CURSO DE ENGENHARIA DE COMPUTAÇÃO

Cássio Giordani Tatsch

UMA PROPOSTA DE IPS PARA SDN UTILIZANDO O PROTOCOLO OPENFLOW

Santa Cruz do Sul

2016

Cássio Giordani Tatsch

UMA PROPOSTA DE IPS PARA SDN UTILIZANDO O PROTOCOLO OPENFLOW

Trabalho de Conclusão apresentado ao Curso de Engenharia de Computação da Universidade de Santa Cruz do Sul, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. Me. Charles Varlei Neu

Santa Cruz do Sul
2016

RESUMO

A segurança tem sido uma das principais preocupações na comunidade de redes devido ao abuso de recursos e intrusão de fluxos maliciosos. Sistemas de Detecção de Intrusão, ou *Intrusion Detection System* (IDS) e Sistemas de Prevenção de Intrusão, ou *Instrusion Prevention System* (IPS) já foram muito utilizados em redes tradicionais para prover a sua segurança. Porém, com o crescimento e a evolução da Internet, muitas alternativas acabaram não sendo utilizadas devido à falta de uma estrutura de desenvolvimento global e testes em ambientes reais. O surgimento do paradigma de Redes Definidas por *Software*, ou *Software Defined Networking* (SDN), e do protocolo OpenFlow, trouxe maior flexibilidade para o programação de novos protocolos e realização de testes em ambientes reais, além da possibilidade de implementação gradativa em ambientes de produção. No entanto, a simples migração de alternativas de IDS/IPS tradicionais para ambientes SDN não é eficaz o suficiente para detectar e prevenir ataques maliciosos. Com isso, diversos trabalhos tem abordado o desenvolvimento de sistemas de detecção e prevenção de intrusão para SDN, baseados em técnicas de análise de fluxos recebidos, o que ocasiona um aumento na latência de chaveamento por parte dos comutadores. Com base no contexto atual e beneficiado pelas características flexíveis de SDN, este trabalho de conclusão de curso provê o estudo e uma abordagem para o desenvolvimento de um sistema de prevenção de intrusão eficiente em SDN utilizando informações estatísticas do protocolo OpenFlow.

Palavras-chave: IDS, IPS, OpenFlow, SDN, Segurança.

ABSTRACT

Security has been one of the major concerns in computer networks comunity due to resource abuse and malicious flows intrusion. Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) have been widely used in traditional networks to provide their safety. However, with the growth and the evolution of the Internet, many alternatives were not used because of the lack of a global development framework and testing in real environments. The emergence of Software Defined Networking (SDN) paradigm and the OpenFlow protocol, brought greater flexibility to programming new protocols and performing tests in real environments, besides the possibility of implementation in production environments. One the other hand, the simple migration of traditional IDS/IPS alternatives to an SDN environment is not efective enough to detect and prevent from malicious attacks. This way, several studies have addressed the development of intrusion detection and prevention systems in SDN, based on received flows data analysis, resulting in an increase latency in switches switching. Based on current context and benefited from flexible features of SDN, this paper provides the study and an approach to a development of an intrusion prevention system for SDN using statistical information from the OpenFlow protocol.

Keywords: IDS, IPS, OpenFlow, SDN, Security.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

API	*Application Programming Interface*
ARP	*Address Resolution Protocol*
BGP	*Border Gateway Protocol*
CERT.br	Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil
DDoS	*Distributed Denial of Service*
DHCP	*Dynamic Host Configuration Protocol*
DNS	*Domain Name System*
DoS	*Denial of Service*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IP	*Internet Protocol*
IPS	*Instrusion Prevention System*
JSON	*JavaScript Object Notation*
NAT	*Network Address Translation*
NFV	*Network Functions Virtualization*
NV	*Network Virtualization*
OSPF	*Open Shortest Path First*
QoS	*Quality of Service*
REST	*Representational State Transfer*
SDN	*Software Defined Networking*
SNMP	*Simple Network Management Protocol*
SO	Sistema Operacional
SOM	*Self Organized Maps*
SSL	*Secure Socket Layer*
TCP	*Transmission Control Protocol*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Com o advento de novos recursos que podem ser provisionados sob demanda através da Internet, a chamada 'nuvem' envolve milhares de conexões entre servidores e usuários, provendo armazenamento, comunicações unificadas e alocação de recursos. Pessoas passaram a estar conectadas o tempo todo, com qualquer dispositivo que permita o intercâmbio de dados (SEEBER; STIEMERT; RODOSEK, 2015). As tecnologias atuais já não conseguem mais atender à todas as exigências dos usuários por causa de sua complexidade e quantidade de protocolos utilizados. Geralmente desenvolvidas e definidas de forma isolada e, para dificultar ainda mais, alguns fabricantes desenvolvem protocolos proprietários (KIM; FEAMSTER, 2013; SOARES *et al.*, 2015). Desta forma, a tarefa de alocar novos dispositivos para escalar a rede torna-se cada vez mais complexa e lenta, inviabilizando a implantação de novas tecnologias em uma rede já existente (KREUTZ *et al.*, 2014). Essa inflexibilidade na arquitetura da Internet, traz um desafio para os pesquisadores da área, pois seus experimentos acabam não sendo validados em redes reais.

O paradigma de Redes Definidas por Software (SDN) e o protocolo OpenFlow oferecem um caminho para vencer este desafio, por meio de uma solução de implementação gradativa em redes de produção. O paradigma SDN possibilita a rápida configuração de uma rede conforme a demanda de serviços, além de permitir adição de recursos, independente da fabricante (SAYEED; SAXENA, 2015). A arquitetura SDN provê uma abstração entre o plano de controle e o plano de dados, transformando *switches* de rede em encaminhadores de pacote e a lógica, por sua vez, passa para controladores centralizados (KREUTZ; RAMOS; VERISSIMO, 2013) facilitando o gerenciamento da rede. Para que essa abstração ocorra, é necessária uma comunicação entre os planos de controle e dados. Com este propósito, foi desenvolvido o protocolo OpenFlow, sendo atualmente o protocolo com maior relevância em SDN (MCKEOWN *et al.*, 2008). Em um *switch* OpenFlow, há tabelas de fluxo (*flow tables*) contendo regras para a manipulação de pacotes e estatísticas. Cada regra corresponde a um subconjunto do tráfego e para cada uma delas uma sequência de ações podem ser tomadas, como descarte, modificação do cabeçalho ou encaminhamento, além disso são armazenadas estatísticas de cada fluxo recebido (OPEN NETWORKING FOUNDATION, 2016).

SDN provê uma visão global da rede, o que facilita o seu controle, porém a segurança continua sendo uma das principais preocupações na comunidade de rede devido ao abuso de recursos e intrusos maliciosos. Somente em 2015, foram reportadas 722.205 incidentes de

segurança à CERT (CERT.BR, 2016), responsável por tratar incidentes de segurança e computadores que envolvam redes conectadas à Internet brasileira, destes, 53% foram de ataques do tipo *port scan*, ou em português, varredura de porta.

Tradicionalmente, Sistemas de Detecção de Intrusão, ou *Intrusion Detection Systems* - IDS (COMER, 1988) são consideradas ferramentas comuns para detectar e prevenir ataques maliciosos dentro de uma rede. Esses sistemas monitoram eventos de rede e tráfego para identificar atividades maliciosas e em seguida, emitir alertas e informar os administradores do sistema. A natureza de "detecção e alerta" das soluções de IDS atuais exige profissionais especialistas em segurança. Além disso, IDSs atuais carecem da atividade pró-ativa para evitar ataques em seu estágio inicial. Assim, Sistemas de Prevenção de Intrusão, ou *Intrusion Prevention Systems* - IPS, se tornaram preferência sobre o IDS, a fim de, automaticamente, tomar a ação sobre uma atividade suspeita na rede. Basicamente, o IPS pode ser construído com base em um IDS pois a função de detecção é necessária em uma solução de IPS, contudo, a maioria das soluções existentes de IPS são desenhadas para a rede tradicional e uma simples migração para SDN não é suficientemente eficaz para detectar e defender de ataques maliciosos (XIONG, 2013).

Observando as estatísticas de segurança e a falta de soluções IPS para SDN, este trabalho de conclusão apresenta uma metodologia de sistema de prevenção contra ataques de varredura de porta (*port scan*) em SDN, utilizando como fonte de informações para análise de tráfego malicioso, os contadores das tabelas de fluxos de *switches* OpenFlow, possibilitando assim, uma medida de detecção eficiente e proteção a nível global da rede.

O restante deste trabalho está dividido como segue. O Capítulo 2 apresenta os conceitos necessários para o entendimento deste trabalho em SDN, descrevendo o funcionamento e arquitetura de SDN, além de desafios no que diz respeito à segurança. O Capítulo 3 apresenta algumas soluções IDS/IPS existentes na literatura bem como os objetivos deste. A proposta deste trabalho é apresentada no capítulo 4, bem como os detalhes de sua implementação e funcionamento. E por fim, no Capítulo 5, são descritas algumas considerações sobre este trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados tópicos relacionados ao presente trabalho que se fazem necessários para o entendimento deste Trabalho de Conclusão. O capítulo está organizado da seguinte forma: na seção 2.1 são abordados os conceitos sobre SDN e Openflow; na seção 2.2 é feita uma abordagem referente a virtualização de redes; na seção 2.3 é abordado o software de simulação de rede Mininet, utilizado para realização de testes neste trabalho; e por fim, na seção 2.4, é discutido o assunto de segurança em redes de computadores, os principais tipos de ataques e soluções.

2.1 Redes definidas por software e OpenFlow

As redes de computadores se tornaram parte da infraestrutura crítica de empresas, escolas e residências, tendo crescido bastante desde a sua origem vem tomando aplicações cada vez mais diversificadas. O sucesso das redes de computadores se deve, em grande medida, à simplicidade de seu núcleo. Na arquitetura atual, a inteligência está localizada nos sistemas de borda, enquanto que o núcleo é simples e transparente. Embora essa simplicidade tenha tido sucesso, também é razão para o seu engessamento, pois apresenta limitações estruturais que são difíceis de serem resolvidas, tais como escalabilidade, mobilidade e gerenciamento de serviço (CLARK *et al.*, 2004).

Por causa desta expansão, o trabalho dos pesquisadores da área tornou-se muito mais importante, porém mesmo com o grande número de equipamentos e protocolos criados para suportar essa expansão, surgiu uma enorme barreira: a maioria das ideias que surgem não conseguem ser testadas por falta de maneiras práticas que possibilitem a realização de experimentos com novos protocolos em uma rede realista, para que possa obter a confiança necessária para uma implantação em escala global (MCKEOWN *et al.*, 2008).

Como apresentado por Kreutz *et al.* (2014), redes de computadores podem ser separadas em três planos: de controle, de dados e de gerência. Entende-se por plano de controle a porção da rede que abriga os *softwares* responsáveis por ditar o comportamento da rede, se incluem nestes protocolos como *Border Gateway Protocol* (BGP) (REKHTER; LI; HARES, 2006) e *Open Shortest Path First* (OSPF) (MOY, 1998). O plano de dados é o que executa o encaminhamento dos pacotes com base nas regras ditadas pelo plano de controle. Já o plano de gerência inclui serviços utilizados para monitorar a rede e configurar remotamente o plano de controle utilizando protocolos como *Simple Network Management Protocol* (SNMP) (CASE *et*

al., 1990). Em síntese, o plano de gerência define as regras da rede, o de controle implementa essas regras e o plano de dados realiza o encaminhamento de pacotes de acordo com as regras impostas pelo plano de controle. Em redes *Internet Protocol* (IP) tradicionais, os planos de controle e dados são acoplados em um mesmo hardware, tornando a arquitetura de rede complexa e por consequência dificulta a sua configuração e o seu gerenciamento.

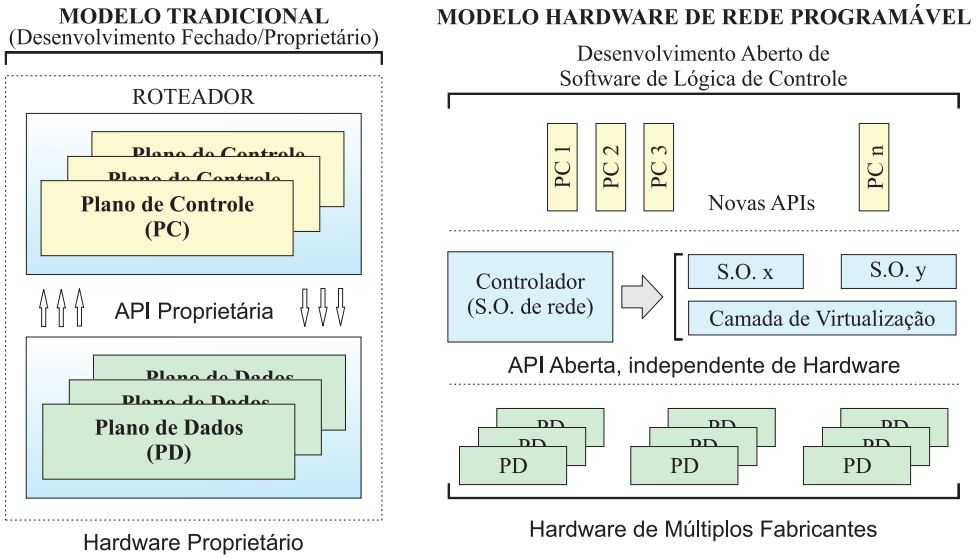
Para tentar contornar esse problema, a comunidade de pesquisa em redes de computadores tem investido em iniciativas que levem a implantação de redes com maiores recursos de programação, de forma que novas tecnologias possam ser inseridas na rede de forma gradual. Exemplos de iniciativas desse tipo são as propostas de redes ativas (*active networks*) (TENNENHOUSE *et al.*, 1997), de *testbeds* como o PlanetLab (CHUN *et al.*, 2003), GENI (TURNER, 2006) e, mais recentemente o FIBRE (SALMITO *et al.*, 2014). Redes ativas, tiveram pouca aceitação pela necessidade de alteração dos elementos de rede para permitir que se tornassem programáveis. Iniciativas mais recentes como PlanetLab, GENI e FIBRE, apostam na adoção de recursos de virtualização para facilitar a transição para novas tecnologias. Apesar de serem consideradas de grande potencial a longo prazo, tais iniciativas ainda enfrentam desafios como garantir o desempenho exigido pelas aplicações utilizadas hoje utilizando-se tais elementos virtualizados (GUEDES, 2012).

Uma outra forma de abordar esse problema, consiste em estender o *hardware* de encaminhamento de pacotes de forma mais restrita. Considerando-se que a operação que necessita de alto desempenho nos elementos de comutação é o encaminhamento de pacotes (plano de dados), algumas iniciativas propõem manter essa operação pouco alterada, para manter a viabilidade de desenvolvimento de hardware de alto desempenho, mas com uma possibilidade de maior controle por parte do administrador da rede.

SDN introduz uma perspectiva flexível para programar e manter a operacionalidade da rede buscando desacoplar os planos de dados e de controle, desta forma, tira-se a autonomia dos equipamentos de rede que se tornam apenas encaminhadores de pacotes. Já a lógica de controle é movida para uma entidade externa, centralizada, implementada em *software*. Esta, chamada de controlador, tem por funcionalidade prover a lógica de funcionamento da rede o que torna o desenvolvimento de serviços mais facilmente implementáveis, já que não há a necessidade de implementação em cada dispositivo. No plano de dados, o encaminhamento de pacotes, que antes era baseado em destino, passa a ser por fluxo que é definido pela combinação de campos das camadas de enlace, de rede ou de transporte, segundo o modelo *Transmission Control Pro-*

tocol (TCP)/IP. Dessa forma mantém-se o alto desempenho no encaminhamento de pacotes em *hardware*, aliado à flexibilidade de se implementar aplicações em *software*, utilizando protocolo aberto para programação da lógica do equipamento que é abstraída dos dispositivos de encaminhamento (KIM; FEAMSTER, 2013; TOOTOONCHIAN; GANJALI, 2010; ROTHENBERG *et al.*, 2010). Pensando nisso, nasceu o OpenFlow (MCKEOWN *et al.*, 2008), que por sua vez, deu origem ao conceito de *Software Defined Networking*, ou redes definidas por software. A Figura 1 apresenta um comparativo entre o modelo tradicional de rede, onde onde ambos os planos, de controle e de dados, são localizados em um mesmo dispositivo e o modelo SDN que possui controle centralizado e apenas o plano de dados no dispositivo comutador.

Figura 1 – Modelos de rede tradicional e SDN



Fonte: Rothenberg *et al.* (2010)

O protocolo OpenFlow é implementado em ambos os planos e dispõe de um protocolo de comunicação entre o controlador e *switches*. Para garantir a confiabilidade dessa comunicação é recomendado a utilização do protocolo *Secure Socket Layer* (SSL) (FREIER; KARLTON; KOCHER, 2011) porém algumas alternativas incluem TCP, utilizadas especialmente em redes virtuais devido à sua simplicidade, pois não necessitam de chaves criptográficas (ROTHENBERG *et al.*, 2010).

OpenFlow explora a existência de tabelas de fluxo (*flow tables*) em dispositivos *Ethernet* modernos. Essas tabelas são alimentadas em tempo de execução e utilizadas para implementar *firewalls* (OPPLIGER, 1997), *Network Address Translation* (NAT) (SRISURESH; EGEVANG, 2001), *Quality of Service* (QoS) (AURRECOECHEA; CAMPBELL; HAUW, 1998) e coleta de estatísticas. Normalmente são proprietárias mas há um conjunto de funções que são comuns na

maioria dos dispositivos. Com isso, uma forma padrão de manipulação das tabelas de fluxo pode ser implementada, independente de fornecedor. Desta maneira, OpenFlow fornece um padrão para manipulação das tabelas de fluxo, permitindo assim a partição do tráfego, o agrupamento ou isolamento da rede e o processamento ou controle do fluxo de dados, da forma desejada com base no fluxo (KONTESIDOU, 2012).

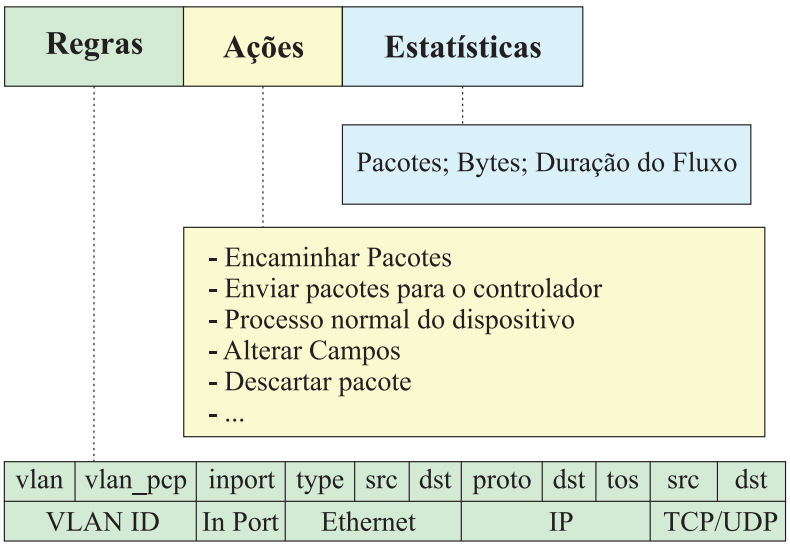
Os principais componentes de uma arquitetura OpenFlow são:

- Comutadores (*switches*) *OpenFlow*;
- Controlador; e
- Protocolo de comunicação.

2.1.1 Comutadores OpenFlow

É o elemento responsável pelo encaminhamento dos pacotes pela rede. Podem ser específicos para OpenFlow, ou ter suporte ao mesmo. No comutador (*switch*) é mantida uma tabela de fluxo (*flow table*) que mantêm informações sobre como os pacotes serão processados e estatísticas, prioridade e tempo limite para novos fluxos. Além disso, cada regra é composta por um conjunto de campos do cabeçalho do pacote que podem ser visualizadas na Figura 2 assim como as informações de ações e estatísticas.

Figura 2 – Tabela de fluxo

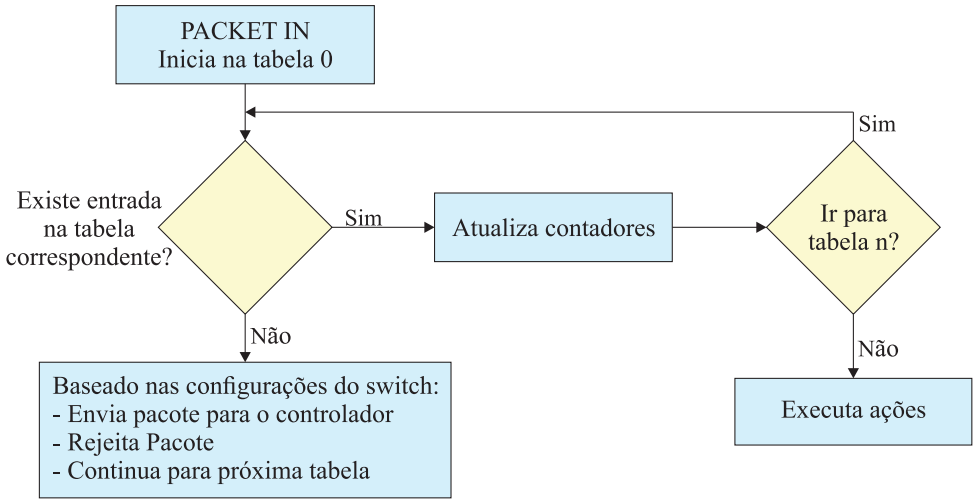


Fonte: Costa (2014)

Quando um pacote chega a um equipamento com OpenFlow habilitado, os cabeçalhos do pacote são comparados (*match*) às regras das entradas das tabelas de fluxos, os contadores

são atualizados e as ações correspondentes são realizadas. Se não houver correspondência (*table miss*) entre o pacote e alguma entrada da tabela de fluxos, o pacote é encaminhado, por completo, ao controlador. Alternativamente, apenas o cabeçalho é encaminhado ao controlador mantendo o pacote armazenado no *buffer* do *hardware*. A Figura 3 ilustra através de um diagrama simplificado, o tratamento recebido por pacotes em um *switch* OpenFlow. Os pacotes que chegam ao controlador normalmente correspondem ao primeiro pacote de um novo fluxo ou, em função do tipo de pacote e da aplicação, o controlador pode decidir por instalar uma regra no *switch* para que todos os pacotes de determinado fluxo sejam enviados para o controlador para serem tratados individualmente. Esse último caso corresponde, em geral, a pacotes de controle (*Internet Control Message Protocol* (ICMP) (POSTEL, 1981a), *Domain Name System* (DNS) (HOFFMAN; SULLIVAN; FUJIWARA, 2015), *Dynamic Host Configuration Protocol* (DHCP) (DROMS, 1997)) ou de protocolos de roteamento (OSPF (MOY, 1998), BGP (REKHTER; LI; HARES, 2006)). Todos os pacotes de uma mesma faixa de endereços IP, ou uma conexão TCP em determinada porta são considerados fluxos.

Figura 3 – Diagrama simplificado do tratamento de um pacote no *switch* OpenFlow



Fonte: Elaborado pelo autor a partir da especificação OpenFlow (OPEN NETWORKING FOUNDATION, 2014)

A cada pacote recebido, é realizada a atualização dos contadores na tabela de fluxo. Esses contadores são usados para geração de estatísticas, de maneira a monitorar o número de pacotes e bytes de cada fluxo, além do tempo de duração desde o seu início. O Quadro 1 apresenta alguns dos contadores disponíveis na tabela de fluxo, com o auxílio deste podem ser implementados recursos de monitoramento e segurança do tráfego na rede.

Quadro 1 – Contadores da tabela de fluxo

Contator	**Tamanho em bits**
Por Tabela	
Número de entradas Ativas	32
Número de pacotes pesquisados	64
Número de pacotes encontrados na tabela	64
Por fluxo	
Número de pacotes recebidos	64
Número de bytes recebidos	64
Duração (segundos)	32
Duração (nano segundos)	32
Por porta	
Número de pacotes recebidos	64
Número de pacotes transmitidos	64
Número de bytes recebidos	64
Número de bytes transmitidos	64
Número de pacotes perdidos no recebimento	64
Número de pacotes perdidos na transmissão	64
Número de erros recebidos	64

Fonte: Elaborado pelo autor a partir da especificação OpenFlow
(OPEN NETWORKING FOUNDATION, 2014)

2.1.2 Controlador OpenFlow

O controlador OpenFlow, como já citado, é o *software* responsável por tomar decisões e adicionar e remover as entradas na tabela de fluxos, de acordo com o objetivo desejado. O controlador exerce a função de uma camada de abstração da infraestrutura física, facilitando a criação de aplicações e serviços que gerenciem as entradas de fluxos na rede. Esse modelo assemelha-se a outros sistemas de *software* que proveem abstração do *hardware* e funcionalidade reutilizável. Dessa forma, o controlador OpenFlow atua como um Sistema Operacional (SO) para gerenciamento e controle das redes, e oferece uma plataforma com base na reutilização de componentes e na definição de níveis de abstração. Contudo, novas aplicações de rede podem ser desenvolvidas rapidamente (GUDE *et al.*, 2008).

O controlador fornece uma interface para criar, modificar e controlar o fluxo de tabelas do comutador. É executado normalmente em um servidor conectado à rede e pode ser um para todos os comutadores da rede, um para cada comutador ou um para um conjunto de comutadores. Portanto, a funcionalidade da rede de controle pode ser completamente ou localmente centralizada de acordo de como o gerenciamento dos comutadores é realizada. A exigência, no entanto, é que, se houver mais do que um controlador de processos, eles devem ter a mesma

visão da topologia da rede, em qualquer momento dado. A visão de rede inclui a topologia a nível de *switch*, as localizações dos usuários, *hosts*, *middleboxes* e outros elementos de rede e serviços. Além disso inclui todas as ligações entre os nomes e endereços.

Atualmente, existem várias implementações controlador disponíveis, entre os principais não comerciais estão (KREUTZ; RAMOS; VERISSIMO, 2013; XIA *et al.*, 2015):

- **NOX** - Desenvolvido em C++, foi o primeiro controlador OpenFlow (GUDE *et al.*, 2008). Porém não foi fortemente utilizado por causa de deficiências na sua implementação e na documentação.
- **POX** - Sucessor do NOX, foi desenvolvido como uma alternativa mais amigável e tem sido implementado por um grande número de engenheiros e programadores SDN. Comparando com NOX, POX tem um ambiente de desenvolvimento mais fácil de trabalhar com uma API razoavelmente bem escrita e documentada. Também fornece uma interface Web e é escrito em Python (MCCAULEY, 2016).
- **Beacon** - É um controlador SDN bem escrito e organizado. Escrito em Java, Beacon foi o primeiro controlador com o qual iniciantes pudessem trabalhar e criar um ambiente SDN, no entanto, era limitado à topologias de rede estrela (ERICKSON, 2013).
- **Floodlight** - Uma ramificação do Beacon. Enquanto que seu início tenha sido baseado no Beacon este foi desenvolvido utilizando Apache Ant, uma ferramenta popular para compilação e construção de *software*, o que tornou o desenvolvimento do Floodlight mais fácil e flexível. Floodlight possui uma comunidade ativa e um grande número de recursos que podem ser adicionados ao sistema. Possui interface baseada em java e baseada em Web, além de possuir uma Interface de Programação de Aplicações (*Application Programming Interface* (API)) *Representational State Transfer* (REST) ou, em português, Transferência de Estado Representacional (FLOODLIGHT, 2016).
- **OpenDayLight** - É um projeto colaborativo da Linux Fundation e tem sido altamente suportado por empresas como Cisco e Big Switch. Desenvolvido em Java, também inclui API REST e interface web. Possui suporte à SDN, *Network Virtualization* (NV) , ou Virtualização de redes (CHOWDHURY; BOUTABA, 2009) e *Network Functions Virtualization* (NFV), ou Virtualização da Funções da Rede (HAWILO *et al.*, 2014). Além disso, possui um grande número de módulos que podem ser utilizados para atender aos requisitos de uma organização (OPENDAYLIGHT, 2016).
- **Ryu NOS** - É um *framework* de SDN baseado em componentes. O Ryu fornece compo-

nentes de software com APIs bem definidas que tornam mais fácil para os desenvolvedores criar novas aplicações de gerenciamento e controle de rede. O Ryu suporta vários protocolos para gerenciar dispositivos de rede, como OpenFlow, Netconf, OF-config, etc. Sobre o OpenFlow, o Ryu suporta totalmente as extensões 1.0, 1.2, 1.3, 1.4, 1.5 e Nicira. Todo o código está disponível gratuitamente sob a licença Apache 2.0 (RYU SDN FRAMEWORK COMMUNITY, 2016).

2.1.3 Protocolo OpenFlow

O protocolo de comunicação entre os dois planos é realizada por três tipos de mensagens: controlador para o *switch*, assíncrona e simétricas.

Mensagens do tipo controlador para *switch* são mensagens que o controlador envia para obter informações sobre o estado do *switch*, como por exemplo verificar estatísticas de um determinado fluxo (OPEN NETWORKING FOUNDATION, 2014). Essas mensagens podem ser:

- ***Features***: ao estabelecer uma conexão, o controlador envia esta mensagem requisitando que o *switch* informe suas capacidades.
- ***Configuration***: o controlador envia parâmetros de configuração para os *switches*.
- ***Modify-State***: utilizado pelo controlador para gerenciar o estado dos *switches*, deletar ou modificar regras na tabela de fluxos.
- ***Read-State***: utilizado pelo controlador para coletar estatísticas das tabelas de fluxos do *switch*.
- ***Packet-Out***: utilizada pelo controlador para enviar pacotes por uma porta específica.
- ***Barrier***: utilizada para verificar se as dependências das mensagens foram alcançadas ou receber notificação sobre tarefas concluídas.
- ***Role Request***: mensagens usadas pelo controlador para configurar seu canal OpenFlow.

Mensagens assíncronas são enviadas pelo *switch* sem a solicitação do controlador. *Switches* enviam mensagens assíncronas para os controladores para denotar uma chegada de pacotes ou mudança de estado (OPEN NETWORKING FOUNDATION, 2014). Os principais tipos de mensagens assíncronas são descritas abaixo.

- ***Packet-In***: enviado pelo *switch* quando há uma ação explícita na tabela de fluxos para que seja enviado para o controlador ou quando não há um *match* para o pacote.

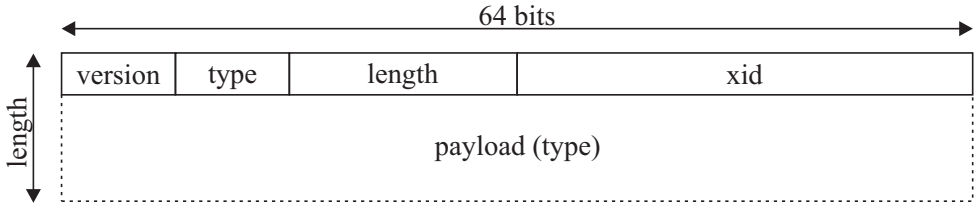
- ***Flow-Removed***: informa o controlador sobre a remoção de regras no *switch*.
- ***Port Status***: obtém status das portas do *switch*.
- ***Role Status***: *switch* informa o controlador sobre a alterações em suas regras.
- ***Controller Status***: *switch* informa o controlador sobre a mudança em um canal OpenFlow.
- ***Flow-monitor***: informa o controlador sobre uma mudança na tabela de fluxo.

Finalmente, mensagens simétricas são iniciadas tanto pelo controlador como pelo *switch* sem nenhuma solicitação, por exemplo o início de conexão entre controlador e *switch* (OPEN NETWORKING FOUNDATION, 2014). Essas mensagens são:

- ***Hello***: esta mensagem é utilizada no início da conexão entre *switch* e controlador.
- ***Echo***: utilizado para obter informações sobre a conexão entre *switch* e controlador como: latência, largura de banda e conectividade.
- ***Error***: o *switch* pode enviar mensagens para notificar problemas ao controlador por mensagens de erro.
- ***Experimenter***: na versão 1.5.0 do protocolo OpenFlow, esta mensagem é utilizada para adicionar funcionalidades experimentais.

Cada mensagem é enviada encapsulada em um pacote definido pelo protocolo OpenFlow e que é representado na Figura 4 abaixo.

Figura 4 – Formato da mensagem OpenFlow



Fonte: Elaborado pelo autor a partir de informações da especificação OpenFlow.

O campo *version* indica a versão do protocolo que está sendo utilizada, *type*, indica o tipo de mensagem que está sendo enviada. O campo *length* informa o tamanho da mensagem enquanto que *xid* representa o ID de transação associado à mensagem. Por último, o campo *payload* representa o corpo da mensagem, é neste campo onde são adicionados os diferentes tipos de mensagens apresentados anteriormente.

2.2 Virtualização

O conceito de virtualização de redes define uma infraestrutura de redes de computadores virtuais. São definidos por *software*, executando sobre máquinas físicas, de forma que toda infraestrutura virtual seja isolada da infraestrutura física, não interferindo na mesma. Um dos *softwares* mais usados na criação de redes virtuais em nível de *software* é o Xen (FERNANDES *et al.*, 2011). Esse programa é usado na criação de máquinas virtuais em computadores pessoais e servidores, e oferece a opção de criar roteadores virtuais que podem ser utilizados na interligação de máquinas virtuais para formação de uma rede. Em SDN a construção de redes virtuais acontece em nível de *hardware*, através da separação do tráfego da rede física em *slices*, porções de fluxo do tráfego total. O FlowVisor (SHERWOOD *et al.*, 2009) possibilita virtualização em SDN. O uso de virtualização de redes possibilita execução de experimentos distintos, sobre a mesma infraestrutura, em paralelo, sem interferência entre experimentos. Virtualização de redes também pode ser usada para isolamento de serviços, sendo assim, uma organização pode oferecer diversos serviços, com cada serviços executando em uma rede virtual diferente (WU *et al.*, 2010; MATTOS; DUARTE, 2012).

2.3 Emulador Mininet

Mininet (HANDIGOL *et al.*, 2012) é um emulador de rede para prototipação em SDN. A razão pela sua utilização deve-se ao fato de apenas alguns dispositivos de rede estarem disponíveis para SDN, uma vez que ainda não é uma tecnologia difundida a nível industrial. Além disso, a implementação de rede com elevado número de dispositivos de rede é muito difícil e dispendioso. Por isso, para contornar estes problemas, a virtualização foi realizada com a finalidade de prototipar e emular este tipo de tecnologia de rede e um dos mais importantes é o Mininet (WENDONG *et al.*, 2012). Mininet tem a capacidade de imitar diferentes tipos de elementos de rede, tais como: *host*, *switches* (camada de enlace), roteadores (camada de rede) e conexões. Ele funciona em um único núcleo de Linux(NEGUS; BRESNAHAN, 2015) e utiliza virtualização com a finalidade de emular uma rede completa que utiliza apenas um único sistema. No entanto, o *host*, roteadores e links criados são elementos do mundo real, embora eles sejam criados por meio de software (MININET, 2016).

Criar uma rede no Mininet é relativamente simples, pode-se usar linha de comando ou um componente chamado *miniedit.py*, que implementa uma interface gráfica para o Mininet, este porém, possui algumas limitações em relação à linha de comando. Pela linha de comando, ao

chamar o Mininet são passados os parâmetros sobre as características da rede como: topologia, número de *hosts*, *switches*, taxa de perda de pacotes, largura de banda, tipo de controlador, entre outros. O *switch* padrão é o OpenSwitch (PETTIT *et al.*, 2010), um *switch* virtual desenvolvido especialmente para trabalhar com o protocolo Openflow. Para estudo mais aprofundado, recomenda-se a leitura da sua documentação em (MININET, 2016).

2.4 Segurança em redes

A segurança no nível de rede indica uma área de pesquisa muito importante, já que os consumidores estão continuamente colocando seus dados em ambientes em nuvem e mais dados são transferidos através de grandes distâncias. A razão para esta evolução é a crescente popularidade dos serviços em nuvem, bem como a simplicidade e rápida capacidade dos recursos sob demanda. Os impactos variam de acordo com os tipos de ameaças, e como defesa são criados diversos sistemas de segurança que agem como barreira de proteção como *firewalls* por exemplo. Os principais tipos de ameaças serão estudadas a seguir, também é apresentado um estudo mais detalhado do ataque do tipo varredura, foco deste trabalho.

2.4.1 Tipos de ameaças

Dos diversos tipos de ameaças que podem ocorrer nas redes de computadores, destacam-se algumas que são notórias por causar frequentes transtornos aos usuário, tais como:

Fraude: Segundo Houaiss, Villar e Francisco (2001), é "qualquer ato ardiloso, enganoso, de má-fé, com intuito de lesar ou ludibriar outrem, ou de não cumprir determinado dever; logro". Esta categoria engloba as notificações de tentativas de fraudes, ou seja, de incidentes em que ocorre uma tentativa de obter vantagem, sejam por meios como correios eletrônicos não solicitados em massa (*spam*) e páginas falsas.

Ataque de negação de serviço (*Denial of Service (DoS)*): Um ataque de negação de serviço busca sobrecarregar serviços na rede dificultando o seu uso por usuários legítimos. Esse tipo de ataque, por sua natureza, pode produzir variações no volume de tráfego que normalmente são visíveis no gráfico de fluxo. Segundo Sperotto *et al.* no entanto, na detecção de intrusão por fluxo, é abordado implicitamente o problema de ataques DoS por força bruta, ou seja, um tipo de DoS que depende de esgotamento de recursos ou sobrecarga da rede. Infelizmente, é quase impossível de detectar diretamente ataques DoS semânticas.

Infestações viróticas automatizadas (*Worms*): São pequenos programas de computador

criados para a causar danos na máquina infectada e se auto replica pela rede, tirando cópias de si em cada computador (SPEROTTO *et al.*, 2010).

Exército de máquinas controladas sem autorização (*Bootnets*): Grupo de computadores comprometidos, chamados de computadores zumbis que são controlados remotamente por um centro de controle. *Botnets* são muito utilizados para lançamento de ataques como *spams*, DoS e *worms* (SPEROTTO *et al.*, 2010).

Varredura de portas maliciosa (*port scans*): Técnica utilizada para encontrar fraquezas de um computador ou de uma rede. Enquanto esta técnica não é um ataque real, os hackers a usam para detectar quais portas estão abertas em um computador. Baseado nas informações sobre portas abertas, o acesso não autorizado pode ser obtido.

Os métodos citados também podem também ser utilizados em conjunto, como por exemplo a utilização de *botnets* que, controlados remotamente, podem efetuar ataques DoS a um mesmo servidor e ao mesmo tempo. A esse tipo de ataque é dado o nome de Negação de Serviço Distribuída (*Distributed Denial of Service* (DDoS))

Do ponto de vista de segurança, existe uma quantidade crescente de incidência de ataques de negação de serviço, DoS, durante os últimos anos (SEEBER; STIEMERT; RODOSEK, 2015). Além disso, segundo a Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br), responsável por tratar incidentes de segurança e computadores que envolvam redes conectadas à Internet brasileira, foram reportados 722.205 incidências de segurança somente no ano de 2015, sendo mais da metade (53%), ataques do tipo *port scan*.

2.4.2 Técnicas de varredura de porta

Um dos tipos mais comuns de ataques, a varredura consiste no envio de diversos tipos de pacotes com o intuito de se conhecer mais sobre o nó alvo ou a rede em questão. Através das respostas obtidas para esses pacotes, o atacante é capaz de chegar a diversas informações que possam ajudar em futuros ataques de diversos tipos. Alguns tipos de informações que podem ser descobertas incluem (não somente): A atividade dos servidores, informações relativas a softwares utilizados no sistema, informações sobre o *firewall* e topologia da rede.

Uma das principais dificuldades nas soluções desse tipo de ataque é que as varreduras são consideradas atividades legais, e ocorrem na Internet de forma rotineira, inclusive com fins não maliciosos.

Antes de explorar as técnicas de varredura de porta, faz-se necessário o entendimento

de alguns conceitos de comunicação TCP. Para obter um serviço TCP, uma conexão necessita ser efetivada entre os computadores origem e destino. Esta conexão é realizada através dos chamados *sockets*, formados pelo par endereço IP e número de porta, de ambos, computador de origem e computador de destino. Entre estes dois *sockets* ocorre a transferência de segmentos.

Um segmento consiste em um cabeçalho TCP seguido, opcionalmente, por informação. Um cabeçalho TCP pode possuir seis *flags* que podem ser ativadas ou ativadas ao mesmo tempo (COMER, 1988), são elas:

- **SYN** - *bit* de sincronismo, é o *bit* que informa que este é um dos dois primeiros segmentos de estabelecimento da conexão.
- **ACK** - *bit* de reconhecimento, indica que o valor do campo de reconhecimento está carregando um reconhecimento válido.
- **PSH** - *bit* de *push*, este mecanismo, que pode ser acionado pela aplicação, informa ao TCP origem e destino que a aplicação solicita a transmissão rápida dos dados enviados, mesmo que ela contenha um número baixo de *bytes*, não preenchendo o tamanho mínimo do *buffer* de transmissão.
- **RST** - *bit* de *reset*, informa o destino que a conexão foi abortada neste sentido pela origem
- **FIN** - *bit* de terminação, indica que este pacote é um dos pacotes de finalização da conexão.

Em uma comunicação TCP, uma conexão deve ser estabelecida entre os dois pontos (*sockets*) para que a transferência de dados ocorra. Inicialmente a máquina emissora, também chamada de cliente, transmite um segmento cuja *flag* SYN é de 1 (para assinalar que se trata de um segmento de sincronização), com um número de ordem X, que se chama número de ordem inicial do cliente.

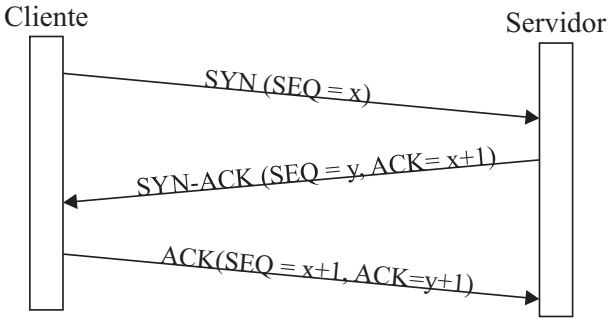
A seguir, a máquina receptora, chamada de servidor, recebe o segmento inicial que provém do cliente e envia-lhe um aviso de recepção, isto é, um segmento cuja *flag* ACK é de 1 e a *flag* SYN é de 1 (porque ainda se trata de uma sincronização). Este segmento contém o número de ordem do servidor, que é o número de ordem inicial do cliente. O campo mais importante deste segmento é o campo de aviso de recepção, que contém o número de ordem inicial do cliente, incrementado de 1.

Por último, o cliente transmite ao servidor um aviso de recepção, ou seja, um segmento cuja *flag* ACK é de 1, cuja *flag* SYN é de zero (não se trata mais de um segmento de sincroni-

zação). O seu número de ordem é incrementado e o número de aviso de recepção representa o número de ordem inicial do servidor, incrementado de 1.

Depois dessa sequência de trocas (Figura 5), também chamada de *handshake*, ou, aperto de mãos em português, as duas máquinas estão conectadas e a comunicação pode ser efetivada.

Figura 5 – Estabelecimento de conexão TCP



Fonte: Elaborado pelo autor.

Os passos a seguir são definidos pela RFC 793 (POSTEL, 1981b), utilizada pela grande maioria das implementações TCP e exploradas em técnicas de varredura.

- Quando um segmento SYN chega em um aporta aberta, é continuado o procedimento de *handshake* como discutido anteriormente;
- Quando um segmento SYN (ou FIN) chega em uma porta fechada, o segmento é descartado e um segmento RST é retornado para o cliente;
- Quanto um segmento FIN chega em uma porta que esteja aberta, o segmento é descartado.
- Quando um segmento RST chega em uma porta que esteja ouvindo (aberta), o segmento é descartado;
- Quando um segmento RST chega em uma porta que não esteja ouvindo (fechada), o segmento é descartado;
- Quando um segmento ACK chega à uma porta aberta, o mesmo é descartado e retornado um segmento RST.

Devido à sua natureza, *scans* podem facilmente criar um vasto número diferente de fluxos. Há três categorias de *scans*: *scan* horizontal - onde um *host* varre uma porta especifica em diferentes *hosts*; *scan* vertical - um *host* verifica várias portas em um outro *host* específico; e *block scan* que é a combinação dos dois (SPEROTTO *et al.*, 2010).

Existem várias técnicas de varredura de porta disponíveis e podem facilmente ser automatizadas por ferramentas como Nmap (LYON, 2016). Alguns métodos utilizados para varredura

serão estudados a seguir (VIVO *et al.*, 1999; CHRISTOPHER, 2001).

TCP Connect - É a forma mais comum de *scanning*. Basicamente uma conexão TCP regular (*handshake* completo) para cada porta definida na varredura. Para cada porta, a conexão pode resultar em sucesso, indicando uma porta aberta ou em falha caso contrário. Essa técnica é facilmente implementada pois não necessita de privilégios especiais e, do mesmo modo, é facilmente detectável. Através de *logs* do sistema alvo é possível verificar mensagens de requisição de conexão e de erro para as conexões negadas. Neste método, o *scanner* envia uma mensagem SYN para o sistema alvo. Se uma porta estiver (aberta) ouvindo com um serviço, a conexão se sucederá. Um SYN é retornado estabelecendo o número de sequência inicial. Um ACK considera o campo numérico de confirmação válido. Se a porta estiver (fechada) sem serviço ouvindo, uma mensagem RST é retornada, para reiniciar o pedido de conexão. Alguns exemplos de *scanners* podem ser Nmap, Amap e Blaster.

TCP SYN - Também conhecida por *Half Open* por não explorar um *handshake* completo. Nesta técnica o *scanner* envia uma mensagem SYN, como se estivesse pedindo uma conexão. Se responder como um RST, indica que a porta está fechada, e uma nova porta é testada. Se a resposta da máquina alvo for um SYN/ACK, indica que a porta se encontra ouvindo. O *scanner* envia então um RST cancelando o *handshake*. A vantagem desse tipo de *scanning* é o fato de, mesmo ainda podendo ser detectado, tentativas de conexões SYN são menos frequentemente registradas se comparadas com conexões TCP completas.

Exploração FIN - Neste método, quando um segmento FIN é enviado para uma porta fechada, o computador alvo responde com um TCP RST. Quando a porta estiver aberta, o segmento é ignorado e o computador alvo não responde. O *scanner* não recebe nunhuma resposta, pois não podem pertencer a uma conexão estabelecida.

Xmas Tree - é uma variação do método TCP FIN, neste, são utilizadas mensagens com prioridade TCP FIN/URG/PSH. Quando estiver ouvindo, o *host* alvo não responde, caso contrário, responde com um TCP RST.

TCP Null (sem flags ativos) - também é uma variação do método TCP FIN, neste, tem-se resposta para portas fechadas, mas não para portas abertas.

Varredura ACK - Técnica utilizada para identificar *firewalls*. Um segmento ACK que não pertença a nenhum conexão é gerado pelo *scanner*. Se um RST é devolvido pela máquina alvo, tanto em uma porta aberta como em uma fechada, as portas são classificadas como não tendo *firewall*.

Varredura ARP - Não se trata exatamente de varredura de portas mas essa técnica é utilizada para descobrir dispositivos ativos na rede local, para depois realizar a varredura de portas somente nos computadores ativos. O *scanner* envia uma série de pacotes de protocolo *Address Resolution Protocol* (ARP) (PLUMMER, 1982) e incrementa o valor do IP alvo a cada *broadcast*.

2.4.3 Sistemas de detecção e prevenção de intrusão

O isolamento da rede em redes virtuais permite uma maior segurança devido ao seu isolamento, porém problemas tradicionais relacionados à segurança continuam existindo em ambientes virtualizados pois 60% a 70% dos ataques à segurança da rede são de origem interna segundo Lynch (2006).

Uma das formas de se proteger desses ataques é monitorar o tráfego em busca de atividades maliciosas ou violação de políticas. Para realizar o monitoramento de pacotes na rede, a solução mais apropriada é o sistema de detecção de intrusão, que realiza o monitoramento passivo dos pacotes na rede. Porém, esse tipo de análise não permite que sejam tomadas ações para prevenir tais ataques, e então faz se necessário um sistema de prevenção de intrusão para bloquear esses pacotes.

Segundo Kruegel (2004), "Detecção de intrusão é o processo de identificar e responder à atividades maliciosas na computação e redes de dados". Uma tentativa de intrusão, também chamada de ataque, refere-se a uma série de ações em que um intruso tenta obter o ganho do sistema e o objetivo de um IDS é discriminar tentativas de intrusão e preparação de intrusão do uso normal do sistema.

Em redes tradicionais, para detectar e prevenir intrusos maliciosos na rede de dados, administradores normalmente necessitam implantar diversos detectores de intrusão em diferentes locais da rede, e então analisar dados do tráfego coletados localmente ou em um nodo centralizado. Infelizmente, arquiteturas IDS/IPS utilizadas atualmente possuem muitas barreiras para gerir nós distribuídos. Em primeiro lugar, os nós de detecção de intrusão devem suportar diferentes configurações. Como as configurações dependem da topologia da rede, configurações manuais e mudanças frequentes são inevitáveis para tornar a política em nós distribuídos eficaz e coerente. Em segundo lugar, algoritmos de detecção de intrusão eficazes normalmente são desenhados para um determinado tipo de ataque. Para desenvolver sistemas de detecção eficazes, mais e mais protocolos de proteção são criados, o que resulta na redução do desempenho

da rede. Além disso, dispositivos de rede normalmente possuem protocolos proprietários, o que torna mais difícil desenvolver interfaces de gerenciamento automáticas (WANG; HE; SU, 2015).

Vários trabalhos para IDS tem sido desenvolvidos desde o início de sua pesquisa no anos 1980. Essas propostas podem ser classificadas de acordo com várias características, como tipo de dados analisado (*logs* ou dados do pacote), tipo de análise (em tempo real ou *offline*) ou pelo tipo de processamento (centralizado ou distribuído). No entanto, os modelos de classificação mais conhecidos são os baseados em assinatura e os baseados em anomalia (AXELSSON, 2000).

Sistemas de detecção de intrusão baseados em assinatura realizam a detecção através da comparação de dados do pacote com uma base de dados conhecida. O IDS Snort (ROESCH, 1999) é um dos exemplos mais utilizados dessa técnica, verificando padrões de pacote através da análise de dados da carga útil (*payload*) do pacote. O Snortik (FAGUNDES *et al.*, 2016) também é um bom exemplo, neste, é proposto uma integração entre o IDS Snort e o sistema de *firewall* do sistema MikroTik RouteOS (MIKROTIK, 2016) com a finalidade de automatizar o processo de reação à ataques. IDSs baseados em assinatura possuem alta precisão, raramente apresentando alarmes para fluxos normais, porém, não reconhecem fluxos novos, não presentes na sua base de dados. Além disso, a inspeção de pacotes é difícil e até mesmo impossível de ser realizada em redes com taxas com múltiplos Gigabits por segundo (LAI *et al.*, 2004; GAO; ZHANG; LU, 2006).

Sistemas de detecção de intrusão baseados em anomalia por sua vez, comparam dados recebidos com um "modelo de normalidade" que descreve o comportamento normal da rede. Alterações significativas desse modelo são consideradas como anomalias. Exemplos de criação de comportamentos podem ser redes neurais, técnicas de análise de estatísticas e teoria das probabilidades. A principal vantagem desse tipo de detecção e o fato de também detectar fluxos não conhecidos anteriormente (OWEZARSKI; MAZEL; LABIT, 2010). No entanto, podem existir casos em que fluxos podem ser diferentes da normalidade esperada mas não necessariamente serem maliciosos resultando em alarmes falsos positivos.

Um IDS deve ser capaz de lidar com o número crescente do tráfego e ataques na rede. No entanto, alternativas baseadas nas análise de carga útil possuem eficácia em redes entre 100Mbps e 200Mbps (LAI *et al.*, 2004; GAO; ZHANG; LU, 2006) podendo chegar a 1Gbps quando hardware dedicado é empregado (VASILIADIS *et al.*, 2008). Sistemas como Bro (PAX-

SON, 1998) e Snort (ROESCH, 1999) apresentam alto consumo de recursos quando confrontado com a enorme quantidade de dados de alta taxa de transferência encontrados atualmente (DREGER *et al.*, 2004). Além disso, protocolos criptografados podem representar um desafio a mais para sistemas de carga útil. Para redes de alta taxas de transmissão, alternativas à inspeção de pacotes são muito importantes. Uma dessas alternativas e que tem atraído pesquisadores é a detecção de intrusão de anomalias baseada em fluxo.

Com esta abordagem, os padrões de comunicação dentro da rede são analisados, em vez de o conteúdo dos pacotes individuais. Hoje em dia os sistemas de medição especiais são capazes de fornecer, para cada par de endereços IP e números de porta, informações agregadas, como a troca de dados em tempo começou, o tempo parou, a quantidade de bytes transferidos e o número de pacotes enviados. Essas informações podem então ser exportadas para outros sistemas analisarem, para serem usados então para detectar intrusões (SPEROTTO *et al.*, 2010).

Considerando essa inflexibilidade sobre os equipamentos atuais, os interesses sobre abstrair funções de rede de *switches* dedicados para aplicações SDN vem aumentando. Sendo assim, as políticas de segurança podem ser instaladas pelo controlador como regras nas tabelas de fluxo (KIM; FEAMSTER, 2013), em vez de configurações manuais e independentes. Com isso, o *switch* provê apenas a função de filtro de acordo com a regra na tabela de fluxo, não influenciando significativamente no desempenho da rede. Além disso, SDN tem recursos naturais de estatísticas que são úteis para a análise de detecção de intrusão, de modo que o controlador obtém mais visibilidade sobre o tráfego da rede. Portanto, SDN parece fornecer uma arquitetura mais adequada para IPS.

3 TRABALHOS RELACIONADOS

Em estudos recentes existem algumas propostas para mecanismos de detecção e/ou prevenção de intrusão em SDN. Isto se deve basicamente por possuir um controle centralizado e uma visão global da rede, o que as torna eficientes na detecção e reação a intrusos maliciosos. A programabilidade do protocolo OpenFlow permite um gerenciamento mais dinâmico dos fluxos nos comutadores da rede. Esta característica permite o seu uso na área de segurança de redes e vem sendo abordada em diferentes trabalhos que são abordados nesta seção.

3.1 Soluções de IDS

O OpenSAFE, ou *Open Security Auditing and Flow Examination*, abordado por Ballard, Rae, Akella (2010), é uma proposta de solução para direcionamento de tráfego à altas taxas de transmissão para propósitos de monitoramento. OpenSAFE pode tratar diversas entradas de rede e gerir o tráfego de tal forma que este pode ser usado por diversos serviços enquanto filtra pacotes na linha. OpenSAFE possui três componentes importantes um conjunto de abstrações de *design* para discutir sobre o fluxo de tráfego na rede; uma linguagem de políticas para facilmente especificar e gerenciar rotas chamada ALARMS (A Language for Arbitrary Route Management for Security); e um componente OpenFlow que implementa a política.

OpenNetMon (ADRICHEM; DOERR; KUIPERS, 2014) é outra abordagem para aplicação de monitoramento de rede na plataforma OpenFlow. Este trabalho implementa um monitor de fluxo para entregar uma entrada refinada para a engenharia de tráfego. Beneficiado das interfaces OpenFlow, que permitem a consulta de estatísticas a partir do controlador, os autores propuseram uma maneira precisa de medir o *throughput* por fluxo, atraso e perdas de pacotes.

Trabalhos como OpenSAFE (BALLARD; RAE; AKELLA, 2010) e OpenNetMon (ADRICHEM; DOERR; KUIPERS, 2014) propõem um serviço de monitoramento da rede para eficientemente coletar estatísticas e detectar atividades maliciosas. No entanto, essas obras não vão além do estágio de detecção e não são capazes de fornecer uma análise mais aprofundada e contramedidas para ataques. A natureza de "detectar" e "alertar" das soluções, exige interação humana para inspecionar os alertas e tomar ações manualmente, não podendo assim, responder à ataques de forma rápida.

No trabalho de Shin e Gu (2012) foi proposto o CloudWatcher para resolver o problema de detecção em redes *cloud* grandes e dinâmicas. Um *framework* para manipular fluxos de rede para nós de segurança onde dispositivos de redes pré instalados possam inspecionar os pacotes,

garantindo assim, que todos os pacotes sejam inspecionados. Basicamente o CloudWatcher pode ser realizado como um aplicativo ligado ao controlador de rede, e possui três componentes principais: dispositivo gerenciador de políticas e gestão das informações de dispositivos de segurança, um gerador de regras de roteamento para criar regras para cada fluxo, e um aplicador da regra de fluxo ao comutador. Esta abordagem também permite a implantação de serviços através de *scripts*. No entanto, também não discute contramedidas para atividades maliciosas, mas apenas fornece os serviços de monitoramento.

Zhang (2013) , aborda um método de contagem de fluxo adaptivo para detecção de anomalias, que provê um eficiente mecanismo para detecção de anomalias a um baixo custo. Em sua metodologia, uma abordagem dinâmica é obtida através da atualização de regras para reunir as estatísticas e detectar anomalias de acordo com a contagem de tráfego na rede. Os fluxo são agregados e predição linear é utilizada para prever o valor da próxima contagem de fluxo. Desta forma, elimina-se a necessidade de monitorar cada pacote recebido, diminuindo a sobrecarga de monitoramento do controlador, porém, este trabalho também não provê ações de contramedida para proteger dos ataques detectados.

Braga, Mota e Passito (2010) presentam uma implementação leve, baseada em fluxo para detecção de ataques DDoS. Este método consiste em monitorar *switches* de uma rede durante intervalos predeterminados de tempo. Durante esses intervalos, são extraídas características de interesse das tabelas de fluxo de todos os *switches*. Cada amostra é então enviada para um módulo classificador que vai indicar, através de algoritmo utilizando técnica de mapas auto-organizáveis (*Self Organized Maps* (SOM)) (KOHONEN; SCHROEDER; HUANG, 2001), se a informação corresponde ao tráfego normal ou à um ataque. Este trabalho é mais leve comparado aos outros que podem exigir processamento pesado, a fim de extrair a informação característica necessária para a análise de tráfego. No entanto, este documento fornece ênfase apenas a ataques DDoS e além disso, não fornece contramedida correspondente ao ataque.

Jankowsky e Amanowicz (2015) também abordam um conceito de classificação de fluxo, com base em informações do cabeçalho da camada de transporte, utilizando redes neurais artificiais. Neste modelo, um *testbed* é utilizado para gerar classes de fluxo benignas e maliciosas. Esse fluxo é amostrado e armazenado na memória do controlador *OpenDaylight*. Paralelamente, um cliente coleta estatísticas de fluxo, as armazena e coleta as informações necessárias para posterior classificação, que é realizada utilizando mapas auto-organizáveis Kohonem. Essa abordagem é interessante por prover a detecção de diferentes classes de ataques a um baixo

overhead da rede, porém possui desvantagens no que diz respeito ao desempenho devido a comparação de todos os pacotes recebidos e, assim como a proposta de assim como a de Braga (2010), as redes neurais requerem um treinamento prévio com conjuntos de dados artificiais, o que é uma limitação importante na área de IDS. Além disso, essa proposta também não oferece contramedidas correspondentes aos ataques.

3.2 Soluções de IPS

O IPSFlow (NAGAHAMA *et al.*, 2012) propõe um mecanismo de bloqueio automático de tráfego malicioso utilizando o protocolo OpenFlow. A aplicação atua sobre o controlador para gerenciar e armazenar as regras que definem o encaminhamento dos fluxos na rede baseadas nas definições de segurança. Ao receber um pacote encaminhado pelo *switch*, o controlador consulta o aplicativo IPSFlow para verificar a existência de regras para a captura e análise do tráfego recebido. Caso esteja marcado para ser analisado, pode ser enviado para o destinatário e uma cópia enviada para análise em um IDS externo. Caso o IDS conclua que se trata de um fluxo malicioso, o tráfego passa a ser bloqueado no *switch*. Nesta abordagem o tráfego é duplicado para análise no IDS, gerando fluxos novos na rede. Além disso, os fluxos são analisados de maneira seletiva, havendo grande possibilidade de não inspecionar fluxos maliciosos durante a seleção, já que em um ataque DoS todos os campos são similares aos benignos.

Avant-Guard (SHIN *et al.*, 2013) é apresentado como uma extensão SDN, uma implementação em dois módulos: de migração de conexão e de disparo de atuação. Este trabalho é eficiente para filtrar conexões TCP incompletas, onde apenas requisições de fluxo que completam o *handshake* vão para o plano de controle. Conexões TCP são mantidas pelo módulo de migração de conexão para evitar ameaças de saturação TCP (*SYN Flooding*). O módulo de disparo de atuação permite ao plano de dados informar o *status* da rede ativar uma regra de fluxo específica baseadas em condições pré-definidas. Essa pesquisa melhorou a robustez do sistema SDN e fornece recursos adicionais ao plano de dados baixando assim o *overhead* da rede. Este trabalho no entanto é eficiente para casos de ataques de saturação, não absorvendo ataques com *handshake* completo.

Xing *et al.* (2013) propôs um trabalho chamado SnortFlow, que consiste em um IPS em ambiente de nuvem baseado no analisador de tráfego Snort (ROESCH, 1999). O analisador Snort é instalado no domínio de gerência do *hypervisor* XEN, que por sua vez é conectado ao *switch* ligado às máquinas virtuais para inspecionar o tráfego entre elas. Esse trabalho focou

basicamente na análise de desempenho, não detalhando características do tráfego e a análise que estava sendo realizada. Além disso, o Snort só consegue analisar o tráfego entre as máquinas, para uma visão global da rede seria necessário alguma forma de sincronização com o controlador.

O NICE proposto por Chung *et al.* (2013), é uma solução IDS/IPS para SDN que implementa um gráfico de ataques para gerar dinamicamente contramedidas adequadas em ambientes na nuvem. Este trabalho utiliza a teoria dos grafos para gerar um gráfico de vulnerabilidade e escolher uma solução otimizada na decisão da contramedida. Este modelo porém, é lento na geração do gráfico de ataque para a topologia não sendo prático em uma rede dinâmica.

Lopez *et al.* (2014) propõem o sistema BroFlow, um sistema IPS que utiliza a ferramenta de análise de tráfego Bro (SOMMER; PAXSON, 2010) para inspecionar os pacotes. Esta ferramenta possui sensores distribuídos em pontos estratégicos da rede e emitem alertas quando uma anomalia é detectada. A informação é enviada à um controlador OpenFlow que aciona uma contramedida para bloquear o ataque de maneira global. Este sistema no entanto é baseado em assinatura, não sendo eficiente em redes de altas taxas de transmissão, além disso, possui um problema de otimização da localização dos sensores na rede.

Wang, He e Su (2015) proveem em seu trabalho o suporte de funcionalidades mais complexas ao comutador OpenFlow através de *middleboxes* (CARPENTER; BRIM, 2002). Cada *switch* detecta e previne atividades maliciosas através do *middlebox* local e envia alertas para controlador. O controlador por sua vez possui somente a responsabilidade de prover a atualização dos *middleboxes*. Esta abordagem é interessante pois reduz a computação e a comunicação no controlador centralizado porém há a necessidade de equipamentos de rede mais robustos para os dispositivos de rede.

Quadro 2 – Comparativo entre os trabalhos estudados

Trabalho	**IDS**	**IPS**	**Tipo Ataque**	**Fonte de Coleta**
Ballard, Rae, Akella (2010)	Sim	Não	N/A	Tráfego
Adrichem, Doerr, Kuipers (2014)	Sim	Não	N/A	Contadores
Shin, Gu (2012)	Sim	Não	N/A	Tráfego
Zhang (2013)	Sim	Não	N/A	Contadores
Braga, Mota, Passito (2010)	Sim	Não	DDoS	Contadores
Jankowsky, Amanowicz (2015)	Sim	Não	*scan*,DoS	Tráfego
Nagahama *et al.* (2012)	Sim	Sim	N/A	Tráfego
Shin *et al.* (2013)	Não	Sim	DoS	Tráfego
Xing *et al.* (2013)	Não	Sim	N/A	Tráfego
Lopez *et al.* (2014)	Sim	Sim	DoS	Tráfego
Wang *et al.* (2015)	Sim	Sim	N/A	Tráfego
Este trabalho	Sim	Sim	*scan*	Contadores

Fonte: Elaborado pelo autor.

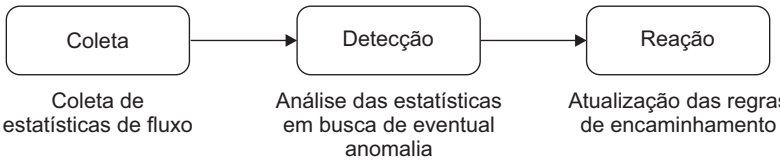
3.3 Objetivos

Considerando os trabalhos estudados na literatura, os quais são sumarizados no Quadro 2, percebe-se que o uso de OpenFlow para a implementação de segurança mostrou-se bastante promissor. Porém, observou-se que o foco dado pela maioria destes não implementa recursos de contramedida para ataques. Mesmo aqueles que apresentam uma proposta de IPS, não contemplam uma solução de prevenção para ataques de varredura de porta. Neste sentido, motivado pelas limitações dos trabalhos discutidos anteriormente, este trabalho de conclusão de curso propõe uma alternativa para completar as metodologias de detecção e prevenção de intrusão já existentes, através do desenvolvimento de um IPS baseado em anomalia, fazendo uso do protocolo OpenFlow e utilizando como fonte de informação, contadores das tabelas de fluxo presentes nos *switches* SDN. O desenvolvimento deste trabalho, bem como sua arquitetura serão apresentados no Capítulo 4 deste trabalho.

4 SOLUÇÃO PROPOSTA

Esta solução proposta foi desenvolvida considerando um ciclo de vida de três etapas (Figura 6): A primeira refere-se à coleta, por parte do controlador, de informações presentes nas tabelas de fluxo dos *switches* OpenFlow. A segunda etapa, de detecção, tem por finalidade a análise das informações coletadas, decidir se um fluxo é ou não malicioso e informar ao controlador para que tome as ações necessárias para prevenir o ataque. Por fim, na etapa de reação, são implementadas contramedidas para o bloqueio de ataques. Nesta etapa o controlador envia regras, definidas conforme os dados analisados, para atualização das regras de fluxo nos *switches* OpenFlow. Cada uma dessas etapas será discutida no decorrer deste capítulo.

Figura 6 – Ciclo de vida do IPS proposto



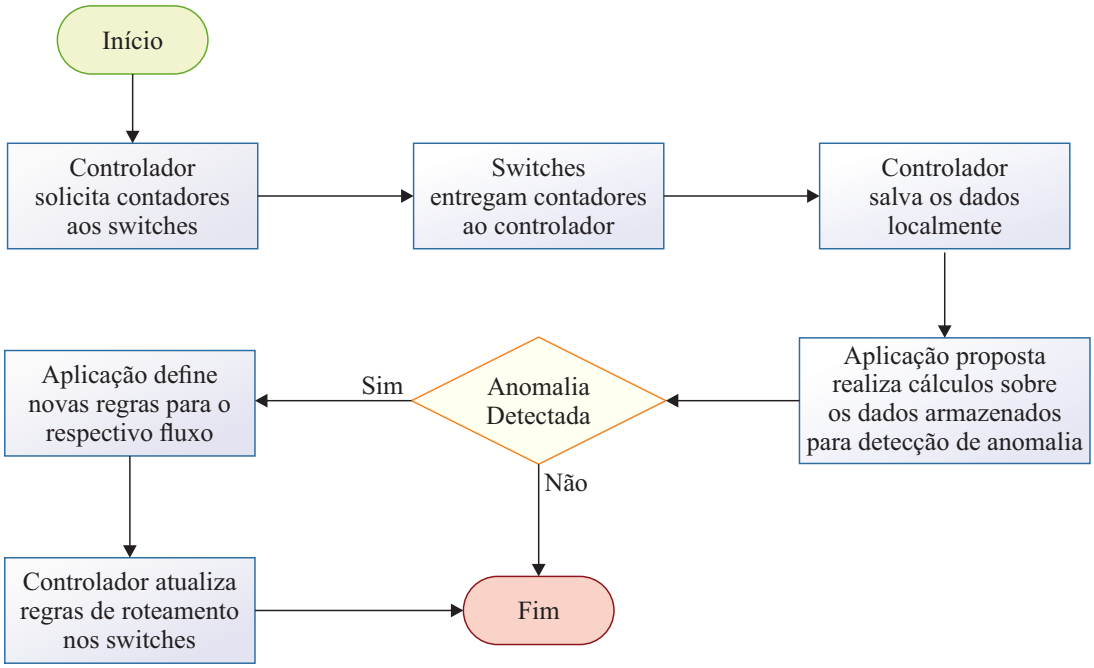
Fonte: Elaborado pelo autor.

Este capítulo está organizado da seguinte forma: na seção 4.1 é apresentado à arquitetura do sistema proposto,na seção 4.2 é abordada a forma de monitoramento e obtenção de estatísticas presentes na tabela de fluxo por parte do controlador. Na seção 4.3 diferentes algoritmos são discutidos de forma a analisar e detectar padrões de anomalia na rede. Por fim, na seção 4.4, ações de contramedida são abordadas com o objetivo de prevenir e combater atividades maliciosas na rede.

4.1 Arquitetura

Este trabalho possui uma arquitetura de IPS implementada para SDN e utiliza o protocolo OpenFlow para possibilitar a construção de um IPS distribuído. Por atuar sobre SDN, o controlador possui visão global da rede podendo, ao detectar uma ameaça, efetuar o bloqueio de um fluxo malicioso logo na sua origem. Na arquitetura proposta, o controlador gerencia e armazena as regras que definem o encaminhamento de pacotes na rede além de coletar estatísticas dos *switches* OpenFlow conforme pode ser observado na Figura 7 e apresentado nas seções seguintes.

Figura 7 – Fluxo de funcionamento do IPS proposto



Fonte: Elaborado pelo autor.

4.2 Coleta

A fase de coleta é responsável por solicitar periodicamente estatísticas de fluxo dos *switches* com suporte à OpenFlow e disponibilizar as informações coletadas para a etapa de detecção. Como já discutido na seção 2.4, a análise de fluxo pode ser realizada utilizando dados dos pacotes como endereços IP de origem e destino, ou através da análise de *logs* de dispositivos. Tradicionalmente, para análise dos dados dos pacotes, várias técnicas diferentes de monitoramento são utilizadas. Cada técnica de monitoramento requer uma instalação separada de hardware ou configuração de software, tornando isso moroso e com custo alto de implementação. No entanto, OpenFlow provê as interfaces necessárias para implementar a maioria dos métodos discutidos, sem a necessidade de grandes customizações (ADRICHEM; DOERR; KUIPERS, 2014).

A definição do intervalo de coleta das entradas de fluxo é de grande importância. Se a coleta é realizada com períodos muito grandes, haverá um atraso na detecção de ataques. Por outro lado, se o intervalo for curto demais, haverá um aumento de tráfego referente às requisições de coleta.

Utilizando as mensagens definidas pelo protocolo OpenFlow e brevemente discutidas na seção 2.1.3, a coleta de estatísticas é facilitada, passando a ser feita pelo controlador da rede. O

controlador realiza, através do protocolo OpenFlow, a coleta de contadores presentes nas tabelas de fluxo dos *switches*. Esses contadores foram definidos com o objetivo de facilitar a criação de mecanismos de QoS (OPEN NETWORKING FOUNDATION, 2016) e possuem informações agrupadas por fluxo, por porta, etc. e serão a fonte de informação para este trabalho.

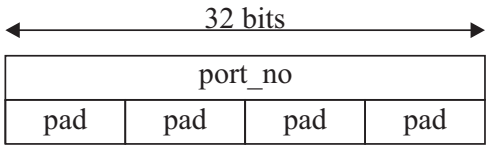
Analisando o que foi discutido na seção 2.4.2, técnicas para detecção de varredura de portas podem facilmente ser implementadas utilizando informações do cabeçalho dos pacotes e que também estão presentes nas tabelas de fluxo, como *host* de origem e destino e portas de destino. A fim de se obter estatísticas referentes em nível de porta, OpenFlow fornece alguns métodos para obtenção de informações detalhadas de uma porta específica, ou de portas em geral em um *switch* específico.

Segundo a especificação OpenFlow (OPEN NETWORKING FOUNDATION, 2014) se o controlador desejar obter estatísticas de uma porta OpenFlow, ele deve enviar uma mensagem *multipart* (ofp_multipart_request) do tipo OFPMP_PORT_STATS para o *switch*.

Mensagens *multipart* são usadas para codificar pedidos ou respostas que podem carregar grande quantidade de informações que nem sempre se encaixam em uma única mensagem OpenFlow, que é limitada a 64KB. O pedido ou resposta é codificada como uma sequência de mensagens de várias partes de determinado tipo em uma mesma conexão, e remontadas pelo receptor. Cada sequência de mensagens *multipart*, carrega apenas uma solicitação ou resposta. Mensagens *multipart* são comummente utilizadas para solicitações de estatísticas ou informações do *switch* (OPEN NETWORKING FOUNDATION, 2014).

O corpo (*payload*) da mensagem de requisição deve ser preenchido segundo o formato exibido pela Figura 8 a seguir:

Figura 8 – Estrutura do corpo da mensagem de solicitação de estatísticas



Fonte: Elaborado pelo autor a partir de informações da especificação OpenFlow.

O campo *port_no* deve ser preenchido com o número da porta cujas estatísticas devem ser buscadas pelo *switch*. Se o controlador quiser obter estatísticas de todos as portas do sistema, então este valor deve ser definido como OFPP_ANY (segundo a especificação OpenFlow, todos os *bits* iguais à um). Já os campos *pad* não necessitam de informação, servem apenas para

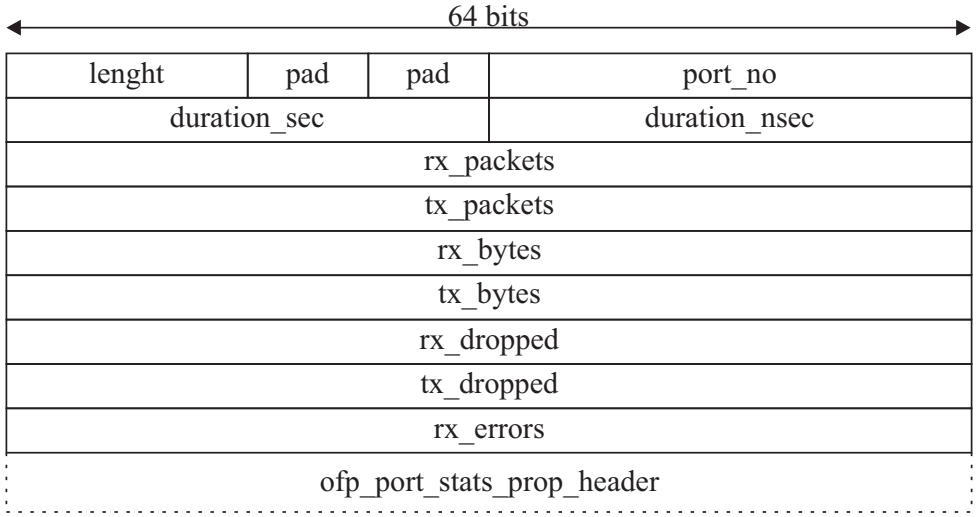
completar o pacote.

Após enviar e mensagem de requisição, o controlador deve esperar por uma resposta da mensagem pelo *switch*. Se a resposta exceder o limite máximo da mensagem OpenFlow (64KB) o *switch* então, irá enviar uma sequência de múltiplas mensagens com a *flag* OFPMP_REPLY_MORE no cabeçalho da mensagem *multipart* habilitada.

Se o controlador recebe a mensagem de resposta com esta *flag* habilitada, ele deve armazenar a sequência de mensagens até que a última mensagem da sequência seja recebida. Como o *switch* envia a sequência de mensagens de várias partes com o mesmo ID de transação (xid), o controlador deve mapear todas as mensagens na sequência e deve decodificar a mensagem para obter as informações estatísticas reais da mensagem.

Quando o *switch* recebe a mensagem OFPMP_PORT_STATS, ele primeiramente obtém a entrada de porta correspondente com base no campo port_no. Uma vez obtidas as entradas de porta correspondentes, o mesmo constrói uma mensagem de resposta (Figura 9) e a envia de volta para o controlador. Esta mensagem pode ser enviada através de um ou mais pacotes, dependendo do tamanho da mesma.

Figura 9 – Corpo da resposta à requisição OFPMP_PORT_STATS.



Fonte: Elaborado pelo autor a partir de informações da especificação OpenFlow.

Os campos *duration_sec* e *duration_nsec* indicam o tempo decorrido do fluxo desde sua entrada no *pipeline* OpenFlow. A duração total em nano segundos pode ser obtido pelo cálculo $duration\_sec * 10^9 + duration\_nsec$. Os campos *rx_packets*, *tx_packets*, *rx_bytes* e *tx_bytes* são contadores de medição de pacotes que passam pela porta. O campo *ofp_port_stats_prop_header* é uma lista de zero ou mais propriedades específicas para determinado tipo de comunicação

(*Ethernet*, Óptica ou Experimental) e inclui informações como número de colisões, potência de transmissão entre outros que não serão abordados neste trabalho.

O controlador a ser utilizado para o desenvolvimento desta solução é o OpenDayLight (OPENDAYLIGHT, 2016) que atualmente está na versão "Boron". Esta escolha deve-se ao fato deste ser um projeto desenvolvido em Java, cuja linguagem o autor deste possui experiência, além de possuir uma grande comunidade de desenvolvimento e ótima documentação. A coleta de estatísticas de porta no OpenDayLight pode ser obtida de forma muito simples através da API REST em um *browser*. O resultado será um arquivo em formato *JavaScript Object Notation* (JSON) ou *eXtensible Markup Language* (XML) como o exemplo abaixo:

Código 1 – Parte do corpo da resposta ofp-port-stats no formato XML

```
<node-connector xmlns="urn:opendaylight:inventory">
    <id>openflow:1:1</id>
     ...
    <flow-capable-node-connector-statistics
        xmlns="urn:opendaylight:port:statistics">
        <receive-over-run-error>0</receive-over-run-error>
        <duration>
            <second>1627</second>
            <nanosecond>323000</nanosecond>
        </duration>
        <receive-crc-error>0</receive-crc-error>
        <receive-errors>0</receive-errors>
        <transmit-errors>0</transmit-errors>
        <bytes>
            <transmitted>19152</transmitted>
            <received>39715</received>
        </bytes>
        <collision-count>0</collision-count>
        <transmit-drops>0</transmit-drops>
        <receive-frame-error>0</receive-frame-error>
        <packets>
            <transmitted>304</transmitted>
            <received>612</received>
        </packets>
        <receive-drops>0</receive-drops>
    </flow-capable-node-connector-statistics>
</node-connector>
```

Fonte: OpenDayLight (2016).

4.3 Detecção

Conforme discutido na seção 2.4.2, há três categorias de *scans*: *scan* horizontal, vertical e *block* e os algoritmos para detectar cada uma delas será discutido nesta seção.

4.3.1 Detecção de varredura horizontal

A detecção de varredura horizontal de portas é desenvolvido pela premissa de que vários fluxos, para uma mesma porte em diferentes *hosts* alvo são originadas de um mesmo *host* de origem. Nesse tipo de intrusão a aplicação proposta irá analisar a origem e destino das conexões. Endereços IP de origem que estejam tentando conectar-se à um vasto número de *hosts* em uma mesma porta podem ser consideradas anomalias. Além disso, esse tipo de conexão em geral possui curta duração (somente a duração do *handshake*, se houver), o que também pode ser um indício de anomalia.

4.3.2 Detecção de varredura vertical

Na detecção de varredura vertical, tem-se a premissa de que um *host* de origem realiza a varredura de várias portas em um mesmo *host* de destino. Neste caso, *hosts* de origem que possuam inúmeros fluxos para diferentes portas, podem ser reconhecidos pela aplicação como uma anomalia. Assim como na análise de varredura horizontal, são considerados a duração das conexões .

4.3.3 Detecção de varredura *block*

Nesse tipo de varredura, ambas as premissas já citadas devem ser consideradas. Neste, a aplicação proposta realiza a verificação de *hosts* de origem que tentam estabelecer conexões em diferentes portas e/ou para diferentes *hosts* de destino.

Para as três categorias citadas, a aplicação irá também analisar tentativas de *half handshake*. Um número elevado de tentativas de conexão de uma mesma origem que não estabeleçam conexão, podem ser consideradas como anomalia. Varreduras ARP também poderão ser detectados através da análise da quantidade de *hosts* de destino para o qual o *host* de origem enviou pacotes ARP pois não é comum um *host* de origem enviar pacotes ARP para um número elevado de *hosts*.O recebimento de segmentos FIN para conexões inexistentes à um elevado número de *hosts* ou portas também poderá ser considerado pela aplicação proposta como uma anomalia.

A medida que o aplicativo de detecção encontre anomalias, o mesmo poderá criar regras e enviá-las para o controlador efetuar a atualização das regras nos *switches* OpenFlow, ou então enviar os resultados para o controlador para que este então decida a ação a ser tomada. Se nenhuma anomalia for detectada, nenhum procedimento se faz necessário, neste caso, o aplicativo irá apenas registrar a detecção em arquivo.

Como pode ser analisado, diversos fatores devem ser considerados para que se possa reduzir ao máximo o número de falsos positivos, assim como falsos negativos. A realização de testes através do *software* Mininet (MININET, 2016) poderá auxiliar na estabilização de parâmetros considerados normais em um rede, obtendo-se assim uma limiar mais próxima da ideal para a normalidade da rede.

4.4 Ações de reação

Recebidas as informações sobre os dados analisados, o controlador pode decidir sobre as ações a serem tomadas. Se não houverem indícios de anomalias, o controlador irá desconsiderar a resposta do algoritmo de análise, caso contrário, uma ação de prevenção deve ser tomada. Neste trabalho a ação de prevenção é realizada através da atualização das regras de encaminhamento em todos os *switches* OpenFlow.

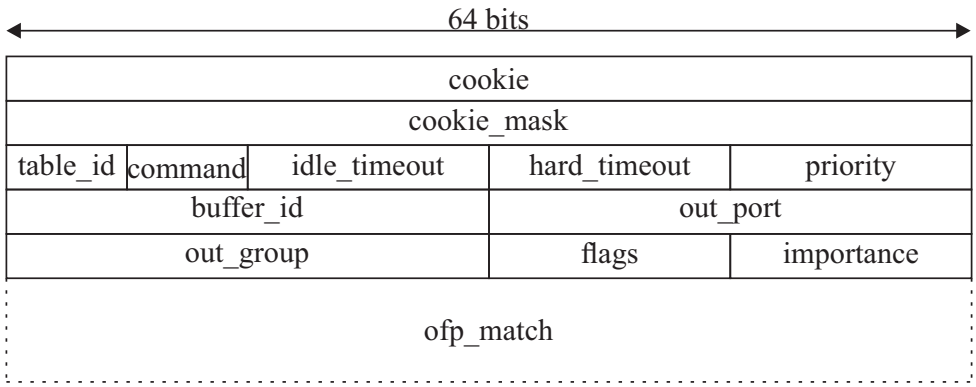
Mensagens para alteração da tabela de encaminhamento podem ter os seguintes tipos:

- **OFPFC_ADD** - Adiciona um novo fluxo;
- **OFPFC_MODIFY** - Modifica entradas de fluxo existentes;
- **OFPFC_MODIFY_STRICT** - Modifica entradas de fluxo existentes validando estritamente seus campos, ou seja, apenas uma entrada da tabela é modificada;
- **OFPFC_DELETE** - Remove entradas de fluxo existente; e
- **OFPFC_DELETE_STRICT** - Remove entradas de fluxo validando estritamente seus campos, ou seja, apenas uma entrada da tabela é removida.

Entradas de fluxos podem ser removidos da tabela de encaminhamento por três maneiras: através da requisição do controlador, pelo mecanismo de *timeout*, ou pelo mecanismo de despejo opcional. O mecanismo de *timeout* é executado pelo *switch* independentemente do controlador e é baseado na configuração e no estado das entradas de fluxo.

Para alterar ou remover alguma entrada da tabela de fluxo, deve ser enviada uma mensagem do tipo OFPT_FLOW_MOD que é definida conforme ilustra a Figura 10.

Figura 10 – Corpo da mensagem de alteração de entradas na tabela de fluxo.



Fonte: Elaborado pelo autor a partir de informações da especificação OpenFlow.

O campo *table_id* refere-se à tabela a ser atualizada, que pode ser por fluxo, porta, como já citado. O campo *command* refere-se á ação a ser executada (adição, alteração e remoção), campos *timeout* indicam o tempo máximo de espera por um fluxo antes do descarte, *priority* armazena a prioridade da entrada na tabela de fluxo. O campo *ofp_match* indicam as regras de encaminhamento de fluxos. Os demais campos não serão abordados neste mas seu estudo pode ser realizado através especificação OpenFlow (OPEN NETWORKING FOUNDATION, 2014). Alterando a regras de encaminhamento através destas mensagens, além de reagir ao ataque em questão, o sistema também estará protegido contra novos fluxos de mesma origem.

5 CONSIDERAÇÕES FINAIS

O objetivo geral desse trabalho é apresentar uma arquitetura leve que permita, de maneira eficaz, coletar estatísticas de fluxo em redes SDN, verificar a ocorrência de fluxos de varredura de porta e então, prover a segurança da rede através da atualização das regras de encaminhamento. Nesse sentido, foi realizada uma revisão da área de redes definidas por software visando suportar a proposição do trabalho.

Na base da arquitetura encontra-se um modelo de análise de estatísticas da tabela de fluxo. Esse modelo procura simplificar a coleta de informações visto que, ao invés de inspecionar todos os pacotes, ocasionando atraso na comutação da rede, considera-se apenas informações presentes na tabela de fluxo dos *switches* OpenFlow. Tal abordagem possibilita um ganho de desempenho se comparado com abordagens de coleta de fluxo já que as informações são alimentadas pelo próprio *switch*.

Desta forma, salienta-se que, em função da simplificação, a precisão do processo também é minimizada. Por outro lado, esta pode ser obtida em um estágio posterior, ou seja, a medida que o sistema encontre fluxos duvidosos, pode encaminhá-los para um IDS mais preciso (e mais custoso) como SNORT (MCKEOWN *et al.*, 2008). Outro ponto a ser considerado é o fato de existirem, na literatura, poucos trabalhos relacionados à prevenção de ataques de varredura de porta para SDN.

Buscando o aperfeiçoamento do presente trabalho, sugere-se o desenvolvimento da comunicação entre o controlador e *switches* para a troca das mensagens de requisição de estatísticas e atualização das tabelas. Além disso, faz-se necessária a implementação do algoritmo de detecção de anomalias apresentado no presente instrumento de estudo.

Destaca-se também a necessidade de realizar testes de verificação, sendo que para tais, sugere-se inicialmente a utilização de *softwares* simuladores como Mininet (MININET, 2016), uma vez que, por tal meio, não se faz necessário dispositivos físicos. Posteriormente, sugere-se a realização de testes em ambientes de experimentação (*testbeds*), como por exemplo FIBRE (SALMITO *et al.*, 2014), por permitirem um ambiente de rede mais próximo da realidade.

Na tentativa de maximizar os resultados, faz-se também necessária a análise de falsos positivos e negativos, bem como a verificação dos resultados. No entanto, o desenvolvimento, análises e resultados serão apresentados, na segunda parte deste trabalho de conclusão de curso, o qual será desenvolvido no primeiro semestre de 2017.

REFERÊNCIAS

ADRICHEM, N. L. M. van; DOERR, C.; KUIPERS, F. A. OpenNetMon: Network monitoring in openflow software-defined networks. In: *2014 IEEE Network Operations and Management Symposium*. Cracóvia, POL: IEEE Computer Society, 2014. p. 1–8.

AURRECOECHEA, C.; CAMPBELL, A. T.; HAUW, L. A survey of QoS architectures. *Multimedia Systems*, v. 6, n. 3, p. 138–151, 1998.

AXELSSON, S. *Intrusion Detection Systems: A Survey and Taxonomy*. 2000.

BALLARD, J. R.; RAE, I.; AKELLA, A. Extensible and scalable network monitoring using OpenSAFE. In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. Berkeley, CA, USA: USENIX Association, 2010. p. 8–8.

BRAGA, R.; MOTA, E.; PASSITO, A. Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*. Washington, DC, USA: IEEE Computer Society, 2010. p. 408–415.

CARPENTER, B.; BRIM, S. *Middleboxes: Taxonomy and Issues*. 2002. Disponível em: <http://www.rfc-editor.org/rfc/rfc3234.txt>. Acesso em: 21 set. 2016.

CASE, J. D. *et al. Simple Network Management Protocol (SNMP)*. 1990. Disponível em: <http://www.rfc-editor.org/rfc/rfc1157.txt>. Acesso em: 23 out. 2016.

CERT.BR. *Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil*. out. 2016. 2016. Disponível em: <http://www.cert.br/>. Acesso em: 02 out. 2016.

CHOWDHURY, N. M. M. K.; BOUTABA, R. Network virtualization: State of the art and research challenges. *Comm. Mag.*, IEEE Press, Piscataway, NJ, USA, v. 47, n. 7, p. 20–26, jul. 2009.

CHRISTOPHER, R. Port scanning techniques and the defense against them. *SANS Institute InfoSec Reading Room*, out. 2001.

CHUN, B. *et al.* Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 33, n. 3, p. 3–12, jul. 2003.

CHUNG, C. J. *et al.* Nice: Network intrusion detection and countermeasure selection in virtual network systems. *IEEE Transactions on Dependable and Secure Computing*, IEEE Computer Society, v. 10, n. 4, p. 198–211, jul. 2013.

CLARK, D. *et al.* New arch: Future generation internet architecture. Rome, NY,USA, ago. 2004.

COMER, D. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

COSTA, L. R. OpenFlow e o paradigma de redes definidas por software. Universidade de Brasília, Brasília, DF, mar. 2013.

DREGER, H. *et al.* Operational experiences with high-volume network intrusion detection. In: *Proceedings of the 11th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2004. p. 2–11.

DROMS, R. *Dynamic Host Configuration Protocol*. 1997. Disponível em: <http://www.rfc-editor.org/rfc/rfc2131.txt>. Acesso em: 25 out. 2016.

ERICKSON, D. The beacon openflow controller. In: *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. p. 13–18.

FAGUNDES, B. *et al.* Snortik - uma integração do IDS SNORT e o sistema de firewall do MikroTik RouterOS. In: *Anais da 14a Escola Regional de Redes de Computadores*. Porto Alegre, RS: [s.n.], 2016.

FERNANDES, N. C. *et al.* Virtual networks: Isolation, performance, and trends. *Annals of Telecommunications*, Springer-Verlag, New York, NY, USA, v. 16, n. 5-6, p. 339–355, Jun 2011.

FLOODLIGHT, P. *Project Floodlight*. Set 2016. 2016. Disponível em: <http://www.projectfloodlight.org/floodlight/>. Acesso em: 30 out. 2016.

FREIER, A.; KARLTON, P.; KOCHER, P. *The Secure Sockets Layer (SSL) Protocol Version 3.0*. 2011. Disponível em: <http://www.rfc-editor.org/rfc/rfc6101.txt>. Acesso em: 23 out. 2016.

GAO, M.; ZHANG, K.; LU, J. Efficient packet matching for gigabit network intrusion detection using tcams. In: *Proceedings of 20th International Conferece on Advanced Information Networking and Applications*. Washington, DC, USA: IEEE Computer Society, 2006. p. 249–254.

GUDE, N. *et al.* Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 3, p. 105–110, jul. 2008.

GUEDES. Redes definidas por software: uma abordagem sistêmica para o desenvolvimento das pesquisas em redes de computadores. ACM, New York, NY, USA, Abr 2012.

HANDIGOL, N. *et al.* Reproducible network experiments using container-based emulation. In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. New York, NY, USA: ACM, 2012. p. 253–264.

HAWILO, H. *et al.* NFV: state of the art, challenges, and implementation in next generation mobile networks (vepc). *IEEE Network*, v. 28, n. 6, p. 18–26, nov. 2014.

HOFFMAN, P.; SULLIVAN, A.; FUJIWARA, K. *DNS Terminology*. 2015. Disponível em: <https://www.rfc-editor.org/rfc/rfc7719.txt>. Acesso em: 21 out. 2016.

HOUAISS, A.; VILLAR, M.; FRANCISCO. *Dicionário Houaiss da língua portuguesa*. Rio de Janeiro: Editora Objetiva, 2001.

JANKOWSKY I. D.; AMANOWICZ, M. Intrusion detection in software defined networks with self-organized maps. In: *Journal os Telecommunications and Information Technology*. Warsaw, POL: National Institute of Telecommunications, 2015.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, IEEE Computer Society, v. 51, n. 2, p. 114–119, fev. 2013.

KOHONEN, T.; SCHROEDER, M. R.; HUANG, T. S. (Ed.). *Self-Organizing Maps*. 3rd. ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2001.

KONTESIDOU, G. OpenFlow virtual networking: A flowbased network virtualization architecture. *S*, ACM, New York, NY, USA, abr. 2012.

KREUTZ, D.; RAMOS, F. M. V.; VERISSIMO, P. Towards secure and dependable software-defined networks. In: ACM. *Proceendings of the second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013. Workshop, p. 55–60.

KREUTZ, D. *et al.* Software-defined networking: A comprehensive survey. *ArXiv e-prints*, jun. 2014.

KRUEGEL, C. *Intrusion Detection and Correlation: Challenges and Solutions*. Santa Clara, CA, USA: Springer-Verlag TELOS, 2004.

LAI, H. *et al.* A parallel intrusion detection system for high-speed networks. In: ____. *Applied Cryptography and Network Security: Second International Conference, ACNS 2004, Yellow Mountain, China, June 8-11, 2004. Proceedings*. Berlin, Heidelberg, GER: Springer Berlin Heidelberg, 2004. p. 439–451.

LOPEZ, M. A. *et al.* Broflow: Um sistema eficiente de detecção e prevenção de intrusão em redes definidas por software. Brasília, DF, jul. 2014.

LYNCH, D. M. Securing against insider attacks. *EDPACS*, v. 34, n. 1, p. 10–20, 2006.

LYON, G. *Nmap.org*. out 2016. 2016. Disponível em: <https://nmap.org>. Acesso em: 30 out. 2016.

MATTOS, D. M. F.; DUARTE, O. C. M. B. Qflow: Um sistema com garantia de isolamento e oferta de qualidade de serviço para redes virtualizadas. In: *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC'2012*. Ouro Preto, MG: [s.n.], 2012. p. 536–549.

MCCAULEY, M. *POX*. Out 2016. 2016. Disponível em: <http://www.noxrepo.org>. Acesso em: 30 out. 2016.

MCKEOWN, N. *et al.* Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 38, n. 2, p. 69–74, mar. 2008.

MIKROTIK. *MikroTik*. Out 2016. 2016. Disponível em: <http://www.mikrotik.com>. Acesso em: 25 out. 2016.

MININET. *Mininet. An Instant Virtual Network on your Laptop*. Set 2016. 2016. Disponível em: <http://mininet.org>. Acesso em: 28 out. 2016.

MOY, J. *OSPF Version 2*. 1998. Disponível em: <http://www.rfc-editor.org/rfc/rfc2328.txt>. Acesso em: 22 out. 2016.

NAGAHAMA, F. Y. *et al.* IPSFlow – uma proposta de sistema de prevenção de intrusão baseado no framework openflow. *III Workshop de Pesquisa Experimental da Internet do Futuro*, Ouro Preto, MG, maio 2012.

NEGUS, C.; BRESNAHAN, C. *Linux Bible*. 9th. ed. [S.l.]: Wiley Publishing, 2015.

OPEN NETWORKING FOUNDATION. *OpenFlow Switch Specification*. 2014. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf>. Acesso em: 10 out. 2016.

OPEN NETWORKING FOUNDATION. *OpenFlow*. out. 2016. 2016. Disponível em: <https://www.opennetworking.org/sdn-resources/openflow>. Acesso em: 30 out. 2016.

OPENDAYLIGHT. *OpenDayLight, A Linux Foundation Collaborative Project*. Out 2016. 2016. Disponível em: <http://www.opendaylight.org>. Acesso em: 30 out. 2016.

OPPLIGER, R. Internet security: Firewalls and beyond. *Commun. ACM*, ACM, New York, NY, USA, v. 40, n. 5, p. 92–102, maio 1997.

OWEZARSKI, P.; MAZEL, J.; LABIT, Y. 0day anomaly detection made possible thanks to machine learning. In: *Proceedings of the 8th International Conference on Wired/Wireless Internet Communications*. Berlin, Heidelberg: Springer-Verlag, 2010. p. 327–338.

PAXSON, V. Bro: a system for detecting network intruders in real-time. computer networks. In: *Proceedings of the 7th USENIX Security Symposium*. Berkeley, CA, USA: Lawrence Berkeley National Laboratory, 1998. p. 26–29.

PETTIT, J. *et al.* Virtual switching in an era of advanced edges. In: *Proceedings of the 2nd workshop on data center-converged and virtual ethernet switching (DC-CAVES)*. Lauderdale, FL, USA: [s.n.], 2010.

PLUMMER, D. C. *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*. 1982. Disponível em: <http://www.rfc-editor.org/rfc/rfc826.txt>. Acesso em: 21 out. 2016.

POSTEL, J. *Internet Control Message Protocol*. 1981. Disponível em: <http://www.rfc-editor.org/rfc/rfc792.txt>. Acesso em: 22 out. 2016.

POSTEL, J. *Transmission Control Protocol*. 1981. Disponível em: <http://www.rfc-editor.org/rfc/rfc793.txt>. Acesso em: 24 set. 2016.

REKHTER, Y.; LI, T.; HARES, S. *A Border Gateway Protocol 4 (BGP-4)*. 2006. Disponível em: <http://www.rfc-editor.org/rfc/rfc4271.txt>. Acesso em: 23 out. 2016.

ROESCH, M. Snort - lightweight intrusion detection for networks. In: *Proceedings of the 13th USENIX Conference on System Administration*. Berkeley, CA, USA: USENIX Association, 1999. p. 229–238.

ROTHENBERG, C. E. *et al.* Openflow e redes definidas por software: um novo paradigma de controle e inovação em redes de pacotes. In: *Caderno CPqD Tecnologia*. [S.l.]: Centro de Pesquisa e Desenvolvimento em Telecomunicações, 2010. v. 7, p. 1–6.

RYU SDN FRAMEWORK COMMUNITY. *Ryu Network Operation System*. Out 2016. 2016. Disponível em: <https://osrg.github.io/ryu/>. Acesso em: 30 out. 2016.

SALMITO, T. *et al.* FIBRE - An International Testbed for Future Internet Experimentation. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2014*. Florianópolis, SC: Sociedade Brasileira de Computação, 2014. p. 969.

SAYEED, M. A.; SAXENA, S. Intrusion detection system based on software defined network firewall. In: *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*. Dehradun, Uttarakhand, IND: IEEE Computer Society, 2015. p. 379–382.

SEEBER, S.; STIEMERT, L.; RODOSEK, G. D. Towards an sdn-enabled ids environment. In: *2015 IEEE Conference on Communications and Network Security (CNS)*. Florence, ITA: IEEE Computer Society, 2015. p. 751–752.

SHERWOOD, R. *et al. FlowVisor: A Network Virtualization Layer*. 2009. Disponível em: <http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf>. Acesso em: 03 out. 2016.

SHIN, S.; GU, G. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In: *2012 20th IEEE International Conference on Network Protocols (ICNP)*. Austin, TX, USA: IEEE Computer Society, 2012. p. 1–6.

SHIN, S. *et al.* Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*. New York, NY, USA: ACM, 2013. p. 413–424.

SOARES, S. G. L. *et al.* Estudo da implementação de voip em redes sdn. dez. 2015.

SOMMER, R.; PAXSON, V. Outside the closed world: On using machine learning for network intrusion detection. In: *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2010. p. 305–316.

SPEROTTO, A. *et al.* An overview of ip flow-based intrusion detection. *Commun. Surveys Tuts.*, IEEE Press, Piscataway, NJ, USA, v. 12, n. 3, p. 343–356, jul. 2010.

SRISURESH, P.; EGEVANG, K. *Traditional IP Network Address Translator (Traditional NAT)*. 2001. Disponível em: <https://www.rfc-editor.org/rfc/rfc3022.txt>. Acesso em: 24 out. 2016.

TENNENHOUSE, D. L. *et al.* A survey of active network research. *IEEE Communications Magazine*, IEEE Computer Society, v. 35, n. 1, p. 80–86, Jan 1997.

TOOTOONCHIAN, A.; GANJALI, Y. Hyperflow: A distributed control plane for openflow. In: *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*. Berkeley, CA, USA: USENIX Association, 2010. p. 3–3.

TURNER, J. S. A proposed architecture for the geni backbone platform. In: *Proceedings of the 2006 ACM/IEEE Symposium on Architecture for Networking and Communications Systems*. New York, NY, USA: ACM, 2006.

VASILIADIS, G. *et al.* Gnort: High performance network intrusion detection using graphics processors. In: *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2008. p. 116–134.

VIVO, M. de *et al.* A review of port scanning techniques. *SIGCOMM Comput. Commun. Rev.*, ACM, New York, NY, USA, v. 29, n. 2, p. 41–48, abr 1999.

WANG, W.; HE, W.; SU, J. Network intrusion detection and prevention middlebox management in sdn. In: *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*. Nanjing, Jiangsu, CHN: IEEE Computer Society, 2015. p. 1–8.

WENDONG, W. *et al.* Autonomicity design in openflow based software defined networking. In: *2012 IEEE Globecom Workshops*. Anaheim, CA, USA: IEEE Computer Society, 2012. p. 818–823.

WU, H. *et al.* Network security for virtual machine in cloud computing. In: *2010 5th International Conference on Computer Sciences and Convergence Information Technology (ICCIT)*. Seoul, KOR: IEEE Computer Society, 2010. p. 18–21.

XIA, W. *et al.* A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, IEEE Computer Society, v. 17, n. 1, p. 27–51, jun 2015.

XING, T. *et al.* Snortflow: A openflow-based intrusion prevention system in cloud environment. In: *Proceedings of the 2013 Second GENI Research and Educational Experiment Workshop*. Washington, DC, USA: IEEE Computer Society, 2013. p. 89–92.

XIONG, Z. An sdn-based ips development framework in cloud networking environment. *IEEE Communications Magazine*, v. 51, n. 2, p. 114–119, fev 2013.

ZHANG, Y. An adaptive flow counting method for anomaly detection in sdn. In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. New York, NY, USA: ACM, 2013. p. 25–30.