ENGENHARIA DE TRÁFEGO EM REDES DEFINIDAS POR SOFTWARE PARA DIVIDIR TRÁFEGO DE ENTRADA EM DATA CENTER

ERIK DE BRITTO E SILVA

ENGENHARIA DE TRÁFEGO EM REDES DEFINIDAS POR SOFTWARE PARA DIVIDIR TRÁFEGO DE ENTRADA EM DATA CENTER

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Luiz Filipe Menezes Vieira
Coorientador: Marcos Augusto Menezes Vieira

Belo Horizonte - MG

Agosto de 2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Silva, Erik de Britto e.

S586e Engenharia de tráfego em redes definidas por software para dividir tráfego de entrada em datacenter. / Erik de Britto e Silva. – Belo Horizonte, 2016.
xxiii, 61 f.: il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Luiz Filipe Menezes Vieira.
Coorientador: Marcos Augusto Menezes Vieira.

1. Computação - Teses. 2. Engenharia de tráfego 3. Redes de computadores I. Orientador. II. Coorientador. III.Título.

CDU 519.6*22(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Engenharia de tráfego em redes definidas por software para dividir tráfego de entrada em datacenter

ERIK DE BRITTO E SILVA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LUIZ FILIPE MENEZES VIEIRA - Orientador
Departamento de Ciência da Computação – UFMG

PROF. MARCOS AUGUSTO MENEZES VIEIRA - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. DANIEL FERNANDES MACEDO
Departamento de Ciência da Computação - UFMG

PROF. DORGIVAL OLAVO GUEDES NETO
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 18 de agosto de 2016.

Essa dissertação é dedicada aos meus familiares, principalmente meus avós, pais, irmã e tios, incansáveis motivadores que sempre me apoiaram em todos os momentos.

Agradecimentos

Em primeiro lugar eu gostaria de agradecer ao meu orientador Luiz Filipe Menezes Vieira por toda a liberdade, conselhos e direção que me proporcionou, e ao meu coorientador Marcos Augusto Menezes Vieira, por todo o seu empenho em me ajudar e motivar.

Agradeço a todos os professores do Departamento de Ciência da Computação da UFMG, em especial ao professor Daniel Fernandes Macedo com a sua presença, estímulo e colaboração constantes no Laboratório Winet. Também ao professor José Marcos Silva Nogueira por seu apoio e apreço, e ao professor Dorgival Olavo Guedes Neto pelas direções. Na verdade sou grato a todos os integrantes do corpo docente do Departamento, bem como seus funcionários por seus serviços, dedicação e colaboração.

Além de meus familiares, agradeço também ao valioso apoio dos colegas, pesquisadores e amigos: Alisson Rodrigues Alves, Aloizio Pereira da Silva, Bruno Pereira de Souza, Ewerton Monteiro Salvador, Frederico Martins Biber Sampaio, Gustavo Pantuza Coelho Pinto, Jesse Leandro Leoni, Lúcio Alberto de Resende Júnior, Henrique Duarte Moura, Pablo Goulart Silva, Vinícius Fernandes Soares Mota, Vinicius Fonseca e Silva e Wendley Souza da Silva. Sem a ajuda destes não imagino como seria essa dissertação.

Por fim, agradeço o às instituições de apoio e fomento à pesquisa: Capes, CNPq, Fapemig e Fundep, as quais direta ou indiretamente proporcionaram os recursos para essa dissertação.

"O coração do homem propõe o seu caminho, mas o Senhor lhe dirige os passos."
(Provérbios 16: 9 - Bíblia)

Resumo

O volume global de dados trafegados na Internet tem crescido consideravelmente, resultando no problema de atender esse crescimento nos data centers. Caso esse crescimento não seja atendido as sobrecargas e gargalos resultantes podem causar perda de clientes. Uma solução através de investimentos em infraestrutura pode abranger dois fatores: o aumento da taxa de transmissão dos dados e o aumento da capacidade dos servidores. Com isso, os data centers atenderão a mais usuários e a mais aplicações em um mesmo intervalo de tempo.

A presente dissertação propõe uma solução que emprega os dois fatores acima citados, baseada em Engenharia de Tráfego em um ambiente de Redes Definidas por Software, empregando o protocolo OpenFlow, que pode ser aplicada ao tráfego de entrada em um data center. Redes Definidas por Software também são conhecidas como SDN (Software Defined Networking). O tráfego de entrada das requisições de serviço dos clientes é dividido por enlaces distintos para servidores réplicas, obedecendo a uma política de balanceamento de carga. Por ser uma solução que monitora o volume de dados já trafegados nas portas de um switch OpenFlow, é uma solução simples, não adiciona grande custo computacional e nem os atrasos provenientes do monitoramento da carga em cada servidor. A solução proposta foi implementada em um ambiente físico e o tempo médio das requisições desses serviços ficaram próximos aos de uma solução similar, implementada utilizando *sockets*.

Palavras-chave: Engenharia de Tráfego, SDN, Openflow, Datacenter, Balanceamento de Carga.

Abstract

Internet global data traffic has been growing considerably, creating the problem to meet this growth in data centers. If this growth is not met, resulting overloads and bottlenecks can cause loss of customers. A solution through infrastructure investments might involve two factors: increasing the data transmission rate and increasing the capacity of the servers. Thus, more users and more applications will be serviced in the same time interval in data centers.

This thesis proposes a solution employing the two factors above, based on Traffic Engineering in a Software Defined Networking (SDN) environment using the OpenFlow protocol, which might be applicable to incoming traffic in a data center. Incoming traffic of client service requests is divided into distinct links to replica servers, following a load balancing policy. For being a simple solution that monitors the amount of data already flowed on the ports of an OpenFlow switch, it does not add high computational cost and the delays from monitoring the load on each server. The proposed solution has been implemented in a physical environment where the average duration of these service requests were close to the results of a similar solution implemented using sockets.

Keywords: Traffic Engineering, SDN, Openflow, Datacenter, Load Balancing.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O volume global de dados trafegados a cada ano na Internet tem crescido consideravelmente, conforme se verifica no relatório *Cisco Visual Networking Index* publicado em 2015 [CISCO, 2015b]. O relatório prevê que tal tráfego ultrapassará 1,1 Zettabytes (10^{21} bytes) em 2016, alcançando 2 Zettabytes em 2019. Nos últimos cinco anos esse tráfego foi multiplicado por cinco, com previsão de ser multiplicado por três nos próximos cinco anos.

Quanto ao tráfego global nos dispositivos móveis, há outro relatório publicado pela Cisco em 2016 [CISCO, 2016b], prevendo que o volume total de 3,7 Exabytes por mês em 2015 se multiplicará cerca de oito vezes em cinco anos, atingindo 30,6 Exabytes em 2020, conforme mostrado na Figura 1.1[1] (1 Exabyte = 10^{18} bytes). Em 2014 a Internet possuía 2,8 bilhões de usuários e, segundo prevê o primeiro relatório, atingirá 3,9 bilhões em 2019. O relatório mostra que o tráfego nos dispositivos móveis aumentou 74% entre 2014 e 2015, ou seja, aumentou quatro mil vezes nos últimos dez anos e 400 milhões de vezes nos últimos 15 anos.

Outro relatório da Cisco, sobre o tráfego global na nuvem (*cloud computing*) [CISCO, 2015a], prevê que o tráfego global nos data centers se multiplicará cerca de 3 vezes entre 2014 e 2019, passando de 3,4 Zettabytes para 10,4 Zettabytes, conforme mostrado na Figura 1.2[2]. Portanto, o tráfego dos data centers é o de maior magnitude entre os citados. Data centers tradicionais não conseguem escalabilidade de armazenamento e nem de carga computacional a não ser através da compra e instalação de mais equipamentos. Data centers na nuvem são escaláveis de acordo com a necessidade de seus sistemas, tem capacidade potencialmente ilimitada de crescimento, de acordo com

[1] `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf`

[2] `http://www.statista.com/statistics/267199/global-consumer-internet-traffic-by-region/`

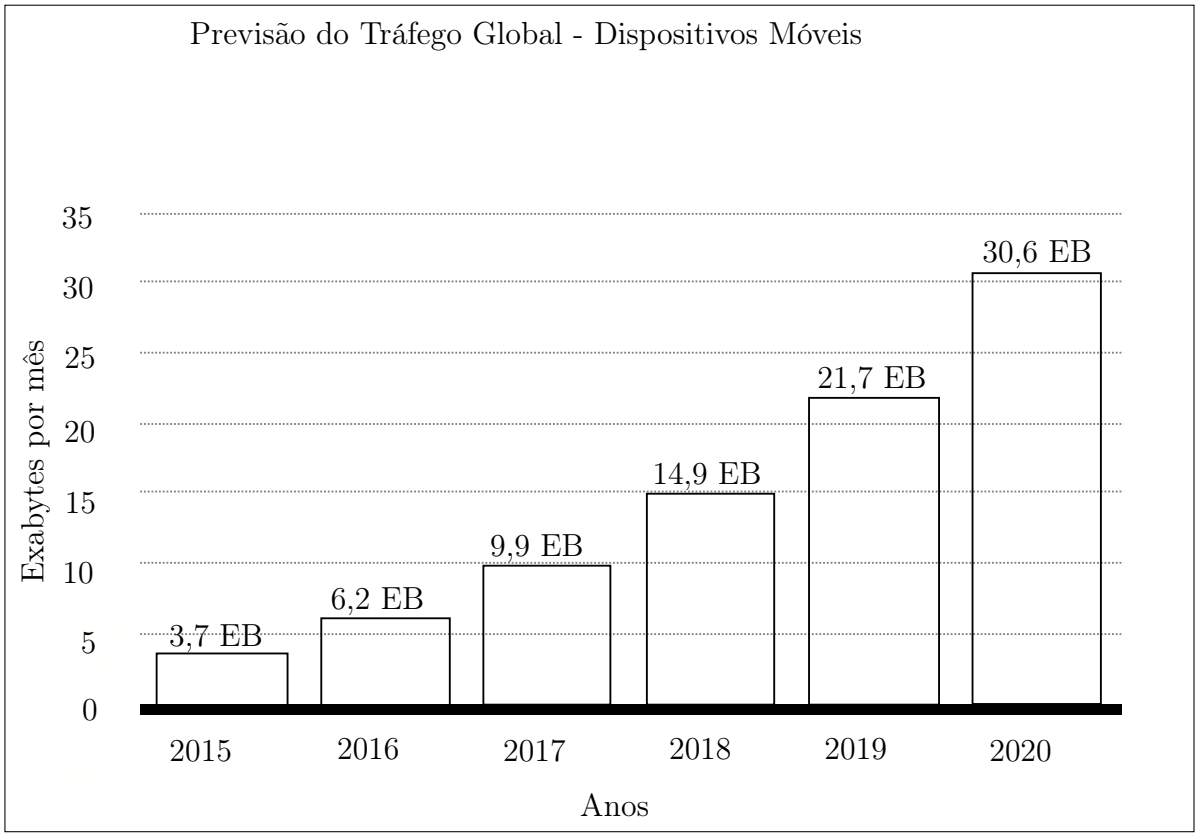

Figura 1.1: Previsão do tráfego global para dispositivos móveis na Internet de 2015 a 2020 - Cisco©

os planos de seus provedores [Van den Bossche et al., 2010].

Conforme o exposto, surge o problema de atender não só o volume futuro de tráfego nos data centers, mas atender também o crescente tráfego atual e suas oscilações, através do aumento da velocidade. Uma solução que aumente a velocidade de atendimento dos data centers poderá abranger dois fatores. O primeiro fator é aumentar a capacidade dos enlaces físicos para aumentar a taxa de transmissão dos dados, porém é de alto custo e não é escalável. Os enlaces físicos podem chegar ao limite de velocidade da tecnologia e serão necessárias outras tecnologias. Uma alternativa é utilizar enlaces paralelos entre origem e destino e empregar um método para a divisão do tráfego total entre os enlaces. O segundo fator é a disponibilização de vários servidores réplica para aumentar a capacidade de execução de mais serviços e mais aplicativos, aumentando a capacidade para atender mais serviços ao mesmo tempo. Segundo Cardellini et al. [2002], o desempenho das aplicações WEB percebido pelos usuários finais está sendo dominado pela latência dos servidores e, ao mesmo tempo, a capacidade da rede está aumentando mais rapidamente do que capacidade dos servidores. Isto indica que o uso combinado destes dois fatores, aumento da velocidade dos enlaces e uso de vários

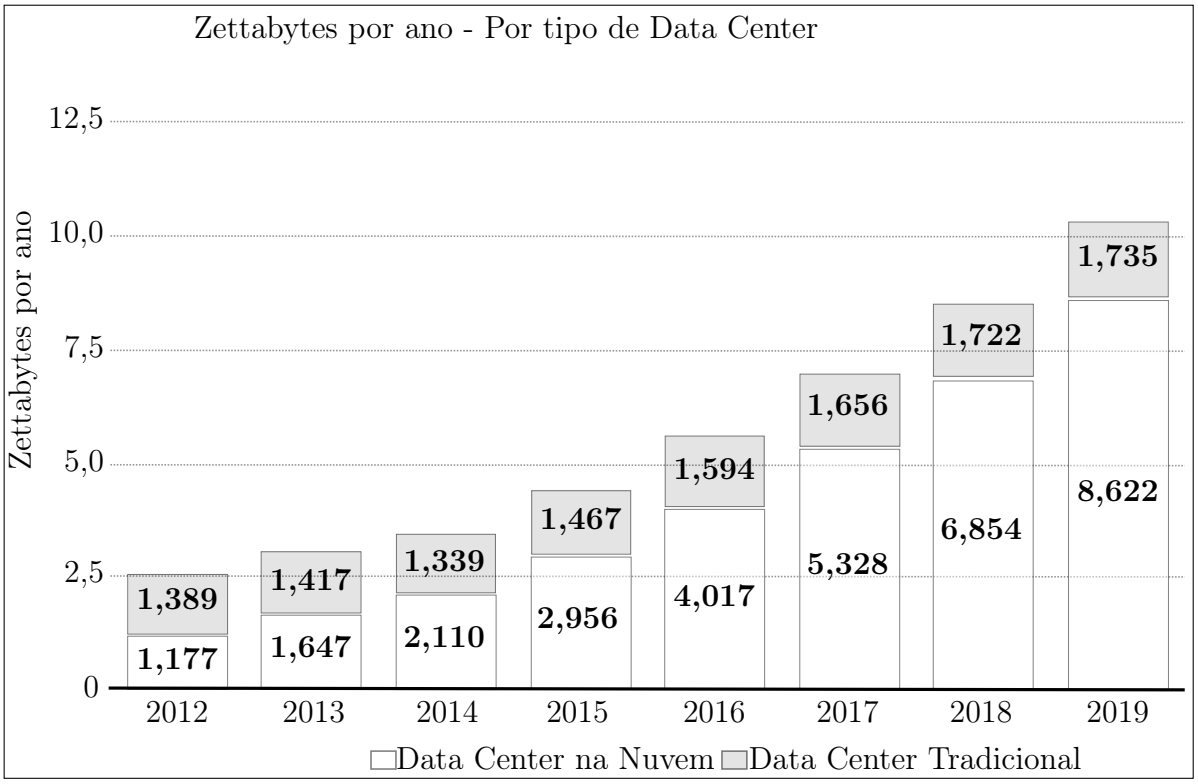

Figura 1.2: Previsão do tráfego global anual por tipo de data center na Internet de 2012 a 2019 - Cisco© e Statista©

servidores réplica, são adequados para se obter uma solução.

É esperado que um data center na Internet seja dimensionado para atender o seu tráfego de pico. Entretanto, mesmo utilizando diversos meios para otimizar o atendimento ao tráfego, data centers são provisionados com capacidade em excesso [Xu et al., 2014]. Dessa forma, fora do horário de pico ocorre o problema da subutilização de recursos [Armbrust et al., 2010], o que causa desperdício de recursos. Empresas menores podem obter soluções escaláveis, como por exemplo o *AWS — Amazon Web Services* da Amazon [Ferraris et al., 2012] que oferece serviços e infraestruturas completa de data center por demanda. Entretanto, tais soluções podem não atender a todos os perfis de instituições. Em alguns casos, as soluções são implementadas através de investimentos em infraestrutura física própria, de maneira semelhante à solução proposta no presente trabalho.

Este trabalho apresenta uma solução para o problema do crescente volume de tráfego de entrada em um data center, com o objetivo de diminuir o tempo de serviço das requisições dos usuários.

1.1 Motivação

A motivação do presente trabalho é propor uma solução para o problema do crescente aumento de tráfego na Internet através da divisão do tráfego de entrada em um data center. A resolução do problema evitará prejuízos de perda de clientes através da falta de atendimento ou excessiva demora na resposta dos serviços Internet fornecidos. Embora se aplique a um data center, mesmo instalações de menor porte podem utilizar a solução.

A primeira motivação é a simplicidade e eficiência da utilização de Engenharia de Tráfego. O serviço de encaminhar um grande volume de tráfego dividido por vários fluxos é um método simples e eficiente de Engenharia de Tráfego, dentre os vários disponíveis [Leduc et al., 2006]. Um método de divisão é chamado de balanceamento de carga. Neste trabalho, o tráfego é considerado a carga de rede a ser distribuída em fluxos por vários caminhos.

Outra motivação é a nova abordagem fornecida pelo paradigma SDN, com várias vantagens mostradas na seção 2.4.1. SDN, através do protocolo OpenFlow, traz melhores abstrações e plataformas para a inovação, permitindo propor novos serviços que antes não eram possíveis com switches comerciais, e não com hardware proprietário e de alto custo [McKeown et al., 2008]. SDN possibilita a seleção dos caminhos pelos enlaces físicos[3], através de switches e roteadores, os elementos que encaminham pacotes de dados nas redes. Estes dispositivos SDN podem realizar dinamicamente a divisão do tráfego pelos diferentes caminhos disponíveis.

1.2 Contribuição

O presente trabalho define e implementa um serviço de Engenharia de Tráfego em uma rede SDN para dividir dinamicamente o tráfego de entrada em um data center através dos enlaces físicos disponíveis. A divisão visa o balanceamento do tráfego de entrada considerando a quantidade de bytes trafegados em cada enlace.

Como contribuição é proposta uma solução que diminui o tempo médio de duração das requisições de serviço divididas entre vários servidores. Ou seja, o tráfego de entrada, com as requisições de clientes, dividido pela Engenharia de Tráfego em ambiente SDN, resulta em um menor tempo médio de duração dessas requisições. Com isso poderão ser atendidas mais requisições em um mesmo intervalo de tempo, contribuição que pode ser aplicada na solução do problema de crescimento do tráfego da Internet.

[3]Os enlaces físicos são os caminhos disponíveis para o tráfego fluir, os termos enlaces físicos e caminhos são utilizados como sinônimos neste trabalho.

Por ser uma solução que monitora o volume de dados já trafegados nas portas de um switch[4], é uma solução simples, não adiciona grande custo computacional e nem os atrasos provenientes do monitoramento da carga em cada servidor.

Ao comparar a solução proposta com uma implementação na qual o controlador OpenFlow lê os valores de carga de bytes transmitidos diretamente nas interfaces de rede de cada servidor réplica, utilizando *sockets* [Kurose, 2005], a solução proposta obteve tempos bem próximos.

Finalmente, abaixo estão as principais contribuições do trabalho:

1. Divide o tráfego através de Engenharia de Tráfego, com algoritmos simples e eficientes.

2. Utiliza o protocolo OpenFlow baseado na arquitetura SDN, com as vantagens citadas na Seção 2.4.1, destacando: maior agilidade devido à programabilidade e baixo custo de compra e operação.

3. Diminui o tempo médio de requisição em um switch comercial e, ao ser comparada com uma solução similar utilizando *sockets* apresenta pequena diferença no desempenho.

4. Pode dividir o tráfego entre switch e servidores réplica e também entre switches em camadas, conforme apresentado na Seção 2.1.1.

Foram escolhidas requisições de serviço HTTP para realizar os experimentos executados. Outros diferentes tipos de serviços também podem ser usados com a mesma solução, sejam serviços sem conexão ou com conexão. A solução pode ser alterada entre estes dois tipos de serviços que são o UDP (*User Datagram Protocol*) e o TCP (*Transmission Control Protocol*), respectivamente. Ganhos da solução deste trabalho podem se aplicados beneficiando qualquer tipo de tráfego de dados, inclusive as soluções de Vídeo por Demanda (VOD — *Video On Demand*). Há uma previsão que, em 2019, o tráfego de Vídeo por Demanda será responsável por 80% do Tráfego Global da Internet [CISCO, 2015b].

1.3 Estrutura da Dissertação

O restante deste documento está organizado da forma a seguir. No Capítulo 2, são apresentados alguns fundamentos visando o entendimento do presente trabalho. O Ca-

[4]Switch ou comutador de rede: Dispositivo que realiza a comutação de quadros para a camada de redes do modelo OSI e realiza o roteamento de pacotes para a camada de transporte em uma rede local de computadores.

pítulo 3 apresenta alguns trabalhos relacionados. Características relevantes dos mesmos são comparadas com a solução proposta no presente trabalho, utilizando ou não Engenharia de Tráfego e a arquitetura SDN. No Capítulo 4 é apresentada a solução, que utiliza Engenharia de Tráfego em ambiente de SDN para a divisão de tráfego de entrada em data center. O ambiente experimental, implementado com a solução proposta, e os resultados obtidos são apresentados no Capítulo 5. Finalmente, o Capítulo 6 conclui a dissertação e apresenta a discussão sobre os resultados obtidos, a contribuição do trabalho e sugestão de trabalhos futuros.

Capítulo 2

Fundamentos

No presente capítulo são apresentados alguns fundamentos sobre data centers, Engenharia de Tráfego, Balanceamento de Carga e de SDN, necessários para o entendimento desta dissertação. Também é apresentado o protocolo OpenFlow, uma interface padronizada para controle de redes seguindo a arquitetura SDN.

2.1 Data Centers

Pode-se definir data centers como instalações físicas onde vários servidores e equipamentos de comunicação estão presentes devido às suas necessidades ambientais e de segurança comuns, bem como a facilidade de manutenção [Barroso et al., 2013].

Com o objetivo de facilitar o entendimento do ambiente físico, é adotada uma caracterização de data center denominada Duas Camadas (*Two Tier*) [Al-Fares et al., 2008]. O data center é representado apenas por Roteadores, switches Núcleo (*Core*) - formando a primeira camada, switches Borda (*Edge*) - formando a segunda camada, servidores e conexões entre esses elementos, conforme exibido na Figura 2.1.

No modelo em Duas Camadas os roteadores são responsáveis pelas conexões do data center com a Internet e se conectam a vários switches Núcleo. Cada switch Núcleo se conecta a vários switches Borda. Cada switch Borda se conecta a vários servidores réplica. Cada Servidor se conecta a um único switch Borda. O tráfego externo ao data center com os clientes na Internet, proveniente dos roteadores, é denominado tráfego de entrada. O tráfego entre os servidores é chamado de tráfego interno. Essas duas camadas possibilitam que os servidores comuniquem-se entre si e com a Internet, através de suas interfaces de rede. Como podemos observar, fornecem também caminhos redundantes. Os switches Borda são também conhecidos como *ToR (Top of Rack)* — conectam vários servidores em um mesmo rack. Todos os switches Borda se conectam

à estrutura de rede do data center através dos switches Núcleo. Os switches Núcleo têm taxa de transferência maior do que os switches Borda, por exemplo, os switches Borda têm taxa de 1 Gb/s e os switches Núcleo 10 Gb/s [Benson et al., 2010].

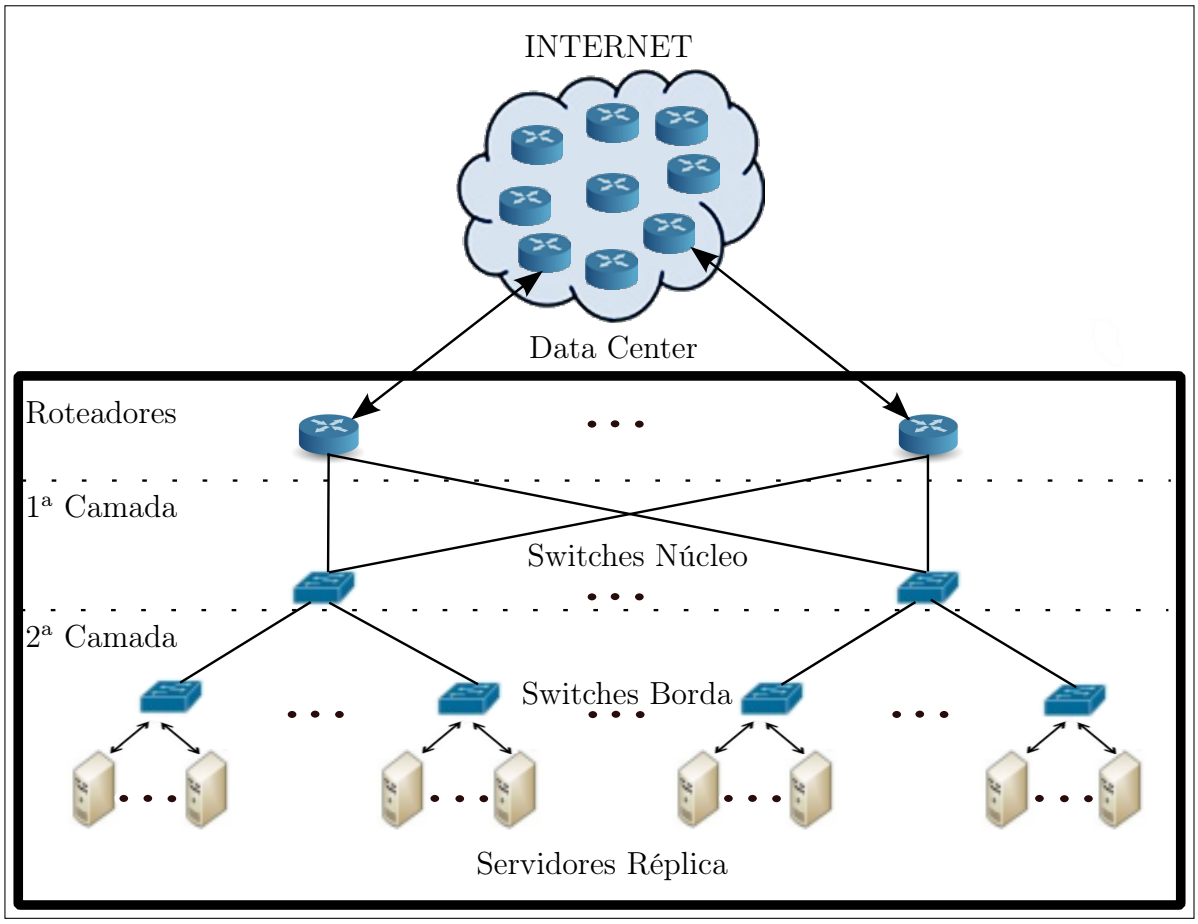

Figura 2.1: Caracterização de data center de duas camadas (*Two Tier*)

2.1.1 Região de Interesse

A região de interesse do presente trabalho, considerada como parte de um data center em duas camadas, é formada pelos switches Borda e os servidores Réplica com o tráfego de entrada a ser encaminhado. A região também pode ser formada pelos os switches Núcleo e os switches Borda [Al-Fares et al., 2008], com um tráfego de entrada. Assim, constrói-se um elemento básico que é um switch submetido ao tráfego de entrada a ser dividido para os servidores réplica ou a outros switches, que pode ser visualizado como: Tráfego de Entrada -> switch -> servidores Réplica ou Tráfego de Entrada -> switch -> outros switches No switch a solução do trabalho realiza a divisão do tráfego, proveniente de uma entrada.

2.2 Engenharia de Tráfego

Engenharia de Tráfego é a adaptação do tráfego às condições da rede, com os objetivos conjuntos de simplicidade, bom desempenho para o usuário e uso eficiente dos recursos da rede [Awduche et al., 2002]. Isto é, busca uma distribuição ótima do tráfego por todos os caminhos ou recursos da rede. Um dos objetivos da Engenharia de Tráfego, em uma rede como a Internet, é o de eliminar ou minimizar as situações de perdas de pacotes em grande escala que diminuem a vazão e aumentam o atraso da entrega de pacotes, decorrentes de altos volumes de tráfego. Com a Engenharia de Tráfego podemos controlar parâmetros de fluxos tais como Qualidade de Serviço (*QoS — Quality of Service*) e realizar gerenciamento de banda (*traffic shaping*).

O tráfego de entrada em um Sistema Autônomo (*AS - Autonomous System*) em um Provedor de Serviços Internet (*ISP - Internet Service Provider*), por exemplo, possui grande elasticidade [Lim & Tang, 2013], podendo passar por enormes variações dependendo de horários e outras condições. O mesmo se aplica aos data centers. Uma solução de Engenharia de Tráfego nos períodos de pico, por exemplo, pode suavizar ou evitar o congestionamento na entrada, resultar em maior utilização dos enlaces físicos dessa entrada, fornecer escalabilidade e aumentar a robustez, atingindo também um maior aproveitamento econômico da estrutura de comunicação. A economia de energia também pode ser realizada ao desativar ou diminuir o consumo dos enlaces em momentos de inatividade ou de baixo volume de tráfego, de acordo com as variações do mesmo.

Entre diversas soluções de Engenharia de Tráfego já pesquisadas, podemos citar: protocolos de roteamento intradomínio [Fortz et al., 2002], alteração das regras do protocolo BGP (*Border Gateway Protocol*) entre domínios [Feamster et al., 2003], balanceamento de carga entre roteadores [Wang et al., 2011a], utilizando MPLS (*Multiprotocol Label Switching*) [Barabas et al., 2011], e até mesmo soluções que verificam a ocupação dos *buffers* nos roteadores em Redes Definidas por Rádio [Farmanbar & Zhang, 2014].

2.3 Balanceamento de Carga

O balanceamento de carga é um método de Engenharia de Tráfego que divide uma demanda ou carga por vários recursos computacionais [Rodrigues et al., 2015]. Tal balanceamento pode ser feito por destino, por fluxo ou por pacote [CISCO, 2016a; Augustin et al., 2007]. O balanceamento por destino é realizado distribuindo todos os pacotes para um mesmo destino através de uma mesma interface de saída. O balan-

ceamento por pacote é realizado distribuindo os pacotes por diferentes interfaces de saída, e finalmente, o balanceamento por fluxo é realizado distribuindo cada fluxo por uma única interface de saída. No balanceamento por pacotes, há o custo adicional de reordenação dos mesmos no destino. O balanceamento por fluxo é realizado através de uma quíntupla que identifica cada fluxo: endereço IP origem, endereço IP destino, protocolo (UDP ou TCP), porta de origem e porta de destino. No presente trabalho a divisão de um volume de tráfego (a carga) é realizada por fluxos que são distribuídos por caminhos distintos, cada um levando a um único servidor réplica, que fornece o recurso para alimentar a carga. Tal divisão em fluxos segue uma política de acordo com um determinado critério.

Políticas de balanceamento de carga podem ser criadas para considerar diversas variáveis, alguns exemplos: horários, eventos, clientes, robustez, segurança, destinos, serviços e máquinas; através delas todo o fluxo de uma rede pode ser dividido pela Engenharia de Tráfego.

No presente trabalho, o tráfego a ser dividido entre os servidores réplicas origina-se em estações clientes que fazem requisições a um serviço HTTP em um endereço IP virtual. O modo utilizado para encaminhar cada fluxo aos servidores réplica é similar ao de um sistema Web *cluster* ou sistema Web baseado em *cluster* [Cardellini et al., 2002]. Tal sistema encaminha os fluxos das requisições HTTP de cada cliente reescrevendo o endereço IP virtual para o endereço IP interno de um servidor réplica através do switch. O respectivo fluxo da resposta do servidor também tem o seu endereço IP interno reescrito para o endereço IP virtual no switch ao ser encaminhado para o cliente. Outra técnica que pode ser utilizada, para a divisão do tráfego, é a de configurar o mesmo endereço IP em todos os servidores réplicas, e encaminhar cada fluxo ao endereço físico da camada de enlace de cada servidor.

2.4 SDN e OpenFlow

Nesta seção são descritos a arquitetura de redes SDN e o protocolo OpenFlow baseado nesta arquitetura.

2.4.1 SDN

SDN é um novo paradigma de arquitetura de redes que separa o plano de controle e o plano de encaminhamento (ou plano de dados) em uma rede de computadores. Esta separação funcional traz enormes benefícios tornando os sistemas de roteamento mais gerenciáveis, menos complexos e mais adaptáveis às necessidades de novos serviços.

Uma SDN, representada pela rede local na Figura 2.2, é composta por dois componentes principais:

1. **Controlador SDN** - onde ficam as funções logicamente centralizadas. Uma rede pode ser controlada por um ou mais controladores, que determinam o caminho a ser utilizado para cada fluxo de dados da rede.

2. **Elemento de Encaminhamento SDN** - é o plano de dados da rede. A lógica para encaminhamento dos pacotes é determinada pelo controlador SDN e implementada na tabela de encaminhamento deste elemento. Este elemento é o **switch SDN**.

Além de trazer inovações e proporcionar novas soluções para os problemas em redes [McKeown et al., 2008], a arquitetura adiciona as seguintes vantagens [Valdiviveso Caraguay et al., 2013]:

1. Independência de fabricante - a rede não é mais proprietária.

2. Maior agilidade - o controle da rede é facilmente programável.

3. Substitui os equipamentos de hardware dedicado (*hardware appliances*) por equipamentos baseados em software (*software appliances*).

4. Facilita a virtualização da rede.

5. Os equipamentos são mais baratos.

6. Reduz gastos operacionais.

A maior parte das desvantagens de SDN são inerentes ao mercado de novas tecnologias, por ter apenas 8 anos [McKeown et al., 2008] desde a publicação do protocolo OpenFlow até a data de publicação deste trabalho. A própria maturação do mercado e disseminação da tecnologia, poderá eliminar tais desvantagens. Porém, permanece a desvantagem de se programar dinamicamente a rede, o que resulta em um atraso para se estabelecer uma nova regra devido a um novo fluxo. Deve-se verificar se este atraso compromete o desempenho em cada solução implementada. Grandes atrasos podem invalidar a implementação, pois somente após o estabelecimento da regra no switch os pacotes desse fluxo podem transitar ou não. Tal atraso é a soma dos tempos de: Ida da informação do novo fluxo detectado no switch SDN ao controlador SDN + Processamento dessa informação no controlador SDN + Retorno para o switch SDN da nova regra a ser gravada + Gravação da nova regra no switch SDN + Início do novo fluxo encaminhado pelo switch SDN.

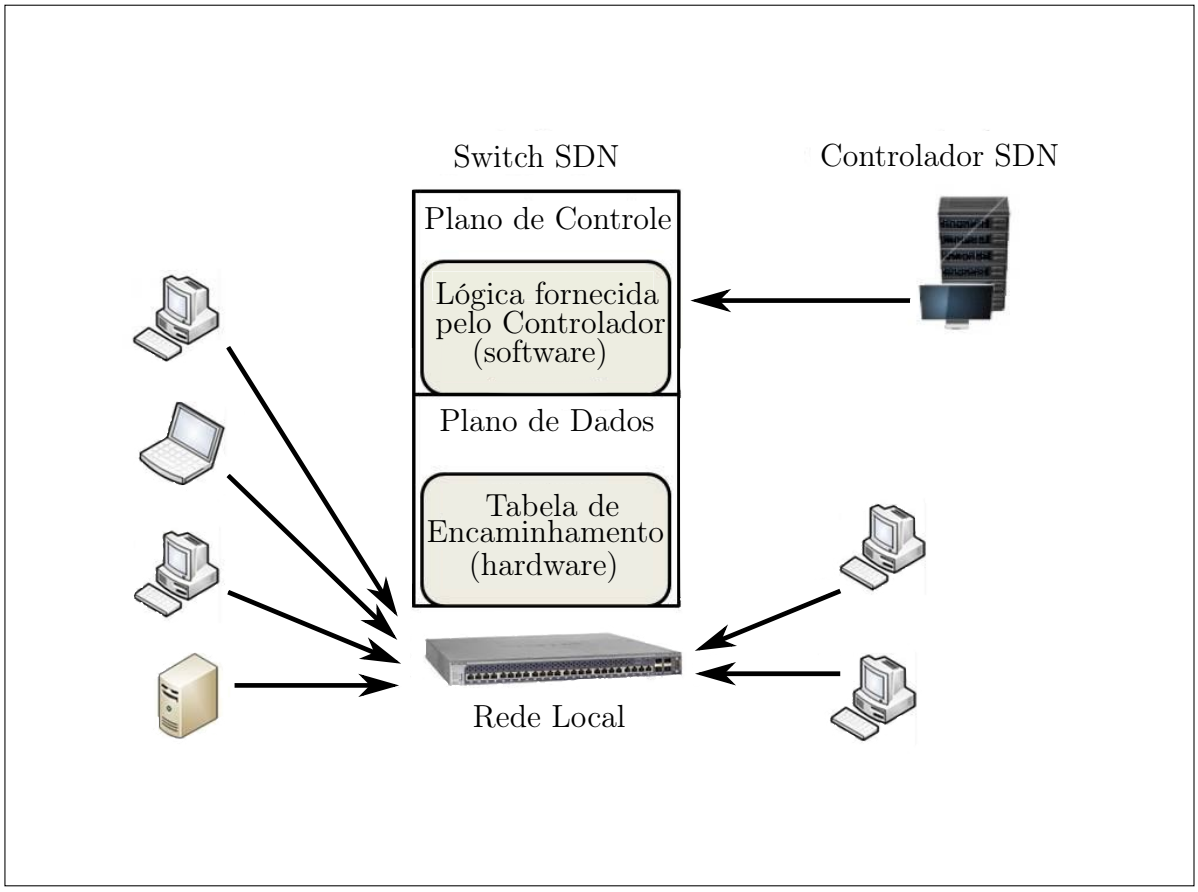

Figura 2.2: Exemplo de Redes Definidas por Software - SDN

Para eliminar este atraso, procura-se instalar as regras antes das ocorrências de novos fluxos, desde que sejam previsíveis, assim denominadas regras proativas. Tais regras são instaladas na inicialização do switch. As regras que são instaladas pelo controlador após a ocorrência de um novo fluxo são denominadas regras reativas.

2.4.2 OpenFlow

O protocolo OpenFlow [McKeown et al., 2008], baseado no paradigma de SDN, é uma interface padronizada, utilizada por um controlador OpenFlow para se comunicar com um switch OpenFlow. É apoiado por várias empresas participantes da Open Networking Foundation [ONF, 2016], organização que mantém o protocolo OpenFlow. Este protocolo foi originalmente criado na Universidade de Stanford [McKeown et al., 2008], como o primeiro padrão de ambiente SDN. Em um ambiente OpenFlow, são válidas as correspondências abaixo:

- Cada controlador OpenFlow é um controlador SDN e vice-versa.

- Cada switch OpenFlow é switch SDN e vice-versa.
- Uma rede OpenFlow é composta por 1 ou mais switches OpenFlow conectados a: controladores OpenFlow, estações, servidores e outros elementos de rede, portanto é um ambiente SDN.

Quando um fluxo de dados é iniciado em uma rede OpenFlow, a seguinte sequência de ações ocorre:

1. Internamente no switch é verificado se alguma regra existente engloba esse fluxo, se existir executa a regra, se não existir é executada a sequência abaixo.
2. O primeiro pacote do fluxo é enviado pelo switch OpenFlow ao controlador OpenFlow.
3. O controlador OpenFlow decide como encaminhar o fluxo ou até mesmo descartá-lo.
4. O controlador OpenFlow envia a regra apropriada para ser instalada na tabela de encaminhamento de cada switch OpenFlow.
5. Todos os pacotes subsequentes, daquele fluxo, são tratados como descrito na regra recém instalada, e passam a não mais necessitar de qualquer ação do controlador (Plano de Controle).

O controlador OpenFlow pode ser responsável, entre outras atividades, pela seleção dos caminhos. É possível implementar essa função de gerenciamento de tráfego em modo centralizado ou parcialmente centralizado no controlador OpenFlow [Jain et al., 2013]. O uso do protocolo OpenFlow permite controlar diretamente os fluxos possibilitando que parâmetros de gerência de tráfego (banda, caminhos, *QoS*, etc.) sejam implementados. O objetivo do presente trabalho é dividir o tráfego (ou fluxo) de entrada por vários caminhos.

2.4.2.1 Controladores OpenFlow

A primeira especificação do protocolo OpenFlow foi implementada através do controlador NOX. A implementação de um controlador NOX é através de código na linguagem C++ e Python. Com o objetivo de atender à demanda dos usuários que utilizam Python para projetar aplicações para o NOX, foi criado o ambiente POX para confecção de controlador totalmente na linguagem Python. O controlador POX foi criado com base do modelo NOX, com o objetivo de substituir o controlador NOX em C++.

Contudo, o POX é adequado aos casos em que o desempenho não seja um requisito crítico [Guedes et al., 2012]. Facilita a prototipação rápida e desenvolvimento de aplicações de teste para OpenFlow.

O ambiente POX fornece uma API para controle de switches OpenFlow por controladores OpenFlow. O controlador e o switch trocam mensagens. Estas mensagens são geradas no switch em função de eventos (como por exemplo, a chegada do primeiro pacote de um novo fluxo). Podem ainda ser enviadas ativamente pelo controlador, solicitando informações do switch e demandando que ele realize uma determinada ação.

Além do POX [Rodrigues Prete et al., 2014] ser adequado para pesquisa e desenvolvimento de protótipos. É uma plataforma com o protocolo OpenFlow, com as seguintes características:

1. Plataforma para linguagem Python, orientada a eventos.

2. Disponível no simulador Mininet [de Oliveira et al., 2014].

3. Facilmente instalável e configurável em plataformas Linux.

4. O código também é aberto, além de rodar em sistemas abertos.

5. Pode controlar dispositivos físicos reais da rede tais como switches OpenFlow.

A desvantagem do controlador POX é sua lentidão [Erickson, 2013], porém as soluções que obtêm sucesso neste controlador devem obter sucesso em controladores mais rápidos.

O POX utiliza a versão do protocolo OpenFlow 1.0 que é a versão disponível na maioria dos switches OpenFlow existentes. Através do parâmetro *openflow.of_01*, faz-se com que o switch passe a operar nesta versão. Isso permite utilizar o controlador POX em um grande número de switches OpenFlow. Abaixo estão descritos outros controladores que também utilizam o protocolo OpenFlow [Guedes et al., 2012]:

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

2. FloodLight: FloodLight 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 gerencia a entrada e saída dos

switches OpenFlow, rastreia a localização dos dispositivos na rede e fornece suporte para integração em redes não OpenFlow.

3. Ryu: O controlador Ryu 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.

4. Pyretic: Pyretic 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.

5. Maestro: Maestro é um controlador extensível baseado na linguagem Java disponibilizado pela Universidade de Rice. É direcionado para pesquisadores e suporta *multi-threading*.

6. OpenDaylight: OpenDaylight é um projeto de software aberto da Linux Foundation com o objetivo de promover a adoção e a inovação de SDN através da criação de uma plataforma comum suportada pela indústria. Emprega uma arquitetura baseada em micro serviços e utiliza a linguagem Java. Suporta vários protocolos emergentes e tradicionais tais como NETCONF e BGP [OpenDaylight, 2016].

7. Onos: Open Network Operating System — Sistema Operacional Aberto de Redes, é o primeiro sistema operacional de rede de código aberto na arquitetura SDN, voltado especificamente para redes de provedor de serviços e de missão crítica. Seu propósito é fornecer alta disponibilidade, escalabilidade e o desempenho que essas redes demandam. Além disso, Onos tem criado abstrações *Northbound* e APIs para facilitar o desenvolvimento de aplicações, abstrações *Southbound* e interfaces para o controle de redes com dispositivos OpenFlow e não OpenFlow [Onosproject, 2016].

Conforme o exposto, métodos de Engenharia de Tráfego podem ser implementados em um controlador de SDN POX, no protocolo OpenFlow, através de programas em Python, para obter um serviço simples, flexível e eficiente para a divisão de grandes volumes de tráfego, em vários fluxos, pelos caminhos disponíveis.

2.5 Conclusão

Nesse capítulo foi definido o que é um data center e foi exibida uma visão geral de Engenharia de Tráfego, SDN e protocolo OpenFlow. Essas definições objetivam um melhor entendimento deste trabalho.

As vantagens do uso de Engenharia de Tráfego, tais como simplicidade, bom desempenho e eficiência, validam a sua utilização. As principais vantagens de SDN

entre as citadas na Seção 1.1 são: menor complexidade e encaminhamento programável dinamicamente. O método de balanceamento de carga da Engenharia de Tráfego visa otimizar o uso dos recursos disponíveis, maximizar a vazão dinamicamente e minimizar a sobrecarga de um único recurso.

Dois fatos indicam a necessidade de um novo paradigma de arquitetura para as soluções de Engenharia de Tráfego. Primeiramente, a necessidade de uma arquitetura que possa diferenciar vários tipos diferentes de tráfego de diferentes aplicações e de fornecer um serviço adequado e específico para cada tipo de tráfego, em um período de tempo muito curto, da ordem de milissegundos. Em segundo lugar, para atender ao rápido aumento da computação em nuvem, uma gerência de rede deve melhorar a utilização dos recursos para maior desempenho do sistema [Akyildiz et al., 2014]. Conforme apresentado na Seção 2.4.1, SDN é uma arquitetura de redes que tem a capacidade de atender essas necessidades [Agarwal et al., 2013].

Capítulo 3

Trabalhos Relacionados

Nesse capítulo são discutidos alguns trabalhos relacionados. Tais trabalhos são relevantes por empregarem soluções com algumas características que podem ser comparáveis à solução proposta.

3.1 Trabalho 1 - Engenharia de Tráfego em Rede Não SDN

A Engenharia de Tráfego vem evoluindo desde quando começou a lidar com o tráfego de ligações telefônicas [Akyildiz et al., 2014], portanto é uma área antiga se comparada com o paradigma de SDN [McKeown et al., 2008]. Em redes, a divisão de fluxos é um problema similar ao de manter a Qualidade de Serviço (*QoS - Quality of Service*). Podemos entender que a QoS depende da existência de um fluxo garantido. Tal fluxo poderá ou não ser distribuído por diferentes enlaces.

Existe um trabalho de Engenharia de Tráfego, em rede utilizando os padrões DIFFSERV (*Differentiated Services*) e MPLS (*Multiprotocol Label Switching*), similar a este, que lida com *QoS* num ambiente de testes com TCP/IP [Akyildiz et al., 2003]. É citado que o tráfego de pacotes em MPLS, ao ter o LSP (*Labeled Switch Path*) marcado como mais importante, prioriza e direciona tais pacotes marcados pelos switches, criando túneis com rotas estáticas que não escalam. Na solução proposta, a divisão dos fluxos é ajustada dinamicamente entre os enlaces e pode escalar aumentando a carga em cada enlace. Em MPLS é exigida a disponibilização de rotas de backup, mesmo com Engenharia de Tráfego, para manter a *QoS*. A vantagem da Engenharia de Tráfego em SDN é a agilidade de poder ser programada para isolar os enlaces em falha. SDN também adiciona escalabilidade e eficiência a *QoS*, cada pacote é encaminhado através

de um fluxo.

No DIFFSERV, há uma reserva de banda para cada classe diferente de tráfego, por exemplo, para cada *QoS*. Isto adiciona o custo de se manter um registro da quantidade de banda que está disponível para cada tráfego, durante todo o tempo por todos os roteadores da rede. Os *LSP* com Engenharia de Tráfego para garantir a reserva de banda são denominados DIFFSERV-TE LSP. Da mesma maneira não escalam bem como a solução de SDN aqui apresentada. Adicionalmente, se uma classe tem seu tráfego reduzido ou é removida, não há a liberação da banda reservada para as outras classes de tráfego, em tempo real.

No caso do DIFFSERV, o protocolo IGP (*Interior Gateway Protocol*) anuncia a alocação de banda para cada classe de tráfego e as *BC (Bandwidth Constraints)*, de uma maneira mais complexa e de propagação mais lenta do que em SDN. Se o DIFFSERV utilizar o MAM (Modelo de Alocação Máxima) há um desperdício de banda, pois todas as classes de tráfego são isoladas e a reserva de banda para cada uma, por ser garantida, não pode ser redividida ou compartilhada. Porém redes DIFFSERV utilizando MAM são de fácil entendimento e gerência. Se o DIFFSERV utilizar o *RDM* (Modelo de Alocação de Banda *Russian Dolls*), as classes de tráfego compartilham a banda no modelo do melhor esforço, ou seja, faz-se necessária uma preempção para garantir que uma classe de tráfego tenha uma banda garantida mínima, do contrário ela pode consumir a banda máxima disponível em detrimento das outras classes. A vantagem da presente solução deve-se ao fato que SDN é dinâmica e não passa pelas limitações que existem tanto em MPLS quanto em DIFFSERV.

3.2 Trabalho 2 - Engenharia de Tráfego com SDN

A solução B4 na rede *WAN* da Google [Jain et al., 2013], interliga cerca de uma dúzia de data centers desta empresa distribuídos em alguns continentes empregando Engenharia de Tráfego com SDN. Switches comerciais foram modificados para funcionarem com o protocolo OpenFlow, que sob um controle centralizado são programados dinamicamente para encaminhar o tráfego entre tais data centers. O controle visa manter uma alta utilização dos enlaces, prioriza alguns tipos de tráfegos e tolera perdas, reescalonando os fluxos perdidos.

A solução proposta no presente trabalho é de baixo custo, com perfil de atuação em tempo real, eficiente e escalável, em redes que possuem enlaces para dividir um tráfego de entrada. Parte da motivação originou-se devido ao sucesso da B4 que é uma das primeiras e grandes aplicações de Engenharia de Tráfego com SDN. Embora a di-

mensão da presente solução seja em uma escala menor de quantidade, de complexidade e de valores, objetiva-se também conseguir as vantagens alcançadas com sucesso em B4, através de Engenharia de Tráfego em SDN.

Algumas características consideradas comuns aos dois trabalhos:

- Maximizar a utilização dos enlaces.
- Utilizar Engenharia de Tráfego para manter fluxos dinamicamente divididos.
- Obter uma rede mais simples e mais eficiente através do emprego de protocolos, funcionalidades de gerenciamento e monitoramento mais avançadas.
- Utilizar Engenharia de Tráfego centralizada.
- Utilizar switches comerciais para SDN (no caso da Google, estes são modificados por ela).
- Utilizar um algoritmo da Engenharia de Tráfego para a divisão em fluxos com critério de justiça (no presente trabalho é divisão segundo o volume trafegado).
- Separar o plano de dados do plano de controle (SDN).

Embora várias vantagens sejam similares, existem algumas diferenças quanto às desvantagens. Embora B4 tolere perda de pacotes e maior atraso para algumas aplicações, a solução proposta exclui as situações onde ocorrem a perda de pacotes e considera somente a carga em cada enlace para o experimento. A latência do estabelecimento de um novo fluxo no switch OpenFlow é uma desvantagem inerente à arquitetura SDN, mas em termos de seus valores em uma rede local, pode ser considerada desprezível quando comparada com alguns serviços disponíveis. Deve-se observar que a latência não pode ser evitada em vários serviços na Internet. Na solução proposta a perda de pacotes é um problema não tolerável a não ser em escala muito pequena. A realocação dinâmica de banda que ocorre ao falhar um enlace na Rede B4, por razões de desempenho dos experimentos, não é implementada no trabalho.

3.3 Trabalho 3 - Balanceamento de Carga Híbrido com SDN

Uma solução denominada DUET [Gandhi et al., 2014], utilizando o paradigma SDN, proporciona resultados relevantes. Na DUET, para gerenciar data center provendo serviços na nuvem, é realizado o balanceamento do tráfego nos servidores. Utiliza um

ambiente híbrido, formando por switches comuns não SDN e de switches em software [Costa et al., 2016], para garantir a redistribuição das rotas em caso de falhas. Os switches comuns, via API disponível, são programados como um multiplexador físico através de suas tabelas de ECMP (*Equal-cost Multi-Path*) e de tunelamento. Há um controlador DUET monitorando a carga em cada switch e a topologia da rede. Ele calcula as rotas pela rede e distribui a programação das rotas de ECMP e de tunelamento para os switches físicos e os de software. Desta maneira é feito o balanceamento. Assim para cada conjunto de endereços de clientes de serviços, há uma tabela que os divide entre os switches físicos. Mudanças de carga e de topologia são monitoradas e recalculadas, levando o controlador DUET a reprogramar as tabelas dos switches físicos e de software. Os switches de software recebem atribuições de endereços virtuais dos servidores (IP e porta) e os direcionam a um conjunto de switches físicos e de software. DUET distribui a carga entre os IPs virtuais nos vários switches de software, que mapeiam alguns IPs Virtuais entre si. Cada switch de software, a seguir, mapeia o seu endereço virtual, alvo dos clientes, para um conjunto de IPs de switches físicos da rede interna.

O emprego do DUET que se destina a um conjunto de switches dentro de um data center, também motiva a solução proposta com alguns de seus resultados: alta capacidade e baixa latência, alta disponibilidade e a atribuição de rotas que se adaptam dinamicamente a variações nos padrões de tráfego e as falhas da rede. Ao comparar o DUET com soluções inteiramente baseadas em switches de software para data centers, o mesmo fornece dez vezes mais capacidade do que um balanceador de carga em software, com a latência também reduzida em dez vezes. Custando apenas uma fração do custo do balanceador em software e ainda se adapta rapidamente à dinâmica da rede, inclusive às falhas. Em comparação, no presente trabalho não há redundância de switches para alta disponibilidade.

3.4 Trabalho 4 - Balanceamento de Tráfego através de Programação Proativa com SDN

Há uma solução de balanceamento de carga com switches OpenFlow, que não insere regras para cada pacote de novos fluxos. Tal solução utiliza programação proativa que insere na inicialização de um switch OpenFlow regras coringa (*wildcard*), de todos os possíveis prefixos dos endereços IPs dos clientes, para os enlaces de um data center. Assim, os possíveis fluxos já estão divididos entre os vários caminhos para os servidores réplicas [Wang et al., 2011b]. Para tal é utilizada uma tabela *hash*. Desse modo a

distribuição da carga é feita através de caminhos previamente escolhidos. Com isto poucos pacotes são direcionados ao controlador, resultando em um mínimo impacto na vazão da rede. As requisições de serviços dos clientes são encaminhadas diretamente a um servidor réplica já designado. Há um algoritmo para isso e outro para adaptar as regras a novos pesos de balanceamento dinamicamente. A solução proposta poderia montar um mapa de origens e distribuí-los por vários switches para o balanceamento de tráfego de entrada, e nos últimos switches realizar o balanceamento dinâmico. Tal programação proativa não na solução proposta, pois realiza o balanceamento dinâmico, que é realizado segundo a quantidade total de bytes trafegados em cada porta de um switch.

3.5 Trabalho 5 - Balanceamento de Tráfego Web em Rede Não Estruturada com SDN

O balanceamento de tráfego denominado *Plug-n-Serve*, é realizado dentro de uma rede local composta de switches Openflow implementado em ambiente real [Handigol et al., 2009], no prédio de um departamento de uma universidade. O *Plug-n-Serve* acompanha a localização e a carga de servidores réplicas na rede local, que podem estar desligados e serem religados em diferentes switches ao acaso, também monitora o congestionamento da rede através de medições de latência. Assim, a cada novo fluxo da rede para um servidor, a solução faz o direcionamento do mesmo pelos caminhos com menor congestionamento (menor latência), até o servidor disponível com a menor carga.

Tal solução obteve algumas vantagens de balanceamento do tráfego similares às do presente trabalho. Porém algumas características são diferentes: considera o congestionamento da rede ao monitorar a latência e encaminha os pacotes pelos caminhos com menor tempo de resposta. O uso da latência é um parâmetro que pode ser adicionado futuramente, no presente trabalho, como peso para a escolha do melhor caminho até um servidor réplica.

3.6 Trabalho 6 - Balanceamento de Carga Web em SDN

O trabalho de Avaliação de Balanceamento de Carga Web em Redes Definidas por Software [Rodrigues et al., 2015], considera como carga a ser balanceada a utilização de CPU (*Central Processing Unit*) de cada servidor web réplica. O trabalho apresenta

diversas características semelhantes à solução proposta, mas com três características diferentes. A primeira é que utiliza em seus experimentos um *Open vSwitch* [Pfaff et al., 2015] em ambiente virtual, ao passo que na solução proposta o ambiente de experimentos é totalmente composto por dispositivos reais. A segunda diferença é que a carga a ser considerada é o tráfego em cada porta do switch real (cada porta corresponde ao enlace que conecta um único servidor réplica) e não a carga de CPU. A terceira é como a carga é obtida, sendo esta através da leitura de estatísticas presentes nos switches OpenFlow, conhecida como mensagem de "*Portstats*". Ao passo que no trabalho citado as estatísticas do switch são obtidas através de *sockets*, por meio de uma rede externa adicional que conecta o controlador OpenFlow a cada servidor réplica, individualmente. A leitura das estatísticas de todas as portas do switch OpenFlow pelo controlador possui um tempo de atraso diferente da leitura de cada carga de CPU em cada servidor réplica. Portanto espera-se que a solução do presente trabalho seja mais rápida, mais simples e mais econômica por não utilizar uma rede adicional.

3.7 Tabela de Comparação com os Trabalhos Relacionados

Na Tabela 3.1 são exibidas definições de características que são utilizadas para comparar o presente trabalho com os trabalhos relacionados.

Tabela 3.1: Definição das características comparadas

CARACTERÍSTICA	DEFINIÇÃO
Reserva estática de banda	A banda disponível é dividida em faixas estáticas de menores bandas
Tempo real	A resposta do sistema à variação é considerada rápida, menor que 1 segundo
Escalável	A capacidade do sistema varia para atender às flutuações do tráfego
Alta disponibilidade	O sistema é redundante o suficiente para ser considerado como não interrompível
Atraso de programação em tempo real	Se existir programação do sistema e o atraso é menor que 1 segundo
Monitora o congestionamento	O congestionamento da rede é levado em conta como peso na divisão do tráfego

Na Tabela 3.2, são comparadas as características da solução proposta com as dos trabalhos relacionados. Embora não haja reserva estática de banda no presente trabalho e sim uma escalabilidade conforme a carga, podem ser adicionadas em trabalhos futuros as características de alta disponibilidade e monitorar o congestionamento.

Tabela 3.2: Comparação de características com os trabalhos relacionados

	Reserva Estática de Banda	Tempo Real	Escalável	Alta Disp.	Atraso de Programação em Tempo Real	Monitora o Congestionamento
SOLUÇÃO DO PRESENTE TRABALHO	X	I	I	+	I	+
Engenharia de Tráfego em Rede Não SDN	I	X	X	+	X	X
Engenharia de Tráfego com SDN	X	I	I	I	I	I
Balanceamento de Carga Híbrido com SDN	X	I	I	I	I	+
Balanceamento de Tráfego através de Programação Proativa com SDN	X	I	I	+	X	+
Balanceamento de Tráfego Web em Rede Não Estruturada com SDN	X	I	I	+	I	I
Balanceamento de Carga Web em SDN	X	I	I	+	I	+

LEGENDA = I: Implementada X: Inexistente +: Pode ser adicionada

Capítulo 4

Solução Proposta

Neste capítulo é discutida a solução proposta para a divisão de tráfego de entrada em um data center, com o objetivo de diminuir o tempo médio das requisições de serviço HTTP dos usuários. São implementadas três políticas de balanceamento de tráfego [Uppal & Brandon, 2010]:

1. ***Round Robin***: divisão por números iguais de fluxos sequencialmente para cada caminho disponível.

2. **Carga = Volume de Tráfego**: divisão de fluxos de acordo com um parâmetro de carga avaliado em cada caminho. A carga no presente trabalho é o volume de tráfego avaliado em cada caminho.

3. ***Random***: cada fluxo é assinalado para ir por um caminho de forma aleatória.

A política mais interessante é de carga, onde a divisão do tráfego em fluxos é equilibrada entre os caminhos de acordo com o volume de tráfego atual em cada caminho. Através da mesma pode ser possível obter uma diminuição do tempo médio de duração das requisições de serviços. O caminho com o menor volume de tráfego atual recebe o próximo fluxo que chegar. Políticas de balanceamento de carga podem ser criadas para considerar diversas variáveis, alguns exemplos: horários, eventos, clientes, robustez, segurança, destinos, serviços e máquinas; através delas todo o fluxo de uma rede pode ser dividido pela Engenharia de Tráfego.

No presente trabalho, o tráfego a ser dividido entre os servidores réplicas origina-se em estações clientes que fazem requisições a um serviço HTTP em um endereço IP virtual. O modo utilizado para encaminhar cada fluxo aos servidores réplica é similar ao de um sistema Web *cluster* ou sistema Web baseado em *cluster* [Cardellini et al., 2002]. Tal sistema encaminha os fluxos das requisições HTTP de cada cliente

reescrevendo o endereço IP virtual para o endereço IP interno de um servidor réplica através do switch. O respectivo fluxo da resposta do servidor também tem o seu endereço IP interno reescrito para o endereço IP virtual no switch ao ser encaminhado para o cliente.

A rede local de data center utiliza a arquitetura SDN, onde o controlador SDN executa uma política ativa de balanceamento de carga.

4.1 Arquitetura

A arquitetura da solução é exibida na Figura 4.1 abaixo. Embora a solução possa ser empregada para dividir o tráfego entre $\boldsymbol{N}$ servidores, um em cada enlace, é exibido N = 4 servidores, conforme o ambiente montado para os experimentos. O tráfego de entrada de requisições HTTP dos clientes chega por uma porta física no switch OpenFlow e é dividido entre os N servidores réplicas. Neste switch outra porta conecta o controlador POX, possibilitando a comunicação entre o controlador e o switch através do protocolo OpenFlow.

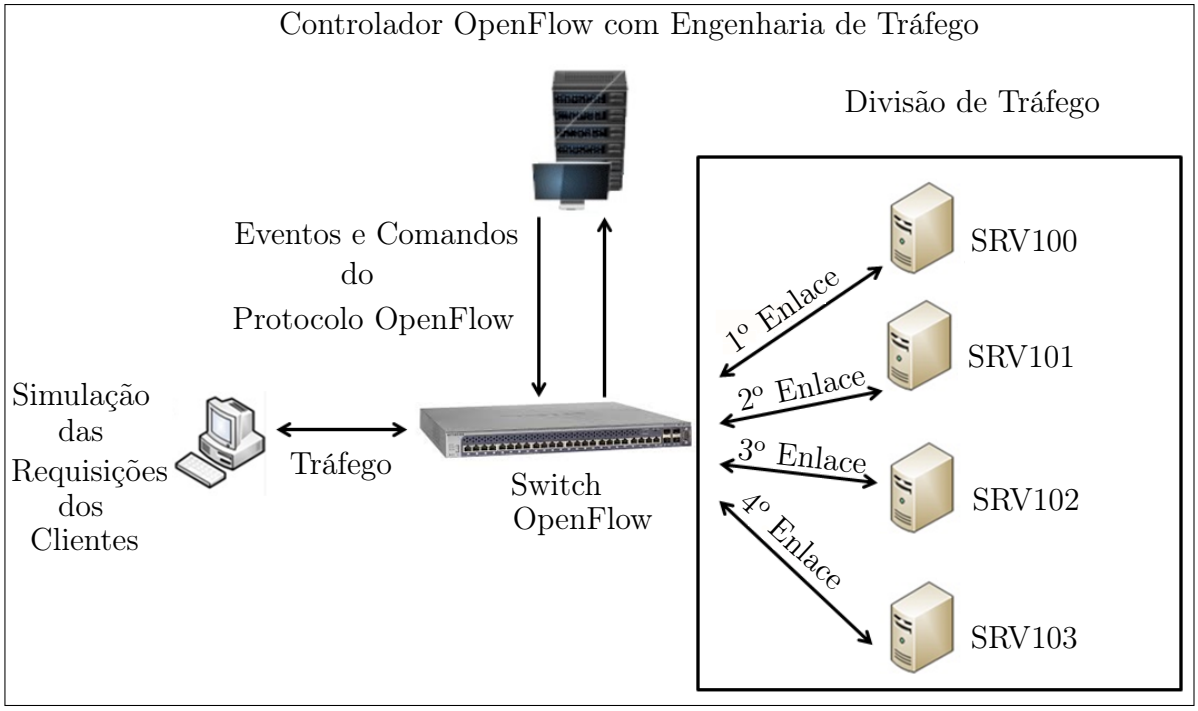

Figura 4.1: Arquitetura da solução para $\boldsymbol{N}$ = 4 servidores e 4 enlaces

4.2 Balanceamento de Carga

O Balanceamento de Carga pode ser modelado como um sistema de controle com retroalimentação, também chamado de sistema de controle fechado. A carga atual do switch OpenFlow é o valor de bytes já trafegados em cada caminho que é balanceado com a escolha de um próximo fluxo. Portanto, no presente trabalho, as variáveis de controle são os bytes já trafegados em cada caminho que termina em um servidor réplica distinto.

O tráfego atual obtido sofre um atraso para ser lido, que dependerá da atividade do switch. Assim, quanto mais alto o tráfego presente no switch, maior o atraso do ambiente, maior o erro de avaliação da carga comparado com valor real instantâneo. Para diminuir este problema poderia ser utilizando algum esquema de previsão de tráfego futuro, já que os fluxos são homogêneos e o ambiente também, mas a opção do presente trabalho é a de não implementar a previsão de carga com o objetivo de exibir diretamente o comportamento do switch utilizado.

4.2.1 Reescrita de Endereço IP

Todas as requisições dos clientes são destinadas ao IP virtual de um mesmo servidor HTTP virtual, conforme já visto na Seção 2.3. Deste modo, ao ser feita a divisão do fluxo de cada requisição pelo controlador, o mesmo define para qual servidor réplica o fluxo irá e insere uma regra que reescreve o endereço IP de tal servidor no lugar do endereço IP virtual, para todos os pacotes deste fluxo – tal ação é conhecida como reescrita de endereço IP e também por reescrita de cabeçalho IP. O fluxo da resposta dos servidores HTTP, para cada fluxo das requisições dos clientes, também sofre esta reescrita. Assim, todos os pacotes de cada fluxo de resposta têm o endereço IP virtual reescrito no lugar do endereço IP do servidor réplica.

Para melhor entendimento, abaixo estão exibidas as duas sequências de fluxo com reescrita de endereço IP:

- Fluxo de tráfego com as requisições dos clientes

  Clientes → IP Virtual → Switch reescreve IP → IP do servidor réplica

- Fluxo de tráfego com as respostas dos servidores réplicas

  Servidor réplica → Switch reescreve IP para IP Virtual → Clientes

4.2.2 Algoritmos de Controle Executados no Controlador OpenFlow

4.2.2.1 *Round Robin*

A política *Round Robin* simplesmente alterna sequencialmente todos os fluxos entre os servidores disponíveis. É rápida, pois apenas atribui o próximo fluxo para o próximo caminho de servidor. Realiza uma distribuição justa de fluxos entre todos os caminhos. A distribuição justa significa que um mesmo número de fluxos será distribuído por todos os servidores. Essa equidade de fluxos possibilita melhor desempenho quando todos os fluxos possuem características homogêneas e também a mesma duração. Entretanto, quando os fluxos são heterogêneos tal política não mantém a equidade na divisão dos mesmos. Esta política tem o seu algoritmo exibido através de pseudocódigo no Algoritmo 1

Algoritmo 1: Política *Round Robin*

```
1  // SERVIDORES é a lista com N servidores a serem utilizados
2  SERVIDORES ← [SRV1, SRV2, SRV3 ... SRVN];
3  NUM_SERV ← Length(SERVIDORES);
4  INDICE ← 0;
5  SERVIDOR ← 0;
6  while NOVA_REQUISIÇÃO do
7  |  INDICE++;
8  |  if INDICE > NUM_SERV then
9  |  |  INDICE ← INDICE − NUM_SERV;
10 |  end
11 |  SERVIDOR ← SERVIDORES[INDICE];
12 |  // Controlador OpenFlow cria nova regra no switch,
13 |  // com reescrita de endereço IP
   |   NOVA_REGRA_SWITCH(NOVA_REQUISIÇÃO,SERVIDOR);
14 end
```

4.2.2.2 Carga = Volume de Tráfego

A política de Carga atribui um próximo fluxo para o caminho que até então registrou o menor volume de tráfego. Assim é esperado que as pequenas disparidades de capacidade de desempenho em cada caminho sejam compensadas nessa política, com o objetivo de realizar uma distribuição justa do tráfego, considerando que a infraestrutura física é homogênea. Devido à carga ser lida através da primitiva do protocolo OpenFlow de estatística de portas - *PortStats*, uma contribuição do presente trabalho é o uso de uma

primitiva simples e padrão em todos os switches OpenFlow, portanto é esperado que fosse mais rápida e com menor custo computacional.

Uma desvantagem que pode ocorrer quando se emprega esta política é de haver um atraso considerável no cálculo da carga. Neste caso esta política obterá um erro na divisão, o que aumentará a complexidade em sua implementação.

Embora a carga no presente trabalho seja o volume de tráfego, outros parâmetros podem ser definidos como carga, tais como: o consumo de Entrada/Saída nos servidores, o uso de CPU nos servidores, o número de requisições pendentes na fila dos servidores e o número de transações em execução. É possível também implementar uma combinação de dois ou mais destes parâmetros como a carga.

Escolha da Primitiva de Leitura de Estatísticas do Switch

A primitiva do protocolo OpenFlow de Estatísticas de Portas *PortStats* foi escolhida em detrimento da primitiva de Estatísticas de Fluxos *FlowStats*, conforme explicado na Seção 4.3.6.

O algoritmo da política de Carga é exibido através de pseudocódigo no Algoritmo 2 abaixo.

Algoritmo 2: Política Carga = Volume de Tráfego

```
1  // SERVIDORES é a lista com N servidores a serem utilizados
2  SERVIDORES ← [SRV1, SRV2, SRV3 ... SRVN];
3  // As estatísticas são inicializadas em zero no switch via
4  // interface de administração
5  ZERA_SERVIDORES(SERVIDORES);
6  while NOVA_REQUISIÇÃO do
7      // Dispara mensagem para o switch solicitando estatísticas via PortStats
       SOLICITA_ESTATÍSTICAS(ESTATÍSTICAS);
8      // Atualiza estatísticas ao ocorrer evento com a resposta do PortStats,
9      // caso o evento atrase, os valores antigos das estatísticas são utilizados
       ATUALIZA(ESTATÍSTICAS,SERVIDORES);
10     // Calcula qual servidor está com o menor número de bytes trafegados,
11     // PortStats retorna o volume de bytes trafegados em cada
12     // porta física do switch conectada a um servidor
13     SERVIDOR ← MÍNIMO(SERVIDORES);
14     // Controlador OpenFlow cria nova regra no switch,
15     // com reescrita de endereço IP
       NOVA_REGRA_SWITCH(NOVA_REQUISIÇÃO,SERVIDOR);
16 end
```

As mensagens de *PortStats* enviadas pelo controlador ao switch são sincronizadas

com a chegada de novas requisições. O objetivo é que as taxas de leitura das estatísticas acompanhem as taxas de variações da chegada de novas requisições no switch. Com o aumento da taxa de novas requisições no switch é aumentada a taxa de mensagens do controlador, caso esta diminua, também é diminuída a taxa de mensagens *PortStats* do controlador.

4.2.2.3 *Random*

A política *Random* divide aleatoriamente todos os fluxos entre os servidores disponíveis. Pode ser considerada rápida, pois apenas realiza uma escolha aleatória do próximo fluxo para o próximo caminho de servidor. Não é uma distribuição justa por não distribuir com equidade fluxos homogêneos, porém é útil para ser comparada com os efeitos das outras políticas.

O algoritmo da política *Random* é exibido através de pseudocódigo no Algoritmo 3 abaixo.

Algoritmo 3: Política *Random*

```
1 // SERVIDORES é a lista com N servidores a serem utilizados
2 SERVIDORES ← [SRV1, SRV2, SRV3 ... SRVN];
3 while NOVA_REQUISIÇÃO do
4 |   // Calcula aleatoriamente qual o próximo servidor
5 |   SERVIDOR ← Random(SERVIDORES);
6 |   // Controlador OpenFlow cria nova regra no switch,
7 |   // com reescrita de cabeçalho IP
  |    NOVA_REGRA_SWITCH(NOVA_REQUISIÇÃO,SERVIDOR);
8 end
```

4.3 Dimensionamento das Restrições do Ambiente

Para uma solução ser implementada, é necessário dimensionar o ambiente físico através da avaliação de alguns parâmetros. Tais parâmetros podem ser considerados como as restrições daquele ambiente físico específico. Ao avaliar a vazão máxima pode-se considerar como a restrição do valor máximo da vazão naquele ambiente, por exemplo. Nesta seção são citadas algumas restrições que afetam a solução proposta. Seus valores terão que ser avaliados, pois são dependentes de cada ambiente físico. Restrições existem e são provenientes do conjunto de equipamentos utilizados: switch, controlador, placas de rede Ethernet, capacidades de servidores, capacidades de estações e nos cabos da rede. Serão avaliadas somente as restrições relevantes ao presente trabalho.

Alguns parâmetros foram levantados experimentalmente com o objetivo de se obter a magnitude dos mesmos no ambiente físico montado.

Segundo os experimentos do trabalho de Costa et al. [2016], o switch HP 2920-24G não apresenta um bom desempenho do plano de dados, assim, no presente trabalho são realizados experimentos numa de até 15 novas requisições por segundo, valores onde o switch apresenta um comportamento satisfatório. Todas as requisições são realizadas são de um mesmo arquivo de 1 Mbyte, no máximo 100 requisições simultâneas e com taxas menores do que 20 requisições/conexões novas por segundo. Valores maiores de taxas de novas conexões saturam o switch e retornam a leitura das estatísticas via *PortStats* com grandes atrasos.

4.3.1 Atraso da Programação de Novo Fluxo

Conforme citado na Seção 2.4.1, o atraso da programação de um novo fluxo na tabela do switch OpenFlow resultará em dois problemas que podem gerar perda de pacotes. O primeiro quando da ocorrência de rajadas de fluxos e o segundo quando o switch estiver próximo de sua saturação. Pode ser feita uma avaliação de qual é o atraso médio da programação de apenas um novo fluxo e considerá-lo como o valor mínimo de atraso. É esperado que no caso de altas taxas de programação de novos fluxos, o atraso seja maior. Considera-se que o atraso para a inserção de um novo fluxo seja da ordem de alguns milissegundos [Costa et al., 2016].

O atraso necessário para a programação de uma regra para um novo fluxo, no instante que ele ocorre, é denominado atraso de inserção de regra reativa. Quando são inseridas regras na inicialização do switch, já com a previsão dos possíveis fluxos que podem ocorrer, não há o atraso para a inserção da regra quando do instante da ocorrência deste novo fluxo. Tais regras são denominadas regras proativas, se diferenciando assim das regras reativas. As vantagens da eliminação do atraso de programação, ao se utilizar as regras proativas são exibidas no próximo Capítulo.

4.3.2 Latência Média da Rede

A latência da rede é o tempo que um pacote qualquer demora para ser transmitido da origem até o destino e voltar um pacote de resposta de confirmação (ACK) do destino para a origem. Considera-se latência da rede a soma dos tempos: de propagação, de transmissão e de processamento de cada pacote no switch. Nos experimentos realizados é desejado que o tempo de processamento de cada pacote no switch seja o menor possível, porém é esperado que seja maior que os tempos de propagação e transmissão.

A latência média da rede foi obtida empiricamente através do comando ping (ICMP - *Internet Control Message Protocol*) medido do cliente para cada servidor. Os resultados obtidos estão mostrados abaixo na Tabela 4.1, onde os quatro servidores utilizados são denominados SRV100, SRV101, SRV102 e SRV103. Foram enviados 1000 pacotes do cliente para cada servidor e a latência média do ambiente pode ser considerada da ordem de alguns milissegundos. O controlador OpenFlow estava operando no modo *Learning switch*, quando se comporta como um switch Ethernet comum. Conforme os resultados, a latência apresentada foi de no mínimo 1,6 milissegundos.

Tabela 4.1: Vazão máxima média e latência média

	iperf	ping	
Conexão entre Cliente e Servidor	Vazão Mb/s	Latência Média milissegundos	Desvio padrão
SRV100	19,3	1,505	0,074
SRV101	16,8	1,500	0,075
SRV102	16,4	1,506	0,072
SRV103	16,8	1,530	0,095

4.3.3 Vazão Máxima Média do Switch

Embora o valor da vazão máxima seja declarado na folha de dados pelo fabricante do switch, o valor real deve ser obtido, pois afeta a avaliação dos resultados. É avaliada a vazão máxima média da rede com um só fluxo. Os valores foram obtidos experimentalmente através do utilitário iperf[1] executado durante 1 hora entre o cliente e cada servidor. Os valores obtidos são exibidos na Tabela 4.1. A vazão máxima média da rede pode ser considerada como da ordem de 20 Mb/s. O controlador OpenFlow também estava no modo *Learning switch*. O switch atinge 1 Gb/s em seu modo não OpenFlow ou modo "Legacy". A baixa vazão no modo OpenFlow frente o seu modo "Legacy" é uma indicação também de seu baixo desempenho, confirmada no trabalho de Costa et al. [2016].

4.3.4 Simulação de Tráfego

Devido ao funcionamento não satisfatório do switch físico em modo OpenFlow, não foi possível a execução de requisições de clientes (*traces*) anotadas em servidores reais. Apenas foram utilizadas transferência de arquivos de 1 Mbyte, perfil de carga típica de WEB 2.0, onde os usuários recuperam arquivos de texto ou fotos maiores que simples

[1] `https://iperf.fr/`

páginas [Summers et al., 2012]. Todas as medições das transferências são exibidas na Seção 5.5.

4.3.5 Pacotes/Conexões Perdidas

A ocorrência de pacotes e conexões perdidas somente pode ser avaliada experimentalmente, depende do ambiente e afeta os resultados de cada experimento. Os experimentos foram dimensionados para gerarem um máximo de 100 conexões simultâneas e em uma faixa onde não há perda de pacotes.

4.3.6 Avaliação da Carga

A obtenção da carga atual em cada caminho é feita através da leitura do total de bytes trafegados na porta física do switch para cada caminho. É uma operação de leitura de estatísticas do switch e depende do tempo de resposta do switch para retornar esse total. Se o tempo de resposta for demasiadamente alto pode-se ter um grande erro de leitura entre o valor obtido e o valor real. No caso de uma vazão de, por exemplo, 16 Mb/s, se o tempo de resposta for 1 μs o erro relativo é de 16 bits. Uma medição de atrasos de *PortStats* exibiu atrasos de até 0,8 segundos, o que resulta em um erro de 12.800.000 bits. O erro no atraso de 1 μs é calculado multiplicando-se 16×10^6 bit/s por 1×10^{-6} s e o do atraso de 0,8 segundos é calculado multiplicando-se 16×10^6 bit/s por 0,8 s. Devido a este erro, no presente trabalho foi adotado um intervalo de valores para experimentos que não fossem comprometidos pelos atrasos do switch utilizado.

Para a avaliação da carga no experimentos foi escolhida a primitiva OpenFlow *PortStats* em detrimento da primitiva *FlowStats* devido ao atraso intrínseco de cada uma. As medições realizadas no switch HP 2920-24G indicam que para um total de 1000 fluxos este switch OpenFlow demora até de 0,8 segundos para retornar a leitura via *FlowStats* enquanto via *PortStats* demora apenas 0,04 segundos. Uma razão para a *PortStats* ser mais rápida é porque a estrutura de dados que retorna por porta é menor e também por existirem apenas 24 portas físicas no switch HP 2920-24G. A estrutura de dados retornada pela *FlowStats* além de ser maior para cada fluxo individualmente, também é multiplicada pelo número de fluxos ativos que pode ser da ordem dos milhares. Uma característica que também diferencia *PortStats* de *FlowStats* é que quando o fluxo é removido do switch as suas estatísticas também são, enquanto que *PortStats* acumula todos os bytes e pacotes trafegados em cada porta desde que o switch OpenFlow foi iniciado.

4.4 Conclusão

Este capítulo apresentou as três políticas do método de Balanceamento de Carga de Engenharia de Tráfego: Carga, *Round Robin* e *Random*. A carga considerada é a quantidade de bytes já trafegados em cada porta do switch OpenFlow. Tal carga é lida através da primitiva *PortStats*. Explica-se o uso da política *Random* para comparação com as demais políticas. A seguir são comentados alguns parâmetros levantados empiricamente ou não, para dimensionar as restrições do ambiente físico onde se implementa a solução proposta.

No capítulo seguinte são exibidos o ambiente experimental, a sua solução implementada com seus algoritmos, quatro variações de código, os resultados e análise. Ao fim é realizada uma comparação dos ganhos obtidos com um código similar que não emprega a leitura de estatísticas diretamente no switch OpenFlow via *PortStats*.

Capítulo 5

Ambiente Experimental e Resultados

5.1 Modelagem do Problema

O problema abordado neste trabalho ocorre quando há um grande volume de tráfego de entrada da Internet em um data center, mostrado abaixo na Figura 5.1, causando uma saturação física ou congestionamento de pacotes em um ou mais caminhos da entrada. Existe o risco de ser atingida a capacidade máxima de alguns caminhos, enquanto outros caminhos poderão estar com capacidade ociosa. Tal problema poderá ocorrer em curtos ou longos períodos de tempo, independente da alta ou baixa capacidade contratada para os enlaces físicos dessa entrada.

A saturação física ou congestionamento ocorrerá quando a vazão dos enlaces atinge seu valor máximo e se torna insuficiente para atender as condições mínimas de utilização dos serviços, ou seja, o tráfego sofrerá atrasos, perdas e poderão ocorrer travamentos.

Uma vez ocorrendo o problema em questão, vários outros ocorrem como consequência, segundo a nossa experiência. Abaixo estão os principais:

1. Clientes, serviços e aplicações com a Internet travam (deadlocks) ou se desconectam.
2. A velocidade ou o atraso das transações inviabilizam o uso.
3. Novas conexões com a Internet não conseguem ser iniciadas.
4. Serviços da rede local sofrem lentidão devido às retransmissões.

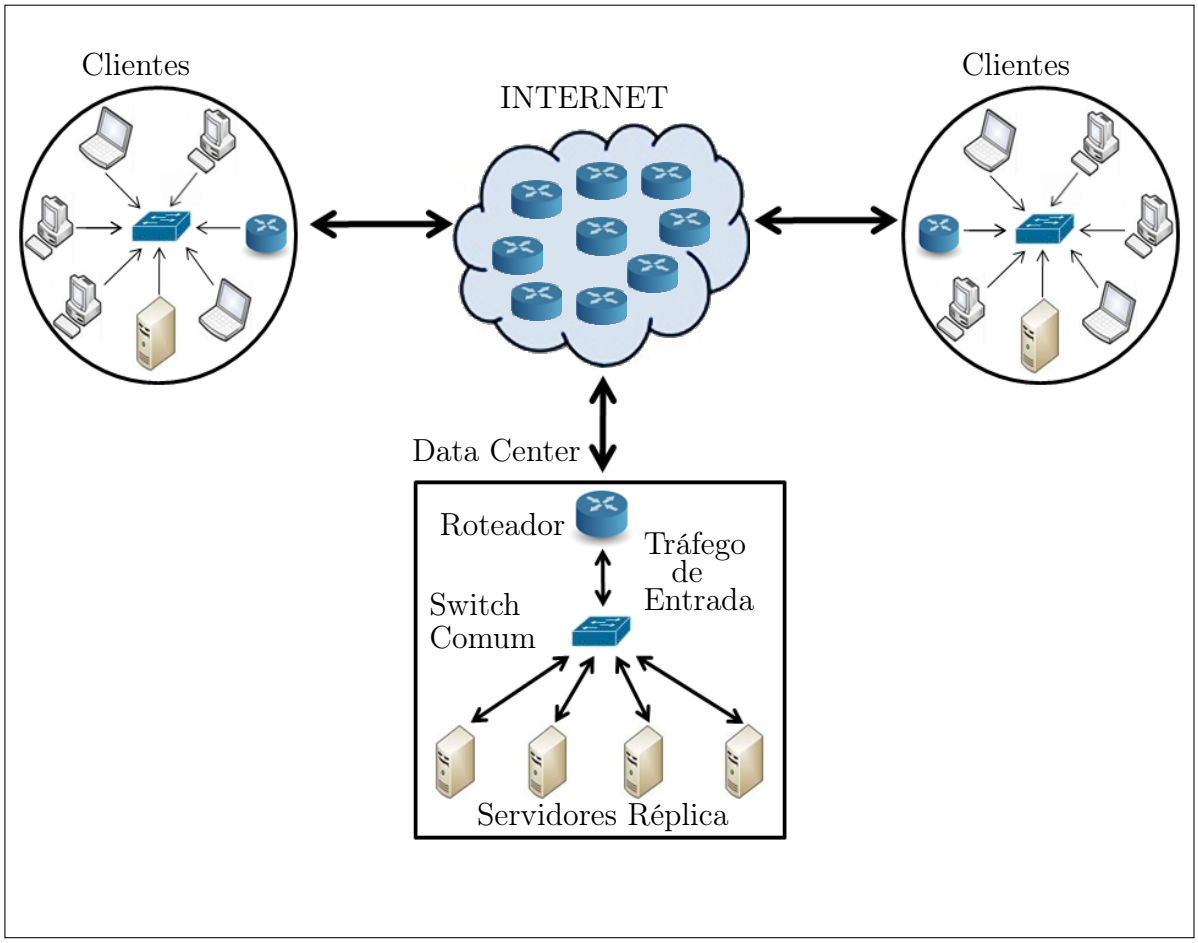

Figura 5.1: Sistema com switch comum, não OpenFlow, com grande volume de tráfego

5.2 Solução Implementada

A solução implementada neste trabalho possui dois componentes. O primeiro é o método de Engenharia de Tráfego como serviço, para a divisão do tráfego de entrada seguindo uma política para melhorar a vazão. O segundo componente é a implementação em um ambiente físico SDN, onde o controlador OpenFlow executa o serviço em um switch OpenFlow com a reescrita de pacotes descrita no Capítulo 4.

O controlador OpenFlow determina os caminhos e faz a alocação de fluxos aos mesmos, utilizando Engenharia de Tráfego. O parâmetro utilizado para avaliar a vazão é o tempo médio das requisições de serviços, medido no cliente.

5.2.1 Aplicações da Solução

A solução proposta poderá ser aplicada, em alguns ambientes para:

1. Divisão do tráfego de entrada em data centers conforme Figura 5.2. Como vemos

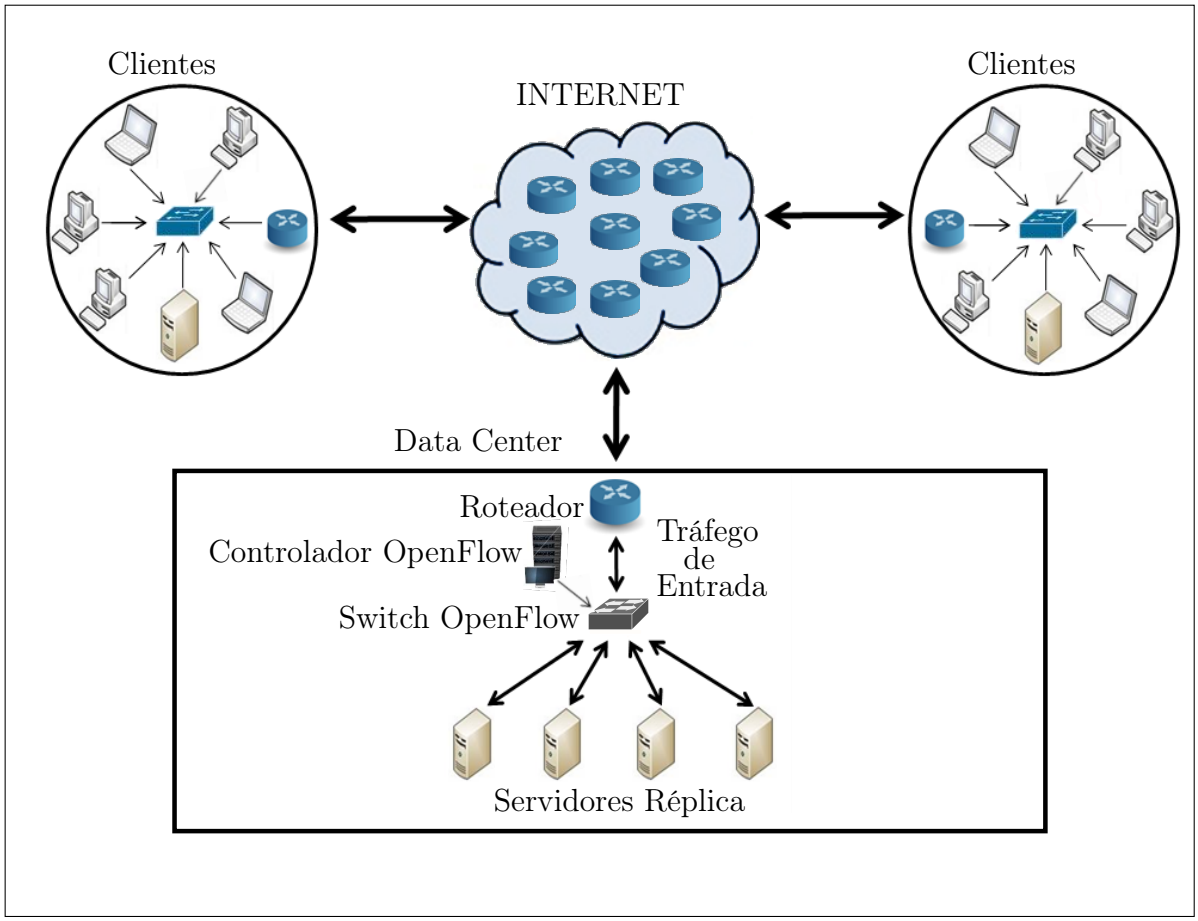

Figura 5.2: Solução com switch e controlador OpenFlow

na figura, todo o tráfego de entrada na borda de um data center é dividido por Engenharia de Tráfego para seus servidores em um ambiente OpenFlow na rede interna.

2. Divisão do tráfego de entrada em ambientes não caracterizados como data center, mas com o mesmo problema.

3. Divisão do tráfego interno do data center - entre seus servidores

5.3 Experimentos

Para avaliar a solução proposta foram feitos vários experimentos que são exibidos nessa seção. Essa seção foi dividida em duas partes:

1. Ambiente de Hardware: parte física com os equipamentos utilizados.

2. Ambiente de Software: algoritmos, programas e sistemas utilizados.

5.3.1 Ambiente de Hardware

Foi montada em um laboratório do DCC/ICEx/UFMG, a plataforma física para a implementação da solução de Engenharia de Tráfego em SDN, simulando um ambiente real. A seguir foi avaliado o desempenho de diferentes políticas sob diferentes cargas geradas.

A plataforma física implementada, exibida na Figura 4.1, foi composta pelos seguintes equipamentos:

A. Clientes - Uma estação física simulando os clientes gerando requisições HTTP:
   CPU Intel Core i5-2310 2,9 GHz
   8MB de memória DDR3 1,333 GHz
   Disco de 1GB SATA 2
   Ethernet 10/100/1000 Mbps - Realtek r8169
   CentOs 6 - Linux - 64 bits.

B. Switch Openflow - Switch HP 2920-24G:
   24 portas Ethernet 10/100/1000 Mbps
   Protocolo OpenFlow versão 1.0
   Modo OpenFlow Instância Agregada (*Instance Aggregate Mode*).
   Foi o primeiro e único switch comercial OpenFlow disponível para ser utilizado durante a maior parte do desenvolvimento do presente trabalho.

C. Servidores réplica - Quatro servidores de HTTP "Apache Web Server 2.4.6" [1]:
   CPU Intel Core i7-3770 3,4 GHz
   16GB de memória DDR3 1,333 GHz
   Disco de 500MB SATA 2
   Ethernet 10/100/1000 Mbps - Realtek r8169
   CentOs 7 - Linux 3.14.4-200 - 64 bits.

D. Controlador OpenFlow 1.0 – Servidor Linux ambiente POX:
   Sony Vaio VGN-C240E
   CPU Intel Core2 Duo T5500 1,66 GHz
   2GB de memória DDR2 533 MHz
   Disco de 160GB SATA 2
   Ethernet 10/100 Mbps - Marvell Yukon
   Ubuntu 12.04.4 LTS – Linux 3.11.0-15 - 64 bits
   POX 0.2.0 (Carp) - Python 2.7.3.

[1] `http://people.apache.org/~jim/presos/ACNA11/Apache_httpd_cloud.pdf`

5.3.2 Ambiente de Software

O ambiente de software executado no controlador, clientes e servidores é descrito a seguir:

- Controlador - O método de balanceamento de carga com a política, em Python 2.7, compõe o controlador OpenFlow do ambiente POX versão 0.2.0 (Carp) executando no Sistema Operacional Linux.

- Clientes - Os clientes são representados por um único servidor Linux, que executa programas para simular a carga dos clientes. Programas em Python e em Shell scripts geram a carga e coletam os dados para análises dos serviços. É utilizada a ferramenta httperf[2] para gerar a carga simulada de requisições HTTP dos clientes, e registrar os resultados.

- Servidores - São quatro servidores Linux com o servidor de HTTP Apache, todos são réplicas compondo um servidor virtual.

O controlador OpenFlow contém o algoritmo de controle com a política a ser utilizada na divisão de fluxos. Na inicialização, o algoritmo de controle limpa a tabela de fluxos OpenFlow no switch. A partir de então, quando o switch recebe o primeiro pacote de um fluxo novo, um evento é acionado no programa do controlador que verifica se o fluxo é uma requisição HTTP (TCP) de cliente para o endereço IP do servidor virtual. Somente em caso positivo é tomada a decisão de encaminhar esse novo fluxo para um dos quatro servidores réplicas.

São empregadas as três políticas de Engenharia de Tráfego diferentes, cada uma com seu próprio algoritmo de controle, diferenciado com as respectivas decisões de divisão de fluxos.

5.3.2.1 Obtenção dos Dados do Experimento

No cliente são registradas quantas requisições foram solicitadas, o tempo médio de cada requisição (tempo de conexão), o tempo total, erros, conexões perdidas e a vazão média. O objetivo é usar o tempo médio de cada requisição para avaliar cada política de Engenharia de Tráfego. Os valores de tempo de conexão são anotados e caso aconteceram erros que não sejam CONNECTION RESET [Stevens, 1993], o experimento é refeito. Somente no caso do experimentos com o Código 1 da Seção 5.5.1 aconteceram erros CONNECTION RESET.

[2] `https://github.com/httperf/httperf`

O tráfego de pacotes no cliente foi analisado com o ferramenta Wireshark[3] que mostrou a causa do CONNECTION RESET enviado pelo cliente a um endereço de serviço HTTP. Segundo o tráfego analisado, em algumas situações um pacote ACK vindo de um dos servidores não sofre a reescrita para o endereço IP virtual e continua com o IP real deste servidor. Ao sair do switch e chegar no cliente, é um ACK de uma conexão *socket* [Kurose, 2005] que não foi estabelecida, logo tal ACK é considerado pelo cliente como um erro no protocolo TCP, que envia um pacote com a mensagem CONNECTION RESET destinada ao IP do servidor real. Logo a seguir, o pacote é reenviado pelo mesmo servidor HTTP, no mesmo *socket* – porta origem, ip origem e porta destino, ip destino; mas com a reescrita correta do endereço IP virtual. Agora, como é um ACK de uma conexão existente, o cliente prossegue normalmente o recebimento do fluxo desta conexão. Tal problema provavelmente deve ser causado por um ***BUG*** no switch empregado.

5.4 Metodologia

Para gerar as requisições, simulando tráfego de vários clientes no servidor Linux, foi utilizada a ferramenta HTTPERF, com as opções abaixo:
httperf - -server 10.10.42.111 - -port 8080 - -uri=/obj1mb.bin - -num-conn 100 - -rate N
Onde:

- server 10.10.42.111 é o endereço IP do servidor HTTP virtual - destino do tráfego.
- port 8080 é a porta de serviço TCP utilizada pelos servidores réplica HTTP.
- uri=/obj1mb.bin é o arquivo de 1 MByte a ser transferido em cada requisição.
- num-conn 100 são as 100 conexões a serem abertas (1 conexão = 1 requisição).
- rate N é o número N novas requisições que serão realizadas a cada segundo.

O controlador OpenFlow gerenciará a política ativa em cada experimento e atuará no switch OpenFlow a cada novo fluxo direcionando-o para um dos caminhos (enlaces) disponíveis. Cada um dos servidores estará em cada enlace disponível, recebendo os fluxos de ida das requisições e respondendo com fluxos de volta. Na Figura 4.1 são exibidos os elementos da plataforma física.

[3] `https://www.wireshark.org/`

5.4.1 Baixas Taxas de Novas Conexões

Uma avaliação do switch HP 2920-24G, utilizado nos experimentos [Costa et al., 2016], conclui que o referido switch em modo OpenFlow tem baixo desempenho. Na verdade é uma caixa-preta da qual o fabricante não informa parâmetros de desempenho e utilização. Alguns parâmetros levantados empiricamente são exibidos na Seção 4.3. Não é esperado que a vazão do switch em modo OpenFlow fosse muito abaixo de 1 Gb/s, contudo foram observados valores da ordem de 17 Mb/s. Assim todos os experimentos executados foram modelados com cargas leves que não saturam o switch, ou seja, foram experimentados tráfegos em vários perfis com o objetivo de avaliar o funcionamento dentro de uma faixa estável em modo OpenFlow. O switch por ser uma implementação em software, além de apresentar baixo desempenho possui grande variações de atrasos. Em alguns casos as requisições HTTP foram perdidas sem se encontrar uma justificativa, sob alta demanda surgiam mensagens de pacote com erro no cabeçalho.

Embora seja esperado que outros switches possuam um desempenho muito superior, na época da realização desta dissertação, somente era possível o uso deste "hardware" switch. Portanto, este foi o único switch utilizado.

De acordo com o parágrafo anterior, os perfis de carga foram experimentados em torno de variações do padrão de uma carga típica de vídeo conforme sugerido por Summers et al. [2012], que é a transmissão de um arquivo entre os tamanhos de 0,5 MB e 2 MB. Embora a carga para testes seja de 1.000 requisições de acordo com o citado padrão, através dos perfis experimentados, a carga escolhida foi de 100 requisições de um arquivo de 1 MB. Como switch HP 2920-24G em modo OpenFlow não respondeu satisfatoriamente sob altas taxas de novas conexões, um levantamento empírico chegou a estes valores. Foi observado que até a uma taxa de 10 novas conexões por segundo os experimentos estariam dentro de uma faixa estável de operação, para serem avaliados. As taxas utilizadas nos experimentos foram: 1, 2, 5, 7, 10 e 15 conexões por segundo. Assim, todas as conexões são requisições HTTP de um arquivo de 1 MB.

5.4.2 Três Códigos do Controlador com Diferentes Inserções de Fluxo

Foram implementados três códigos diferentes para o mesmo controlador, com os mesmos algoritmos das políticas, com o objetivo de avaliar se há um melhor desempenho em algum deles. As três variações do código do controlador apenas alteram o modo de inserção das regras de divisão de tráfego no switch. O objetivo destes três códigos é de entender como as restrições do ambiente real podem afetar a solução proposta,

conforme as variações implementadas nestes códigos.

A inserção de uma regra no switch OpenFlow ocorre quando surge o primeiro pacote de um novo fluxo de dados em uma porta do switch e é denominada regra reativa. Após calcular por qual caminho ou enlace o pacote trafegará, o controlador insere no switch a regra de ida deste primeiro pacote, que o encaminha para a porta calculada. Essa regra apenas permite a ida do pacote, do cliente para o servidor. É possível inserir ao mesmo tempo uma regra para o fluxo na direção inversa que corresponde a resposta do servidor a esta requisição. Como as requisições de serviço HTTP abrem conexões via *sockets* TCP entre o cliente e o servidor HTTP, então há um fluxo de ida no sentido cliente para servidor e o fluxo de volta das respostas destas requisições, no sentido servidor para cliente. São necessárias então duas regras novas inseridas no switch OpenFlow quando há um novo fluxo, a saber: a regra de ida do fluxo (a requisição do cliente para o servidor) e a regra de volta do mesmo (a resposta para atender esta requisição, do servidor para o cliente).

As três variações de código que foram experimentadas, referentes à inserção das regras de encaminhamento no switch OpenFlow, são:

1. Código 1 - Denominado Regras Reativas, implementa a inserção reativa da regra de ida para a requisição do cliente e depois a inserção reativa da regra de volta da resposta do servidor. Todas as inserções de regras são reativas.

2. Código 2 - Denominado Regras Reativas com Volta, implementa a inserção reativa da regra de volta antes da inserção da regra de ida. A regra de ida para a requisição do cliente é inserida imediatamente depois da inserção da regra de volta necessária para a resposta do servidor.

3. Código 3 - Denominado Regras Proativas, implementa a inserção proativa de todas as regras de volta na inicialização do switch e reativa da regra de ida. Estas regras proativas inserem todos os possíveis fluxos de volta das respostas dos servidores réplica para o cliente. Para tal são utilizadas regras com coringa(*wildcard*) abrangendo todas as possíveis portas de *sockets* a serem utilizadas entre os servidores réplica e o cliente.

As regras reativas mantém os valores das portas dos *sockets* utilizados desde o primeiro pacote de cada fluxo e realizam a reescrita de endereço IPs. Se o pacote originar nos clientes a reescrita é do endereço IP virtual destino com o endereço IP de um servidor réplica. Caso o pacote originar de um servidor réplica a reescrita é do endereço do servidor réplica origem com o endereço IP virtual do servidor HTTP, conforme visto na Seção 2.3.

5.4.3 Quarto Código para Comparação - Leitura via *sockets*

Após as três variações de código que utilizam a leitura de estatísticas no switch Openflow, um quarto código foi implementado, denominado Código 4. Tal código serve para comparar a eficiência da leitura das estatísticas no switch OpenFlow via *Portstats* frente outro tipo de leitura do controlador via *sockets*, que obtém diretamente os valores de bytes trafegados em cada interface de rede dos servidores. Para obter essa leitura, uma segunda rede local foi montada em um switch Ethernet comum não OpenFlow, o controlador e os servidores foram conectados através de interfaces de redes adicionais. Isto ocorre devido ao fato do controlador OpenFlow se conectar ao switch através de um canal de comunicação exclusivo – *out of band*, isolado dos tráfegos das outras portas, então para este controlador obter os dados em cada servidor réplica é necessária esta segunda rede.

A análise dos resultados e a comparação são discutidas na próxima Seção 5.5 a partir das estatísticas e gráficos gerados. É exibido o tempo médio de duração das conexões e as taxas de requisições geradas pelo cliente para cada uma das diferentes políticas de Engenharia de Tráfego. Os resultados obtidos das políticas são comparados entre si. Os resultados do código para comparação, Código 4, são exibidos e comparados com a política de melhor resultado dos três códigos anteriores.

5.5 Resultados e Análise

Espera-se atingir o melhor resultado entre todas as políticas com a política de Balanceamento por Carga. Nessa política a divisão de fluxos é equilibrada de acordo com a carga atual. O caminho com a menor tráfego atual recebe o próximo fluxo e com isso espera-se o menor tempo médio de conexão em todos os experimentos. Embora todos os servidores do experimento sejam semelhantes espera-se uma desigualdade de desempenho.

Os resultados dos experimentos dependem da rapidez da obtenção da carga em um dado instante. Atrasos grandes na obtenção do valor da carga levarão a uma escolha errônea. As medições a taxas de 1 e de 2 novas requisições por segundo não mostram que os tempos de cada requisição são afetados, por durarem pouco tempo no switch, possibilitando apenas uma pequena diferença entre as três políticas. O código para comparação será utilizado para validar a contribuição da proposta do presente trabalho.

5.5.1 Código 1 - Regras Reativas

O primeiro código realiza a inserção reativa da regra de ida para cada novo fluxo no switch, criado pelo cliente por suas requisições HTTP e depois a inserção reativa da regra de volta de cada novo fluxo criado por um dos servidores para responder a requisição. Na Figura 5.3 são exibidos os tempos das requisições médias obtidos pelas três políticas, para 100 requisições uma em cada conexão TCP, para as taxas de 1, 2, 4, 5, 17, 10 e 15 requisições por segundo. Na Tabela 5.1 são exibidos os valores com os intervalos de confiança de 95% e o desvio padrão relativo a 33 repetições de cada experimento.

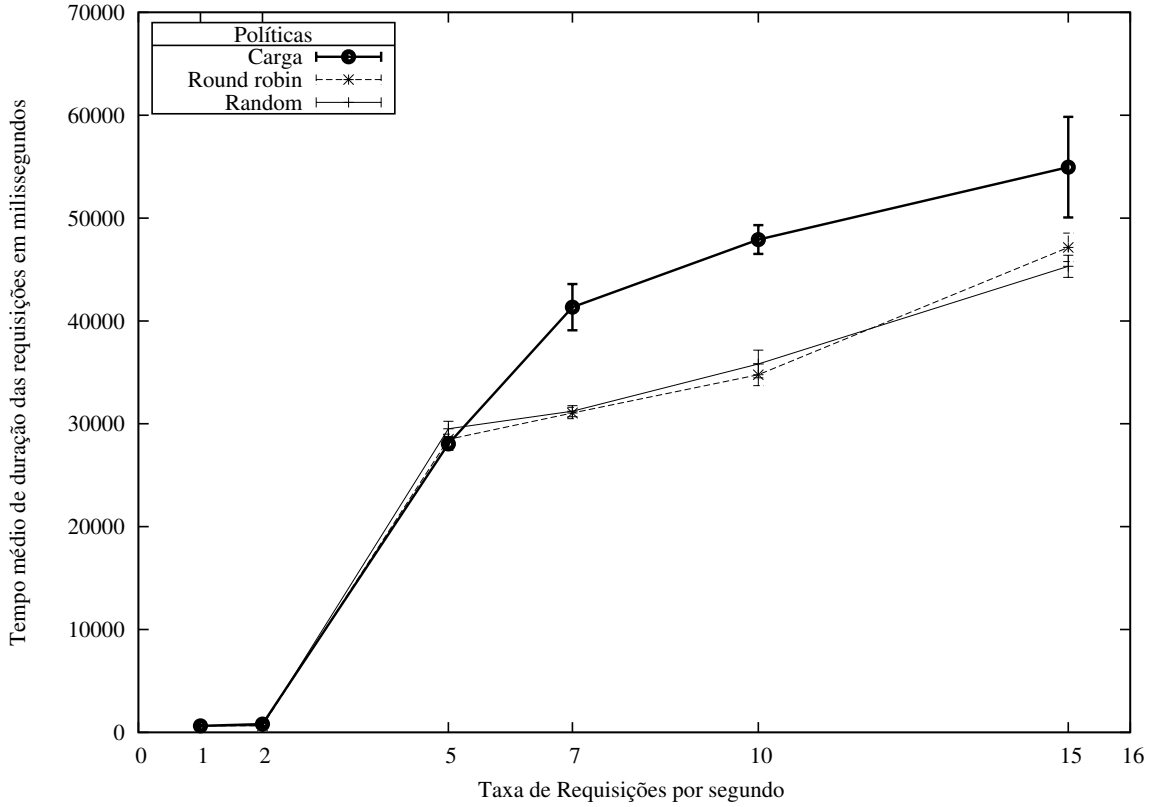

Figura 5.3: Código 1 - Regras Reativas

Tabela 5.1: Código 1 - Tempos de requisições com as Regras Reativas (milissegundos)

Requisições por Segundo	Política de Carga			Política Round Robin			Política Random		
	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão
1	633,82	626,19 - 641,46	26,66	637,75	635,95 - 639,55	6,29	631,94	629,77 - 634,11	7,58
2	808,45	544,06 - 1072,84	923,15	652,18	650,16 - 645,19	7,02	651,23	647,60 - 654,86	12,67
5	28041,95	27484,05- 28599,85	1948,01	28940,02	27993,17 - 28986,87	1734,84	29513,65	28778,65 - 30248,65	2566,40
7	41333,21	39086,70 - 43579,71	7844,10	31048,27	30495,87 - 31600,67	1928,80	31223,65	30684,68 - 31762,62	1881,92
10	47914,23	46515,17 - 49313,29	4885,08	34775,22	33714,56 - 35835,87	3703,48	35813,02	3446,82 - 37159,21	4700,49
15	54948,46	50060,04 - 59836,89	17068,84	47154,29	45763,65 - 48544,93	4855,68	45316,20	44239,82 - 46392,57	3758,37

Os resultados mostram que a política de Carga é a melhor na taxa de 5 novas requisições por segundo, a política *Round Robin* é melhor nas taxas de 7 e 10 novas requisições por segundo. A política *Random* somente obteve os melhores valores na taxa de 15 novas requisições por segundo. Observa-se um desvio padrão irregular em todas as medições, com tendência crescente à saturação com o aumento das taxas. Foi observada uma taxa de perdas de pacotes da ordem de 10%, devido a erros de não reescrita de endereço IP durante as execuções deste código, o que gera mensagens de CONNECTION RESET [Stevens, 1993] enviadas pelo cliente, conforme explicado na Seção 5.3.2.1. Conforme o esperado no caso de pior desempenho do switch e perdas de pacotes, a política *Round Robin* obteve os menores tempos de conexão.

5.5.2 Código 2 - Regras Reativas com Volta

O segundo código realiza a inserção reativa da regra de volta de um servidor para o cliente antes da inserção da regra de ida de cada novo fluxo criado pelo cliente, por suas requisições HTTP. Na Figura 5.4 são exibidos os tempos das requisições médias obtidos pelas três políticas, para 100 requisições, uma em cada conexão TCP, para as taxas de 1, 2, 4, 5, 17, 10 e 15 requisições por segundo. Na Tabela 5.2 são exibidos os valores com os intervalos de confiança de 95% e o desvio padrão relativo a 33 repetições de cada experimento.

Os resultados mostram que a política de divisão *Round Robin* é a melhor nas taxas de 7, 10 e 15 novas requisições por segundo. Na taxa de 5 novas requisições por segundo a política de Carga é a melhor. Observa-se também um desvio padrão irregular em todas as medições, com tendência crescente à saturação com o aumento das taxas. Não houve perdas de pacotes nas amostras.

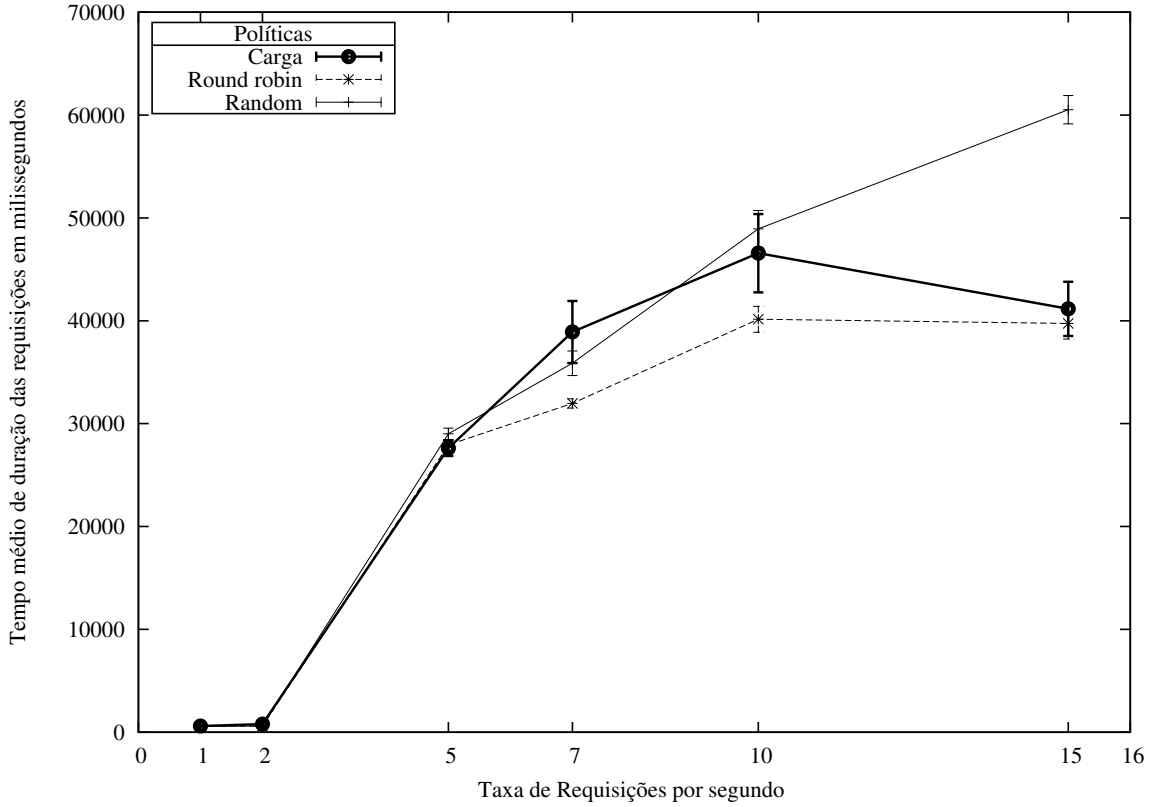

Figura 5.4: Código 2 - Regras Reativas com Volta

Tabela 5.2: Código 2 - Tempos de requisições com as regras Reativas com Volta (milissegundos)

Requisições por Segundo	Política de Carga			Política Round Robin			Política Random		
	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão
1	601,53	597,45 - 605,62	14,27	586,61	584,57 - 588,66	7,15	582,56	580,51 - 584,62	7,18
2	794,12	495,03 - 1093,21	1044,33	603,19	593,40 - 612,98	34,19	604,22	595,67 - 612,78	29,87
5	27610,30	26860,72 - 28359,89	2617,31	27926,35	27407,83 - 2844,86	1810,50	29004,20	28448,17 - 29560,23	1941,49
7	38910,71	35896,69 - 41924,73	10524,01	31962,20	31509,83 - 32414,57	1579,54	35870,50	34677,97 - 37063,03	4163,95
10	46573,04	42766,49 - 50379,59	13291,28	40144,04	38880,21 - 41407,88	4412,91	48930,28	47143,37 - 50717,18	6239,31
15	41156,67	38525,49 - 43787,85	9187,25	39746,88	38224,68 - 41269,08	5315,05	60517,72	59138,63 - 61896,82	4815,37

5.5.3 Código 3 - Regras Proativas

O terceiro código realiza a inserção proativa de todas as possíveis regras de volta (regra com coringa - *wildcard*) de todos os servidores para o cliente antes da inserção reativa da regra de ida de cada novo fluxo criado pelo cliente por suas requisições HTTP. Na Figura 5.5 são exibidos os tempos das requisições médias obtidos pelas três políticas, para 100 requisições uma em cada conexão TCP, para as taxas de 1, 2, 4, 5, 17, 10 e 15 requisições por segundo. Na Tabela 5.3 são exibidos os valores com os intervalos de confiança de 95% e o desvio padrão relativo a 33 repetições de cada experimento.

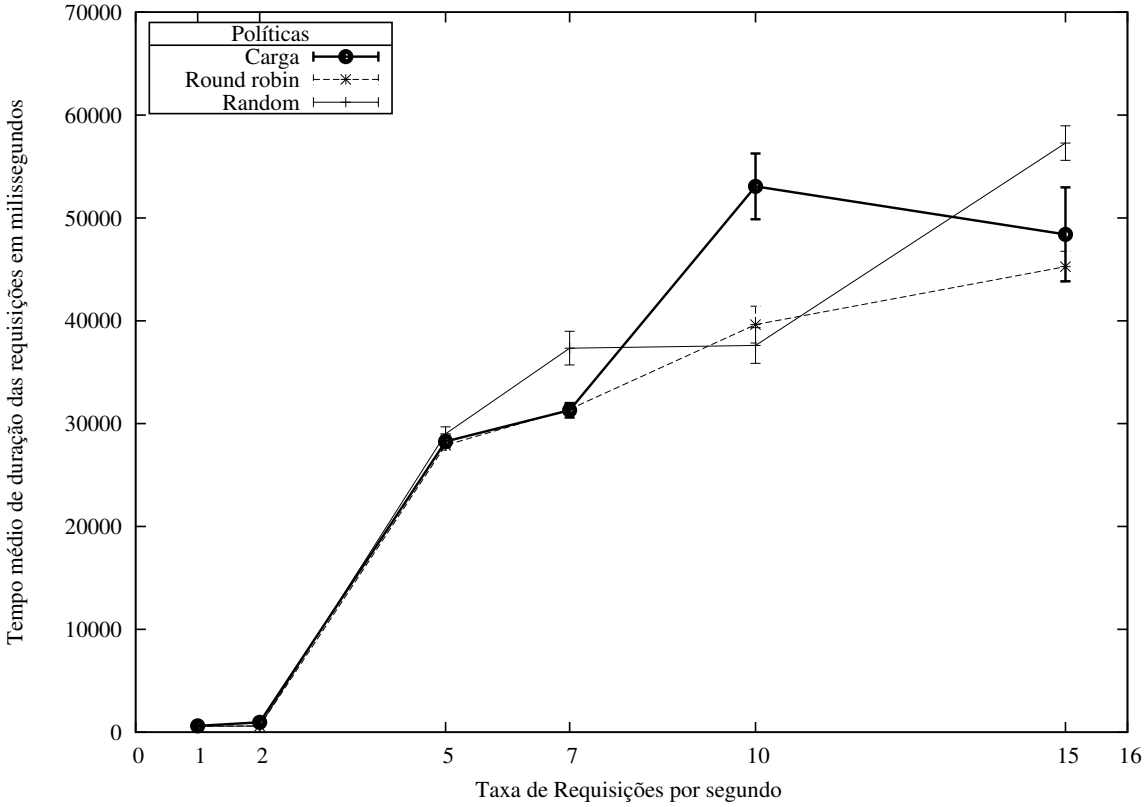

Figura 5.5: Código 3 - Regras Proativas

Os resultados mostram que a política de divisão por Carga é a melhor na taxa de 7 novas requisições por segundo e a política *Random* nas taxas de 5 e 15 novas requisições por segundo. A política *Random* somente na taxa de 10 novas requisições por segundo. Também se observa um desvio padrão irregular em todas as medições, com tendência crescente à saturação com o aumento das taxas. Não houveram perdas de pacotes nas amostras.

Tabela 5.3: Código 3 - Tempos de requisições com as Regras Proativas (milissegundos)

Requisições por Segundo	Política de Carga			Política Round Robin			Política Random		
	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão	Tempo Médio	Intervalo de Confiança 95%	Desvio Padrão
1	610,43	601,86 - 619,00	29,93	596,71	593,96 - 599.47	9,62	592,08	589,80 - 594,37	8,00
2	963,23	555,78 - 1370,69	1422,70	609,10	608,05 - 610.15	3,66	601,44	598,39 - 604,48	10,63
5	28261,90	27700,62 - 28823,18	1959,83	27908,67	27428,74 - 28388,60	1675,77	29016,51	28346,83 - 29686,19	2338,31
7	31298,52	30594,04 - 32003,00	2459,81	31403,05	30948,65 - 31857,46	1586,63	37336,24	35698,22 - 38974,26	5719,44
10	53069,18	49876,30 - 56262,07	11148,57	39630,29	37847,41 - 41413,17	6225,27	37596,57	35867,20 - 39325,95	6038,44
15	48404,85	43832,51 - 52977,20	15965,19	45267,75	43783,70 - 46751,80	5181,85	57276,79	55598,08 - 58955,50	5861,53

Dos três códigos de controlador OpenFlow implementados que fazem a leitura das estatísticas do switch via *Portstats*, o Código 3 é o que obteve os resultados da política de carga mais próximos dos resultados da política *Round Robin*. Foi conseguido devido ao efeito da inserção de todas as regras de fluxo de volta dos servidores na inicialização

do switch, pois elimina o atraso para a inserção da regra na tabela do switch Openflow, para cada novo fluxo de volta.

5.5.4 Código 4 - Carga Medida na Interface de Rede

O quarto código realiza a inserção proativa de todas as possíveis regras de volta (regra coringa - *wildcard*) de todos os servidores para o cliente antes da inserção reativa da regra de ida de cada novo fluxo criado pelo cliente por suas requisições HTTP. Na Figura 5.6 são exibidos os tempos das requisições médias obtidos apenas para a política de Carga, para 100 requisições uma em cada conexão TCP, para as taxas de 1, 2, 4, 5, 17, 10, e 15 requisições por segundo. Na Tabela 5.4 são exibidos os valores com os intervalos de confiança de 95% e o desvio padrão relativo a 33 repetições de cada experimento. Conjuntamente são exibidos os valores obtidos do Código 3 na Seção 5.5.3, com o objetivo de facilitar a comparação entre ambos.

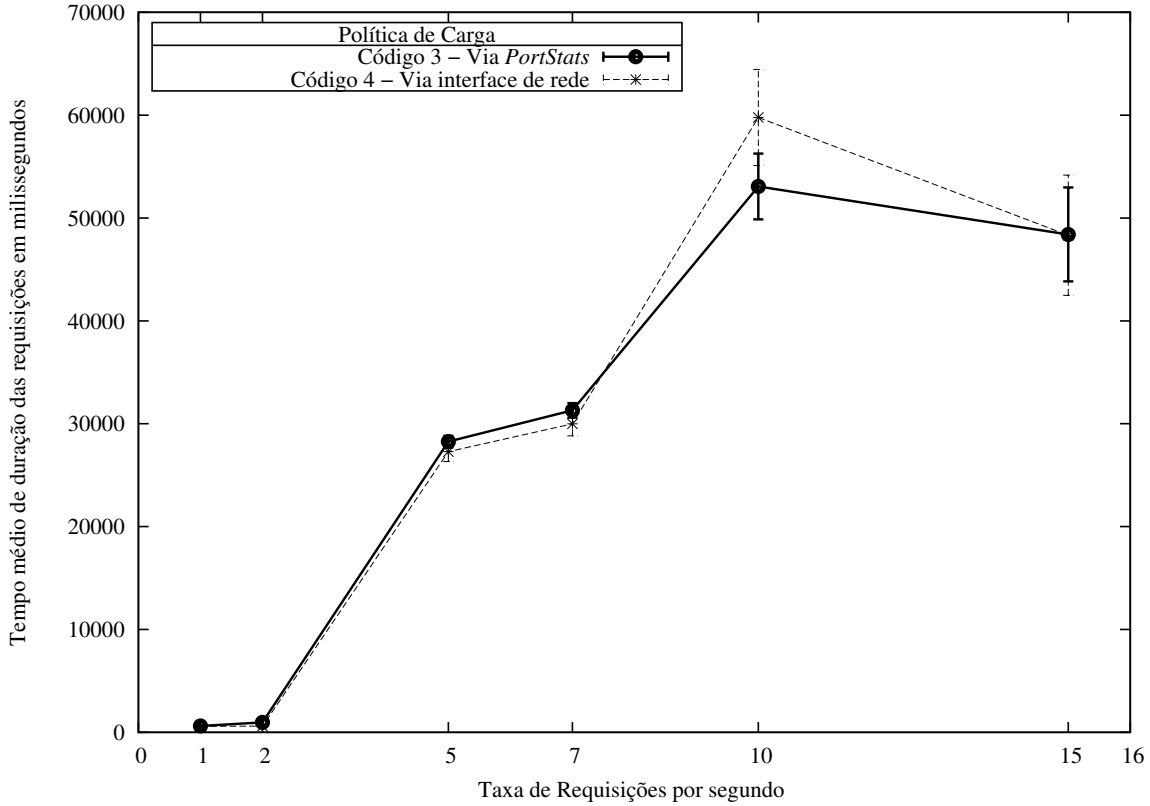

Figura 5.6: Carga medida via *PortStats* e carga medida na interface de rede

Neste caso, cada servidor e o controlador recebem uma nova interface de rede conectada a um switch Ethernet comum. Através dessa nova interface o controlador utiliza um serviço de *sockets* instalado nos servidores. O serviço fornece a quantidade

de bytes já transmitidos nas interfaces de rede de cada um dos servidores, que está conectada via switch OpenFlow. Deste modo, a leitura da carga é realizada diretamente na interface de rede de cada servidor. O Código 4 é similar ao proposto no trabalho relacionado da Seção 3.6 - Balanceamento de Carga Web em SDN.

Quase todos os valores obtidos são muito próximos aos obtidos no Código 3, pela política de Carga na Seção 5.5.3. Tais medições indicam a eficiência do Código 3 - Inserção Proativa de todas as Regras de Volta e Reativa da Regra de Ida, como validação da solução proposta, com esta comparação.

Tabela 5.4: Código 4 - Tempo médio das requisições com a carga medida na interface de rede (em milissegundos)

Requisições por Segundo	Política Carga	Intervalo de Confiança 95%	Desvio Padrão
1	592,40	587,13 - 597,67	18,39
2	603,81	602,62 - 605,00	4,16
5	27277,87	26335,12 - 28220,62	3291,80
7	29990,25	28816,45 - 31164,07	4098,57
10	59770,24	55102,72 - 64437,77	16297,53
15	48322,57	42472,81 - 54172,33	20425,52

5.6 Discussão

Em todos os experimentos realizados as taxas de 1 e 2 requisições por segundo não são relevantes por serem muito leves. As medições com a taxa de 5 novas requisições por segundo obtiveram o menor desvio padrão no Código 3 da solução proposta. Tal fato ocorre em uma região de desempenho normal do switch com leituras via *PortStats*, conforme levantado empiricamente. A taxa de 10 novas requisições por segundo já produz um considerável atraso frente à vazão da rede. Estima-se que a leitura da carga na taxa de 15 novas requisições por segundo seja retornada ao controlador após a escolha do caminho com a menor carga, conforme no Algoritmo 2 da Seção 4.2.2, o valor da carga real de bytes trafegados em cada enlace somente é atualizado após a escolha ser realizada. Portanto pode-se considerar que os resultados obtidos nas taxas de 10 e 15 novas requisições por segundo já estejam com atrasos grandes, invalidando a escolha do caminho de menor carga pelo controlador.

Pode-se observar que em cada código das Seções 5.5.1, 5.5.2 e 5.5.3, a divisão de tráfego através da política de Carga foi evoluindo para uma maior aproximação dos valores obtidos da política *Round Robin*. Nos experimentos a política de *Round Robin*

é a mais justa na divisão, pois o tráfego e os servidores são homogêneos. Isso é devido ao fato de que no Código 2 as regras de volta são inseridas antes das de ida, eliminando o atraso da inserção da regra de volta (inserida conjuntamente antes da regra de ida) e alguma perda de pacotes. No Código 3, todas as regras de volta já estão inseridas proativamente no início de cada experimento, o que elimina o atraso para a inserção da regra de volta, bastando apenas a inserção reativa da regra de ida de cada novo fluxo.

Finalmente, o Código 3, com a leitura das estatísticas do switch OpenFlow através da primitiva *Portstats* e regras de volta proativas é comparado com o Código 4, que utiliza serviço de *sockets* para ler as estatísticas de bytes transmitidos por cada servidor. O Código 4 necessita de uma rede adicional com um switch e a adição de mais uma interface de rede no controlador e nos servidores do experimento.

A solução proposta no presente trabalho obtém um bom resultado referente ao tempo médio das requisições de serviço, no caso HTTP. Na Tabela 5.5 é exibida uma comparação da solução proposta no Código 3 na Seção 5.5.3 com a solução através de *sockets* para ler a carga de cada servidor em sua interface de rede do Código 4, na Seção 5.5.4. Pode-se observar que no caso da taxa de 5 e 7 novas requisições por segundo os valores são muito próximos, como pode ser vista na Tabela 5.5. Considerando-se o comportamento observado do switch utilizado, esperam-se melhores resultados em switches SDN de melhor desempenho.

Tabela 5.5: Redução percentual do tempo de requisição

	Divisão de Carga através de *PortStats* (A)	Divisão de Carga através de *sockets* (B)	
Requisições por segundo	Tempo médio (milissegundos)	Tempo médio (milissegundos)	Redução Percentual (B/A).100% - 100%
1	610,43	592,40	-2,95%
2	963,23	603,81	-37,31%
5	28261,90	27277,87	-3,48%
7	31298,52	29990,26	-4,18%
10	53069,18	59770,24	12,63%
15	48404,85	48322,57	-0,17%

Finalmente, na Tabela 5.6 é exibida uma tabela com as diferenças entre o Código 3 - divisão de carga através de *PortStats* com regras proativas e o Código 4 - divisão de carga medida na interface de rede através de *sockets*.

Tabela 5.6: Diferenças entres as divisões de carga através de *PortStats* e através de *sockets*

Divisão de Carga através de *Portstats*	Divisão de Carga através de *sockets*
Carga lida via primitiva OpenFlow no switch	Carga lida na interface de n servidores réplica
Uma leitura	n leituras
Utiliza a conexão existente entre o controlador e o switch OpenFlow	Usa uma rede adicional interligando interfaces extras no controlador e nos n servidores
Evento do OpenFlow	Executa processos adicionais nos servidores (processo usando *sockets* para leitura da carga local)

Capítulo 6

Conclusão e Trabalhos Futuros

6.1 Conclusão

Nesta dissertação foram implementados três algoritmos de divisão de tráfego utilizando uma rede SDN através do protocolo OpenFlow. O algoritmo que resulta no maior desempenho dos três não se destacou nos experimentos quanto ao esperado, mas obteve resultados próximos aos de uma solução similar implementada via *sockets*. Mesmo ocorrendo falhas de processamento de pacotes no switch conforme na Seção 5.3.2.1 os experimentos foram realizados satisfatoriamente.

Pelas implementações e resultados obtidos pode-se confirmar a validade das soluções de divisão de tráfego desta dissertação. Novas gerações de dispositivos SDN e do protocolo OpenFlow podem ser esperadas para superar problemas de desempenho inadequado e sanar falhas detectadas nos experimentos realizados. Sem dúvida os switches SDN possuem requisitos que demandam maior desempenho do hardware, comparado com redes em arquiteturas anteriores não SDN. Tal fato indica que os dispositivos de comutação devem sofrer ainda considerável melhora no desempenho com a maturação dos produtos disponíveis.

O switch utilizado foi alvo de aprendizado e de razoável interesse. Com a observação de seu limitado desempenho nos experimentos, conforme exibido na próxima seção, foi publicado um artigo: "Avaliação de Desempenho de Planos de Dados OpenFlow" [Costa et al., 2016].

6.1.1 Limitações de Desempenho

Embora o switch OpenFlow HP 2920-24G utilizado nos experimentos não tenha um desempenho satisfatório, como comprovado em [Costa et al., 2016] e confirmado nos

experimentos, foi realizado um dimensionamento para obter número de fluxos, conexões simultâneas e rajadas de novas conexões para que o switch se comportasse dentro de um padrão adequado de uso normal. Tal limitação não se aplicaria aos experimentos realizados que poderiam ter mais fluxos, mais conexões simultâneas e rajadas com maiores números de novas conexões.

O principal problema de desempenho são os atrasos do plano de dados do switch. O atraso da operação de modificação do cabeçalho dos pacotes TCP provavelmente é o que mais impacta o desempenho do switch. No trabalho de Costa et al. [2016] o switch HP 2920-24G teve o atraso medido em cerca de 1 milissegundo, mas com apenas um fluxo de dados no switch. No caso de uma carga elevada podemos considerar que a operação de modificação do cabeçalho dos pacotes das requisições, nas regras de ida e nas regras de volta atrasem pelos menos 2 milissegundos.

O atraso do tempo de leitura via *Portstats* é de 40 milissegundos no switch utilizado, conforme citado na Seção 4.2.2.2. Como o ambiente do controlador POX possui um alto tempo de resposta [Shalimov et al., 2013], pode-se supor que controladores e switches de menor tempo de resposta fornecerão uma solução para divisão de tráfego em data center de maneira satisfatória.

O ambiente controlador e switch OpenFlow utilizados responde erraticamente a rajadas de 20 novos fluxos, conforme avaliado previamente para dimensionar os experimentos. Mesmo *pipelines* de 20 requisições (em paralelo) em uma mesma conexão HTTP, via ferramenta HTTPERF, resultaram em altos desvios padrões e perdas. Acredita-se que isto seja resultado do atraso da operação de reescrita do cabeçalho TCP.

6.2 Trabalhos Futuros

Após os resultados obtidos, variações da solução podem ser executadas utilizando outros controladores, outros protocolos SDN e outros switches físicos. Futuramente também pode ser configurado um cenário diferente no qual não seja utilizada a regra de reescrita de cabeçalho - *Packet Rewrite*, por ser a provável causa do atraso existente na solução proposta. Nas sessões abaixo são indicadas possíveis direções para trabalhos futuros.

6.2.1 Outras Arquiteturas de Rede

Embora a solução tenha sido desenvolvida em um ambiente em SDN, outras soluções existentes não SDN [Bourke, 2001] podem ser avaliadas com o mesmo switch físico para

validar os ganhos obtidos.

6.2.2 Outros Controladores

Poderão ser utilizados controladores de maior desempenho para avaliar a solução proposta, com isso, estima-se melhorias nos resultados. Por exemplo, utilizar o Controlador Beacon que, conforme uma avaliação tem maior desempenho [Erickson, 2013]. Também utilizar controladores ou ambientes de controle OpenFlow além do Beacon, como Onos [Berde et al., 2014], Onix [Koponen et al., 2010], entre outros, mas também ambientes não OpenFlow como Netconf (IETF RFC 6241) e Netflow (Cisco RFC 3954).

6.2.3 Switches de Maior Desempenho no Plano de Dados

Outros switches OpenFlow também podem ser utilizados para exibir melhores resultados. Devido à sua menor capacidade, por ser uma implementação via software, o switch dos experimentos não atingiu desempenho razoável. Vários switches OpenFlow comerciais de maior vazão real, sejam de 1 Gb/s, 10 Gb/s, 40 Gb/s e até mesmo de 100 Gb/s já estão disponíveis, por exemplo, a linha de switches Xpliant® da Cavium[1].

6.2.4 Novas Plataformas

Novas plataformas surgiram desde o término dos experimentos e podem ser um campo promissor para se avaliar e desenvolver a solução proposta. A plataforma PISA - *Protocol Independent Switch Architecture*, anunciada em Julho de 2016[2] pela empresa Barefoot Networks[3], exibe uma série de processadores denominada Toffino™, com a maior vazão anunciada até então de **6,5 Tb/s**, suportando 65 portas de 100 Gb/s. O ambiente é totalmente programável através da linguagem P4[4]. De acordo com o anunciado pela Barefoot Networks, PISA é uma plataforma promissora para testes e validação da solução proposta.

[1] `http://www.cavium.com/Table.html#Xpliant`
[2] `http://www.sdxcentral.com/articles/news/barefoot-networks/2016/06/`
[3] `https://www.barefootnetworks.com/`
[4] `http://p4.org`

Referências Bibliográficas

Agarwal, S.; Kodialam, M. & Lakshman, T. (2013). Traffic Engineering in Software Defined Networks. Em *INFOCOM, 2013 Proceedings IEEE*, pp. 2211–2219. ISSN 0743-166X.

Akyildiz, I. F.; Anjali, T.; Chen, L.; De Oliveira, J. C.; Scoglio, C.; Sciuto, A.; Smith, J. A. & Uhl, G. (2003). Invited A New Traffic Engineering Manager for DiffServ/MPLS Networks: Design and Implementation on an IP QoS Testbed. *Comput. Commun.*, 26(4):388–403. ISSN 0140-3664.

Akyildiz, I. F.; Lee, A.; Wang, P.; Luo, M. & Chou, W. (2014). A Roadmap for Traffic Engineering in SDN-OpenFlow Networks. *Computer Networks*, 71(0):1–30. ISSN 1389-1286.

Al-Fares, M.; Loukissas, A. & Vahdat, A. (2008). A Scalable, Commodity Data Center Network Architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74. ISSN 0146-4833.

Armbrust, M.; Fox, A.; Griffith, R.; Joseph, A. D.; Katz, R.; Konwinski, A.; Lee, G.; Patterson, D.; Rabkin, A.; Stoica, I. & Zaharia, M. (2010). A View of Cloud Computing. *Commun. ACM*, 53(4):50–58. ISSN 0001-0782.

Augustin, B.; Friedman, T. & Teixeira, R. (2007). Measuring load-balanced paths in the internet. Em *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pp. 149–160, New York, NY, USA. ACM.

Awduche, D.; Chiu, A.; Elwalid, A.; Widjaja, I. & Widjaja, I. (2002). RFC 3272: Overview and Principles of Internet Traffic Engineering. Relatório técnico, IETF.

Barabas, T.; Ionescu, D. & Veres, S. (2011). PIM-SSM within DiffServ-aware MPLS Traffic Engineering. Em *Applied Computational Intelligence and Informatics (SACI), 2011 6th IEEE International Symposium on*, pp. 263–268.

Barroso, L. A.; Clidaras, J. & Hölzle, U. (2013). The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition. *Synthesis Lectures on Computer Architecture*, 8(3):1–154.

Benson, T.; Akella, A. & Maltz, D. A. (2010). Network traffic characteristics of data centers in the wild. Em *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 267–280. ACM.

Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W. & Parulkar, G. (2014). Onos: Towards an open, distributed sdn os. Em *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pp. 1–6, New York, NY, USA. ACM.

Bourke, T. (2001). *Server load balancing*. "O'Reilly Media, Inc.".

Cardellini, V.; Casalicchio, E.; Colajanni, M. & Yu, P. S. (2002). The state of the art in locally distributed web-server systems. *ACM Computing Surveys (CSUR)*, 34(2):263–311.

CISCO (2015a). White Paper - Cisco Global Cloud Index: Forecast and Methodology, 2014–2019. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf`. [Online; acessado 12-Fevereiro-2016].

CISCO (2015b). White Paper - Cisco Visual Networking Index: Forecast and Methodology, 2014-2019. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf`. [Online; acessado 12-Fevereiro-2016].

CISCO (2016a). How does load balancing work? `http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/5212-46.html`. [Online; acessado 12-Julho-2016].

CISCO (2016b). White Paper - Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf`. [Online; acessado 12-Fevereiro-2016].

Costa, L. C.; Vieira, A. B.; Silva, E. B.; Macedo, D. F.; Gomes, G.; Correia, L. H. & Vieira, L. F. (2016). Avaliação de desempenho de plano de dados openflow. Em *XXXIV Simpósio Brasileiro de Rede de Computadores, 2016*. SBC.

de Oliveira, R.; Shinoda, A.; Schweitzer, C. & Rodrigues Prete, L. (2014). Using Mininet for Emulation and Prototyping Software-Defined Networks. Em *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pp. 1–6.

Erickson, D. (2013). The beacon openflow controller. Em *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pp. 13–18, New York, NY, USA. ACM.

Farmanbar, H. & Zhang, H. (2014). Traffic Engineering for Software-defined Radio Access Networks. Em *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–7.

Feamster, N.; Borkenhagen, J. & Rexford, J. (2003). Guidelines for Interdomain Traffic Engineering. *SIGCOMM Comput. Commun. Rev.*, 33(5):19–30. ISSN 0146-4833.

Ferraris, F.; Franceschelli, D.; Gioiosa, M.; Lucia, D.; Ardagna, D.; Di Nitto, E. & Sharif, T. (2012). Evaluating the Auto Scaling Performance of Flexiscale and Amazon EC2 Clouds. Em *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on*, pp. 423–429.

Fortz, B.; Rexford, J. & Thorup, M. (2002). Traffic Engineering With Traditional IP Routing Protocols. *Communications Magazine, IEEE*, 40(10):118–124. ISSN 0163-6804.

Gandhi, R.; Liu, H. H.; Hu, Y. C.; Lu, G.; Padhye, J.; Yuan, L. & Zhang, M. (2014). Duet: Cloud Scale Load Balancing with Hardware and Software. Em *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pp. 27–38, New York, NY, USA. ACM.

Guedes, D.; Vieira, L.; Vieira, M.; Rodrigues, H. & Nunes, R. V. (2012). Redes definidas por software: uma abordagem sistêmica para o desenvolvimento de pesquisas em redes de computadores. *Minicursos do Simpósio Brasileiro de Redes de Computadores-SBRC 2012*, 30(4):160–210.

Handigol, N.; Seetharaman, S.; Flajslik, M. & Johari, R. (2009). Plug-n-Serve: Load-balancing web traffic using OpenFlow. *Demo at ACM SIGCOMM*.

Jain, S.; Kumar, A.; Mandal, S.; Ong, J.; Poutievski, L.; Singh, A.; Venkata, S.; Wanderer, J.; Zhou, J.; Zhu, M.; Zolla, J.; Hölzle, U.; Stuart, S. & Vahdat, A. (2013). B4: Experience with a Globally-deployed Software Defined Wan. *SIGCOMM Comput. Commun. Rev.*, 43(4):3–14. ISSN 0146-4833.

Koponen, T.; Casado, M.; Gude, N.; Stribling, J.; Poutievski, L.; Zhu, M.; Ramanathan, R.; Iwata, Y.; Inoue, H.; Hama, T. et al. (2010). Onix: A distributed control platform for large-scale production networks. Em *OSDI*, volume 10, pp. 1–6.

Kurose, J. F. (2005). *Computer Networking: A Top-Down Approach Featuring the Internet, 3/E*. Pearson Education India.

Leduc, G.; Abrahamsson, H.; Balon, S.; Bessler, S.; D'Arienzo, M.; Delcourt, O.; Domingo-Pascual, J.; Cerav-Erbas, S.; Gojmerac, I.; Masip, X.; Pescapè, A.; Quoitin, B.; Romano, S.; Salvadori, E.; Skivée, F.; Tran, H.; Uhlig, S. & Ümit, H. (2006). An Open Source Traffic Engineering Toolbox. *Computer Communications*, 29(5):593 – 610. ISSN 0140-3664. Networks of Excellence.

Lim, C. L. & Tang, A. (2013). Traffic Engineering with Elastic Traffic. Em *Global Communications Conference (GLOBECOM), 2013 IEEE*, pp. 3095–3101.

McKeown, N.; Anderson, T.; Balakrishnan, H.; Parulkar, G.; Peterson, L.; Rexford, J.; Shenker, S. & Turner, J. (2008). OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74. ISSN 0146-4833.

ONF (2016). Open Networking Foundation. `http://www.opennetworking.org`. [Online; acessado 12-Fevereiro-2016].

Onosproject (2016). White Paper - Onos Overview Whitepaper. `http://onosproject.org/wp-content/uploads/2014/11/Whitepaper-ONOS-final.pdf`. [Online; acessado 02-Agosto-2016].

OpenDaylight (2016). OpenDaylight Platform Overview. `https://www.opendaylight.org/platform-overview`. [Online; acessado 02-Agosto-2016].

Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; Amidon, K. & Casado, M. (2015). The design and implementation of open vswitch. Em *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 117–130, Oakland, CA. USENIX Association.

Rodrigues, C. P.; Costa, L. C.; Vieira, M. A. M.; Vieira, L. F. M.; Macedo, D. F. & Vieira, A. B. (2015). Avaliação de balanceamento de carga web em redes definidas por software. Em *XXXIII Simpósio Brasileiro de Rede de Computadores, 2015*, pp. 585–597. SBC.

Rodrigues Prete, L.; Schweitzer, C.; Shinoda, A. & Santos de Oliveira, R. (2014). Simulation in an SDN network scenario using the POX Controller. Em *Communications and Computing (COLCOM), 2014 IEEE Colombian Conference on*, pp. 1–6.

Shalimov, A.; Zuikov, D.; Zimarina, D.; Pashkov, V. & Smeliansky, R. (2013). Advanced study of sdn/openflow controllers. Em *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR '13, pp. 1:1–1:6, New York, NY, USA. ACM.

Stevens, W. R. (1993). *TCP/IP Illustrated (Vol. 1): The Protocols*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. ISBN 0-201-63346-9.

Summers, J.; Brecht, T.; Eager, D. & Wong, B. (2012). Methodologies for generating http streaming video workloads to evaluate web server performance. Em *Proceedings of the 5th Annual International Systems and Storage Conference*, SYSTOR '12, pp. 2:1–2:12, New York, NY, USA. ACM.

Uppal, H. & Brandon, D. (2010). Openflow based load balancing. *CSE561: Networking Project Report, University of Washington*.

Valdivieso Caraguay, A.; Barona Lopez, L. & Garcia Villalba, L. (2013). Evolution and Challenges of Software Defined Networking. Em *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–7.

Van den Bossche, R.; Vanmechelen, K. & Broeckhove, J. (2010). Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. Em *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 228–235. IEEE.

Wang, N.; Fagear, A. & Pavlou, G. (2011a). Adaptive Post-failure Load Balancing in Fast Reroute Enabled IP Networks. Em *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pp. 470–477.

Wang, R.; Butnariu, D. & Rexford, J. (2011b). OpenFlow-based Server Load Balancing Gone Wild. Em *Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'11, pp. 12–12, Berkeley, CA, USA. USENIX Association.

Xu, D.; Fan, B. & Liu, X. (2014). Efficient Server Provisioning and Offloading Policies for Internet Datacenters With Dynamic Load-Demand. *IEEE Transactions on Computers*, 99(PrePrints):1. ISSN 0018-9340.