UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Thiago Moratori Peixoto

Aumentando a resiliência em SDN quando o Plano de Controle se encontra sob ataque

Juiz de Fora

2017

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Thiago Moratori Peixoto

Aumentando a resiliência em SDN quando o Plano de Controle se encontra sob ataque

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Alex Borges Vieira

Juiz de Fora

2017

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Peixoto, Thiago Moratori.
Aumentando a resiliência em SDN quando o Plano de Controle se encontra sob ataque / Thiago Moratori Peixoto. -- 2017.
67 f. : il.

Orientador: Alex Borges Vieira
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós Graduação em Ciência da Computação, 2017.

1. SDN. 2. Segurança. 3. Desempenho. I. Vieira, Alex Borges, orient. II. Título.

Thiago Moratori Peixoto

Aumentando a resiliência em SDN quando o Plano de Controle se encontra sob ataque

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 31 de Agosto de 2017.

BANCA EXAMINADORA

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

Prof. D.Sc. Edelberto Franco Silva
Universidade Federal de Juiz de Fora

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

A Deus em primeiro lugar. Aos meus pais, namorada e amigos pelo apoio incondicional.

AGRADECIMENTOS

Primeiramente gostaria de agradecer à Deus, sem Ele, me dando força, calma, paz e plenitude, nada nessa vida seria possível e essa jornada não faria sentido.

Agradeço especialmente aos meus pais, Antônio e Ruth pelo amor, carinho, proteção e apoio incondicional, tanto psicológico quanto financeiro!

Aos meus irmãos Mayara e Bernardo pela presença, pelo ombro amigo, pelos conselhos e pelas implicâncias, que só quem tem irmãos sabe, são as maiores demonstrações de afeto.

À minha namorada Luana pelo amor e compreensão nas horas mais complicadas, pelas palavras de conforto e pela perspectiva de um futuro melhor.

Também fica o meu muito obrigado ao meu orientador Alex pela compreensão nos momentos passados, pela força quando precisei trocar de tema, pelo incentivo e principalmente pelos puxões de orelha, muitos deles aliás.

Muito obrigado também à minha segunda família, pessoal do forró Pé Descalço, que por vários anos tem me ajudado a respirar e manter a sanidade mental.

Aos meus colegas de mestrado Laura Lima, Bruno Marques e Marco Aurélio Freesz, fica o agradecimento pelas horas de estudo, pelas risadas, pelas conversas, confissões e apoios diversos, técnicos ou não, tudo isso fez com que o processo de aprendizado fosse enriquecido de maneira substancial. Um agradecimento especial ao Wallace que por diversas vezes me auxiliou tanto com apoio moral, como intelectual e muitas vezes técnico.

Por último, mas não menos importante, gostaria de agradecer à todos que contribuíram de maneira direta ou indireta para que eu entrasse no mestrado e concluísse mais essa etapa na vida.

"You shout it loud
But I can't hear a word you say
I'm talking loud not saying much
I'm criticized but all your bullets ricochet
You shoot me down, but I get up
I'm bulletproof, nothing to lose
Fire away, fire away
Ricochets, you take your aim
Fire away, fire away
You shoot me down but I won't fall
I am titanium
You shoot me down but I won't fall
I am titanium [...]"
Trecho da Música "Titanium",
por David Guetta e Sia

RESUMO

SDN (Software Defined Networks) é um paradigma de redes que permite aos operadores gerenciar os elementos de rede usando software que executa em um servidor externo sendo possível fazer a divisão do plano de controle e do plano de dados. Contudo, como em toda tecnologia, principalmente as mais incipientes, existem problemas de segurança e vulnerabilidades a serem investigadas. Por exemplo interrupções de rede causadas por erro humano. Um administrador que configura um controlador de maneira errada pode facilmente incorrer em diminuição de desempenho da rede, mesmo que o controlador funcione corretamente e não haja problemas com as regras. A esse cenário dá-se o nome de *problema do administrador mal configurado*, onde um administrador configura de maneira equivocada um controlador em plenas capacidades de maneira tal que prejudica o desempenho da rede. O trabalho proposto nessa dissertação tem dois objetivos: primeiro, avaliar o impacto no desempenho do plano de dados decorrentes de problemas causados por um sistema mal configurado; e segundo, propor, através do desenvolvimento de um módulo adjacente ao controlador, medidas para mitigar esses impactos. Os resultados obtidos por experimentação em um cenário realista mostram que a utilização desse módulo é capaz de melhorar o desempenho médio do sistema em 4,82%.

Palavras-chave: SDN. Segurança. Desempenho.

ABSTRACT

SDN (Software Defined Networks) is a network paradigm that allow operators to manage network elements using software that executes on an external server making possible to divide the control plane from the data plane. However, as in all technologies, mainly the newer ones, there are security problems and vulnerabilities to be investigated. Network outages caused by human error for instance. An administrator that misconfigures a controller can easily reduce the network performance, even if the controller works properly and there are no errors with the installed rules. This scenario is also called *the misconfigured administrator problem*, where the administrator misconfigures a full capacity controller in a way that impairs the network performance. The work proposed in this dissertation has two objectives: first, to evaluate the impact on the performance of the data plan resulting from problems caused by a misconfigured system; And second, propose, through the development of a module adjacent to the controller, measures to mitigate these impacts. The results obtained by experimentation in a realistic scenario show that the use of this module is capable of improving the average performance of the system by 4,82%.

Keywords: SDN. Security. Performance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API *Application Programming Interface*

ARP *Address Resolution Protocol*

ATM *Asynchronous Transfer Mode*

ASICs *Application Specific Integrated Circuits*

CDF *Cumulative Distribution Function*

DCAN *Devolved Control of ATM Networks*

DoS *Denial of Service*

DPI *Deep Packet Inspection*

ForCES *Forwarding and Control Element Separation*

GSMP *General Switch Management Protocol*

IP *Internet Protocol*

IP/MPLS *Internet Protocol / Multi Protocol Label Switching*

MAC *Media Access Control*

OF *OpenFlow*

OPENSIG *OpenSignaling*

ONF *Open Networking Foundation*

QoS *Quality of Service*

RCP *Routing Control Platform*

SAG *Scenario Attack Graph*

SDN *Software Defined Network*

TCAM *Ternary Access Content Memory*

TCP *Transmission Control Protocol*

TE *Traffic Engineering*

UDP *User Datagram Protocol*

VM *Virtual Machine*

WAN *Wide Area Network*

SUMÁRIO

1 INTRODUÇÃO

O conceito de Redes definidas por software (Software Defined Networks - SDN) é um paradigma que permite, aos operadores de redes, gerenciar os elementos de rede de maneira diferente da convencional. De acordo com (MACEDO et al., 2015) e (MATSUMOTO et al., 2014), com ele é possível fazer a divisão do plano de controle e do plano de dados. Dessa maneira, o processo de controle se encontra centralizado no elemento de rede chamado Controlador enquanto os switches simplesmente realizam a operação de encaminhar os pacotes de acordo com as decisões tomadas. Decisões de roteamento, as chamadas regras de fluxo, são instaladas nos switches e, para cada fluxo de entrada, é realizada uma operação de verificação dos campos do pacote com o intuito de tomadas de decisão, seja para encaminhamento ou para bloqueio de tráfego. Com essa divisão, torna-se possível, e facilitado, o desenvolvimento e experimentação de novos protocolos, topologias, contextos e tecnologias relacionados ao ambiente de redes de computadores.

Contudo, como em toda tecnologia, principalmente as mais incipientes, existem problemas e vulnerabilidades a serem investigados. Na área de resiliência por exemplo, (HU et al., 2014) define dois termos para divisão de áreas: *security*, aqui chamado de segurança e *safety*, aqui chamado de proteção.

Os problemas de segurança são aqueles que oriundam de ações externas ao sistema. Consiste da existência de maus elementos, invasores, que visam, através de intervenções maliciosas, prejudicar o funcionamento normal da rede. Já os problemas de proteção, são aqueles que residem internos à própria rede, como falhas de configuração ou de equipamentos por exemplo. Dessas duas áreas propostas por (HU et al., 2014), nosso trabalho visa focar problemas que podem ser abordados na segunda, a de proteção.

Dado o contexto da área de proteção, observamos, em um estudo da (NETWORKS, 2008), que 50 a 80 % de interrupções de rede são causadas por erro humano, dados estes que despertaram o interesse de uma investigação mais profunda. (MATSUMOTO et al., 2014) diz que um administrador que configura um controlador de maneira errada pode facilmente incorrer em diminuição de desempenho da rede, mesmo que o controlador funcione corretamente e não haja problemas com as regras. A esse cenário (MATSUMOTO et al., 2014) da o nome de Problema do Administrador Malicioso, onde um administrador

configura de maneira equivocada um controlador em plenas capacidades de maneira tal que prejudica o desempenho da rede.

Nesse problema citado, temos duas considerações a serem feitas: (i) qual é de fato o impacto gerado pela má configuração do sistema? e (ii) como aumentar a resiliência do sistema com o intuito de diminuir esse impacto?

Levando em conta essas duas considerações, o trabalho aqui proposto visa investigá-las e tentar respondê-las. Tais investigações e respostas são realizadas e obtidas através de duas etapas. Na primeira analisamos o impacto no desempenho do sistema, em ambientes real e virtual. Em seguida, na segunda, fazemos a proposta de um módulo sensível ao decréscimo de desempenho e que é capaz de contornar dada situação uma vez que verificada a sua ocorrência.

Nossos testes mostram que a presença de um componente mal configurado no controlador da rede, de fato, é capaz de diminuir o desempenho da rede através da diminuição da vazão de dados causada pelo aumento do atraso no envio de pacotes. Através de experimentação em condições realistas, usando máquinas reais que emulavam um switch virtual através da tecnologia OpenVSwitch, mostramos também que tal situação é contornável e pode ser amortizada com a utilização do nosso módulo proposto. Houve uma melhora de aproximadamente 4,82% nos resultados de desempenho do sistema, quando suas médias foram comparadas às do pior caso, sem o uso do nosso módulo.

Ao se considerar o Problema do Administrador Malicioso em SDN, (MATSUMOTO et al., 2014) provê uma solução através da utilização de um módulo controlador chamado Fleet, contudo, o ambiente abordado é diferente, neste caso é considerado um ambiente com múltiplos controladores e não é feita nenhuma análise de impacto no Plano de Dados.

O trabalho proposto por essa dissertação dá uma visão complementar à de (MATSUMOTO et al., 2014) de como mitigar esse problema. Aqui mostramos uma maneira de implementar um componente do controlador POX para contornar o problema da diminuição de desempenho quando o sistema é atacado, para um ambiente de controlador único, além de fornecer uma visão do desempenho das redes SDN em diferentes cenários (virtual e real) com diferentes interferências.

Pela falta de equipamentos de switches de hardware necessários para a execução de testes, o módulo proposto foi utilizado apenas para testes em equipamentos que emulavam um switch virtual (OpenVSwitch), foram utilizadas 5 máquinas para esse propósito, todas

elas com sistema operacional baseado em Linux (Lubuntu), com 512 MBytes de memória RAM e processador Celeron de 2.4 GHz (com apenas um núcleo).

1.1 CONTRIBUIÇÕES DA DISSERTAÇÃO

Como contribuições do presente trabalho, temos o seguinte:

- **Avaliação do impacto no plano de dados de diferentes tipos de interferência causadas pelo plano de controle mal configurado:** Existem diferentes tipos de interferência que podem ser causadas na má configuração do controlador, em nosso trabalho fazemos a verificação e avaliação desses três tipos possíveis de intervenção (inserção, remoção e alteração de regras) comparando-os com intuito de verificar qual deles gera o maior impacto. O trabalho de (KUŹNIAR et al., 2015) realiza testes de desempenho similares aos utilizados aqui, contudo não fizeram a separação de cada tipo de interferência, e nem, por consequência, a análise comparativa entre cada uma delas.

- **Criação de um componente simples de proteção:** É feita uma proposta de utilização de um componente sensível à variação de vazão de dados que, ao verificar uma variação anormal da mesma, toma medidas de alteração de rota para contornar o ataque e manter o desempenho do Plano de Dados.

1.2 ORGANIZAÇÃO DA DISSERTAÇÃO

A dissertação foi dividida da seguinte maneira: o capítulo 1 faz uma introdução ao tema de pesquisa, explicita os problemas da área e apresenta as propostas da dissertação, o capítulo 2 introduz conceitos relativos à grande de área de SDN contemplando um histórico de surgimento, modelos de arquitetura e objetos dos contextos de software e hardware da mesma. No capítulo 3 serão abordados conceitos sobre o que se entende por desempenho quando se trata de SDN, principalmente para o Plano de Dados, alguns desafios na área e o que já se tem feito no estado-da-arte, bem como explicitará conceitos referentes à grande área de segurança em se tratando de SDN, bem como apresentará trabalhos relacionados que propõem contemplar o assunto. Serão apresentados, no capítulo 4, a metodologia de avaliação de desempenho, bem como os resultados numéricos obtidos. A proposta de

mitigação do Problema do Administrador Mal configurado será contemplada no capítulo 5. Por fim, no capítulo 6, serão explicitados os conhecimentos adquiridos com a realização do trabalho e sugestões de trabalhos futuros.

2 REDES DEFINIDAS POR SOFTWARE

Neste capítulo fazemos uma apresentação do paradigma SDN contemplando o histórico de seu surgimento, sua arquitetura básica, exemplos de utilizações comuns do mesmo, bem como de contextos relacionados à ele, no caso, principalmente, o protocolo OpenFlow, ferramenta que permite a implementação do conceito de redes definidas por software.

2.1 VISÃO GERAL

A Engenharia de Tráfego (*Traffic Engineering* - TE) é um dos mecanismos de análise, predição e regulamentação dinâmica de dados em uma rede. Ela foi, e tem sido amplamente utilizada em redes de dados, no passado e presente, como ATM e IP/MPLS por exemplo.

Entretanto, essas redes, bem como suas respectivas ferramentas de TE, não são favoráveis para os novos paradigmas de rede e seus gerenciamentos que vem surgindo. Duas são as principais razões, primeiramente, as aplicações de Internet de hoje necessitam de uma arquitetura de rede de reaja em tempo real e que seja escalável para um conjunto grande de tráfego de dados, essa arquitetura deve ser capaz de classificar vários tipos de tráfego de vários tipos de aplicações e ser capaz de prover, dessa maneira, um tipo de serviço específico para cada um desses tipos de tráfego em um período muito curto, na ordem de milisegundos (ms); segundo, com o rápido crescimento de computação em nuvem e, consequentemente, alta demanda por *data centers* de larga escala, há a necessidade de uma nova técnica de gerência de redes capaz de gerir recursos de maneira tal que permita um melhor desempenho dos sistemas(AKYILDIZ et al., 2014).

Um recente paradigma, o SDN (*Software Defined Network* - Rede Definida por Software)(AGARWAL et al., 2013), separa o plano de dados do plano de controle e provê às aplicações uma visão centralizada dos estados da rede distribuída. Esse modelo inclui três camadas e interações mostradas na Figura 2.1.

Uma visão geral sobre a arquitetura SDN pode ser explicada da seguinte forma: o plano de controle gerencia os estados da rede através de políticas de rede programáveis, a camada de aplicação provê serviços de rede comuns como roteamento, controle de acesso, *multicasting*, ciência do uso de energia, segurança, QoS, entre outros, através do uso de

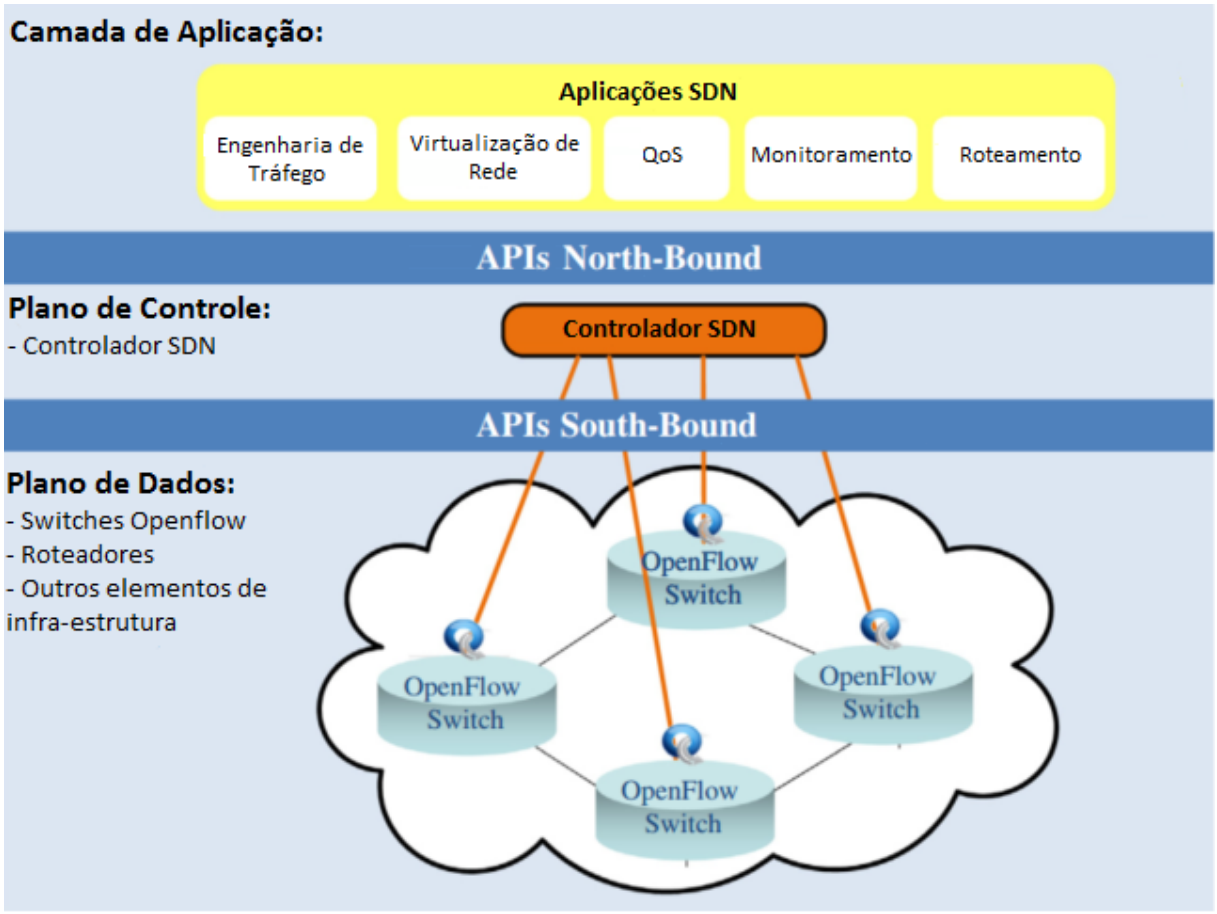

Figura 2.1: Arquitetura de uma rede SDN. Adaptado de: (AGARWAL et al., 2013)

interfaces, as chamadas *APIs North-bound*, representadas na Figura 2.1. O encaminhamento dos pacotes é feito no plano de dados através de switches, roteadores ou outros equipamentos de comutação. Esses, por sua vez, se comunicam com a camada de controle através de *APIs South-bound*, sendo o OpenFlow (OF) a API padrão. É essa API que possibilita que os switches tenham acesso ao plano de dados e sejam capazes de executar programas que fazem a verificação para tomada de decisão de encaminhamento ou não do pacote. Em resumo, através da interação das três camadas, o paradigma SDN permite a visualização e controle global de redes potencialmente grandes e complexas provendo uma plataforma de controle unificada para gestão de fluxos.

Na seção 2.3 serão apresentados, com maiores detalhes, cada um dos componentes básicos da arquitetura SDN.

2.2 LINHA DO TEMPO

A conceito de se adicionar mais recursos para o controle das redes de computadores permitindo maior flexibilidade para experimentação, desenvolvimento e inovação, é mais antigo que o próprio conceito de SDN (COSTA, 2016). A Figura 2.2 mostra uma linha do tempo com os predecessores do paradigma.

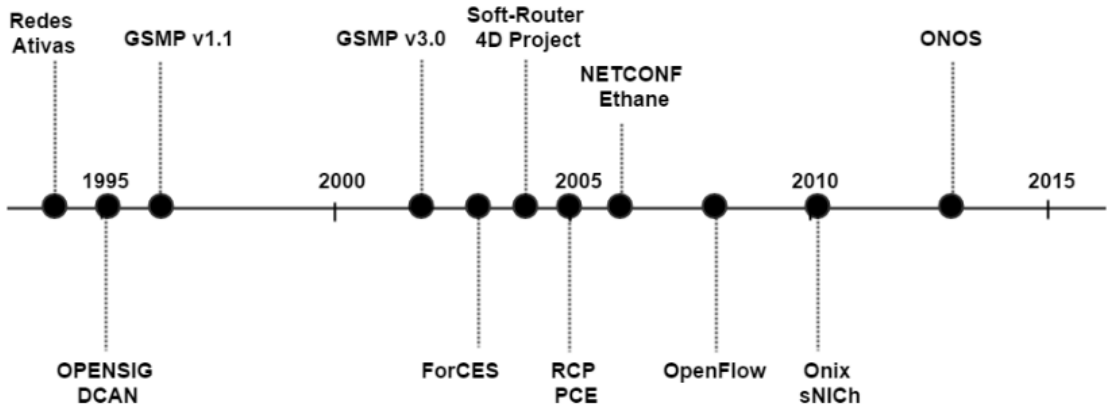

Figura 2.2: Predecessores do paradigma SDN. Fonte: (COSTA, 2016)

O primeiro trabalho na linha do tempo é o de Redes Ativas de (TENNENHOUSE et al., 1997). Ele propõe que os elementos dessas redes ativas, switches e roteadores, por exemplo, sejam capazes de realizar cálculos sobre os pacotes recebidos e até mesmo modificá-los em seu conteúdo. Para isso são propostas as abordagens de switches programáveis e cápsulas. Na primeira abordagem de switches programáveis, não há modificação nos pacotes, os dispositivos no plano de dados devem ser capazes de receber instruções externas sobre como lidar com os fluxos recebidos. Na segunda abordagem, de cápsulas, há modificação dos pacotes, sendo os mesmos encapsulados em programas, num formato chamado quadro de transmissão, que devem ser executados nó a nó no trajeto da rede. As Redes Ativas e seus princípios apresentados no trabalho de (TENNENHOUSE et al., 1997) contribuiram significativamente para o desenvolvimento do conceito de SDN, introduzindo os conceitos de funções programáveis de rede e virtualização de rede (COSTA, 2016). Apesar de promissor, o projeto foi interrompido, dentre outros fatores, por não alcançar público suficiente para continuação da pesquisa.

Posterior ao trabalho de Redes Ativas, o grupo OPENSIG (*OpenSignaling*) começou a proporcionar vários encontros, simpósios e *workshops* nesse contexto de melhoria da

gestão de redes, em um desses encontros foi apresentado o projeto DCAN (DCAN, 1995), que visava proporcionar o controle de redes ATM utilizando a separação dos planos de dados e controle.

Também na mesma época surgiu, por iniciativa final do mesmo grupo (COSTA, 2016), o protocolo GSMP (*General Switch Management Protocol*) (GSMP, 2002) que permitia que o estabelecimento de um controlador capaz de gerenciar conexões através do switch, modificasse portas e outras informações de configuração. A versão mais recente do protocolo foi estabelecida em 2002.

Depois do GSMP, foi proposto o ForCES (*Forwarding and Control Element Separation*) (FORCES, 2004). Essa iniciativa fazia a separação dos planos de controle e dados em duas entidades: o elemento de controle (CE - *control element*) e o elemento de encaminhamento (FE - *forwarding element*). Foi a primeira proposta que de fato separou os dois planos (COSTA, 2016). Ela foi uma arquitetura de muitos recursos, que atendia topologias compostas pelos dois tipos de elementos propostos e contava com uma interface aberta e programável entre os dois planos, além de um conjunto de módulos que implementam tarefas de controle específicas.

Logo após a inicativa do ForCES o controle lógico centralizado da rede surgiu com as arquiteturas RCP e Soft-Router (FEAMSTER et al., 2014). Tal controle era feito através de uma interface aberta com o plano de dados. Para a arquitetura da Soft-Router, foi utilizada a API do 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), (COSTA, 2016). Já a RCP utiliza o protocolo padrão BGP para instalar entradas na tabela de fluxo nos roteadores normais(CAESAR et al., 2005)

No projeto de (CASADO et al., 2009) surgiu o Ethane. Este 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. Em resumo, as 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), (COSTA, 2016). No Ethane, a função dos switches já era simplesmente de encaminhamento, cujas decisões eram tomadas a partir de

regras instaladas pelo controlador da rede(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 descartá-lo ou se vai criar uma entrada na tabela de fluxos para que o pacote possa ser encaminhado) (COSTA, 2016), (CASADO et al., 2009).

Também, proposto pelo Ethane, e altamente utilizado posteriormente, surgiu o conceito de fluxo. A definição deste, segundo (CASADO et al., 2009), é: um fluxo é um conjunto de pacotes de dados que possuem exatamente os mesmos campos de cabeçalho. Dessa forma, quando um conjunto de pacotes com as mesmas características chega na entrada do switch, os pacotes são tratados da mesma maneira. Dessa maneira, reduz-se a quantidade de regras necessárias para tratamento de pacotes, visto que os pacotes de mesmo fluxo obedecem a uma regra só. 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, é possível que se considere o serviço de segurança em uma rede Ethane como sendo a primeira aplicação desenvolvida na história do SDN. (COSTA, 2016).

Para alguns autores, a história do paradigma SDN se inicia em 2007, 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 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 (COSTA, 2016).

2.3 PRINCIPAIS COMPONENTES DE UMA REDE SDN

Como já dito nas seções anteriores, uma rede SDN é constituída de vários elementos em diferentes camadas como exposto na Figura 2.1 da seção 2.1. Nesta seção serão explicados em maiores detalhes cada um desses componentes.

2.3.1 CAMADA DE APLICAÇÃO

A camada de aplicação é a camada que engloba todos os serviços de software e funcionalidades que podem ser executadas pelo controlador da rede. Esses serviços podem englobar engenharia de tráfego, virtualização de rede, balanceamento de carga, monitoramento, roteamento, QoS, dentre outros. Ou seja, é essa camada que é responsável pela programação da rede, sendo o Controlador, apenas um instrumento, ou seja, um meio para a realização do controle lógico.

2.3.2 APIS *NORTH-BOUND*

É a camada de interface responsável pela interação entre a camada de controle e a camada de aplicação. Ela permite que programas desenvolvidos em uma linguagem de mais alto nível (Python por exemplo) sejam capazes de controlar a rede. Comparadas às APIs *South-bound*, as interfaces *North-bound* possuem um nível maior de abstração.

Para a direção controlador-aplicação é utilizada a expressão "norte". É nela em que 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 *South-bound*, que já possui um padrão amplamente aceito (OpenFlow), não existe uma definição de uma

API *North-bound* padrão para SDN. A maioria dos controladores (Floodlight, Trema, NOX, POX, entre outros) 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. (COSTA, 2016).

2.3.3 CAMADA DE CONTROLE

A camada de controle de uma rede SDN é composta de uma ou mais instâncias do elemento chamado Controlador. Esse elemento é o responsável pela realização das operações de controle sobre a rede através do monitoramento e modificação de elementos da camada de dados (switches e roteadores por exemplo). 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 (COSTA, 2016).

2.3.4 APIS *SOUTH-BOUND*

As interfaces South-bound, 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 South-bound, uma vez que ele implementa as interações e a comunicação entre o controlador da rede e os dispositivos de comutação (Figura 2.1). A maioria dos controladores suporta apenas o OpenFlow como API South-bound, 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 CAMADA DE DADOS

Nessa camada se encontram os dispositivos de encaminhamento de pacotes. Tais equipamentos de comutação são 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.

2.4 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 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 2.3 (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.

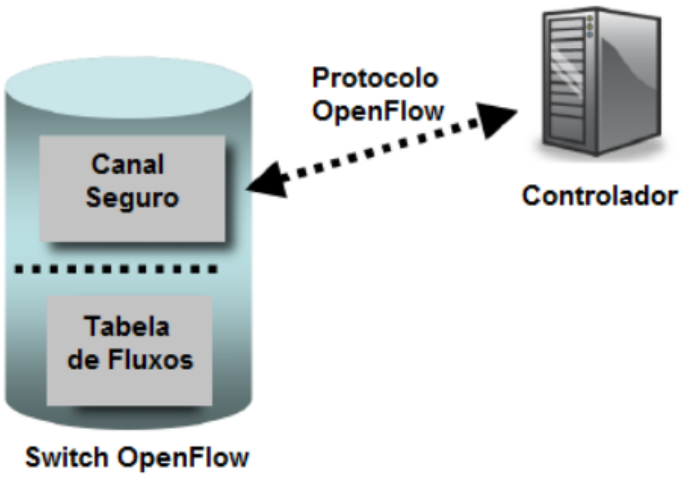

Figura 2.3: Arquitetura OpenFlow. Fonte: (COSTA, 2016)

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

tradas 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 2.4). 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).

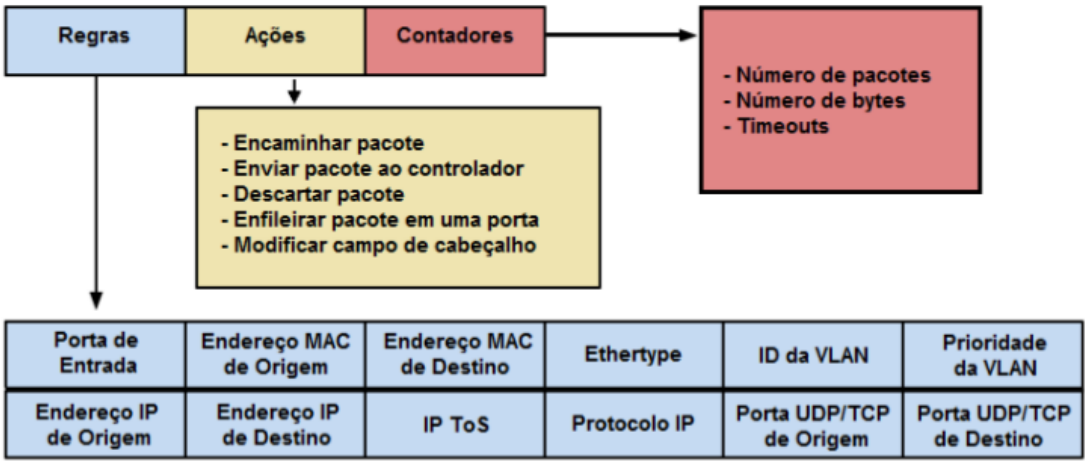

Figura 2.4: Formato de entrada na tabela de fluxos. Fonte: (COSTA, 2016)

Os campos de cabeçalho de uma entrada da tabela (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 2.4 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, gradativa-

mente, 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 2.4 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 Open Network Foundation (ONF) havia escolhido essa versão para a validação de novos recursos.

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 (COSTA, 2016).

Para essa dissertação foi escolhida a versão 1.0 do protocolo Openflow visto que o equipamento disponível para testes no laboratório implementa essa versão do protocolo e porque ainda é aversão do protocolo mais utilizada e suportada (LARA et al., 2014).

3 DESEMPENHO E SEGURANÇA EM REDES SDN

Neste capítulo vamos conceituar desempenho em redes SDN. Vamos exemplificar maneiras de medir desempenho em redes SDN. Além disso, vamos apresentar conceitos de segurança nesse tipo de arquitetura. Nesse sentido, vamos apresentar trabalhos do estado da arte em ambos os temas.

3.1 DESEMPENHO DO PLANO DE DADOS

Existem vários fatores que podem influenciar o desempenho do Plano de Dados de uma rede SDN, desde características do próprio hardware até peculiaridades do protocolo OpenFlow e suas versões. Alguns desses fatores são:

- **Tipo do Switch.** São dois os principais tipos de switch com suporte OpenFlow: *switch em hardware* (exemplo: Extreme Summit x460-24p, utilizado nesse trabalho) e *switch em software* (exemplo: OpenVSwitch, também utilizado nesse trabalho).

  Switches em hardware fazem o processamento dos pacotes através de chips especializados para essa função, os chamados ASICs (*Application Specific Integrated Circuits*). Esses 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.

  Switches em software, por sua vez, processam dados através de CPUs, ficando delimitados pelo poder de seu processamento. Switches em hardware são considerados mais rápidos do que software switches, pois usam memórias especializadas e hardware para realizar *matches* em paralelo, enquanto as CPUs de switches em software utilizam altas taxas de processamento para alcançar um menor desempenho em relação aos ASICs. Portanto, em termos de desempenho e disponibilidade, switches em hardware são melhores do que switches em software. Apesar disso, enquanto

switches de hardware suportam um número limitado de entradas na tabela de fluxos, os switches de software podem suportar uma quantidade de regras em ordens de magnitude superiores, além de possuírem maior flexibilidade para implementar ações mais complexas (COSTA, 2016).

- **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 (COSTA, 2016). 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. 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.

- **Tipo de regras 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 (COSTA, 2016). Portanto, a capacidade de armazenamento de regras exatas é

maior do que a de regras curinga (BANERJEE; KANNAN, 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; 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; GANJALI, 2015).

- **Forma 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 (COSTA, 2016). 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; 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 SEGURANÇA EM REDES SDN

Além de todos os lugares de ataque das redes tradicionais (como roteadores, servidores, dentre outros), SDN tem novos alvos como Controlador SDN, infra-estrutura virtual e rede OpenFlow (HOTSDN..., 2013). Nos próximas subseções serão descritos alguns problemas típicos de proteção em OpenFlow/SDN (como recuperação de falhas) e esquemas de segurança. Na Tabela 3.1 temos as fontes desses esquemas de segurança, enquanto isso, na Tabela 3.2, mostramos as características desses esquemas. Proteção aqui se refere à esquemas para mitigar falhas naturais, e segurança se refere à meios para se contornar

ataques intencionais (HU et al., 2014).

Tabela 3.1: Referência de esquemas de segurança

Número para identificação do trabalho	Referência
1	(SHARMA et al., 2011)
2	(KEMPF et al., 2012)
3	(CHUNG et al., 2013)
4	(SHIN et al., 2013)
5	(MEHDI et al., 2011)
6	(SCHLESINGER, 2017)
7	(KORDALEWSKI; ROBERE, 2012)

Tabela 3.2: Comparação de diferentes esquemas de segurança em SDN. Adaptado de: (HU et al., 2014)

Trabalho (ver Tabela 3.1)	1	2	3	4	5	6	7
Usa OpenFlow	sim	sim	sim	sim	não	sim	sim
Introduz Nova Arquitetura Baseado em OpenFlow	sim	não	sim	não	não	sim	sim
Experimentos (E) ou Redes Reais (R)	E	E	R	E	E	E	E
Software (S) ou Hardware (H)	S	S	S	S	S	S	S
Introduz Nova Linguagem para SDN	não	não	não	não	sim	não	não

3.2.1 DETECÇÃO DE INVASORES

Um esquema de detecção de invasores e seleção de medidas de proteção para SDN (nomeado NICE) é abordado em (CHUNG et al., 2013). Ele visa conseguir segurança em redes virtuais tais como SDN e computação em nuvem. O NICE é composto de duas fases:

1. Usa um agente de detecção chamado NICE-A para capturar tráfego no servidores em nuvem. Usa uma estrutura chamada SAG (Scenario Attack Graph) que é atualizada cada vez que o NICE-A verifica a rede. Baseado nos padrões da análise do SAG, o NICE-A sabe quando deve agir.

2. A DPI (Deep Packet Inspection) é ativada se a VM entrar em estado de inspeção. Ela pode usar o SAG para verificar ameaças à segurança e vulnerabilidades da VM.

O NICE executa um software de segurança de baixo impacto em cada servidor em nuvem. Ele inclui três módulos de software, um analisador de ataque, um controlador

de rede e um servidor em VM. Esse servidor pode monitorar o estado da rede em tempo real e construir o perfil de todos os serviços e portas. O analisador de ataque pode deduzir a correlação de eventos através da análise dos nós do SAG e então achar potenciais brechas de segurança e detectar a ameaça corrente. O controlador da rede pode controlar todas as configurações em cada dispositivo de hardware e software baseado em protocolo OpenFlow, dessa maneira, o NICE se encaixa bem com o paradigma SDN (HU et al., 2014).

3.2.2 SEGURANÇA MODULAR

Embora o OpenFlow (OF) separe o plano de dados do plano de de controle e simplifique, dessa maneira, as operações de hardware, ele também trás problemas de segurança de ponto único: uma vez que o controlador é atacado, todos os switches são influenciados e não conseguem entregar os pacotes de maneira correta (HU et al., 2014).

O FRESCO-DB (SHIN et al., 2013), módulo de banco de dados, é capaz de simplificar a gestão de chaves de segurança. Ele utiliza um formato de chave de sessão unificado e modelo de reputação IP. Consiste de duas partes importantes:

1. Camada de Aplicação: usa APIs e interpretadores para suportar aplicações modulares

2. SEK (security enforcement kernel), entidade que pode ser utilizada para executar todas as ações relacionadas às políticas de segurança.

Várias políticas de segurança como DROP, REDIRECT e QUARANTINE podem ser reforçadas por aplicações de segurança desenvolvidas com scripts FRESCO para reagir a ameaças de rede através da simples configuração de uma variável de ação (HU et al., 2014). As duas partes acima são desenvolvidas no controlador NOX. Um usuário de rede pode utilizar a linguagem de script FRESCO para definir vários módulos de segurança.

3.2.3 DETECÇÃO DE ANOMALIA DE TRÁFEGO SDN

Em (MEHDI et al., 2011) são propostos quatro diferentes algoritmos de detecção de anomalia. Cada um deles é avaliado em redes reais, tanto domicialiares quanto empresariais. Em resumo, as ideias dos quatro algoritmos são (HU et al., 2014):

1. Algoritmo *TRW-CB (Threshold Random Walk with Credit Based Rate Limiting)*: como sabemos, uma conexão TCP pode ser estabelecida com uma taxa de sucesso muito mais alta se não estiver sob ataque. Utilizando testes de hipótese sequencial (como testes de vazão por exemplo, como o utilizado em nosso trabalho), ele analisa cada estado de conexão e tenta detectar uma infecção.

2. *Rate-Limiting*: Uma infecção de vírus pode causar várias requisições de conexão num período curto de tempo, enquanto um tráfego benigno nunca geraria uma taxa tão alta. Esse é o princípio do *Rate-Limiting*, que é, checar a requisição e detectar eventos maliciosos.

3. *Maximum Entropy Detector*: um calculador de entropia máxima pode ser usado para achar características estatísticas de tráfego. Ao utilizar uma distribuição de linha base, o modelo de entropia máxima pode ser usado para classificar pacotes em diferentes categorias e cada categoria ser caracterizada como benigna ou anormal (HU et al., 2014).

4. *NETAD*: Funciona como um filtro ou firewall. Simplesmente escaneia o cabeçalho do pacote e bloqueia qualquer pacote suspeito baseado nas atribuições dos campos.

3.2.4 SEGURANÇA BASEADA EM LINGUAGEM

A análise de como programar SDN de forma segura e confiável é discutida em (SCHLESINGER, 2017). A solução envolve o desenvolvimento de um novo modelo de programação que suporte o conceito de fatia de rede. O isolamento do tráfego de um programa proveniente de outro é conseguido com ajuda dessas fatias. Elas também isolam um tipo de tráfego de umas das outras. Eles desenvolveram uma semântica para essas fatias, e ilustram novos tipos de princípios de raciocínio modular formal que os programadores de rede agora podem explorar. Isto fornece definições de propriedades de segurança de ponta a ponta que fatiam, implicam e verificam a correção de um compilador para uma cálculo de núcleo ideal em uma programação de rede baseada em fatia. Eles também descreveram sua implementação, que está equipada com uma estrutura de validação de tradução que verifica automaticamente programas compilados usando o provador do teorema Z3.

Hoje é um desafio implementar o isolamento nas redes. A maioria dos sistemas ainda usa a configuração manual para bloquear o tráfego suspeito. Essa configuração geralmente

é muito intensiva em mão-de-obra e específica do fornecedor. Em (SCHLESINGER, 2017), o autor sugere utilizar uma linguagem de programação de alto nível para configurar as políticas de entrega de dados e isolar diferentes domínios. Isso deixa as configurações de dispositivo de baixo nível, propensas a erros, para os compiladores SDN. Esse esquema supera a deficiência do NOX, que não pode facilmente isolar diferentes sub-redes quando furos de segurança são detectados.

A segurança baseada em linguagem (SCHLESINGER, 2017) alivia os programadores de complicadas programações de segurança devido ao uso do conceito de isolamento de fatias. Uma fatia é definida como uma conexão virtual que consiste em roteadores, switches, portas de comunicação e enlaces. As fatias são definidas com atributos e ações. Uma fatia pode ser isolada de outra se ao executar uma em paralelo à outra não resultar em vazar pacotes de uma fatia para a outra. Eles definiram várias propriedades de isolamento e desenvolveram uma condição operacional chamada de separação que implica na propriedade de isolamento. Finalmente, formalizaram um algoritmo de compilação e provaram que ele estabelece separação e isolamento (HU et al., 2014).

3.2.5 PROBLEMA DE DETECÇÃO DE LAÇOS

Os laços de roteamento fazem com que os pacotes nunca cheguem ao seu destino. Em (KORDALEWSKI; ROBERE, 2012) é proposto um algoritmo dinâmico que é construído da análise de espaços de cabeçalho and permite a detecção de laços em SDNs. Nele, o modelo de rede pode ser ilustrado como um grafo direcionado. Consequentemente, os conceitos de análise de espaços de cabeçalho foram traduzidos para a teoria de grafos. Grafos de regra e o problema de detecção de laços dinâmicos são estudados em (KORDALEWSKI; ROBERE, 2012). Eles mostraram como modelar a rede como um grafo direcionado. Ao analizar o alcance e conectividade do grafo de topologia, nó a nó, não se pode achar laços. Um algoritmo dinâmico de componente fortemente conexo é proposto em (KORDALEWSKI; ROBERE, 2012) para permitir verificar inserções e deleções de arestas. Ele também é capaz de detectar laços em uma rota.

3.2.6 PROTEÇÃO E RECUPERAÇÃO DE FALHAS EM SDN

Para construir uma rede SDN confiável, é necessário que ela seja resistente tanto à falhas internas quanto externas (HU et al., 2014). Neste trabalho, falhas externas se

referem à ataques intencionais de adversários. As soluções de segurança discutidas nas subseções anteriores visam detectar e sobrepujar ataques externos. As falhas internas se referem à erros de hardware e erros humanos não intencionais, como o abordado por (MATSUMOTO et al., 2014) em sua definição do Problema do Administrador Malicioso. Esses erros são aqui chamados de erros de proteção. Por exemplo, uma rede SDN pode falhar se a comunicação entre o controlador e os switches seja cortada devido falta de banda. Dessa maneira os comandos do controlador não seriam enviados para os switches. Se o enlace switch-a-switch tiver falha, muitos pacotes podem ser perdidos. Portanto, algum tipo de monitoramento e esquema de recuperação de falhas é necessário para tratar essas situações.

Podem existir muitos outros problemas de proteção em SDN. Por exemplo, o controlador pode não ser capaz de sincronizar a atualização em todas as tabelas de fluxo de todos os switches devido à uma falha de gerência de agendamento. O switch pode não ser capaz de reportar o estado de entrega de tráfego (então o controlador poderia não ser capaz de atualizar a tabela de fluxos por um determinado tempo) (HU et al., 2014). Quando utiliza-se múltiplos controladores em SDN, os controladores podem não ser capazes de manter a consistência de controle devido à atrasos de comunicação. Nas discussões seguintes, serão ilustrados alguns esquemas que visão contornar problemas de proteção, assim como fazemos em nossa proposta nessa dissertação.

3.3 TRABALHOS RELACIONADOS

Como dito anteriormente, um dos fatores que pode acarretar na variação do desempenho do Plano de Dados é a forma de instalação de regras, normalmente quando as atualizações de regras se dão de maneira reativa (COSTA, 2016). No que diz respeito ao desempenho, o trabalho proposto por essa dissertação visa analisar, através dessa ótica do contexto de averiguação do tempo de instalação de regras, o impacto de tais atualizações quando realizadas de maneira proativa, uma vez que essa é a maneira de influência do módulo afetado para o problema do administrador mal configurado.

Não existe, ao nosso conhecimento, no estado-da-arte, nenhum estudo que verifique, tanto para o ambiente real quanto para o virtual, o desempenho do Plano de Dados através da metodologia utilizada nesse trabalho. De fato, o trabalho de (KUŹNIAR et al., 2014) utiliza uma metodologia parecida com a aqui proposta fazendo a distinção entre

atualizações do controlador sob os comutadores da rede. Contudo, o trabalho de (KUŹ-NIAR et al., 2014), visa avaliar o desempenho do Plano de Controle no processamento de atualizações, em três switches de hardware, e a real taxa de atualização de regras nesse cenário.

Vários foram os trabalhos e áreas mencionados na seção anterior desse capítulo, muitos deles tem correlação forte com a proposta dessa dissertação como explicitaremos nessa seção.

Como mencionado em (HU et al., 2014), apesar dos benefícios do SDN, a dependência centralizada da rede em um único elemento (Controlador), apresenta problemas à segurança da rede uma vez que, uma brecha de segurança nesse sistema, quando explorada, pode levar ao mal-funcionamento da rede como um todo. O trabalho de(MATSUMOTO et al., 2014) introduz esse conceito, mas invés de considerar ataques provenientes de influências externas, o mesmo considera brechas de origem interna, no caso os próprios componentes do controlador da rede, que, uma vez que esteja mal configurado, pode levar ao mal desempenho da rede. Este cenário é descrito por (MATSUMOTO et al., 2014) como o Problema do Administrador Malicioso e, em seu trabalho, há uma proposta de um controlador chamado Fleet que, através do uso de dois protocolos, também propostos em seu trabalho, visa diminuir o impacto negativo no desempenho da rede que esse cenário pode causar.

(MATSUMOTO et al., 2014) considera um cenário com múltiplos controladores e faz a simulação em software usando máquinas virtuais e um emulador de redes chamado Mininet. Nossa proposta fará uso de máquinas reais usando um emulador de switch chamado OpenVSwitch, contudo, nosso cenário possuirá apenas um controlador.

Uma ideia parecida com a do SAG e do NICE-A de (CHUNG et al., 2013) é utilizada no mecanismo de proteção proposto dessa dissertação. SAG é uma estrutura de dados que guarda o "estado da rede", NICE-A é um agente que toma atitudes quando julgar que houve um ataque; nosso algoritmo proposto funciona de maneira similar uma vez que há uma verificação constante do "estado da rede", no nosso caso medido através da vazão de dados durante o envio de pacotes de um *host* transmissor para um *host* receptor, e em seguida, caso verificada uma anomalia, de maneira análoga ao algoritmo TRW-CB proposto em (MEHDI et al., 2011), o nosso componente proposto realiza as decisões necessárias para contornar o problema, de maneira análoga ao NICE-A.

Em (SHARMA et al., 2011), um esquema de recuperação rápida de falhas é proposto para redes SDN/OpenFlow. Foi investigada a freqüência de troca e a taxa de perda de pacotes em sua avaliação. Ele usa o controlador NOX para este serviço de recuperação.

Na rede OpenFlow, podemos adicionar imediatamente ou proativamente uma entrada de fluxo à tabela após a ocorrência de uma falha. O tempo de recuperação total é determinado pelo tempo de vida das entradas de fluxo. Em (SHARMA et al., 2011), dois valores de tempo limite são definidos, um é chamado *idle timeout*, o que significa que o intervalo de tempo que uma entrada de fluxo deve ser removida se não for usada por um determinado período de tempo (ou seja, nenhum pacote para esse tipo de entrada de fluxo está passando pelo switch); O outro é *hard timeout*, que é o intervalo de tempo máximo que uma entrada de fluxo pode permanecer. Independentemente do tempo limite, ele irá desencadear a recuperação de falhas. Observe que o sistema não pode ser recuperado se o controlador não tem ideia sobre o tipo de falha ocorrida. O controlador pode apenas adicionar aleatoriamente uma entrada de fluxo na tabela se o tipo de falha não for reconhecido.

Em (SHARMA et al., 2011) o controlador NOX foi utilizado para implementar um esquema de *L2-learning* para detecção de falhas. Se ocorrer uma falha, as entradas de fluxo incorretas devem ser apagadas de todos os switches e novas entradas devem ser imediatamente adicionadas a cada switch. O controlador deve ter esquemas robustos para detectar a falha e encontrar o novo caminho de roteamento para entregar os fluxos. O controlador verificará o caminho de roteamento antigo associado aos enlaces com falha. Se o caminho antigo ainda é utilizável, não estabelecerá um novo caminho. Caso contrário, o novo caminho precisa ser adicionado às entradas de fluxo e as entradas antigas devem ser removidas imediatamente. O ambiente de testes em (SHARMA et al., 2011) consiste de uma máquina com sistema operacional Ubuntu 9.04 para instalar OpenVSwitch 1.1.0 e NOX 0.9.0. Mais de 10 mil pacotes de ping foram enviados a cada 10 ms. A taxa de perda de pacotes é calculada ao se contar o número de pacotes ping recebidos. O *hard timeout* é configurado para cada 20 segundos e o *idle timeout* para cada 10 segundos. Os laços de roteamento são evitados usando algoritmos *spanning tree*. O reestabelecimento de rota em (SHARMA et al., 2011) é mais rápido que o convencional re-convergência MAC ou ARP. Ele usa apenas 12 ms para recuperar uma falha de enlace.

Em (KEMPF et al., 2012) é estabelecido um esquema chamado Operations, Admi-

nistration and maintenance (OAM), ferramenta utilizada para re-estabelecer um novo caminho. Para minimizar a troca de caminhos, ele usa uma abordagem pro-ativa, que é, um caminho de *backup* é pré-instalado na tabela de fluxos caso um caminho falhe. Esse esquema pode recuperar caminhos em tempos menores que 50 ms. Em adição, alguns pacotes de sonda são periodicamente enviados na rede. Se esses pacotes não forem recebidos, o sistema sabe que houve uma falha de enlace. Se demorar um tempo relativamente longo para o recebimento do pacote de sonda, uma falha também é detectada. Então (KEMPF et al., 2012) provê uma maneira eficiente de recuperar de uma falha de rota.

4 AVALIAÇÃO DE REDES SDN SOB ATAQUE

Ao analisar sistemas SDN, podemos verificar, de acordo com (MATSUMOTO et al., 2014) que os mesmos são vulneráveis a ataques como qualquer outro tipo de sistema, mesmo antes de se aplicar uma camada de segurança. Uma dessas vulnerabilidades, nomeado por (MATSUMOTO et al., 2014) como Problema do Administrador Malicioso, consiste na má configuração de elementos próprios da rede já em sua concepção. No caso em questão, a entidade que será simulada como "mal configurada", será o Controlador. Este executará dois componentes principais: o primeiro, mal configurado, realizará atualizações, de maneira proativa, em um elemento comutador da rede com o intuito de diminuir seu desempenho; e o segundo, componente proposto por essa dissertação para mitigar a diminuição de desempenho causada pelo primeiro.

Apesar de ser capaz de realizar operações sobre os switches, o componente mal configurado não pode influenciar o Controlador POX. Entretanto, estamos considerando a possibilidade do componente mal configurado ser capaz de mandar mensagens arbitrárias para um elemento comutador da rede (switch).

Com o intuito de averiguar qual tipo de atualização, capaz de ser realizada pelo Controlador em um elemento comutador, gera o maior impacto no desempenho do Plano de Dados, neste capítulo optou-se por realizar a simulação de ataques em um dos nós da rede SDN. Tal simulação é descrita em detalhes da Seção 4.1.

Após a verificação de qual interferência gera maior impacto sobre o Plano de Dados no ambiente real e no virtual, no capítulo 5 apresentaremos com detalhes a proposta de mitigação da redução de desempenho.

4.1 METODOLOGIA DE AVALIAÇÃO DE DESEMPENHO

Para conduzir o processo de avaliação de desempenho em redes SDN, foram realizadas as verificações do impacto da variação na taxa de 3 operações de atualização em regras na tabela de fluxos da rede SDN. Tais atualizações são: a inserção, a remoção e a alteração de regras nas supracitadas tabelas.

Foram montados dois cenários de testes, um em *hardware* outro em *software*, apresentados nas Figuras 4.1 e 4.2. As arquiteturas desses testes são compostas de quatro

elementos:

- Host Receptor: elemento que possui o endereço alvo de ping do Host Transmissor
- Host Transmissor: elemento que realizará a ação de ping para o Host Receptor, ele será responsável pela coleta de informações
- Controlador POX: elemento responsável pelo controle da rede e pelas atualizações forçadas no Switch OpenFlow, OpenVSwitch no caso do ambiente virtual, e switch Extreme Summit x460 para o caso do ambiente real.
- Switch OpenFlow: elemento que fará o encaminhamento dos pacotes ping do Host Transmissor para o Host Receptor, e que será alvo de investigação de processamento

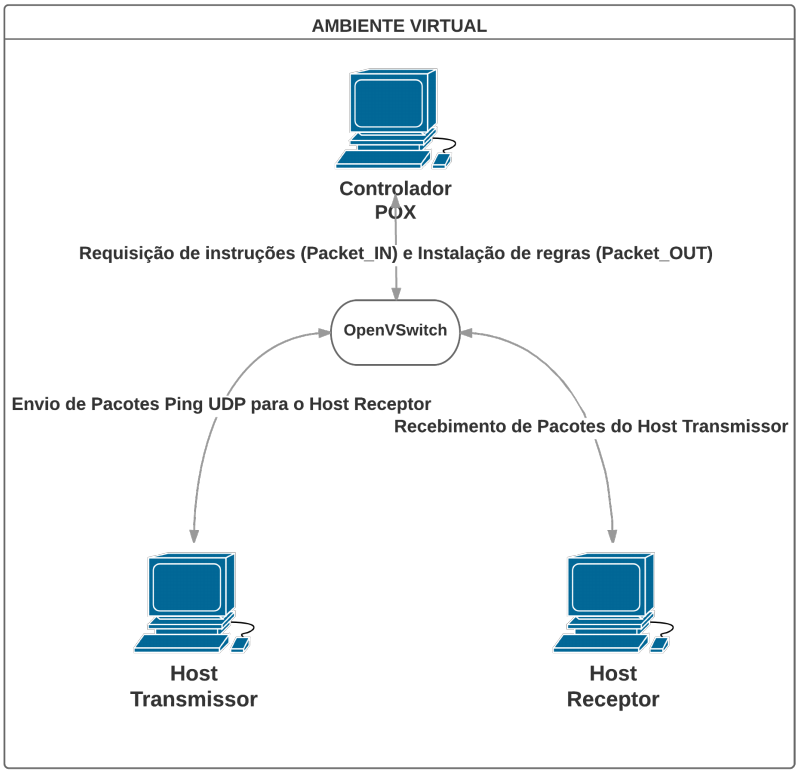

Figura 4.1: Arquitetura de testes usando OpenVSwitch

Para cada cenário o teste foi repetido 30 vezes. Em cada teste foram enviados 1000 pacotes do Host Transmissor para o Host Receptor, para cada taxa de atualização nas regras (inserção, remoção e alteração). As taxas de atualização são de 1 regra a cada "X" milisegundos, onde "X" varia de 10 em 10 milisegundos num intervalo de 10 até 100. Para comparação de resultados, foi ainda realizado um experimento apenas de envio de

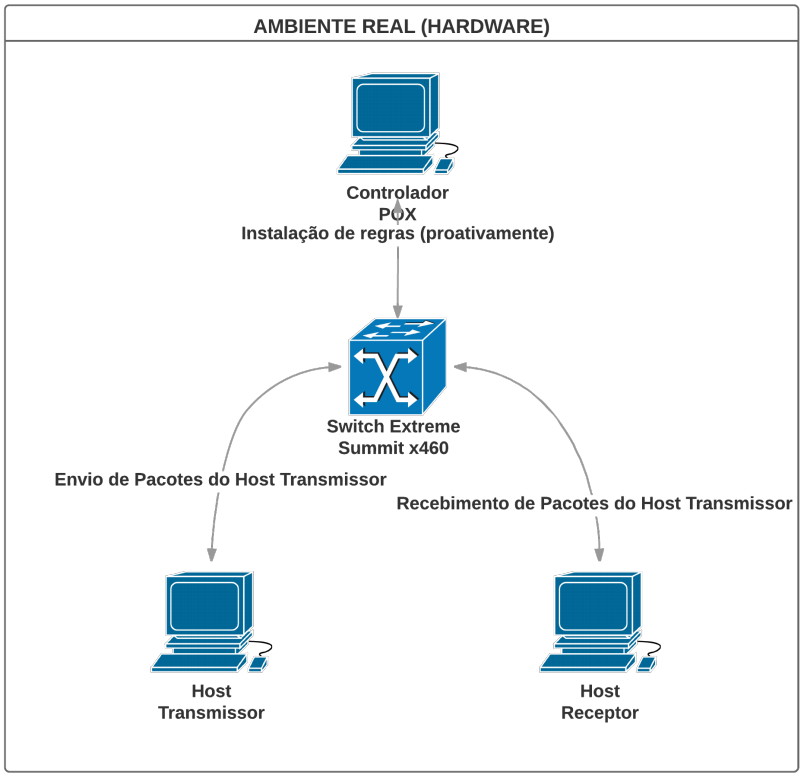

Figura 4.2: Arquitetura de testes usando Switch Extreme Summit x460

pacotes do Host Transmissor para o Host Receptor, sem influência do Controlador POX, o resultado de tal experimento será identificado em cada gráfico com o rótulo “base”.

4.2 RESULTADOS DE DESEMPENHO

Em estatística descritiva, segundo (CHAMBERS et al., 1983), o modelo de gráfico Box Plot é um gráfico em que:

- O eixo vertical representa uma variável a ser avaliada
- O eixo horizontal representa um fator de interesse

No experimento em questão, a variável a ser avaliada é o RTT (*Round-Trip Time*) dos pacotes, ou seja, o tempo que o pacote leva para ir ao Host Transmissor e o mesmo receber uma resposta do Host Receptor. Já o fator de interesse, nesse caso, são as taxas de variação de operações de atualização conduzidas sobre as regras na tabela de fluxos do switch OpenFlow.

Gráficos do tipo Box Plot são usados para representação dos quartis Q1 (25% dos valores medidos) e Q3 (75% dos valores medidos) assim como a mediana (ou Q2, repre-

sentante de 50% dos valores medidos) e os *outliers* (valores que se distanciam da maioria dos resultados envolvidos entre Q1 e Q3).

4.2.1 AMBIENTE VIRTUAL

O gráfico da Figura 4.3 representa o Box Plot referente ao resultado do experimento conduzido de inserção de regras na tabela de fluxos com variação de taxa.

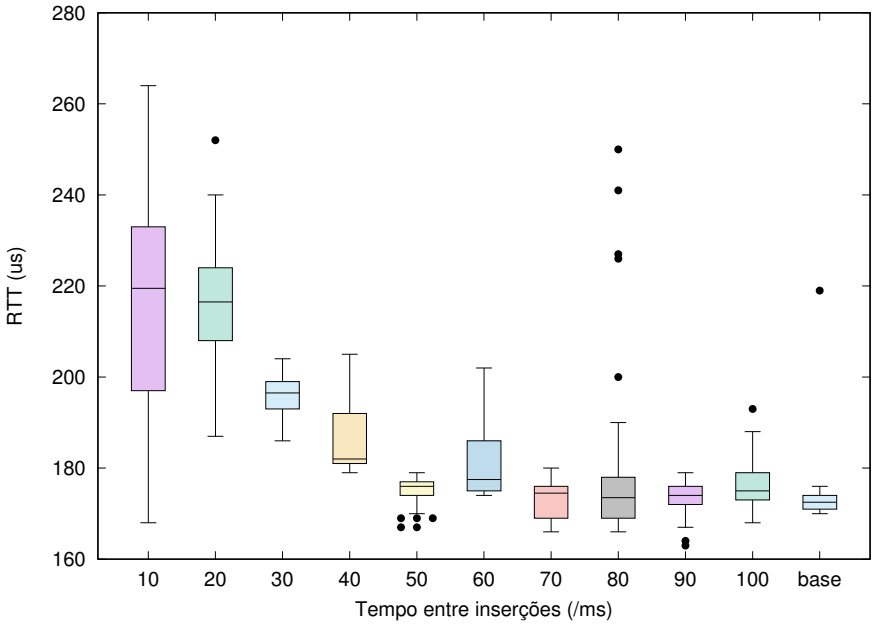

Figura 4.3: Inserção de Regras (Ambiente Virtual)

Ao observar os resultados da Figura 4.3, podemos verificar que, à medida em que as regras vão sendo inseridas com uma taxa mais alta, o RTT dos pacotes apresenta uma tendência de aumento.

Já o gráfico da Figura 4.4 representa o Box Plot referente ao resultado do experimento conduzido de remoção de regras na tabela de fluxos com variação de taxa.

Ao observar os resultados da Figura 4.4, podemos verificar que, à medida em que as regras vão sendo removidas com uma taxa mais alta, o RTT dos pacotes apresenta uma tendência de aumento. Podemos verificar ainda que, em comparação com o experimento conduzido de inserção, há uma influência muito maior nos RTTs das operações de remoção do que de inserção.

O terceiro gráfico da Figura 4.5 representa o Box Plot referente ao resultado do experimento conduzido de alteração de regras na tabela de fluxos com variação de taxa.

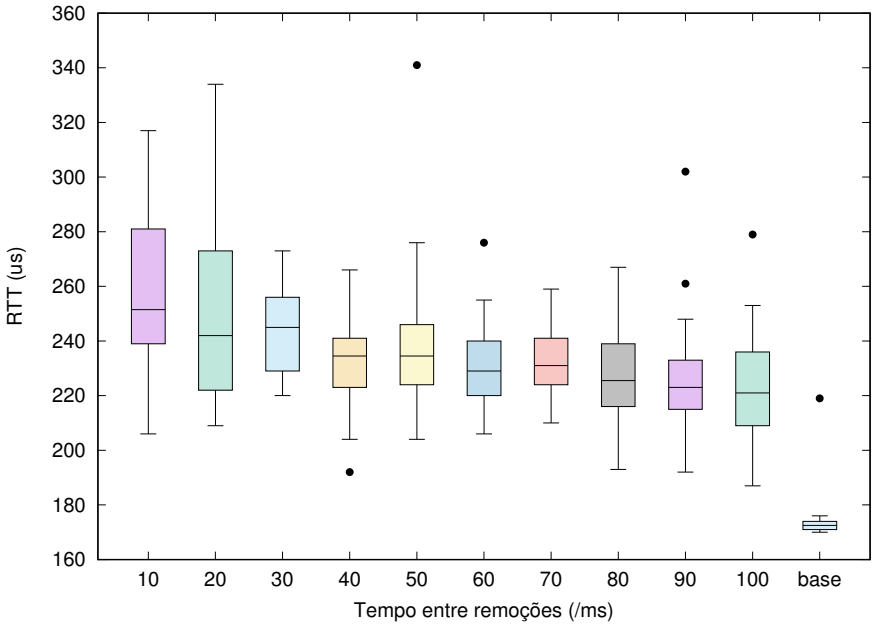

Figura 4.4: Remoção de Regras (Ambiente Virtual)

Ao observar os resultados da Figura 4.5, podemos verificar que, à medida em que as regras vão sendo alteradas com uma taxa mais alta, o RTT dos pacotes apresenta uma tendência de aumento. Apesar de haver tal tendência, podemos verificar que, em comparação com os experimentos de inserção e remoção, o impacto das operações de alteração é menor.

4.2.2 AMBIENTE REAL

O gráfico da Figura 4.6 representa o Box Plot referente ao resultado do experimento conduzido de inserção de regras na tabela de fluxos com variação de taxa.

Ao observar os resultados da Figura 4.6, podemos verificar que o RTT dos pacotes apresenta uma tendência de estabilidade na média do atraso, independente da alteração da taxa, contudo, pode-se verificar uma alteração na variação do atraso (*jitter*).

Já o gráfico da Figura 4.7 representa o Box Plot referente ao resultado do experimento conduzido de remoção de regras na tabela de fluxos com variação de taxa.

Ao observar os resultados da Figura 4.7, podemos verificar que, independente da taxa de remoções, o RTT dos pacotes apresenta uma tendência de aumento. Podemos verificar ainda que, em comparação com o experimento conduzido de inserção, há uma influência muito maior nos RTTs das operações de remoção do que de inserção.

O terceiro gráfico da Figura 4.8 representa o Box Plot referente ao resultado do expe-

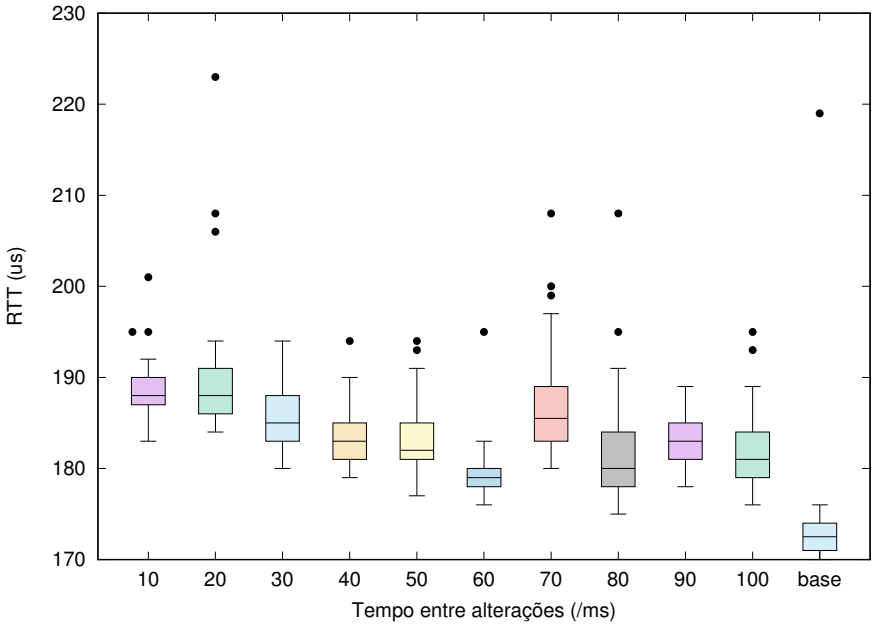

Figura 4.5: Alteração de Regras (Ambiente Virtual)

rimento conduzido de alteração de regras na tabela de fluxos com variação de taxa.

Ao observar os resultados da Figura 4.8, podemos verificar que, independente da taxa e sua variação, o RTT dos pacotes apresenta uma tendência de aumento. Podemos verificar que, em comparação com o experimento de remoção o impacto das operações de alteração é mais constante (menor *jitter*).

4.2.3 COMPARAÇÃO DOS TESTES ENTRE O AMBIENTE VIRTUAL E O REAL

Após a realização dos testes podemos verificar que o comportamento dos switches em software e em hardware foi diferente para os experimentos conduzidos. A seguir fazemos uma comparação, entre os ambientes real e virtual, dos resultados obtidos para as interferências de inserção, remoção e alteração de regras.

- **Inserção**

  Ao comparar os gráficos das Figuras 4.6 e 4.3, podemos verificar que o impacto das operações de inserção em ambiente virtual (22% de acréscimo no pior caso comparado com a base) é muito maior do que o impacto averiguado (variação apenas no *jitter*) no ambiente real. Isso se deve ao fato de que os switches em hardware são capazes de realizar a pesquisa nas tabelas de fluxo de maneira paralela, enquanto

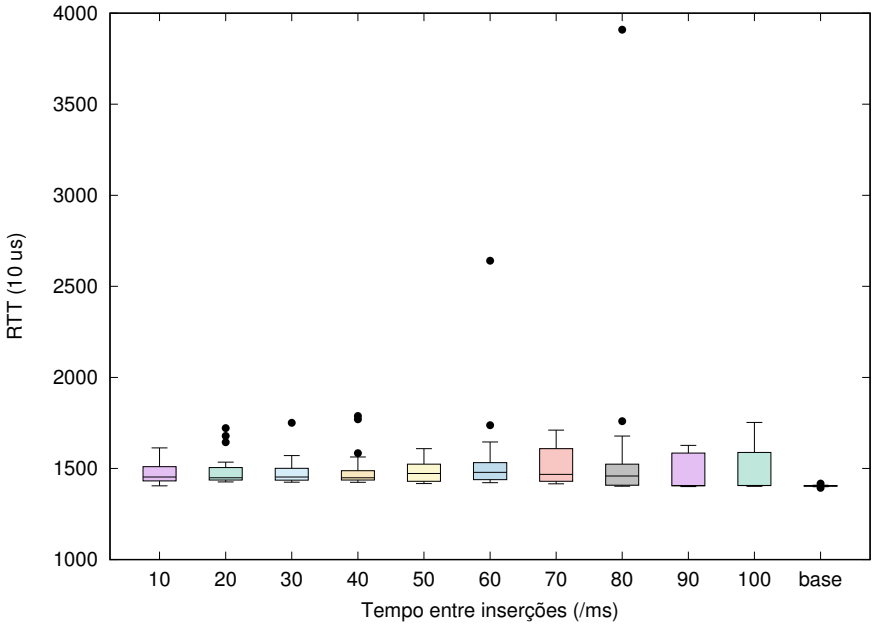

Figura 4.6: Inserção de Regras (Ambiente Real)

os switches de software não.

- **Remoção**

  A comparação dos gráficos das Figuras 4.7 e 4.4 nos permite verificar que o impacto dessa operação, em ambos os ambientes (virtual e real) é alto, cerca de 21.7% em média para o ambiente de virtual, e cerca de 12.5% no ambiente real, em média, no que concerne o aumento no RTT máximo.

- **Alteração**

  Quando comparamos os gráficos das Figuras 4.8 e 4.5 podemos notar que as operações de alteração tem um impacto ligeiramente menor em ambiente virtual (7%) do que em ambiente real (12.5%), na média.

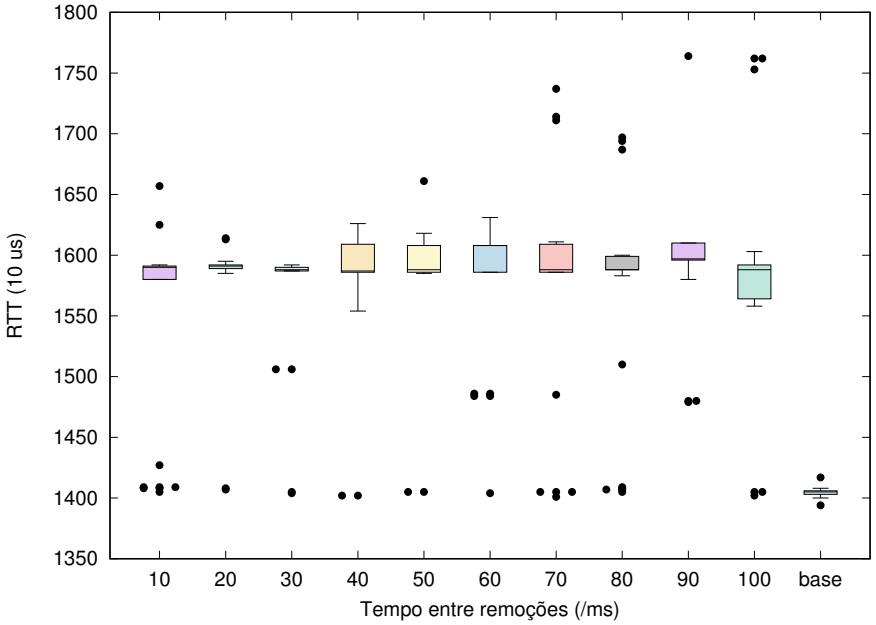

Figura 4.7: Remoção de Regras (Ambiente Real)

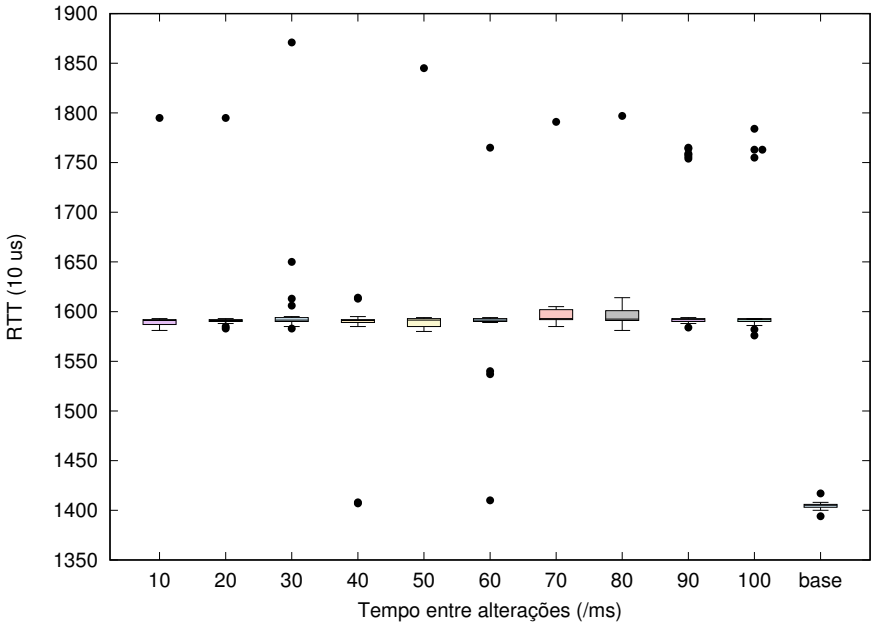

Figura 4.8: Alteração de Regras (Ambiente Real)

5 MECANISMO DE PROTEÇÃO PARA O PROBLEMA DO ADMINISTRADOR MAL CONFIGURADO

Os resultados analisados nos testes de desempenho da seção 4.2, induzem à ideia de que, de fato, pode haver influência no desempenho do Plano de Dados quando um elemento comutador se encontra sob ataque. Nesse capítulo apresentamos uma simulação de um cenário em que o Problema do Administrador Mal configurado se aplica e uma proposta para mitigar os efeitos do mesmo através da utilização de nosso componente de proteção.

5.1 METODOLOGIA

A metodologia consiste na execução paralela de componentes do Controlador POX. Será feita a utilização de dois componentes principais. O primeiro será o componente mal configurado, este fará intervenções no sistema com o intuito de diminuir o desempenho do Plano de Dados. O segundo é o componente de proteção, este funcionará de maneira tal que, ao verificar que o sistema está sendo atacado, tome decisões com o intuito de restaurar o desempenho do dele.

Na Figura 5.1 temos a ilustração da topologia do cenário de ataque. Tal topologia é composta dos seguintes elementos:

- Controlador POX: máquina que serve como controladora da rede SDN, é nela que são executados os módulos do POX, tanto proteção, quanto o que simula má configuração do sistema e que intervém nas regras de maneira imprópria na entidade Switch Atacado.

- Switch Atacado: máquina que executa OpenVSwitch e que é "atacada" de maneira tal que será realizada, sobre ela, um conjunto de ações (inserção, remoção ou alteração) nas regras instaladas na tabela de fluxos. A medida em que é verificada uma variação significativa na vazão dos pacotes enviados entre o Host Transmissor e o Receptor, fazemos, sobre ela, uma ação de retirada das regras de encaminhamento de pacotes através dela para que o encaminhamento de pacotes, a partir desse

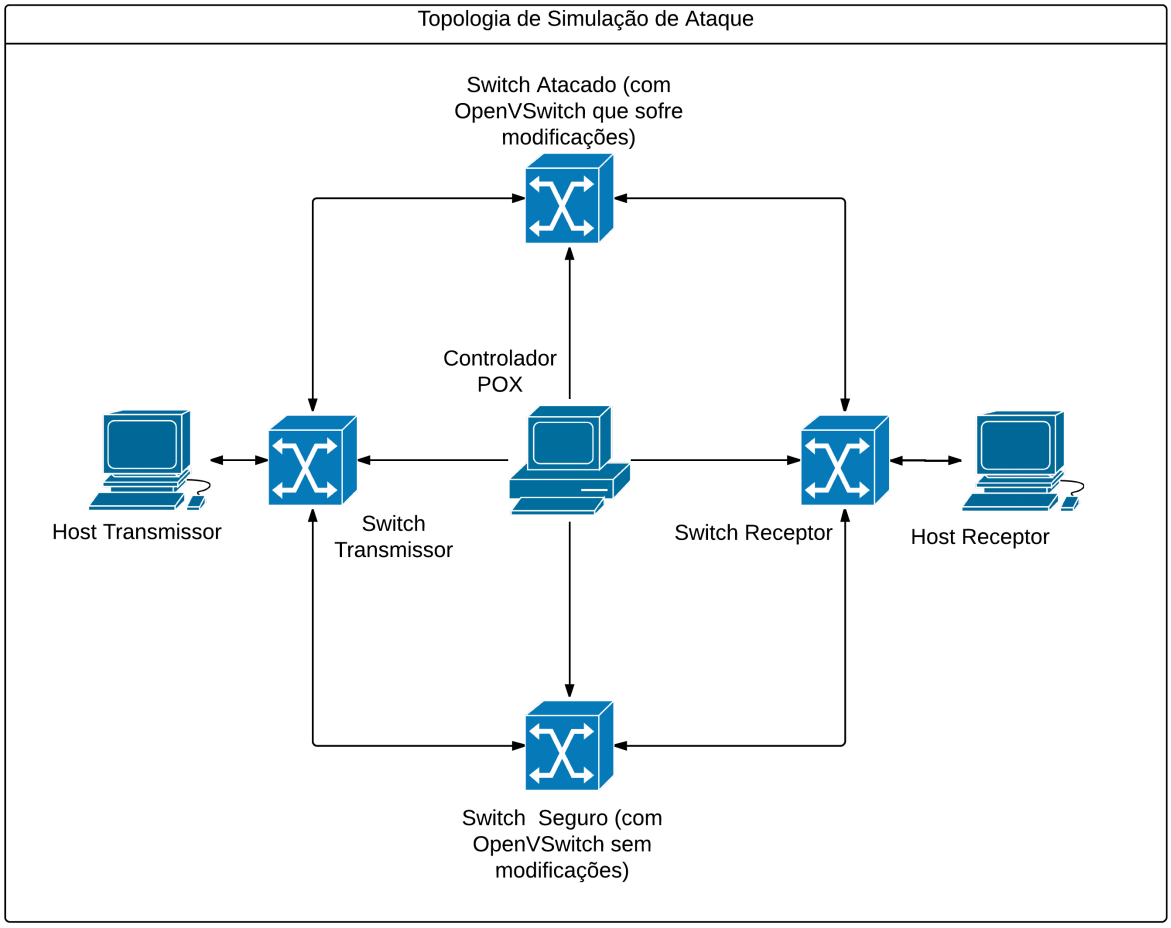

Figura 5.1: Arquitetura do cenário de testes de ataque do componente mal configurado

momento, seja feito através do Switch Seguro

- Switch Seguro: máquina que executa OpenVSwitch e que não é alvo de "ataque", serve como rota alternativa para os pacotes entre o Host Transmissor e o Host Receptor.
- Switch Transmissor: elemento intermediário de encaminhamento
- Host Transmissor: host que realiza o envio de pacotes para o Host Receptor.
- Switch Receptor: elemento intermediário de encaminhamento
- Host Receptor: host que recebe pacotes vindos do Host Transmissor.

Esse experimento é conduzido, separadamente, em momentos distintos e em três ambientes:

1. **Ambiente sem ataque:** Host Transmissor envia pacotes para o Host Receptor, através do Switch Atacado, e não há ataque do Controlador POX. Esse é o melhor caso do experimento.

2. **Ambiente com ataque:** Host Transmissor envia pacotes para o Host Receptor, através do Switch Atacado, e há ataque do Controlador POX, porém não há troca do Switch Atacado pelo Switch Seguro. Esse é o pior caso do experimento.

3. **Ambiente com ataque e troca de switches:** Host Transmissor envia pacotes para o Host Receptor através do Switch Atacado. Enquanto é realizado ataque na rota, verifica-se tal ocorrência através da análise de variações de atraso e, então, é realizada a troca de rota. Dessa forma os pacotes serão transmitidos através do Switch Seguro ao invés de pelo Switch Atacado. Esse é o caso intermediário do experimento, é da comparação dele com os outros dois supracitados que seremos capazes de avaliar o ganho obtido ao utilizarmos o componente proposto nessa dissertação.

O controlador POX é um programa que consiste de vários módulos executados em paralelo, alguns interdependentes e outros não. Todos esses módulos, para o nosso cenário de testes, são executados em uma mesma máquina. No modelo de ataque assumido temos a existência de um módulo, de um conjunto de aproximadamente seis (módulo de proteção, módulo de exibição de pacotes, módulo de roteamento, dentre outros), responsável pela configuração e monitoramento de um único switch, no caso, o switch atacado. Este módulo, mal configurado, irá produzir uma diminuição no desempenho do switch em questão. É importante salientar também que esse é um modelo de ataque não intencional. Interações com intenções maléficas, como infecção de outros módulos do controlador e em consequência, outros switches da rede estão fora do escopo aqui definido.

Na Figura 5.3 temos o fluxograma de execução dos componentes do Controlador POX, tanto o do componente mal configurado quanto do componente de proteção. Na Figura e 5.4 temos o fluxograma do Host Transmissor, que faz uma simulação de envio de pacotes e grava as informações de atraso no envio. É com esses dados que fazemos a avaliação do módulo de proteção. A Figura 5.2 é um foto real do nosso ambiente de teste, a título de curiosidade, para realizar a conexão entre as máquinas foram utilizadas 3 placas de rede para cada máquina de switch e 4 para máquina do Controlador. A conexão entre cada uma das máquinas foi feita com cabos de rede Ethernet de 1 Gigabit *cross-over* feitos e

testados no próprio laboratório.

Figura 5.2: Foto do ambiente de teste.

Para a execução dos testes nos três ambientes, são realizados os seguintes procedimentos: primeiramente iniciam-se todas as máquinas, configura-se o switch virtual, os IPs das placas de rede das máquinas (que estão funcionando como switch) e do Controlador POX. Em seguida inicia-se, no Host Transmissor, o programa geração de pacotes com destino ao Host Receptor.

No primeiro ambiente, sem ataque, apenas realiza-se, ao final das transmissões, a coleta de informações do RTT entre os envios do Host Transmissor para o Host Receptor.

Para o segundo ambiente, com ataque durante toda a transmissão, inicia-se o módulo mal configurado do Controlador POX e ao final das transmissões, realiza-se a coleta de informações do RTT entre os envios do Host Transmissor para o Host Receptor.

Já para o terceiro ambiente, com ataque e troca de switches, inicia-se o módulo mal configurado do Controlador POX, o módulo de proteção (proposto pela dissertação) e ao final das transmissões, realiza-se a coleta de informações do RTT entre os envios do Host Transmissor para o Host Receptor.

O pseudo-código do componente de proteção do Controlador POX para aumento de

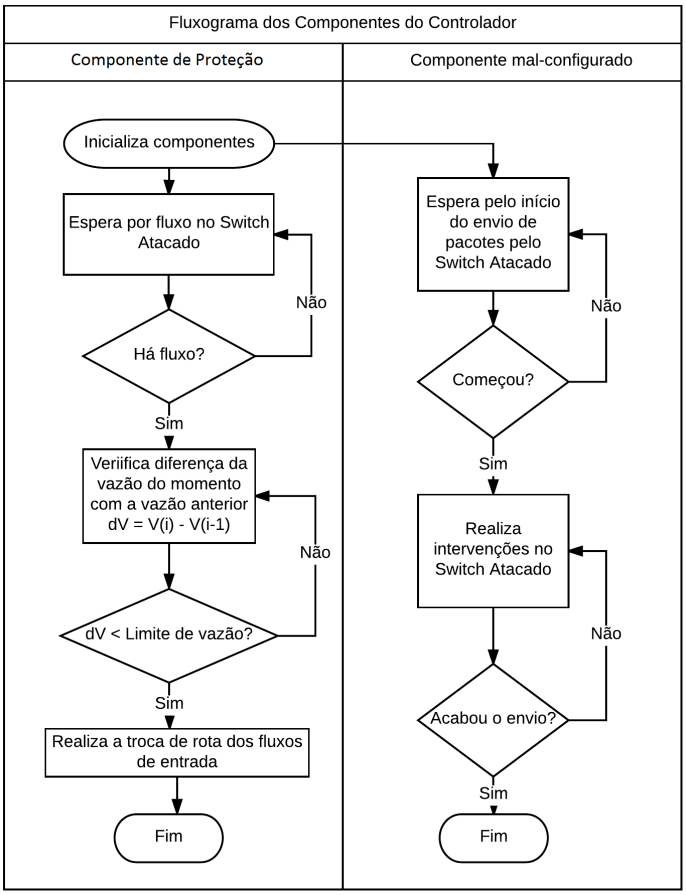

Figura 5.3: Fluxograma de funcionamento dos componentes do Controlador POX.

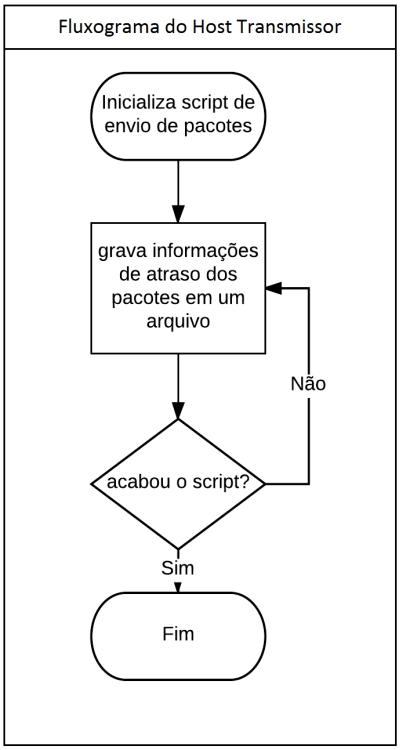

Figura 5.4: Fluxograma de funcionamento do Host Transmissor.

resiliência é mostrado no Algoritmo 1.

Algoritmo 1: Algoritmo do componente de mitigação de falhas no roteamento do controlador POX

Data: Vazão de pacotes no switch atacado

Result: Troca de rota do switch atacado para o switch seguro

Inicialização;

while *há tráfego no switch atacado* **do**

  lê vazão no momento no switch atacado;

  verifica vazão anterior;

  if *vazão* $\leq$ *limiar de diminuição* **then**

    habilita switch alternativo;

    transfere rotas do switch atacado para o switch seguro;

    desabilita switch atacado;

  end if

end while

Como verificado na seção 4.2, para o ambiente de software, a interferência do Controlador POX que gera maior impacto do desempenho do Plano de Dados é a operação de remoção de regras no switch da rota entre o Host Transmissor e o Host Receptor (OpenVSwitch, único comutador no caso). Portanto, com o intuito de verificar o desempenho do componente de proteção proposto nessa dissertação quando sujeito à pior situação de interferência, a operação utilizada para os testes foi a de remoção de regras.

No Algoritmo 1, existe uma variável chamada "limiar de diminuição", esse valor foi calculado em observância aos valores encontrados na Figura 4.4. Os valores "base" de RTT variam entre 160 e 180 microssegundos. Para a situação de ataque, levando em consideração a menor interferência, quando o tempo entre remoções é da ordem de 1 a cada 100 milissegundos, vemos que a média dos RTTs está entre 210 e 220 microssegundos. Dessa maneira, se o valor de RTT aumentar em 25% em relação ao máximo sem ataque (cerca de 180 milissegundos), o sistema reconhece que está sob ataque e toma as devidas providências já explicitadas no Algoritmo 1.

5.2 RESULTADOS DE DESEMPENHO DO COMPONENTE DE PROTEÇÃO

Em (ZWILLINGER DANIEL; KOKOSKA, 2010) temos a definição de que a CDF (*cumulative distribution function*) de uma variável aleatória "X" é a probabilidade de que "X" obtenha um valor menor ou igual a "x". Para os resultados dos testes de utilização do componente de proteção proposto nessa dissertação, utilizaremos a CDF da variável aleatória "a", que representará o atraso para o três ambientes testados e descritos na seção 5.1. A Figura 5.5 nos mostra os resultados obtidos após a realização dos testes.

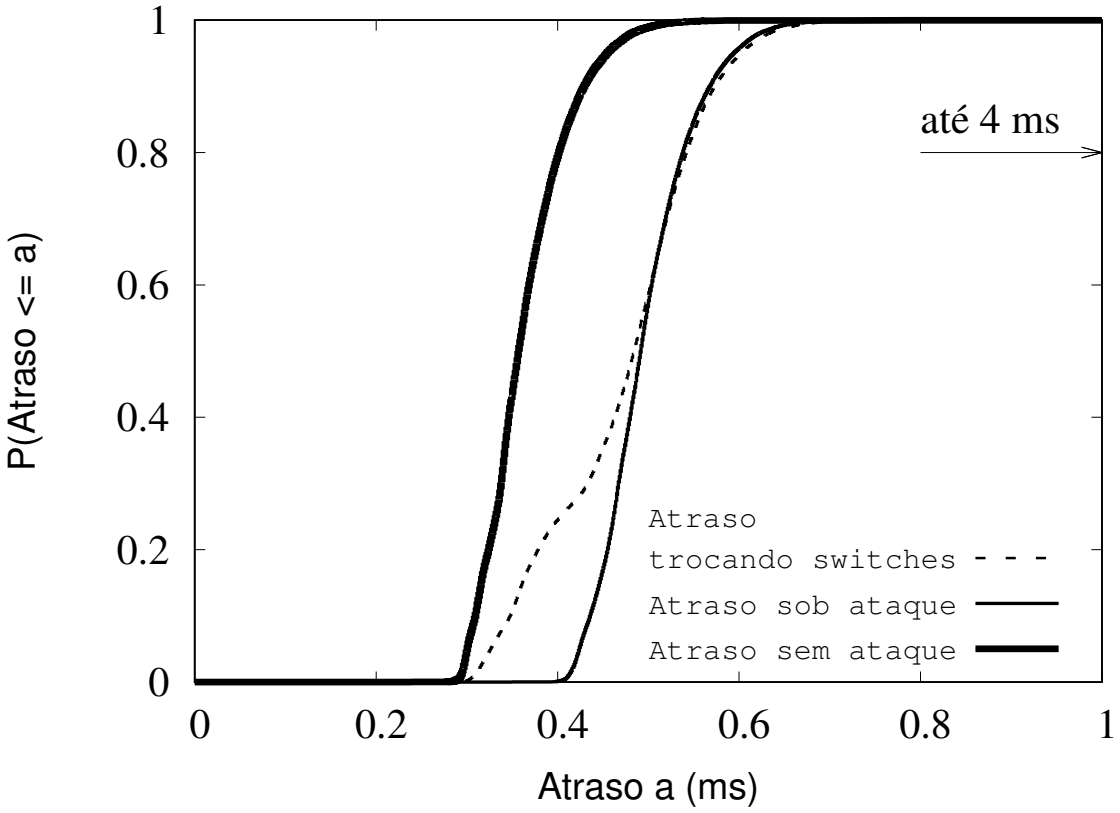

Figura 5.5: Distribuição acumulada dos testes de ataque.

Ao observar a Figura 5.5, podemos verificar três curvas referentes aos resultados dos três ambientes de teste. A curva do ambiente de atraso sem ataque é o nosso ambiente ótimo, nesse ambiente não há nenhuma interferência do Controlador POX no Switch Atacado. Já a curva do ambiente de atraso sob ataque é o nosso ambiente no pior caso, visto que nesse cenário há interferências do Controlador POX sob o Switch Atacado durante todo o processo de trocas de mensagens entre o Host Transmissor e o Host Receptor. A terceira curva, do ambiente de atraso trocando switches, é a curva que representa o

desempenho do componente proposto nessa dissertação, nesse ambiente há a interferência do Controlador POX sob o Switch Atacado até o momento em que outro componente do Controlador POX verifica que a vazão do sistema está diminuindo e toma a decisão de redirecionar o fluxo para o Switch Seguro.

A análise da terceira curva nos permite verificar que, na média, há um ganho de desempenho de 4,82% no tempo médio de atraso do sistema quando comparado ao tempo médio do pior caso, representado pela curva do ambiente sob ataque.

6 CONCLUSÕES E TRABALHOS FUTUROS

Nessa dissertação, levando em consideração o Problema do Administrador Mal configurado, é proposto encontrar a resposta para duas perguntas (i) qual é de fato o impacto gerado pela má configuração do sistema? e (ii) como aumentar a resiliência do sistema com o intuito de diminuir esse impacto?

Para responder à primeira pergunta, foram apresentadas metodologias de avaliação de desempenho de três maneiras de interferência no sistema, em dois ambientes, virtual e real, a fim de verificar qual dessas maneiras, para cada ambiente, produz o maior impacto no desempenho da rede.

No que diz respeito ao ambiente virtual, nossas simulações mostram que as instruções de remoção de regras são as que geram maior impacto na rede, seguida de perto pelas instruções de alteração e, com uma distância um pouco maior, pelas de inserção.

Já no que diz respeito ao ambiente real, o impacto das instruções de remoção e alteração de regras tiveram impacto similar ao evidenciado no ambiente virtual, contudo, as operações de inserção foram significativamente menores no que se refere ao impacto no desempenho da rede.

Para as questões de desempenho, trabalhos futuros incluem a investigação mais aprofundada do porquê desses resultados e a realização do mesmo experimento no ambiente real com switches em *hardware* para comparação de resultados obtidos.

Já para responder à segunda pergunta, foi apresentado um ambiente de testes em que se simula a ação de um componente mal configurado, bem como a de um componente de proteção simples capaz de mitigar os efeitos de diminuição de desempenho causados no Plano de Dados.

Os resultados obtidos nos mostram que a utilização do componente de proteção é capaz de gerar um ganho médio de até 4,82% nos resultados de desempenho do sistema, quando comparados aos do pior caso, ambiente com ataque, sem o uso do nosso componente.

Apesar de simples, nosso componente de proteção teve um resultado expressivo quando submetido aos testes propostos nessa dissertação. Dessa maneira, podemos verificar que heurísticas mais complexas podem, com base nele, realizar outros tipos de cálculos e averiguações e obter resultados ainda mais expressivos.

A solução proposta pode ser entendida como uma tolerância à ataques, não é uma solução que visa acabar com eles. Como trabalhos futuros pretende-se criar um novo módulo, capaz de interagir com os demais módulos em execução, verificar o módulo que causa os ataques e reconfigurá-lo e/ou desligá-lo. Além disso, os mesmos incluem a realização de experimentos numa rede SDN maior, utilizando hardware real e com instrumentação de mais nós da rede (além do Controlador POX) para identificação de informações como a origem dos ataques por exemplo. Além disso, há a intenção de verificar como os ataques podem acontecer, não só dentro da rede, mas por intervenções externas e averiguar se o uso do nosso componente de proteção pode, também nesse cenário, propiciar melhoras de desempenho.

REFERÊNCIAS

AGARWAL, S.; KODIALAM, M.; LAKSHMAN, T. V. Traffic engineering in software defined networks. In: **2013 Proceedings IEEE INFOCOM**, 2013. p. 2211–2219. ISSN 0743-166X. Disponível em: <http://ieeexplore.ieee.org/document/6567024/>.

AKYILDIZ, I. F.; LEE, A.; WANG, P.; LUO, M.; CHOU, W. A roadmap for traffic engineering in sdn-openflow networks. **Comput. Netw.**, Elsevier North-Holland, Inc., New York, NY, USA, v. 71, p. 1–30, out. 2014. ISSN 1389-1286. Disponível em: <http://dx.doi.org/10.1016/j.comnet.2014.06.002>.

BANERJEE, S.; KANNAN, K. Tag-in-tag: Efficient flow table management in sdn switches. In: **10th International Conference on Network and Service Management (CNSM) and Workshop**, 2014. p. 109–117. ISSN 2165-9605. Disponível em: <http://ieeexplore.ieee.org/document/7014147/>.

CAESAR, M.; CALDWELL, D.; FEAMSTER, N.; REXFORD, J.; SHAIKH, A.; MERWE, J. van der. Design and implementation of a routing control platform. In: **Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2**, 2005. (NSDI'05), p. 15–28. Disponível em: <http://dl.acm.org/citation.cfm?id=1251203.1251205>.

CASADO, M.; FREEDMAN, M. J.; PETTIT, J.; LUO, J.; GUDE, N.; MCKEOWN, N.; SHENKER, S. Rethinking enterprise network control. **IEEE/ACM Transactions on Networking**, v. 17, n. 4, p. 1270–1283, Aug 2009. ISSN 1063-6692. Disponível em: <http://ieeexplore.ieee.org/document/5169973/>.

CHAMBERS, J.; CLEVELAND, W.; KLEINER, B.; TUKEY, P. Graphical methods for data analysis. **The Wadsworth Statistics/Probability Series. Boston, MA: Duxury**, 1983.

CHUNG, C.-J.; KHATKAR, P.; XING, T.; LEE, J.; HUANG, D. Nice: Network intrusion detection and countermeasure selection in virtual network systems. **IEEE**

Trans. Dependable Secur. Comput., IEEE Computer Society Press, Los Alamitos, CA, USA, v. 10, n. 4, p. 198–211, jul 2013. ISSN 1545-5971. Disponível em: <http://dx.doi.org/10.1109/TDSC.2013.8>.

COSTA, L. C. **Balanceamento de Carga Utilizando Planos de Dados OpenFlow Comerciais**. Dissertação (Mestrado) — UFJF, June 2016. Disponível em: <http://www.ufjf.br/pgcc/files/2014/06/Leonardo-Costa.pdf>.

DCAN. **Devolved control of atm networks**. July 1995. [Online; acessado em 25-Julho-2017]. Disponível em: <http://www.cl.cam.ac.uk/research/srg/netos/projects/archive/dcan/>.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. The road to sdn: An intellectual history of programmable networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 44, n. 2, p. 87–98, abr. 2014. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2602204.2602219>.

FORCES. **ForCES protocol specification**. July 2004. [Online; acessado em 25-Julho-2017]. Disponível em: <https://tools.ietf.org/html/draft-ietf-forces-protocol-00>.

GSMP. **General switch management protocol (GSMP) v3 specification**. July 2002. [Online; acessado em 25-Julho-2017]. Disponível em: <https://www.ietf.org/rfc/rfc3292.txt>.

HOTSDN '13: Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, 2013. ISBN 978-1-4503-2178-5.

HU, F.; HAO, Q.; BAO, K. A survey on software-defined network and openflow: From concept to implementation. **IEEE Communications Surveys Tutorials**, v. 16, n. 4, p. 2181–2206, Fourthquarter 2014. ISSN 1553-877X. Disponível em: <http://ieeexplore.ieee.org/document/6819788/>.

KEMPF, J.; BELLAGAMBA, E.; KERN, A.; JOCHA, D.; TAKACS, A.; SKOLDSTROM, P. Scalable fault management for openflow. In: **2012 IEEE International Conference on Communications (ICC)**, 2012. p. 6606–6610. ISSN 1550-3607. Disponível em: <http://ieeexplore.ieee.org/document/6364688/>.

KORDALEWSKI, D.; ROBERE, R. **A Dynamic Algorithm for Loop Detection in Software Defined Networks**. July 2012. [Online; acessado em 25-Julho-2017]. Disponível em: <http:// www.cs.toronto.edu/robere/paper/networkgraph-1214.pdf>.

KREUTZ, D.; RAMOS, F. M. V.; VERÍSSIMO, P. E.; ROTHENBERG, C. E.; AZODOLMOLKY, S.; UHLIG, S. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE**, v. 103, n. 1, p. 14–76, Jan 2015. ISSN 0018-9219. Disponível em: <http://ieeexplore.ieee.org/document/6994333/>.

KUŹNIAR, M.; PEREŠÍNI, P.; KOSTIĆ, D. **What you need to know about SDN control and data planes**, 2014. Disponível em: $<https://infoscience.epfl.ch/record/199497/files/switches-tr-oct14_1.pdf>$.

KUŹNIAR, M.; PEREŠÍNI, P.; KOSTIĆ, D. What you need to know about sdn flow tables. In: _____. **Passive and Active Measurement: 16th International Conference, PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings**, 2015. p. 347–359. ISBN 978-3-319-15509-8. Disponível em: $<https://doi.org/10.1007/978-3-319-15509-8\_26>$.

LAKSHMAN, T. V.; NANDAGOPAL, T.; RAMJEE, R.; , K. S. ; WOO, T. The softrouter architecture. In: , 2004. Disponível em: <https://www.microsoft.com/en-us/research/publication/the-softrouter-architecture/>.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network innovation using openflow: A survey. **IEEE Communications Surveys Tutorials**, v. 16, n. 1, p. 493–512, First 2014. ISSN 1553-877X. Disponível em: <http://ieeexplore.ieee.org/document/6587999>.

MACEDO, D. F.; GUEDES, D.; VIEIRA, L. F. M.; VIEIRA, M. A. M.; NOGUEIRA, M. Programmable networks - from software-defined radio to software-defined networking. **IEEE Communications Surveys Tutorials**, v. 17, n. 2, p. 1102–1125, Secondquarter 2015. ISSN 1553-877X. Disponível em: <http://ieeexplore.ieee.org/document/7039225/>.

MATSUMOTO, S.; HITZ, S.; PERRIG, A. Fleet: Defending sdns from malicious administrators. In: **Proceedings of the Third Workshop on Hot Topics in Software Defined Networking**, 2014. (HotSDN '14), p. 103–108. ISBN 978-1-4503-2989-7. Disponível em: <http://doi.acm.org/10.1145/2620728.2620750>.

MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. Openflow: Enabling innovation in campus networks. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1355734.1355746>.

MEHDI, S. A.; KHALID, J.; KHAYAM, S. A. Revisiting traffic anomaly detection using software defined networking. In: _____. **Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011, Menlo Park, CA, USA, September 20-21, 2011. Proceedings**, 2011. p. 161–180. ISBN 978-3-642-23644-0. Disponível em: $<https://doi.org/10.1007/978-3-642-23644-0\_9>$.

NETWORKS, J. **What's Behind Network Downtime?** June 2008. [Online; acessado em 25-Julho-2017]. Disponível em: <https://www-935.ibm.com/services/au/gts/pdf/200249.pdf>.

NUNES, B. A. A.; MENDONCA, M.; NGUYEN, X. N.; OBRACZKA, K.; TURLETTI, T. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys Tutorials**, v. 16, n. 3, p. 1617–1634, Third 2014. ISSN 1553-877X. Disponível em: <http://ieeexplore.ieee.org/document/6739370/>.

QU, Y.; ZHOU, S.; PRASANNA, V. K. Scalable many-field packet classification on multi-core processors. In: **2013 25th International Symposium on Computer Architecture and High Performance Computing**, 2013. p. 33–40. ISSN 1550-6533. Disponível em: <http://ieeexplore.ieee.org/document/6702577/>.

SCHLESINGER, C. **Language-Based Security for Software-Defined Networks**. July 2017. [Online; acessado em 25-Julho-2017]. Disponível em: <http://www.cs.princeton.edu/cschlesi/isolation.pdf>.

SHARMA, S.; STAESSENS, D.; COLLE, D.; PICKAVET, M.; DEMEESTER, P. Enabling fast failure recovery in openflow networks. In: **2011 8th International Workshop on the Design of Reliable Communication Networks (DRCN)**, 2011. p. 164–171. Disponível em: <http://ieeexplore.ieee.org/document/6076899/>.

SHIN, S.; PORRAS, P.; YEGNESWARAN, V.; FONG, M.; GU, G.; TYSON, M. **FRESCO: Modular Composable Security Services for Software-Defined Networks**. 4 2013. Disponível em: <http://www.csl.sri.com/users/vinod/papers/fresco.pdf>.

SHIRALI-SHAHREZA, S.; GANJALI, Y. Rewiflow: Restricted wildcard openflow rules. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 45, n. 5, p. 29–35, set. 2015. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/2831347.2831352>.

TENNENHOUSE, D. L.; SMITH, J. M.; SINCOSKIE, W. D.; WETHERALL, D. J.; MINDEN, G. J. A survey of active network research. **IEEE Communications Magazine**, Institute of Electrical and Electronics Engineers Inc., v. 35, n. 1, p. 80–86, January 1997. ISSN 0163-6804. Disponível em: <http://ieeexplore.ieee.org/document/568214/>.

ZWILLINGER DANIEL; KOKOSKA, S. **CRC Standard Probability and Statistics Tables and Formulae**, 2010. ISBN 978-1-58488-059-2.