Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Leonardo Chinelate Costa

Balanceamento de Carga Utilizando Planos de Dados OpenFlow Comerciais

Juiz de Fora
2016

Leonardo Chinelate Costa

Balanceamento de Carga Utilizando Planos de Dados OpenFlow Comerciais

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, na área de concentração em Redes de Computadores, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Alex Borges Vieira

Coorientador: Daniel Fernandes Macedo

Juiz de Fora

2016

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Costa, Leonardo Chinelate.
Balanceamento de Carga Utilizando Planos de Dados OpenFlow Comerciais / Leonardo Chinelate Costa. – 2016.
106 f. : il.

Orientador: Alex Borges Vieira
Coorientador: Daniel Fernandes Macedo
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2016.

1. Redes Definidas por Software. 2. Balanceamento de Carga. 3. Avaliação de Desempenho de Planos de Dados OpenFlow. I. Vieira, Alex Borges, orient. II. Macedo, Daniel Fernandes, coorient. III. Título

Leonardo Chinelate Costa

Balanceamento de Carga Utilizando Planos de Dados OpenFlow Comerciais

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, na área de concentração em Redes de Computadores, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em: 10 de junho de 2016

BANCA EXAMINADORA

Prof. D. Sc. Alex Borges Vieira - Orientador
Universidade Federal de Juiz de Fora

Prof. D. Sc. Daniel Fernandes Macedo - Coorientador
Universidade Federal de Minas Gerais

Prof. D. Sc. Magnos Martinello
Universidade Federal do Espírito Santo

Prof. D. Sc. Marcelo Ferreira Moreno
Universidade Federal de Juiz de Fora

AGRADECIMENTOS

Agradeço a Deus, por me dar força e coragem em todos os momentos desses dois anos de caminhada. Tenho a certeza de que Suas ações, principalmente nos detalhes, foram decisivas nas decisões que tomei nos últimos anos. Que Ele continue me abençoando, nos bons e maus momentos, para sempre!

Agradeço aos meus pais, Bernadete e Emanoel, os maiores colaboradores da minha vida, pelo amor incondicional, pela compreensão nos momentos de ausência, por todo o esforço para criar e formar seus filhos, por todo o apoio irrestrito em todos os projetos de vida. Sem eles, nada seria possível. As minhas conquistas são, na verdade, conquistas deles. A eles, a minha gratidão eterna!

Agradeço ao meu irmão Eduardo, pelo incentivo na vida acadêmica e pelo exemplo como pessoa e como profissional, sempre com extrema dedicação e amor ao que faz. Seu apoio sempre tornou tudo mais fácil. Foi um prazer caminhar ao seu lado, por todos os momentos de dificuldades e conquistas. Tenho certeza de que coisas boas estão reservadas para nós. Obrigado pela amizade, mestre!

Agradeço à minha namorada Cristiane, que mesmo de longe, foi a companhia mais presente que alguém poderia ter. Obrigado pelo amor, pelo companheirismo e pelo apoio em vários momentos de dificuldade. Parte deste trabalho nasceu da nossa parceria. Espero que, em breve, possamos compartilhar nossas batalhas e conquistas lado a lado. Ao meu bem mais precioso, todo o meu amor!

Agradeço ao meu orientador Alex, pela parceria desde a época da graduação. Seu apoio foi fundamental na decisão de entrar para o mestrado e na conclusão de mais esta etapa. Desejo a ele muita força para continuar! Agradeço também ao meu coorientador Daniel, pela disponibilidade e pela valorosa contribuição para este trabalho, que sem a sua ajuda, seria muito mais difícil de ser realizado. Agradeço aos companheiros da UFMG e da UFLA - Erik, Geraldo e professor Luiz Henrique - pela ajuda e pelo conhecimento compartilhado na realização do trabalho.

Agradeço aos amigos e colegas de mestrado que estiveram ao meu lado nessa caminhada. Obrigado por tornarem muito mais fácil e mais empolgante essa longa jornada. Um agradecimento especial aos amigos do NetLab, com os quais compartilhei momentos sempre muito agradáveis e divertidos.

Agradeço à UFJF e ao PGCC por possibilitarem essa formação. Obrigado pelo apoio e pela bolsa de pesquisa, indispensáveis para a conclusão dessa etapa. Obrigado por me enriquecerem pessoalmente e profissionalmente.

"When you're in pain
When you think you've had enough
Don't ever give up!"
Up & Up (Coldplay)

RESUMO

O paradigma de Redes Definidas por Software (SDN) vem mudando a forma como gerenciar e operar redes de computadores através da sua principal ideia, a separação dos planos de dados e de controle. O protocolo OpenFlow implementa este conceito e, devido às vantagens de menor custo de operação e maior facilidade de adaptação a projetos de comutadores já existentes, é encontrado hoje em diversos equipamentos de rede comercializados por muitas empresas. Com o uso do paradigma SDN e do protocolo OpenFlow, a inovação e a evolução da rede são facilitadas. Dessa forma, muitos serviços típicos de rede podem ser repensados, de forma a torná-los mais flexíveis. Um desses serviços é o balanceamento de carga. Neste trabalho é realizado um estudo sobre a viabilidade de se implementar um balanceador de carga OpenFlow em uma rede SDN real, considerando as restrições existentes nos equipamentos OpenFlow comerciais atuais. Para isso, foi proposto um modelo de balanceamento de carga em SDN que leva em consideração diferentes perfis de carga mais realistas e que é baseado na utilização de diferentes políticas para a realização do balanceamento. Contudo, antes de reproduzir esse cenário em um ambiente real, foi realizada uma avaliação de desempenho de alguns planos de dados OpenFlow a fim de se verificar se as implementações OpenFlow atuais são capazes de suportar o balanceamento de carga ou outros serviços em uma rede de produção. Foi avaliada a qualidade de diferentes implementações OpenFlow de hardware switches comerciais e de implementações *open source* de software switches, através de métricas de desempenho em operações típicas de um switch OpenFlow. Os resultados mostram que as implementações OpenFlow dos hardware switches avaliados ainda não atingiram um nível de maturidade suficiente para serem utilizadas em larga escala. Apesar de desempenhos similares entre os modos OpenFlow e *legacy* na maioria dos casos, as implementações OpenFlow em hardware apresentaram problemas como implementações incompletas do padrão, baixo número de regras suportadas, funcionamento instável para tabelas de fluxo cheias e problemas no processamento de múltiplos comandos.

Palavras-chave: Redes Definidas por Software. Balanceamento de carga. Avaliação de desempenho de planos de dados OpenFlow.

ABSTRACT

Software Defined Networks paradigm (SDN) is changing the way how we manage and operate computer networks by its main idea, the decoupling of data and control planes. OpenFlow protocol implements this concept and, due to the advantages of lower operating expenditures and greater ease of adaptation to existing switches projects, it is found today in various network equipment sold by many companies. Using SDN paradigm and OpenFlow protocol, network innovation and evolution are facilitated. Thus, many typical network services can be rethought in order to make them more flexible. An example of such services is load balancing. This work is a study about the feasibility of implementing an OpenFlow load balancer in a real SDN network, considering the restrictions in current commercial OpenFlow equipment. For this, we propose a SDN load balancing which considers different more realistic workload profiles and is based on using different policies for performing the balancing. However, before reproducing this scenario in a real environment, a performance evaluation of some OpenFlow data planes was conducted in order to verify that the current OpenFlow implementations are able to support load balancing or other services in production networks. The quality of different commercial OpenFlow hardware switch implementations and open source software switch implementations was evaluated, using performance metrics in typical operations of an OpenFlow switch. The results show that OpenFlow implementations of the evaluated hardware switches have not yet reached a sufficient level of maturity to be used on a large scale. Despite similar performances between OpenFlow and legacy modes in most cases, OpenFlow hardware implementations have presented problems such as standard incomplete implementations, low number of supported rules, unstable operation for full flow tables and problems in processing multiple commands.

Key-words: Software Defined Networks. Load balancing. Performance evaluation of OpenFlow data planes.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SDN	Software Defined Networks
ONF	Open Networking Foundation
GSMP	General Switch Management Protocol
BGP	Border Gateway Protocol
API	Application Programming Interface
TCAM	Ternary Content Addressable Memory
SRAM	Static Random Access Memory
DRAM	Dynamic Random Access Memory
VLAN	Virtual Local Area Network
MPLS	MultiProtocol Label Switching
DDoS	Distributed Denial of Service
TCP	Transmission Control Protocol
IP	Internet Protocol
MAC	Media Access Control
UDP	User Datagram Protocol
QoS	Quality of Service
RTT	Round Trip Time
TTL	Time to Live
ASIC	Application Specific Integrated Circuits
NAT	Network Address Translation

SUMÁRIO

1 INTRODUÇÃO

O paradigma de Redes Definidas por Software (SDN) vem mudando a forma de criar, modificar e gerenciar as redes de computadores. E com o crescimento da importância das redes como parte crítica da infraestrutura dos mais variados serviços cotidianos, o paradigma encontrou um cenário ideal para a sua utilização e seu desenvolvimento. Nesse cenário, a estabilidade da rede passou a ser crucial para o sucesso dos negócios, o que tornou praticamente inviável a experimentação de novas tecnologias, protocolos e configurações sob ambientes em produção. Isso, consequentemente, culminou com a "ossificação" da infraestrutura das redes.

Com o uso do paradigma SDN, que tem como ideia principal a separação dos planos de dados e de controle, a inovação e a evolução da rede são facilitadas. Dessa forma, o desenvolvimento de novos serviços e a realização de novos experimentos se tornaram possíveis nesses ambientes, sem que se afetasse o tráfego ou a disponibilidade da rede. Por consequência, diversos serviços típicos de rede estão sendo repensados, de forma a torná-los mais flexíveis.

Considerando esse contexto, o paradigma SDN tem atraído a atenção tanto da comunidade acadêmica quanto da indústria. Apesar de existirem algumas propostas mais flexíveis de SDN, tais como o POF e o B4 (SONG, 2013; BOSSHAART et al., 2014), grande parte dessa atenção se voltou ao padrão OpenFlow. Desenvolvido em 2008, o padrão OpenFlow é um dos principais protocolos relacionados a SDN e está associado a esse paradigma desde a sua concepção. Atualmente, o OpenFlow é a plataforma SDN mais popular, utilizada na maioria dos desenvolvimentos e em pesquisas envolvendo o paradigma. O seu sucesso se deve, em grande parte, à facilidade de sua implementação nos projetos já existentes de switches e roteadores de rede. Inclusive, foi isso que despertou o interesse de vendedores de equipamentos, operadores de rede e também de empresas, como Google, Microsoft e Facebook. Essas empresas, no ano de 2011, formaram a Open Networking Foundation (ONF), uma organização que tem como objetivo principal promover a tecnologia OpenFlow e o paradigma SDN no mercado (ONF, 2016).

Esse crescente interesse pelo paradigma SDN e pela tecnologia OpenFlow, seja por parte da indústria de tecnologia em redes, dos operadores de rede ou pela própria academia, aliado ao fato de ambos serem relativamente recentes e ainda estarem sob intenso desenvolvimento, motivam os esforços em pesquisas e trabalhos relacionados ao tema. E, de fato, muitas pesquisas e trabalhos relacionados à SDN e ao padrão OpenFlow vêm sendo realizados nos últimos anos. Além disso, em termos de infraestrutura, a utilização do protocolo OpenFlow em redes operacionais vem se tornando uma realidade.

O foco do presente trabalho é o balanceamento de carga em SDN utilizando o protocolo OpenFlow. Contudo, para uma análise mais detalhada, esse problema geral

pode ser dividido em outros dois problemas mais específicos. O primeiro deles se refere ao serviço de balanceamento de carga propriamente dito em SDN e como ele é caracterizado atualmente. Já o segundo problema, que decorreu dos resultados parciais obtidos ao longo da metodologia do primeiro, se refere à caracterização das implementações do protocolo OpenFlow em equipamentos de rede. Esses problemas serão detalhados nas próximas seções.

1.1 O PROBLEMA DO BALANCEAMENTO DE CARGA

O balanceamento de carga é, notadamente, um mecanismo importante para serviços em redes de computadores tradicionais. Sua importância se dá a partir do fato de que redes como a Internet, por exemplo, vêm se tornando estruturas complexas devido à inclusão de novas tecnologias, fazendo com que serviços que são executados nessas redes devam ser capazes de lidar com milhares de requisições simultaneamente. Como existem milhares de requisições, é necessário utilizar uma estrutura de redundância para atender a todos os clientes de forma satisfatória e em um tempo suficientemente rápido.

Atualmente, uma típica configuração de serviços em redes consiste em um balanceador de carga que recebe requisições recém-chegadas e as distribui para múltiplos servidores pertencentes a uma rede. Normalmente, esses balanceadores são sistemas de hardware ou software altamente especializados para determinado tipo de serviço. Por serem altamente especializados, esses balanceadores são equipamentos muito caros, possuem regras rígidas de funcionamento e necessitam de administradores capacitados para sua operação (UPPAL and BRANDON, 2010).

Na literatura, são encontrados alguns esforços na direção de propor balanceamento de carga utilizando Redes Definidas por Software (SHERWOOD et al., 2009; HANDIGOL et al., 2009; UPPAL and BRANDON, 2010; WANG et al., 2011; RAGALATHA et al., 2013; ZHOU et al., 2014). Os trabalhos citados propõem o balanceamento de carga através da construção de uma arquitetura SDN com o padrão OpenFlow, porém nem todos eles avaliam o comportamento do balanceador proposto. E os poucos que realizam tal avaliação, o fazem em ambiente muito específico, sem se preocupar com o perfil de carga a qual serão submetidos os serviços a serem balanceados.

1.2 O PROBLEMA DA AVALIAÇÃO DO PLANO DE DADOS

Para uma adequada realização do balanceamento de carga ou de qualquer outro serviço de rede em uma SDN, é necessário também avaliar o comportamento dos comutadores da rede, que são elementos de extrema importância em sua arquitetura. Os comutadores (como roteadores ou switches) compõem o plano de dados – ou de encaminhamento – em uma Rede Definida por Software. Esse plano é responsável por apenas encaminhar pacotes,

de acordo com o que for definido pelo controlador, no plano de controle.

Como consequência do crescente interesse da indústria pelo OpenFlow, cada vez mais equipamentos de rede, como switches e roteadores, passaram a suportar o protocolo (KREUTZ et al., 2015). Tendo isso em vista, torna-se extremamente importante verificar se as implementações existentes de OpenFlow, tanto na forma de equipamentos comerciais quanto em implementações via software *open source*, possuem características de desempenho e robustez adequadas para o uso em redes de produção.

Contudo, ainda são raros os trabalhos que enfocam o plano de dados dos switches OpenFlow, bem como suas implementações e mecanismos internos (BIANCO et al., 2010; SÜNNEN, 2011; APPELMAN and DE BOER, 2012). E os que o fazem, avaliam o desempenho de apenas um plano de dados ou ainda analisam esse desempenho em operações que envolvem ações diretas do controlador. Existe, portanto, uma lacuna na avaliação de desempenho dos comutadores que implementam o protocolo OpenFlow quando consideramos apenas o plano de dados, descartando qualquer interação entre controlador e comutadores.

1.3 OBJETIVO DA DISSERTAÇÃO

Considerando os problemas apontados, o presente trabalho tem o objetivo de estudar uma maneira de implementar um balanceador de carga OpenFlow em uma rede SDN real, considerando perfis de carga de trabalho mais próximos da realidade e as restrições existentes nos equipamentos OpenFlow comerciais atuais.

Para isso, foi proposto um modelo de balanceamento de carga em SDN que leva em consideração diferentes perfis de carga e que é baseado na utilização de diferentes políticas para a realização do balanceamento. Além disso, foram mostrados os benefícios alcançados e as limitações do balanceador quando este tem que lidar com os diferentes tipos de carga. Esse modelo foi implementado e submetido a testes em um ambiente de rede virtual, com o intuito de ser aprimorado e reproduzido em uma rede operacional real.

No entanto, antes que se procedesse a reprodução desse modelo em um ambiente real, foi realizada também uma avaliação de desempenho de alguns planos de dados OpenFlow a fim de se verificar a viabilidade das implementações OpenFlow atuais em suportar balanceamento de carga ou outros serviços de rede em uma SDN real. Foi avaliada a qualidade de diferentes implementações OpenFlow de hardware switches comerciais e de implementações *open source* de software switches, através de métricas tradicionais de desempenho, que levam em conta aspectos como latência e *jitter*, em operações comuns dos comutadores.

1.4 CONTRIBUIÇÕES DA DISSERTAÇÃO

O presente trabalho apresenta contribuições de fato, tanto para a pesquisa em balanceamento de carga em SDN quanto para a avaliação de desempenho de comutadores OpenFlow. Em relação ao balanceamento, a principal contribuição foi a definição de uma avaliação mais realista de um balanceador de carga SDN, baseada nos seguintes pontos:

1. **Diferenciação do tipo de carga a ser balanceado.** Nos trabalhos anteriores, o balanceamento foi avaliado apenas sob a utilização de um único perfil de carga, retratando o acesso a um único arquivo genérico, com tamanho e tipo não representativos. No presente trabalho, foram individualizados três tipos de carga mais significativos, que podem melhor representar o acesso a diferentes perfis de arquivos. Além disso, foi identificado para quais perfis de carga o balanceamento proposto é mais atrativo.

2. **Reprodução do esquema de balanceamento em um ambiente de experimentação mais representativo.** Os trabalhos anteriores reproduziram a arquitetura em um ambiente de emulação de rede (Mininet) dentro de uma única máquina virtual; ambiente esse utilizado geralmente para testes e simulações anteriores à implementação de uma rede física de fato. Já o presente trabalho reproduziu uma arquitetura, também virtual, em execução sobre uma máquina física. Essa arquitetura é composta por máquinas virtuais distintas que representam as máquinas do modelo de balanceamento proposto, além de um switch virtual (Open vSwitch). Com o advento da virtualização, a nova arquitetura pode representar um cenário existente de fato, como por exemplo, em *data centers*, além de ser uma evolução do esquema de simulação de rede.

Já em relação à avaliação dos planos de dados OpenFlow, o presente trabalho trouxe as seguintes contribuições:

1. **Comparação do desempenho de implementações reais de switches OpenFlow em hardware e software.** Trabalhos anteriores apresentaram avaliações de poucos comutadores OpenFlow e sem foco nos planos de dados. Neste trabalho foram avaliadas oito implementações OpenFlow de hardware switches comerciais e de implementações *open source* de software switches.

2. **Avaliação da maturidade da implementação dos planos de dados OpenFlow.** Os resultados dos testes apresentam um indicativo do nível de desenvolvimento da tecnologia OpenFlow em hardware switches de marcas comerciais, além das limitações e restrições encontradas nessas implementações.

3. **Discussão sobre como implementar o balanceamento de carga em hardware switches OpenFlow de produção.** É elencado um conjunto de fatores que precisam ser considerados na análise de viabilidade do uso desses planos de dados na realização do balanceamento de carga em uma SDN real com o protocolo OpenFlow.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

A dissertação está organizada da seguinte forma: no capítulo 2 são apresentados os aspectos fundamentais do paradigma de Redes Definidas por Software, sua definição, um breve histórico de sua evolução, exemplos de aplicações práticas em SDN, além da descrição de seus componentes principais e do protocolo OpenFlow. No capítulo 3 são apresentados alguns aspectos referentes a desempenho em SDN e alguns trabalhos relacionados ao escopo dos dois problemas específicos tratados neste texto. No capítulo 4 será apresentada a pesquisa referente ao balanceamento de carga em SDN, com a apresentação da arquitetura utilizada, além da definição das diferentes políticas de balanceamento e dos diferentes cenários de simulação e seus respectivos resultados. No capítulo 5 será apresentada a pesquisa referente à avaliação de desempenho de hardware e software switches OpenFlow, com a comparação das diferentes implementações através de métricas de interesse, tendo em vista avaliar a maturidade das soluções comerciais existentes no mercado. No capítulo 6 a dissertação é concluída e são relatados possíveis trabalhos futuros.

2 REVISÃO SOBRE REDES DEFINIDAS POR SOFTWARE

Este capítulo apresenta os aspectos fundamentais do paradigma de Redes Definidas por Software. Inicialmente, na seção 2.1, serão apresentadas diversas definições do paradigma, além de algumas de suas características essenciais. Em seguida, na seção 2.2, será apresentado um breve histórico do desenvolvimento do paradigma SDN, partindo das ideias anteriores à sua proposição, passando pela formulação de seus conceitos fundamentais e culminando com a definição atual do paradigma. Na seção 2.3 serão listados os principais componentes de uma arquitetura SDN típica, com suas características principais e algumas observações. Na seção 2.4 será dado um maior destaque ao protocolo OpenFlow, o principal padrão que permite a implementação do paradigma SDN. Por fim, na seção 2.5, serão apresentados exemplos de aplicações e serviços de rede que podem se beneficiar com a utilização dos conceitos de SDN.

2.1 DEFINIÇÃO

De maneira sucinta, Redes Definidas por Software (SDN) são um paradigma de redes de computadores que define a realização do encaminhamento de pacotes de uma maneira diferente da tradicional. A implementação do paradigma consiste na inserção de um novo elemento na rede - o controlador – que passa a ser o responsável por definir as ações que devem ser tomadas pelos elementos comutadores (switches ou roteadores) no encaminhamento de pacotes (MACEDO et al., 2015). Essa proposta de separar a entidade controladora e tomadora de decisões (plano de controle) da entidade controlada e que executa o que foi determinado (plano de dados) é justamente a base sobre a qual está assentada a formulação do paradigma. Assim, essa nova ideia de inserir controle sobre as operações da rede oferece um contraponto ao modelo tradicional de repasse de dados, no qual o controle e a operação de encaminhamento são executados pelo equipamento de comutação da rede. No entanto, a ideia por trás do paradigma SDN é muito mais profunda e significativa do que a simples inserção de um elemento de controle na rede. As definições a seguir irão tratar alguns aspectos contidos na formulação do paradigma de uma forma mais detalhada.

O termo SDN, em si, foi criado recentemente, contudo todo o conceito por trás da sigla SDN vem sendo desenvolvido desde meados da década de 1990 (ZILBERMAN et al., 2015). Portanto, múltiplas definições do que vem a ser de fato SDN têm sido elaboradas ao longo dos últimos anos. Uma das definições amplamente utilizadas é justamente a da Open Networking Foundation (ONF), a organização criada para a promoção, adoção e implementação de SDN na indústria, citada anteriormente. Segundo essa definição, SDN é uma arquitetura emergente, dinâmica, gerenciável, de ótimo custo-benefício, adaptável, ideal para a natureza dinâmica das aplicações de hoje, e que deve, principalmente, ser

capaz de separar o controle da rede e as funções de encaminhamento, permitindo que o controle da rede se torne diretamente programável e que a infraestrutura subjacente seja abstraída para aplicações e serviços de rede (ONF, 2016).

De forma mais específica, o ponto chave da definição anterior é o mesmo dos trabalhos de NUNES et al. (2014); ZILBERMAN et al. (2015); SEZER et al. (2013); KREUTZ et al. (2015); FEAMSTER et al. (2014); LEVIN et al. (2012). Esse ponto trata justamente da separação dos planos de controle (elemento controlador que decide como manipular o tráfego) e de dados (elementos que manipulam o tráfego de acordo com as decisões estabelecidas pelo plano de controle). Essa dissociação entre o hardware de encaminhamento e as decisões de controle pode ser considerada a ideia básica do paradigma SDN, pois ela influencia diretamente as demais características. Utilizando essa abordagem, o controlador toma para si a responsabilidade de toda e qualquer decisão sobre como manipular os dados, deixando a cargo de switches e roteadores apenas a tarefa de encaminhamento desses dados de acordo com a decisão estabelecida. Segundo ZILBERMAN et al. (2015), a separação dos planos não é um conceito novo, mas foi popularizado com o advento do paradigma.

Uma segunda característica chave do paradigma SDN é a questão da centralização do controle e da visão da rede. Segundo MACEDO et al. (2015), SDNs são caracterizadas pela existência de um sistema de controle programável - em software – e centralizado que é capaz de controlar os mecanismos do plano de dados através de uma interface de programação bem definida. Em redes SDN, a inteligência da rede é logicamente centralizada no controlador SDN, que mantém uma visão global da rede, simplificando a aplicação de políticas, a configuração e a evolução da rede (KREUTZ et al., 2015; ONF, 2016).

Outro ponto chave trata da questão de programabilidade da rede. Segundo NUNES et al. (2014), a separação dos planos de controle e dados simplifica drasticamente o gerenciamento da rede e permite a inovação e a evolução da mesma. Em contraste, as redes tradicionais são formadas por um grande conjunto de protocolos, que evoluem lentamente e são difíceis de gerenciar, devido à configuração de baixo nível dos componentes individuais, o que torna complicado qualquer tipo de modificação na rede (FEAMSTER et al., 2014). Através de uma programação de mais alto nível proporcionada pelo paradigma SDN, novos serviços podem ser implantados mais rapidamente, diminuindo os custos operacionais das redes.

Uma última característica fundamental de SDN é a utilização de interfaces abertas para a programação dos dispositivos da rede. Com a separação dos planos de dados e de controle, os comutadores se tornam simples equipamentos de encaminhamento de pacotes, que podem ser programados através de uma interface aberta de programação bem definida (NUNES et al., 2014; GUEDES et al., 2012). Segundo ONF (2016), quando

implementado em padrão aberto, o paradigma SDN simplifica o projeto e a operação da rede, uma vez que as instruções são fornecidas unicamente pelo controlador, e não por múltiplos protocolos e dispositivos proprietários. Um exemplo disso é o padrão OpenFlow, que será discutido mais adiante neste trabalho.

Em resumo, considerando todos os aspectos discutidos anteriormente, uma possível definição para SDN poderia ser a seguinte: SDN é um paradigma de redes de computadores, cuja implementação consiste na inserção de um elemento centralizado na rede, denominado controlador. O controlador é o elemento responsável por decidir como os dados (pacotes) serão manipulados pelos elementos comutadores, que por sua vez, irão apenas se concentrar na tarefa de manipular os dados de acordo com as decisões tomadas pelo controlador. Dessa forma, passa a existir uma clara separação entre o plano de controle e o plano de dados, o que simplifica o gerenciamento e a configuração da rede e, principalmente, permite a flexibilização da rede, a evolução da rede e o desenvolvimento de novos serviços, tudo isso através de uma programação de alto nível dos dispositivos comutadores por meio de uma interface aberta bem definida entre esses dispositivos e o elemento de controle programável.

2.2 BREVE HISTÓRICO

A Figura 1 ilustra uma resumida linha do tempo da evolução do conceito de SDN, por meio do desenvolvimento de tecnologias e projetos precursores que serviram de base para o desenvolvimento do atual paradigma SDN. Essas tecnologias serão detalhadas ao longo desta seção.

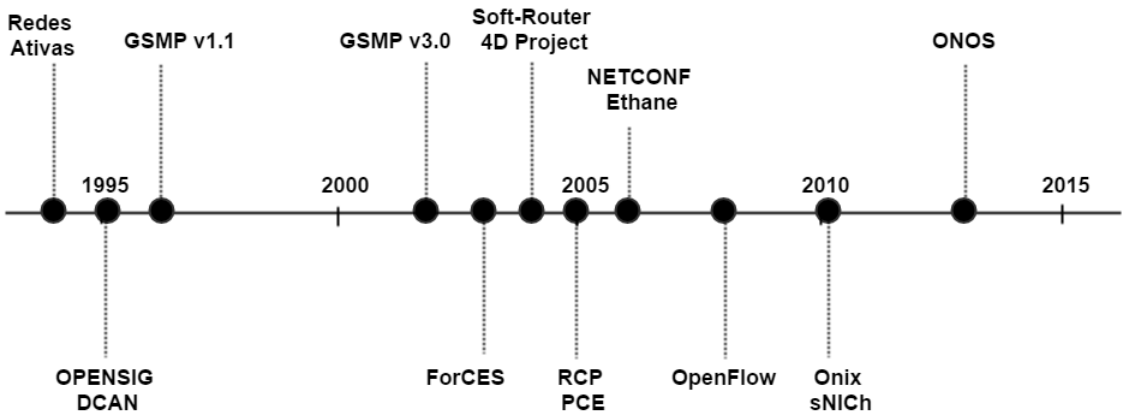

Figura 1 – Linha cronológica de tecnologias que serviram como base para o desenvolvimento do paradigma SDN.

O termo original Software-Defined Networks (SDN) foi utilizado pela primeira vez no ano de 2009, no artigo tecnológico de GREENE (2009) sobre o projeto OpenFlow da Universidade de Stanford, que fora lançado dois anos antes. Apesar de a nomenclatura SDN ser relativamente recente e o conceito por trás dela ter evoluído ao longo do tempo, a base ideológica para a formulação do paradigma remonta de meados da década de 1990.

Até então, as redes tradicionais operavam com os planos de controle e de dados combinados dentro de cada nó da rede. Essas redes transportavam dados passivamente, de um sistema final para outro, e essa transferência era a mais simples possível, sem qualquer tipo de modificação da unidade de informação. O papel da computação dentro das redes era limitado a tarefas pontuais como, por exemplo, o processamento dos cabeçalhos dos pacotes, que antecedia ao encaminhamento dos mesmos (TENNENHOUSE et al., 1997). Além disso, nessa abordagem, apenas a política de encaminhamento era definida e o único jeito de fazer ajustes nessa política era através de mudanças na configuração dos dispositivos (SEZER et al., 2013). Tudo isso acabou restringindo a capacidade de configuração e a extensibilidade das redes tradicionais, tornando-se um gargalo importante ao longo dos anos, com o aumento da complexidade das redes. As redes passaram a ficar ossificadas, sem espaço para inovação, o que culminou em uma certa estagnação tanto na criação de novos serviços e protocolos quanto em sua arquitetura.

Para tentar solucionar esse problema, muitas tentativas de adicionar mais recursos de programação às redes de computadores foram sendo desenvolvidas ao longo dos anos. O objetivo dessas tentativas era tornar as redes mais flexíveis, permitindo a inovação, a experimentação e o desenvolvimento de novos serviços e protocolos.

Uma das primeiras iniciativas propostas, nesse sentido, foi a de Redes Ativas (Active Networks - AN). Formulada entre 1994 e 1996, e posteriormente publicada no trabalho de TENNENHOUSE et al. (1997), a iniciativa de Redes Ativas representa uma forma preliminar de programação da rede. Segundo MACEDO et al. (2015), a criação das primeiras funções de virtualização de computadores, como máquinas virtuais e *hypervisors*, influenciaram seu desenvolvimento. A principal ideia dessa proposta consiste no fato de que roteadores ou switches em uma Rede Ativa passam a ter a capacidade de realizar cálculos personalizados sobre os pacotes, ou ainda, modificar o conteúdo dos mesmos (TENNENHOUSE et al., 1997; KREUTZ et al., 2015). Para realizar essas tarefas, Redes Ativas propõem duas abordagens distintas: switches programáveis e cápsulas. Na abordagem de switches programáveis, o formato dos pacotes (ou células) não é modificado. Além disso, os dispositivos comutadores devem ser capazes de suportar o *download* de programas com instruções específicas sobre como processar os pacotes. Já na abordagem de cápsulas, por outro lado, os pacotes são substituídos por pequenos programas que são encapsulados em quadros de transmissão e executados em cada nó ao longo do caminho percorrido (TENNENHOUSE et al., 1997; KREUTZ et al., 2015). Redes Ativas trouxeram

grandes contribuições que se relacionam intimamente com a ideia de SDN que se tem hoje. Elas foram pioneiras na utilização de funções programáveis de rede que permitiram a inovação e a criação de novos serviços em redes em produção. Também foram as primeiras que trataram da questão de virtualização da rede, além de realizarem a demultiplexação personalizada de programas em software de acordo com campos específicos do cabeçalho dos pacotes (FEAMSTER et al., 2014). Infelizmente, a iniciativa de Redes Ativas nunca conseguiu reunir uma grande massa crítica e acabou não sendo implantada em redes de produção de larga escala devido, principalmente, a questões de segurança e desempenho dos roteadores AN, que não eram capazes de processar a mesma quantidade de pacotes quando comparados a roteadores tradicionais (MACEDO et al., 2015; NUNES et al., 2014).

Outra iniciativa proposta foi a do grupo de trabalho OpenSignaling (OPENSIG), que iniciou, em 1995, uma série de discussões e *workshops* com o objetivo de tornar redes como ATM, Internet e redes móveis mais abertas, extensíveis e programáveis. Esse grupo defendia a ideia de que a separação entre o hardware de comunicação e o software de controle era necessária, mas difícil de realizar, devido, principalmente, a switches e roteadores verticalmente integrados, cuja natureza fechada tornou praticamente impossível o rápido desenvolvimento de novos serviços e ambientes de rede (NUNES et al., 2014). A proposta dessa iniciativa estava centrada na implantação do acesso ao hardware de rede através de interfaces de rede abertas e programáveis, o que coincide com uma das características do atual padrão SDN. Dessa forma, novos serviços poderiam ser desenvolvidos através de programação em um ambiente distribuído (NUNES et al., 2014). Nessa proposta também estavam elencados quatro níveis de abstração: elemento físico, elemento de rede virtual, serviços genéricos e serviços de valor adicionado. O grupo de trabalho determinou e alocou recursos e serviços para cada um desses níveis, e criou, posteriormente, interfaces para o controle desses níveis (MACEDO et al., 2015). Como resultado final desse esforço, o grupo elaborou a especificação de um novo protocolo, o GSMP (General Switch Management Protocol), de uso geral para controle de um switch. Esse protocolo, de forma resumida, permitia que um controlador estabelecesse e liberasse conexões através do switch, gerenciando portas, informações de configuração, estatísticas e reserva de recursos do switch (NUNES et al., 2014; GSMP, 2002). A última versão do protocolo data do ano de 2002.

Com a criação do protocolo GSMP, houve uma primeira tentativa de se criar uma entidade rudimentar de controle de equipamentos de comutação, a qual estivesse separada fisicamente desses equipamentos. Uma outra tentativa de separação dos planos fora desenvolvida ainda em meados dos anos 1990, pela ATM. O denominado DCAN (Devolved Control of ATM Networks) foi um projeto que tinha o objetivo de desenvolver a infraestrutura para controle e gerenciamento escalável das redes ATM. A sua premissa de separação dos planos vai ao encontro da premissa do OPENSIG, inclusive na criação de um protocolo minimalista entre o gerenciador e a rede (DCAN, 1995; NUNES et al., 2014).

Contudo, até por volta dos anos 2000, os esforços se centravam na melhoria da programabilidade das redes iniciada pela proposta de Redes Ativas. Somente a partir de então, uma série de esforços que visavam a separação dos planos de dados e de controle começou a emergir. Foi o caso da arquitetura ForCES (Forwarding and Control Element Separation). Inicializada em 2003, ela definiu dois elementos funcionais, o elemento de encaminhamento (FE) e o elemento de controle (CE), o que passou a separar de fato os planos de controle e de dados. ForCES é uma arquitetura de muitos recursos, capaz de atender a topologias arbitrárias compostas por grandes quantidades de FEs e CEs, além de contar uma interface aberta programável entre os dois planos e um conjunto de módulos que implementam tarefas de controle específicas (MACEDO et al., 2015; ForCES, 2004).

Foram desenvolvidas outras arquiteturas nos anos seguintes, como a Routing Control Platform (RCP) e a Soft-Router, assim como o protocolo Path Computation Element (PCE), que trouxeram como contribuição principal o controle lógico centralizado da rede, através de uma interface aberta para o plano de dados (FEAMSTER et al., 2014). A arquitetura Soft-Router usa a API de programação da ForCES para permitir que o controlador instale entradas na tabela de fluxos do plano de dados, removendo completamente a funcionalidade de controle dos roteadores (LAKSHMAN et al., 2004). Já a RCP utiliza o protocolo padrão BGP para instalar entradas na tabela de fluxo nos roteadores normais (CAESAR et al., 2005).

Ainda em meados do ano 2000, houve uma série de esforços na construção e projeto de novos sistemas e arquiteturas que ampliassem os benefícios alcançados até então, em termos de programabilidade da rede e separação entre controle e encaminhamento, para novas áreas de aplicação. Neste contexto, surgiu o projeto Ethane. Proposto no trabalho de CASADO et al. (2009) e desenvolvido pelo departamento de Ciência da Computação da Universidade de Stanford, o Ethane apresentou um projeto de arquitetura de redes corporativas baseado em uma solução que envolvia um controlador logicamente centralizado, que atuaria como gerenciador de políticas de encaminhamento, de segurança e de acesso à rede. Resumidamente, redes Ethane são compostas por duas entidades: um controlador centralizado, que decide se um pacote deve ser encaminhado, e um switch Ethane, que consiste em uma tabela de fluxo e um canal seguro para comunicação com o controlador (NUNES et al., 2014). O Ethane reduziu os switches a simples tabelas de fluxo que são manipuladas pelo controlador com base em políticas de segurança de alto nível (FEAMSTER et al., 2014). Nesta arquitetura, o controlador tem o conhecimento da topologia global da rede e realiza a computação das rotas apenas para fluxos permitidos, através da escrita das ações devidas na tabela de fluxo dos switches. Ou seja, quando um pacote chega e não há registro de um fluxo que corresponda a esse pacote, ele é repassado ao controlador, que, por sua vez, decidirá o que fazer com o pacote (se vai descarta-lo ou se vai criar uma entrada na tabela de fluxos para que o pacote possa ser encaminhado) (CASADO et al., 2009).

Além disso, as redes Ethane adotaram um conceito que passou a ser amplamente utilizado a partir de então: o conceito de fluxo. Segundo ele, um fluxo é um conjunto de pacotes que possuem campos idênticos de cabeçalho (como endereços de origem iguais ou tipo de pacote, por exemplo). Ou seja, quando pacotes diferentes chegam ao switch, mas possuem determinados campos de cabeçalho idênticos, eles serão tratados da mesma forma, uma vez que serão considerados integrantes de um mesmo fluxo. Assim, apenas uma única política de decisão pode ser feita para cada fluxo, e não mais para cada pacote individual que chega à rede (CASADO et al., 2009). Além disso, houve a utilização dos conceitos desenvolvidos, até então, para a realização de um serviço específico que, no caso em questão, foi o de segurança da rede. De certa forma, pode se considerar o serviço de segurança em uma rede Ethane como sendo a primeira aplicação desenvolvida na história do SDN.

Para alguns autores, a história do paradigma SDN se inicia aqui, com a criação do projeto Ethane, uma vez que grande parte das ideias elaboradas nesse projeto e vários dos conceitos adotados serviram como base para o desenvolvimento do paradigma SDN. Além disso, o projeto serviu como base para a elaboração do projeto que tornou possível, de fato, a implementação do paradigma SDN: o protocolo OpenFlow.

O protocolo OpenFlow foi desenvolvido também na Universidade de Stanford, assim como seu antecessor, e publicado em 2008, no trabalho de McKEOWN et al. (2008). Na prática, ele adotou as premissas de SDN previamente discutidas e seguiu a estrutura adotada pelo projeto Ethane. A sua grande contribuição foi ter conseguido trazer toda a ideia de programabilidade de redes e de separação entre controle e dados para um desenvolvimento real e prático do paradigma SDN. Por ser uma solução simples, que não requer modificações significativas nos dispositivos comutadores, tornou-se atrativa não somente para a comunidade acadêmica, mas também para a indústria (KREUTZ et al., 2015). A arquitetura e outras características do protocolo OpenFlow serão mais bem detalhadas em uma seção específica adiante neste trabalho.

Concluindo, o paradigma SDN foi evoluindo, gradativamente, capturando ideias de cada iniciativa proposta, principalmente as ideias que tratavam de aspectos como controle centralizado e separação dos planos, que revolucionaram a área. Foi essa evolução que transformou SDN na iniciativa de sucesso que ela é hoje. Daí a importância de se analisar o que as tecnologias anteriores trouxeram como colaboração para o estado-da-arte que existe atualmente.

2.3 COMPONENTES DE UMA ARQUITETURA SDN SIMPLES

Considerando o que já foi discutido nas seções anteriores, pode-se dizer, de forma simplista, que uma Rede Definida por Software é caracterizada pela existência de duas entidades principais: um elemento controlador, que tem o poder de determinar o modo

como os dados devem ser manipulados em uma rede; e os elementos de encaminhamento, que são os equipamentos de comutação que devem manipular os dados de acordo com o que foi definido pelo elemento controlador. Contudo, uma SDN não pode ser caracterizada apenas dessa maneira. Existem outros elementos, físicos e abstratos, que desempenham importantes papéis em uma estrutura típica do paradigma. A Figura 2 ilustra uma arquitetura SDN típica, composta por seus principais elementos. Nas subseções a seguir, esses elementos serão apresentados com detalhes.

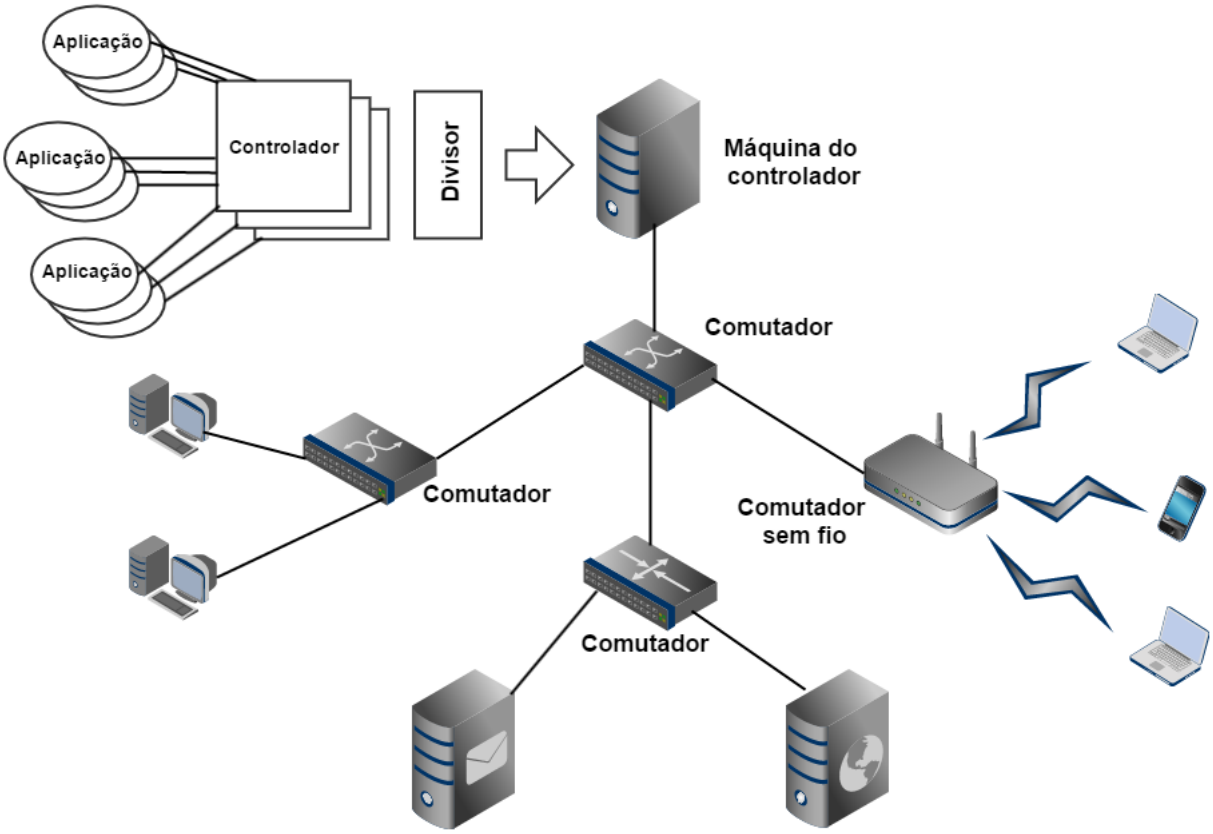

Figura 2 – Arquitetura SDN típica.

2.3.1 **Dispositivos de usuário final**

São os equipamentos localizados na ponta das redes, tais como computadores, servidores, notebooks, celulares e demais dispositivos que acessam e utilizam os serviços providos pela rede. Eles são elementos não programáveis, contudo, sofrem influência indireta das decisões tomadas pelo controlador. Podem ser equipamentos ligados à rede por meio de cabos ou por antenas.

2.3.2 **Equipamentos de comutação programáveis**

São os equipamentos de comutação similares aos das redes tradicionais, como switches e roteadores - virtuais ou físicos – que, ao serem inseridos em uma arquitetura SDN, se tornam simples dispositivos de encaminhamento de pacotes, uma vez que toda e qualquer tarefa de decisão ou controle foi retirada de seu escopo e atribuída ao controlador, devido à separação física dos planos de dados e de controle. Portanto, resta a esses dispositivos apenas a tarefa do plano de dados, que é a de encaminhar pacotes.

2.3.3 **Controlador**

O controlador é o dispositivo da rede que monitora e modifica o comportamento dos elementos programáveis da rede, como switches e roteadores, além dos elementos finais de acesso à rede. É também o componente principal do plano de controle, e pode estar associado a outros controladores. Em uma SDN, o controlador utiliza um conjunto de APIs para interagir com os elementos programáveis, que são geralmente os dispositivos comutadores. O controlador envia seus comandos de controle para esses elementos programáveis através do canal de controle seguro, utilizando uma API de baixo nível, que geralmente é dependente de hardware e tende a ter um grau limitado de abstração (MACEDO et al., 2015). Para facilitar a programação da rede, o controlador age como se fosse um sistema operacional para ela, exportando uma interface de programação de alto nível para operadores e desenvolvedores que abstraia os detalhes de operação de cada componente e, além disso, que permita a execução simultânea de diferentes programas de controle. Além de ser denominado como sistema operacional de redes, o controlador também é tratado por alguns autores como *hypervisor* da rede, por muitas vezes permitir o particionamento da rede em redes virtuais.

Além da característica principal, que é estabelecer a comunicação com todos os elementos programáveis de uma SDN, o controlador também fornece uma visão unificada do status da rede. Essa visão unificada (e centralizada) da rede, um dos pontos fortes do paradigma SDN, permite o desenvolvimento de análises detalhadas para determinar a melhor decisão operacional sobre como o sistema, como um todo, deve atuar. Essa visão global da rede acaba simplificando a tarefa de gerenciamento, a descoberta e resolução de problemas e a tomada de decisão. Além disso, é importante salientar que essa visão única e centralizada da rede fornecida pelo controlador é uma visão lógica, não exigindo, necessariamente que o controlador seja um componente concentrador da rede, fisicamente localizado em um ponto específico e único do sistema. Essa visão pode ser abstraída através da implementação de soluções de forma distribuída, quer seja pela utilização de múltiplos controladores resolvendo problemas específicos em pontos distintos da rede ou pela divisão dos elementos da visão entre domínios diferentes em um mesmo ponto da rede. Ademais, um único controlador centralizado representa um único ponto de falha

em toda a rede, contudo protocolos como o OpenFlow permitem a conexão de múltiplos controladores a um switch, o que garantiria a redundância no plano de controle, no caso em que um evento de falha no controlador principal venha a acontecer.

Em geral, o controlador é responsável pela execução de códigos de programas de controle que podem ser desenvolvidos utilizando-se linguagens de alto nível, que posteriormente serão traduzidos pelo próprio controlador em ações que podem ser enviadas a cada elemento da rede. Basicamente, aqui reside a diferença funcional entre o código do programa, que determina o que será feito, e o controlador, que transforma o código de alto nível de uma linguagem de programação em comandos entendidos pela API, que serão enviados aos comutadores. Daí o conceito de sistema operacional da rede atribuído ao controlador.

Geralmente, o controlador é composto também por duas interfaces, *Southbound* e *Northbound*, cada qual responsável por estabelecer a relação do controlador com algum elemento da arquitetura. As subseções seguintes abordarão esses dois elementos. A relação entre o controlador e essas interfaces está ilustrada na Figura 3.

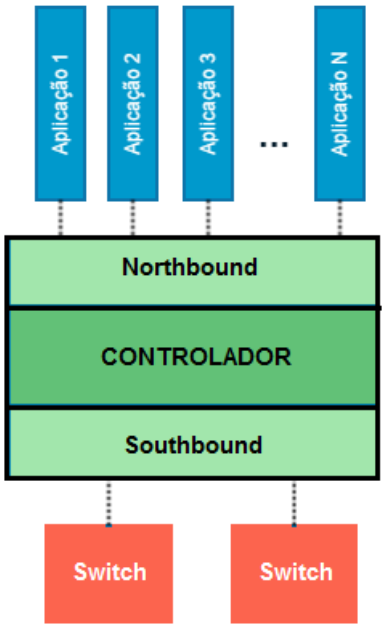

Figura 3 – Relação entre o controlador e as interfaces *Northbound* e *Southbound*.

2.3.4 API Southbound

As interfaces *Southbound*, localizadas no nível mais baixo da plataforma de controle, são as responsáveis por controlar a interação entre o controlador e os dispositivos de encaminhamento. Podem ser vistas também como uma camada de drivers de dispositivos. A expressão “sul” refere-se à direção controlador–switch, em cuja relação há o estabelecimento

de comunicação do controlador para um nível mais baixo da arquitetura, que são os equipamentos físicos da rede. O protocolo OpenFlow pode ser visto como exemplo de uma API *Southbound*, uma vez que ele implementa as interações e a comunicação entre o controlador da rede e os dispositivos de comutação (Figura 3). A maioria dos controladores suporta apenas o OpenFlow como API *Southbound*, mas alguns como o OpenDaylight e o Onix suportam uma gama maior de APIs ou *plugins* de protocolo. Alguns outros exemplos desse tipo de interface são a ForCES, o NETCONF e o protocolo SMNP.

2.3.5 **API Northbound**

As interfaces *Northbound*, localizadas no nível mais alto da plataforma de controle, são as responsáveis por controlar a interação entre o controlador e as aplicações de controle. A API *Northbound* pode ser vista como uma interface de programação que permite que os programas desenvolvidos em linguagens de alto nível sejam capazes de controlar a rede (Figura 3). Comparadas às APIs *Southbound*, as interfaces *Northbound* possuem um nível maior de abstração. A expressão "norte" refere-se à direção controlador-aplicação, em cuja relação há a comunicação entre os serviços e aplicações que devem ser executados (no nível mais alto da arquitetura), e o controlador, que deve ser capaz de "entender" o código para posteriormente transformá-lo em comandos para a rede. Ao contrário da API *Southbound*, que já possui um padrão amplamente aceito (OpenFlow), não existe uma definição de uma API *Northbound* padrão para SDN. A maioria dos controladores (Floodlight, Trema, NOX, etc...) define suas próprias API *Northbound*. No entanto, o projeto OpenDaylight vem tentando se tornar uma padronização para a interface *Northbound*, a fim de aumentar a reutilização de programas de controle e fomentar inovações relacionadas a esse domínio (MACEDO et al., 2015). Um outro exemplo desse tipo de interface é a SFNet, que traduz requisitos da aplicação para requisições de serviço de nível mais baixo.

2.3.6 **Aplicação**

Como os controladores são considerados os sistemas operacionais das redes SDN, os códigos em software que são criados para criar novos serviços e funcionalidades, e que são executados pelo controlador, podem ser considerados como sendo as aplicações que são executadas sobre a rede física. Em outras palavras, as aplicações de rede implementam o controle lógico que será traduzido em comandos que instalarão as regras no plano de dados, ditando o comportamento dos dispositivos comutadores. Existem muitas aplicações de redes tradicionais que podem ser implementadas com SDN, tais como balanceamento de carga, roteamento ou *firewall*. Entretanto, novos serviços podem ser implementados agora em mais alto nível, como o monitoramento das redes ou a economia de energia. Em uma seção adiante, serão abordadas algumas das aplicações possíveis atualmente em SDN.

2.3.7 **Virtualizador de rede (ou divisor)**

Uma vez que, com a criação de uma SDN, se tornou possível identificar padrões diferentes de pacotes que passam pelo comutador, que são posteriormente classificados e segmentados em fluxos, tornou-se viável também o tratamento diferenciado para fluxos diferentes. Um exemplo clássico disso é a separação feita entre os fluxos tradicionais de produção e os fluxos de pesquisa, que pode ser realizada agora, com SDN, sem trazer nenhum tipo de prejuízo para a rede ou para o serviço relacionado a ambos os fluxos. Essa divisão "virtual" no tratamento de diferentes padrões de fluxo é realizado pelo virtualizador de rede. Em outras palavras, o virtualizador de rede - ou divisor - é o componente de uma SDN responsável pelo gerenciamento de redes virtuais, que além de prover essa separação entre visões é capaz de prover uma divisão dos recursos físicos entre essas visões. Cada rede virtual formada pelo divisor pertence a uma visão diferente e possui seu próprio software de plano de controle. Dessa forma, podem coexistir vários controladores rodando sobre uma mesma rede física, mas que estão virtualmente separados. Os recursos da rede podem ser virtualizados e apresentados de forma isolada para cada visão, que por sua vez só terá acesso aos elementos e recursos da rede reservados para a sua fatia da rede física.

2.4 O PROTOCOLO OPENFLOW

A premissa mais importante por trás do paradigma de Redes Definidas por Software é a capacidade de controlar, à distância, a tarefa de encaminhamento de pacotes, através de uma interface de programação bem definida. Foram desenvolvidos, ao longo dos anos, alguns protocolos capazes de implementar essa comunicação entre dispositivos inseridos em uma SDN. Contudo, um desses protocolos está associado ao paradigma desde a sua concepção, e, de fato, foi o que tornou possível a implementação prática da SDN que é conhecida atualmente: o protocolo OpenFlow.

O protocolo OpenFlow, desenvolvido na Universidade de Stanford, e proposto no trabalho de McKEOWN et al. (2008) em 2008, é um padrão aberto para Redes Definidas por Software que tem o objetivo de permitir que aplicações em software possam programar simplificadamente as tabelas de fluxo dos dispositivos de comutação de pacotes presentes em uma rede, através de uma interface aberta de programação. Essa programação implica no controle desses dispositivos por uma entidade centralizada, o controlador, o que acaba ocasionando, na prática, a separação dos planos de dados e de controle da rede. Nesse ponto, vale ressaltar a diferença entre o paradigma SDN e o protocolo OpenFlow, que muitas vezes são considerados sinônimos. Enquanto SDN é um paradigma que consiste na ideia de separação dos planos, o OpenFlow descreve o modo como o software controlador e os switches devem se comunicar em uma arquitetura SDN.

Com a proposição do OpenFlow, houve a padronização da comunicação entre os dispositivos de encaminhamento e o controlador. Dessa forma, o controlador, que é

composto por um código em software de alto nível, desenvolvido em uma linguagem de programação determinada, pode enviar uma série de operações em formato de comandos OpenFlow ao comutador, que serão entendidas pelo comutador graças a essa padronização na comunicação. Esses comandos OpenFlow são capazes de programar a tabela de fluxos dos switches quando o protocolo está habilitado.

Para todos os efeitos, neste trabalho, a versão de referência do protocolo OpenFlow utilizada para ilustrar exemplos e situações é a versão 1.0.0. Esta versão foi escolhida porque é a versão que será avaliada nos testes dos próximos capítulos, e porque ainda é a versão do protocolo mais utilizada e suportada (LARA et al., 2014).

2.4.1 **Arquitetura do protocolo OpenFlow**

Quando se habilita o protocolo OpenFlow em um switch (que suporte o protocolo), a arquitetura do dispositivo passa a ser constituída por três elementos, como ilustrado na Figura 4: (i) uma **tabela de fluxos**, na qual existe, para cada entrada da tabela, uma ação associada que define como o switch vai processar esse fluxo relacionado; (ii) um **canal seguro**, que liga o switch ao controlador, e por onde passam os comandos e os pacotes trocados entre eles; e (iii) o **protocolo OpenFlow**, que fornece uma interface padrão aberta de comunicação entre controlador e switch. Devido à especificação dessa interface padrão (o protocolo OpenFlow), as entradas da tabela de fluxos podem ser definidas (programadas) externamente pelo controlador, que envia seus comandos através do canal seguro.

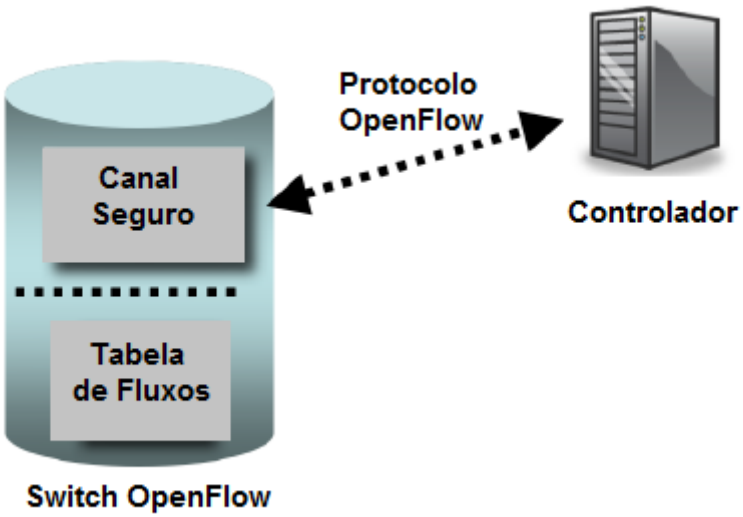

Figura 4 – Um switch OpenFlow se comunica com um controlador através de um canal seguro, usando o protocolo OpenFlow.

A tabela de fluxos é composta por um conjunto de entradas, e cada uma dessas

entradas da tabela possui três campos: (i) um campo composto por **múltiplos campos de cabeçalho**, que definem o fluxo associado a essa entrada; (ii) um campo de **ações**, que definem como os pacotes desse fluxo serão processados; e (iii) um campo com **contadores**, que mantém os registros de números de pacotes e bytes de cada fluxo, bem como tempo decorrido desde a chegada do último pacote associado ao fluxo (Figura 5). Em resumo, a união entre uma determinação de um fluxo e um conjunto de ações associado ao fluxo forma uma entrada na tabela de fluxos (McKEOWN et al., 2008). Além disso, cada entrada na tabela de fluxos de um switch OpenFlow pode ser armazenada em memória local. Tipicamente, é utilizada memória SRAM (Static Random Access Memory) ou uma memória TCAM (Ternary Content-Addressable Memory), memória na qual os bits podem ser representados por valores iguais a "zero", "um" ou "não importa", esse último indicando que ambos os valores anteriores são aceitáveis naquela posição (LARA et al., 2014; GUEDES et al., 2012; XIA et al., 2015).

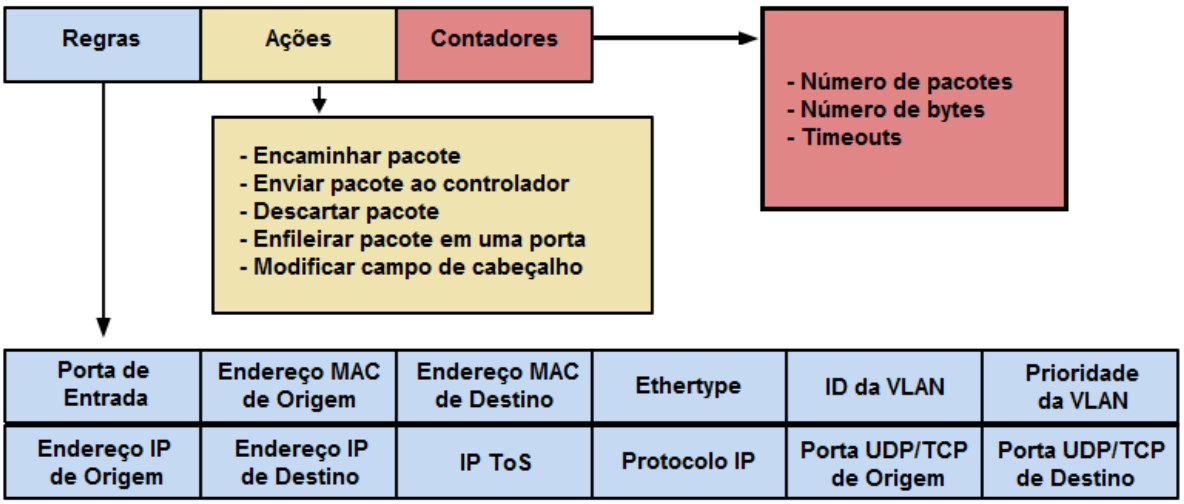

Figura 5 – Entrada da tabela de fluxos: atributos que determinam o fluxo (ou regra) associado, ações e contadores.

Os campos de cabeçalho de uma entrada da tabela, que são semelhantes aos campos de cabeçalho típicos de um pacote (como endereços MAC e IP de origem e destino, tipo de protocolo, dentre outros), são associados a valores pré-definidos. Esses valores serão comparados aos valores dos campos de cabeçalho de cada pacote que chega ao comutador, com o objetivo de determinar se o pacote pertence ao fluxo definido por essa entrada da tabela (também conhecida como regra). Se os valores dos campos do pacote forem iguais aos valores dos respectivos campos definidos na entrada da tabela, há um *match* (ou casamento), o que significa que o pacote pertence àquele fluxo definido pela entrada da tabela. Se algum dos valores dos campos do pacote não coincidir com o valor do respectivo campo da entrada da tabela, não haverá um *match* para esse fluxo (ou seja, o pacote não

pertence a esse fluxo). Assim sendo, o pacote continuará a ser comparado com as demais entradas da tabela, até sofrer um *match* ou, no caso de insucesso em todas as comparações, será enviado ao controlador, que decidirá o que fazer com ele. Exemplificando, se uma entrada da tabela define o campo de endereço MAC de origem como sendo o valor A, e chega um pacote ao switch com o campo de endereço MAC de origem com o valor A, haverá um *match*, e o pacote será atribuído a esse fluxo (e sofrerá as ações que couberem a esse fluxo). No entanto, se chegar um pacote com endereço MAC de origem com valor B, esse pacote não sofrerá *match*, e, consequentemente, não será atribuído a esse fluxo. A parte inferior (em azul) da Figura 5 mostra quais são os atributos utilizados nas entradas da tabela de fluxo para a realização de *matches*, implementados pela versão 1.0.0 do protocolo OpenFlow.

Já as ações determinam como o switch deve proceder com os pacotes correspondentes a um determinado fluxo. Cada fluxo pode ser associado a nenhuma, uma ou várias ações. Contudo, muitas das implementações OpenFlow existentes hoje são capazes de realizar uma única ação. As possíveis ações que podem ser tomadas por um switch OpenFlow, na versão 1.0 do protocolo, em relação aos pacotes são: (i) o **encaminhamento**, que consiste no repasse do pacote para uma porta específica do switch, para todas as portas, de volta para a porta de entrada, ou para o controlador (caso em que o pacote é encapsulado e enviado pelo canal seguro); (ii) o **enfileiramento**, que consiste em enviar um pacote para uma fila associada a uma porta específica, geralmente para prover um serviço básico de QoS; (iii) o **descarte**, que consiste em descartar pacotes explicitamente ou, caso não exista ação especificada para um determinado fluxo, descartar todos os pacotes que casarem com esse fluxo; e (iv) a **modificação de campo de cabeçalho**, que consiste na mudança do valor de algum campo do cabeçalho do pacote que chegou. Contudo, para versões mais recentes, novas ações e operações foram sendo adicionadas. Essas modificações serão apresentadas na subseção a seguir.

2.4.2 Versões do protocolo OpenFlow

Atualmente, estão disponíveis diferentes versões do protocolo OpenFlow, sendo que a mais recente delas, a versão 1.5, data do final de 2014. Cada uma delas, gradativamente, vem inserindo modificações, melhorias ou até novas funcionalidades no protocolo. Contudo, a versão mais amplamente utilizada e suportada, no momento, é a versão 1.0 (LARA et al., 2014).

As primeiras versões do protocolo OpenFlow (0.2, 0.8 e 0.9) datam de 2008 e 2009. Entretanto, essas eram versões não-estáveis, que, inclusive, já foram descontinuadas. Somente em dezembro de 2009 foi lançada a primeira versão estável do protocolo, que é justamente a versão 1.0. Nesta versão, o controlador dispõe de apenas uma única tabela de fluxos em cada switch, e as operações de *match* podem ser realizadas sobre os 12

campos de cabeçalho apresentados anteriormente na parte inferior da Figura 5: porta física de entrada, endereços MAC de origem e destino, ethertype, ID da VLAN, prioridade da VLAN, endereços IP de origem e destino, protocolo IP, IP *Type of Service* e portas TCP/UDP de origem e destino. O número de ações também era resumido a apenas quatro: encaminhamento, descarte, enfileiramento e modificação de campos de cabeçalho.

Por sua vez, a versão 1.1, lançada em fevereiro de 2011, passou a suportar operações em múltiplas tabelas de fluxos. Outras grandes novidades dessa versão foram o suporte a *match* em campos MPLS (Multiprotocol Label Switching) e a tabela de grupos, que permitia que operações em comum de múltiplos fluxos fossem agrupadas e realizadas. Foi aumentado o número de ações (alteração de TTL, agrupamento, cópia de campos, operações de QoS e *push/pop* de *tags* de cabeçalho), o que aumentou o poder e a flexibilidade do protocolo no reconhecimento de fluxos e na realização de operações.

Em dezembro de 2011 foi lançada a versão 1.2 do protocolo OpenFlow. A principal funcionalidade trazida por essa versão foi a possibilidade de conectar um switch a múltiplos controladores simultaneamente, o que tornou possível tarefas como recuperação de falhas e balanceamento de carga entre controladores. Outras características adicionadas por essa versão foram o suporte aos protocolos IPv6 e ICMPv6.

A versão 1.3 do protocolo OpenFlow, que foi lançada em junho de 2012, trouxe melhorias como o controle da taxa de pacotes através de *meters* por fluxo, a habilitação de conexões auxiliares entre o switch e o controlador, o suporte aos cabeçalhos de extensão do IPv6 e o suporte para comunicação encriptada por TLS. Esta foi uma versão amplamente aceita pela comunidade, tanto pela sua facilidade de uso quanto pela facilidade de prototipação de novas funcionalidades. Até por isso, a ONF havia escolhido essa versão para a validação de novos recursos (FERNANDES and ROTHENBERG, 2014).

Em outubro de 2013 foi lançada a versão 1.4 do protocolo OpenFlow, que trouxe uma grande quantidade de melhorias e novidades. As principais foram: o suporte a execução de ações conjuntas, denominadas *bundles*, por meio de uma única operação; o monitoramento de fluxos em tempo real por parte do controlador para detecção de alterações nos switches; a exclusão de regras de menor importância para a inserção de novas regras quando os switches operam com a tabela de fluxos totalmente ocupada; a comunicação ao controlador de que a tabela de fluxos está próxima do seu limite (*vacancy events*); a mudança da porta TCP padrão usada pelo protocolo (da 6633 para a 6653); dentre outras mudanças.

Por fim, a última versão do protocolo OpenFlow lançada até então, a 1.5, que data de dezembro de 2014, traz ainda mais melhorias ao protocolo. As principais são: o suporte a *egress table*, que habilita o processamento no contexto da porta de saída; a extensão das estatísticas das entradas de fluxo; o *match* por *flags* TCP como ACK, SYN e FIN; o monitoramento das conexões com os controladores; o agendamento de *bundles*;

o processamento de outros tipos de pacotes, como IP e PPP, e não apenas de pacotes Ethernet e MPLS, como era até então; dentre outras melhorias.

2.4.3 **Suporte ao protocolo OpenFlow**

Uma série de empresas de tecnologia de rede vem, ao longo dos anos, implementando e habilitando o padrão OpenFlow em seus equipamentos de comutação como switches, roteadores, pontos de acesso, *chips* e placas (*cards*). Algumas empresas como Arista, Brocade, Centec, Ciena, Cyan, Cisco, Dell, Extreme, EZchip, HP, Huawei, IBM, Juniper, LinkSys, NEC, NoviFlow, Pica8, Plexxi, Pronto, Quanta, Toroki, dentre outras, já disponibilizaram modelos de seus equipamentos com suporte a OpenFlow (KREUTZ et al., 2015; MACEDO et al., 2015; LARA et al., 2014; NUNES et al., 2014). Além dessas implementações em hardware, também existem as implementações em software. Algumas delas, como o Open vSwitch, foram desenvolvidas para serem executadas em um computador ou processador embarcado; outras foram desenvolvidas para placas como a NetFPGA; e outras ainda desenvolvidas para pontos de acesso, como o sistema Pantou/OpenWRT (NUNES et al., 2014; KREUTZ et al., 2015).

2.5 APLICAÇÕES PRÁTICAS EM SDN

Aspectos do paradigma SDN como a programabilidade de elementos comutadores e a visão logicamente centralizada da rede contribuem de forma fundamental para o desenvolvimento de novos serviços, assim como a evolução daqueles já existentes. Essa flexibilidade trazida por SDN para a estruturação das redes tornou mais fácil e prática a criação de serviços típicos de redes comuns. Nesta seção, serão apresentados diferentes tipos de aplicações de interesse que já podem ser desenvolvidos em SDN. Vale ressaltar que a lista não é exaustiva e que a quantidade de serviços que podem ser desenvolvidos em SDN tende a aumentar dada a evolução constante do paradigma e do suporte oferecido a ele.

Gerenciamento de redes corporativas. Atualmente, grande parte das políticas de gerenciamento das redes é configurada individualmente em cada elemento da rede. À medida que as redes vão crescendo e que são inseridos mais e mais dispositivos nessas redes, torna-se mais complicado estabelecer configurações de interesse na rede como um todo, uma vez que a complexidade para manter sincronizadas as configurações individuais dos equipamentos aumenta proporcionalmente ao crescimento da rede. Dessa forma, as políticas definidas raramente são modificadas. Com a adoção do paradigma SDN, a premissa de centralizar logicamente o plano de controle permite que uma visão global da rede seja gerada, o que simplifica as ações de configuração e a monitoração de fluxos de interesse (JARSCHEL et al., 2014). Mais ainda, com a variedade de informações obtidas através da rede, as políticas de gerenciamento podem ser adaptadas dinamicamente e

de forma automática de acordo com o estado atual da rede, além de serem repassadas para cada dispositivo, através de regras específicas que podem ser transcritas nas tabelas de fluxo de cada comutador, através de comandos únicos compreendidos pelos diferentes equipamentos.

Gerenciamento de energia. Atualmente, as técnicas de *green networking* vêm se tornando importantes no projeto e no desenvolvimento das redes, principalmente no que se relaciona à otimização dos recursos e, sobretudo, à economia de energia nas redes das empresas. Embora dispositivos comutadores em SDN não sejam capazes de diminuir seu próprio consumo de energia diretamente em operações na rede, a própria arquitetura SDN pode ser adaptada para prover um mecanismo que determine um uso mínimo de enlaces e switches para operações eficientes em energia. Ações como a redução da transmissão em enlaces subutilizados ou o desligamento por completo de dispositivos da rede podem ser realizadas uma vez que, com SDN, pode-se ter uma visão global da rede e, consequentemente, identificar quais elementos ociosos da rede podem ser desligados. Além disso, pode-se programar diretamente o controlador para realizar serviços de acordo com o estado atual da rede, como, por exemplo, o redirecionamento dinâmico de tráfego, visando a economia de recursos computacionais e, por conseguinte, a economia de energia.

Virtualização. A virtualização da rede é uma técnica difundida e já existente há algum tempo que permite que arquiteturas de rede heterogêneas compartilhem a mesma infraestrutura. Ela consiste em dividir a rede física em múltiplas instâncias virtuais que podem ser alocadas a diferentes usuários ou aplicações. Embora SDN e virtualização sejam entidades independentes e existam separadamente, a relação entre as duas abriu um novo campo para pesquisas na área. Um exemplo de benefício dessa relação é a criação de tecnologias por parte de SDN para a virtualização da rede, como, por exemplo, na criação de plataformas abstratas e equipamentos em software, como o Open vSwitch. Outro benefício trata do teste de aplicações SDN. A capacidade de separar os planos de dados e de controle torna possível o teste e o desenvolvimento de aplicações de controle SDN em ambientes virtuais, antes que as mesmas sejam implementadas em uma rede em operação. O desenvolvimento do ambiente Mininet, que virtualiza uma rede SDN com switches, *hosts* e controladores, dentro de uma máquina virtual, é um exemplo disso. Um último exemplo trata-se, de fato, da divisão de recursos em uma SDN. Enquanto a virtualização convencional utiliza túneis e campos VLAN e MPLS para realizar a configuração nada prática de um por um dos dispositivos da rede para a divisão dos recursos, SDN oferece uma plataforma que permite a configuração simplificada de cada um dos dispositivos da rede a partir de um único controlador (XIA et al., 2015). O FlowVisor é um exemplo de software desenvolvido para realizar essa divisão de recursos e administrar a múltipla divisão entre as visões dentro de uma mesma rede.

Segurança de redes. Tradicionalmente, a segurança em redes se dá através da

implantação de equipamentos físicos como *firewalls* e servidores *proxy*. Com a utilização de SDN, a segurança nessas redes pode ser aumentada, uma vez que o paradigma pode fornecer mais uma camada de segurança (antes de permitir que o tráfego adentre a rede), além de fornecer um conjunto de serviços em software capaz de melhorar a capacidade de defesa da rede. Mais detalhadamente, esses serviços podem prover novas formas de detecção e defesa de ataques em redes. Através da habilidade de coletar dados e estatísticas da rede, o paradigma SDN permite o monitoramento de todos os elementos da rede, a fim de detectar e analisar padrões de tráfego que possam estar associados a possíveis ameaças de segurança, em qualquer ponto da rede. Dessa forma, ataques simples ou de maior escala, como os de DDoS, podem ser detectados pela simples análise de padrões de tráfego, e podem ser tratados da forma mais adequada possível (XIA et al., 2015). Se ataques forem detectados, o próprio controlador pode instalar, dinamicamente, regras de encaminhamento que possam bloquear o ataque, através do descarte de pacotes. Se houver apenas suspeitas, o tráfego pode ser identificado e encaminhado para algum sistema de inspeção de pacotes. Outras técnicas de proteção como modificação aleatória de *hosts* internos, identificação de sobrecarga de tráfego, identificação de tráfego anômalo e segurança da própria infraestrutura também podem ser realizadas e facilmente programadas em uma SDN.

Controle de acesso. Intimamente relacionado à questão de segurança, o controle de acesso foi um dos projetos precursores de SDN, remetendo à criação das redes Ethane, que consistiam em um sistema de controle de acesso distribuído baseado na visão global da rede e em políticas de controle. Com SDN, pode-se estabelecer uma relação entre a identificação de fluxos (através de dados como endereços IP e MAC de origem e destino, tipo de protocolo, porta TCP/UDP, etc.) e a identidade de usuários que desejam acessar a rede, permitindo o reconhecimento de usuários cadastrados e de novas tentativas de acesso à rede. Além disso, a criação de políticas de acesso através do controlador pode também definir a rota que cada fluxo deve tomar, facilitando a implantação de filtros especiais de tráfego na rede, como *proxies*, servidores especializados, dispositivos de inspeção de pacotes e outros dispositivos auxiliares. Um exemplo disso seria a identificação de consultas DNS, que poderiam ser redirecionadas para um servidor SDN configurado especialmente na rede.

Balanceamento de carga. O balanceamento de carga é um serviço muito comum em redes atualmente, uma vez que seu objetivo principal é alcançar o melhor uso possível dos recursos disponíveis, através da divisão das requisições em servidores replicados. Essa otimização dos recursos acarreta o aumento da vazão das tarefas, a diminuição do tempo de resposta e, consequentemente, a melhoria do desempenho. E ainda mais, traz vantagens em termos de escalabilidade e disponibilidade. Contudo, as soluções para balanceamento existentes hoje no mercado, apesar de serem extremamente eficientes, possuem uma certa limitação quanto à flexibilização do serviço de acordo com as características próprias do negócio. Além disso, balanceadores de carga proprietários são geralmente muito caros, o que

inviabiliza sua implantação em muitas redes. Com SDN, o balanceamento de carga pode ser realizado diretamente pelos dispositivos de comutação da rede, sem a necessidade de se utilizar algum outro dispositivo específico para isso, bastando apenas o desenvolvimento de um código no controlador que execute tal serviço. Algumas soluções para esse serviço utilizam técnicas SDN de escrita de regras de encaminhamento ou de reescrita de campos de cabeçalho para realizar o balanceamento já no dispositivo comutador. Como este é um dos focos do presente trabalho, mais detalhes serão tratados ao longo do mesmo.

Monitoramento e medição. A utilização de SDN provê à rede a capacidade de realizar certas operações de monitoramento e medição sem a utilização de equipamentos e protocolos adicionais e sem ocasionar *overheads* (JARSCHEL et al., 2014). Isso ocorre devido a características intrínsecas do paradigma, que são: coletar as informações sobre a rede, a fim de manter uma visão global de seu estado; e transferir essas informações para um controlador logicamente centralizado. Assim, a informação pode ser processada em software com o objetivo de fornecer dados que atuem como parâmetros de monitoramento. Dessa forma, o administrador da rede pode tomar decisões estratégicas dinamicamente, uma vez que, com os parâmetros de interesse, ele passa a ter uma visão do estado corrente da rede, além de reagir a algum eventual mau funcionamento que venha a ocorrer em equipamentos da rede.

Gerenciamento de *data centers*. Atualmente, em TI, muitos serviços são altamente dependentes de *data centers* eficientes e escaláveis. E ao longo dos últimos anos, vem sendo registrado um aumento na demanda por esses serviços, o que torna crítico qualquer tipo de operação em *data centers*. Qualquer interrupção ou atraso nesses serviços pode acarretar prejuízos ou perdas severas em negócios (NUNES et al., 2014). Com SDN, diversas questões clássicas em *data centers* podem ser resolvidas ou mais bem tratadas, tais como a migração ativa de redes, a otimização da utilização da rede, o monitoramento e detecção de problemas em tempo real, o uso de mecanismos de segurança, evitar falhas iminentes, a adaptação de protocolos de transporte, a detecção de comportamentos anômalos na rede, o suporte a QoS, a economia de energia, dentre outras (HU et al., 2014; KREUTZ et al., 2015). Além disso, SDN pode ser capaz de fornecer isolamento de tráfego entre diversos usuários, utilizando virtualização de comutadores para interligar máquinas de usuários independente de suas localizações físicas. Inclusive, foi utilizando SDN que o Google implementou o B4 (BOSSHAART et al., 2014), um sistema que interconecta os *data centers* do Google ao redor do mundo, permitindo níveis de escalabilidade, tolerância a falhas e controle tais, que não poderiam ser alcançados por meio da arquitetura WAN existente.

Redes sem fio. Segundo KREUTZ et al. (2015), o atual plano de controle distribuído de redes sem fio é considerado de baixa qualidade no gerenciamento do espectro limitado, na alocação de recursos de rádio, no gerenciamento de interferências,

na implementação de mecanismos *handover* e no desempenho do balanceamento da carga entre as células. Com SDN torna-se mais fácil o desenvolvimento e gerenciamento de diferentes tipos de redes sem fio. Já existem soluções SDN que visam prover camadas programáveis e flexíveis para essas redes. Algumas soluções, como a OpenRadio (BANSAL et al., 2012), propõem uma camada de abstração para a separação entre o protocolo *wireless* padrão e o hardware, permitindo que as camadas MAC sejam compartilhadas através de diferentes protocolos utilizando plataformas multi-core. Outras, como a Odin (SURESH et al., 2012), utilizam o próprio protocolo padrão 802.11, mas realizam o isolamento lógico entre fatias da rede.

Ciência da aplicação. Intimamente relacionada ao que foi dito anteriormente em monitoramento, a ideia de ciência da aplicação consiste na capacidade de uma rede em manter as informações sobre as aplicações que se conectam a ela, e em utilizar essas informações para otimizar seu funcionamento, bem como o de outras aplicações ou outros sistemas que elas controlam. Utilizando SDN, a própria aplicação pode informar a rede sobre seu estado e propriedades. Dessa forma, decisões estratégicas de encaminhamento podem ser revistas dinamicamente, de acordo com as mudanças em tempo real na rede e em outras aplicações. Essa técnica também pode ser usada no caso em que há falta de recursos disponíveis para uma aplicação. Assim, a rede pode modificar o comportamento de outras aplicações para liberar recursos que, por sua vez, serão usados por uma aplicação com maior nível de criticidade, por exemplo.

Concluindo, existem ainda várias outras aplicações, áreas e serviços que podem se beneficiar das ideias do paradigma SDN, tais como redes ópticas, redes não baseadas em IP, VLANs, *cloud computing*, redes domésticas, *roaming* em telefonia, redes VoIP, dentre outras. Como já dito, essa lista de serviços em SDN é exemplificativa, uma vez que muitas aplicações ainda vêm sendo desenvolvidas e muitos trabalhos vêm sendo publicados na área.

3 DESEMPENHO EM SDN: DA TEORIA À PRÁTICA

O presente capítulo tem o propósito de discutir alguns aspectos de desempenho envolvidos na implementação de aplicações SDN em geral, e também na realização de balanceamento de carga em SDN, além de apresentar os trabalhos relacionados. Na seção 3.1 são apresentados os aspectos relacionados ao desempenho em planos de dados OpenFlow. Na seção 3.2 são apresentados os aspectos relacionados ao desempenho em operações de balanceamento de carga. Por fim, na seção 3.3 são sumarizados os trabalhos relacionados à pesquisa desenvolvida.

3.1 DESEMPENHO EM UM PLANO DE DADOS OPENFLOW

O desempenho de um plano de dados OpenFlow pode ser influenciado por uma série de fatores decorrentes do protocolo OpenFlow, além da própria característica do hardware utilizado atualmente nos switches. A seguir, estão listados alguns desses fatores.

Natureza do switch. Existem alguns tipos de switches OpenFlow, mas dois são os principais. O primeiro tipo é o hardware switch, geralmente um equipamento comercial, que utiliza memória especializada (TCAM) e sistema operacional proprietário para implementar a tabela de fluxos e o protocolo OpenFlow. O segundo tipo é o software switch, dispositivo que implementa tabelas de fluxos, entradas de tabela e campos de *match* como estruturas de dados através de software, geralmente utilizando sistemas UNIX/Linux e memória comum (SRAM ou DRAM) para implementar as funções de *switching* OpenFlow (BIANCO et al., 2010). Além desses tipos, existem também os denominados *whitebox* swicthes, que são equipamentos compostos por um hardware básico, mas que não mantêm tabelas de fluxos, deixando a cargo da aplicação a implementação dessas tabelas. Existem plataformas *whitebox* que oferecem aceleração no encaminhamento de pacotes por meio de software com DPDK.

Hardware switches processam pacotes através de componentes especializados de hardware denominados Circuitos Integrados de Aplicação Específica (ASICs), que geralmente alcançam vazão à velocidade de *link* sem degradação de desempenho, além de realizar o encaminhamento de dados em *line-rate* (velocidade conjunta de todas as portas enviando tráfego simultaneamente, em *full-duplex* e na máxima velocidade da interface), e implementar características avançadas como NAT, QoS, controle de acesso e reescrita IP. Software switches, por sua vez, processam dados através de CPUs, entretanto ficam delimitados pelo poder de seu processamento. Hardware switches são considerados mais rápidos do que software switches, pois usam memórias especializadas e hardware para realizar *matches* em paralelo, enquanto as CPUs de software switches utilizam altas taxas de processamento para alcançar um menor desempenho em relação aos ASICs. Portanto, em termos de desempenho e disponibilidade, hardware switches são melhores do que software

switches. Apesar disso, enquanto hardware switches suportam um número limitado de entradas na tabela de fluxos, os software switches podem suportar uma quantidade de regras em ordens de magnitude superiores, além de possuírem maior flexibilidade para implementar ações mais complexas (GORANSSON and BLACK, 2014).

Armazenamento de regras. Tipicamente, switches OpenFlow utilizam memórias TCAM, DRAM ou SRAM para implementar suas tabelas de fluxos e armazenar as regras sobre as quais serão realizados os *matches*. A memória TCAM (Ternary Content Addressable Memory) é um tipo de memória, desenvolvida em hardware especializado, que além de ser extremamente rápida, cara e com pouca capacidade de armazenamento, é utilizada para aumentar a velocidade de checagem de regras em uma tabela longa, uma vez que é possível fazer operações de *lookup* em várias entradas da tabela ao mesmo tempo. Nesse tipo de memória os bits podem ser representados com os valores "zero", "um" ou "não importa", sendo que este último indica que ambos os valores (zero ou um) são aceitáveis naquela posição (GUEDES et al., 2012). Memórias TCAM derivaram das memórias CAM, sendo que uma das poucas diferenças entre elas é que CAMs só armazenam bits "zero" e "um". Já as memórias DRAM e SRAM são memórias de acesso direto, comuns em computadores, que armazenam bits de dados em um capacitor. A diferença entre elas é que as DRAM são muito mais baratas, têm uma capacidade maior de armazenamento e um pior desempenho do que as SRAM. As memórias TCAM possuem um desempenho superior às memórias SRAM e DRAM na realização de *matches* e outras operações, embora sejam mais caras e tenham menor capacidade de armazenamento (KREUTZ et al., 2015). Geralmente hardware switches possuem tanto TCAM quanto SRAM, e utilizam uma ou outra de acordo com o tipo de regra a ser instalado.

Tipos de regra armazenadas e de *match* realizado. Uma regra pode ser armazenada em um switch OpenFlow tanto como uma regra de correspondência exata (na qual todos os atributos estão definidos) ou como uma regra coringa (ou *wildcard*, na qual existem atributos preenchidos com bits "não importa", que podem assumir qualquer valor). Em geral, switches comerciais utilizam SRAM para armazenar regras de *match* exato, que podem ser acessadas usando um algoritmo de *hash*, enquanto utilizam um esquema com TCAM para armazenar as regras curinga. Portanto, a capacidade de armazenamento de regras exatas é maior do que a de regras curinga (BANERJEE and KANNAN, 2014; YAN et al., 2014). Comparado com as regras de *match* exato, regras curinga melhoram a reutilização de regras na tabela de fluxo e reduzem o número de requisições de configuração de fluxo ao controlador, aumentando a escalabilidade do sistema (SHIRALI-SHAHREZA and GANJALI, 2015). Além disso, a realização de *matches* segue uma prioridade. *Matches* exatos sempre têm prioridade mais alta do que *matches* em regras curinga. Portanto, regras exatas, apesar de armazenadas em SRAM, devem ser verificadas antes das regras curinga, armazenadas em TCAM.

Verificação da tabela de fluxos. A busca por regras nas tabelas de fluxos para a realização de *matches* pode se dar de maneiras distintas. Alguns switches OpenFlow realizam pesquisas *hash* nas tabelas que contêm regras exatas, e pesquisa linear nas tabelas que contêm regras coringa (QU et al., 2013). Como as tabelas de regras coringa têm um tamanho bem limitado, a pesquisa linear acaba não comprometendo o desempenho da realização dos *matches*. As tabelas de regras exatas, por sua vez, podem possuir uma grande quantidade de entradas, mas como as regras são acessadas com complexidade O(1) devido ao *hash*, o impacto de busca da regra é mínimo. Memórias TCAM também podem realizar pesquisas em paralelo, em N regras ao mesmo tempo. Dependendo do custo do hardware, o N pode ser igual ao tamanho da tabela, o que geraria pesquisas com complexidade O(1).

Tamanho máximo das tabelas de fluxos. Quanto menor a capacidade de armazenamento de regras na tabela de fluxos um switch possuir, maior a chance de o switch apresentar problemas de desempenho ou de sobrecarga. Um exemplo disso ocorre quando a tabela de fluxos do switch fica lotada. O switch deve remover uma entrada mais antiga ou a menos utilizada antes de inserir uma nova entrada, o que já insere um atraso na instalação da regra. E mais, se a entrada que foi removida representa um fluxo que ainda está ativo na rede, então o switch, ao receber os pacotes subsequentes daquele fluxo, os deve encaminhar para o controlador, uma vez que, possivelmente, não realizarão *match* em nenhuma das entradas restantes da tabela de fluxos. Além de criar uma sobrecarga súbita no controlador, isso pode resultar em uma queda significativa na vazão do fluxo (SHIRALI-SHAHREZA and GANJALI, 2015).

Tempo de instalação das regras. A instalação de regras nas tabelas de fluxos pode seguir dois modelos: um modelo reativo e um modelo proativo. No modelo proativo, as regras são previamente instaladas pelo controlador diretamente no switch, não havendo a necessidade de enviar o primeiro pacote de cada fluxo para o controlador. Já no modelo reativo, o switch deve consultar o controlador cada vez que um pacote de um novo fluxo chega ao switch. Inclusive, é nesse tipo de modelo que surge a pequena queda de desempenho característica de uma SDN, causada pela chegada do primeiro pacote de um novo fluxo, que deve sempre ser encaminhado ao controlador. Os demais pacotes irão combinar com a regra recém-instalada no switch. Associado a isso, existe o problema da limitação de tamanho do *buffer*. Switches OpenFlow normalmente armazenam os pacotes recebidos em um *buffer* e enviam apenas uma pequena parte de cada pacote para o controlador, quando o pacote não corresponde a qualquer entrada na tabela de fluxos. No entanto, os *buffers* de switch são geralmente pequenos. Mesmo em software switches, onde quase não há problemas de custo ou de energia, *buffers* também são relativamente pequenos. Por exemplo, no Open vSwitch o tamanho padrão desse *buffer* é de apenas 256 pacotes. Como resultado, sob cargas pesadas, este *buffer* pode tornar-se cheio rapidamente, forçando o switch a enviar os pacotes inteiros para o controlador, o que aumenta tanto

o atraso de transmissão quanto a carga do controlador, levando assim a maiores atrasos totais e a queda no desempenho (SHIRALI-SHAHREZA and GANJALI, 2015). Portanto, múltiplas mensagens de instalação de regras quando chegam simultaneamente ao switch OpenFlow podem acarretar queda no desempenho ou até mesmo a falha dos dispositivos.

3.2 DESEMPENHO EM UM BALANCEADOR DE CARGA

O balanceamento de carga em redes de computadores é uma técnica conhecida e amplamente utilizada para a distribuição da carga de trabalho entre vários enlaces ou servidores, geralmente replicados (UPPAL and BRANDON, 2010). O balanceador de carga melhora o desempenho da rede através do uso otimizado dos recursos disponíveis e ajuda a minimizar a latência e o tempo de resposta e maximizar a vazão. Assim, mais tarefas poderão ser realizadas na mesma quantidade de tempo e, de forma geral, todas as requisições poderão ser respondidas em menos tempo (RAGALATHA et al., 2013). Além disso, o balanceamento de carga traz vantagens como o aumento da escalabilidade, uma vez que passará a existir uma maior flexibilidade na adição de novos recursos à rede, sempre de forma rápida e transparente aos usuários finais. Outras vantagens trazidas pelo balanceamento de carga são o aumento do desempenho do sistema como um todo, devido à utilização dos recursos de uma forma mais inteligente; e uma maior disponibilidade do sistema, já que se um dos servidores falhar, a carga poderá ser redistribuída aos outros servidores existentes sem comprometer o processamento das requisições (KOERNER and KAO, 2012).

Em um modelo tradicional de balanceamento de carga, as requisições dos clientes são encaminhadas a um balanceador especializado, localizado à frente da rede de destino, que, através de um algoritmo de balanceamento, irá redistribuir os acessos externos dos clientes para diferentes servidores *backend*. Esse balanceador deve ser capaz de manter as sessões, manipular pacotes, funcionar como NAT, além de realizar o balanceamento das requisições. Já em um modelo de balanceamento de carga em SDN, o controlador da rede assume as funções decisórias de um balanceador e passa a comandar o processo de balanceamento através da programação da tabela de fluxos do switch, que irá redistribuir as requisições de acordo com o que foi definido pelo controlador (QILIN and WEIKANG, 2015). Dessa forma, de acordo com o algoritmo implementado pelo controlador, o processo de balanceamento ocorre diretamente no switch, sem a necessidade de um balanceador dedicado, que pode ser extremamente caro (RAGALATHA et al., 2013). Contudo, alguns fatores podem influenciar diretamente o desempenho de um balanceador de carga em SDN. Alguns desses fatores serão descritos a seguir.

Política de balanceamento e número de servidores. O algoritmo de balanceamento pode impactar diretamente o funcionamento do serviço, uma vez que, dependendo do tipo de algoritmo, um servidor pode ser submetido a uma carga maior de trabalho do

que outro. Por exemplo, uma escolha aleatória entre servidores para atender requisições pode distribuir a carga desigualmente, deixando alguns servidores mais sobrecarregados do que outros. Ou ainda, se a escolha for baseada no uso de CPU, o servidor que possui o menor uso de CPU pode receber uma grande quantidade de requisições seguidamente, até ficar sobrecarregado e ter seu valor de CPU aumentado. Além disso, quanto mais servidores replicados um sistema possuir, tende a existir uma melhor distribuição da carga de trabalho nesse sistema.

Características dos fluxos. Quando as requisições chegam ao controlador, a princípio não é possível determinar as características dos fluxos que serão instalados. Depois que o controlador, a partir dos dados do primeiro pacote, instala uma regra na tabela de fluxos, podem se estabelecer fluxos suficientemente breves (denominados fluxos "camundongo"), composto por poucos pacotes que, individualmente, não trarão impacto significativo ao sistema. Contudo, podem também se estabelecer fluxos muito intensos (denominados fluxos "elefante") que, dependendo da programação estabelecida, vão ser direcionados a apenas um servidor, o sobrecarregando, e consequentemente influenciando o desempenho do sistema como um todo (CURTIS et al., 2011).

Instalação de regras e quantidade de requisições. No modelo proativo, as regras já são previamente instaladas nas tabelas de fluxo, o que já adianta o processo de balanceamento das requisições. Contudo, dependendo das características dos fluxos que chegam ao switch, muitas das requisições podem ser redirecionadas ao mesmo servidor, impactando o desempenho e a otimização do sistema. Por outro lado, no modelo reativo, no qual as regras são instaladas dinamicamente de acordo com as características de cada fluxo, o balanceamento pode ser mais bem executado, uma vez que a escolha do servidor que vai atender as requisições pode ser feita de acordo com as características momentâneas dos servidores ou com algum outro processo justo de escolha (por exemplo, *Round-Robin*). Entretanto, como cada primeiro pacote de novos fluxos deve ser inspecionado antes que seja instalada uma entrada referente ao seu fluxo na tabela, uma chegada massiva de novos fluxos ao switch pode ocasionar uma queda de desempenho do mesmo, uma vez que o controlador pode não conseguir lidar com tantas requisições de instalação de fluxo simultâneas.

3.3 TRABALHOS RELACIONADOS

Muitas pesquisas e trabalhos relacionados à SDN e ao padrão OpenFlow vêm sendo realizados nos últimos anos. Contudo, ainda existem alguns problemas que não foram completamente tratados em SDN e sobre os quais ainda não foram desenvolvidos muitos trabalhos. Dentre eles se destacam os dois problemas que são o foco dessa dissertação: o balanceamento de carga em SDN e a caracterização das implementações OpenFlow em equipamentos de rede. Nas seções a seguir, serão apresentados trabalhos relacionados a

cada um dos problemas supracitados. Esses trabalhos fornecem um panorama geral sobre como cada um dos problemas vem sendo tratado atualmente, refletindo o estado-da-arte atual e apontando possíveis deficiências encontradas nas pesquisas acerca desses temas.

3.3.1 Trabalhos sobre avaliação de planos de dados

Apesar da importância do protocolo OpenFlow ressaltada anteriormente, principalmente no que se refere ao seu desenvolvimento associado aos projetos já existentes de equipamentos de rede, também ainda não existem muitos trabalhos que enfoquem o plano de dados dos switches OpenFlow, bem como suas implementações e mecanismos internos. Alguns trabalhos até chegam a abordar, de certa forma, a temática do plano de dados. No entanto, tais trabalhos possuem abordagem e objetivos diferentes dos definidos para esta dissertação.

A título de exemplo, é o que ocorre no trabalho de BIANCO et al. (2010), que consiste na análise do desempenho de um único switch, implementado virtualmente sobre uma bridge. Os autores comparam o desempenho desse switch em modo OpenFlow, com o desempenho do mesmo operando através de roteamento IP e de *switching* Ethernet, em seu modo normal (*legacy*). Ou seja, o principal objetivo do trabalho consiste na comparação de técnicas de encaminhamento distintas, utilizando o mesmo equipamento. Esse trabalho apresenta métricas de desempenho bem definidas, com foco em vazão de encaminhamento e latência de pacotes, sob condições normais e de sobrecarga, com diferentes padrões de tráfego e diferentes tamanhos de pacotes na rede. Algumas características de sua metodologia, assim como algumas métricas de desempenho, serviram como base para a formulação de testes no presente trabalho.

O trabalho de SÜNNEN (2011) também avalia o desempenho de um único switch OpenFlow, entretanto, nesse caso, o equipamento é um hardware switch. De um modo geral, o trabalho tenta buscar métricas que melhor caracterizem um ambiente OpenFlow, e estabelece uma infraestrutura básica para testes composta por máquinas virtuais. Algumas das métricas de avaliação definidas dependem fortemente do desempenho do controlador, o que não é o foco nesta dissertação. Apesar disso, algumas outras métricas como, por exemplo, a medição do atraso em diferentes tipos de *matches*, inspiraram a elaboração de alguns testes. Inclusive, a medição do atraso foi realizada a partir do cálculo do Round Trip Time (RTT), através dos *timestamps* de cada pacote nos momentos de partida e de retorno à máquina geradora de tráfego.

Por sua vez, no trabalho de APPELMAN and DE BOER (2012), foi realizada, pela primeira vez, uma comparação entre diferentes switches OpenFlow. Nesse caso, foram comparadas três plataformas OpenFlow, sendo dois hardware switches e uma versão em software do Open vSwitch com sua típica implementação do protocolo OpenFlow. Contudo, a série de testes de desempenho contou fortemente com a participação do controlador, o que

influenciou de forma considerável as avaliações conduzidas. Esse trabalho foi tecnicamente bem definido, e algumas de suas análises, como a interoperabilidade entre firmware OpenFlow, hardware do switch e software de controle, na influência do desempenho dos switches, podem servir como grau de comparação para alguns dos resultados obtidos nesta dissertação. Um dos principais resultados do trabalho é a constatação de que, embora a maioria das características de encaminhamento básico esteja disponível nas implementações, a escalabilidade é uma característica que ainda não está sendo atendida efetivamente pelo padrão OpenFlow.

Finalmente, o foco principal do trabalho de ROTSOS et al. (2012) é a proposição de um *framework* aberto e genérico que permite o teste de switches OpenFlow. Eles desenvolveram uma ferramenta que testa a interação entre o mecanismo de encaminhamento do switch e a aplicação de controle remota, tanto para hardware switches quanto para software switches, medindo suas capacidades e seus gargalos. O trabalho apresenta a ferramenta, assim como avalia algumas implementações de switches OpenFlow. Um dos principais resultados do trabalho aponta que o desempenho do switch depende de seu firmware e das ações realizadas diretamente por ele. Assim como os trabalhos supracitados, a participação do controlador nos experimentos não permite que os resultados sejam conclusivos sobre o desempenho único e exclusivo dos switches.

Dessa forma, podem ser destacados os seguintes diferenciais do presente trabalho para a avaliação de planos de dados OpenFlow: em primeiro lugar, serão comparadas diferentes implementações OpenFlow de hardware switches comerciais e de software switches *open source*. Essa comparação é realizada através de métricas de desempenho de interesse, que levam em consideração dados como latência e *jitter*, em determinadas operações do switch. E nessa comparação, foram analisados apenas aspectos referentes ao plano de dados dos switches, descartando a influência do controlador nas operações. Em seguida, serão realizadas comparações entre os desempenhos de cada switch em seus modos normal e OpenFlow. Por fim, será discutido se é vantajoso ou não utilizar o protocolo OpenFlow nos switches analisados. Em parte, os resultados permitem avaliar se as implementações do protocolo são definidas, de fato, em hardware, ou se são apenas instâncias de um software switch adaptadas ao mecanismo interno do switch.

3.3.2 **Trabalhos sobre balanceamento de carga**

Balanceamento de carga é um tema de pesquisa atrativo, uma vez que um balanceador dedicado pode se tornar um ponto crítico de falha e congestionamento (WANG et al., 2011). Além disso, algumas das soluções existentes são específicas para determinados serviços, o que as tornam pouco flexíveis com relação à visão da rede como um todo. Isso sem contar que esses equipamentos dedicados podem ser extremamente caros, o que inviabilizaria a sua utilização em muitas redes existentes. Entretanto, pode-se realizar o ba-

lanceamento de carga de uma forma mais simples, prática, rápida e barata, empregando-se os conceitos de SDN.

Alguns trabalhos recentes em balanceamento de carga apresentam propostas baseadas em SDN, entretanto não apresentam uma avaliação aprofundada das suas propostas (HANDIGOL et al., 2009; UPPAL and BRANDON, 2010; WANG et al., 2011; RAGALATHA et al., 2013; ZHOU et al., 2014). De fato, alguns destes são trabalhos ainda em desenvolvimento. Praticamente todos eles compartilham de semelhanças nas arquiteturas propostas: utilizam o padrão OpenFlow e um controlador baseado em NOX, e realizam um esquema de reescrita tanto de cabeçalho de pacotes quanto das entradas da tabela de fluxos do switch OpenFlow.

Em alguns casos, o balanceamento de carga é um efeito colateral da flexibilidade em se adicionar recursos de forma transparente à rede. Um exemplo disso ocorre no trabalho de HANDIGOL et al. (2009), no qual os autores desenvolveram um esquema que realiza o monitoramento da realidade da rede em termos de disponibilidade de recursos e distribui a carga de trabalho entre os servidores. Para realizar essa distribuição, existem módulos específicos que monitoram os servidores existentes e seu estado. Esses módulos modificam ativamente as regras no controlador para realizar a distribuição de carga. Os testes foram realizados fisicamente com servidores *Web* e com switches comerciais que possuem o OpenFlow.

Outros trabalhos procuram realizar um balanceamento dinâmico de carga. Por exemplo, no trabalho de RAGALATHA et al. (2013) é implementado um balanceador no qual a carga de trabalho é calculada e distribuída entre os servidores existentes em tempo de execução e não no início da execução do serviço. Há um monitoramento contínuo da carga dos servidores e dos caminhos até os servidores para a realização do balanceamento. O algoritmo desenvolvido pelos autores calcula o melhor caminho para realizar o balanceamento das requisições dos clientes, considerando também o caminho para o servidor com menor carga de trabalho que possa atender essas requisições. Já no trabalho de WANG et al. (2011) o balanceamento dinâmico é realizado através da instalação de regras curingas nos switches. Eles propõem algoritmos para computar essas regras e ajustá-las de acordo com as mudanças no sistema. Isso ocorre sem quebras nas conexões já existentes. Contudo, esse trabalho avalia o balanceamento das requisições a um único arquivo sintético, o que não reflete a realidade de uma rede de fato, implicando, assim, nos resultados dos experimentos.

Contudo, esse trabalho avalia o balanceamento das requisições a um único arquivo sintético, de tamanho não representativo,

O trabalho de KOERNER and KAO (2012) também se preocupa com balanceamento de carga dinâmico. Porém, os autores tratam a rede de forma diferente dos trabalhos discutidos anteriormente. Nesse caso, o controlador NOX é utilizado juntamente com o

FlowVisor, que divide os recursos de rede e utiliza um controlador diferente para cada divisão. Dessa forma, cada fatia da rede pode ter seu próprio balanceamento, independente das demais. Isso provê maneiras de alocar recursos e balancear diferentes serviços, independentes um do outro. Neste trabalho, os testes também foram feitos em uma rede física, onde foram avaliados os elementos da rede e a viabilidade do conceito proposto.

Por sua vez, no trabalho de UPPAL and BRANDON (2010) foi proposta e avaliada uma arquitetura típica de balanceamento de carga, composta por um switch OpenFlow que interconecta um controlador NOX e múltiplos servidores. A proposta consiste na avaliação de diferentes políticas de balanceamento e seu efetivo impacto no desempenho do sistema, através da medição de métricas como vazão e latência. Inclusive, as políticas de balanceamento descritas em UPPAL and BRANDON (2010) foram utilizadas na proposta do presente trabalho. Apesar disso, o balanceamento é realizado com um único perfil de carga *Web*, que pode não representar a realidade.

Apesar dos trabalhos apontados mostrarem estratégias diversas para realizar balanceamento de carga em SDN, não ficam claros os benefícios alcançados. Alguns deles não realizam uma avaliação consistente, pois utilizam cargas simplistas quando comparados a cenários mais próximos à realidade, como os descritos em JEON et al. (2012); POLFLIET et al. (2012); BOVENZI et al. (2011). Dessa forma, esses trabalhos apresentam resultados que podem não ser conclusivos.

Dessa forma, podem ser destacadas as seguintes contribuições do presente trabalho para o balanceamento de carga: em primeiro lugar, será apresentado um mecanismo simples e flexível para realizar balanceamento de carga em SDN. A implementação desse mecanismo será apresentada e discutida, além de serem mostradas soluções simples para capturar informações da rede, sem que ocorra a quebra da premissa básica de divisão entre os planos de dados e de controle. Em seguida, será apresentada uma avaliação do mecanismo implementado. Serão propostos cenários de acesso a diferentes perfis de arquivo para realizar tal avaliação, além de serem frisadas algumas limitações ainda existentes para o balanceamento de carga usando SDN.

4 AVALIAÇÃO DE BALANCEAMENTO DE CARGA WEB EM SDN

Neste capítulo será apresentada a pesquisa relacionada à avaliação de mecanismos de balanceamento de carga em uma SDN utilizando OpenFlow, com foco em cargas típicas relacionadas a *Web*. Parte desta pesquisa e de seus resultados pode ser encontrada no artigo de RODRIGUES et al. (2015), que foi submetido, aprovado e apresentado no XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2015).

Na seção 4.1 se encontra um breve resumo dos objetivos dessa pesquisa. Na seção 4.2 estão descritos o mecanismo de balanceamento proposto, o método de funcionamento do balanceamento e, por fim, a arquitetura utilizada e as políticas de balanceamento propostas. Na seção 4.3 está descrita a metodologia de pesquisa utilizada nesta parte do trabalho, abrangendo o ambiente de experimentação, de uma forma mais detalhada, com o papel de cada elemento da arquitetura, assim como o funcionamento dos testes e os três cenários para a avaliação de desempenho das políticas de balanceamento. Na seção 4.4 estão os resultados obtidos na realização dos testes e as avaliações desses resultados, para cada um dos cenários propostos. Por fim, a seção 4.5 traz algumas considerações gerais sobre o trabalho, os experimentos, os resultados e possíveis melhorias.

4.1 OBJETIVOS GERAIS

De um modo sucinto, o objetivo desta parte do trabalho é apresentar uma proposta de balanceamento de carga em SDN, na qual diferentes políticas podem ser utilizadas para balancear requisições que chegam em um determinado sistema. Será avaliado o comportamento do sistema quando da utilização de três políticas de balanceamento diferentes: uma política aleatória, uma que segue o padrão *Round-Robin* e uma que leva em consideração a carga de trabalho dos servidores.

O funcionamento desse arcabouço de balanceamento de carga será avaliado com foco em serviços *Web*, nos quais um cliente fará requisições por um determinado arquivo armazenado em servidores *Web*, e tais requisições serão balanceadas de acordo com a política selecionada. Para melhor caracterizar diferentes perfis de carga *Web*, foram formulados três cenários, um para cada tipo de carga, onde serão avaliados perfis de carga mais leve, cargas médias e cargas pesadas. Serão mostrados os benefícios alcançados e as limitações quando o balanceador tem que lidar com diferentes tipos de carga.

A ideia inicial é avaliar o mecanismo de balanceamento de carga proposto em um ambiente SDN virtual, para posteriormente ampliá-lo e experimentá-lo em um ambiente SDN real (com implementações OpenFlow de hardware switches) a fim de comparar os comportamentos observados em cada um dos ambientes e analisar a maturidade do protocolo OpenFlow para realizar esse tipo de serviço em redes de produção. A primeira tentativa de montagem do mecanismo de balanceamento foi desenvolvida em

um ambiente Mininet, que é um emulador de rede virtual preparado para desenvolver e experimentar sistemas baseados em SDN e OpenFlow. Em um ambiente Mininet, toda a arquitetura necessária para realizar a simulação (máquinas, switches, controladores e *links*) é virtualizada e agrupada em uma única máquina virtual.

Contudo, para o mecanismo de balanceamento proposto, foi encontrada uma limitação que impediu a utilização do ambiente Mininet. Essa limitação ocorria na obtenção da carga de trabalho dos servidores virtuais. Os valores de carga de trabalho, obtidos por meio do sistema operacional Linux de cada servidor virtual, eram sempre iguais em ambos os servidores. Isso ocorria porque esses valores eram cópias exatas do valor da carga da máquina virtual Mininet que continha os servidores virtuais. Esse fato impossibilitou a realização do balanceamento baseado nos valores de carga nesse ambiente. A partir disso, foi desenvolvido um novo ambiente de experimentação, também virtual, mas baseado na utilização do Open vSwitch e de máquinas virtuais distintas. A pesquisa apresentada no presente capítulo refere-se à avaliação do mecanismo de balanceamento de carga implantado nesse novo ambiente.

4.2 MECANISMO DE BALANCEAMENTO PROPOSTO

A Figura 6 apresenta a arquitetura de rede proposta nesse trabalho. Os servidores ilustrados hospedam os servidores *Web* da arquitetura e são os responsáveis por atender as requisições efetuadas pelo cliente. Esses servidores se ligam ao mundo externo por meio de um único endereço IP, que será referido aqui como endereço IP externo do serviço. Quando um cliente faz uma nova requisição para o endereço IP externo do serviço, ela é encaminhada para o switch OpenFlow. Esse switch possui regras específicas e redireciona a requisição a um dos servidores do sistema, balanceando assim a carga total de serviço. As regras de balanceamento que o switch executa são definidas pelo controlador OpenFlow.

Mais detalhadamente, o funcionamento do sistema ocorre como descrito a seguir: quando o primeiro pacote de uma nova solicitação chega ao switch, ele é repassado ao controlador. O controlador, inicialmente, escolhe qual servidor do conjunto de servidores que respondem ao serviço irá atender a requisição. Após essa escolha, o controlador cria uma entrada (ou regra) na tabela de fluxos do switch associada a esse pacote, que define a ação de encaminhamento do pacote para a porta de saída específica associada ao servidor escolhido. Além disso, a entrada na tabela também define a realização da reescrita dos campos de endereços MAC e IP de destino desse pacote, substituindo os endereços MAC e IP do endereço de serviço pelos endereços MAC e IP do servidor escolhido. Vale ressaltar que os demais pacotes pertencentes à mesma solicitação, quando chegarem ao switch, já encontrarão as regras de encaminhamento instaladas na tabela, não sendo necessário o repasse dos mesmos ao controlador. Esses pacotes apenas sofrerão as ações (encaminhamento e modificação) definidas pela entrada da tabela que foi associada à

requisição. Além de instalar as regras que ligam o cliente ao servidor, o controlador também instala, ao mesmo tempo, regras para permitir o fluxo de retorno do servidor para o cliente. Neste caso serão modificados os campos de origem do pacote. Para a resposta à requisição, os endereços IP e MAC de origem do pacote de resposta, que são os endereços do servidor, serão substituídos pelos endereços IP e MAC externos de serviço. Com isso, o cliente não sabe para qual servidor foi enviada sua requisição e de qual servidor ele obteve resposta, mantendo um certo nível de transparência e de segurança no serviço.

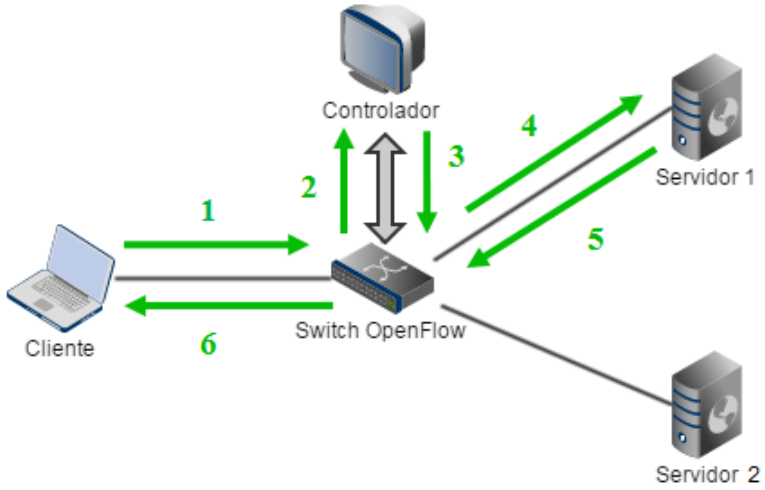

Figura 6 – Arquitetura da rede proposta: (1) A requisição do cliente é enviada ao switch; (2) Como não há entrada especificada na tabela de fluxos, o primeiro pacote da requisição é repassado ao controlador; (3) O controlador escolhe o servidor de destino, cria uma entrada correspondente na tabela de fluxos e devolve o pacote ao switch; (4) A requisição é enviada ao servidor escolhido; (5) A requisição é atendida pelo servidor, que envia a resposta de volta ao switch; (6) A resposta da requisição é enviada ao cliente.

O processo de escolha do servidor que vai atender as requisições, e, por consequência, realizar o balanceamento, fica a cargo do controlador. Neste trabalho, são utilizadas três políticas de balanceamento das requisições, cada uma definida por um código diferente a ser executado pelo controlador. As políticas de balanceamento de carga, que foram implementadas por códigos na linguagem Python, são detalhadas a seguir:

1. **Política aleatória**: O controlador escolhe aleatoriamente para qual servidor enviar a solicitação. Essa escolha é uniformemente distribuída entre os servidores existentes.

2. **Política Round-Robin**: Para cada fluxo novo, um servidor é escolhido alternadamente para receber a solicitação do cliente, a partir do primeiro servidor. Assim, as requisições são divididas uniformemente entre todos os servidores.

3. **Política baseada na carga**: Essa política envia a solicitação para o servidor com a menor carga média de CPU. Essa carga média é calculada pelo sistema operacional através da média do último minuto de uso da CPU. O valor da carga é obtido no arquivo de sistema /PROC/LOADAVG de cada servidor Linux. Os servidores enviam as suas cargas para a máquina onde se encontra o controlador, a uma taxa de 10 vezes por segundo, através de um canal de comunicação estabelecido via *Sockets* Python, por uma rede externa à rede SDN.

4.3 METODOLOGIA DE AVALIAÇÃO

Nesta seção serão descritos os aspectos relacionados à execução dos testes de balanceamento de carga realizados em uma SDN, utilizando OpenFlow. A seguir estão descritos, dentre outros detalhes, as características da topologia da rede de avaliação, com os papéis de cada elemento na arquitetura; a caracterização da carga de testes, através da descrição do funcionamento dos experimentos e alguns detalhes dos procedimentos adotados; e a descrição de três cenários de teste referentes a diferentes tipos de carga *Web*, o que permite uma avaliação mais realística dos diferentes perfis de carga.

4.3.1 **Ambiente de testes**

A Figura 6, mostrada anteriormente, ilustra a arquitetura SDN criada para a realização dos testes do esquema de balanceamento de carga proposto. Essa arquitetura foi criada virtualmente, ou seja, foi desenvolvida dentro de uma máquina física, com a utilização de três máquinas virtuais e um switch virtual Open vSwitch. A máquina física é um computador com processador Intel Core i7-860 2.80 GHz, configurado com sistema operacional Linux Ubuntu 14.04, 4 GB de memória e 256 GB de armazenamento em SSD.

Uma das máquinas virtuais existentes na arquitetura representa uma máquina cliente de testes, responsável por realizar requisições. As outras duas máquinas virtuais representam os dois servidores *Web* que irão atender às requisições e realizarão o balanceamento. Todas essas máquinas virtuais estão em execução dentro da máquina física onde se encontra o switch virtual OpenFlow e são gerenciadas através do VirtualBox. A máquina cliente possui sistema operacional Linux Ubuntu 14.04, com 512 MB de memória. Nela foi instalada a ferramenta que foi responsável por enviar as requisições ou cargas: o Httperf (MOSBERGER and JIN, 1998). Já os servidores possuem sistema operacional Linux Ubuntu Server 12.04, com 1 GB de memória. Em cada servidor virtual foi instalado o servidor *Web* Apache versão 2.2. O Apache, que é o servidor *Web* mais utilizado atualmente, foi escolhido por ser de fácil uso e por atender bem aos requisitos da arquitetura, não havendo necessidade de serem realizadas configurações adicionais nele.

Para realizar a comunicação direta entre os servidores e o controlador, sem violar o princípio básico de separação dos planos de controle e dados existente em Redes Definidas

por Software, uma rede externa à SDN foi criada. Essa rede externa foi criada para o envio de dados entre os servidores (através da segunda interface virtual de rede) e o controlador, que trocarão entre si informações sobre as cargas dos servidores. Essa rede externa foi criada utilizando uma segunda placa de rede contida na máquina física, tornando possível a comunicação direta com os servidores externamente à rede SDN já existente.

O Open vSwitch (OvS) (PFAFF et al., 2009) foi utilizado como elemento de chaveamento da arquitetura implementada. O OvS é um switch virtual multi-camadas com suporte ao protocolo OpenFlow e de código aberto, construído para funcionar em ambientes virtualizados, e que suporta a criação de VLANS e GRE *tunneling*. O OvS é instalado na máquina física que comporta toda a arquitetura virtual e é associado à rede virtual por meio de uma *bridge* (ponte) configurada em uma das interfaces físicas dessa máquina. Essa *bridge* fornece conectividade entre as máquinas virtuais e a interface física por meio da criação de interfaces virtuais que são associadas a essas máquinas virtuais.

O controlador SDN empregado foi o POX (POX, 2015), desenvolvido na linguagem Python. Ao contrário dos servidores da arquitetura, o controlador não foi instalado em uma máquina virtual; ele foi implementado diretamente na máquina física. No controlador foram implementadas as três políticas de escalonamento avaliadas nesse trabalho. Conforme mencionado anteriormente, o controlador recebe informações sobre os servidores em uma interface à parte da SDN. Por essa interface, o controlador recebe a carga de CPU de cada servidor.

Tanto a rede quanto as máquinas virtuais utilizadas foram dedicadas exclusivamente ao experimento, de modo que houvesse o mínimo de impacto possível causado tanto pela latência da rede quanto por alguma configuração da rede ou das máquinas virtuais. Vale ressaltar também que a largura de banda máxima alcançada foi de 300 Mb/s.

4.3.2 **Cargas de trabalho**

Para enviar as cargas de teste, a partir da máquina cliente, foi utilizada a ferramenta Httperf. O Httperf é uma ferramenta que tem como propósito medir o desempenho de um servidor *Web*. Ela é capaz de gerar diversas cargas de trabalho HTTP a partir de parâmetros pré-definidos. Ao final de sua execução, o Httperf sumariza as estatísticas gerais de desempenho do servidor avaliado, como o tempo total de conexão, por exemplo. Cada requisição gerada pelo Httperf é enviada através de uma porta TCP/UDP distinta, o que faz com que os pacotes dessa requisição tenham pelo menos o atributo *TCP/UDP src port* diferente dos pacotes das outras requisições, tornando cada requisição uma entidade individual. Como nesses testes o controlador foi configurado para instalar regras utilizando todos os 12 atributos especificados no OpenFlow 1.0, cada requisição, por ser única, vai necessariamente especificar um novo fluxo distinto que, ao chegar ao switch, será sempre encaminhado ao controlador.

Os testes foram realizados em três cenários distintos, sendo que, em cada um deles, foram utilizadas as três políticas de balanceamento de carga descritas na Seção 4.2. O objetivo dos testes é obter os tempos médios de conexão, através de experimentos, para diferentes taxas de chegada de requisições por segundo, para a posterior avaliação de desempenho dos algoritmos em cada situação. Um experimento consiste em 10 execuções do Httperf a uma taxa de chegada pré-definida e fixa, que retornará como resultado a média dos 10 tempos totais de conexão das execuções, para aquele valor de taxa. Cada experimento é executado diversas vezes, variando-se a taxa de chegada, de acordo com o cenário avaliado. O intervalo de confiança dos resultados obtidos é de 95%. As taxas de chegada são um conjunto de valores determinísticos, definidos como parâmetro no Httperf. Várias taxas foram utilizadas nos experimentos em cada cenário, porém as mais representativas em termos gráficos foram escolhidas para ilustrar o comportamento de cada política de balanceamento.

4.3.2.1 Cenário 1: Carga *Web* leve

O primeiro cenário definido retrata o balanceamento em um sistema com perfil de carga mais leve. Um comportamento típico de usuários nesse cenário seria o de realizar um acesso a uma página comum da *Web* com textos e figuras pequenas (BOVENZI et al., 2011; WILLIAMS et al., 2005; GOSEVA-POPSTOJANOVA et al., 2004). No teste desse cenário foi simulado o acesso a um pequeno arquivo HTML, que representa uma página *Web* com texto e *links* para outros objetos. Cada acesso a esse pequeno arquivo gera o total de 20 requisições de 10 KB cada, totalizando 200 KB no acesso à pagina. Cada experimento consistiu na realização de 1000 acessos ao arquivo, a uma taxa de chegada fixa. Foram realizados vários experimentos sendo que, a cada novo experimento, uma nova taxa de chegada era configurada. Os valores de taxa de chegada utilizados variaram entre 100 e 1000 requisições por segundo.

4.3.2.2 Cenário 2: Carga *Web* média

O segundo cenário definido retrata o balanceamento em um sistema com carga *Web* média. Nesse cenário, usuários enviam e recuperam de servidores *Web* arquivos nitidamente maiores do que simples páginas. Como exemplo, esse arquivos poderiam ser fotos ou arquivos de texto com cerca de 1 MB de dados (JEON et al., 2012; POLFLIET et al., 2012; NAGPURKAR et al., 2008). Neste teste, o balanceador de carga será avaliado através de requisições a um arquivo JPG de 1 MB. Novamente, cada experimento consistiu na realização de 1000 acessos ao arquivo, a uma taxa de chegada fixa. Foram realizados vários experimentos sendo que, a cada novo experimento, uma nova taxa de chegada era configurada. Os valores de taxa de chegada utilizados variaram entre 10 e 1000 requisições por segundo.

4.3.2.3 Cenário 3: Carga *Web* pesada

O terceiro cenário definido retrata o balanceamento em um sistema com perfil de carga mais pesado. Para esse teste, será simulado o *download* de um arquivo de vídeo de 100 MB. Neste cenário, cada experimento consistiu na realização de 100 *downloads* de um arquivo de vídeo, a uma taxa de chegada fixa. Foram realizados vários experimentos sendo que, a cada novo experimento, uma nova taxa de chegada era configurada. Os valores de taxa de chegada utilizados variaram entre 1 e 100 requisições por segundo.

Em um primeiro momento, não foram avaliados cenários compostos por diferentes tipos de carga atuando simultaneamente. Contudo, o presente trabalho é capaz de refletir, de fato, um cenário realístico, uma vez que vem se tornando cada vez mais comum a utilização de servidores especializados para cada tipo de requisição (por exemplo, um servidor para as requisições *Web* e outro para acesso às mídias relacionadas).

4.4 RESULTADOS DOS EXPERIMENTOS

Nesta parte do trabalho serão mostrados os resultados dos testes realizados para cada um dos cenários citados na seção anterior. Os gráficos apresentam a relação entre o tempo médio de duração das conexões e a taxa de requisições enviadas pelo cliente. O propósito de analisar essa relação é verificar o impacto do aumento da taxa sobre o tempo de processamento das conexões pela perspectiva do cliente, na utilização das diferentes políticas de balanceamento.

4.4.1 **Carga *Web* leve**

No primeiro cenário, os testes foram realizados na simulação de acesso a um arquivo HTML. Em cada um dos experimentos foram estabelecidas 1000 conexões ao arquivo, com 20 requisições em cada conexão. A Figura 7 apresenta o tempo médio de resposta de uma conexão e o intervalo de confiança correspondente, enquanto é variada a taxa de chegada nos servidores.

Analisando-se o gráfico do cenário em questão, para valores menores de taxa de chegada, as três políticas de balanceamento notadamente apresentaram resultados semelhantes entre si. De fato, pode-se observar que, para todas as taxas avaliadas, os intervalos de confiança de todas as políticas apresentam interseções entre si. Isso pode ser explicado pelo fato de que as cargas leves, como, por exemplo, acessos a simples arquivos, podem ser atendidas muito rapidamente. Dessa forma, qualquer um dos servidores que for selecionado para atender uma nova requisição estará com sobra de recursos computacionais. Ou seja, para esse caso, a forma de escolha do servidor não impactará nos tempos médios de conexão.

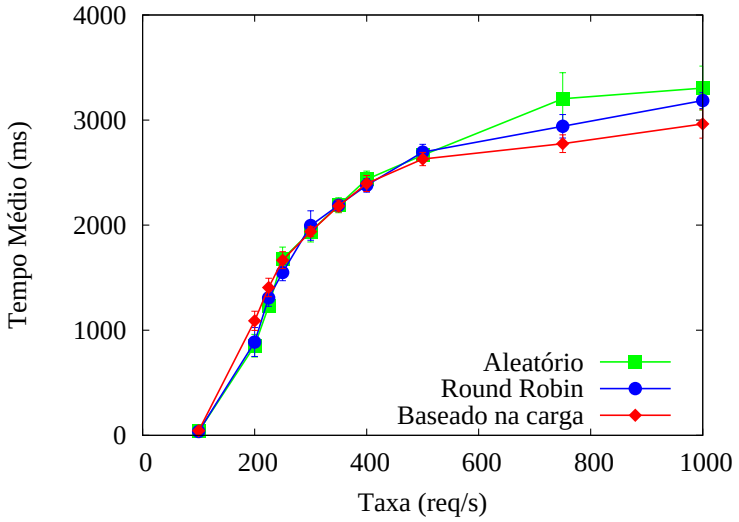

Figura 7 – Tempo médio por taxa para o arquivo HTML

Já para taxas mais elevadas, como, por exemplo, entre 500 e 750 requisições por segundo, há uma leve diferença entre as políticas adotadas. Nesse cenário, a política baseada na carga é estatisticamente diferente da abordagem aleatória. Mais precisamente, para a taxa de 750 req./s, as abordagens baseada na carga, *Round-Robin* e aleatória apresentam tempos médios de resposta de 2775, 2941 e 3202 milissegundos respectivamente. Ou seja, a política baseada na carga passa a ser levemente mais vantajosa que as demais, com tempos médios de resposta menores para esses valores de taxa de chegada. Vale ressaltar que, para o valor de 750 req./s, as recusas de conexão pelo servidor *Web* foram baixas para todas as políticas e variaram entre 0,67% e 1,8%.

Além disso, com uma alta taxa de requisições, a utilização dos servidores pode mudar rapidamente. Infelizmente, o mecanismo Linux que provê estatísticas de carga de um servidor apresenta médias do último minuto avaliado. Assim, um servidor com baixa carga no último minuto, pode receber uma rajada de conexões. Isso faz com que o tempo de resposta médio e a taxa de recusas piorem, se comparados com taxas menores de chegada, para a política baseada na carga. De fato, como observado na Figura 7, para a taxa de 1.000 req./s, a política baseada na carga tem resultados equivalentes às políticas aleatória e *Round-Robin*. Porém, a taxa de recusa de conexões chega a 7,5% para a política baseada na carga dos servidores, contra 0,65% da política *Round-Robin*, por exemplo.

4.4.2 Carga *Web* média

No segundo cenário, os testes foram realizados na simulação de acesso a um arquivo JPG. Novamente, em cada um dos experimentos foram estabelecidas 1000 conexões ao arquivo, com uma requisição em cada uma das conexões.

De acordo com a Figura 8, para uma taxa baixa de requisições, todas as três

políticas possuem resultados equivalentes. Nessa situação de taxas baixas de chegada, os servidores *Web* do sistema possuem recursos suficientes para atender às requisições. Taxas baixas não geram rajadas a um único servidor, o que geralmente pode ocorrer quando utilizadas as políticas aleatória e por carga.

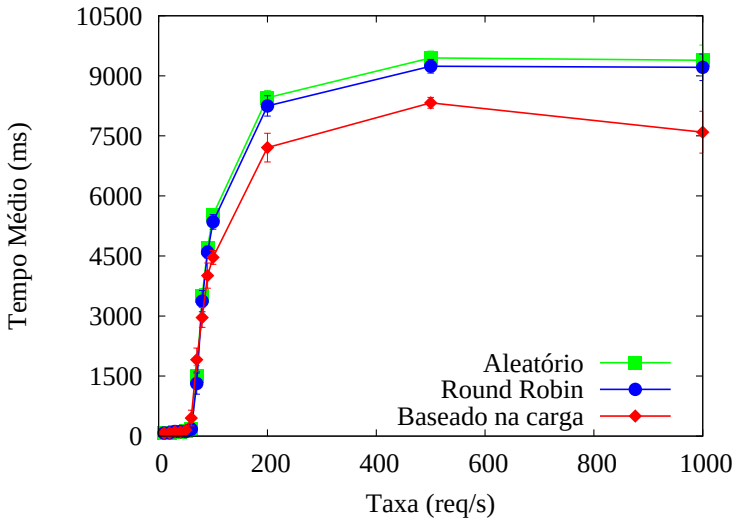

Figura 8 – Tempo médio por taxa para o arquivo JPG

Contudo, para taxas de chegada a partir de 90 req./s, a política baseada na carga passa a ter, visivelmente, um melhor desempenho que as demais. Por exemplo, para uma taxa de 500 req./s, a política baseada na carga é cerca de 12% melhor do que a política aleatória. Mais detalhadamente, a Figura 9 apresenta a distribuição de probabilidade acumulada do tempo de resposta para taxas de chegada de 500 req./s e 20 execuções de experimentos. Por essa figura, fica mais fácil verificar que as políticas *Round-Robin* e aleatória são equivalentes. Entretanto, a política baseada na carga apresenta um desempenho superior, podendo apresentar respostas até 1 segundo mais rápidas que as demais políticas.

Novamente, para taxas de chegada muito altas, foram observadas rajadas de requisições. Isso faz com que os servidores rejeitem novas conexões e aumentem, assim, a taxa de recusa. No cenário atual, o percentual de recusa é consideravelmente maior quando comparado ao primeiro cenário. Por exemplo, para a política baseada na carga, uma taxa de 1000 req./s gerou 13,16% de recusas, enquanto as outras políticas possuíam taxas de erro entre 6 e 7%.

4.4.3 **Carga *Web* pesada**

O terceiro cenário de testes, por sua vez, apresenta a simulação de *downloads* de um arquivo de vídeo de 100 MB, representando um cenário de carga mais pesada a

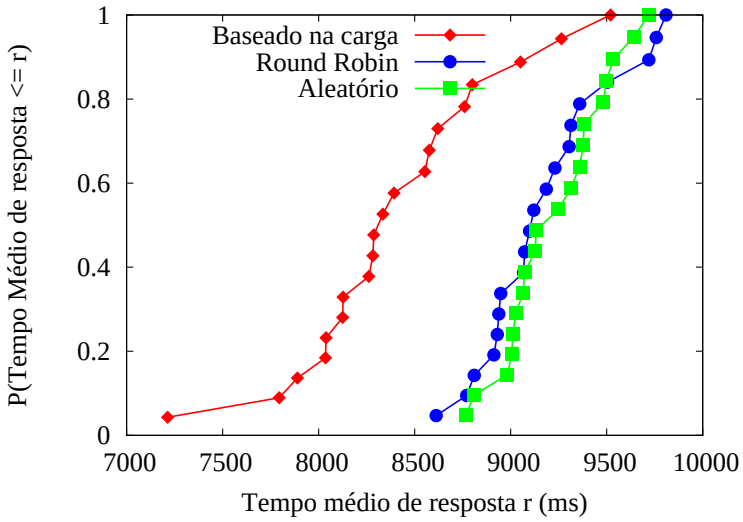

Figura 9 – Tempo de Resposta para Taxas de Chegada de 500 req./s

ser processada pelo sistema de balanceamento. Nesse cenário, foram estabelecidas 100 conexões em cada execução do experimento, com uma requisição para cada conexão.

A Figura 10 apresenta os tempos médios de resposta para taxas de chegada variando de 1 a 100 req./s. Para praticamente todas as taxas avaliadas, as três políticas não são equivalentes. Porém, ao contrário dos outros dois cenários, a política baseada na carga se mostrou como a de pior desempenho nesse cenário.

Nesse cenário, as requisições demoram a ser processadas pelo servidor *Web*. Arquivos grandes são processados em minutos, ao contrário dos poucos segundos observados nas cargas dos dois cenários anteriores. Como os servidores *Web* virtuais utilizados nos experimentos possuem recursos limitados, eles acabam se saturando rapidamente. Como consequência disso, tão logo algumas poucas requisições cheguem aos servidores *Web* e consumam os seus recursos, qualquer tentativa de balanceamento de carga pode se mostrar ineficiente, sobretudo a solução baseada em carga.

Além disso, nesse cenário, a letargia em atualizar o valor da carga média de trabalho da CPU pelo sistema operacional de cada um dos servidores gera resultados ainda piores que o esperado. Como, no início dos experimentos, o servidor que apresenta o menor valor médio de carga de trabalho será escolhido, toda nova requisição será direcionada para esse servidor. Porém, como uma requisição a um arquivo de vídeo demora a ser processada, ela ficará impactando o servidor até que o sistema operacional informe ao controlador da rede uma atualização de sua carga de trabalho. Vale ressaltar que a atualização do valor da carga da CPU é lenta, pois este valor refere-se à média de cargas do último minuto. Como, neste caso, apenas um servidor será responsável por atender a quase todas as requisições, e o outro servidor praticamente não terá nenhuma requisição para atender, os tempos médios de resposta ficarão maiores para esse cenário.

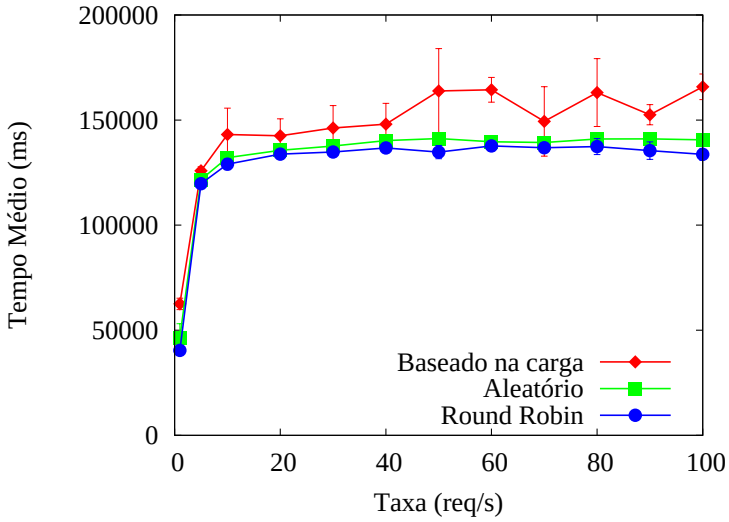

Figura 10 – Tempo médio por taxa para o *download* do arquivo de vídeo

Vale notar que as políticas aleatória e *Round-Robin* apresentaram resultados semelhantes para todos os valores de taxa de chegada. Inclusive, para taxas acima de 10 req./s, os tempos médios de conexão variam entre 130 e 140 segundos para as duas políticas citadas. Já para a política baseada na carga, o problema da atualização da carga média de trabalho da CPU acabou influenciando no aumento do tempo médio de resposta e no tamanho dos intervalos de confiança, fazendo com que, para esse cenário, essa fosse a pior política a ser escolhida. Acredita-se que, com uma nova forma de captura das cargas de trabalho da CPU e máquinas (virtuais ou físicas) com melhores configurações, a política baseada em carga pode sim obter resultados iguais ou superiores às demais políticas de balanceamento.

4.5 CONSIDERAÇÕES GERAIS

Nesta etapa do trabalho foi proposto um mecanismo para o balanceamento de carga utilizando o paradigma SDN, além de ter sido realizada uma avaliação desse mecanismo através de cenários e arquitetura mais realistas. Foram avaliadas três políticas de balanceamento de carga, a saber: uma política aleatória, uma política estilo *Round-Robin* e uma política ciente da carga dos servidores de aplicação. O desempenho de cada uma dessas políticas propostas foi avaliado em três cenários típicos encontrados hoje na *Web*.

No primeiro cenário, equivalente a um cenário *Web* típico onde páginas HTML são requisitadas a um servidor, as três políticas de balanceamento apresentaram desempenhos semelhantes. Isso pode ser explicado pelo fato de que, nesse caso, as requisições consomem poucos recursos computacionais e são atendidas rapidamente. Dessa forma, qualquer servidor com recursos suficientes pode atender as requisições de forma satisfatória,

independente da forma de escolha do servidor que realizará o atendimento, fazendo com que as três políticas de balanceamento tenham desempenhos bem semelhantes. Já para cargas *Web* médias, como fotos, arquivos PDF e arquivos maiores que simples páginas *Web*, a política que leva em consideração a carga apresentou uma melhora de até 12% no tempo de resposta em relação às outras políticas. Em termos de comparação, isso representa respostas aos clientes *Web* até 1 segundo mais rápidas. Surpreendentemente, para cargas pesadas, cada vez mais comuns, a política que leva em consideração a carga apresenta o pior resultado entre as três políticas consideradas. Nesse caso, o tempo de reação requisitado pela aplicação *Web* é muito inferior ao tempo de atualização de dados da carga nos servidores *Web*. Essa letargia por parte do sistema operacional em atualizar o valor de sua carga de CPU faz com que rajadas de requisições sejam encaminhadas a um único servidor, aumentando, por consequência, o tempo médio de resposta das requisições, e prejudicando o desempenho da política baseada em carga. Acredita-se que esse resultado também ocorreu devido à limitação do poder computacional da arquitetura. Pode ser que, com uma nova forma de captura das cargas de trabalho da CPU e máquinas (virtuais ou físicas) com melhores configurações, a política baseada em carga pode sim obter resultados iguais ou superiores às demais políticas de balanceamento nesse cenário de cargas mais pesadas.

A arquitetura proposta pode ter melhores resultados a partir de duas frentes: em primeiro lugar, o esquema de medição de carga disponível no sistema operacional do servidor pode ser refinado, de tal forma que ele se torne mais rápido e reaja a novas cargas tão rápido quanto mudam as condições da rede; e em segundo lugar, podem ser mescladas as políticas de balanceamento de carga e adicionados alguns mecanismos para evitar rajadas.

A evolução natural desse trabalho é implementar a arquitetura proposta inteiramente em um ambiente real, com servidores físicos e hardware switches, e verificar se o balanceamento observado para esse ambiente seria semelhante ao encontrado em ambientes virtualizados, como o reproduzido neste capítulo. Com o desenvolvimento de diversas implementações do protocolo OpenFlow para hardware switches conhecidos, é possível tentar implantar a arquitetura proposta. Contudo, o desempenho de hardwares switches OpenFlow ainda pode não ser o ideal para realizar o balanceamento de carga ou outros serviços de rede. Questões de implementação desses hardware switches como a realização de operações de encaminhamento, o suporte a operações de reescrita de cabeçalho, a quantidade de regras que podem ser armazenadas e o processamento na inserção dessas regras na tabela de fluxos podem ser fatores limitadores de desempenho dos hardware switches em uma arquitetura de balanceamento de carga em SDN com o protocolo OpenFlow. Portanto, antes de proceder a avaliação do mecanismo de balanceamento de carga proposto em uma rede real, deve-se avaliar a qualidade das implementações OpenFlow de hardware switches.

5 AVALIAÇÃO DE DESEMPENHO DE PLANOS DE DADOS OPENFLOW

Neste capítulo, será apresentada a pesquisa relacionada à avaliação de desempenho das implementações OpenFlow de alguns hardware switches comerciais e de software switches *open source*. Todo o processo de pesquisa, bem como a obtenção de equipamentos para os testes, é resultado dos esforços em comum de equipes de pesquisa compostas por membros da Universidade Federal de Juiz de Fora (UFJF), Universidade Federal de Minas Gerais (UFMG) e Universidade Federal de Lavras (UFLA). Inclusive, parte dessa pesquisa e de seus resultados se encontram no artigo de COSTA et al. (2016), submetido e aprovado para o XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2016).

Avaliar o desempenho dos planos de dados OpenFlow é muito importante para o balanceamento de carga em SDN pois várias das características dessas implementações influenciam diretamente o desempenho do serviço de balanceamento. Portanto, avaliar aspectos como o atraso no envio de pacotes, o suporte a uma grande quantidade de regras, o suporte a vários tipos de *match*, o processamento de múltiplos comandos OpenFlow, o suporte à reescrita de cabeçalho, dentre outros, é essencial para verificar a viabilidade de utilização do protocolo OpenFlow para a realização de balanceamento de carga em SDN, sobretudo em redes de produção.

Na seção 5.1 se encontra um breve resumo dos objetivos dessa pesquisa. Na seção 5.2 está descrita toda a metodologia de pesquisa utilizada nesta parte do trabalho, abrangendo o ambiente de experimentação, o funcionamento dos testes, os switches avaliados e os cenários para avaliação de desempenho dos equipamentos. A seção 5.3 concentra os resultados obtidos na realização dos testes para cada um dos cenários propostos. A seção 5.4 apresenta uma análise comparativa desses resultados, ressaltando os pontos positivos e negativos encontrados para as diferentes implementações do protocolo OpenFlow. Por fim, na seção 5.5, é realizado um estudo de viabilidade da utilização dos planos de dados avaliados para a realização do balanceamento de carga.

5.1 OBJETIVOS GERAIS

De um modo sucinto, o objetivo desta parte do trabalho é avaliar a qualidade de diferentes implementações OpenFlow de hardware switches comerciais, além de algumas implementações *open source* de software switches, a fim de introduzir uma discussão sobre a possibilidade ou não de uma adoção efetiva da tecnologia OpenFlow para a realização de balanceamento de carga em redes de produção. De fato, essa avaliação será conduzida com foco voltado ao desempenho exclusivo dos switches (plano de dados), em operações que descartem totalmente a influência do controlador de rede (plano de controle). Para

realizar tal avaliação, foram determinadas algumas métricas de desempenho em operações específicas às quais os switches serão submetidos. Essas métricas levam em consideração aspectos como latência e *jitter* dos pacotes processados pelos switches.

Dessa forma, o presente trabalho tenta avaliar o nível de maturidade no desenvolvimento das implementações OpenFlow. Além disso, procurou-se verificar se é possível identificar, a partir das métricas de desempenho, se as implementações do protocolo OpenFlow são realizadas, de fato, em hardware, ou se são apenas instâncias de um software switch adaptadas ao mecanismo interno do switch. Além da avaliação das implementações OpenFlow, também serão realizadas comparações entre os desempenhos de cada switch operando em seu modo OpenFlow e em seu modo normal (com OpenFlow desabilitado), a fim de identificar se há alguma vantagem ou gargalo na utilização do protocolo.

5.2 METODOLOGIA DE AVALIAÇÃO

Nesta seção, serão descritos os aspectos relacionados à execução dos testes de avaliação de desempenho dos switches OpenFlow. A seguir, estão descritas, dentre outros detalhes, as características da topologia da rede de avaliação, com os papéis de cada elemento na arquitetura SDN; os switches a serem avaliados em seus modos normal e OpenFlow; a caracterização da carga de testes, através da descrição do funcionamento dos experimentos e alguns detalhes dos procedimentos adotados; e a descrição de cinco cenários de teste para uma avaliação mais extensiva dos switches.

5.2.1 **Topologia de rede da avaliação**

Para realizar as avaliações e testes de comparação das diferentes implementações de switches OpenFlow, foi montada uma topologia de rede típica SDN, como a ilustrada na Figura 11. De forma geral, essa topologia é composta por três máquinas interligadas através de um switch OpenFlow. Uma dessas máquinas atua como cliente, enviando requisições; outra atua como servidor, que responde as requisições dos clientes; e a terceira atua como controlador da rede. Essa arquitetura foi projetada de modo que em cada conjunto de testes apenas o switch fosse variado. Ou seja, o ambiente de testes foi replicado na avaliação de cada um dos switches OpenFlow. Nos casos dos testes em que foram utilizados hardware switches, as máquinas eram físicas, e nos casos em que foram utilizados software switches, as máquinas eram virtuais.

Nos casos em que foram utilizadas máquinas físicas, a máquina cliente e o controlador da rede eram computadores com processador Intel Core i7-860 2.80 GHz, configurados com sistema operacional Linux Ubuntu 14.04, com 4 GB de memória. Já o servidor era um computador com processador Intel Core i7-860 2.80 GHz, configurado com sistema operacional Linux Ubuntu Server 12.04, com 4 GB de memória. Já nos casos em que foram utilizadas máquinas virtuais, todas elas possuíam sistema operacional Linux Ubuntu 12.04,

com 1 GB de memória. A precisão do sistema operacional de cada uma das máquinas, virtuais ou físicas, está na casa dos microssegundos.

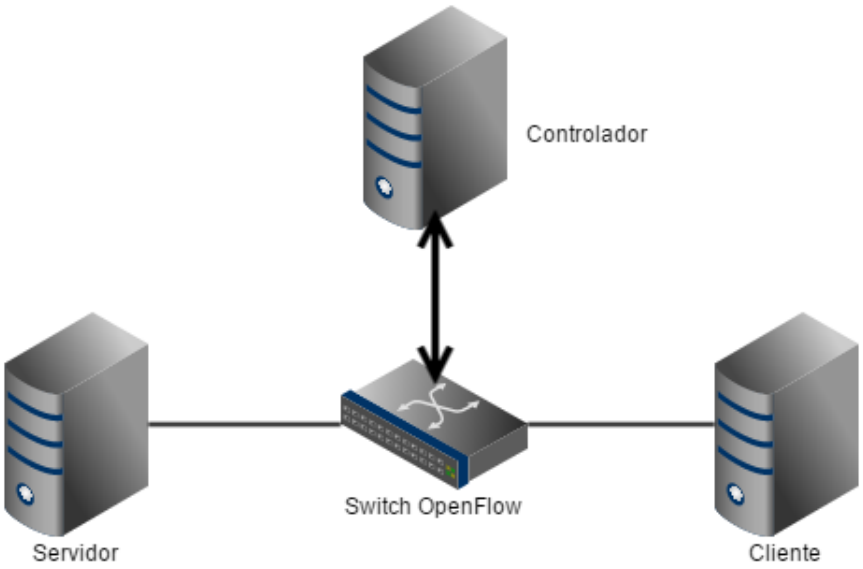

Figura 11 – Arquitetura da rede de avaliação.

Tanto a rede quanto as máquinas utilizadas eram dedicadas exclusivamente ao experimento, de modo que houvesse o mínimo de impacto possível causado tanto pela latência da rede quanto por alguma configuração ou componente das máquinas. Vale ressaltar também que foi disponibilizada a mesma largura de banda em todos os testes. Foram realizados testes prévios com a vazão de cada switch, que apontaram valores máximos de vazão e curvas de desempenho muito semelhantes entre si. Além disso, a vazão também foi controlada por parâmetro nos testes, de tal modo que pode ser descartada qualquer influência da vazão dos switches nesse tipo de teste. Como as aplicações executadas nas máquinas são extremamente leves, controlador, cliente e servidor não operaram em saturação total em momento algum, o que, a princípio, descartaria a influência de picos de CPU, por exemplo.

O papel do controlador nesta arquitetura é o de instalar regras pré-definidas necessárias para a realização do encaminhamento de dados, fato esse que não influencia nenhuma outra operação e nem os resultados dos experimentos. Vale ressaltar que o controlador, neste caso, segue o modelo proativo na instalação das regras da tabela de fluxo. Dessa forma, o código que roda no controlador já possui comandos específicos de instalação de regras de interesse, sem a necessidade de que ocorra algum evento explícito no switch que instale regras dinamicamente. Para instalar as regras OpenFlow de avaliação, foi utilizado o controlador POX (POX, 2015). O controlador instala regras, com hardware timeout ilimitado, no momento da inicialização da rede, de modo que as mesmas fiquem

em atividade desde o momento que o switch é inserido na rede, até o momento em que ele é desconectado. Nos testes com switches operando em modo *legacy*, sem estar ativo o protocolo OpenFlow, o controlador não realiza nenhuma operação.

5.2.2 **Switches avaliados**

Com o surgimento do OpenFlow e sua grande aceitação no mercado, várias empresas passaram a produzir equipamentos com suporte a OpenFlow, seja através de prototipação, de atualização de firmware ou do desenvolvimento direto dessa tecnologia (ROTSOS et al., 2012). Neste trabalho foram avaliadas e comparadas algumas dessas implementações OpenFlow para hardware switches comerciais e software switches conhecidos, considerando a versão 1.0 do OpenFlow. Essa versão foi a escolhida pois é a mais amplamente utilizada e suportada no momento, inclusive dentro do conjunto de switches avaliados. Devido ao alto custo de switches OpenFlow, os testes se limitaram aos equipamentos adquiridos pelas universidades parceiras e pelos laboratórios de pesquisa. A Tabela 1 apresenta os switches avaliados e sumariza suas características.

Marca	**Modelo**	**S.O. Firmware**	**HW / SW Switch**	**Núm. de portas**	**CPU**
Extreme	**Summit x460-24p**	ExtremeXOS 15.4.2.8	HW	28	Single Core CPU 500 MHz
Extreme	**Summit x440-48p**	ExtremeXOS 15.4.2.8	HW	52	Single Core CPU 500 MHz
HP	**HP2920-24G**	Firmware K 15.5 i	HW	24	Tri Core ARM1176 625 MHz
LinkSys	**WRT54GL**	OpenWRT Pantou	SW	4	Broadcom BCM5352 200 MHz
Open vSwitch	**OvS 2.3.0**	Linux Ubuntu 14.04	SW	-	Intel Core i7 CPU 2.80 GHz
Mikrotik	**Atheros AR9344**	RouterOS 6.34.2	HW	24	Single Core CPU 600 MHz
NetFPGA	**FPGA Xilinx Virtex-II Pro 50**	Linux CentOS 5.11	HW	4	AMD Athlon II X4 800 MHz
Datacom	**DM4001**	Datacom Flash 2.22	HW	26	PowerPC e500 990 MHz

Tabela 1 – Switches avaliados no trabalho.

Na lista se encontram oito switches com suporte à utilização do protocolo OpenFlow. Desses equipamentos, seis são hardware switches - Extreme Summit x460, Extreme Summit x440, HP, Mikrotik, Datacom e uma placa NetFPGA, esta última uma plataforma aberta para pesquisa, experimentação e prototipação de dispositivos de rede - e dois são implementações de software switches em diferentes plataformas - o Open vSwitch, que é a implementação de um switch virtual multicamadas bastante conhecido e utilizado; e um roteador sem fio LinkSys previamente instalado com o firmware OpenWRT, uma distribuição Linux para dispositivos embarcados.

A quantidade de equipamentos avaliados não cobre a diversidade de implementações OpenFlow existentes no mercado, contudo, o objetivo do trabalho não é realizar uma avaliação geral ou quantitativa da tecnologia OpenFlow. Esse subconjunto de equipamentos avaliados é composto por switches de relevância, de naturezas bem distintas, com implementações em hardware e software, incluindo até plataformas de experimentação. Devido a isso, esse subconjunto pode servir como um parâmetro para a análise do desenvolvimento e da evolução da tecnologia OpenFlow no mercado de equipamento de redes.

5.2.3 **Caracterização da carga de testes**

De forma sucinta, os testes consistem no envio de pacotes UDP entre a máquina cliente e o servidor através do switch OpenFlow a ser avaliado. Para realizar os testes referentes aos quatro primeiros cenários, foi implementado e utilizado um gerador de tráfego próprio em linguagem C, composto por dois módulos: um módulo instalado na máquina cliente e outro instalado no servidor. O módulo instalado no cliente envia os pacotes UDP para o servidor, com o valor do *timestamp* de envio do pacote armazenado em uma variável. O módulo executado no servidor, ao receber os pacotes, os retorna imediatamente para o cliente. O módulo cliente, ao receber a resposta, calcula o Round Trip Time (RTT) de cada um dos pacotes utilizando os *timestamps* de envio e de chegada. A partir disso, o módulo cliente pode computar as métricas de interesse, como atraso e *jitter* dos pacotes. Vale ressaltar que a precisão dos *timestamps* está na casa dos microssegundos. Para avaliar o desempenho dos switches com diferentes *overheads* de processamento de dados, o tamanho dos pacotes UDP foi variado entre 64, 128 e 256 bytes. Esses valores foram escolhidos com base nos trabalhos relacionados e na RFC 2544.

Vale frisar que foram utilizados pacotes UDP nos testes devido à facilidade de manipulação desses pacotes e, principalmente, pelo fato de o protocolo TCP utilizar alguns mecanismos internos, como o controle de congestionamento e o controle de fluxo, que acabam influenciando a medição e o envio dos pacotes, o que inviabilizaria o uso do TCP nos experimentos.

Todos os testes referentes aos quatro primeiros cenários foram baseados no envio de 10.000 pacotes UDP do cliente para o servidor através do switch OpenFlow avaliado. A taxa de envio de pacotes é de 10 pacotes por segundo. Os resultados obtidos são compostos pelos valores dos atrasos individuais de cada pacote e os *jitters* correspondentes, além do cálculo das médias e desvios padrões dos conjuntos de atrasos e *jitters*, e os cálculos dos intervalos de confiança desses conjuntos. Neste trabalho, foram utilizados intervalos de confiança de 99% para uma distribuição normal de amostras.

Já os testes referentes ao quinto cenário possuem natureza e execução diferentes, embora também avaliem o atraso de pacotes. Maiores detalhes sobre esse cenário serão explicitados nas subseções correspondentes.

5.2.4 **Cenários de teste**

Foram definidos cinco cenários para a avaliação de desempenho das implementações OpenFlow de cada switch. Para cada um deles, foi elaborado um tipo de teste diferente, de acordo com as especificidades de cada caso. Características como a quantidade de regras instaladas, os tipos de *match* (casamento) de regras a serem realizados, os modos de operação do switch e o tamanho dos pacotes enviados foram consideradas nos experimentos. A metodologia de realização dos testes e de coleta de resultados, explicitada na seção anterior, funciona igualmente para quase todos os cenários, excetuando-se o caso do quinto cenário. A diferença entre os testes se dá, portanto, na natureza dos diferentes tipos de regras a serem instaladas no controlador em cada teste, e nos diferentes modos de operação dos switches avaliados. Nas subseções a seguir serão descritos os cenários para a avaliação das implementações.

5.2.4.1 Cenário 1: Desempenho em diferentes tipos de *match*

O primeiro cenário definido avalia o desempenho do switch na realização de diferentes tipos de *match* (ou casamento). Foram definidos cinco tipos genéricos de *match*, compostos por subconjuntos de *matches* individuais formados dentre os 12 atributos da versão 1.0 do OpenFlow. A Tabela 2 define quais são os tipos genéricos de *match* e seus respectivos atributos individuais sobre os quais serão realizados os *matches*. O objetivo desse teste é verificar se o tempo de execução dos *matches* é influenciado pela quantidade ou tipo dos atributos individuais, e, principalmente, analisar a diferença do desempenho dos mesmos tipos de *match* em diferentes switches.

5.2.4.2 Cenário 2: Desempenho dos switches nos modos *legacy* e OpenFlow

O segundo cenário definido compara os desempenhos de um determinado switch quando operando nos modos OpenFlow e normal (*legacy*). Serão comparados os desempenhos da operação de encaminhamento dos pacotes entre a porta de chegada e a porta de saída do switch nos dois modos de operação, com o objetivo de verificar se existe alguma diferença substancial de desempenho entre os modos. Além disso, será discutido se o desempenho das implementações OpenFlow já é aceitável o suficiente para a adoção da tecnologia em uma escala maior.

Tipo de Match	Campos de Match											
	Porta de Entrada	**End. MAC Origem**	**End. MAC Destino**	**Ethertype**	**ID da VLAN**	**Prioridade da VLAN**	**End. IP Origem**	**End. IP Destino**	**Protocolo**	**IP ToS**	**Porta UDP Origem**	**Porta UDP Destino**
Porta	x											
MAC		x	x									
IP				x			x	x				
Porta UDP				x			x	x	x		x	x
Exato	x	x	x	x	x	x	x	x	x	x	x	x

Tabela 2 – Subconjuntos de atributos utilizados para *matches*.

5.2.4.3 Cenário 3: Desempenho com uma ou múltiplas regras na tabela de fluxos

O terceiro cenário definido avalia o desempenho do switch em duas situações: *a)* quando a tabela de fluxos possui apenas uma única regra, e que esta necessariamente irá retornar um *match*; e *b)* quando a tabela está completamente preenchida por múltiplas regras, mas apenas uma delas retorna um *match*. O objetivo é verificar se o desempenho pode ser influenciado pela quantidade de regras instaladas na tabela, pela disposição das regras na tabela de fluxos ou pelo processamento interno de cada switch.

5.2.4.4 Cenário 4: Desempenho na modificação de campos do cabeçalho dos pacotes

O quarto cenário avalia o desempenho de cada switch em modificações de campos específicos do cabeçalho de cada pacote. No teste desse caso, foram realizadas operações de modificação em campos do cabeçalho dos pacotes como endereços MAC e IP de destino, prioridade de VLAN, IP *Type of Service* e porta UDP, com o objetivo de comparar essas operações nos diferentes switches, além de verificar o efetivo suporte a essas operações em cada um dos switches avaliados.

5.2.4.5 Cenário 5: Desempenho nas operações de *flowstats* e *portstats*

O quinto e último cenário avalia o desempenho de cada switch nas operações de *flowstats* e *portstats*. Nos testes desse cenário, o controlador requisita estatísticas (*flowstats* ou *portstats*) ao switch e calcula o atraso de cada resposta enviada por ele. Durante esses testes, o switch é submetido a variações na carga de trabalho e na quantidade de regras instaladas na tabela de fluxos, com o objetivo de verificar a influência dessas variações no envio das mensagens estatísticas.

5.3 RESULTADOS DOS EXPERIMENTOS

5.3.1 **Desempenho em diferentes tipos de *match***

O primeiro cenário definido avalia o desempenho do switch na realização de diferentes tipos de *match*. Como já dito, foram definidos cinco tipos genéricos de *match*, compostos por subconjuntos de *matches* individuais apresentados na seção anterior. A saber, os tipos de *match* definidos foram: *match* por porta, por endereço MAC, por endereço IP, por porta UDP e um tipo de *match* denominado exato. Com exceção do *match* exato, que utiliza todos os atributos existentes na versão 1.0 do OpenFlow para a realização de *matches*, os outros tipos utilizam uma parte dos atributos disponíveis. A ideia por trás dessa classificação de diferentes tipos de *match* é verificar o desempenho dos switches nas diferentes operações de *match* e se a quantidade e tipos dos atributos influenciam no atraso dos pacotes.

Switch	Tipos de Match				
	Porta	**MAC**	**IP**	**Porta UDP**	**Exato**
Extreme x460-24p	0.125 – 0.127	0.124 – 0.126	0.124 – 0.126	0.124 – 0.126	0.121 – 0.125
Extreme x440-48p	0.174 – 0.175	0.131 – 0.133	0.130 – 0.132	0.119 – 0.121	0.117 – 0.119
HP	1.069 – 1.147	1.049 – 1.113	1.064 – 1.309	1.038 – 1.106	0.743 – 1.001
LinkSys - OpenWRT	1.025 – 1.051	1.017 – 1.045	1.019 – 1.049	1.017 – 1.046	1.018 – 1.047
Open vSwitch	0.514 - 0.519	0.502 - 0.508	0.504 - 0.508	0.513 - 0.517	0.508 - 0.511
Mikrotik	0.248 – 0.248	0.248 – 0.249	0.251 – 0.253	0.248 – 0.249	0.243 – 0.243
NetFPGA	0.201 – 0.202	0.197 – 0.198	0.196 – 0.202	0.201 – 0.201	0.194 – 0.197
Datacom	0.198 - 0.200	0.201 - 0.202	0.199 - 0.200	0.199 - 0.200	0.199 - 0.200

Tabela 3 – Intervalo de confiança do atraso médio para diferentes tipos de *match* em pacotes de 64 bytes (em ms).

Os testes desse cenário consistiram na execução das cinco operações de *match* em cada um dos oito switches avaliados, para três tamanhos diferentes de pacotes UDP (64, 128 e 256 bytes), gerando um total de 120 experimentos neste cenário. Como já mencionado, em cada um dos experimentos, foram enviados 10.000 pacotes UDP através dos switches avaliados.

Na Tabela 3 estão listados os intervalos de confiança dos atrasos médios de pacotes de 64 bytes, obtidos na realização de cada um dos diferentes tipos de *match*, em cada um dos switches. Analisando a tabela horizontalmente, pode-se descobrir o comportamento dos diferentes tipos de *match* em um mesmo switch. É possível verificar que em alguns switches, como o Extreme x460 e o LinkSys, as diferentes operações de *match* dentro de cada um deles possuem intervalos de atraso médio muito próximos um dos outros, havendo, inclusive, interseção entre todos esses intervalos. E até nos casos do Open vSwitch, Mikrotik, Datacom e NetFPGA, onde nem todos os intervalos se interceptam, as diferenças entre eles são mínimas, na casa de poucos microssegundos. Isso leva a crer que o tipo de atributo ou a quantidade de atributos utilizados para realizar um *match* não afetam substancialmente o atraso dos pacotes, quando são comparados os diferentes tipos de *match*. Inclusive, nos switches citados anteriormente, o *match* exato, que é justamente o tipo que utiliza mais atributos para realizar um *match*, aparece como a operação de melhor desempenho, mesmo que de forma bem sutil.

No entanto, outras duas situações chamam a atenção. A primeira delas se refere ao desempenho dos *matches* no switch Extreme x440. Nesse caso, o desempenho do *match* por porta é sensivelmente pior do que os demais, chegando a ser 40 microssegundos mais lento que as operações de *match* por IP e por MAC, e a 50 microssegundos mais lento do que os *matches* por porta UDP e exato. Mesmo assim, nesse caso, a quantidade de atributos parece não afetar proporcionalmente o desempenho dos *matches*, uma vez que as operações de *match* com mais atributos estão sendo feitas de forma mais rápida.

A outra situação merece um destaque ainda maior. Para o switch HP, os desempenhos dos tipos de *match* por porta, IP, MAC e porta UDP são bem próximos entre si,

com intervalos de atraso coincidentes. Contudo, na realização do *match* exato, o intervalo de confiança do atraso médio não ficou próximo aos dos outros tipos de *match*. O tempo de atraso médio para o *match* exato foi destacadamente menor quando comparado aos demais *matches*, chegando a mais de 0.2 milissegundos de diferença em alguns dos casos. Ainda nesta seção, será apresentada uma possível explicação para essa diferença entre o *match* exato e os demais no switch HP.

Analisando a tabela verticalmente, pode-se comparar o desempenho de cada switch na realização de um tipo *match* específico. Nessa comparação, foram encontradas substanciais diferenças entre os desempenhos dos switches quando analisados os intervalos dos atrasos médios. Para os tipos de *match* por porta, MAC, IP e porta UDP, podem ser notados três comportamentos distintos de desempenho dos switches, de uma forma geral, considerando o valor dos intervalos. Os switches Extreme (x460 e x440) apresentam os menores valores de atraso médio dentre todos os switches comparados, com aproximadamente 0.120 milissegundos de atraso médio. Com atraso médio próximo a 0.200 milissegundos, a placa NetFPGA e o switch Datacom também apresentaram um bom desempenho, assim como o switch Mikrotik, que possui um atraso médio de aproximadamente 0.250 milissegundos. Vale ressaltar que esses cinco switches já citados, com os melhores desempenhos, são hardware switches.

Por sua vez, a implementação do Open vSwitch apresenta um valor médio de atraso um pouco maior que os anteriores, beirando 0.5 milissegundos, o que representaria um desempenho mediano em comparação com os demais. Já o switch HP e o roteador LinkSys operando via OpenWRT apresentaram os piores desempenhos, com os intervalos de confiança do atraso médio maiores que 1.0 milissegundo, o que significaria um desempenho duas vezes pior que o Open vSwitch, ou cerca de oito vezes pior que os switches Extreme, quando considerados os apenas os valores médios de atraso dos pacotes.

Para ilustrar melhor a situação e para a análise não se restringir apenas aos valores médios, serão apresentados, a seguir, os gráficos com as distribuições cumulativas dos atrasos dos pacotes de 64 bytes em cada switch, para os cinco tipos de *match*. Para as distribuições dos *matches* por porta (Figura 12), MAC (Figura 13), IP (Figura 14) e porta UDP (Figura 15), o comportamento das curvas de cada switch é bem semelhante, independente do tipo de *match* realizado. Nesses gráficos, é possível notar a diferença entre três grupos de curvas, que se relacionam bastante com os grupos de dados apresentados na Tabela 3. Localizadas mais a esquerda dos gráficos citados anteriormente, estão as curvas referentes aos switches Extreme (x460 e x440), Datacom, Mikrotik e à placa NetFPGA, os equipamentos de melhor desempenho, cujos atrasos dos pacotes variam entre 0.1 e 0.25 milissegundos para quase 100% dos pacotes enviados. Já a curva referente ao Open vSwitch está mais centralizada, com a maioria dos atrasos localizados em torno dos 0.5 milissegundos.

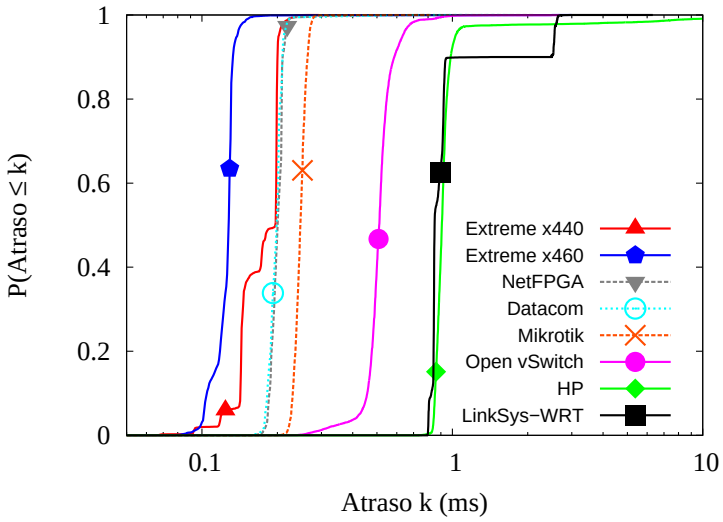

Figura 12 – Distribuições cumulativas dos atrasos para o tipo de *match* por porta (pacotes de 64 bytes).

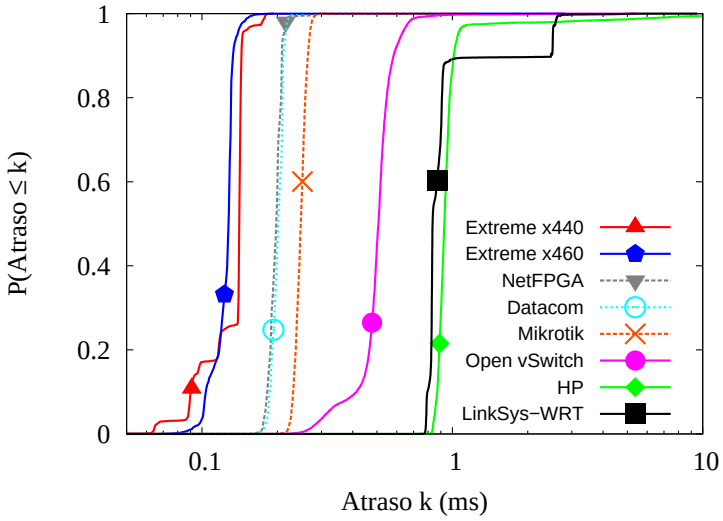

Figura 13 – Distribuições cumulativas dos atrasos para o tipo de *match* por MAC (pacotes de 64 bytes).

Por sua vez, as curvas referentes ao switch HP e ao roteador LinkSys OpenWRT se destacam mais à direita, com distribuições bem semelhantes em sua maior parte, porém cabem alguns comentários sobre o desempenho desses dois switches. Cerca de 90% dos atrasos no LinkSys variam entre 0.8 e 0.9 milissegundos. Contudo os outros 10% são de valores acima de 2.5 milissegundos, o que acabou elevando a média do atraso nos quatro tipos de *match* referidos (média em torno de 1.05 milissegundos). Ao se investigar os *logs* relacionados aos atrasos do roteador LinkSys, foi observado que a cada dez pacotes enviados em sequência, o primeiro deles apresentava um atraso maior, enquanto os nove seguintes apresentavam atrasos menores (entre 0.8 e 0.9 milissegundos). Isso deve ocorrer

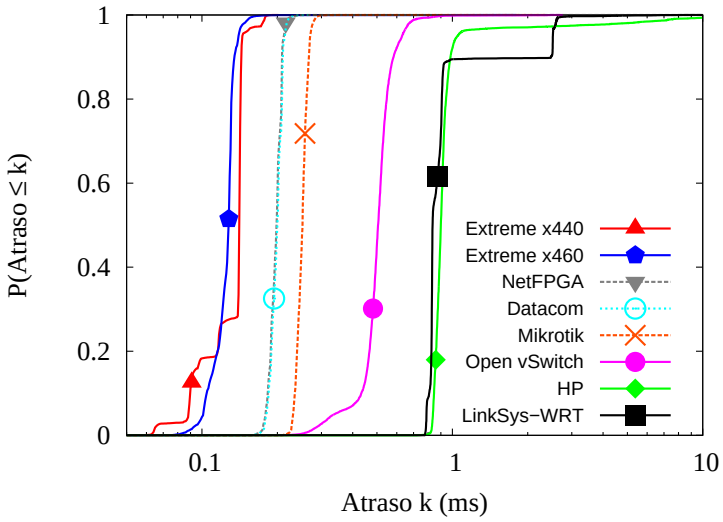

Figura 14 – Distribuições cumulativas dos atrasos para o tipo de *match* por IP (pacotes de 64 bytes).

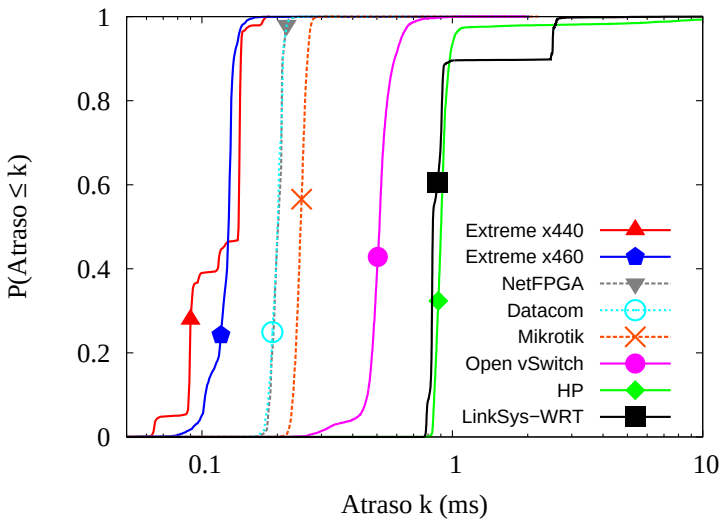

Figura 15 – Distribuições cumulativas dos atrasos para o tipo de *match* por porta UDP (pacotes de 64 bytes).

devido algum mecanismo interno do OpenWRT para o encaminhamento de pacotes. Já para o switch HP, cerca de 95% dos atrasos dos pacotes variam entre 0.8 e 1.0 milissegundo. Contudo, a média do atraso também ficou mais elevada devido à influência dos outros 5% dos pacotes, que apresentaram atrasos extremamente altos, entre 1 e 20 milissegundos (não ilustrados nos gráficos).

Entretanto, ao se analisar o gráfico com as distribuições cumulativas dos atrasos dos pacotes de 64 bytes para o *match* exato (Figura 16), nota-se uma pequena diferença no comportamento de uma das curvas, quando comparada a dos outros tipos de *match*. Como dito anteriormente, o switch HP possui um atraso médio notadamente menor para

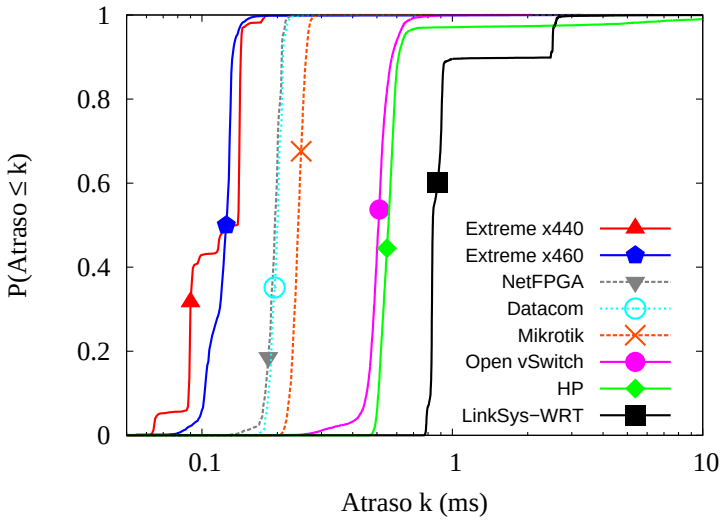

Figura 16 – Distribuições cumulativas dos atrasos para o tipo de *match* exato (pacotes de 64 bytes).

o *match* exato quando comparado ao atraso dos outros tipos de *match*. Observando esse gráfico, nota-se que a curva correspondente ao switch HP fica deslocada mais à esquerda, se aproximando da curva do Open vSwitch. Isso significa que os tempos de atraso dos pacotes nesse *match*, para o switch HP, são relativamente menores quando comparados aos atrasos nos outros tipos de *match* no mesmo switch HP, em cerca de 25%. Na distribuição, cerca de 95% dos pacotes possui atrasos entre 0.5 e 0.7 milissegundos. Isso se deve a algum mecanismo do switch HP que realiza *matches* exatos possivelmente em hardware, de forma mais rápida, ao contrário dos *matches* não exatos, que devem ser realizados em software, de forma mais lenta. Esse comportamento é típico de software switches, contudo ocorreu em um hardware switch HP, o que levanta suspeitas sobre a natureza da implementação OpenFlow para esse switch – se foi implantada em hardware de fato ou se é uma adaptação de um software switch ao mecanismo interno do hardware switch. O próprio desempenho do switch HP, ilustrado pela curva correspondente nos cinco gráficos, pode denunciar uma implementação em software do protocolo OpenFlow, uma vez que essas curvas se localizam muito próximas às curvas de outros software switches como o Open vSwitch e o OpenWRT instalado no LinkSys, e mais afastadas das curvas dos hardware switches.

Ainda para o *match* exato, os switches Extreme (x460 e x440), Datacom e Mikrotik, e a placa NetFPGA continuaram como os mais rápidos, e a implementação OpenWRT do LinkSys como o mais lento dos switches analisados.

Para testar a influência do tamanho dos pacotes na realização dos *matches*, foram realizados os mesmos testes variando-se o parâmetro de tamanho dos pacotes UDP para 128 e 256 bytes. A ideia, nesse caso, é verificar se existe algum *overhead* substancial no processamento dos pacotes e seu eventual reflexo nos atrasos e no desempenho dos *matches*

Switch	Tipos de Match				
	Porta	**MAC**	**IP**	**Porta UDP**	**Exato**
Extreme x460-24p	0.192 – 0.194	0.192 – 0.194	0.195 – 0.197	0.195 – 0.197	0.198 – 0.200
Extreme x440-48p	0.175 – 0.176	0.133 – 0.134	0.129 – 0.131	0.124 – 0.126	0.121 – 0.122
HP	1.156 – 1.255	1.172 – 1.233	1.171 – 1.247	1.210 – 1.480	0.803 – 0.872
LinkSys - OpenWRT	1.043 – 1.069	1.037 – 1.065	1.033 – 1.061	1.038 – 1.067	1.035 – 1.065
Open vSwitch	0.501 - 0.505	0.501 - 0.505	0.501 - 0.504	0.503 - 0.506	0.509 - 0.512
Mikrotik	0.344 – 0.345	0.345 – 0.346	0.345 – 0.346	0.345 – 0.345	0.340 – 0.341
NetFPGA	0.292 – 0.293	0.292 – 0.293	0.292 – 0.292	0.292 – 0.293	0.289 – 0.290
Datacom	0.293 - 0.297	0.294 - 0.299	0.294 - 0.295	0.294 - 0.295	0.294 - 0.295

Tabela 4 – Intervalo de confiança do atraso médio para diferentes tipos de *match* em pacotes de 128 bytes (em ms).

em cada switch. A Tabela 4 lista os intervalos de confiança dos atrasos médios de pacotes de 128 bytes, obtidos na realização de cada um dos diferentes tipos de *match*, em cada um dos switches.

Ao analisar essa tabela, pode-se observar algumas características idênticas às da tabela de atrasos de 64 bytes. Os atrasos médios entre os diferentes tipos de *match* para um mesmo switch se mantiveram próximos um dos outros ou, pelo menos, seguiram os mesmos padrões da tabela anterior. O comportamento do atraso para o *match* exato no switch HP também foi o mesmo, notadamente menor do que o atraso dos outros tipos de *match* nesse switch. Entretanto, quando são comparadas as Tabelas 3 e 4 pode-se observar que, em alguns switches, houve um aumento maior nos tempos médios de atraso. Ou seja, quando foi dobrado o tamanho dos pacotes UDP, de 64 para 128 bytes, alguns dos switches aumentaram substancialmente as suas taxas de atraso. É o caso dos switches HP, Extreme x460, Mikrotik, Datacom e NetFPGA, que apresentaram aumentos significativos. O switch Extreme x460 teve um aumento médio de 70 microssegundos nos atrasos dos pacotes, enquanto os switches HP, Mikrotik, Datacom e NetFPGA tiveram um aumento de 100 microssegundos cada um, em seus respectivos atrasos. Os demais switches (Extreme x440, LinkSys-OpenWRT e Open vSwitch) apresentaram aumentos marginais nos atrasos médios.

As situações descritas também podem ser notadas nas distribuições cumulativas dos atrasos dos diferentes tipos de *match* para os pacotes de 128 bytes. O gráfico das distribuições no *match* por porta (Figura 17) ainda mostra uma certa semelhança entre as curvas dos dois switches Extreme, mas nas distribuições dos *matches* por MAC (Figura 18), por IP (Figura 19) e por porta UDP (Figura 20), já é possível notar visualmente o aumento dos atrasos dos pacotes nos switches Extreme x460, Mikrotik, Datacom, NetFPGA e HP, quando comparados aos atrasos ilustrados nos gráficos referentes aos pacotes de 64 bytes. O gráfico das distribuições para o *match* exato (Figura 21) ainda ilustra a típica melhora de desempenho do switch HP em *matches* exatos, discutida anteriormente. Contudo, neste caso, o desempenho do switch HP não ficou tão próximo ao do Open vSwitch, piorando

em torno de 0.2 milissegundos.

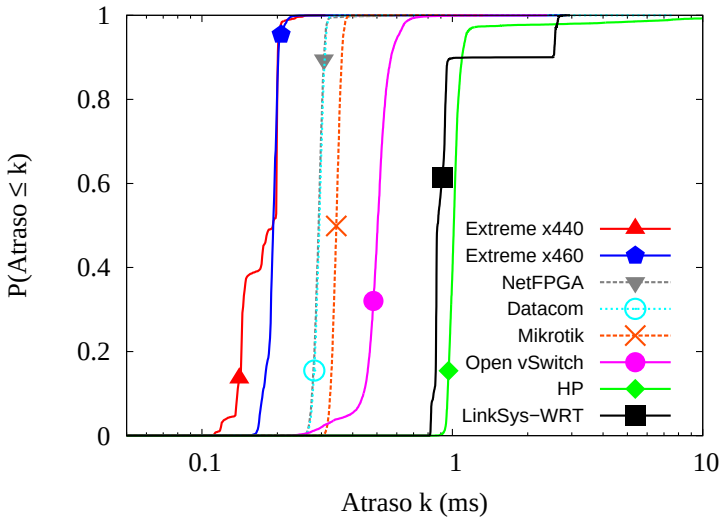

Figura 17 – Distribuições cumulativas dos atrasos para o tipo de *match* por porta (pacotes de 128 bytes).

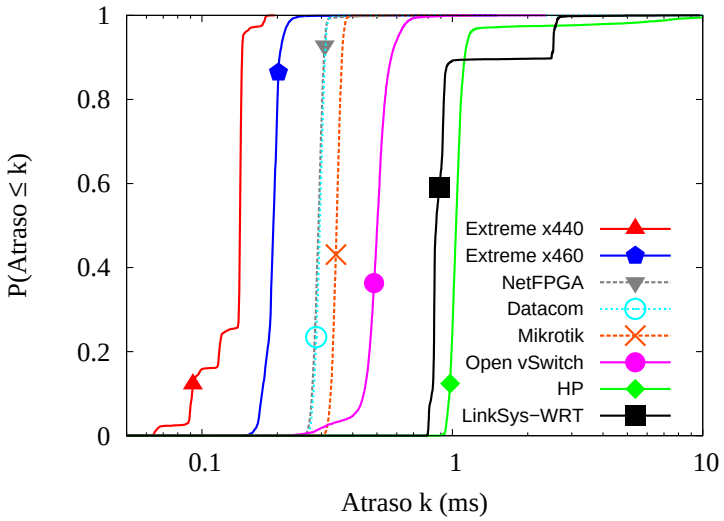

Figura 18 – Distribuições cumulativas dos atrasos para o tipo de *match* por MAC (pacotes de 128 bytes).

Dessa forma, um dos fatos mais relevantes ocorridos com o aumento do tamanho do pacote básico UDP, de 64 para 128 bytes, foi o aumento substancial do atraso apenas para alguns switches específicos. Vale ressaltar que esse aumento foi observado apenas para hardware switches. Isso pode indicar que algumas características físicas internas dos switches e da rede podem influenciar o desempenho quando da utilização do protocolo OpenFlow. Outro fato que chamou a atenção foi a diferenciação do atraso entre os dois modelos Extreme. Apesar de ambos possuírem a mesma versão de firmware com a implementação do protocolo OpenFlow habilitada, houve certa diferença entre os

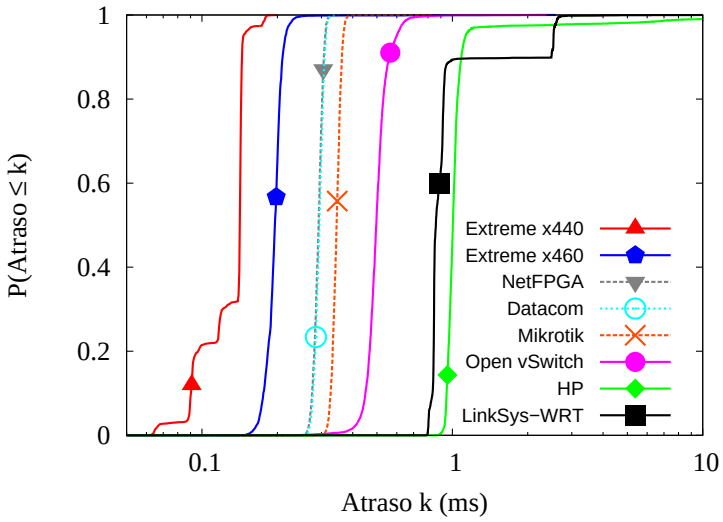

Figura 19 – Distribuições cumulativas dos atrasos para o tipo de *match* por IP (pacotes de 128 bytes).

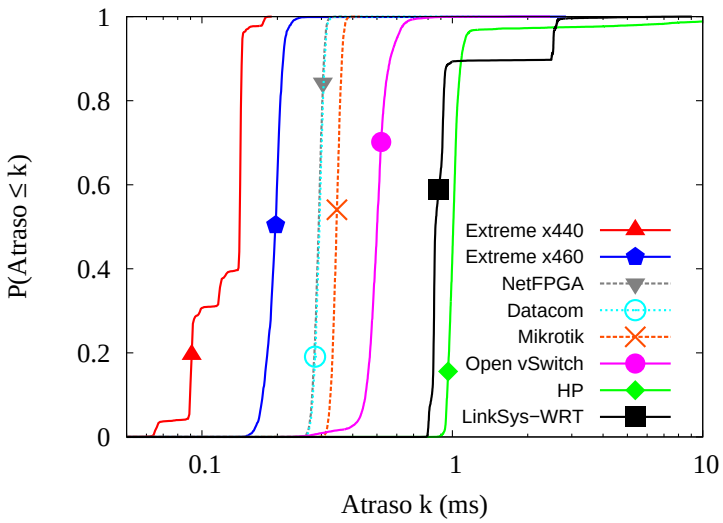

Figura 20 – Distribuições cumulativas dos atrasos para o tipo de *match* por porta UDP (pacotes de 128 bytes).

desempenhos com o aumento do tamanho do pacote UDP. Esse fato, de certa forma, pode justificar a realização de testes com modelos diferentes de equipamentos de um mesmo fabricante. Um último fato a ser destacado se refere à aproximação dos atrasos dos hardware switches e da placa NetFPGA ao atraso do Open vSwitch.

Por sua vez, a Tabela 5 mostra os intervalos de confiança dos atrasos médios de pacotes de 256 bytes, obtidos na realização de cada um dos diferentes tipos de *match*, em cada um dos switches. O que pode ser notado a partir da análise da tabela é que, apesar do aumento do tamanho do pacote UDP dos testes, de 128 para 256 bytes, não houve aumentos significativos nos tempos médios de atraso. Os gráficos das distribuições

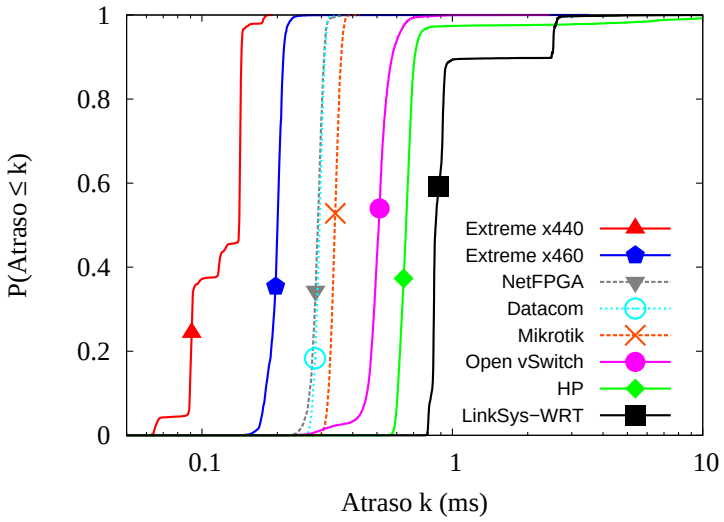

Figura 21 – Distribuições cumulativas dos atrasos para o tipo de *match* exato (pacotes de 128 bytes).

Switch	**Tipos de Match**				
	Porta	**MAC**	**IP**	**Porta UDP**	**Exato**
Extreme x460-24p	0.200 – 0.202	0.202 – 0.204	0.202 – 0.204	0.200 – 0.202	0.198 – 0.200
Extreme x440-48p	0.166 – 0.167	0.133 – 0.134	0.125 – 0.126	0.124 – 0.126	0.118 – 0.119
HP	1.159 – 1.227	1.124 – 1.179	1.153 – 1.225	1.103 – 1.156	0.819 – 0.889
LinkSys - OpenWRT	1.073 – 1.099	1.065 – 1.093	1.067 – 1.095	1.066 – 1.095	1.073 – 1.102
Open vSwitch	0.499 - 0.503	0.505 - 0.508	0.501 - 0.505	0.507 - 0.510	0.506 - 0.509
Mikrotik	0.348 – 0.350	0.348 – 0.351	0.351 – 0.352	0.351 – 0.352	0.342 – 0.347
NetFPGA	0.294 – 0.294	0.297 – 0.298	0.294 – 0.295	0.292 – 0.298	0.292 – 0.295
Datacom	0.297 - 0.298	0.296 - 0.296	0.296 - 0.297	0.295 - 0.296	0.296 - 0.297

Tabela 5 – Intervalo de confiança do atraso médio para diferentes tipos de *match* em pacotes de 256 bytes (em ms).

cumulativas dos atrasos também refletem essa situação. A título de exemplo, os gráficos referentes aos *matches* por IP (Figura 22) e exato (Figura 23) para pacotes de 256 bytes são idênticos aos gráficos correspondentes de 128 bytes.

Analisando os atrasos obtidos para os diferentes tamanhos de pacotes testados (64, 128 e 256 bytes), pode-se sugerir que o tamanho dos pacotes pode acarretar, de fato, *overheads* de processamento na realização de *matches* em casos de switches específicos. Contudo, verificou-se que para um determinado ponto (256 bytes), esses overheads já não eram mais observados.

Outra métrica utilizada para avaliar o desempenho dos switches é o *jitter*. O *jitter* é importante pois mede a variação do atraso entre chegadas de pacotes sucessivos ao longo do tempo em uma rede. Portanto, quanto maior o *jitter*, mais irregular é a chegada dos pacotes em um dispositivo da rede, o que pode indicar problemas de desempenho por parte dos comutadores. Foi calculado o *jitter* de cada par de pacotes imediatamente sucessivos (pacotes X e X+1) que chegavam após serem enviados através dos switches

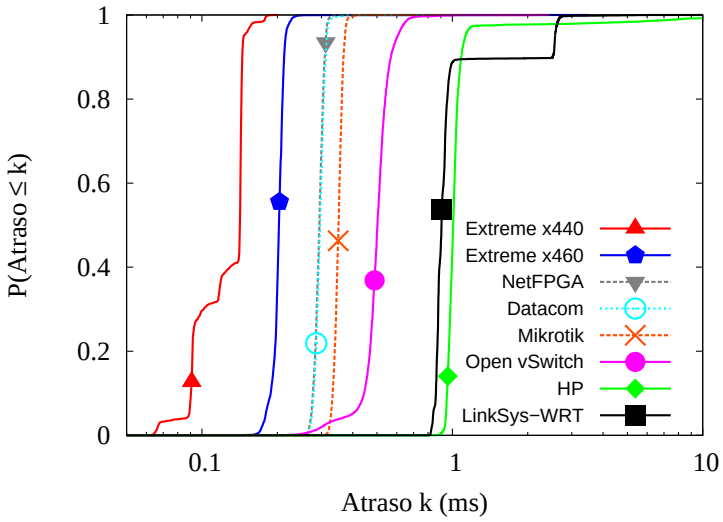

Figura 22 – Distribuições cumulativas dos atrasos para o tipo de *match* por IP (pacotes de 256 bytes).

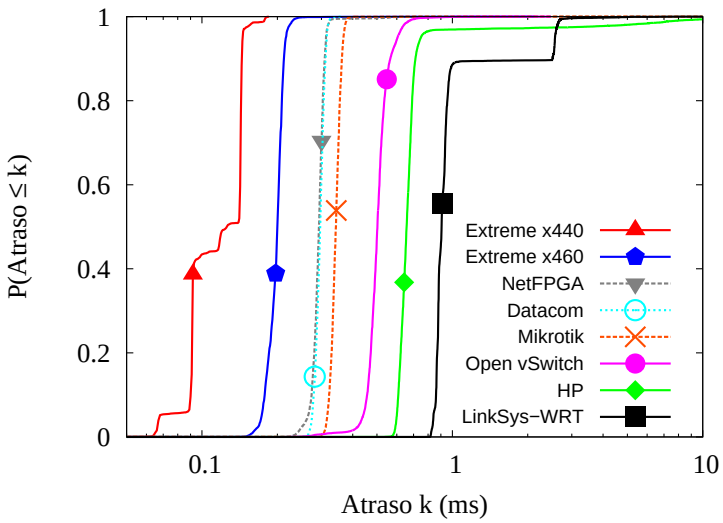

Figura 23 – Distribuições cumulativas dos atrasos para o tipo de *match* exato (pacotes de 256 bytes).

avaliados. Dessa forma, par a par de pacotes sucessivos, foram calculados 10.000 jitters de pacotes enviados. O gráfico da Figura 24 mostra a distribuição cumulativa dos jitters dos pacotes de 64 bytes em *matches* exatos. Analisando as curvas do gráfico, pode-se observar que os jitters dos switches Extreme (x460 e x440), Mikrotik, Datacom e a placa NetFPGA foram notoriamente menores quando comparados aos outros, ficando abaixo de 0.1 milissegundo. Por sua vez, a curva do switch HP se assemelha às curvas dos software switches estudados (LinkSys e Open vSwitch). Isso pode corroborar a suspeita de que a implementação OpenFlow do switch HP é apenas uma instanciação de um software switch rodando em memória, dentro de um hardware switch.

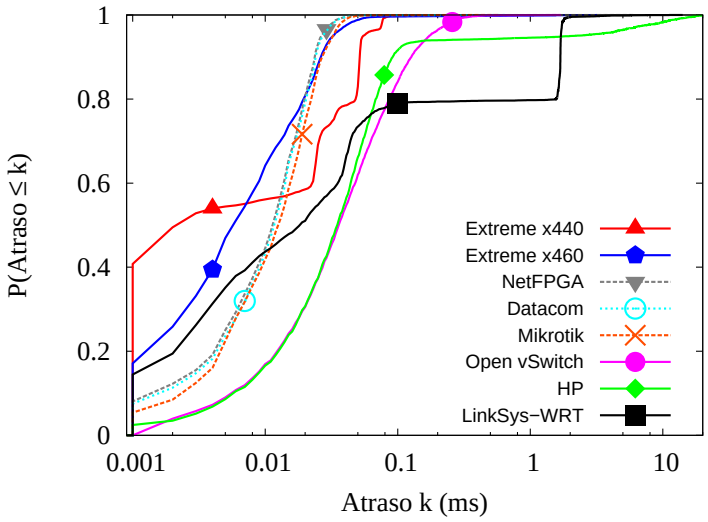

Figura 24 – Distribuições cumulativas dos jitters para o tipo de *match* exato (pacotes de 64 bytes).

Vale ressaltar que os resultados encontrados para os outros tipos de *match* foram bastante semelhantes aos resultados do *match* exato. Além disso, quando alterado o tamanho dos pacotes UDP, os resultados obtidos também foram bastante semelhantes entre si, para 64, 128 e 256 bytes. A Figura 25, que representa a distribuição cumulativa dos *jitters* dos pacotes de 64 bytes para o *match* por IP, e a Figura 26, que representa a distribuição cumulativa dos *jitters* para pacotes de 256 bytes para o *match* exato, são exemplos de distribuições que ilustram as situações relatadas anteriormente. É possível notar a semelhança entre essas figuras e a Figura 24. Esses fatos podem ser um indicativo de que tanto o tipo de *match* quanto o tamanho dos pacotes UDP não influenciam o valor do *jitter* dos pacotes.

5.3.2 Desempenho dos switches nos modos *legacy* e OpenFlow

O segundo cenário definido compara os desempenhos dos switches avaliados quando estes operam em seus modos OpenFlow e normal (*legacy*). Serão comparados os desempenhos da operação de encaminhamento dos pacotes entre a porta de chegada e a porta de saída do switch nos dois modos de operação, por meio do valor do atraso de cada pacote enviado. O objetivo deste teste é verificar como os switches se comportam em cada modo de operação, e, principalmente, verificar se existe alguma diferença substancial de desempenho entre esses modos, e quantificar essa diferença, a fim de mostrar se há vantagem ou não na utilização do protocolo OpenFlow.

Os testes desse cenário consistiram na execução da operação de encaminhamento de pacotes nos dois modos citados, em cada um dos oito switches avaliados, para três tamanhos diferentes de pacotes UDP (64, 128 e 256 bytes), gerando um total de 48

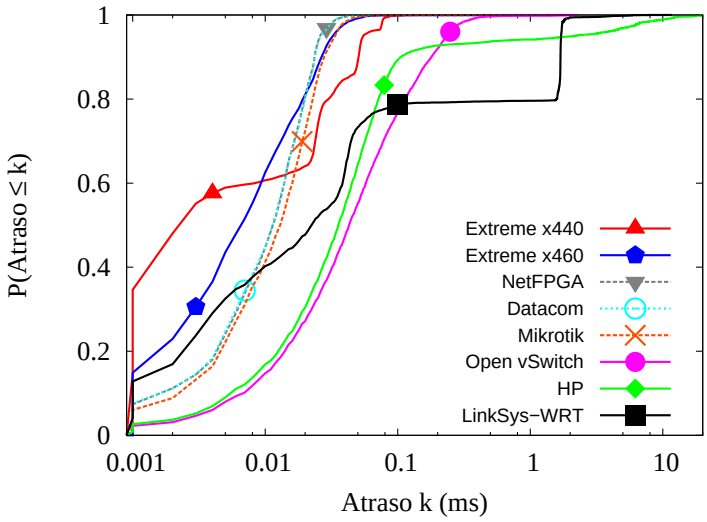

Figura 25 – Distribuições cumulativas dos *jitters* para o tipo de *match* por IP (pacotes de 64 bytes).

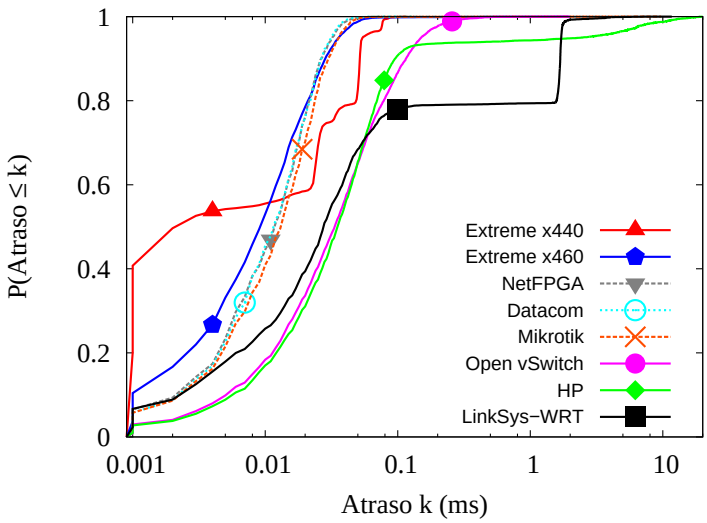

Figura 26 – Distribuições cumulativas dos *jitters* para o tipo de *match* exato (pacotes de 256 bytes).

experimentos neste cenário. Como já mencionado, em cada um dos experimentos, foram enviados 10.000 pacotes UDP através dos switches. Para as operações nos switches com modo normal, houve apenas o encaminhamento normal de pacotes da máquina cliente para o servidor passando através do switch, sem nenhuma configuração adicional. Contudo, para reproduzir o encaminhamento porta a porta de um switch normal no modo OpenFlow, foi necessário, primeiramente, habilitar o protocolo nos switches, e, em seguida, instalar uma regra no controlador que realizava *matches* apenas no atributo de porta de entrada (*in_port*) e repassava os pacotes que sofriam esse *match* para uma porta de saída determinada. Essa seria uma maneira mais igualitária de comparar os dois modos de

operação.

As Figuras 27 e 28 apresentam as distribuições cumulativas dos atrasos dos pacotes de 64 bytes para os modos *legacy* e OpenFlow, respectivamente. Para os switches em modo *legacy*, nota-se que, à exceção do Open vSwitch, todos os outros switches apresentam atrasos entre 0.1 e 0.2 milissegundos para quase 100% de seus pacotes. Os switches Extreme x440 e x460 possuem os menores atrasos, em torno de 0.1 milissegundo, e os switches HP, Datacom e Mikrotik, o roteador LinkSys-OpenWRT e a placa NetFPGA vem logo em seguida, com atrasos em torno de 0.2 milissegundos. Já o Open vSwitch possui o pior desempenho no modo *legacy*, com atrasos distribuídos entre 0.2 e 0.8 milissegundos. Esse resultado é até esperado, uma vez que o Open vSwitch é uma implementação dependente do ambiente virtual no qual está inserido, o que pode influenciar as operações de comutação do dispositivo.

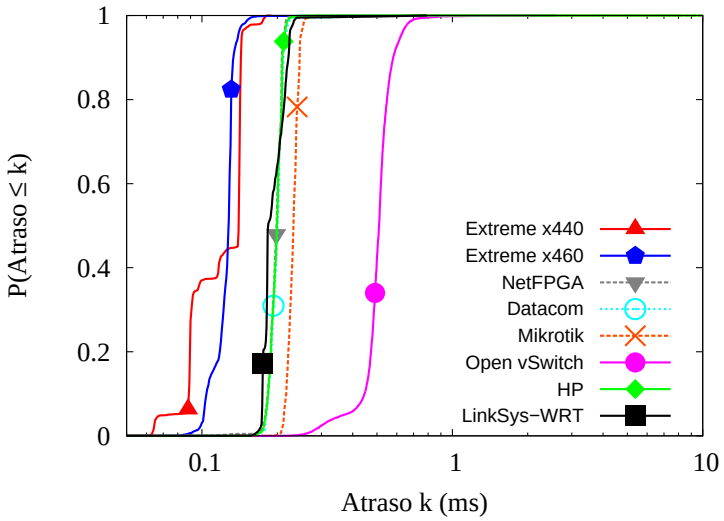

Figura 27 – Distribuições cumulativas dos atrasos para o modo *legacy* (pacotes de 64 bytes).

Já para os desempenhos dos switches em modo OpenFlow, existe uma mudança no cenário. Ao inserir a tecnologia OpenFlow nas operações de um switch, insere-se também um *overhead* de processamento, uma vez que passa a existir um novo tipo de pacote (pacote OpenFlow) com novos campos no cabeçalho, além do processamento dessas novas informações. Isso sem contar a influência do controlador, que também insere um atraso nas operações (vale lembrar que, para os experimentos deste trabalho, a influência do controlador está descartada). Alguns dos switches praticamente não sofreram a influência desse *overhead* e apresentaram bons resultados. Para outros, a adição da tecnologia trouxe problemas em termos de desempenho. De acordo com a Figura 28, os switches Extreme (x440 e x460), Datacom e Mikrotik, e a placa NetFPGA mantiveram bons desempenhos, com atrasos variando entre 0.1 e 0.25 milissegundos. O Open vSwitch manteve seus atrasos distribuídos entre 0.3 e 0.8 milissegundos. Já a maior queda de desempenho se

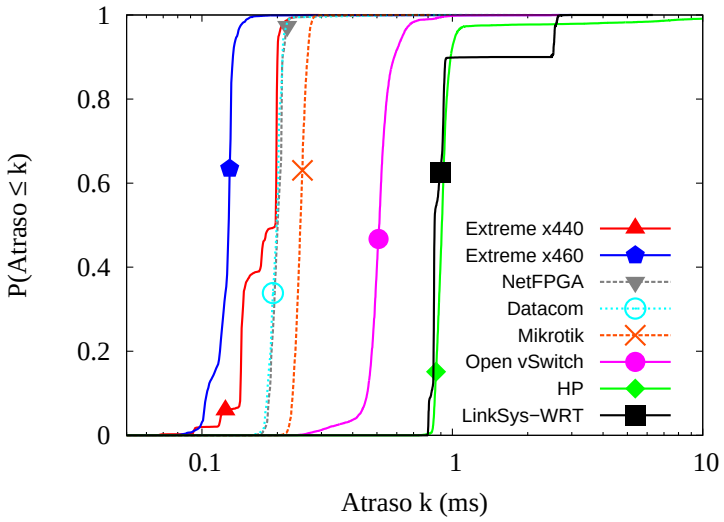

Figura 28 – Distribuições cumulativas dos atrasos para o modo OpenFlow (pacotes de 64 bytes).

deu para o switch HP e o roteador LinkSys com OpenWRT, que apresentaram atrasos entre 0.8 e 1.0 milissegundos para 90% de seus pacotes, sem falar no atrasos superiores a 2.0 milissegundos, que influenciaram bastante o desempenho de ambos.

Para tamanhos maiores de pacote UDP, o fenômeno descrito anteriormente é o mesmo. A única diferença reside no aumento dos atrasos em alguns switches devido ao tamanho dos pacotes, mas o comportamento nos dois modos é equivalente ao dos pacotes de 64 bytes. As Figuras 29 e 30 apresentam as distribuições cumulativas dos atrasos dos pacotes de 128 bytes para os modos *legacy* e OpenFlow, respectivamente. No gráfico referente ao modo *legacy*, enquanto o switch Extreme x440 mantém os atrasos em torno de 0.1 milissegundos e o Open vSwitch entre 0.3 e 0.8 milissegundos, os demais apresentam atrasos de pacote entre 0.2 e 0.3 milissegundos. No modo OpenFlow, alguns switches também tiveram aumento nos atrasos de seus pacotes, mas mantiveram o padrão da distribuição no modo OpenFlow para pacotes de 64 bytes. Para pacotes de 256 bytes, os comportamentos e as distribuições foram muito semelhantes aos dos obtidos com os pacotes de 128 bytes, e, por isso, não foram representados em gráficos.

Na tentativa de quantificar a diferença entre os dois modos de operação nos switches avaliados, utilizou-se a métrica de Speedup para a comparação dos desempenhos. Essa métrica consiste na divisão da latência no modo normal pela latência no modo OpenFlow. No gráfico da Figura 31 é mostrado o desempenho dos switches em modo OpenFlow quando comparados ao desempenho dos mesmos em modo normal, para os três tamanhos de pacotes utilizados neste trabalho, utilizando a métrica de *Speedup*. No gráfico, valores próximos a 1.0 correspondem a desempenhos semelhantes dos modos OpenFlow e normal. Valores menores que 1.0 indicam uma piora no desempenho no modo OpenFlow, enquanto valores superiores a 1.0 indicam um desempenho superior utilizando OpenFlow. Os

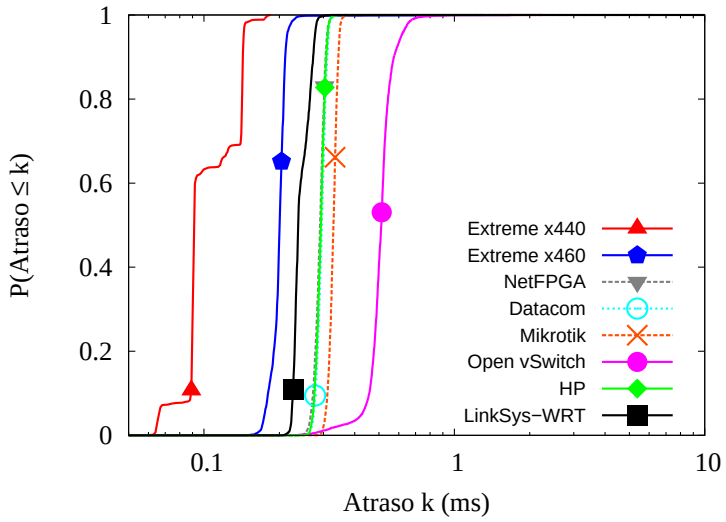

Figura 29 – Distribuições cumulativas dos atrasos para o modo *legacy* (pacotes de 128 bytes).

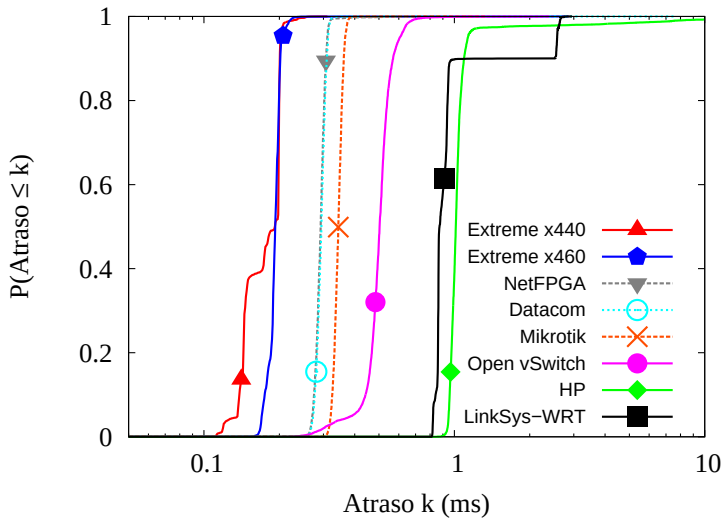

Figura 30 – Distribuições cumulativas dos atrasos para o modo OpenFlow (pacotes de 128 bytes).

intervalos de confiança nas barras são bem pequenos, por isso não estão reproduzidos na imagem.

Ao analisar o gráfico, nota-se que o desempenho do modo OpenFlow é bem próximo ao desempenho do modo *legacy* nos dois switches Extreme (x460 e x440), no Open vSwitch, nos switches Mikrotik e Datacom, e na placa NetFPGA. Para esses equipamentos, o desempenho varia entre 90% e 100% do desempenho obtido para o modo normal, para todos os tamanhos definidos de pacote. Já os desempenhos do modo OpenFlow nos switches HP e LinkSys-OpenWRT são bem inferiores aos desempenhos em modo *legacy*, chegando a taxas entre 20% e 25%, apenas, do desempenho normal em ambos, para os todos os tamanhos de pacote. Essas observações levam a crer que já é vantagem, em

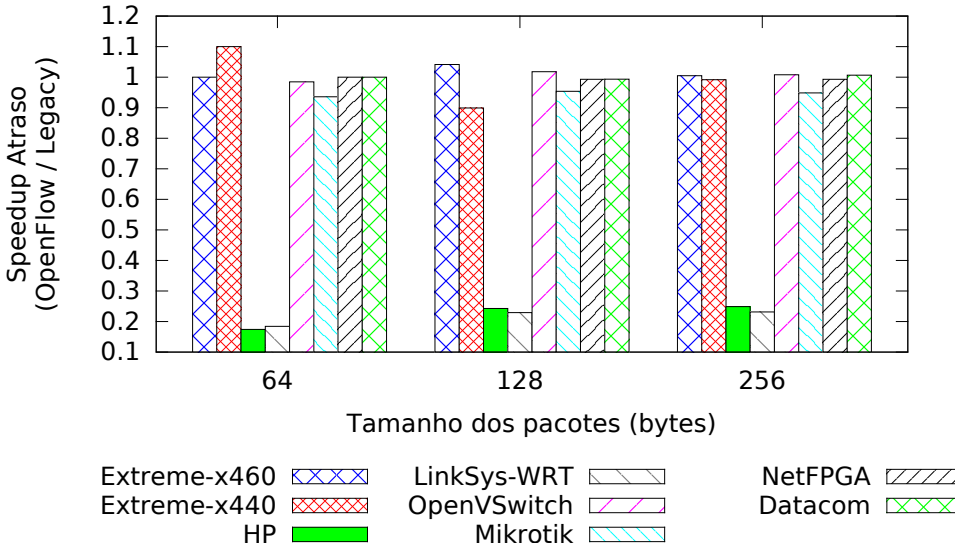

Figura 31 – Desempenhos dos switches em modo OpenFlow em relação aos desempenhos em modo *legacy*.

alguns equipamentos, utilizar a tecnologia OpenFlow, quando se considera apenas as taxas de latência dos pacotes. Nesses casos, a utilização do OpenFlow praticamente não acarretou sobrecarga em operações normais do switch que não sofrem influência direta do controlador. Além disso, é possível pensar que os mecanismos de encaminhamento internos dos switches de bom desempenho podem também ser utilizados pela tecnologia OpenFlow, o que explicaria o desempenho semelhante entre os dois modos.

Finalizando, vale destacar ainda a diferença observada entre o formato de algumas curvas de distribuição nos modos *legacy* e OpenFlow (Figuras 27 e 28). Observando as curvas referentes aos switches HP e LinkSys, pode-se notar que, além de terem se deslocado para a direta, o formato de ambas mudou quando foi variado o modo de operação de *legacy* para OpenFlow. Essa mudança no formato das curvas poderia indicar, por exemplo, uma mudança na natureza do switch, passando um hardware switch a se comportar como um software switch, com a habilitação do protocolo OpenFlow. Os valores dos atrasos em implementações em hardware tendem a ser menos distribuídos, uma vez que não há troca de contexto na operação do switch. Já uma forma de curva que apresenta valores mais distribuídos poderia indicar a ocorrência de outros eventos (por exemplo, troca de contexto ou escalonamento de outras tarefas) que geram interrupções no processo, o que poderia caracterizar a atuação de um software switch. Além disso, o formato da curva pode indicar o tipo de pesquisa de regras nos diferentes modos de operação. Em modo *legacy*, o switch pode realizar pesquisas em paralelo (com TCAM), enquanto em modo OpenFlow essas pesquisam podem ser realizadas de forma sequencial em uma tabela em memória.

5.3.3 **Desempenho com uma ou múltiplas regras na tabela de fluxos**

O terceiro cenário definido busca avaliar o desempenho do switch em duas situações opostas. A primeira situação ocorre quando a tabela de fluxos do switch possui apenas uma regra instalada, com a qual necessariamente casará o fluxo de teste enviado, ou seja, irá ocasionar um *match* válido. A segunda situação ocorre quando a tabela de fluxos está completamente preenchida por múltiplas regras, mas apenas uma delas casará com o fluxo de teste enviado. O objetivo deste teste é verificar se o desempenho de um switch em modo OpenFlow pode ser influenciado pela quantidade de regras instaladas na tabela, pela disposição das regras na tabela de fluxos ou pelo processamento interno de cada switch.

Dessa forma, pode ser possível determinar se a busca de regras para a realização de *matches* foi realizada em paralelo (através de TCAM), em *hash* ou de forma sequencial. Se o atraso aumentar, possivelmente foi realizada pesquisa sequencial. Se não aumentar, possivelmente foi realizada busca em *hash* ou paralela. O tipo de busca de regras pode impactar o desempenho do balanceamento de carga, dependendo da quantidade de regras que pode ser suportada.

Entretanto, antes de serem realizados os experimentos, foi necessário descobrir o número máximo de regras que podem ser instaladas em cada switch em seu modo OpenFlow. Descobrir esse número é importante pois, em switches com mais regras, podem ser acomodadas mais aplicações ao mesmo tempo, além de serem realizadas menos atualizações da tabela, o que é um fator positivo para o desempenho do switch. Foram executados códigos no controlador que instalam regras em cada switch OpenFlow de forma automática e sequencial, até que fosse atingido algum determinado limite informado pelo protocolo OpenFlow através de mensagens de erro. Um exemplo de erro é o OFPFMFC_ALL_TABLES_FULL, que informa que todas as tabelas de fluxo estão completamente ocupadas. Com o limite atingido, bastou verificar o número de regras instaladas até o momento do erro. A Tabela 6 apresenta esse número máximo de regras instaladas obtido para cada switch OpenFlow, além do número máximo de regras suportadas pelos switches em modo *legacy*, obtido por meio da documentação dos fabricantes dos switches. Ao analisar esses dados, foi observado que, na maioria dos casos, o número de regras suportadas no OpenFlow é menor do que o número de regras suportadas nativamente.

Ao se investigar o motivo dessa diferença, observou-se que os switches Extreme (x460 e x440) armazenam as regras OpenFlow em formato de *access-lists* (ACLs) em hardware, assim como fazem com as regras quando operando em modo normal. Isso pode explicar o desempenho semelhante dos dois modos de operação (OpenFlow e *legacy*) para os switches Extreme, mostrados no cenário anterior. Como, no modo OpenFlow, existem mais alguns atributos e operações a serem armazenadas, e o tamanho da ACL é definido como *double*, menos regras podem ser armazenadas no modo OpenFlow. Por sua vez, o

Switches	Número de regras no modo legacy	Número de regras no modo OpenFlow
Extreme x460-24p	2048	1200
Extreme x440-48p	1024	248
HP	2048	16000
LinkSys - OpenWRT	100	100
Open vSwitch	1000000	750000
Mikrotik	16318	389
NetFPGA	16	124
Datacom	32768	2051

Tabela 6 – Quantidade de regras suportadas nos modos *legacy* e OpenFlow.

Open vSwitch possui capacidade de armazenamento de até 1 milhão de regras em uma tabela *hash* desenvolvida para armazenar apenas *matches* exatos, mas escolhe o limite de acordo com os tipos de regras e as condições da rede. Para armazenar regras de *matches* não exatos, o Open vSwitch utiliza uma tabela linear de até 100 posições.

Em relação ao switch HP, na instalação da tecnologia OpenFlow, ele passa a possuir além de uma tabela em hardware, uma outra tabela em software, com uma grande capacidade de armazenamento de regras, e que não pode ser desativada. Esse fato pode indicar que, possivelmente, essa implementação do OpenFlow seja uma implementação em software do Open vSwitch, sem maior integração ao projeto de hardware do switch HP. Isso explicaria o baixo desempenho no modo OpenFlow do switch HP, apresentado no cenário anterior.

Por sua vez, o NetFPGA passa a possuir três tabelas com a instalação do OpenFlow: uma tabela *hash*, para armazenar entradas referentes a *matches* exatos; uma tabela linear, para armazenar entradas referentes a *matches* não-exatos (*wildcards*); e uma tabela em software, denominada “nf2”, que é uma cópia das tabelas anteriores, que são em hardware. Quando uma nova regra deve ser instalada na NetFPGA, a primeira tentativa de armazenamento é realizada justamente na tabela “nf2”. No entanto, se as ações requisitadas não são suportadas pela NetPFGA ou se é excedido o número máximo de entradas, então a entrada é escrita ou na tabela *hash*, ou na tabela linear, dependendo do tipo do *match*, se for exato ou não.

Switches	Uma regra	Múltiplas regras
Extreme x460-24p	0.121 - 0.125	0.124 - 0.128
Extreme x440-48p	0.109 - 0.110	0.114 - 0.115
HP	1.049 - 1.120	1.046 - 1.330
LinkSys - OpenWRT	1.028 - 1.055	1.215 - 1.242
Open vSwitch	0.508 - 0.511	0.513 - 0.517
Mikrotik	0.243 - 0.244	0.246 - 0.247
NetFPGA	0.197 - 0.198	0.196 - 0.197
Datacom	0.199 - 0.200	0.199 - 0.200

Tabela 7 – Intervalos de confiança dos atrasos em tabelas com uma e múltiplas regras (em ms).

Na Tabela 7 estão listados os intervalos de confiança do atraso médio dos pacotes em

duas situações: no caso em que a tabela de fluxos contém apenas uma regra, e no caso em que a tabela de fluxos possui múltiplas regras falsas e apenas uma regra como verdadeira. Em ambos os casos, os *matches* foram exatos. A quantidade de regras instaladas, no último caso, equivale ao número limite de regras encontrado no experimento anterior, listado na Tabela 6 anteriormente. Vale ressaltar que nos switches com regras escritas em ACLs, a regra verdadeira foi inserida por último na tabela de fluxos. Já nos switches baseados em software, como as regras são inseridas utilizando índices *hash*, é impossível determinar sua efetiva posição na tabela de fluxos.

Pode ser observado que, na maioria dos switches, os intervalos se interceptam ou apresentam diferença de poucos microssegundos entre si. Isso pode indicar que a grande quantidade de regras instaladas e a disposição das mesmas na tabela de fluxos não sobrecarregam significativamente os switches, nem causam um sensível aumento nos atrasos. A única exceção a essa observação ocorre com o LinkSys-OpenWRT, que possui um aumento médio no atraso dos pacotes em torno de 0.2 milissegundos. Possivelmente, para o LinkSys, a pesquisa das regras para a realização de *matches* se dá de forma sequencial na tabela de fluxos.

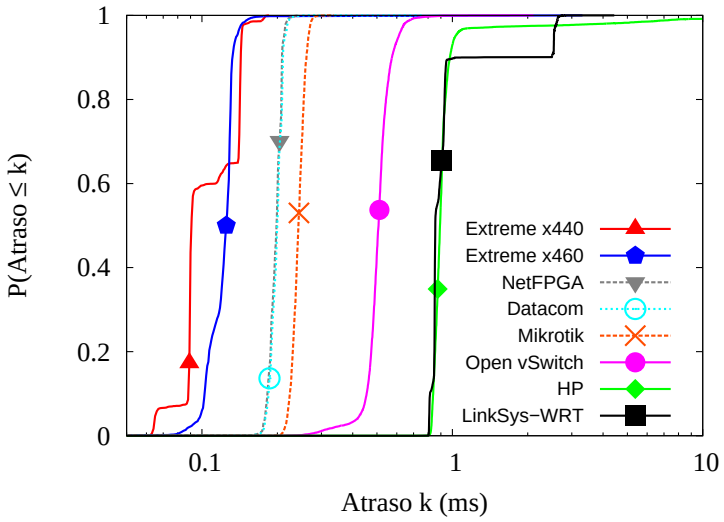

Figura 32 – Distribuições cumulativas dos atrasos para switches com apenas uma regra instalada (pacotes de 64 bytes).

Os gráficos das Figuras 32 e 33, que apresentam as distribuições cumulativas dos atrasos para os switches com uma única regra instalada e com múltiplas regras instaladas na tabela de fluxos, respectivamente, também ilustram a situação explicitada anteriormente. Os switches Extreme (x460 e x440), Datacom, Mikrotik e a placa NetFPGA apresentam distribuições bem semelhantes nos dois casos, variando entre 0.1 e 0.25 milissegundos para quase a totalidade de seus atrasos. O Open vSwitch e o switch HP também apresentaram comportamentos semelhantes em suas distribuições nos dois casos, contudo seus atrasos variaram entre 0.3 e 0.7 milissegundos para o Open vSwitch), e 0.8 e 1.0 milissegundo para

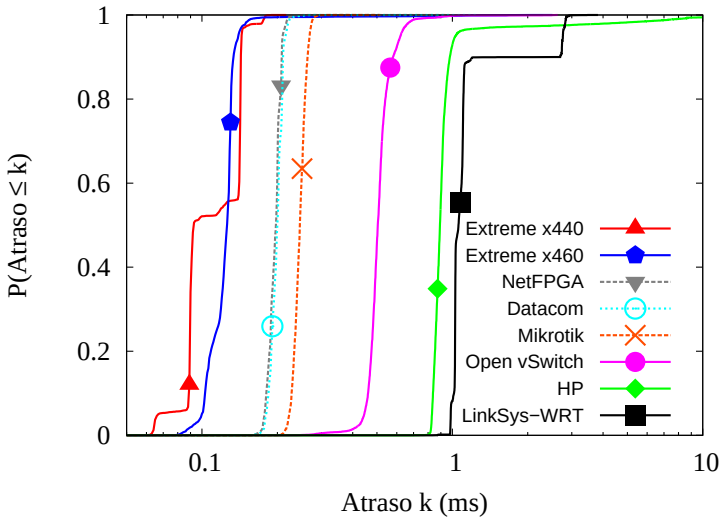

Figura 33 – Distribuições cumulativas dos atrasos para switches com múltiplas regras instaladas (pacotes de 64 bytes).

95% dos atrasos do switch HP. Para a distribuição dos atrasos para o LinkSys-OpenWRT, é possível notar visualmente o deslocamento da curva para a direita entre os dois casos. Para o switch com apenas uma regra instalada, cerca de 90% de seus atrasos variaram de 0.8 a 1.0 milissegundo, chegando até a possuir um desempenho semelhante ao do switch HP. No entanto, com múltiplas regras, a distribuição encontrada passou a variar entre 1.0 e 1.2 milissegundos para os mesmos 90% dos atrasos encontrados.

5.3.4 **Desempenho na modificação de campos do cabeçalho dos pacotes**

O quarto cenário avalia o desempenho de cada switch em operações de modificação de campos individuais específicos do cabeçalho de cada pacote. No teste desse caso, foram realizadas operações de modificação nos seguintes campos de cabeçalho: endereço MAC de destino, endereço IP de destino, prioridade de VLAN, IP *Type of Service* e porta UDP. A ideia por trás desses testes é verificar o atraso de cada uma dessas operações, a fim de compará-lo com operações de *match* que não realizam modificação de cabeçalho, e também comparar as mesmas operações realizadas nos diferentes switches avaliados. Além disso, deseja-se verificar o suporte a essas operações em cada um dos switches. Vale ressaltar que as operações de modificação de cabeçalho são opcionais para o protocolo OpenFlow em sua versão 1.0.

Os testes desse cenário consistiriam na execução de cinco operações de modificação de cabeçalho realizadas individualmente, em cada um dos oito switches avaliados, para pacotes UDP de 64 bytes, contudo, algumas operações de modificação não são suportadas por todos os switches. Como já mencionado, em cada um dos experimentos, foram enviados 10.000 pacotes UDP através dos switches. Além disso, foram realizados experimentos nos

quais não ocorria modificação de cabeçalho, com o intuito de comparar os atrasos obtidos na ausência de modificação com os atrasos verificados quando da ocorrência de alguma operação de modificação. Cada código executado no controlador, um para cada ação de modificação, instala regras que realizam *match* exato na verificação dos pacotes, e também realizam a (única) operação de modificação correspondente. Devido a certas configurações da arquitetura da rede, a operação de modificação de identificador da VLAN não pôde ser realizada. Além disso, vale salientar que alguns switches como Mikrotik e HP, por exemplo, utilizam esse identificador para a configuração do OpenFlow e, por isso, não suportam a sua reescrita. Também existem operações de modificação dos endereços MAC e IP de origem e porta UDP de origem, mas, por simplificação de testes, foram incluídos apenas os resultados referentes aos endereços MAC e IP e porta UDP de destino.

Switches	**Tipo de modificação**					
	Sem modificação	**MAC**	**IP**	**Prioridade de VLAN**	**IP ToS**	**Porta UDP**
Extreme x460-24p	0.125 – 0.127	0.125 – 0.129	-	0.125 – 0.128	-	-
Extreme x440-48p	0.114 - 0.115	0.113 - 0.115	-	-	-	-
HP	1.121 – 1.212	1.128 – 1.203	1.107 – 1.178	1.107 – 1.183	1.122 – 1.194	1.071 – 1.132
LinkSys WRT	1.032 – 1.058	1.034 – 1.061	1.034 – 1.060	-	1.030 – 1.056	1.033 – 1.060
Open vSwitch	0.508 - 0.511	0.511 - 0.516	0.510 - 0.515	-	0.506 - 0.510	0.512 - 0.515
Mikrotik	0.244 – 0.248	0.244 – 0.247	-	0.245 – 0.248	-	-
Net FPGA	0.193 – 0.197	0.191 – 0.193	-	0.194 – 0.195	-	0.193 – 0.194
Datacom	0.198 - 0.203	0.199 - 0.200	-	-	-	-

Tabela 8 – Intervalos de confiança dos atrasos médios em modificações dos campos de cabeçalho (em ms).

Na Tabela 8 estão listados os intervalos de confiança dos atrasos dos pacotes quando estes sofrem ações de modificação em cinco campos de seus cabeçalhos, individualmente. Para efeito de comparação, também foram inseridos na tabela os valores do caso em que os pacotes não sofreram nenhuma alteração em seus cabeçalhos. Para todos os casos, não se notou um considerável aumento nos tempos de atraso médio dos pacotes nas modificações, quando comparados ao caso em que não há modificação. Inclusive, os intervalos de confiança para as operações de modificação em um mesmo switch se interceptam, em todos os casos. Isso indica que a operação de modificação ocorre na ordem de poucos microssegundos, independente do tipo de operação ou do switch avaliado, não ocasionando nenhum impacto severo nos atrasos dos pacotes.

Entretanto, um fato observado merece destaque. Alguns switches ainda não possuem suporte para algumas operações de modificação de cabeçalho. De fato, essa limitação está prevista na documentação de cada um dos switches. Como já dito, a versão 1.0

do protocolo OpenFlow traz as operações de modificação de cabeçalho como sendo de implementação opcional. E com a ausência de algumas dessas operações, algumas tarefas, como o balanceamento de carga, por exemplo, acabam se tornando muito mais complexas. Essa falta de suporte, aliada ao fato de que só é permitido realizar uma única operação de modificação de cabeçalho por regra, mostra que ainda existem aspectos a se melhorar nas implementações OpenFlow já existentes, sobretudo nas soluções em hardware.

5.3.5 **Desempenho nas operações de *flowstats* e *portstats***

O quinto e último cenário avalia o desempenho de cada switch nas operações de *flowstats* e *portstats*. Nos testes desse cenário, o controlador requisita estatísticas (*flowstats* ou *portstats*) ao switch uma vez a cada segundo, e calcula, posteriormente, o atraso de cada resposta enviada pelo switch. Vale ressaltar que esses testes foram executados com o switch operando em quatro situações distintas, através de combinações que envolvem a variação da carga de trabalho a qual o switch foi submetido (carga leve ou carga pesada) e a variação da quantidade de regras instaladas na tabela de fluxos do switch (tabela com uma única regra ou tabela completamente preenchida de regras).

Observar o desempenho dos switches nas operações de *flowstats* e *portstats* pode ser importante para definir se esse é o método mais vantajoso para obtenção de dados estatísticos em um esquema de balanceamento de carga, ou se o método mais vantajoso é requisitar estatísticas de rede de cada servidor.

A carga de trabalho foi gerada com uso do programa Iperf. Para submeter o switch a uma carga leve, o Iperf foi configurado para gerar e enviar pacotes UDP a uma taxa de 1Mb/s. Já para gerar carga pesada, o programa Iperf foi configurado para gerar pacotes UDP à taxa máxima suportada por cada um dos switches avaliados.

Para o teste de múltiplas regras, o número máximo de regras instaladas variou de acordo com o switch, uma vez que cada switch possui um determinado número limite de regras que, se ultrapassado, pode causar instabilidade nas operações do equipamento. Durante a configuração dos experimentos, esse número limite foi empiricamente detectado através da observação da estabilidade de cada switch, indicada pela presença ou ausência de mensagens de erro e pela perda de conexão entre controlador e switch. No geral, esse número máximo de regras é aproximadamente 10% menor do que o limite encontrado para cada switch nos testes do Cenário 3, apresentado na Tabela 6, devido à sobrecarga causada pelas operações de *portstats*/*flowstats* nos switches.

As Figuras 34 e 35 apresentam as distribuições cumulativas para os atrasos do *flowstats* para as tabelas com uma única regra e com múltiplas regras, respectivamente. Em ambos os casos, os switches operavam sob carga de trabalho pesada. É possível notar que existe um considerável aumento no atraso do *flowstats*, para todos os switches, quando comparamos os switches operando com tabelas quase vazias e cheias. Este é um

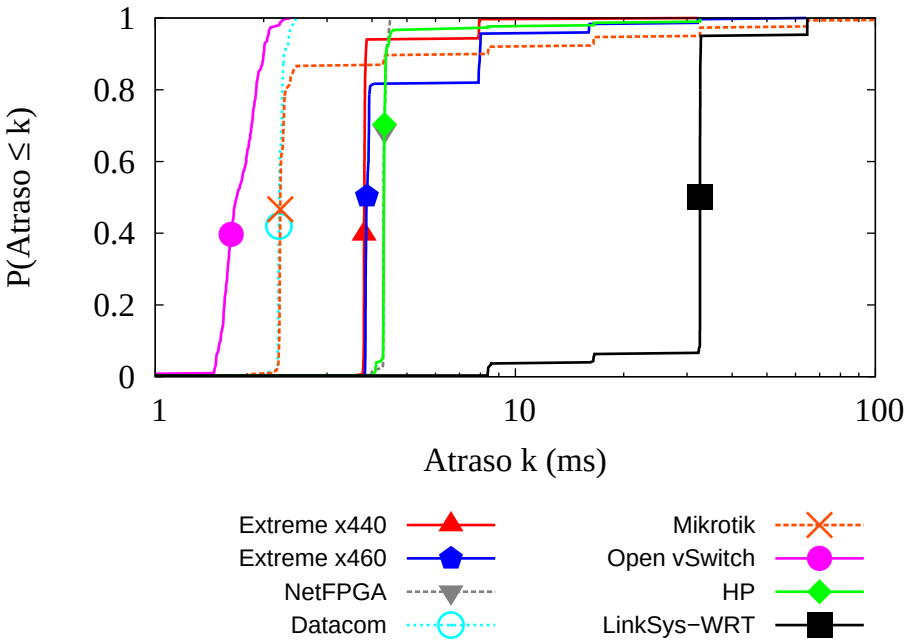

Figura 34 – Distribuições cumulativas dos atrasos de *flowstats* para switches com apenas uma regra instalada, sob carga pesada.

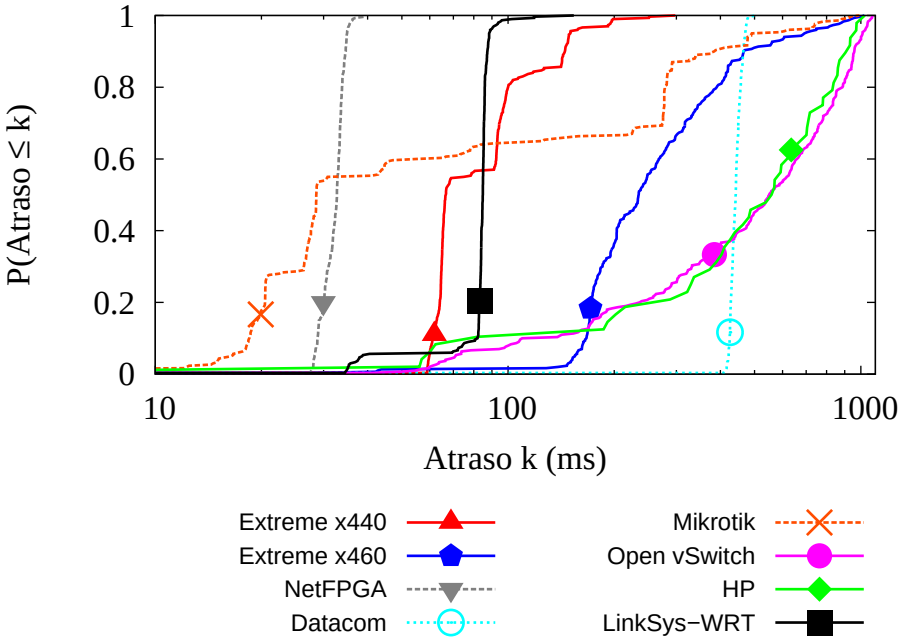

Figura 35 – Distribuições cumulativas dos atrasos de *flowstats* para switches com múltiplas regras instaladas, sob carga pesada.

comportamento esperado, uma vez que a mensagem de *flowstats* retorna informações sobre todas as regras ativas no momento. Vale destacar o aumento do atraso, principalmente, para os switches Datacom, HP e Open vSwitch.

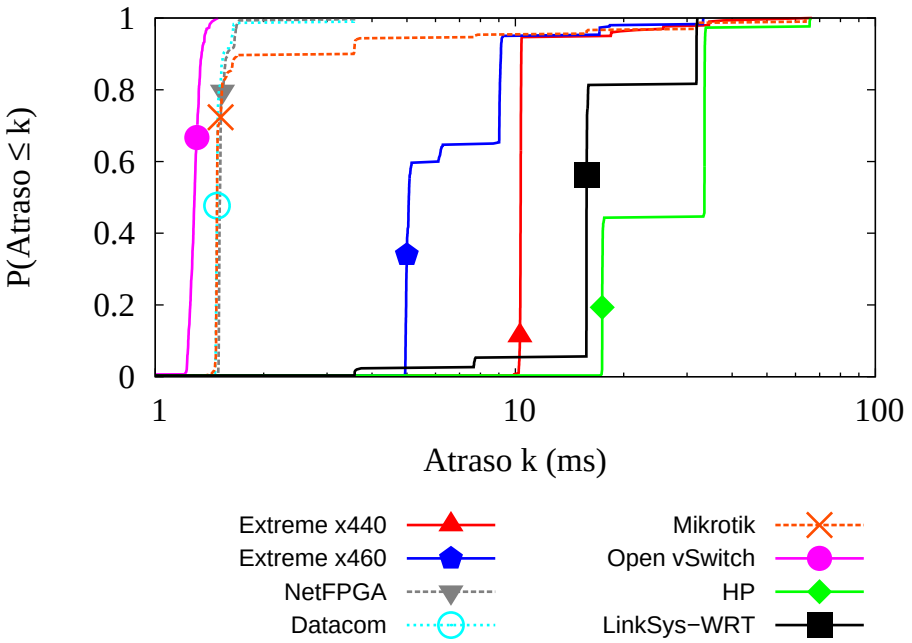

Figura 36 – Distribuições cumulativas dos atrasos de *portstats* para switches com apenas uma regra instalada, sob carga pesada.

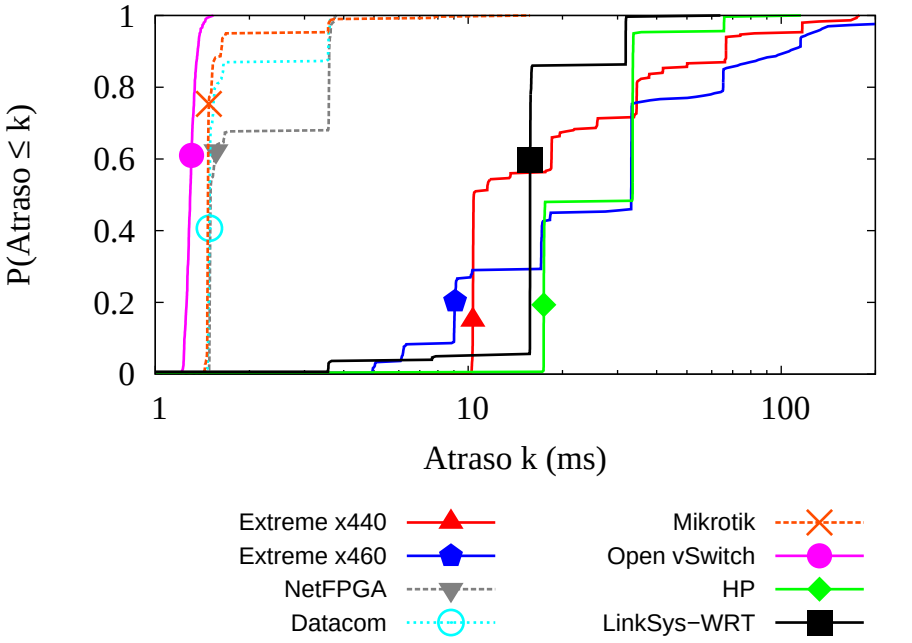

Figura 37 – Distribuições cumulativas dos atrasos de *portstats* para switches com múltiplas regras instaladas, sob carga pesada.

Por sua vez, o número de regras instaladas influencia de forma mais leve a operação de *portstats* quando comparada a operação de *flowstats*. De fato, como ilustrado nas Figuras 36 e 37, essa influência ocorre apenas em alguns switches (Extreme x440 e x460) e de forma bem branda.

Switches	Uma Regra		Múltiplas Regras	
	Carga Leve	Carga Pesada	Carga Leve	Carga Pesada
Extreme x460-24p	4.430 - 5.762	4.646 - 6.042	275.726 - 358.626	256.013 - 332.985
Extreme x440-48p	3.639 - 4.735	3.595 - 4.677	80.360 - 104.520	76.908 - 100.032
HP	4.048 - 5.264	4.224 - 5.494	375.217 - 699.857	363.165 - 720.289
LinkSys - OpenWRT	7.659 - 9.961	28.582 - 37.176	31.619 - 41.125	71.672 - 93.220
Open vSwitch	1.763 - 2.293	1.502 - 1.954	459.776 - 598.012	483.018 - 628.242
Mikrotik	7.340 - 9.548	5.518 - 7.176	13.659 - 17.767	130.058 - 169.160
NetFPGA	3.748 - 4.874	3.758 - 4.888	27.108 - 35.258	27.971 - 36.381
Datacom	1.939 - 2.521	1.938 - 2.520	381.685 - 496.443	384.893 - 500.615

Tabela 9 – Intervalos de confiança para os atrasos de *flowstats* (em ms).

Switches	Uma Regra		Múltiplas Regras	
	Carga Leve	Carga Pesada	Carga Leve	Carga Pesada
Extreme x460-24p	5.987 - 7.787	6.288 - 8.178	25.574 - 33.264	35.053 - 45.591
Extreme x440-48p	9.566 - 12.442	9.913 - 12.893	21.577 - 28.065	23.582 - 30.672
HP	15.728 - 20.456	23.706 - 30.834	15.904 - 20.686	23.862 - 31.036
LinkSys - OpenWRT	4.480 - 5.826	15.876 - 20.650	4.627 - 6.019	15.288 - 19.884
Open vSwitch	1.261 - 1.641	1.120 - 1.456	1.258 - 1.636	1.131 - 1.471
Mikrotik	1.480 - 1.924	2.860 - 3.720	3.017 - 3.923	1.445 - 1.879
NetFPGA	1.332 - 1.732	1.341 - 1.745	2.260 - 2.940	1.903 - 2.475
Datacom	1.345 - 1.749	1.327 - 1.725	1.486 - 1.932	1.542 - 2.006

Tabela 10 – Intervalos de confiança para os atrasos de *portstats* (em ms).

A Tabela 9 apresenta os intervalos de confiança para o atraso médio do *flowstats* para os switches testados, sob as diferentes cargas de trabalho. O aumento da carga de trabalho do sistema influenciou o atraso do *flowstats* de diferentes formas. Por exemplo, considerando a variação da carga de trabalho com os switches operando com múltiplas regras, os switches Mikrotik e LinkSys tiveram um aumento considerável nos atrasos de *flowstats*. Já os switches Open vSwitch, HP, Datacom e NetFPGA mantiveram seus atrasos praticamente inalterados, apesar do aumento da carga. Curiosamente, os switches Extreme (x460 e x440) apresentaram uma diminuição em seus atrasos médios. Enquanto isso, a implementação LinkSys-OpenWRT teve uma severa degradação do desempenho para carga pesada, mesmo com a tabela possuindo apenas uma regra instalada.

Por sua vez, a Tabela 10 apresenta os intervalos de confiança para o atraso médio do *portstats*. Neste caso, foi observado que o aumento da carga de trabalho do sistema também influenciou o atraso do *portstats*, principalmente para os switches Extreme x460, HP e LinkSys. Além disso, observou-se que o desempenho dos switches foi influenciado mais pela variação da carga de trabalho do que pelo número de regras instaladas.

Concluindo, de forma geral, deve-se ressaltar que a operação de *flowstats* é mais sensível às alterações no tamanho da tabela de fluxos e na carga de trabalho do que a operação de *portstats*. Isso deve ocorrer porque o número de portas em um switch é consideravelmente menor quando comparado ao número de regras suportadas, o que impacta diretamente no tamanho das mensagens de *portstats*, que armazenam menos dados estatísticos do que as mensagens de *flowstats*. Além disso, diferente do que ocorre com

o desempenho no encaminhamento de pacotes, não existe uma diferença de desempenho perceptível entre implementações em hardware e software para as operações de *flowstats* e *portstats*.

5.4 DISCUSSÕES GERAIS SOBRE A AVALIAÇÃO DOS PLANOS DE DADOS

Nenhum dos hardware switches avaliados implementa completamente as características opcionais da especificação OpenFlow 1.0, que são muito empregadas nas aplicações de SDN. Além disso, durante a avaliação, foram identificadas várias deficiências dos firmwares. Um exemplo de deficiência encontrada é a de que, dependendo do switch, nem todos os atributos definidos pela especificação OpenFlow 1.0 estão disponíveis para *match*. É o caso do atributo *dl_vlan* (identificador da VLAN), que é utilizado em muitos comutadores para separar as portas do plano de controle das portas do plano de dados, e que apresenta problemas na realização de *matches*, por exemplo, nos switches HP e Mikrotik. Um outro exemplo de deficiência refere-se à quantidade de comandos OpenFlow enviados ao switch. Se forem enviados muitos comandos em sequência como, por exemplo, comandos de instalação de regras na tabela de fluxos, a maioria dos switches não consegue processá-los rapidamente, ocasionando atraso ou até mesmo a falha do switch, o que seria extremamente grave em uma rede em produção. Dessa forma, o controlador deve respeitar um intervalo de tempo entre a instalação de duas regras, para que não haja algum problema nesse sentido.

Além dos problemas já citados, também vale destacar que alguns planos de dados tornam-se instáveis quando a tabela de regras está próxima de atingir a sua capacidade total de armazenamento, e outros simplesmente falham quando esse limite é atingido. Vale destacar também a instabilidade dos switches OpenFlow na instalação de regras ao mesmo tempo em que estão ocorrendo outros processamentos no switch, como por exemplo, a obtenção de dados estatísticos. A queda de desempenho, nesse sentido, é severa e acaba limitando o funcionamento do sistema, e foi observada tanto para hardware quanto para software switches.

Em relação ao desempenho das operações, não é possível escolher onde cada regra será instalada, por exemplo, em TCAM ou SRAM. Switches que suportam ambos os tipos de memória escolhem automaticamente onde instalar a regra, dependendo do tipo de *match* (exato ou inexato). Isso pode mudar em firmwares que implementam o protocolo OpenFlow em sua versão 1.3, uma vez que esta versão possui suporte a múltiplas tabelas.

A escolha de qual o melhor switch OpenFlow para se utilizar em uma rede depende muito da aplicação de interesse, uma vez que as implementações possuem diferentes números máximos de regras OpenFlow para armazenamento na tabela de fluxos, suportam diferentes operações de reescrita de cabeçalhos de pacotes, além de possuir desempenhos distintos tanto na latência quanto no processamento dos comandos. No geral, software

switches são mais compatíveis com a especificação do protocolo OpenFlow e suportam grandes tabelas de fluxos, porém o seu atraso no encaminhamento dos pacotes pode chegar a uma ordem de grandeza mais lenta do que a dos atrasos em hardware switches. No entanto, hardware switches geralmente não suportam grandes tabelas de fluxos e são incapazes de realizar reescritas de cabeçalho de pacotes em campos IP, o que dificulta a realização de muitas tarefas, como o balanceamento de carga.

Ao se analisar apenas os switches baseados em hardware, os switches Extreme tiveram os melhores desempenhos, no geral, devido aos baixos valores de latência e *jitter*. Na verdade, com exceção dos switches LinkSys, HP e Open vSwitch, todos os demais possuíram atrasos relativamente baixos, além de possuírem desempenhos semelhantes nos modos *legacy* e OpenFlow. O ponto negativo destes, como já salientado, é a pequena quantidade de regras suportadas no modo OpenFlow e os problemas severos de processamento de comandos, o que inviabilizaria o seu uso em redes de maior porte. O switch HP, por outro lado, apresentou um desempenho semelhante ao de software switches. Esse fato provê fortes indícios de que o switch HP pode executar operações OpenFlow sobre alguma CPU embutida, sem recorrer a qualquer otimização em hardware, tornando-se uma espécie de software switch ocultado por um hardware. Tanto o switch HP quanto a implementação WRT para o roteador LinkSys apresentaram o pior desempenho em relação aos seus modos *legacy*, chegando a apenas 20% do desempenho normal em relação aos atrasos, o que inviabilizaria o uso dos mesmos em muitas operações envolvendo o protocolo OpenFlow.

Apesar de possuir um atraso um pouco maior do que o dos hardware switches (mas com um desempenho superior a LinkSys e HP), a implementação do OpenFlow para o Open vSwitch merece algum destaque. Como esse switch é altamente dependente do hardware onde se encontra instalado, seu desempenho pode ser melhorado se implantado em um ambiente (hardware) com boa capacidade computacional. Como o Open vSwitch suporta uma grande quantidade de regras e oferece suporte a todas operações de reescrita, ele é uma opção interessante para implantação em larga escala, caso o hardware empregado minimize os atrasos.

5.5 DISCUSSÃO SOBRE BALANCEAMENTO DE CARGA UTILIZANDO IMPLEMENTAÇÕES EM HARDWARE

A partir dos resultados encontrados nos testes e da discussão levantada na seção anterior sobre os problemas e benefícios encontrados com o uso do protocolo OpenFlow 1.0 para os switches avaliados neste trabalho, pode-se iniciar uma discussão sobre qual a melhor maneira de implementar um balanceamento de carga em SDN em uma rede real, utilizando os hardware switches estudados. Vale ressaltar que os hardware switches avaliados são os Extreme x440 e x460, HP, Mikrotik, Datacom e a placa NetFPGA. Em primeiro lugar serão mostradas as limitações encontradas para a realização do balanceamento de carga

utilizando os switches avaliados e, em seguida, serão apresentadas algumas soluções para contornar essas limitações.

Para realizar esse balanceamento de carga da forma mais simples possível, de maneira bem semelhante ao apresentado no Capítulo 4 (com exceção do switch, que era uma implementação em software naquele caso), ele deve ser baseado em endereçamento IP e não em configurações de *tunneling* ou em modificações para ser realizado em camada MAC. Dessa forma, alguns fatores precisam ser considerados para que se possa analisar a viabilidade do uso desses hardware switches para realizar tal tarefa.

Realização de *matches* e latência de pacotes. A maioria dos switches avaliados é capaz de realizar *matches* sobre todos os atributos definidos pelo protocolo OpenFlow 1.0, com exceção dos switches HP e Mikrotik, que apresentaram problemas na realização de *matches* com o atributo de identificação da VLAN (dl_vlan). Além disso, os atrasos obtidos na realização das operações de *match* são bastante similares aos atrasos obtidos em seus modos *legacy* (com exceção feita ao switch HP, que apresentou uma perda de desempenho significativa entre os dois modos). Isso significa dizer que a maioria dos hardware switches avaliados, considerando apenas a latência dos pacotes, poderia ser usada em uma SDN, uma vez que a realização de *matches* praticamente não acarretou sobrecarga na operação de encaminhamento de pacotes. E mais, essas operações de *match* apresentam um bom desempenho, independentemente da quantidade de atributos utilizados para realizar *matches*, da quantidade e da disposição das regras instaladas na tabela de fluxos e do tipo de memória em que essas regras são armazenadas.

Quantidade de regras suportadas. A maioria dos hardware switches avaliados suporta uma quantidade de regras muito inferior em seus modos OpenFlow, quando comparado ao número de regras suportadas em seus modos *legacy*. A exceção é feita ao switch HP, que apresenta um aumento considerável no número de regras suportadas, devido à tabela em software ativada pela utilização do protocolo OpenFlow. Essa limitação da quantidade de regras é uma restrição muito importante e de impacto significativo, ainda mais se considerada a natureza do serviço de balanceamento de carga, que consiste em manipular uma grande quantidade de requisições e redistribui-las. Dessa forma, quanto maior for a capacidade de armazenamento de regras, melhor será a operação de balanceamento de carga. Portanto, nesse caso, a utilização do protocolo OpenFlow para os switches avaliados trouxe uma desvantagem considerável.

Reescrita de campos do cabeçalho. Embora a especificação do protocolo OpenFlow 1.0 considere a reescrita de campos de cabeçalho como sendo uma operação opcional, a sua implementação pode facilitar o esquema de balanceamento de carga. Os hardware switches avaliados disponibilizam apenas algumas operações de reescrita de cabeçalho. Enquanto a operação de reescrita de endereços MAC é suportada por todos os switches avaliados, a reescrita de endereços IP, utilizada para realizar o balanceamento em

camada IP, é suportada apenas pelos software switches e pelo switch HP.

Instalação de regras no switch OpenFlow. Quando o controlador envia ordens para instalação de regras no switch de uma maneira proativa, ou seja, quando as regras já estão pré-definidas estaticamente no código do controlador, o switch é capaz de processar essas ordens e realizar essa instalação de forma bem rápida. Quando são poucas regras, a instalação de todas elas é imediata; quando são definidas quantidades maiores de regras, há um atraso maior devido à sobrecarga de comandos, mas, ao final, todas as regras são instaladas. Contudo, o problema ocorre quando as regras são instaladas de modo reativo, ou seja, quando a regra é instalada dinamicamente, de acordo com os atributos fornecidos pelo pacote que acaba de chegar ao switch. Seguindo esse modo de instalação, existe uma demora maior entre o momento da chegada de um novo fluxo e a instalação da regra correspondente. De fato, em uma SDN, o atraso do primeiro pacote de um fluxo é sempre maior do que o atraso dos demais pacotes do mesmo fluxo, justamente porque esse pacote é levado ao controlador, que vai processá-lo e instalar uma regra no switch de acordo com os atributos do pacote.

Mas o problema principal a ser observado neste tópico se refere a outro fato. À medida que chegam fluxos de diferentes origens, que, por consequência, levam a instalação de regras distintas, uma série de mensagens de instalação de regras é enviada ao switch pelo controlador. Todos os hardware switches avaliados, ao receberem pacotes de diferentes origens em um curto espaço de tempo, não foram capazes de lidar com tantas requisições de instalação de regras ao mesmo tempo. Isso levou os switches a não instalar todas as regras, e, nos piores casos, ao travamento do equipamento. Para o switch Extreme x460, por exemplo, a chegada de 3 pacotes em um segundo já foi o suficiente para gerar problemas de processamento, que culminaram com a falha do switch. Em um ambiente de balanceamento de carga em SDN, no qual a todo o momento chegam requisições, às vezes em rajadas, a falha de um switch por não conseguir processar as informações enviadas pelo controlador é um fato bastante grave e limitador, que pesa negativamente ao OpenFlow.

Processamento de múltiplos comandos simultaneamente. Essa observação se relaciona muito à observação anterior. Em um switch OpenFlow ocorrem vários eventos, que desencadeiam uma série de ações e trocas de mensagem entre as entidades da SDN. Nesse ambiente dinâmico, sujeito a realização de *matches*, instalação de regras, coletas de dados estatísticos e outros eventos característicos, a estabilidade do comutador é algo desejável e, até mesmo, necessário. Durante os testes de *flowstats* e *portstats*, os switches foram incapazes de lidar simultaneamente com a coleta de dados estatísticos e a instalação de regras estáticas – nem eram regras dinâmicas -, principalmente o switch Mikrotik. Por mais que o atraso nas requisições estatísticas não tenha sido afetado na maioria dos casos, apenas o fato de que, para haver coleta de dados de *flowstats* e *portstats*, deveriam ser instaladas menos regras na tabela de fluxos já indica um severo problema

das implementações OpenFlow dos hardware switches avaliados.

Finalizando, de uma maneira geral, considerando os seis hardware switches avaliados neste trabalho, suas implementações do protocolo OpenFlow 1.0, os resultados obtidos por eles nos testes e as observações do comportamento de cada um, pode-se afirmar que nenhum deles ainda é capaz de realizar balanceamento de carga em ambientes reais, sobretudo em redes em produção. Essas implementações OpenFlow em hardware ainda não atingiram um nível de maturidade suficiente para serem utilizadas em larga escala. Apesar da baixa latência encontrada na maioria dos casos, problemas como implementações incompletas do padrão, baixo número de regras suportadas, funcionamento instável para tabelas quase cheias e problemas de processamento de múltiplos comandos contribuíram muito para a avaliação negativa das implementações OpenFlow dos switches avaliados. Possivelmente a implementação do protocolo OpenFlow em sua versão 1.3 traga algumas melhorias, e uma avaliação dessa nova versão de implementação será um trabalho futuro.

Mesmo assim, se fosse preciso realizar um balanceamento de carga mais experimental, que acomodasse as restrições que são impostas aos planos de dados estudados, algumas ações poderiam ser tomadas. Em primeiro lugar, poderiam ser realizados *matches* que não utilizassem todos os campos definidos pelo protocolo. Dessa forma, com regras menores, poderiam ser armazenadas mais regras em memória. Uma outra melhoria seria remover as regras mais antigas com a utilização dos *timeouts* de regra (*hardware* e *idle timeouts*). Dessa forma, regras associadas a novos fluxos poderiam ser instaladas no lugar de regras menos utilizadas, o que também contribuiria para a diminuição da quantidade de regras instaladas. Mais uma medida tomada poderia ser a utilização de estatísticas dos pontos finais, ou seja, dos servidores, e não das estatísticas fornecidas pelo OpenFlow, o que diminuiria a sobrecarga de informações processadas pelo switch.

Contudo, vale ressaltar que essas medidas são paliativas. Uma melhora significativa nas implementações OpenFlow dos switches avaliados passa por uma reavaliação da capacidade de processamento das informações enviadas.

6 CONCLUSÃO

No presente trabalho foi discutida a viabilidade de se implementar um balanceador de carga OpenFlow em uma rede SDN real, considerando cargas de trabalho mais realistas e as restrições existentes nos equipamentos OpenFlow comerciais atuais. A partir desse fato, foi proposto um modelo de balanceamento de carga em SDN que leva em consideração diferentes perfis de carga mais realistas e que é baseado na utilização de diferentes políticas para a realização do balanceamento: uma política aleatória, uma política no estilo *Round-Robin* e outra baseada na carga de trabalho dos servidores que atendem as requisições. Esse modelo foi implementado e submetido a testes em um ambiente de rede virtual, baseado em máquinas virtuais e no Open vSwitch, com o intuito de ser aprimorado e reproduzido em uma rede operacional real.

Os resultados mostraram que, para cargas tradicionais *Web*, as políticas não apresentaram diferenças significativas de desempenho entre si. Nesse cenário, como as requisições são pequenas e de rápida resposta, não há tempo suficiente para que políticas sofisticadas de balanceamento de carga convirjam para um resultado melhor do que em uma abordagem simples, como a *Round-Robin*. Para cargas *Web* médias, a política que leva em consideração a carga dos servidores *Web* apresentou uma melhora de até 12% no tempo de resposta, quando comparada às demais políticas. Já para cargas *Web* pesadas, a política que leva em consideração a carga de trabalho dos servidores apresentou o pior resultado entre as três políticas consideradas. Nesse cenário, o tempo de reação requisitado pela aplicação *Web* é muito inferior ao tempo de atualização dos dados de carga nos servidores *Web*. Assim, rajadas de requisições são direcionadas momentaneamente a um mesmo servidor que, erroneamente, fora interpretado como o menos sobrecarregado pelo controlador SDN.

No entanto, antes que se procedesse à reprodução desse modelo de balanceamento em um ambiente real, foi necessário realizar também uma avaliação da qualidade de diferentes implementações OpenFlow, compostas por alguns hardware switches comerciais e algumas implementações *open source* de software switches, a fim de se introduzir uma discussão sobre a possibilidade ou não de uma adoção efetiva da tecnologia OpenFlow para a realização de balanceamento de carga em redes SDN de produção. Essa avaliação focou no desempenho exclusivo dos switches, descartando qualquer influência por parte do controlador da rede.

As implementações OpenFlow foram avaliadas através de seus desempenhos na realização de diferentes tipos de *match* em campos de cabeçalho, na reescrita de cabeçalho de pacotes, no processamento de múltiplos comandos OpenFlow, na instalação de regras e na realização de *matches* com diferentes quantidades de regras instaladas nas tabelas de fluxos. Além disso, foram realizadas comparações entre os desempenhos dos switches

operando em seu modo OpenFlow e seu modo normal, a fim de identificar vantagens ou gargalos na utilização do protocolo.

Os resultados dessa avaliação mostraram que os hardware switches implementam o protocolo OpenFlow de maneiras diferentes. Alguns o implementam de fato em hardware, outros simplesmente realizam a execução de um software switch adaptada aos mecanismos internos do hardware. E mais, observou-se que, no geral, software switches são mais compatíveis com a especificação do protocolo OpenFlow e suportam grandes tabelas de fluxos, porém o seu atraso no encaminhamento dos pacotes pode chegar a uma ordem de grandeza mais lenta do que a dos atrasos em hardware switches. No entanto, hardware switches geralmente não suportam grandes tabelas de fluxos e são incapazes de realizar reescritas de cabeçalho de pacotes no nível IP, o que dificulta a realização de muitas tarefas, como o balanceamento de carga. Além disso, concluiu-se que as implementações OpenFlow dos hardware switches avaliados ainda não atingiram um nível de maturidade suficiente para serem utilizadas em larga escala. Apesar de, na maioria dos casos, terem sido encontradas latências similares às do modo *legacy*, as implementações OpenFlow em hardware apresentaram problemas como implementações incompletas do padrão, baixo número de regras suportadas, funcionamento instável para tabelas quase cheias e problemas de processamento de múltiplos comandos, o que contribuiu muito para a avaliação negativa dessas implementações.

6.1 TRABALHOS FUTUROS

Como trabalhos futuros, podem ser realizados alguns avanços nos dois problemas apresentados neste trabalho. Em relação ao mecanismo de balanceamento de carga, pode-se aprimorá-lo a partir de uma série de opções. Uma primeira opção seria melhorar o esquema de medição de carga disponibilizado pelo servidor, de tal forma que ele se torne mais rápido e reaja a novas cargas tão rápido quanto mudam as condições da rede. Uma outra opção seria mesclar as diferentes políticas de balanceamento de carga e adicionar alguns mecanismos para evitar rajadas a um servidor. Mais uma opção seria realizar o balanceamento com mais servidores atendendo a requisições e medir o impacto dessa adição nos tempos de resposta aos clientes. Além das possibilidades já levantadas, poderia ser adicionado um valor "delta" à carga do servidor a cada novo fluxo redirecionado, a fim de aumentar a influência do fluxo no processo de medição da carga e diminuir a letargia na mudança desse valor. Pode ser considerado também verificar outros parâmetros como uso de memória, quantidade de operações de I/O e bloqueios, que são aspectos muito importantes em cargas Web, e utilizá-los juntamente ao valor de CPU. Também pode ser tentado um esquema que preveja a carga dos fluxos de uma forma mais inteligente, como, por exemplo, avaliando o histórico dos fluxos recentes de toda a rede ou o histórico dos fluxos anteriores de um determinado *host*. Por fim, pretende-se realizar o balanceamento

de carga através de um cenário mais realista, utilizando cargas de servidores *Web* reais.

Já em relação à avaliação dos planos de dados OpenFlow, pretende-se expandir a lista de switches avaliados através de contatos com a comunidade de pesquisa. Como exemplo, planeja-se avaliar, em breve, os switches Pica8, Huawei e Juniper. Também se pretende avaliar a conformidade dos switches com novas versões do protocolo OpenFlow, sobretudo a versão 1.3, já suportada por alguns dos switches avaliados. Por fim, deseja-se realizar, de fato, o balanceamento de carga utilizando um hardware switch OpenFlow com menos restrições, adaptando o mecanismo proposto às limitações deste switch, a fim de obter os primeiros resultados para comparar aos já obtidos na arquitetura virtual com software switches.

REFERÊNCIAS

M. APPELMAN and M. DE BOER. Performance analysis of openflow hardware. Semester thesis project report, University of Amsterdam, Fevereiro 2012.

S. BANERJEE and K. KANNAN. Tag-in-tag: Efficient flow table management in sdn switches. In *10th International Conference on Network and Service Management (CNSM) and Workshop*, pages 109–117, Nov 2014.

M. BANSAL, J. MEHLMAN, S. KATTI, and P. LEVIS. Openradio: A programmable wireless dataplane. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 109–114, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0.

A. BIANCO, R. BIRKE, L. GIRAUDO, and M. PALACIN. Openflow switching: Data plane performance. In *IEEE International Conference on Communications*, pages 1–5, 2010.

P. BOSSHAART, D. DALY, G. GIBB, M. IZZARD, N. McKEOWN, J. REXFORD, C. SCHLESINGER, D. TALAYCO, A. VAHDAT, G. VARGHESE, and D. WALKER. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014. ISSN 0146-4833.

A. BOVENZI, D. COTRONEO, R. PIETRANTUONO, and S. RUSSO. Workload characterization for software aging analysis. In *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pages 240–249, 2011.

M. CAESAR, N. FEAMSTER, J. REXFORD, A. SHAIKH, and J. MERWE. Design and implementation of a routing control platform. In *Proceedings of the 2nd USENIX NSDI*, Boston, MA, USA, may 2005.

M. CASADO, M. J. FREEDMAN, J. PETTIT, J. LUO, N. GUDE, N. McKEOWN, and S. SHENKER. Rethinking enterprise network control. In *IEEE/ACM Trans. Netw.*, volume 17, pages 1270–1283, Piscataway, NJ, USA, 2009.

L. C. COSTA, A. B. VIEIRA, E. B. SILVA, D. F. MACEDO, G. N. GOMES, L. H. A. CORREIA, and L. F. M. VIEIRA. Avaliação de desempenho de planos de dados OpenFlow. In *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBRC 2016*, 2016.

A. CURTIS, J. MOGUL, J. TOURRILHES, P. YALAGANDULA, P. SHARMA, and S. BANERJEE. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265. ACM, 2011. ISBN 978-1-4503-0797-0.

DCAN. Devolved control of atm networks. 1995. Disponível em: http://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/. Acessado em: Janeiro de 2016.

N. FEAMSTER, J. REXFORD, and E. ZEGURA. The road to SDN: an intellectual history of programmable networks. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 87–98. ACM New York, NY, USA, 2014.

E. L. FERNANDES and C.E. ROTHENBERG. OpenFlow 1.3 Software Switch. In *XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBRC 2014*, 2014.

ForCES. ForCES protocol specification. 2004. Disponível em: https://tools.ietf.org/html/draft-ietf-forces-protocol-00. Acessado em: Janeiro de 2016.

P. GORANSSON and C. BLACK. *Software Defined Networks: A Comprehensive Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2014. ISBN 012416675X, 9780124166752.

K. GOSEVA-POPSTOJANOVA, S. MAZIMDAR, and A. D. SINGH. Empirical study of session-based workload and reliability for web servers. In *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, pages 403–414, 2004.

K. GREENE. TR10: Software-defined networking. *MIT Technology Review*, 2009. Disponível em: http://www2.technologyreview.com/article/412194/tr10-software-defined-networking/. Acessado em: Janeiro de 2016.

GSMP. General switch management protocol (GSMP) v3 specification. 2002. Disponível em: https://www.ietf.org/rfc/rfc3292.txt. Acessado em: Janeiro de 2016.

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

N. HANDIGOL, S. SEETHARAMAN, M. FLAJSLIK, N. McKEOWN, and R. JOHARI. Plug-n-serve: load-balancing web traffic using openflow. In *Proceedings of demo at ACM SIGCOMM*, 2009.

F. HU, Q. HAO, and K. BAO. A survey on software-defined network and openflow: From concept to implementation. *IEEE Communications Surveys and Tutorials*, 16(4): 2181–2206, 2014.

M. JARSCHEL, T. ZINNER, T. HOSSFELD, P. TRAN-GIA, and W. KELLERER. Interfaces, attributes, and use cases: A compass for sdn. volume 52, pages 210–217, 2014.

M. JEON, Y. KIM, J. HWANG, J. LEE, and E. SEO. Workload characterization and performance implications of large-scale blog servers. *ACM Transactions on the Web*, 6 (4), 2012.

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

D. KREUTZ, F. M. V. RAMOS, P. E. VERISSIMO, C. E. ROTHENBERG, S. AZODOLMOLKY, and S. ULHIG. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

T. V. LAKSHMAN, T. NANDAGOPAL, R. RAMJEE, K. SABNANI, and T. WOO. The softrouter architecture. In *Proceedings of the 3nd ACM Workshop on Hot Topics in Networks (Hotnets-III)*, San Diego, CA, USA, nov 2004.

A. LARA, A. KOLASANI, and B. RAMAMURTHY. Network innovation using openflow: A survey. In *Communications Surveys & Tutorials, IEEE*, volume 16, pages 493–512, 2014.

D. LEVIN, A. WUNDSAM, B. HELLER, N.ikhil HANDIGOL, and A. FELDMANN. Logically centralized? state distribution trade-offs in software defined networks. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 1–6, New York, NY, USA, 2012. ISBN 978-1-4503-1477-0.

D. F. MACEDO, D. GUEDES, L. F. M. VIEIRA, M. A. M. VIEIRA, and M. NOGUEIRA. Programmable networks-from software-defined radio to software-defined networking. *IEEE Communications Surveys and Tutorials*, 17(2):1102–1125, 2015.

N. McKEOWN, T. ANDERSON, H. BALAKRISHNAN, G. PARULKAR, L. PETERSON, J. REXFORD, S. SHENKER, and J. TURNER. Openflow: Enabling innovation in campus networks. In *SIGCOMM Comput. Commun. Rev.*, volume 38, pages 69–74, New York, NY, USA, 2008.

D. MOSBERGER and T. JIN. httperf: A tool for measuring web server performance. In *Proceedings of the 1998 Internet Server Performance Workshop*, volume 26, pages 31–37, 1998.

P. NAGPURKAR, W. HORN, U. GOPALAKRISHNAN, N. DUBEY, J. JANN, and P. PATTINAIK. Workload characterization of selected jee-based web 2.0 applications.

In *Proceedings of IEEE International Symposium on Workload Characterization 2008*, pages 109–118, 2008. ISBN 978-1-4244-2777-2.

B. A. A. NUNES, M. MENDONCA, X. NGUYEN, K. OBRACZKA, and T. TURLETTI. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 16(3):1617–1634, 2014.

ONF. What is ONF? 2016. Disponível em: https://www.opennetworking.org /images/stories/downloads/about/onf-what-why.pdf. Acessado em: Janeiro de 2016.

B. PFAFF, J. PETTIT, T. KOPONEN, K. AMIDON, M. CASADO, and S. SHENKER. Extending networking into the virtualization layer. In *8th ACM Workshop on Hot Topics inNetworks (HotNets-VIII). New York, NY, USA*, 2009.

S. POLFLIET, F. RYCKBOSCH, and L. EECKOUT. Studying hardware and software trade-offs for a real-life web 2.0 workload. In *ICPE'12 - Proceedings of the 3rd Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 181–192, 2012.

POX. Pox wiki. 2015. Disponível em: https://openflow.stanford.edu/display/ONL/POX +Wiki. Acessado em: Novembro de 2015.

M. QILIN and S. WEIKANG. A load balancing method based on sdn. In *2015 Seventh International Conference on Measuring Technology and Mechatronics Automation*, pages 18–21, June 2015.

Y. QU, S ZHOU, and V. K. PRASANNA. Scalable many-field packet classification on multi-core processors. In *2013 25th International Symposium on Computer Architecture and High Performance Computing*, pages 33–40, Oct 2013.

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

C. RODRIGUES, L. COSTA, M. A. VIEIRA, L. F. VIEIRA, D. F. MACEDO, and A. B. VIEIRA. Avaliação de balanceamento de carga web em redes definidas por software. In *XXXIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos, SBRC 2015*, 2015.

C. ROTSOS, N. SARRAR, S. UHLIG, R. SHERWOOD, and A. W. MOORE. *OFLOPS: An open framework for OpenFlow switch evaluation*, volume 7192 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2012.

S. SEZER, S. SCOTT-HAYWARD, P. CHOUHAN, B. FRASER, D. LAKE, J. FINNEGAN, N. VILJOEN, M. MILLER, and N. RAO. Are we ready for SDN? Implementation challenges for software-defined networks. In *IEEE Communications Magazine*, volume 51, pages 36–43, 2013.

R. SHERWOOD, G. GIBB, K.-K. YAP, G. APPENZELLER, M. CASADO, N. McKEOWN, and G. PARULKAR. Flowvisor: A network virtualization layer. In *Technical Report Openflow-tr-2009-1, OpenFlow*. Stanford University, 2009.

S. SHIRALI-SHAHREZA and Y. GANJALI. Rewiflow: Restricted wildcard openflow rules. *SIGCOMM Comput. Commun. Rev.*, 45(5):29–35, September 2015. ISSN 0146-4833.

D. SÜNNEN. Performance evaluation of openflow switch. Semester thesis project report, Swiss Federal Institute of Technology Zurich, Fevereiro 2011.

H. SONG. Protocol-oblivious forwarding: Unleash the power of sdn through a future-proof forwarding plane. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 127–132, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2178-5.

L. SURESH, J. SCHULZ-ZANDER, R. MERZ, A. FELDMANN, and T. VAZAO. Towards programmable enterprise wlans with odin. In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, HotSDN '12, pages 115–120, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1477-0.

D.L. TENNENHOUSE, J.M. SMITH, W.D. SINCOSKIE, D.J. WETHERALL, and G.J. MINDEN. A survey of active network research. *Communications Magazine, IEEE*, 35 (1):80–86, Jan 1997. ISSN 0163-6804. doi: 10.1109/35.568214.

H. UPPAL and D. BRANDON. Openflow based load balancing. In *Proceedings of CSE561: Networking Project Report*. University of Washington, Spring, 2010.

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

A. WILLIAMS, M. ARLITT, C. WILLIAMSON, and K. BARKER. Web workload characterization: Ten years later. In *Web Content Delivery*, volume 2 of *Web Information Systems Engineering and Internet Technologies Book Series*, pages 3–21. Springer US, 2005. ISBN 978-0-387-24356-6.

W. XIA, Y. WEN, C. H. FOH, D. NIYATO, and H. XIE. A survey on software-defined networking. *IEEE Communications Surveys and Tutorials*, 17(1):27–51, 2015.

B. YAN, Y. XU, H. XING, K. XI, and H. CHAO. Cab: A reactive wildcard rule caching system for software-defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 163–168, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2989-7.

Y. ZHOU, L. RUAN, L. XIAO, and R. LIU. A method for load balancing based on software defined network. In *Advanced Science and Technology Letters*, volume 45, pages 43–48, 2014.

N. ZILBERMAN, P. M. WATTS, C. ROTSOS, and A. W. MOORE. Reconfigurable network systems and software-defined networking. *Proceedings of the IEEE*, 103(7): 1102–1124, 2015.