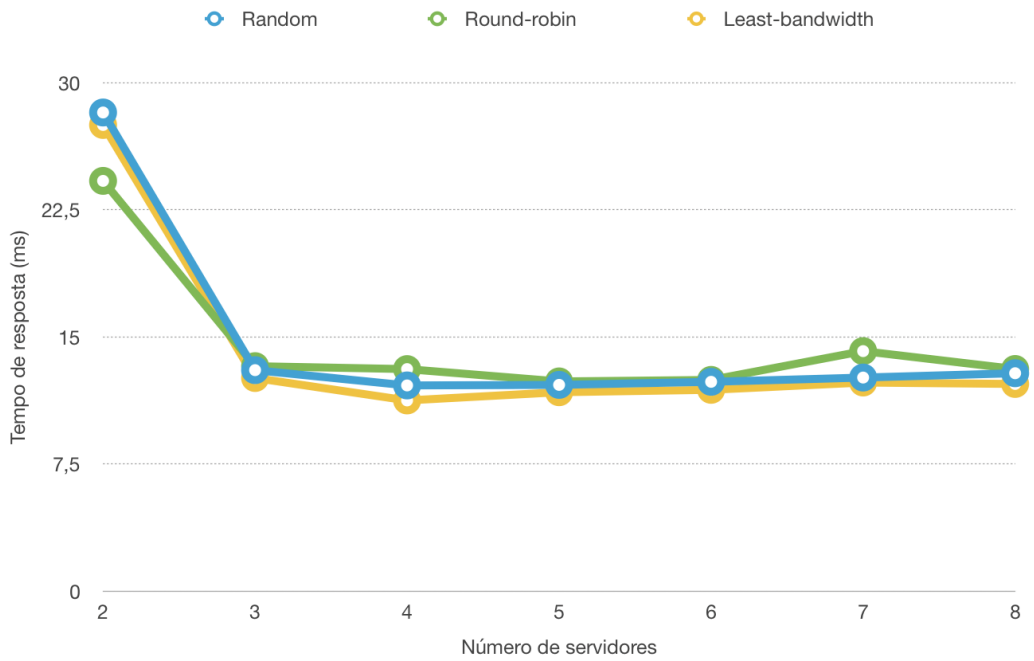

Figura 5.2: *Gráfico mostrando o tempo de resposta com relação ao número de servidores para cada um dos algoritmos.*

Quando são comparado os resultados entre os algoritmos, percebe-se que a diferença entre eles é mínima (a diferença máxima entre os resultados é de 1 ms). Entretanto, pode-se notar que o tempo de resposta do algoritmo aleatório é, em geral, maior que os outros algoritmos.

Já quando são comparado os métodos *round-robin* e *least-bandwidth*, percebe-se que, para poucos servidores, o primeiro funciona relativamente melhor do que o segundo, mas o padrão contrário se apresenta conforme vamos adicionando mais servidores.

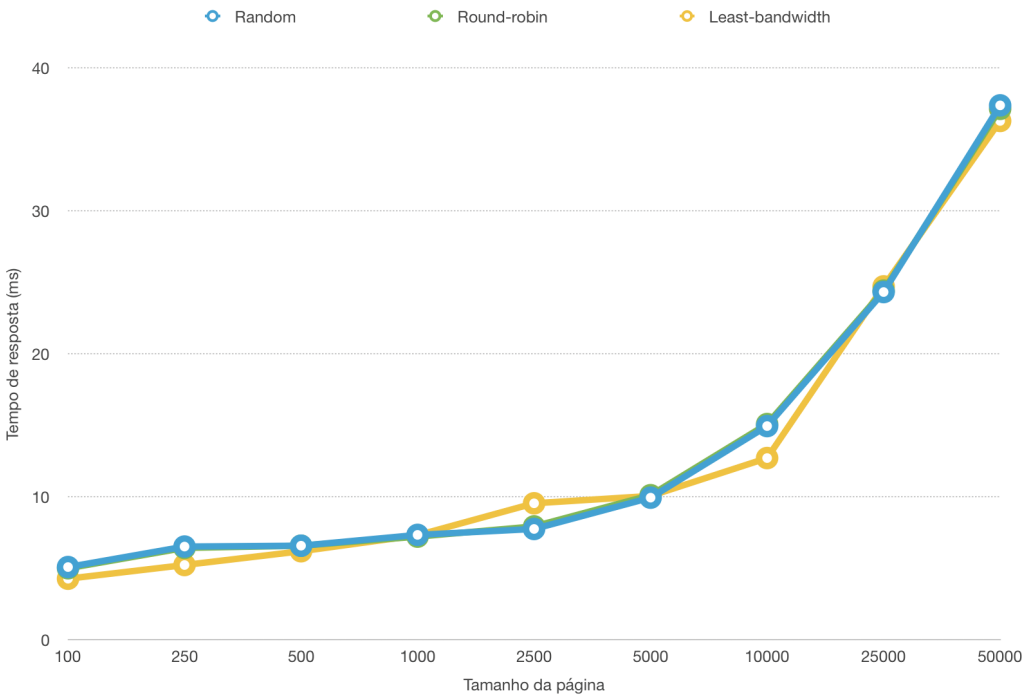

Figura 5.3: *Gráfico mostrando o tempo de resposta com relação ao tamanho das páginas oferecidas pelos servidores para cada um dos algoritmos. Teste realizado com um* server-farm *de 8 servidores.*

A Figura 5.3 apresenta os resultados da segunda parte dos testes. Nela, pode-se perceber que o balanceador criado também tem um comportamento esperado no sentido de que o tempo de resposta aumenta ao passo que aumentamos o tamanho do recurso oferecido pelos servidores. Uma análise mais profunda permite concluir que a diferença entre os tempos de respostas para páginas relativamente pequenas (entre 100KB e 5MB) são bem pequenas, chegando na casa dos 5 milissegundos. Já para páginas maiores (entre 10MB e 50MB) a diferença é bem maior, alcançando a casa dos 23 milissegundos.

Entretanto, o tempo de resposta médio continua aceitável tanto para páginas pequenas quanto para páginas grandes, sendo o máximo alcançado de 38 milissegundos, o que ainda está dentro do que se espera de uma resposta instantânea.

De forma semelhante ao primeiro teste, percebemos que as diferenças entre os algoritmos são bem pequenas, principalmente quando comparamos o *round-robin* com o método aleatório (diferença média menor que 1 ms).

Apesar da pequena diferença, percebe-se que o método *least-bandwidth* possui tempo de resposta baixo em geral quando comparado aos outros algoritmos. Esse fato se dá ao modo em que o método funciona, ou seja, sua avaliação do tráfego consumido entre os servidores permite que o servidor que tenha consumido menos (ou seja, foi menos requisitado) atenda a próxima requisição.

Os testes avaliados aqui não levaram em conta os pesos dos servidores, ou seja, a carga foi igualmente separada para cada uma das máquinas do *server-farm*. Essa decisão foi tomada pois espera-se que o uso da taxa de carga seja feito quando se tenha servidores com diferentes características (ou seja, capacidade de memória RAM diferente, processadores diferentes) que permitem que a carga seja distribuída de forma a aproveitar esses recursos de melhores servidores. No caso dos testes realizados, as máquinas possuíam características

semelhantes, dessa forma, as taxas deveriam ser distribuídas igualmente para obtermos o melhor balanceamento.

Capítulo 6

Conclusões e trabalhos futuros

Com este trabalho, foi exposto o quão eficiente um balanceador de carga pode ser utilizando-se apenas de redes definidas por software. Utilizando como base o controlador POX, os testes mostraram que o tempo de resposta de uma fazenda de servidores que faz uso de um balanceador baseado em SDN está dentro da margem de aceitação do usuário (menor que 0.1 segundo).

Além disso, o balanceador elaborado nesse trabalho possui a característica de ser mais flexível quando comparado a um dispositivo físico, ou seja, o controlador permite a interação com o usuário de tal forma que ele pode escolher os algoritmos em tempo de execução, sem a necessidade de desativar toda a aplicação ou trocar fisicamente o dispositivo de balanceamento. Isso garante uma vantagem ao administrador, que não precisa ficar preso às características do *hardware* de um fabricante específico.

O administrador, ao utilizar um balanceador como o elaborado nesse trabalho poderá também se beneficiar da flexibilidade com relação ao número de servidores disponíveis, ou seja, enquanto dispositivos físicos de balanceamento possuem um limite em geral pequeno, o balanceador utilizando SDN se limita apenas à quantidade de portas disponível nos dispositivos de encaminhamento (que em geral são bem maiores que as dos balanceadores físicos).

Por fim, o maior ganho ao utilizar um balanceamento por SDN se encontra no preço dos *hardwares* específicos e otimizados para realizar essa ação. Enquanto o custo de implementação dos HLDs giram em torno de milhares de dólares (como visto na Seção 3) para se obter um balanceamento de qualidade e com diferentes algoritmos de balanceamento, o custo estimado para a implementação de um balanceador como o elaborado neste trabalho é limitado superiormente pelo preço do *switch* utilizado para encaminhar os pacotes para os servidores (um exemplo de *switch* OpenFlow de boa qualidade é o HP Aruba 2920 24G PoE+ Switch, cujo preço não passa de U$1200.00[1], possui uma taxa de encaminhamento próxima de 128Gbps e permite a conexão com mais de 20 servidores).

Além dos resultados obtidos com a análise de desempenho do balanceador de carga desenvolvido e com a comparação do custo dessa solução em relação à utilização de HLDs, este trabalho também contribuiu com a disponibilização do código do balanceador desenvolvido. O código encontra-se em https://github.com/lcfpadilha/pox sob licença Apache 2.0. Como continuação do trabalho, pode-se focar na adição de recursos que tornem o nosso módulo de balanceamento de carga mais próximo dos ADLs disponíveis, aumentando assim seu valor em comparação com outros balanceadores. A adição de descarregamento SSL (desencriptando o pacote antes de enviá-lo ao servidor) e proteção DDoS podem ser características que diferenciem o balanceador de outros.

[1]HP Aruba 2920 24G PoE+ Switch. Disponível em <https://www.hpe.com/br/pt/product-catalog/networking/networking-switches/pip.aruba-2920-switch-series.5354494.html>. Acessado em 23/11/2018.

Também pode-se trazer mais interação para o usuário, permitindo que ele consiga checar a saúde dos servidores, um *log* com o *status* do balanceador, além de permitir desativar o balanceamento para servidores específicos.

Além disso, pode-se trazer mais inteligência ao balanceador permitindo que ele próprio selecione o algoritmo que melhor executará o balanceamento de carga em tempo de execução através de aprendizagem de máquina, por exemplo.

No mais, também se espera que a facilidade da linguagem utilizada e da técnica de SDN permita ao administrador elaborar o balanceador à sua necessidade, não sendo mais uma dispositivo complexo e rígido como os balanceadores físicos atuais.

Bibliografia

Apache HTTP Server Version 2.4(s/d) Apache HTTP Server Version 2.4. *ab - Apache HTTP server benchmarking tool*, s/d. Último acesso em 22/10/2018. Citado na pág. 23

Duque *et al.*(**2012**) Diego Henrique Duque, Edimilson Joaquim Couto, Marcelo Henrique Pinto e Thiago Leopoldino Magalhães. Redes Definidas por Software, 2012. Monografia (Graduação em Engenharia Elétrica), Inatel (Instituto Nacional de Telecomunicações). Citado na pág. xi, 9, 10

Esteve Rothenberg *et al.*(**2018**) Christian Esteve Rothenberg, Marcelo Nascimento, Marcos Salvador e Maurício Ferreira. OpenFlow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. Citado na pág. xi, 1, 10, 11

Fernández *et al.*(**2018**) Jesús Fernández, Luis García Villalba e Tai-Hoon Kim. Software defined networks in wireless sensor architectures. *Entropy*, 20:225. doi: 10.3390/e20040225. Citado na pág. 9

Gandhi *et al.*(**2014**) Rohan Gandhi, Hongqiang Harry Liu, Y. Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan e Ming Zhang. Duet: Cloud scale load balancing with hardware and software. *SIGCOMM Comput. Commun. Rev.*, 44(4):27–38. ISSN 0146-4833. doi: 10.1145/2740070.2626317. URL http://doi.acm.org/10.1145/2740070.2626317. Citado na pág. 1

Godha e Prateek(2014) Rahul Godha e Sneh Prateek. Load balancing in a network. *International Journal of Scientific and Research Publications*, 4. Citado na pág.

Kaur *et al.*(**2015**) Sukhveer Kaur, Japinder Singh, Krishan Saluja e Navtej Singh Ghumman. Round-robin based load balancing in software defined networking. Citado na pág.

KEMP Technologies(s/d) KEMP Technologies. Comparison of the KEMP LoadMaster with F5 Big-IP LTM and Citrix Netscaler MPX Hardware Load Balancers and ADCs. https://kemptechnologies.com/compare-kemp-f5-big-ip-citrix-netscaler-hardware-load-balancers/, s/d. Último acesso em 22/10/2018. Citado na pág. xiii, 15

Lantz *et al.*(**2010**) Bob Lantz, Brandon Heller e Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. Em *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, página 19. Citado na pág.

Linkletter(2015) Brian Linkletter. Open-Source Routing and Network Simulation. http://www.brianlinkletter.com/using-the-pox-sdn-controller/, 2015. Último acesso em 22/10/2018. Citado na pág.

Linux Virtual Server Knowledge Base(s/d) Linux Virtual Server Knowledge Base. Weighted Least-Connection Scheduling. http://kb.linuxvirtualserver.org/wiki/Weighted_Least-Connection_Scheduling, s/d. Último acesso em 14/05/2018. Citado na pág. 21

Mahler(s/d) David Mahler. Introduction to SDN (Software-defined Networking). https://www.youtube.com/watch?v=DiChnu_PAzA, s/d. Último acesso em 22/10/2018. Citado na pág.

McKeown *et al.***(2008)** Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker e Jonathan Turner. Openflow: Enabling innovation in campus networks. 38:69–74. Citado na pág.

Mininet(s/d) Mininet. Mininet Overview. http://mininet.org, s/d. Último acesso em 22/10/2018. Citado na pág. 11

Moharana(2013) Shanti Moharana. Analysis of load balancers in cloud computing. *International Journal of Computer Science and Engineering*, 2:101–108. Citado na pág. 1, 3

Nielsen(1993) Jakob Nielsen. Response Times: The 3 Important Limits. https://www.nngroup.com/articles/response-times-3-important-limits/, Junho 1993. Último acesso em 24/09/2018. Citado na pág. 25

Ominike *et al.***(2016)** Akpovi Ominike, Ebiesuwa Seun, Adebayo A. O. e F Osisanwo. Introduction to software defined networks (sdn). *International Journal of Applied Information Systems*, 11:10–14. doi: 10.5120/ijais2016451623. Citado na pág.

Open Network Foundation(s/d) Open Network Foundation. OpenFlow Specification version 1.4.0. https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.4.0.pdf, s/d. Último acesso em 22/10/2018. Citado na pág. 9

Pagano(2012) Francesco Pagano. A Distributed Approach to Privacy on the Cloud. *CoRR*, abs/1503.08115. Citado na pág. xi, 13

Sekar(2017) Chandra Sekar. Software Load Balancer Advantages and 9 Reasons to Switch. https://blog.avinetworks.com/9-reasons-to-make-the-jump-to-a-software-load-balancer, Junho 2017. Último acesso em 19/08/2018. Citado na pág. 13

Silva(2016) Guilherme Hiert Silva. Redes Definidas por Software: aplicações práticas, 2016. Monografia (Tecnólogo em Sistemas de Telecomunicações), UTFPR (Universidade Tecnológica Federal do Paraná). Citado na pág. 11

Sreedhar(2018) Pooja Sreedhar. Team NetClaws - Traffic Based Load Balancing using SDN. https://www.youtube.com/watch?v=0eJHf-epyCI, 2018. Último acesso em 22/10/2018. Citado na pág.

Vashistha e Kumar Jayswal(2013) Jyoti Vashistha e Anant Kumar Jayswal. DNS Based Approach Load Balancing In Distributed Web Server System. *IOSR Journal of Engineering*, 3:46–55. Citado na pág.