**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

Cássio Giordani Tatsch

# SISTEMA DE DETECÇÃO E PREVENÇÃO DE ATAQUES PORT SCAN EM REDES OPENFLOW SDN

Santa Cruz do Sul

2017

Cássio Giordani Tatsch

# SISTEMA DE DETECÇÃO E PREVENÇÃO DE ATAQUES PORT SCAN EM REDES OPENFLOW SDN

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
2017

**AGRADECIMENTOS**

Agradeço a esta universidade, seu corpo docente, por me proporcionar não apenas o conhecimento racional, mas também a manifestação do caráter no processo de formação. Agradeço em especial ao meu orientador, Charles Neu, pela colaboração no desenvolvimento deste trabalho.

Aos meus pais Norberto e Ursula que sempre me incentivaram e me deram aporte para que pudesse investir no meu conhecimento.

Agradeço à minha namorada Juliana, que sempre esteve ao meu lado. Obrigado por ter me incentivado e pela compreensão durante este período. Te amo!

A todos que, direta ou indiretamente, contribuíram para que alcançasse meus objetivos acadêmicos, deixo registrado aqui o meu "Muito Obrigado".

*O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis.* (José de Alencar)

**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 a 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. Diversos trabalhos têm abordado o desenvolvimento de sistemas de detecção e prevenção de intrusão para SDN, baseados em técnicas de análise de fluxos recebidos. A análise de fluxo por sua vez, pode ocasionar um aumento na latência de chaveamento dos comutadores, devido a necessidade de analisar o pacote antes de encaminhá-lo. Também pode haver sobrecarga na rede, quando o pacote é duplicado para que seja analisado por um IDS em paralelo com o encaminhamento dos pacotes. Esses problemas tornam o IDS/IPS um gargalo, e é com base nesse contexto e beneficiado pelas características flexíveis de SDN, que este trabalho apresenta um novo IPS para SDN, contra ataques *port scan*, utilizando dados das tabelas de encaminhamento de *switches* OpenFlow. Isto resulta em uma arquitetura não-intrusiva e *lightweight*, com baixo consumo de recursos de banda da rede e de processamento. Os resultados mostram que nosso método foi efetivo na detecção de ataques *port scan*.

**Palavras-chave:** IDS, IPS, OpenFlow, SDN, Segurança, Ataques por Varredura, *Lightweight*.

**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 security. 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 new paradigm called Software Defined Networking (SDN) and the OpenFlow protocol, brought greater flexibility to programming new protocols and performing tests in real environments, besides the possibility of implementationn 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 attacks. Several studies are addressing IDS/IPS methods for SDN, based on received flows data analysis. A flow data analysis, can lead to an increase in switching latency of the switches, due to the need to analyze the packet before routing it. There may also be network overhead when the packet is duplicated to be parsed by an external IDS in parallel with the packet forwarding. These problems make the IDS/IPS a bottleneck, and based on this context and benefited by the flexible features of SDN, this work presents a new port scan IPS for SDN based on the OpenFlow switch counters data. This work results on a non-intrusive and lightweight architecture, with low network overload and processing power consumption. The results show that our method was efective on detecting port scan attacks.

**Keywords:** IDS, IPS, OpenFlow, SDN, Security, Port Scan Attacks, Lightweight.

## LISTA DE ILUSTRAÇÕES

Figura 1 Incidentes Reportados ao Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br) - Janeiro a Dezembro de 2016 ..... 11
Figura 2 Planos de redes de computadores. ..... 15
Figura 3 Modelos de rede tradicional e SDN..... 16
Figura 4 Tabela de fluxo ..... 18
Figura 5 Diagrama simplificado do tratamento de um pacote no *switch* OpenFlow ..... 19
Figura 6 Formato da mensagem OpenFlow ..... 23
Figura 7 Estabelecimento de conexão TCP ..... 28
Figura 8 Ciclo de vida do IPS proposto ..... 39
Figura 9 Fluxo de funcionamento do IPS proposto ..... 40
Figura 10 Estrutura do corpo da mensagem de solicitação de estatísticas ..... 42
Figura 11 Corpo da resposta à requisição OFPMP_FLOW. ..... 43
Figura 12 Representação da árvore de dados no OpenDaylight. ..... 44
Figura 13 Corpo da resposta ofp-flow-stats no formato JSON ..... 45
Figura 14 Exemplo para obtenção de estatísticas da base de dados do OpenDaylight..... 46
Figura 15 Corpo da mensagem de alteração de entradas na tabela de fluxo..... 50
Figura 16 Topologia de rede. ..... 52
Figura 17 *Script* para a geração de requisições ao servidores. ..... 53
Figura 18 Registro de *log* criado com o recebimento de fluxos não maliciosos..... 54
Figura 19 Saída da execução do comando nmap para varredura individual de portas ..... 54
Figura 20 Registro de *log* criado com o recebimento de fluxos de varredura horizontal .. 55
Figura 21 Saída da execução do comando nmap para varredura automática de portas..... 55
Figura 22 Registro de *log* criado com o recebimento de fluxos de varredura vertical..... 56
Figura 23 Saída da execução do comando nmap para varredura FIN..... 56
Figura 24 Registro de *log* criado com o recebimento de fluxos iniciados sem *flag* SYN.. 56
Figura 25 Exemplo de fluxos na base de dados. ..... 57

## LISTA DE QUADROS

Quadro 1 Contadores da tabela de encaminhamento ..... 19
Quadro 2 Comparativo entre os trabalhos estudados ..... 38
Quadro 3 Exemplo de fluxos na tabela de encaminhamento. ..... 51
Quadro 4 Resultados das análises de varreduras. ..... 58
Quadro 5 Resultados das análises de varreduras após estabelecido o descarte. ..... 59

## 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* |
| URL | *Uniform Resource Locator* |
| WEB | *World Wide Web* |
| XML | *eXtensible Markup Language* |

SUMÁRIO


1 INTRODUÇÃO ..... 10
2 FUNDAMENTAÇÃO TEÓRICA ..... 14
2.1 Redes Definidas por Software ..... 14
2.1.1 Comutadores ..... 17
2.1.2 Controlador ..... 20
2.1.3 Protocolo OpenFlow ..... 22
2.2 Virtualização ..... 23
2.3 Emulador Mininet ..... 24
2.4 Segurança em redes ..... 25
2.4.1 Tipos de ameaças ..... 25
2.4.2 Técnicas de varredura ..... 26
2.4.3 Ferramentas de Varredura ..... 30
2.4.4 Sistemas de detecção e prevenção de intrusão ..... 31
3 ESTADO DA ARTE ..... 34
3.1 Soluções de IDS ..... 34
3.2 Soluções de IPS ..... 36
3.3 Objetivos ..... 38
4 DETECÇÃO E PREVENÇÃO DE ATAQUES PORT SCAN EM REDES SDN ..... 39
4.1 Arquitetura ..... 39
4.2 Coleta de estatísticas ..... 40
4.3 Detecção ..... 47
4.3.1 Detecção de varredura horizontal ..... 48
4.3.2 Detecção de varredura vertical ..... 48
4.3.3 Detecção de varredura mista ..... 49
4.4 Ações de reação ..... 49
5 AMBIENTE E RESULTADOS EXPERIMENTAIS ..... 52
5.1 Topologia de rede ..... 52
5.2 Resultados ..... 53
6 CONCLUSÃO E TRABALHOS FUTUROS ..... 60
REFERÊNCIAS ..... 61

# 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 a troca de informações (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 desenvolvidos e definidos 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.

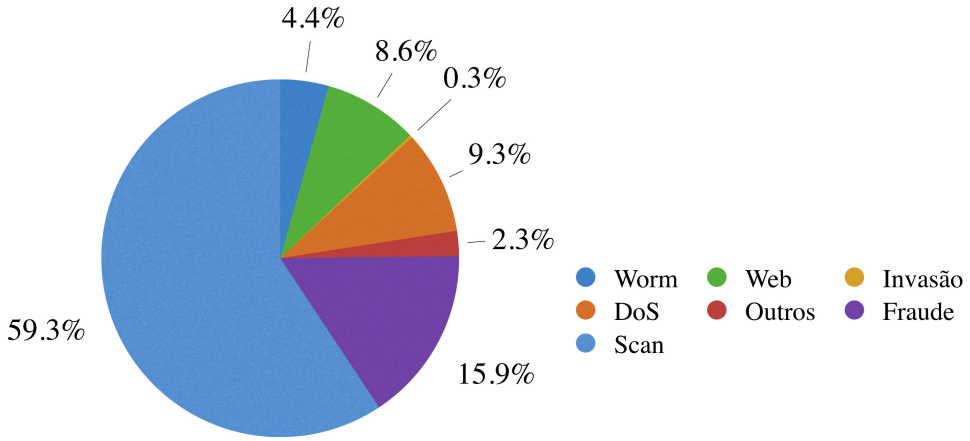
O paradigma de Redes Definidas por Software, SDN, e o protocolo OpenFlow (OPEN NETWORKING FOUNDATION, 2014) oferecem um caminho para vencer estes desafios, por meio de uma solução de implementação gradativa em redes de produção. O paradigma SDN possibilita a rápida configuração de uma rede conforme a 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 facilitando o gerenciamento da rede (KREUTZ; RAMOS; VERISSIMO, 2013).

Para que essa abstração ocorra, é necessária uma comunicação entre os planos de controle e dados. Essa comunicação pode ser realizada através do protocolo OpenFlow, que atualmente vem sendo o protocolo mais utilizado em SDN (LI; MENG; KWOK, 2016). 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 2016, foram reportadas 647.112 incidentes de segurança ao CERT.br (CERT.BR, 2016), responsável por tratar incidentes de segurança e computadores que envolvam redes conectadas à Internet brasileira. Desse número, 59% foram de ataques do tipo *port scan*, ou em português, varredura de porta como pode ser visualizado na Figura 1. Este alto número se deve basicamente por este tipo de ataque permitir a verificação de *hosts* e serviços ativos na rede e que possam ser utilizados como porta de entrada para outros tipos de ataques.

Figura 1 – Incidentes Reportados ao CERT.br - Janeiro a Dezembro de 2016




Fonte: Adaptado de CERT.br, 2017

Tradicionalmente, Sistemas de Detecção de Intrusão, ou *Intrusion Detection Systems* (IDS) (COMER, 2013), são consideradas ferramentas comuns para detectar 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. Em um Sistema de Prevenção de Intrusão, *Intrusion Prevention System* (IPS), essa tarefa de analisar e prevenir passa a ser realizada de forma automática, sem a necessidade de um profissional em segurança.

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).

Uma deficiência da maioria dos sistemas de detecção de intrusão é o fato de o seu funcionamento se basear na utilização de uma base de dados com assinatura de ataques: se um

ataque conhecido é detectado, a tentativa é bloqueada, caso contrário, podem passar sem qualquer restrição. Essa análise também gera uma carga elevada a ser administrada caso a rede possua elevados índices de atividade, já que são verificados todos os pacotes que nela trafegam (NOBRE, 2007).

Uma maneira de contornar o problema da detecção por assinatura, é a utilização de sistemas de detecção que analisam o comportamento da rede. Neste, um modelo de normalidade é estabelecido em condições adequadas de uso (sem ataque), e comparado com a atividade em andamento. Qualquer comportamento suspeito, como aumento do tráfego ou pacotes incompletos, pode vir a ser considerado suspeito (NOBRE, 2007).

Esta segunda técnica pode vir a reduzir a carga de administração por realizar a análise sobre uma amostra de pacotes, ou através da coleta de estatísticas disponíveis. Por outro lado, a detecção baseada em anomalia pode levar a alguns resultados não intuitivos, de forma a gerar um grande número de falsos positivos (HEBERLEIN, 2007).

Observando as estatísticas de segurança e as soluções de IPS disponíveis para SDN, foi discutido o seguinte problema: "É possível, através da coleta de estatísticas de tráfego, analisar, detectar e prevenir intrusões de forma eficaz em uma rede SDN?". Em SDN, através do protocolo Openflow, é possível a obtenção de estatísticas de tráfego de cada fluxo de dados diretamente dos comutadores, isso traz vantagens no que diz respeito a desempenho, pois não há a necessidade de sensores específicos. Além disso, a visão global de SDN permite a proteção em tempo real de todo um sistema, enquanto que em redes tradicionais essa proteção é apenas local, uma vez protegido de um ataque, o fluxo poderá tomar outras rotas para atingir seu objetivo, problema que pode ser evitado facilmente em SDN.

Com base no que foi discutido, este trabalho de conclusão apresenta uma metodologia de sistema de prevenção contra ataques de varredura de porta em SDN. Para isso, utiliza 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, 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. No Capítulo 4 é apresentada a metodologia utilizada no desenvolvimento deste trabalho, detalhes de sua implementação e

funcionamento. O ambiente utilizado para avaliação do IPS desenvolvido bem como os resultados obtidos são apresentados no Capítulo 5 e, por fim, no Capítulo 6, são descritas algumas considerações sobre este trabalho e para trabalhos futuros.

# 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 além da ferramenta utilizada para varredura de portas.

## 2.1 Redes Definidas por Software

As redes de computadores se tornaram parte da infraestrutura crítica de empresas, escolas e residências, tendo crescido bastante desde a sua origem. O sucesso das redes de computadores se deve, em grande parte, 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).

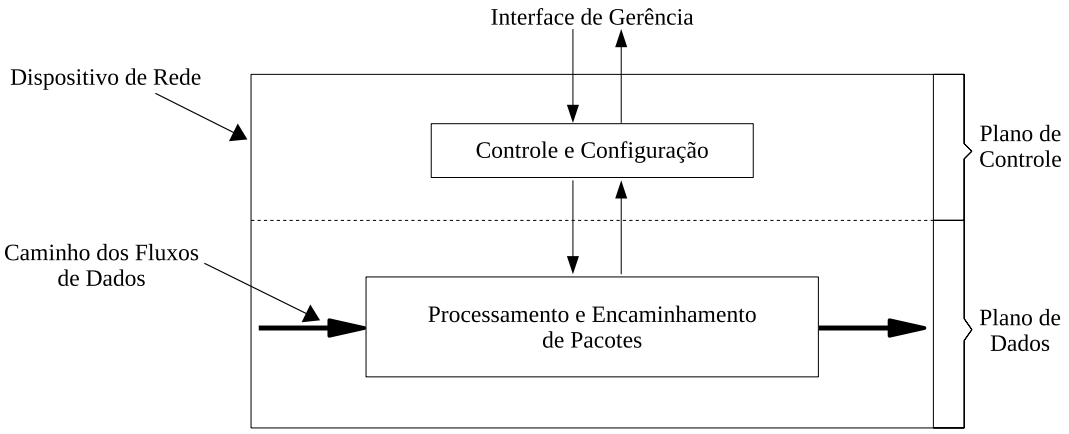
Por causa desta expansão, o trabalho dos pesquisadores da área tornou-se muito mais importante, porém mesmo com o grande número de equipamentos e protocolos criados para suportar essa expansão, ainda tem-se uma grande barreira. A maioria das ideias que surgem não conseguem ser testadas por falta de maneiras 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 (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. Decisões de roteamento, *firewall*, priorização de pacotes são de responsabilidade do plano de controle. 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 es-

sas 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, como pode ser visualizado na Figura 2, tornando a arquitetura de rede complexa e por consequência dificulta a sua configuração e o seu gerenciamento.

Figura 2 – Planos de redes de computadores.




Fonte: Comer, 2013.

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

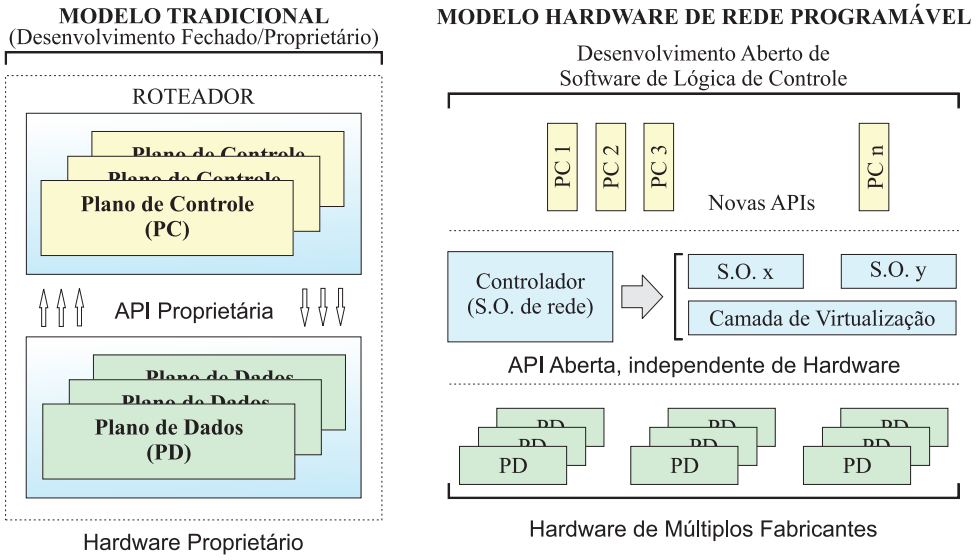
maior controle por parte do administrador da rede.

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 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 3 apresenta um comparativo entre o modelo tradicional de rede, 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 3 – 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 recomendada a utilização do protocolo *Secure Socket Layer* (SSL) (FREIER; KARLTON; KOCHER, 2011) porém algumas alternativas incluem *Transmission Control Protocol* (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 SDN são:

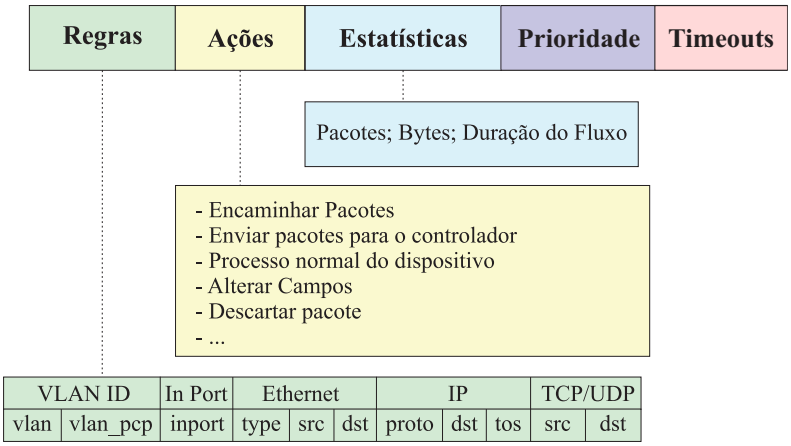
- Comutadores (*switches*) *OpenFlow*;
- Controlador; e
- Protocolo de comunicação.

Estes componentes podem fazer uso do protocolo OpenFlow e/ou de outros protocolos. Por ser o primeiro e também o mais utilizado, o protocolo OpenFlow é utilizado neste trabalho como padrão de comunicação entre os dispositivos.

### 2.1.1 Comutadores

É o elemento responsável pelo encaminhamento dos pacotes pela rede. Pode ser específico para OpenFlow, ou ter suporte ao mesmo. No comutador (*switch*) OpenFlow é mantida uma tabela de fluxo (*flow table*) que armazena informações sobre como os pacotes serão processados, estatísticas, prioridades 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 4, assim como as informações de ações e estatísticas.

Figura 4 – Tabela de fluxo

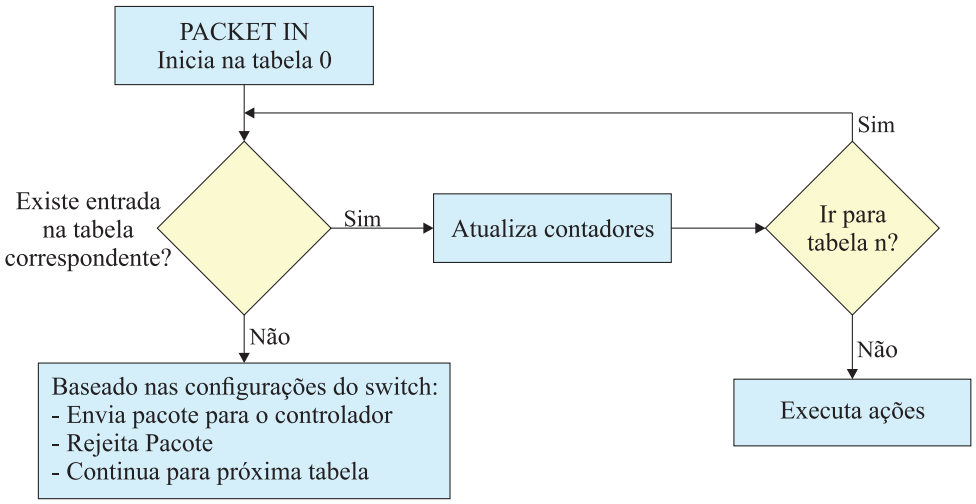



Fonte: Adaptado de Costa, 2014

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 5 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 (*Open Shortest Path First* (OSPF) (MOY, 1998), *Border Gateway Protocol* (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 5 – 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 encaminhamento

| **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)

Neste projeto o comutador a ser utilizado é o Open vSwitch (THE LINUX FOUNDATION, 2017), um *switch* virtual com suporte a OpenFlow. Este comutador é projetado para permitir a automatização de grandes redes através da extensão programática, suportando ainda interfaces e protocolos de gerenciamento como, por exemplo, NetFlow (CLAISE, 2004), sFlow (PANCHEN; MCKEE; PHAAL, 2001) e IPFIX (MARK *et al.*, 2008). Além disso, pode suportar a distribuição através de múltiplos servidores físicos (THE LINUX FOUNDATION, 2017).

### 2.1.2 Controlador

O controlador, como já citado, é o *software* responsável por tomar decisões e adicionar e remover as entradas na tabela de encaminhamento, de acordo com o objetivo desejado. 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 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 dependendo 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.

O controlador é parte integrante de uma arquitetura de rede SDN e para que sua comunicação com *switches* OpenFlow ocorra, o controlador deve ter suporte ao mesmo. Atualmente, existem várias implementações controlador disponíveis que implementam o protocolo OpenFlow, 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 componentes 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 realizado 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.

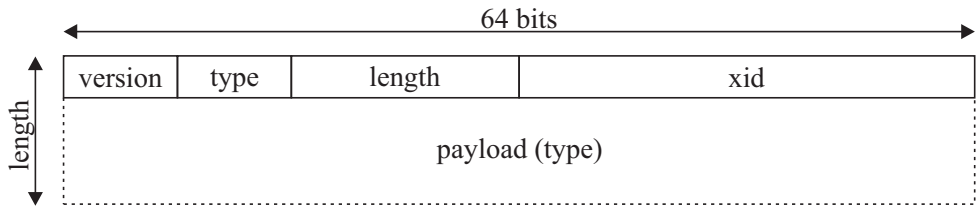
- ***Packet-In***: enviado pelo *switch* quando há uma ação explícita na tabela de fluxos para que seja enviado para o controlador ou quando não há um *match* para o pacote.
- ***Flow-Removed***: informa o controlador sobre a remoção de regras no *switch*.
- ***Port Status***: informa o controlador sobre uma mudança em alguma porta.
- ***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 6.

Figura 6 – 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. Já o *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 a 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. 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 emular 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 usuários 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 por exemplo, *firewalls*. Os principais tipos de ameaças são estudados 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 (HOUAISS; VILLAR; 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 (SPEROTTO *et al.*, 2010) 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 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 (*Botnets*)** - Grupo de computado-

res 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 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 CERT.br, responsável por tratar incidentes de segurança e computadores que envolvam redes conectadas à Internet brasileira, foram reportados 647.112 incidências de segurança somente no ano de 2016, sendo mais da metade (59%), ataques do tipo *port scan*.

### 2.4.2 Técnicas de varredura

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, 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 desativadas ao mesmo tempo (COMER, 2013), 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.

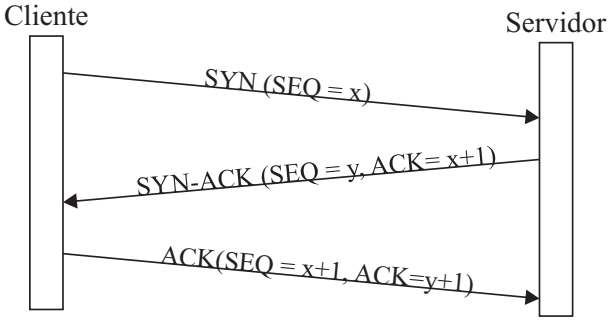
A seguir, a máquina receptora, chamada de servidor, recebe o segmento inicial que provém do cliente e envia-lhe um aviso de recepção, isto é, um segmento cuja *flag* ACK é de 1 e a *flag* SYN é de 1 (porque ainda se trata de uma sincronização). Este segmento contém o número de ordem do servidor, que 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 sincronização). 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 7), também chamada de *handshake*, ou, aperto

de mãos em português, as duas máquinas estão conectadas e a comunicação pode ser efetivada.

Figura 7 – 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. Segundo Speroto *et al.* (2010), há três categorias de varredura, são elas:

- ***scan* horizontal** - quando um *host* de origem varre uma porta especifica em diferentes *hosts* alvo;
- ***scan* vertical** - quando um *host* de origem verifica várias portas distintas de um mesmo *host* alvo; e
- ***scan* misto** - quando há a combinação das varreduras vertical e horizontal.

Existem várias técnicas de varredura de porta disponíveis e podem facilmente ser automa-

tizadas por ferramentas como Nmap (LYON, 2009). Alguns métodos utilizados para varredura são 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 má-

quina 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 Ferramentas de Varredura

Para que as varreduras sejam efetuadas, tem-se a possibilidade de utilizar ferramentas que possibilite a varredura utilizando as diferentes formas citadas na seção anterior. Uma das ferramentas mais utilizadas, e que foi utilizada neste trabalho é o Nmap (LYON, 2009).

O Nmap é um *software* que oferece uma gama muito grande de recursos e funcionalidades, como detecção do Sistema Operacional remoto, o serviço e a versão que está em uso no host, o exame de ociosidade por identificação (ID) de Internet Protocol (IP), o rápido exame de multiportas por ping entre tantas outras. Possui versões para plataformas Unix, Windows, e MacOS sendo utilizado tanto por interface console como também em interface gráfica. O software Nmap é um utilitário livre e de código aberto, usado para exploração de redes, segurança e auditoria, capaz de examinar grandes redes ou simplesmente um único host. A função principal do Nmap é realizar uma varredura em portas TCP e o retorno dessa varredura é classificado em um dos seguintes estados: aberta, fechada, filtrada, não filtrada e a combinação de aberta/filtrada ou fechada/filtrada (LYON, 2009). Vários outros softwares que são utilizados para gerencia e controle de redes de computadores fazem uso do Nmap pois pode ser usado diretamente, sempre que se desejar uma verificação de portas em um *host* que esteja em uma rede local ou na Internet. O uso mais simples do Nmap é escanear diretamente uma máquina da rede, onde uma quantidade enorme de portas TCP será examinada na máquina alvo, e cada porta será classificada de acordo com seu estado. Na linha de comando do Nmap, tudo que não for uma opção ou argumento de opção será tratado como uma especificação de hospedeiro alvo. O caso mais simples é a especificação de um endereço IP ou nome de hospedeiro alvo para exame (LYON, 2009).

### 2.4.4 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 (2014) , "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. O objetivo de um IDS é discriminar tentativas de intrusão e preparação de intrusão do uso normal do sistema.

Infelizmente, arquiteturas IDS/IPS utilizadas atualmente possuem muitas barreiras para gerir nós distribuídos. 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. 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. Além disso, 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 nos 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 (KOLPYAKWAR; INGLE; DESHMUKH, 2017).

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 é 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 (PAXSON, 1998) e Snort (ROESCH, 1999) apresentam alto consumo de recursos quando confrontado com a enorme quantidade de dados de alta taxa de transferência encontrados (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, são analisados os padrões de comunicação dentro da rede, ao invés do 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 quantidade de bytes transferidos, o número de pacotes enviados e o tempo que determinado fluxo de dados esteve ativo. Essas informações podem então ser exportadas para outros sistemas analisarem, para então, serem usados 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 ESTADO DA ARTE

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 e 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) apresentam 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 Braga, Mota e Passito (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 e 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 uma análise do tráfego para a construção de um gráfico de ataques e posteriormente 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 | *scan* | Tráfego |
| Shin *et al.* (2013) | Não | Sim | DoS | Tráfego |
| Xing *et al.* (2013) | Não | Sim | N/A | Tráfego |
| Chung *et al.* (2013) | Sim | Sim | DDoS | 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

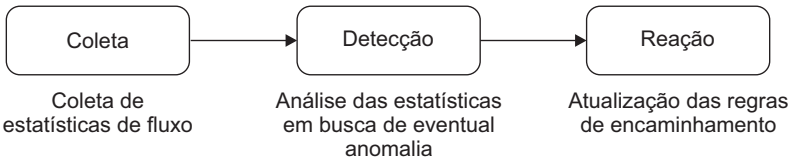
Considerando os trabalhos estudados na literatura, os quais são sumarizados no Quadro 2, percebe-se que o uso de OpenFlow para a implementação de segurança mostrou-se bastante promissor. Porém, observou-se que o foco dado por parte destes não implementa recursos de contramedida para ataques. Dos trabalhos analisados, apenas os trabalhos de Nagahama *et al.* (2012), Chung *et al.* (2013), Lopes *et al.* (2014) e Wang *et al.* (2015) apresentam as funcionalidades de detecção e prevenção contra intrusos porém utilizam a análise de tráfego em suas implementações. Os trabalhos de Adrichem, Doerr e Kuipers (2014), Zhang (2013) e de Braga, Mota e Passito (2010) utilizam, por sua vez, a análise com base em estatísticas utilizando contadores das tabelas de encaminhamento, porém não contemplam *port scan* e não possuem medidas protetivas.

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. A metodologia de desenvolvimento deste trabalho, bem como sua arquitetura serão apresentados no Capítulo 4 deste trabalho.

# 4 DETECÇÃO E PREVENÇÃO DE ATAQUES PORT SCAN EM REDES SDN

O objetivo principal deste trabalho é apresentar um IPS baseado em anomalia que com agrupamento de dados possa efetuar a detecção de intrusão, com foco em ataques de varredura de porta, e prover medidas de proteção contra os mesmos. Para isto, a solução foi desenvolvida considerando um ciclo de vida de três etapas (Figura 8): 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 disponibilizar as informações ao controlador para que este 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 tabelas de encaminhamento nos *switches* OpenFlow. Cada uma dessas etapas será discutida no decorrer deste capítulo. A implementação desse trabalho aborda basicamente fluxos de rede TCP, sendo assim, para outros protocolos alterações se fazem necessárias.

Figura 8 – Ciclo de vida do IPS proposto



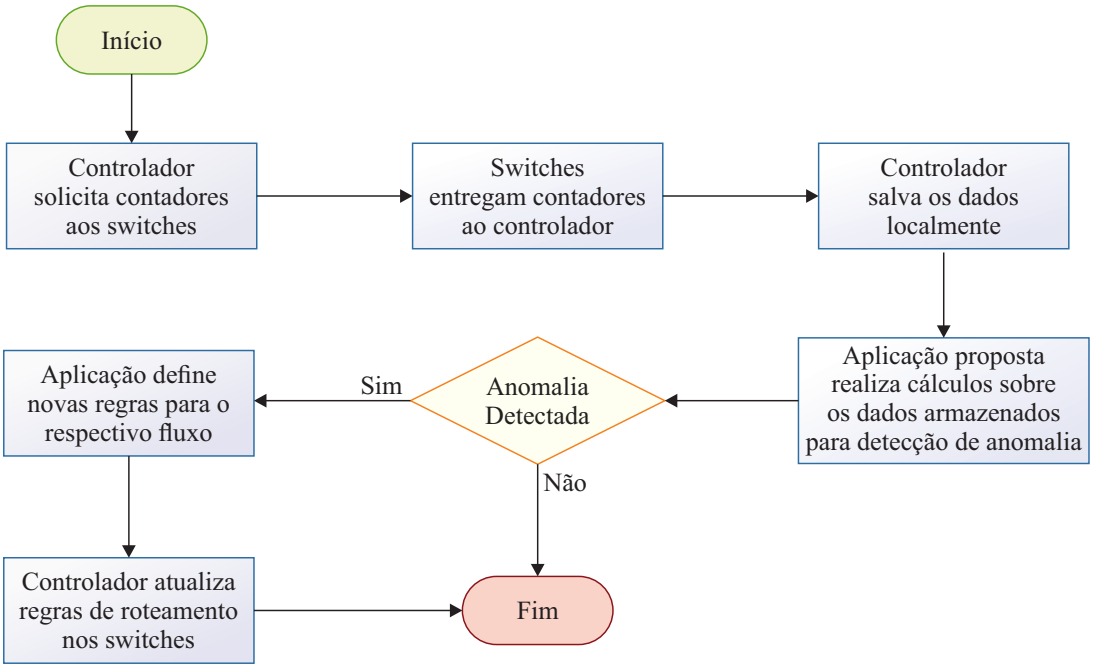

Fonte: Elaborado pelo autor.

Este capítulo está organizado da seguinte forma: na seção 4.1 é apresentado à arquitetura do sistema desenvolvido, 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. Esta aplicação foi desenvolvida como uma extensão (*plu-*

Figura 9 – Fluxo de funcionamento do IPS proposto




Fonte: Elaborado pelo autor.

*gin*) ao controlador OpenDaylight (OPENDAYLIGHT, 2016). Na arquitetura desenvolvida, 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 9 e apresentado nas seções seguintes.

## 4.2 Coleta de estatísticas

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. Neste trabalho este intervalo foi definido em três segundos com base em resultados obtidos em testes realizados. Este período não produz grande carga na rede e possibilita que várias tentativas de varredura de porta possam ser realizadas, obtendo-se mais fluxos distintos a cada coleta. Também foi adicionado um *timeout* de quinze segundos para os fluxos registrados na tabela de encaminhamento, com isso, a frequência na consulta também permite que as estatísticas de fluxo sejam atualizadas com maior frequência. Apesar desse intervalo possibilitar uma grande quantidade de varreduras antes que as mesmas sejam detectadas, o fato de prover um bloqueio posterior impossibilita que o atacante efetue outros ataques ao término da varredura.

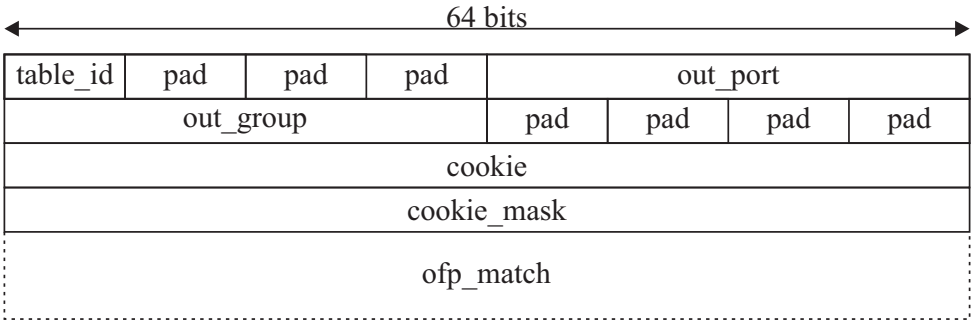
Utilizando as mensagens definidas pelo protocolo OpenFlow e brevemente discutidas na seção 2.1.3, a coleta de estatísticas é facilitada, passando a ser feita pelo controlador da rede. O controlador realiza, através do protocolo OpenFlow, a coleta de contadores presentes nas tabelas de fluxo dos *switches*. 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 aos fluxos, OpenFlow fornece alguns métodos para obtenção de informações detalhadas de um fluxo específico, ou de um conjunto de fluxos.

Segundo a especificação OpenFlow (OPEN NETWORKING FOUNDATION, 2014) se o controlador desejar obter estatísticas de um fluxo OpenFlow, ele deve enviar uma mensagem *multipart* (ofp_multipart_request) do tipo OFPMP_FLOW 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 comumente 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 10 a seguir:

Figura 10 – Estrutura do corpo da mensagem de solicitação de estatísticas




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

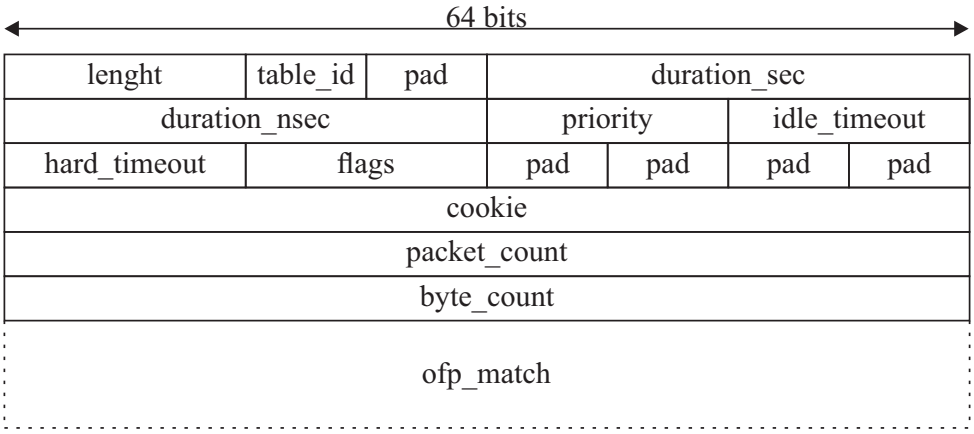
O campo *table_id* deve ser preenchido com o número da tabela onde os fluxos desejados estão armazenados. Se o controlador desejar obter estatísticas de todos os fluxos dessa tabela, apenas essa informação é obrigatória, caso contrário, se o controlador desejar estatísticas de um fluxo específico, devem ser preenchidos os campos de *match*, como endereço de origem e porta destino. Os campos *pad* não necessitam de informação, servem apenas para completar o pacote.

Após enviar a 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 partes na sequência e deve ler a mensagem para obter as informações estatísticas.

Quando o *switch* recebe a mensagem OFPMP_FLOW, ele primeiramente obtém a entrada do fluxo correspondente com base nos campos informados na requisição. Uma vez obtidas as entradas de fluxo correspondentes, o mesmo constrói uma mensagem de resposta (Figura 11) 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 11 – Corpo da resposta à requisição OFPMP_FLOW.



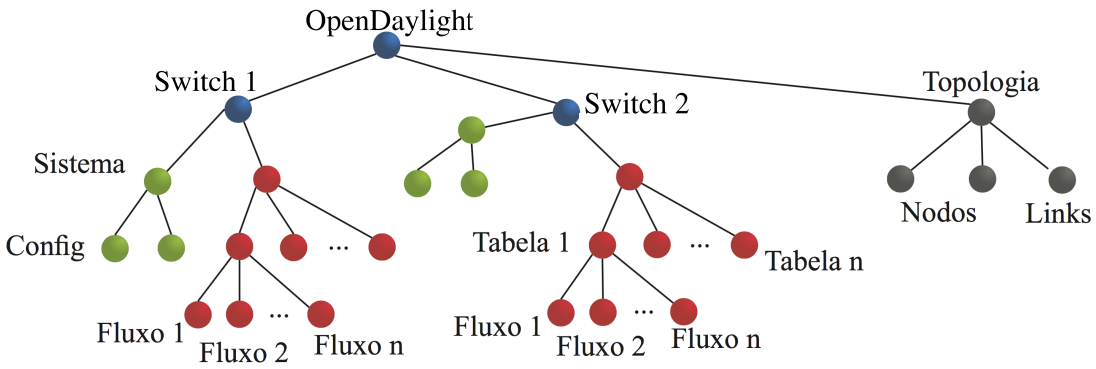

Fonte: Elaborado pelo autor a partir de informações da especificação OpenFlow.

- *lenght* indica o tamanho da respectiva entrada de fluxo;
- *table_id* refere-se ao identificador da tabela cujo fluxo foi solicitado;
- *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 $duration_sec * 10^9 + duration_nsec$;
- *flags* possui informações das *flags* TCP como SYN, ACK e FIN;
- *packet_count* é um contador de medição de pacotes que trafegam com o respectivo *match* desse fluxo;
- *byte_count* indica o número de *bytes* do respectivo fluxo; e
- *ofp_match* é uma lista de zero ou mais propriedades específicas para as regras, como portas de origem e destino, tipo de rede, endereços de origem e destino, etc.

O controlador OpenDaylight implementa uma abstração dos pacotes e protocolos utilizados, tornando a tarefa de desenvolver troca de mensagens com os *switches* mais ágil e prática. A sua estrutura de funcionamento é baseada em uma árvore, o que possibilita o endereçamento de qualquer elemento/sub-árvore que esteja sob o domínio do controlador. Nesta árvore, ilustrada na Figura 12, tem-se o controlador interligado aos *switches*, que por sua vez possuem tabelas e estes, possuem nodos que correspondem às informações de fluxos. Sendo assim, pode-se obter estatísticas de fluxo de um determinado *switch* através da leitura sucessiva de seus nodos. Para obter as estatísticas do primeiro fluxo presente da segunda tabela do primeiro *switch*, deve realizar a leitura do nodo "*Switch* 1", a partir deste nodo pode-se realizar uma leitura de todas as tabelas presentes até que se encontre a "Tabela 2", estando na tabela dois pode-se obter as

estatísticas agregadas de fluxo ou de fluxos específicos, que neste exemplo é o "Fluxo 1".

Figura 12 – Representação da árvore de dados no OpenDaylight.




Fonte: Elaborado pelo autor com base na especificação OpenDaylight.

A configuração dos dispositivos e a coleta de estatísticas de fluxos no OpenDaylight pode ser obtida de forma muito simples através de um *browser* utilizando-se uma API chamada *restconf*. Um exemplo de requisição de estatística de fluxo pode ser obtida através do Localizador Uniforme de Recursos (*Uniform Resource Locator* (URL)): http://localhost/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/tcipsflow-1

Onde:

- http://localhost - é o endereço *World Wide Web* (WEB) do controlador;
- restconf - é o nome da API ao qual está se requisitando a informação;
- config - é a base de dados de configuração do OpenDaylight;
- opendaylight-inventory:nodes - indica que a partir desse ponto a estrutura é baseada em nodos;
- nodo/openflow:1 - indica que o nodo a ser consultado é o chamado openflow:1;
- table/0 - indica que a tabela do nodo acima a ser consultada é a de identificação 0; e
- flow/tcipsflow-1 - indica que o fluxo de nome "tcipsflow-1"deverá ser consultada na tabela acima.

Se o desejado for obter a informação de todos os fluxos da tabela, a informação de fluxo (*flow*) pode ser omitida da URL, como por exemplo: http://localhost/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0. O resultado será um arquivo em formato *JavaScript Object Notation* (JSON) ou *eXtensible Markup Language* (XML) como exibido na Figura 13.

Figura 13 – Corpo da resposta ofp-flow-stats no formato JSON


```
{ "flow-node-inventory:table": [
    {"id": 0,
      "opendaylight-flow-table-statistics:flow-table-statistics": {
        "packets-looked-up": 37,
        "active-flows": 3,
        "packets-matched": 19 },
      "flow": [
        { "id": "tcipsflow-1",
          "table_id": 0,
          "instructions": {
            "instruction": [
              { "order": 0,
                "apply-actions": {
                  "action": [
                    { "order": 0,
                      "output-action": {
                        "output-node-connector": "CONTROLLER"}}]}}]},
          "priority": 1,
          "opendaylight-flow-statistics:flow-statistics": {
            "duration": { "nanosecond": 583000000, "second": 156 },
            "packet-count": 1,
            "byte-count": 74 },
          "match": {
            "ethernet-match": { "ethernet-type": { "type": 2048 }},
            "ip-match": { "ip-protocol": 6 }},
          "idle-timeout": 0,
          "hard-timeout": 0 }]}]}
```


Fonte: Extraido da interface do OpenDaylight executado pelo autor.

Neste exemplo, cada nodo é representado por um sub-elemento JSON. A tabela possui dados agregados de pacotes transmitidos e fluxos. Cada fluxo por sua vez possui informações de *match* para a verificação do pacote recebido, *instructions*, instruções que devem ser tomadas se os campos em *match* forem satisfeitos, campos de *timeout*, *flags* e estatísticas conforme o especificado pelo protocolo Openflow.

Apesar de a coleta ser facilmente realizada através da API *restconf*, esta aplicação implementa a coleta diretamente sobre a base de dados do controlador, fazendo uso dos métodos desenvolvidos pela comunidade de desenvolvedores do OpenDaylight. Esta escolha torna-se mais viável considerando-se que esta aplicação foi desenvolvida como uma parte do controlador, tendo portanto, acesso direto à sua base de dados e dispositivos. Na Figura 14 é ilustrada a obtenção de um fluxo diretamente da base de dados OpenDaylight. Neste exemplo cada elemento da árvore deve ser verificado até que o nodo que contém informações de fluxo seja encontrado.

Figura 14 – Exemplo para obtenção de estatísticas da base de dados do OpenDaylight

```
Nodos nodos = obterNodosDaBaseDeDados();
/* Iteracao sobre cada nodo */
for (Iterator<Node> iterator = nodes.getNode().iterator();
    iterator.hasNext();) {

  Node childNode = iterator.next();
  /* Iteracao sobre cada tabela */
  for (Iterator<Table> iterator2 = childNode.getTable().iterator();
  iterator2.hasNext();) {

    Table table = iterator2.next();
    /* Iteracao sobre cada fluxo */
    for (Iterator<Flow> iterator3 = table.getFlow().iterator();
    iterator3.hasNext();) {

      Flow flow = iterator3.next();
      /* Obtem dados do cabecalho - match */
      Match match = flow.getMatch();
      Layer3Match layer3Match = match.getLayer3Match();
      Layer4Match layer4Match = match.getLayer4Match();
      ...
      /* Obtem estatisticas */
      FlowStatisticsData data = flow.getAugmentation(
        FlowStatisticsData.class);
      FlowStatistics flowStatistics = data.getFlowStatistics();
      /* Salva informacoes para posterior analise */
      setMatch(match);
      setPacketCount(flowStatistics.getPacketCount().getValue());
      setByteCount(flowStatistics.getByteCount().getValue());
      setDurationSeconds(flowStatistics.getDuration().getSecond()
        .getValue());
      setDurationNanoSeconds(flowStatistics.getDuration().getNanosecond()
        .getValue());
    }
  }
}
```

Fonte: Elaborado pelo autor.

A base de dados criada para armazenar as estatísticas é composta por campos de *match* necessários para diferenciar os fluxos e por campos dos contadores por fluxo da tabela de encaminhamento conforme listado abaixo.

- nodeName - nome do *switch* analisado;
- etherType - o tipo de pacote recebido, neste projeto são analisados apenas pacotes TCP;
- srcIpv4 - o IP de origem do fluxo;
- dstIpv4 - o IP de destino do fluxo;
- srcPort - a porta de origem do fluxo;
- dstPort - a porta de destino do fluxo;
- incomeTime - o instante em que o primeiro pacote foi recebido;

- durationSeconds - a duração do fluxo em segundos;
- durationNanoSeconds - a duração do fluxo em nano segundos;
- packetCount - o número de pacotes do fluxo trafegados pelo switch;
- byteCount - o número de bytes do fluxo trafegados pelo switch;

Uma vez armazenadas as estatísticas de fluxo na base de dados criada, o processo de coleta é finalizado até iniciar um novo ciclo.

### 4.3 Detecção

Periodicamente são analisadas as informações de fluxos presentes na base de dados alimentada na coleta dos contadores. A cada análise são executadas as seguintes etapas:

- Seleção dos fluxos armazenados na base de dados que possuam número de pacotes reduzido, característico de fluxos de varredura de porta, e que tenham sido gerados nos últimos segundos;
- Agrupamento dos fluxos selecionados, por endereço de origem e destino e porta de destino;
- Obtenção da quantidade de fluxos com endereços de origem e destino semelhantes em um pequeno intervalo de tempo predefinido. Eventuais tentativas de varredura de porta podem ocorrer em qualquer sistema, sendo assim, um único fluxo não pode gerar um evento para bloqueio.
- Classificação do tipo de varredura (horizontal, vertical e mista) com base nas informações de endereço de origem e destino e porta de destino.
- Inserção do endereço de origem em uma lista de endereços para descarte.

Em geral, fluxos de tentativas de varredura de porta limitam-se em no máximo três pacotes, por exemplo *TCP Connect* possui pacotes SYN, SYN/ACK e ACK, com exceção dos caso onde há retransmissão de pacotes devido à perdas. Além disso, estatisticamente, ataques do tipo *port scan* ocorrem em intervalos pequenos de tempo, em média menos de um minuto. É sobre esse conjunto de fatores que a aplicação foi desenvolvida.

Como há uma coleta periódica das estatísticas de fluxo, é possível analisar se houve aumento no número de pacotes entre uma consulta e outra. Para definir se um fluxo é considerado uma tentativa de varredura, neste projeto foi estabelecido um número máximo de cinco pacotes para o mesmo. Este número corresponde aos três pacotes responsáveis pelo *handshake* da cone-

xão com uma tolerância de dois pacotes em caso de retransmissão. Por este número de pacotes ser baixo, a probabilidade de se ter um fluxo não malicioso detectado como varredura é muito baixa.

Apenas uma varredura de porta não é considerada como ataque, para que fluxos sejam considerados ataques, algumas regras foram definidas de acordo com o tipo de varredura efetuada, essas regras são discutidas nas seções a seguir.

### 4.3.1 Detecção de varredura horizontal

Como discutido na Seção 2.4.2, a detecção de varredura horizontal de portas é desenvolvido pela premissa de que vários fluxos, para uma mesma porta em diferentes *hosts* alvo são originadas de um mesmo *host* de origem. Nesse tipo de intrusão a aplicação desenvolvida analisa a origem e o destino das conexões.

Endereços IP de origem, que possuam fluxos com número de pacotes igual ou inferior a cinco e que estejam tentando conectar-se à no mínimo três *hosts* em uma mesma porta, são consideradas anomalias. Assim, o endereço de origem é adicionado na lista de endereços para descarte. O número de *hosts* alvo necessários para caracterizar o ataque pode ser ajustado conforme o tamanho da rede no qual será utilizado, neste projeto, como a rede possui tamanho limitado, foi definido em três *hosts*.

### 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. Para esta classificação foi criada uma verificação com base em pesos, a fim de possibilitar uma maior sensibilidade aos ataques para portas mais utilizadas. Portas que em geral sofrem mais ataques de varredura, como as de serviços Telnet (POSTEL; REYNOLDS, 1983), possuem um peso maior sobre as demais portas. Quando a soma dos pesos de cada fluxo ultrapassar um limiar predefinido, é caracterizado um ataque de varredura.

Neste projeto foi estabelecido um peso cinco para portas com maior probabilidade de varredura segundo estatísticas de incidentes reportados ao CERT.br (CERT.BR, 2016), sendo estas definidas neste trabalho em 22, 23, 25 e 3389, e peso três para as demais portas. Quando a soma de pesos dos fluxos de mesma origem e destino ultrapassar o limiar estabelecido em

quinze, um ataque de varredura pode ter sido realizado, sendo assim, é inserido seu endereço na lista de endereços para descarte. O limiar de detecção pode ser ajustado conforme necessidade da rede, para este trabalho foi utilizado um limiar quinze para facilitar os testes e cálculos dos resultados, sendo necessário três ataques às portas com maior probabilidade e cinco ataques para as demais portas.

### 4.3.3 Detecção de varredura mista

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 no mínimo dois *hosts*.

No mínimo um destes *hosts* também devem apresentar uma soma de pesos de varredura vertical estabelecido em seis. Resumindo, se dois *hosts* tiverem determinada porta verificada e ao menos um deles tiver uma porta distinta verificada, é considerado um ataque de varredura.

## 4.4 Ações de reação

Inicialmente, todos os comutadores possuem suas tabelas de encaminhamento vazias. Sendo assim, ao receber um pacote ocorre um *table miss* e o pacote é encaminhado para o controlador através de uma mensagem *Packet In*.

O controlador por si só, não realiza operações sobre a mensagem recebida, por isso, foi desenvolvido juntamente com este IPS, um interpretador de mensagens *Packet In*. Este analisa os dados do cabeçalho do pacote obtendo as informações de origem e destino do fluxo e os compara com os endereços armazenados na lista de endereços maliciosos para descarte. Dependendo do resultado, o controlador poderá adicionar no *switch* uma regra de encaminhamento o ou tomar uma ação de prevenção, adicionando uma regra para descarte do fluxo.

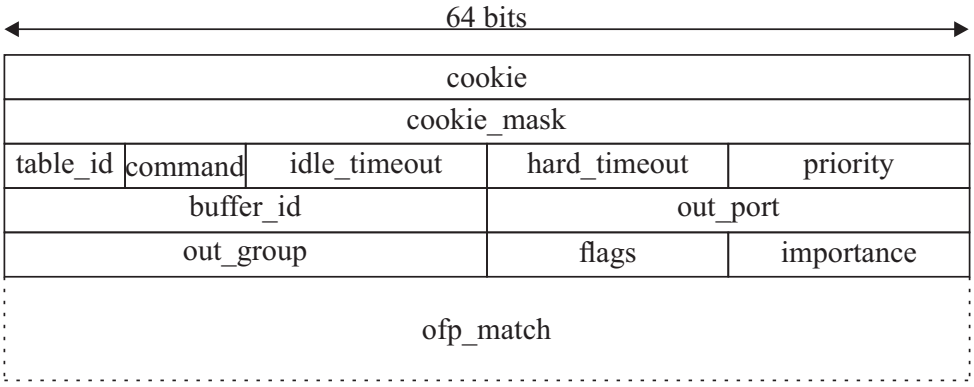
Mensagens para alteração da tabela de encaminhamento podem ter os seguintes tipos:

- **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 duas maneiras: através da requisição do controlador e pelo mecanismo de *timeout*. O mecanismo de *timeout* é executado pelo *switch* independentemente do controlador e é baseado na configuração e no estado das entradas de fluxo. Neste trabalho, foi definido um intervalo de *timeout* para cada fluxo para que não ocorra *overflow* de regras nas tabelas de encaminhamento. Um fluxo só permanecerá na tabela por intervalo de quinze segundos após o recebimento de seu último pacote, com exceção para as entradas de fluxo para descarte, nestes o *timeout* foi definido em zero a fim de prover a proteção até que o administrador, manualmente, o remova da tabela.

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 15.

Figura 15 – 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, sem a necessidade de consultar o controlador.

Algumas medidas de prevenção também foram preestabelecidas no controlador. Por padrão toda conexão não maliciosa, deve iniciar a comunicação com um pacote SYN. Sendo assim, ao receber um pacote com *flag* diferente da SYN, o controlador imediatamente descarta o pacote, prevenindo assim o ataques do tipo ACK, exploração FIN e Xmas Tree.

Os fluxos adicionados pelo controlador nas tabelas de encaminhamento possuem como regras as informações chave do cabeçalho mencionadas, desta forma, cada fluxo recebido irá possuir uma única entrada na tabela. Neste instante é também configurado um *timeout* para este fluxo. Este *timeout* foi estabelecido em quinze segundos, tempo permite que o fluxo não seja removido da tabela caso houver um pequeno atraso na transmissão dos pacotes além de possibilitar que coletas de estatísticas sejam realizadas, pois uma vez removido o fluxo, não é mais possível obter suas informações. O Quadro 3, ilustra uma tabela de fluxo com as regras utilizadas pelo *software* proposto, regras não utilizadas foram omitidas da tabela para melhor leitura.

Quadro 3 – Exemplo de fluxos na tabela de encaminhamento.

| REGRAS | | | | | AÇÃO | PRIORIDADE | TIMEOUT |
|---|---|---|---|---|---|---|---|
| type | src_ip | dst_ip | src_port | dst_port | | | idle_timeout |
| TCP | 10.0.0.11 | 10.0.1.124 | 36987 | 80 | encaminhar | 50000 | 15 |
| TCP | 10.10.0.45 | 10.10.2.43 | 23234 | 80 | encaminhar | 50000 | 15 |
| TCP | 10.0.0.114 | 10.10.2.29 | 35455 | 22 | encaminhar | 50000 | 15 |
| TCP | 10.10.2.24 | 10.0.0.12 | 32444 | 25 | descartar | 50000 | 0 |

Fonte: Elaborado pelo autor.

Neste exemplo, os três primeiros fluxos são de encaminhamento e foram criados com um *idle_timeout* de quinze segundo. Se nenhum novo pacote desse fluxo for recebido nesse intervalo de tempo, o fluxo é removido da tabela de encaminhamento. O quarto fluxo é de descarte, ou seja, quando for recebido um fluxo onde os campos de cabeçalho dos pacotes correspondam aos campos das regras, o mesmo será descartado. Este último possui um *timeout* nulo, o que significa que a regra é permanente e, se necessário for, deverá ser removida da tabela através de uma mensagem do controlador.

A natureza de visão global da rede possibilitada por SDN permite que a aplicação faça a detecção de switches individuais e envie fluxos de descarte para os demais, impossibilitando que tais fluxos tomem rotas diferentes para o *host* alvo. Além disso, controladores distintos podem efetuar a troca de informações referentes a fluxos de varredura de porta, possibilitando a proteção de outras redes não gerenciadas pelo mesmo controlador.

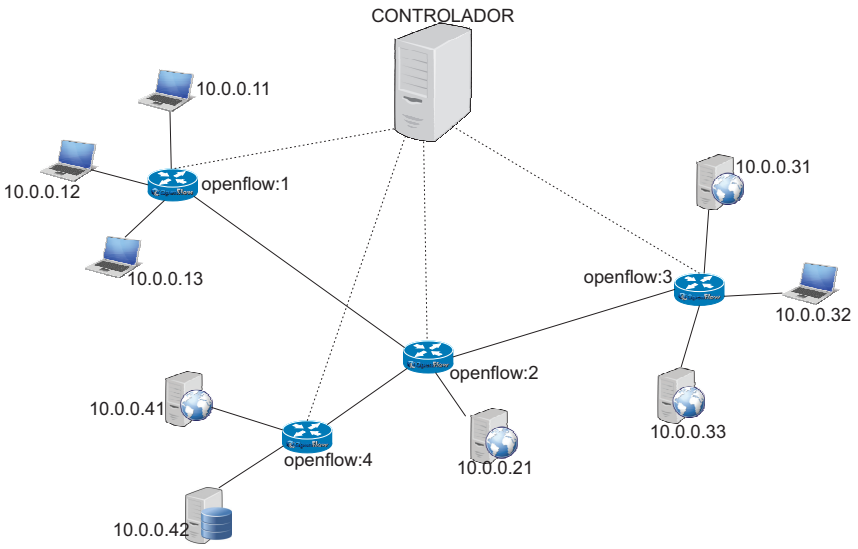
# 5 AMBIENTE E RESULTADOS EXPERIMENTAIS

O sistema desenvolvido foi avaliado através da realização de experiências práticas. Para viabilizar tal demonstração, foi desenvolvido um protótipo virtual utilizando o software Mininet (MININET, 2016). Mininet possibilita criar uma rede virtual escalável utilizando comutadores Open vSwitch (PFAFF *et al.*, 2009) através de um simples *script* em Python(PYTHON SOFTWARE FOUNDATION, 2017), facilitando a configuração de comutadores, bem como também, das conexões entres os mesmos. Além disso, Mininet possibilita a adição de *hosts* virtuais com as mesmas funcionalidades do *host* hospedeiro, tornando possível a criação de servidores WEB, de banco de dados, testes de comunicação entre computadores e testes de ataque à servidores. O plano de controle por sua vez é executado pelo controlador OpenDaylight (OPENDAYLIGHT, 2016) sob uma máquina física, sendo a comunicação entre o mesmo e os comutadores estabelecida através do próprio *software* Mininet.

## 5.1 Topologia de rede

A topologia de rede adotada nos testes é modelada de maneira bem simplificada. Um controlador SDN OpenDaylight realiza o gerenciamento de quatro *switches* OpenFlow virtuais que por sua vez, realizam a interconexão entre servidores com serviços ativos para recepção de ataques, e computadores de uso geral que são utilizados para a realização das requisições e ataques aos servidores. A Figura 16 apresenta o modelo da topologia de rede utilizada para os testes.

Figura 16 – Topologia de rede.




Fonte: Elaborado pelo autor.

## 5.2 Resultados

Para a realização dos testes, diferentes fluxos TCP foram originados a partir dos quatro computadores virtuais. Isto foi possível através da criação de *scripts* que realizam o acesso, a requisição e o *download* de conteúdo dos cinco servidores criados. A Figura 17 exemplifica um *script* que realiza duzentas mil requisições a diferentes servidores de aplicação e banco de dados.

Figura 17 – *Script* para a geração de requisições ao servidores.

```
#!/bin/bash
counter=1
while [ $counter -le 200000 ]
do
  curl 10.0.0.31:22
  psql -h 10.0.0.42 -U postgres -d testdb -c "SELECT *  FROM testtable"
  curl 10.0.0.32:80
  ...
  curl 10.0.0.41:25
  curl 10.0.0.31:23
done
```

Fonte: Elaborado pelo autor.

O número de pacotes gerados pelos fluxos acima, varia de 10 a 3000, dependendo do servidor. Através de registros de *log* gerados pela aplicação desenvolvida e pela lista de endereços considerados ameaças foi possível verificar que o *software* desenvolvido não classificou os fluxo gerados como ataque, não havendo portanto falsos positivos. Um resultado esperado, pois esse número de pacotes não é característico de uma varredura de porta.

O IPS desenvolvido também registra a diferença de tempo entre o instante de tempo em que um novo pacote SYN é recebido pelo controlador e o instante de tempo em que o fluxo é enviado para os *switches*. Essa diferença foi de aproximadamente quatro milissegundos, um tempo relativamente pequeno, que não gera atraso significativo ao fluxo pois não ocorre a cada pacote, sendo apenas um atraso inicial ao encaminhamento dos mesmos.

Na Figura 18 é apresentado parte do *log* gerado pelo IPS desenvolvido e armazenado no diretório de *logs* do controlador OpenDayight, neste, a cada fluxo inserido é registrado um evento indicando o fluxo, tando de encaminhamento como de retorno, e o tempo necessário para que o fluxo fosse registrado na tabela de encaminhamento.

Figura 18 – Registro de *log* criado com o recebimento de fluxos não maliciosos

```
17:57:02,868 | INFO | FlowCommit | Adicionado fluxo tcipsflow-38I em 3ms
17:57:02,871 | INFO | FlowCommit | Adicionado fluxo tcipsflow-38O em 6ms
17:57:02,874 | INFO | FlowCommit | Adicionado fluxo tcipsflow-39I em 2ms
17:57:02,877 | INFO | FlowCommit | Adicionado fluxo tcipsflow-39O em 5ms
17:57:03,002 | INFO | FlowCommit | Adicionado fluxo tcipsflow-40I em 1ms
17:57:03,013 | INFO | FlowCommit | Adicionado fluxo tcipsflow-40O em 4ms
17:57:03,034 | INFO | FlowCommit | Adicionado fluxo tcipsflow-41I em 7ms
```

Fonte: Elaborado pelo autor.

Dada a verificação de fluxos não maliciosos, foram realizados testes fazendo uso da ferramenta Nmap de forma com que fossem originados fluxos de varredura de porta. Foram realizados testes individuais onde a cada chamada do comando Nmap é verificada uma porta de determinado *host*, e automáticos, onde o *software* Nmap envia vários fluxos para diferentes portas.

Um dos testes realizados consiste em verificar uma determinada porta em distintos *hosts* da rede, o que caracteriza a varredura horizontal. Para isso foi utilizado o comando "nmap -p <PORTA> <IP>"onde PORTA e IP são respectivamente, a porta a ser verificada e o endereço IP do *host* desejado. O resultado do comando pode ser visualizado na Figura 19, onde é realizada a varredura da porta 22 do *host* 10.0.0.31. Neste exemplo, pode-se observar que a porta 22 está aberta (*open*), isso significa que há um serviço disponível que pode ser explorado.

Figura 19 – Saída da execução do comando nmap para varredura individual de portas

```
ctatsch@host1:~\$ nmap -p22 10.0.0.32

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-06 17:57 BRT
Nmap scan report for 10.0.0.32
Host is up (0.00072s latency).
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.05 seconds
```

Fonte: Elaborado pelo autor.

O teste acima foi realizado para diferentes *hosts* de modo que produzisse uma varredura horizontal. O resultado esperado pode ser visualizado pelo *log* gerado pelo IPS desenvolvido e exibido na Figura 20. Nesta figura, as primeiras quatro linhas indicam a adição de fluxos por parte do controlador, na quinta houve a impressão do resultado do cálculo da análise que indica que três *hosts* distintos foram escaneados em apenas uma porta. Já na sexta linha, é impresso o resultado da análise.

Figura 20 – Registro de *log* criado com o recebimento de fluxos de varredura horizontal

```
17:57:49,447 | INFO | FlowCommit   | Adicionado fluxo tcipsflow-51I em 2ms
17:57:49,450 | INFO | FlowCommit   | Adicionado fluxo tcipsflow-51O em 5ms
17:57:50,445 | INFO | FlowCommit   | Adicionado fluxo tcipsflow-52I em 2ms
17:57:50,448 | INFO | FlowCommit   | Adicionado fluxo tcipsflow-52O em 5ms
17:57:51,941 | INFO | ScanAnalizer | Soma pesos = 5, Hosts = 3
17:57:51,941 | WARN | ScanAnalizer | IP 10.0.0.11/32 detectado como
   varredura horizontal
```

Fonte: Elaborado pelo autor.

O teste de varredura vertical de portas foi realizado através do comando "nmap -sT <IP>". Este comando realiza a varredura automática em diferentes portas de um determinado *host* de forma a detectar quais estão abertas. A saída deste comando pode ser visualizado na Figura 21, neste teste foi realizada a varredura de portas do servidor de endereço 10.0.0.31 que resultou na verificação de mil portas sendo que três destas (22, 53 e 80) estavam abertas.

Figura 21 – Saída da execução do comando nmap para varredura automática de portas

```
ctatsch@host1:~\$ nmap -sT 10.0.0.31

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-06 17:59 BRT
Nmap scan report for 10.0.0.31
Host is up (0.0088s latency).
Not shown: 997 closed ports
PORT   STATE SERVICE
22/tcp open  ssh
53/tcp open  domain
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 4.23 seconds
```

Fonte: Elaborado pelo autor.

Este teste permitiu que o atacante verificasse uma grande quantidade de portas pois o intervalo de varredura foi bem menor que o intervalo de coleta e análise, porém, se o indivíduo tentar novamente realizar um ataque não irá conseguir pois a soma dos pesos das portas ultrapassou o limite preestabelecido, fazendo com que o endereço IP do atacante fosse adicionado na lista de endereços para descarte. O registro de *log* do momento em que a varredura é detectada pode ser visualizada na Figura 22. Nesta, verifica-se na sétima linha, que a soma de pesos calculada na análise é muito superior a quinze, sendo então detectado o ataque e impresso seu tipo na oitava linha.

Figura 22 – Registro de *log* criado com o recebimento de fluxos de varredura vertical

```
17:59:21,969 | INFO  | FlowCommit   | Added flow tcipsflow-42I em 5ms
17:59:21,976 | INFO  | FlowCommit   | Added flow tcipsflow-42O em 12ms
17:59:22,969 | INFO  | FlowCommit   | Added flow tcipsflow-43I em 3ms
17:59:22,976 | INFO  | FlowCommit   | Added flow tcipsflow-43O em 10ms
17:59:23,968 | INFO  | FlowCommit   | Added flow tcipsflow-44I em 2ms
17:59:23,970 | INFO  | FlowCommit   | Added flow tcipsflow-44O em 4ms
17:59:27,896 | INFO  | ScanAnalizer | Soma pesos = 3004, Hosts = 1
17:59:27,896 | WARN  | ScanAnalizer | IP 10.0.0.12/32 detectado como
   varredura vertical
```

Fonte: Elaborado pelo autor.

Também foram realizados testes de varredura sem o pacote de inicialização SYN. No exemplo da Figura 23, o comando "nmap -sF <IP>"executa varredura através de exploração FIN. Como já abordado, quando um fluxo iniciar sem o pacote de inicialização SYN, o pacote é imediatamente descartado pelo controlador, por esta razão, o Nmap indicou que o *host* está inacessível. O *log* também pode ser visualizado na Figura 24 onde três pacotes foram descartados.

Figura 23 – Saída da execução do comando nmap para varredura FIN.

```
ctatsch@host1:~\$ nmap -sF 10.0.0.21

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-07 01:28 BRT
Note: Host seems down. If it is really up, but blocking our ping probes,
   try -Pn

Nmap done: 1 IP address (0 hosts up) scanned in 34.66 seconds
```

Fonte: Elaborado pelo autor.

Figura 24 – Registro de *log* criado com o recebimento de fluxos iniciados sem *flag* SYN

```
01:28:06,976 | WARN | PacketInHandler | Pacote descartado, sem flag SYN.
01:28:06,977 | WARN | PacketInHandler | Pacote descartado, sem flag SYN.
01:28:06,983 | WARN | PacketInHandler | Pacote descartado, sem flag SYN.
```

Fonte: Elaborado pelo autor.

Destaca-se que imagens de *log* presentes nesta seção correspondem apenas às últimas linhas do mesmo pois a leitura completa dos mesmos se tornaria repetitiva. Também é importante enfatizar que a lista de endereços para descarte foi zerada entre as diferentes técnicas de varredura, se a lista fosse mantida, no segundo teste não haveriam fluxos encaminhados, visto que já foram barrados no teste anterior.

Após os testes, foram comparadas as estatísticas armazenadas na base de dados em relação aos fluxos originados. A Figura 25 exemplifica os fluxos armazenados na base de dados

de determinado *host*. Nesta figura tem-se respectivamente, o IP de origem do fluxo, o IP de destino deste fluxo, a porta de origem, a porta de destino, o número de pacotes deste fluxo e uma indicação se a linha corresponde ao pedido ou à uma resposta da requisição que originou o fluxo. Neste exemplo pode se verificar que o *host* de IP 10.0.0.11 originou nove fluxos, cada fluxo possui duas linhas na figura, uma que indica o destino e outra para retorno da requisição. Através da coluna "Pacotes"observa-se um comportamento normal de acesso nos quatro primeiros fluxos gerados, representados em tons de cinza, pois o número de pacotes é superior a cinco. Nas linhas seguintes, em tons amarelados, observa-se que o número de pacotes dos respectivos fluxos é inferior a cinco, caracterizando uma varredura de porta. De acordo com as regras definidas neste trabalho, cada um dos fluxos de varredura possuem peso cinco, e como a soma destes alcança o limiar quinze, o IPS define estes fluxos como sendo um ataque, e deve ser descartado. Após a análise o atacante originou mais dois fluxos, que foram descartados, sendo armazenado na base de dados a quantia de um pacote por fluxo, correspondente ao pacote com *flag* SYN recebido pelo controlador, isto pode ser visualizado nos fluxos em tons avermelhados representados na figura.

Figura 25 – Exemplo de fluxos na base de dados.

| IP ORI | IP DST | PORTA ORI | PORTA DST | PACOTES | P/R |
|---|---|---|---|---|---|
| 10.0.0.11/32 | 10.0.0.21/32 | 33666 | 80 | 103 | PEDIDO |
| 10.0.0.21/32 | 10.0.0.11/32 | 8080 | 33666 | 96 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.21/32 | 33668 | 22 | 65 | PEDIDO |
| 10.0.0.21/32 | 10.0.0.11/32 | 22 | 33668 | 61 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.21/32 | 33670 | 110 | 68 | PEDIDO |
| 10.0.0.21/32 | 10.0.0.11/32 | 110 | 33670 | 62 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.21/32 | 45985 | 110 | 68 | PEDIDO |
| 10.0.0.21/32 | 10.0.0.11/32 | 110 | 45985 | 61 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.31/32 | 56974 | 23 | 2 | PEDIDO |
| 10.0.0.31/32 | 10.0.0.11/32 | 8080 | 56974 | 1 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.31/32 | 26261 | 22 | 1 | PEDIDO |
| 10.0.0.31/32 | 10.0.0.11/32 | 22 | 26261 | 1 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.31/32 | 56974 | 25 | 2 | PEDIDO |
| 10.0.0.31/32 | 10.0.0.11/32 | 25 | 56974 | 1 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.31/32 | 48906 | 110 | 1 | PEDIDO |
| 10.0.0.31/32 | 10.0.0.11/32 | 110 | 48906 | 0 | RESPOSTA |
| 10.0.0.11/32 | 10.0.0.31/32 | 27695 | 110 | 1 | PEDIDO |
| 10.0.0.31/32 | 10.0.0.11/32 | 110 | 27695 | 0 | RESPOSTA |

Fonte: Elaborado pelo autor.

O Quadro 4 apresenta uma análise quantitativa dos fluxos. Em sua segunda coluna são exemplificados os fluxos não maliciosos. Do total de 200.000 fluxos gerados, todos foram encaminhados, não apresentando falsos negativos. A terceira coluna apresenta os resultados da varredura horizontal, onde o comando "nmap -p"foi utilizado com a mesma porta para diferentes *hosts*. Sendo assim, cada um dos quatro computadores tentou varrer os cinco servidores

disponíveis o que originou vinte fluxos.

Como a validação é realizada após a varredura de três *hosts* distintos, a quantidade máxima de fluxos maliciosos a ser encaminhado esperada é de 12 fluxos, três por computador (*host* origem). Contudo, a análise resultou em uma quantidade de fluxos igual a 14. Isto ocorreu devido a retransmissão de alguns pacotes, aumentando a quantidade de pacotes por fluxo, e assim, sendo identificados como não maliciosos. Estes fluxos passaram a ser considerados maliciosos apenas no quarto ou no quinto *host* verificado, quando o número de pacotes não passou os cinco estabelecidos. Estes valores podem ser visualizados na terceira coluna do Quadro 4.

Na quarta coluna, que representa a execução de varredura vertical, foram gerados mil fluxos maliciosos, essa geração, por ter sido automática, foi consideravelmente mais rápida do que o intervalo de coleta, o que acarretou no encaminhamento de 658 fluxos maliciosos. No momento em que o *software* terminou a análise e os endereços foram adicionados na lista de endereços para descarte, o sistema passou a descartar os 342 pacotes restantes. Além disso, essa varredura também teve alguns pacotes duplicados, fazendo com que doze fluxos que deveriam ser considerados maliciosos, fossem encaminhados como fluxo não malicioso.

Na última coluna são apresentados os números referentes à varredura por exploração FIN. Como citado anteriormente, fluxos não iniciados com pacote SYN são imediatamente descartados, isto pode ser comprovado através dos números apresentados. Todos os mil fluxos originados foram descartados.

Quadro 4 – Resultados das análises de varreduras.

| | Não maliciosos | TCP Con. Hor. | TCP Con. Ver. | Expl. FIN |
|---|---|---|---|---|
| Fluxos Gerados | 200000 | 20 | 1000 | 1000 |
| Fluxos Encaminhados | 200000 | 14 | 658 | 0 |
| Fluxos Bloqueados | 0 | 8 | 342 | 1000 |
| Falsos Positivos | 0 | 0 | 0 | 0 |
| Falsos Negativos | 0 | 2 | 12 | 0 |

Fonte: Elaborado pelo autor.

O mesmo procedimento acima realizado foi novamente executado a fim de verificar se de fato todos os fluxos foram adicionados na lista de endereços para descarte. Neste procedimento a lista de endereços para descarte não foi limpa e o resultado encontra-se no Quadro 5. Neste pode-se observar que todos os fluxos gerados, inclusive os não maliciosos, foram descartados. Isto é resultado da análise das varreduras anteriores onde todos os *hosts* efetuaram ataques de varredura e foram adicionados na lista para descarte.

Quadro 5 – Resultados das análises de varreduras após estabelecido o descarte.

| | Não maliciosos | TCP Con. Hor. | TCP Con. Ver. | Expl. FIN |
|---|---|---|---|---|
| Fluxos Gerados | 200000 | 20 | 1000 | 1000 |
| Fluxos Encaminhados | 0 | 0 | 0 | 0 |
| Fluxos Bloqueados | 200000 | 20 | 1000 | 1000 |
| Falsos Positivos | 0 | 0 | 0 | 0 |
| Falsos Negativos | 0 | 0 | 0 | 0 |

Fonte: Elaborado pelo autor.

Através dos testes realizados foram obtidos os seguintes resultados:

- Tempo médio de atraso por fluxo de 4 milissegundos. Decorridos pela necessidade da verificação da lista de endereços para descarte e pela adição de regras no controlador;
- Nenhum falso positivo. Se deve principalmente pelo fato de não ter sido realizada nenhuma predição para a detecção e sim a validação de fluxos através da quantidade de pacotes;
- 0,034% de falsos negativos. Decorridos devido a quantidade de pacotes retransmitidos. Sendo assim, o que deveria ser um ataque acabou sendo considerado um fluxo normal;
- Grande quantidade de varreduras realizadas até que seu bloqueio fosse efetuado. Apesar do algoritmo de detecção ser assertivo na detecção de varreduras, o intervalo entre uma análise e outra permitiu que o atacante pudesse realizar uma grande quantidade de varreduras.

Não foi possível estabelecer um comparativo de desempenho com as soluções apresentadas no Capítulo 3. Isso decorre em virtude dos projetos não apresentarem resultados de suas análises e/ou suas análises não contemplam ataques de varredura de porta. Cabe salientar que os projetos apresentados que possuem detecção de varredura de porta, não possuem proteção contra os mesmos.

# 6 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo geral desse trabalho foi apresentar uma arquitetura não intrusiva que permita, de maneira eficaz, coletar estatísticas de fluxo em redes SDN, verificar a ocorrência de fluxos de varredura de porta e então, incrementar 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.

As simulações realizadas produziram resultados coerentes com o esperado. Foi possível comprovar a eficácia na detecção de fluxos maliciosos através de estatísticas, que resultaram na ausência de falsos positivos e em um número baixíssimo de falsos negativos. O sistema desenvolvido também apresentou suas limitações, que são: a necessidade de inserção de regras distintas nos *switches*; e o intervalo de coleta superior ao de varreduras, permitindo que o atacante efetuasse uma série de varreduras antes de ser detectado.

Um ponto a ser considerado é o fato de existirem, na literatura, poucos trabalhos relacionados à prevenção de ataques de varredura de porta para SDN. Além disso, os projetos disponíveis que fazem detecção de varredura de porta, não possuem proteção contra os mesmos, ou utilizam tráfego para análise, o que torna este trabalho uma alternativa relevante.

Considerando que o presente trabalho descreveu a prevenção de ataques de varredura de porta sobre ambientes simulados, e as limitações que o sistema desenvolvido possui, como possíveis trabalhos futuros pode-se sugerir:

- a análise de testes utilizando tráfego de dados legítimo, o que pode ser obtido através de testes em redes de experimentação (*testbeds*).
- a análise de protocolos adotados para a implementação de Redes Definidas por Software a fim de obter e analisar estatísticas em um intervalo menor sem sobrecarregar a rede;
- a análise das *flags* dos pacotes recebidos a fim de eliminar pacotes duplicados e/ou retransmitidos, reduzindo o número de falsos negativos gerados; e
- estender a utilização dos contadores/estatísticas para desenvolver métodos de detecção e/ou prevenção de outros tipos de ataques.

# 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.

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.

CLAISE, B. *Cisco Systems NetFlow Services Export Version 9*. RFC Editor, out. 2004. 2004. RFC 3954. (Request for Comments, 3954). Disponível em: <https://rfc-editor.org/rfc/rfc3954.txt>.

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

COMER, D. E. *Internetworking with TCP/IP*. 6th. ed. [S.l.]: Addison-Wesley Professional, 2013.

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.

HEBERLEIN, L. T. *Statistical Problems with Statistical-based Intrusion Detection*. 2007.

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.

KOLPYAKWAR, A. A.; INGLE, M. G.; DESHMUKH, R. V. A survey on data mining approaches for network intrusion detection system. *International Journal of Computer Applications*, Foundation of Computer Science, v. 159, n. 1, 2017.

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.

LI, W.; MENG, W.; KWOK, L. F. A survey on openflow-based software defined networks: Security challenges and countermeasures. *Journal of Network and Computer Applications*, Elsevier, v. 68, p. 126–139, 2016.

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. F. *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. USA: Insecure, 2009.

MARK, L. *et al. IP Flow Information Export (IPFIX) Implementation Guidelines*. RFC Editor, abr. 2008. 2008. RFC 5153. (Request for Comments, 5153). Disponível em: <https://rfc-editor.org/rfc/rfc5153.txt>.

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.

NOBRE, J. Ameaças e ataques aos sistemas de informação: Prevenir e antecipar. Editora FOA, Centro Universitário de Volta Redonda (UniFOA), Volta Redonda, RJ, dez. 2007.

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.

PANCHEN, S.; MCKEE, N.; PHAAL, P. *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks*. RFC Editor, set. 2001. 2001. RFC 3176. (Request for Comments, 3176). Disponível em: <https://rfc-editor.org/rfc/rfc3176.txt>.

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.

PFAFF, B. *et al.* Extending networking into the virtualization layer. In: *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*. [S.l.: s.n.], 2009.

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.

POSTEL, J.; REYNOLDS, J. *Telnet Protocol Specification*. IETF, maio 1983. 1983. RFC 854 (Standard). (Request for Comments, 854). Updated by RFC 5198. Disponível em: <http://www.ietf.org/rfc/rfc854.txt>.

PYTHON SOFTWARE FOUNDATION. *Python*. 2017. Disponível em: <https://www.python.org/>. Acesso em: 29 apr. 2017.

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.

THE LINUX FOUNDATION. *Open vSwitch Project*. jun. 2017. 2017. Disponível em: <https://http://openvswitch.org>. Acesso em: 10 jun. 2017.

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.