RENAN VILLELA OLIVEIRA

DETECÇÃO DE ATAQUES DE DDoS AO PLANO DE CONTROLE DE SDN UTILIZANDO APRENDIZADO DE MÁQUINA

LAVRAS – MG

2023

RENAN VILLELA OLIVEIRA

DETECÇÃO DE ATAQUES DE DDoS AO PLANO DE CONTROLE DE SDN UTILIZANDO APRENDIZADO DE MÁQUINA

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Sistemas de Computação, para a obtenção do título de Mestre.

Prof. DSc. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG
2023

Ficha catalográfica elaborada pelo Sistema de Geração de Ficha Catalográfica da Biblioteca Universitária da UFLA, com dados informados pelo(a) próprio(a) autor(a).

Oliveira, Renan Villela

Detecção de Ataques de DDoS ao Plano de Controle de SDN utilizando Aprendizado de Máquina / Renan Villela Oliveira. – Lavras : UFLA, 2023.

59 p. : il.

Dissertação (Mestrado acadêmico) – Universidade Federal de Lavras, 2023.

Orientador: Prof. DSc. Luiz Henrique Andrade Correia.

Bibliografia.

1. SDN. 2. DDoS. 3. Aprendizado de máquina. I. Correia, Luiz Henrique Andrade. II. Título.

RENAN VILLELA OLIVEIRA

DETECÇÃO DE ATAQUES DE DDoS AO PLANO DE CONTROLE DE SDN UTILIZANDO APRENDIZADO DE MÁQUINA

DETECTION OF DDoS ATTACKS ON SDN CONTROL PLANE USING MACHINE LEARNING

Dissertação apresentada à Universidade Federal de Lavras, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, área de concentração em Sistemas de Computação, para a obtenção do título de Mestre.

APROVADA em 16 de Fevereiro de 2023.

Prof. DSc. Luiz Henrique Andrade Correia	UFLA
Prof. DSc. Renata Lopes Rosa	UFLA
Prof. DSc. Neumar Costa Malheiros	UFLA
Prof. DSc. Fabricio Aguiar Silva	UFV

Prof. DSc. Luiz Henrique Andrade Correia
Orientador

LAVRAS – MG
2023

AGRADECIMENTOS

Agradeço à Universidade Federal de Lavras, especialmente ao Programa de Pós-Graduação em Ciência da Computação, pela oportunidade. Também à CAPES, CNPq e FAPEMIG, pelo fomento à pesquisa no Brasil e Minas Gerais.

RESUMO

O paradigma de *Software Defined Networking* (SDN) é considerado promissor para a inovação das tecnologias de redes de computadores. A arquitetura SDN separa o plano de dados do plano de controle, onde o controlador tem uma visão geral da rede. Segurança em redes é um assunto em constante discussão, pois diariamente surgem novas formas de ataques com objetivos diversos. Em SDN não é diferente, muitos ataques de *Distributed Denial of Service* (DDoS) são realizados ao plano de controle de SDN, portanto, medidas de proteção devem ser desenvolvidas para detectar atividades maliciosas na rede. Apesar das redes SDN fornecerem forte controle sobre o tráfego, também oferecem novos problemas e desafios, pois, por exemplo, um ataque de DDoS a um controlador tem potencial para deixar toda a rede inoperante. Para identificar tráfego malicioso em SDN, neste trabalho, foram analisados e classificados os fluxos de entrada para detectar ataques de DDoS através de técnicas de aprendizado de máquina. Para identificar características cruciais no monitoramento de uma SDN, foram criados *datasets* a partir da captura de tráfego legítimo e malicioso (DDoS) em SDN. Estes *datasets* foram utilizados na construção de modelos de aprendizado de máquina que, por sua vez, foram usados para classificar os fluxos em legítimos ou maliciosos. Os experimentos de classificação de tráfego foram divididos em dois cenários, um com tráfego variável durante o tempo do experimento e outro com tráfego imutável para cada iteração. Os resultados obtidos mostraram que o algoritmo *Naïve Bayes* foi mais assertivo na identificação dos ataques que os demais algoritmos (*Gradient Boosting*, *Decision Tree* e *Support Vector Machine*). Para avaliar os resultados, foram utilizadas as métricas *accuracy*, *precision*, *recall* e *F-score*.

Palavras-chave: SDN. Segurança. Controlador. DDoS. Aprendizado de máquina.

ABSTRACT

The Software Defined Networking (SDN) paradigm is considered promising for the innovation of computer networking technologies. The SDN architecture separates the data plane from the control plane, where the controller has an overall view of the network. Network security is a subject under constant discussion, as new forms of attacks with different objectives appear daily. SDN is no different, many Distributed Denial of Service (DDoS) attacks are performed against the SDN control plane, therefore, protection measures must be developed to detect malicious activities on the network. While SDN networks provide strong control over traffic, they also offer new problems and challenges as, for example, a DDoS attack against a controller has the potential to let the entire network inoperable. In order to identify malicious traffic in SDN, in this work, the input flows were analyzed and classified to detect DDoS attacks through machine learning techniques. In order to identify crucial characteristics in the monitoring of a SDN, datasets were created from the capture of legitimate and malicious traffic (DDoS) in SDN. These datasets were used in the construction of machine learning models which, in turn, were used to classify flows as legitimate or malicious. The traffic classification experiments were divided into two scenarios, one with variable traffic during the experiment time and another with unchanging traffic for each iteration. The results showed that the Naïve Bayes algorithm was more assertive in identifying attacks than the other algorithms (Gradient Boosting, Decision Tree and Support Vector Machine). To evaluate the results, the metrics accuracy, precision, recall and F-score were used.

Keywords: SDN. Security. Controller. DDoS. Machine learning.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Em um mundo cada vez mais digital, onde os dados tendem a ser valorizados, é imprescindível que eles sejam operados e movimentados de maneira segura e íntegra. Por se tratarem de ativos com alto potencial de lucro para corporações ou conter informações sigilosas das mais variadas espécies, inclusive de privacidade pessoal de usuários, as tecnologias envolvidas nos processos de manipulação de dados precisam ser constantemente evoluídas. Redes de computadores são os principais meios de comunicação onde trafegam os dados, e assim sendo, se tornam alvos de potenciais intrusões, ataques e manipulações maliciosas visando prejudicar o fluxo legítimo de dados.

Em um relatório publicado pela empresa NSFOCUS (NSFOCUS, 2020), onde foram apresentados dados sobre a ocorrência de ataques DDoS (*Distributed Denial of Service*) no primeiro semestre do ano de 2020, observou-se um aumento do número de ataques relacionado à decorrência da pandemia de COVID-19. O Brasil ficou classificado entre os principais alvos de ataques, ocupando a quinta colocação global em número de ataques e a quarta colocação em volume de tráfego oriundo de DDoS. Em outro relatório da mesma empresa, durante o primeiro semestre de 2022, foram observados que os protocolos UDP e TCP são os mais explorados nos ataques, seguidos pelo ICMP. Este relatório observou que os ataques com tráfego acima de 1 Tbps estão se tornando mais frequentes (NSFOCUS, 2022).

O conceito de *Software Defined Networking* (SDN) tem sido apontado como promissor para descalcificar as tecnologias de rede e obter-se avanços expressivos na área. O paradigma SDN separa os planos de controle e dados, assim todo o controle lógico da rede, antes alocado nos dispositivos de *hardware*, é migrado para um componente centralizado logicamente, conhecido como controlador de rede (CHICA; IMBACHI; BOTERO, 2020). É possível criar regras de fluxo para que o controlador as envie às tabelas de fluxo dos dispositivos, determinando como lidar com os fluxos que trafegam na rede.

Os autores Feamster, Rexford e Zegura (2014) apontam como foi a evolução da busca por tornar as redes mais programáveis, percorrendo um caminho até chegar nas SDN, dividindo a história em três partes. A primeira parte sendo a ideia de redes ativas (de metade dos anos 1990 até o começo dos anos 2000) que introduziram funções programáveis na rede, com foco na programabilidade do plano de dados. A segunda parte refere-se à ideia de separação dos planos de controle e de dados (por volta de 2001 a 2007), desencadeando o desenvolvimento de interfaces abertas entre os planos. E a terceira, seria o desenvolvimento de sistema operacional de rede e do protocolo *OpenFlow* (de 2007 a 2010 aproximadamente), que permitiu que a separação entre os planos fosse escalável e prática. Também é destacada a importância da virtualização de redes na evolução da SDN. Apesar de conceitos diferentes e independentes entre si, virtualização de redes e SDN podem se cruzar e gerar benefícios em ambas as partes.

O protocolo *OpenFlow*, desenvolvido pela *Open Network Foundation*, é o primeiro padrão de interface de comunicação definido entre as camadas de controle e encaminhamento da arquitetura SDN. *OpenFlow* usa o conceito de fluxo para identificar o tráfego de rede e armazenar suas informações em uma tabela de fluxos (TANG et al., 2016).

Lado a lado com redes de computadores, caminha a área de segurança da informação, pois é preciso obter confiabilidade nos dados trafegados investindo em métodos que tornem seu transporte seguro na rede. Apesar das SDN fornecerem forte controle sobre o fluxo de tráfego, também oferecem novos problemas e desafios (CARVALHO et al., 2018).

Como qualquer nova tecnologia, SDN tem suas vantagens e desvantagens. Na área de segurança, por exemplo, pode alavancar o tratamento ou mitigação de várias ameaças à rede. Infelizmente, no entanto, SDN também traz novas vulnerabilidades que são inerentes à sua arquitetura (CHICA; IMBACHI; BOTERO, 2020).

Ataques de DDoS têm como finalidade tornar um sistema lento ou inoperante através do esgotamento de seus recursos. São ameaças muito comuns atualmente tanto em redes de computadores tradicionais quanto em SDN. Há diversos tipos desses ataques, que podem afetar o funcionamento da rede, tanto por meio do plano de controle, quanto pelo plano de dados. Ataques DDoS podem ser baseados nos protocolos TCP, como o SYN-ACK Flood, em que os atacantes inundam o alvo com pacotes SYN-ACK falsificados. Ao tentar responder essas requisições maliciosas, o alvo pode esgotar seus recursos (memória, processamento, entre outros), se tornando indisponível para as requisições legítimas. Outros protocolos usados como base para DDoS são HTTP, UDP e ICMP (RIOREY, 2015).

Um relatório em um *website* (SECURITY, 2020) reportou que houve 4,83 milhões de ataques DDoS globalmente na primeira metade do ano de 2020, com aumento de 25% de aumento da frequência de ataques entre março e junho.

1.1 Definição do problema

Assim como as tecnologias de rede evoluem, também avançam as técnicas maliciosas visando roubar dados, causar prejuízos ou tornar sistemas inoperantes. Por isso é imprescindível haver estudos de segurança da informação acompanhando a evolução das tecnologias aplicadas.

Há diversos trabalhos na literatura que propõem técnicas e sistemas de detecção de intrusão em SDN. Entretanto, não há segurança plena quando tratamos de sistemas e dados. Sempre há várias vulnerabilidades conhecidas ou que nem sequer foram descobertas, mas possuem grande potencial de impacto negativo. Mesmo que diversos pesquisadores tenham proposto soluções diferentes, ataques DDoS continuam causando danos, tanto em redes tradicionais quanto em SDN.

Dessa forma, o problema pode ser enunciado como: "Dado que muitos ataques DDoS são realizados nas SDN e em seus respectivos controladores, é possível analisar os fluxos de entrada e detectar tráfego malicioso?"

Se o controlador de uma rede definida por *software* sofrer ataques e não receber tratamento desse problema, toda a rede será impactada com a negação de seus serviços. Como consequências, poderá haver saturação do controlador e da banda, além dos outros dispositivos arriscarem ter seus recursos esgotados ao tentar lidar com o tráfego de pacotes oriundos dos ataques, somados aos pacotes de usuários legítimos. Assim, aplicações poderão parar de funcionar e os usuários serão prejudicados. Dessa forma, é importante trabalhar na detecção de ataques ao plano de controle de uma SDN.

1.2 Objetivos

Este trabalho propõe o estudo de técnicas que contribuam de forma eficiente na detecção de ataques de DDoS através da análise de anomalias no tráfego de dados nas redes SDN. Segundo Aladaileh et al. (2020), ataques de DDoS apresentam uma séria ameaça à qualidade das redes SDN, especialmente se afetar um controlador, que se torna um ponto único de falha, o que pode afetar diretamente o desempenho, a disponibilidade e a confiabilidade da rede.

O objetivo deste trabalho é a classificação dos fluxos de uma SDN em legítimo ou malicioso para a detecção e redução de ataques de DDoS. Os objetivos específicos são listados a seguir:

a) Gerar tráfego legítimo e malicioso em uma SDN;
b) Capturar fluxos na SDN e criar *datasets* contendo dados de tráfego legítimo e malicioso para treinar os algoritmos de aprendizado de máquina;
c) Desenvolver modelos de aprendizado de máquina a partir dos *datasets* criados;
d) Realizar experimentos de ataques de DDoS visando afetar o plano de controle da rede;
e) Detectar ataques por meio da classificação de fluxos em legítimos ou maliciosos pelos modelos de aprendizado de máquina;
f) Avaliar os modelos de classificação.

1.3 Solução

Quanto mais estudos visando aprimorar a proteção aos dados, maior confiabilidade os sistemas terão a fornecer aos seus usuários. Neste estudo, o uso de *datasets* de tráfego de rede para treinar algoritmos de aprendizado de máquina auxiliou na identificação de tráfegos anômalos e potenciais ataques.

A partir da análise do tráfego por um módulo de aprendizado de máquina, os fluxos foram classificados em maliciosos ou legítimos.

Os *datasets* foram criados a partir de rodadas de experimentos gerando e capturando tráfego legítimo e malicioso, para serem usados como base de dados para os algoritmos de aprendizado de máquina aprenderem a identificar e classificar o tráfego na rede.

Após a construção dos modelos de aprendizado de máquina, foram executados vários experimentos, com configurações diversas de tráfego gerado e tempo de execução, enquanto o módulo de aprendizado de máquina recebia os fluxos e os classificava conforme foram treinados a partir dos *datasets*.

1.4 Organização do Trabalho

A seguir é mostrada a organização desta dissertação. O Capítulo 2 apresenta o referencial teórico, abordando os principais conceitos nos quais se baseia o trabalho. O Capítulo 3 sintetiza alguns trabalhos relacionados a esta pesquisa, contribuindo para o desenvolvimento da mesma. No Capítulo 4, é descrita a metodologia deste projeto. No Capítulo 5 são detalhados os cenários e experimentos realizados. No Capítulo 6, são descritos os resultados obtidos ao final dos experimentos. Por fim, no Capítulo 6 são apresentadas as conclusões desta pesquisa.

2 REFERENCIAL TEÓRICO

Nesse capítulo, serão descritos os principais conceitos utilizados no trabalho, para o entendimento das tecnologias e técnicas utilizadas para compor o mesmo.

2.1 Redes Definidas por Software

SDN (*Software Defined Networking*) é um novo paradigma, surgido da ideia de redes programáveis, em que o *hardware* de encaminhamento é desacoplado das decisões de controle da rede, simplificando seu gerenciamento e permitindo inovações e evoluções (NUNES et al., 2014). Em uma SDN há um controlador logicamente centralizado que possui uma visão extensa da rede e controla vários dispositivos, como *switches* e roteadores, que realizam encaminhamento de pacotes (FARHADY; LEE; NAKAO, 2015). Em redes tradicionais, o gerenciamento é feito dispositivo a dispositivo, pois o plano de controle e o plano de dados são acoplados. SDN desacopla os planos e facilita o controle da rede por meio de um ponto único: o controlador.

As SDN trazem alguns benefícios de uso em comparação a redes tradicionais, mas também há desafios a enfrentar para sua consolidação. Xia et al. (2015) observam três benefícios, sendo: configuração melhorada devido à centralização no controlador; desempenho aprimorado também devido ao controle central e uma visão ampla da rede por meio do mesmo; e inovação encorajada, devido à programabilidade. No entanto, também são observados alguns desafios, como a padronização dos elementos para serem desenvolvidas aplicações coerentes seguindo um modelo. A centralização do controle facilita o gerenciamento da rede, mas pode se tornar um ponto único de falha operacional e de segurança.

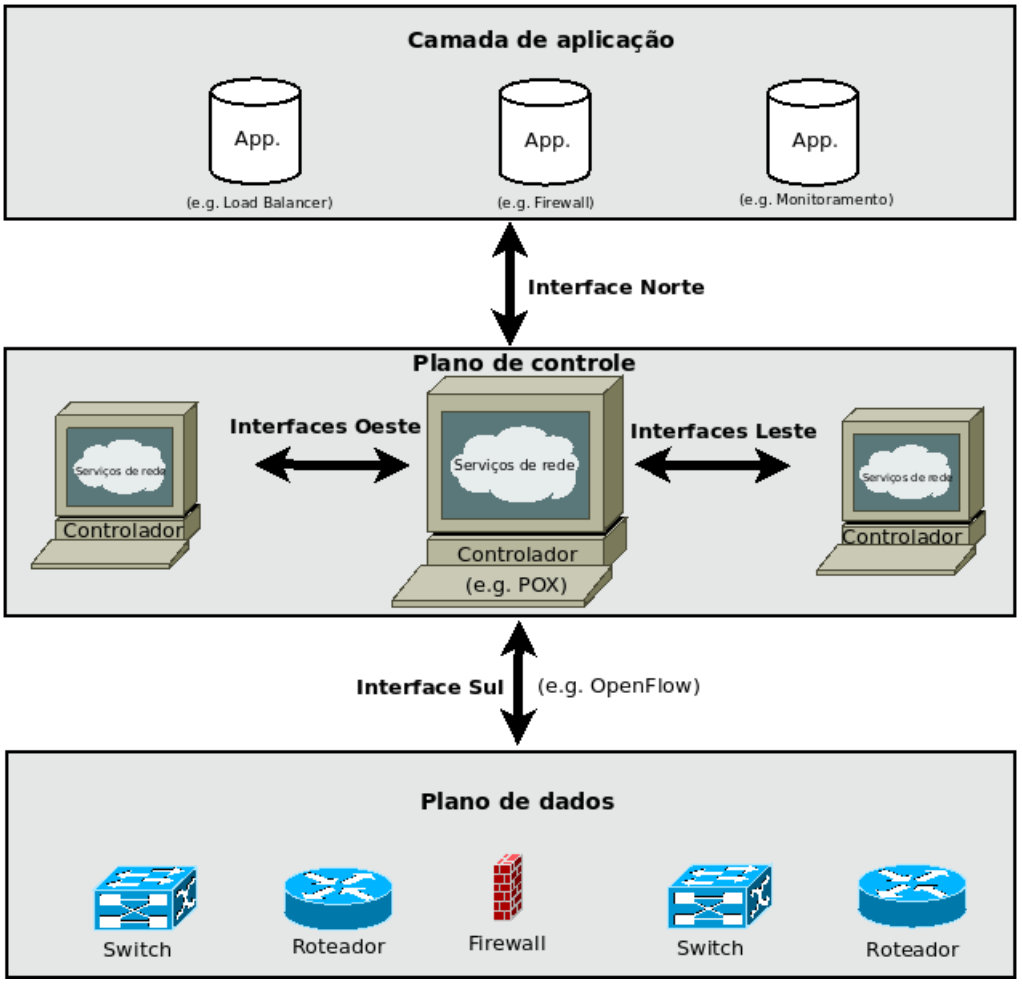
2.1.1 Arquitetura SDN

Chica, Imbachi e Botero (2020) mostram que a arquitetura de uma SDN é definida contendo três camadas e duas interfaces:

a) A camada de aplicação, onde são implementadas várias aplicações personalizadas e de terceiros, como aplicações de rede e segurança;
b) Interface norte, que faz uma ligação entre a camada de aplicação e a de controle;
c) A camada ou plano de controle, onde fica a cadeia de serviços e o controlador da rede;
d) Interface sul, o protocolo (e.g. *OpenFlow*) de comunicação entre o controlador e os dispositivos da rede;
e) A camada de infraestrutura ou plano de dados, onde ficam os dispositivos da rede como *switches*, roteadores, servidores.

Os autores Singh e Behal (2020) apresentam as interfaces leste e oeste, quando há mais de um controlador operando na rede, fazendo a ponte entre os mesmos. A Figura 2.1 ilustra a arquitetura de uma rede SDN e seus componentes, como descritos anteriormente.

Figura 2.1 – Arquitetura SDN



Fonte: adaptado de Singh e Behal (2020).

2.2 Protocolo *OpenFlow*

OpenFlow é a primeira interface de comunicação padrão definida entre as camadas de controle e encaminhamento (plano de dados) de uma arquitetura SDN. O protocolo é implementado nas duas pontas da interface sul de uma SDN, conectando o controlador aos dispositivos de rede do plano de dados. O *OpenFlow* usa o conceito de fluxos (sequência de pacotes entre a origem e o destino, com informações como prioridade, contadores, instruções de processamento, entre outras) para identificar o tráfego de rede com base em regras de correspondência predefinidas que podem ser programadas estática ou dinamicamente pelo controlador. Também permite que seja definido como o tráfego deve fluir pelos

dispositivos de rede com base em parâmetros como padrões de uso, aplicativos e recursos de nuvem (FOUNDATION, 2012). Algumas vantagens apontadas para o uso de SDN baseada em *OpenFlow* incluem o controle centralizado de ambientes de vários fornecedores, a complexidade reduzida por meio da automação, maior confiabilidade da rede, entre outros.

O *OpenFlow* utiliza do fato em que a maioria dos *switches* e roteadores *Ethernet* modernos contêm tabelas de fluxo para funções essenciais de rede, como roteamento, divisão de sub-redes, *firewall* e análise estatística de tráfego de dados (XIA et al., 2015).

Em uma SDN usando *OpenFlow*, o dispositivo de encaminhamento, ou *switch OpenFlow*, contém uma ou mais tabelas de fluxos e uma camada de abstração que se comunica com segurança com um controlador por meio do protocolo. As tabelas de fluxo são feitas de entradas de fluxo, cada uma determinando como pacotes pertencentes a um fluxo serão processados e encaminhados (NUNES et al., 2014).

2.3 Controladores SDN

Um controlador funciona como o cérebro da arquitetura SDN, realizando tarefas de decisão de controle e concentrando toda a lógica de funcionamento da rede. SDNs podem recorrer a duas categorias de controlador: centralizado e distribuído. Controladores centralizados concentram toda a lógica do plano de controle em um único local, oferecendo facilidade de gerenciamento, mas sofrem com escalabilidade. Os distribuídos têm vantagem de escalabilidade e alto desempenho durante aumento da demanda de requisições (PALIWAL; SHRIMANKAR; TEMBHURNE, 2018).

Há diversos controladores disponíveis para SDN. Neste trabalho, será utilizado o POX (POX, 2021a). O Quadro 2.1 mostra alguns dos principais controladores que utilizam o protocolo *Openflow*, assim como seus respectivos desenvolvedores, linguagens de programação usadas em suas implementações, e se são de código aberto.

Quadro 2.1 – Controladores SDN

Controlador	Implementação	Código aberto?	Desenvolvedor
POX	Python	Sim	Nicira
NOX	C++/Python	Sim	Nicira
Floodlight	Java	Sim	Big Switch Networks
ONOS	Java	Sim	Open Networking Lab
Ryu	Python	Sim	NTT Communications

Fonte: adaptado de Nunes et al. (2014).

2.3.1 POX

O POX é um controlador *OpenFlow* (atualmente, versão 1.0) centralizado, de código aberto e escrito em *Python*, para uso geral. Enquanto outros controladores são mais usados em redes de produção, o POX é muito popular para prototipação e pesquisa, devido à sua estrutura simples, visando prover um ambiente eficiente e fácil para experimentação e testes (NOMAN; JASIM, 2020). Possui documentação online, lista de e-mails da comunidade e pode ser obtido de forma fácil. Essas características foram motivadoras para a escolha do POX nessa pesquisa. Outro ponto relevante, é que o POX tem uma ótima integração com o Mininet[1] que, inclusive, distribui suas VMs oficiais já com o controlador instalado, tornando-o popular em pesquisas.

O POX possui vários componentes prontos, que podem fornecer funcionalidades, recursos convenientes ou apenas exemplos (POX, 2021b). Alguns exemplos de componentes são listados abaixo:

a) **py** - POX inicia um interpretador Python interativo;
b) **forwarding.hub** - instala regras de inundação em cada *switch*, como se esses fossem hubs;
c) **forwarding.l2_learning** - este componente faz com que os *switches* OpenFlow atuem como um tipo de switch de aprendizado L2;
d) **forwarding.l3_learning** - este componente faz com que os *switches* OpenFlow atuem como uma espécie de *switch* de camada 3;
e) **openflow.discovery** - envia mensagens dos *switches* OpenFlow para descobrir a topologia da rede;
f) **log** - o POX usa o sistema de *logging* do Python, então este componente permite configurar diversos *logs* pela linha de comandos.

Além de componentes como os citados anteriormente, que são módulos *Python*, é possível criar outros componentes para o POX, através da linguagem de programação. O controlador também possui várias APIs para auxiliar no desenvolvimento de aplicações de controle de rede. A API **pox.lib.addresses** permite trabalhar com endereços IPv4 e IPv6, por exemplo. Já a biblioteca **pox.lib.packet** permite trabalhar com pacotes TCP, UDP, ICMP, dentre outros. Um dos principais propósitos do POX é ser usado para desenvolver aplicações de controle *OpenFlow*, há vários recursos para trabalhar com o protocolo, como o objeto **nexus**. Este último, é um gerenciador para um conjunto de conexões *OpenFlow*.

[1] Mininet é um *software* livre que emula componentes de redes SDN, capaz de simular redes desse tipo. Na Seção 2.4 o Mininet é abordado mais detalhadamente. Mais informações em <http://mininet.org/>.

2.4 Mininet

O Mininet é um *software* emulador de redes de código aberto que cria uma rede de *hosts* virtuais, *switches OpenFlow*, controladores e links. O Mininet fornece uma maneira fácil de obter o comportamento correto do sistema e de experimentar topologias. As redes Mininet executam código real, incluindo aplicativos de rede Unix/Linux padrão, bem como o *kernel* Linux real e pilha de rede. Dessa forma, um código desenvolvido e testado no Mininet, para um controlador *OpenFlow*, *switch* modificado ou *host*, pode ser movido para um sistema real com mudanças mínimas, para teste do mundo real, avaliação de desempenho e implantação (CONTRIBUTORS, 2021). Há algumas topologias padrões (e.g. *minimal*, *linear*, *tree*) disponíveis para simular redes no Mininet, além da possibilidade de gerar topologias customizadas (KAUR; SINGH; GHUMMAN, 2014). É possível obter VMs oficiais do Mininet, que já vêm com o controlador POX instalado, facilitando o uso integrado de ambos. Apesar de ter sido cogitado o uso do Mininet nesta pesquisa, os experimentos foram realizados em ambiente real, devido à possibilidade de uso em laboratório de um *switch* físico compatível com *OpenFlow* e SDN.

2.5 Segurança em SDN

Assim como a segurança é imprescindível nas redes de computadores tradicionais, sendo necessária uma contínua evolução para lidar com o desenvolvimento de novas ameaças, o paradigma das SDNs necessita de pesquisas em segurança. Por ser uma tecnologia emergente, SDN traz melhorias em segurança, mas também potenciais novas vulnerabilidades e vetores de ameaça às redes deste tipo.

Kreutz, Ramos e Verissimo (2013) descrevem sete principais vetores de ameaça identificadas em SDN:

a) Fluxos de tráfego falsos ou forjados, que podem ser usados como portas para atacar *switches* e controladores;
b) Ataques em vulnerabilidades dos *switches*;
c) Ataques nas comunicações do plano de controle, que podem ser usados para gerar ataques de DoS ou roubo de dados;
d) Ataques e vulnerabilidades nos controladores, que pode ser o vetor mais grave, pois ao afetar o controlador, o atacante pode comprometer toda a rede;
e) Falta de mecanismos para garantir confiança entre o controlador e aplicações de gerenciamento, pois aplicações maliciosas podem ser desenvolvidas e implantadas no controlador;
f) Vulnerabilidades e ataques em estações administrativas, que podem ser usadas para acessar o controlador SDN;

g) E a falta de recursos confiáveis para atividades forenses e remediação, que ajudariam a entender a causa de problemas e encontrar soluções mais rapidamente.

Benabbou, Elbaamrani e Idboufker (2018) apontam como risco, nos planos de controle e de dados, a incapacidade do controlador e outros dispositivos da rede de lidar rapidamente com falhas de *link* ou dispositivos, o que pode resultar indisponibilidade ou ruptura dos serviços. Ataques DDoS são citados como vetores para este risco. Apontam, também, problemas em soluções de proteção contra ataques DDoS, como uma negação de serviço ocasionada pela própria solução, devido à sobrecarga durante a coleta de dados estatísticos. Também citam problemas de precisão ao detectar atividades maliciosas de maneira oportuna. E a questão de como limitar o impacto das soluções contra DDoS no tráfego normal.

Aladaileh et al. (2020) analisam técnicas usadas em pesquisas para lidar com ataques DDoS em SDN. Os autores as classificam em duas categorias: técnicas baseadas na fonte e técnicas baseadas no destino. As técnicas baseadas na fonte são implantadas perto da fonte dos ataques para prevenir o ataque em seu início. As técnicas de detecção com base no destino empregam principalmente a detecção e defesa no alvo do ataque. O trabalho aponta três dificuldades em detectar ataques DDoS em SDN: a coleta de dados estatísticos, pois a maioria das técnicas de detecção necessitam coletar dados da camada de infraestrutura (*switches*) para detectar comportamentos anormais; a seleção de algoritmos, pois há várias técnicas que usam algoritmos distintos para realizar a detecção dos ataques, mas não há nenhum algoritmo capaz de, sozinho, detectar todos os tipos de ataques; e a resposta imediata, pois reagir aos ataques imediatamente é muito importante para que o controlador continue operante garantindo a disponibilidade da rede. Os autores ainda deixam três aspectos importantes para pesquisas futuras:

a) Limite dinâmico de pacotes por tempo para detecção de diferentes tipos de ataques, pois a maioria das pesquisas deixa um limite fixo, prejudicando a detecção precisa;
b) A detecção de ataques DDoS de múltiplas fontes com baixo tráfego de cada uma, pois muitas pesquisas analisam apenas uma característica dos cabeçalhos dos pacotes, dificultando detectar se o tráfego é legítimo ou não, já que é oriundo de várias fontes;
c) Implantar as soluções de segurança fora do controlador, pois quando são dispostas no controlador, este usa mais recursos para detectar as ameaças, além de estar operando a rede para tráfego legítimo, podendo levar à sobrecarga do mesmo.

2.6 Ataques de DoS e DDoS

Ataques de negação de serviço ou DoS (*Denial of service*) e ataques distribuídos de negação de serviço ou DDoS são tipos muito semelhantes, com a diferença de que, os distribuídos têm mais de uma fonte atacante. Dessa forma, um ataque distribuído é oriundo de diversos dispositivos simultaneamente, enquanto um DoS ocorre a partir de apenas um dispositivo, mas ambos possuem o mesmo objetivo: tornar um serviço indisponível para seus legítimos usuários.

Segundo Yan et al. (2016), há diversos tipos de ataques de DDoS, que podem ser divididos em duas classificações, baseadas no protocolo a ser usado:

a) Ataques de DDoS de inundação da camada de rede/transporte: geralmente lançam pacotes dos protocolos TCP, UDP, ICMP e DNS e focam em consumir toda a banda do serviço para impedir que usuários legítimos acessem o alvo. Alguns exemplos são: *UDP Flood*, *DNS Flood*, *SYN Flood*, *SYN-ACK-Flood*, *ICMP Flood*, *Ping Flood*.
b) Ataques de DDoS de inundação da camada de aplicação: o foco destes ataques é em esgotar os recursos de um servidor (e.g. CPU, *sockets*, memória, banda de entrada/saída, etc.), assim impedindo que usuários legítimos acessem seus serviços. Alguns exemplos são: *HTTP Fragmentation*, *Recursive GET*, *Faulty Application*.

Ataques de DDoS em SDN também podem ser caracterizados a partir das camadas em que são aplicados: ataques de DDoS na camada de aplicação, a partir de aplicações não autorizadas; ataques de DDoS no plano de controle, com inundação de pacotes e saturação do controlador; e ataques de DDoS no plano de dados, com o transbordamento da tabela de fluxos, a falsificação de *switches* ou a saturação do *buffer* (SINGH; BEHAL, 2020). Outros ataques podem ainda afetar os enlaces de comunicação, como a interface sul.

Os autores Sharafaldin et al. (2019) propõem uma taxonomia de ataques de DDoS em duas categorias: a primeira é baseada em ataques de reflexão (*reflection-based*), em que a identidade do atacante permanece oculta por utilizar componentes legítimos de terceiros. Dessa forma, o agente responsável realiza o ataque mediante intermediários comprometidos, que refletem os pacotes até o alvo. Estes ataques podem ser baseados em TCP, UDP ou ambos, explorando MSSQL, NFT, DNS, etc. A segunda categoria trata dos ataques de exploração (*exploitation-based*), em que a identidade do atacante também permanece oculta por utilizar componentes legítimos de terceiros. O atacante envia uma abundância de pacotes na tentativa de esgotar os recursos do alvo. Exemplos: *SYN Flood* e *UDP Flood*.

Segundo Jazi et al. (2017), ataques de DoS de baixo volume (*low-volume*) funcionam por transmitir pequenas quantidades de tráfego atacante ao alvo. Há três variações destes ataques: ataques de

baixa taxa (low-rate), que enviam tráfego em pulsos de tempo curto periodicamente; ataques de taxa lenta (*slow-rate*), que exploram parâmetros de tempo de um servidor enviando e recebendo tráfego de forma mais lenta que o esperado; e ataques *one-shot*, que causam dano por meio de uma única conexão ou requisição com a vítima, visando consumir recursos em excesso do alvo (e.g. *Apache Range Header attack*).

2.6.1 Ataques de DDoS ao Plano de Controle em SDN

Os autores Yan et al. (2016) colocam que o controlador pode ser um ponto único de falha em uma rede SDN, sendo um alvo atrativo para os ataques. O ataque pode ser direcionado às interfaces sul e norte, ou diretamente ao controlador. De toda forma, se o controlador for afetado, consequentemente toda a rede poderá ser impactada. Por exemplo, muitas regras de fluxo conflitantes de diferentes aplicações, podem causar um ataque DDoS no plano de controle.

Singh e Behal (2020) também apontam que, como o controlador faz o gerenciamento da rede, se falhar, compromete toda ela. O controlador centralizado e a comunicação entre o mesmo e os *switches* deve ser alvo de DDoS. Os autores ainda explicam a saturação do controlador, quando um atacante inunda os *switches* com pacotes falsos, e os *switches* enviam todas as requisições para o controlador, este último ficará ocupado tentando satisfazer todas essas requisições falsas. Assim, a atividade maliciosa poderá causar a exaustão de recursos do controlador, afetando toda a SDN.

2.6.2 Aprendizado de máquina

O aprendizado de máquina é um campo de estudo que visa as máquinas trabalharem em problemas de aprendizado. Um problema de aprendizado geralmente considera um número de amostras, objetivando predizer propriedades de dados desconhecidos. Cada amostra pode possuir vários atributos ou características. Problemas de aprendizado podem ser divididos em algumas categorias, como mostrado a seguir.

a) Aprendizado supervisionado: cujos dados possuem atributos adicionais que poderão ser o objetivo da previsão. Pode ser dividido em:
   - Classificação: as amostras pertencem a duas ou mais classes e a previsão de dados não rotulados é feita a partir do aprendizado com dados já rotulados;
   - Regressão: quando o resultado desejado consiste em uma ou mais variáveis contínuas.

b) Aprendizado não supervisionado: cujos dados de treinamento são um conjunto de vetores de entrada sem nenhum valor alvo correspondente.

O aprendizado de máquina trata de aprender algumas propriedades de um conjunto de dados e, em seguida, testar essas propriedades em relação a outro conjunto de dados. Uma prática comum em aprendizado de máquina é avaliar um algoritmo dividindo os dados em um de conjunto de treinamento, no qual aprendemos algumas propriedades e, um conjunto de teste, no qual testamos as propriedades aprendidas (DEVELOPERS, 2020).

No Capítulo 4, os algoritmos de aprendizado de máquina usados nessa pesquisa serão abordados, além dos conjuntos de dados trabalhados.

3 TRABALHOS RELACIONADOS

A seguir, serão citadas pesquisas de tema relacionado ao abordado nesse trabalho, destacando as principais ideias desenvolvidas nesses trabalhos.

Tang et al. (2016) propõem um *Network Intrusion Detection System* (NIDS) em SDN utilizando *Deep Neural Network - DNN*, *Gated Recurrent Unit Recurrent Neural Network - GRU-RNN* e o *dataset* NSL-KDD. O módulo NIDS é situado junto ao Controlador SDN que tem uma visão de todo tráfego da rede e requisita estatísticas dos *switches*. Ao receber a resposta de cada *switch*, o NIDS analisa potenciais anomalias. Detectadas e identificadas anomalias, o *OpenFlow* consegue mitigá-las modificando a tabela de fluxo. Novas políticas de segurança podem ser propagadas para os *switches* para prevenir outros ataques. Nos resultados os autores apresentam avaliações de precisão e perda para diferentes taxas de aprendizagem. De acordo com seus experimentos, o NIDS obteve uma precisão de 75,5% na detecção de anomalias na rede. Como conclusão, os autores refletem que embora seu sistema ainda não apresente resultados bons o suficiente para ser adotado comercialmente, é um trabalho relevante que pode ser melhorado. Uma solução possível seria aumentar o número de características de tráfego a serem analisadas, de maneira que o sistema proposto consiga uma maior taxa de precisão.

Os autores Mousavi e St-Hilaire (2017) propõem uma solução baseada em entropia para detectar ataques DDoS em uma SDN. Os dois objetivos principais do artigo são: usar a visão ampla do controlador da rede para detectar ataques DDoS, e propor uma solução que é eficaz e leve em termos dos recursos que usa. Para isso, observam o efeito de ataques DDoS ao controlador da rede. Para desenvolver o algoritmo de detecção dos ataques, os autores observaram que, segundo a especificação do *OpenFlow* 1.0, para cada nova conexão que chega, o controlador determinará uma ação. Portanto, sempre que um pacote é visto no controlador, ele é novo. Outro fato conhecido sobre os novos pacotes que chegam ao controlador é que o *host* de destino está dentro da rede do controlador. Sabendo que o pacote é novo e o destino está na rede, o nível de aleatoriedade pode ser quantificado pelo cálculo da entropia com base no endereço IP de destino. A detecção pode ser feita quando comparadas as estatísticas do fluxo com um tamanho de janela e um limite escolhidos e especificados no algoritmo proposto.

Carvalho et al. (2018) apresentam um ecossistema capaz de automatizar a tarefa do administrador em gerenciar redes SDN, realizar detecção e mitigação em tempo real de anomalias. Há um módulo que coleta estatísticas de tráfego e extrai atributos para enviar para outro módulo, que detecta anomalias, utilizando *Multinomial Logistic Regression - MLR*. Se uma anomalia for confirmada após análise dos dados do tráfego, é reconhecido o tipo dessa anomalia e identificados IPs, portas e *switches* afetados. Então, as ações de mitigação são definidas e um relatório de anomalias é gerado para auditoria. Para avaliar a eficácia do sistema proposto, é feita uma emulação de cenários usando Mininet e demonstrada

a habilidade de detectar anomalias em suas corretas classificações, além de medir seu desempenho. Depois é comparado os resultados com outras propostas na literatura. Segundo os autores, o sistema obteve resultados melhores em termos de precisão e baixa taxa de falsos positivos, comparado com outras soluções. Também consegue localizar interfaces comprometidas pelos atacantes e restaurar o funcionamento apropriado da rede após mitigar anomalias. O desempenho varia, mas mesmo sob alta intensidade de ataque os módulos continuam operantes e realizam suas tarefas em milissegundos. Os resultados indicam que o mecanismo proposto poderia rapidamente iniciar a detecção de ataques e prescrever contramedidas para os *switches*, apesar de demandar algum tempo para reconhecer um ataque de DDoS.

Os autores Li et al. (2018) utilizam tecnologia *deep learning* no campo de segurança de redes SDN. É construído um modelo de aprendizado profundo para detectar ataques DDoS, composto por: camada de entrada, camada de encaminhamento recursivo, camada recursiva reversa, camada oculta conectada totalmente, camada de saída. *Recurrent neural network - RNN*, *long short-term memory - LSTM* e *convolutional neural network - CNN* também são usados no modelo. É aplicado o modelo de aprendizado profundo de detecção de ataques DDoS em uma rede SDN baseada em *OpenFlow* e realizado o treinamento do modelo. É implementado o defensor de ataques DDoS de *deep learning*. São realizados testes de ataques de DDoS e ataques de DDoS em tempo real em ambiente experimental para avaliar o método proposto. Se o detector encontrar um ataque, encaminha os pacotes para o módulo *Information Statistics* para gerar estatísticas sobre o ataque. Então são criadas regras na tabela de fluxo e o *switch* pode ignorar os pacotes maliciosos conforme as informações. Segundo os autores, o modelo provou ter alta precisão (98%) de detecção durante os resultados do experimento. O controlador SDN pode gerar política de descarte e enviar ao *switch*, conforme o resultado da detecção obtido do modelo. Os resultados do experimento de ataques de DDoS em tempo real verificam que o defensor pode detectar efetivamente e defender de ataques de DDoS. Como vantagens, este modelo não apenas melhora a precisão de detectar ataques DDoS, mas também reduz o grau de dependência no ambiente de *hardware* e *software*, e simplifica a atualização em tempo real do sistema de detecção e a dificuldade de aprimorar a estratégia de detecção de ataques DDoS. No entanto, o método supervisionado de aprendizado para detectar ataques na rede é um ponto fraco, pois a maneira dos atacantes executarem os ataques é constantemente atualizada.

Os autores Deepa, Sudar e Deepalakshmi (2018) propõem um modelo híbrido de aprendizado de máquina, ao combinar os algoritmos *Support Vector Machine - SVM* e *Self Organized Map - SOM*, para proteger o controlador de uma SDN de ataques de DDoS. Além do modelo híbrido, os autores implementaram os modelos SVM e SOM separadamente, para comparação. Os experimentos foram executados utilizando a ferramenta Mininet para simular uma SDN e o POX como controlador. Ao

detectar um ataque, as conexões atacantes são fechadas e regras na tabela de fluxos são atualizadas. Os autores avaliam que a solução híbrida teve um maior desempenho quando comparado aos algoritmos SVM e SOM separadamente. Para o *dataset*, usaram uma ferramenta para gerar o tráfego.

Sridharan, Gurusamy e Leon-Garcia (2019) desenvolvem uma abordagem que detecta anomalias na tabela de fluxos, causadas por regras maliciosas instaladas por um controlador comprometido, sem sobrecarregar o plano de controle. É proposto um *framework* baseado em *machine learning* para detectar comportamento anômalo na tabela de fluxo e identificar o controlador afetado. É desenvolvida uma técnica para detecção de anomalias em SDN chamada *Machine learning based detection Technique for Anomaly Detection in SDN* (MTADS) como parte do *framework*. São descritos ataques que tentam evadir detecções e parecer comportarem-se normalmente na rede. Para avaliar os resultados são usadas métricas tradicionais para aprendizado de máquina: precisão, revocação e acurácia de classificação. O MTADS proposto usa o algoritmo *Density-based spatial clustering of applications with noise* (DBSCAN) para determinar anomalias, mas também foi experimentado o MTADS com *K-Means* para comparar. Os resultados apresentados mostram um melhor desempenho quando usando o MTADS com o algoritmo DBSCAN do que com o *K-Means*.

Os autores Santos et al. (2019) discutem a ocorrência de ataques DDoS em redes SDN e propõem a implementação de algoritmos de aprendizado de máquina para realizar a detecção das atividades maliciosas. Utilizam a ferramenta *Scappy* para gerar o tráfego legítimo e malicioso para treinar os algoritmos e realizar os testes de detecção. Para os experimentos, utilizaram o Mininet (CONTRIBUTORS, 2021) para criar uma SDN e o POX (POX, 2021a) como controlador. Simularam ataques ao plano de controle, inundando as tabelas de fluxos e comprometendo a largura de banda da rede. Os algoritmos usados na detecção foram *Multiple Layer Perceptron - MLP*, *Random Forest - RF*, SVM e *Decision Tree - DT*. Resultados mostraram que o algoritmo DT obteve o melhor desempenho e, a acurácia de detecção dos ataques realizados diretamente ao controlador foi a pior de tráfego malicioso.

Sen, Gupta e Ahsan (2019) simulam uma SDN para realizar ataques de DDoS e identificar tráfego malicioso utilizando modelos de aprendizado de máquina. Os autores compararam a eficácia de precisão de detecção com alguns algoritmos, como *AdaBoost - AB*, *Bayes network - BN*, *Naïve Bayes - NB*, MLP, SVM, *Random Forest - RF* e *J48 decision tree - JDT*. O ataque foi realizado usando ICMP *flood* e foram coletados dados dos pacotes para formar um conjunto de dados. Os algoritmos de aprendizado de máquina foram treinados com um *dataset* contendo dados de tráfegos legítimo e malicioso de seis protocolos diferentes: TCP, UDP, ICMP, ARP, IPv4, e SSH. No entanto, é realizado apenas um tipo de ataque, utilizando apenas o protocolo ICMP. Os experimentos foram realizados apenas em am-

biente simulado. Resultados mostraram que o algoritmo AB se saiu melhor que os outros, com taxas de *Precision*, *Recall* e *F-measure* de 93%.

Novaes et al. (2020) apresentam um sistema de detecção e mitigação de ataques de DDoS e ataques *Portscan* em ambientes SDN baseado em (LSTM-FUZZY). O sistema consiste em três fases: na primeira fase é feita a predição do comportamento normal do tráfego da rede e a definição de *thresholds* de normalidade; na segunda fase é aplicada a lógica Fuzzy para determinar se há anomalias, usando como parâmetro o tráfego predito e os *thresholds* definidos na primeira fase; e na terceira fase são aplicadas políticas de mitigação com a intenção de tomar contramedidas contra os ataques detectados, garantindo a operação da rede. Comparado a outros trabalhos relacionados que avaliam apenas alguns tipos de ataques DDoS, este trabalho propõe a detecção de 12 tipos. Também, o sistema proposto pode aprender o comportamento normal da rede, conseguindo detectar ataques *zero-day*. Para os experimentos foram utilizados dois cenários emulando uma rede pelo Mininet. As métricas utilizadas foram: *Precision*, *Recall*, *False-positive rate*. Segundo os autores, os resultados foram positivos e mostram que os módulos do sistema o tornam eficiente e garantem a disponibilidade dos serviços.

Sudar et al. (2021) utilizaram os algoritmos de aprendizado de máquina DT e SVM para detectar ataques de DDoS em SDN. Os autores justificaram a escolha dos algoritmos pela acurácia e baixa complexidade. Usaram o *dataset* KDD99 para treinar os modelos. O tráfego foi classificado em normal ou ataque pelos modelos. Quando detectados os ataques, foram enviados alertas ao controlador para descartar os pacotes sinalizados como maliciosos. O ambiente de experimentação foi todo virtualizado, utilizando Mininet. Os resultados apresentados mostraram que o SVM obteve melhores taxas que o DT em todas as métricas (*accuracy*, *precision*, *recall* e *F-score*) calculadas. A acurácia do SVM foi de 85%, enquanto do DT foi de 78%.

Sayed et al. (2022) propõem um modelo de *deep learning* baseado em LSTM e *autoencoder* para detectar ataques de DDoS em SDNs utilizando uma abordagem de seleção de atributos por meio de *random forest* e *information gain*. O sistema proposto consiste em fazer uma modelagem de ataques de DDoS a partir de *datasets*, observando características específicas de SDN. Então, selecionam os atributos usando RF e IG, criando subconjuntos de dados. O modelo de *deep learning* proposto é aplicado aos dados, resultando na classificação dos ataques de DDoS. Os *datasets* utilizados para treinamento foram o InSDN, CICIDS2017 e CICIDS2018. O modelo proposto obteve altas taxas de sucesso nas métricas avaliadas, variando entre 99% e 100%. Os autores ainda relatam que a solução apresentada não oferece sobrecarga ao controlador da rede.

O Quadro 3.1 abaixo apresenta uma comparação entre este projeto e os trabalhos relacionados descritos anteriormente. Em seguida, será apresentada a metologia, dividida entre os capítulos 5 e 6.

Quadro 3.1 – Comparação deste projeto com os trabalhos relacionados descritos anteriormente (continua).

Título	Objetivo	O que foi desenvolvido?	Utiliza ML?	Dataset	DDoS no Plano de Controle?	Testbed	Ano
Deep Learning Approach for Network Intrusion Detection in Software Defined Networking	Detectar intrusões em SDN.	NIDS baseado em fluxos com Deep Neural Network.	DNN, GRU-RNN.	NSL-KDD (2009).	DoS, entre outros tipos de anomalias. Não especifica o Plano.	Virtual.	2016
Early Detection of DDoS Attacks Against Software Defined Network Controllers	Detectar DDoS em controlador SDN.	Algoritmo baseado na variação de entropia do IP de destino.	Não. Utiliza entropia.	Usa uma ferramenta para gerar tráfego.	Sim. DDoS direcionado ao controlador.	Virtual (Mininet).	2017
An ecosystem for anomaly detection and mitigation in software-defined networking	Detectar e mitigar anomalias em SDN.	Ecossistema modular para detectar e mitigar.	MLR.	Próprio.	DDoS, entre outros tipos de anomalias.	Virtual (Mininet).	2018
Detection and defense of DDoS attack–based on deep learning in OpenFlow-based SDN	Detecção e defesa contra DDoS em SDN.	Modelo de deep learning para detectar DDoS.	Deep Learning, RNN, LSTM, CNN.	ISCX2012.	DDoS. Não foca no Plano de Controle.	Híbrido, com a parte de SDN virtual (Open vSwitch).	2018
Detection of DDoS Attack on SDN Control plane using Hybrid Machine Learning Techniques	Detectar DDoS no Plano de controle de uma SDN.	Modelo híbrido SVM-SOM para detectar.	SVM, SOM, SVM-SOM.	Próprio.	Sim. DDoS direcionado ao controlador.	Virtual (Mininet).	2018
Anomalous Rule Detection using Machine Learning in Software Defined Networks	Detectar anomalias em SDN.	Framework baseado em ML para detecção.	MTADS, baseado em DBSCAN.	Próprio.	Inundação de pacotes na rede, sem foco no Plano de Controle.	Virtual (Mininet).	2019
Machine learning algorithms to detect DDoS attacks in SDN	Detectar DDoS em SDN.	Técnicas de detecção de DDoS com ML.	SVM, MLP, Decision tree, Random forest.	Próprio.	Sim. DDoS no controlador, na flow-table e na banda.	Virtual. (Mininet).	2019

Quadro 3.1 Comparação deste projeto com os trabalhos relacionados descritos anteriormente (conclusão).

Título	**Objetivo**	**O que foi desenvolvido?**	**Utiliza ML?**	**Dataset**	**DDoS no Plano de Controle?**	**Testbed**	**Ano**
Leveraging Machine Learning Approach to Setup Software-Defined Network(SDN) Controller Rules During DDoS Attack	Detectar DDoS em SDN.	Técnica baseada em ML para detectar DDoS.	Bayes net, Naive Bayes, MLP, SVM, AdaBoost, J48 decision tree, Random forest.	Próprio.	Testado com ICMP flood apenas, apesar do dataset conter outros tipos. Sem foco no Plano de Controle.	Virtual (Mininet).	2019
Long Short-Term Memory and Fuzzy Logic for Anomaly Detection and Mitigation in Software-Defined Network Environment	Detectar DDoS e portscan em SDN.	Sistema de detecção e mitigação de DDoS e portscan.	Deep learning, LSTM.	CICDDoS 2019.	DDoS, portscan sem foco no Plano de Controle.	Virtual (Mininet).	2020
Detection of Distributed Denial of Service Attacks in SDN using Machine Learning techniques	Detectar DDoS em SDN.	Técnicas de detecção por Aprendizado de Máquina.	SVM, decision tree.	KDD99.	DDoS, não especificado se tem foco no Plano de Controle.	Virtual (Mininet).	2021
A Flow-Based Anomaly Detection Approach With Feature Selection Method Against DDoS Attacks in SDNs	Detectar DDoS em SDN.	Abordagem de detecção utilizando Deep learning.	Deep Learning, LSTM, Autoencoder, Random Forest e Information Gain.	InSDN, CICIDS2017 e CICIDS2018.	DDoS sem focao, mas testando impacto no Plano de Controle.	Virtual.	2022
Detecção de Ataques de Negação de Serviço ao Plano de Controle de SDN Utilizando Aprendizado de Máquina	Detectar ataques de negação de serviço ao controlador de uma SDN.	Técnicas de detecção por Aprendizado de Máquina.	Naïve Bayes, decision tree, Gradient Boosting e SVM.	Próprio.	Foco em DDoS no Plano de Controle.	Real com switch Extreme Summit x440.	2023

Fonte: do autor (2023).

4 METODOLOGIA

Este projeto utilizou algoritmos de aprendizado de máquina para detectar ataques de DDoS em uma SDN, analisando o tráfego em busca de potenciais ataques e classificando os fluxos em maliciosos ou legítimos. Foram realizados experimentos em laboratório, utilizando topologias de rede para simular cenários e executar os ataques, bem como detectá-los e classificá-los para diferenciar as atividades atacantes de um tráfego normal. Os ataques foram direcionados a um servidor, no entanto, além de negar os serviços do mesmo, também tiveram o propósito de afetar o controlador da rede. O impacto ao controlador é obtido devido aos fluxos ainda não reconhecidos que passam por ele antes de chegarem ao destino. Dessa forma, muito fluxos maliciosos destinados ao servidor, passam antes pelo controlador, estressando seus recursos e prejudicando o seu funcionamento pleno.

4.1 *Datasets*

Para ser possível identificar se o tráfego é legítimo ou malicioso, é necessário utilizar *datasets* contendo dados destes tráfegos. *Datasets* são conjuntos de dados compilados com informações relevantes a uma determinada necessidade. Como visto, neste trabalho a necessidade é por dados de pacotes que trafegam em uma rede, sendo oriundos de ataques de DDoS e de serviços legítimos, em SDN. Há *datasets* disponíveis online, como o KDD'99 e o NSL-KDD, populares, porém antigos e não abordam SDN. Há alguns outros mais atuais, com dados de ataques de DDoS, mas sem abordar SDN, ou que abordam SDN, mas não são específicos apenas para ataques de DDoS. Sendo assim, para este trabalho foram construídos *datasets* no mesmo ambiente em que, posteriormente, foram realizados os experimentos de classificação do tráfego. Dessa forma, os conjuntos de dados agregaram atributos importantes, aplicados à pesquisa.

Foram construídos dez *datasets*, através da injeção de tráfego com variações entre legítimo, malicioso e misto. Para cada *dataset*, foi capturado tráfego por cerca de vinte e quatro horas e salvo em arquivos[1], totalizando 3,54GB de dados. Para selecionar os atributos, foram considerados os protocolos utilizados nos ataques (TCP, UDP e ICMP), características da rede, e o código do controlador POX foi utilizado como base, observando-se quais atributos poderiam ser capturados com o uso deste controlador. Assim, cento e vinte e seis atributos foram observados como selecionáveis para a construção dos *datasets*. A princípio, os *datasets* foram construídos usando todos os atributos disponíveis, mas foram reavaliados ao longo do tempo por meio de rodadas empíricas de treinamento e validação. Foi constatado que nem todos os atributos seriam úteis ao problema, pois não apareciam durante as coletas de fluxos. Observando quais apareciam e quais não nas coletas, ao final os *datasets* foram construídos a partir de

[1] <https://drive.google.com/drive/u/0/folders/0AKH22IFb8bxUUk9PVA>

cento e quatro atributos, listados no Quadro 4.1. Apesar da criação de dez *datasets*, quando os modelos foram construídos, estes *datasets* foram combinados. A combinação foi possível devido a todos eles terem os mesmos atributos, mesmo possuindo características distintas de tráfego.

Quadro 4.1 – Lista de atributos selecionados para a construção dos *datasets*.

portaSwitch	IPversao	TCPdstport	PStx_errors	FSpacket$_c$*ount*	FSdl_vlan	CALCcollisions	CALCbyte_count
macOrigemPacote	IPhl	TCPseq	PSrx$_c$*rc*$_e$*rr*	FShard_timeout	FSnw_src	CALCrx_errors	CALCduration_sec
macDestinoPacote	IPiplen	TCPack	PScollisions	FSbyte_count	FSin_port	CALCtx_bytes	CALCmatch.dl_type
ARPop	IPttl	TCPf	PSrx_errors	FSduration_sec	FSmax_len	CALCrx_dropped	CALCmatch.dl_vlan_pcp
ARPhwtypw	UDPsrcport	TCPops	PStx_bytes	FSdl_type	FStype	CALCtx_packets	CALCmatch.nw_proto
ARPprototype	UDPdstport	VLANid	PSrx_dropped	FSnw_dst	FSport	CALCport_no	CALCmatch.nw_tos
ARPhwsrc	UDPlen	VLANpcp	PStx_packets	FSdl_vlan_pcp	CALCrx_over_err	CALCduration_nsec	CALCmatch.tp_dst
ARPhwdst	ICMPtipo	VLANeth_type	PSport_no	FSdl_src	CALCtx_dropped	CALCpriority	CALCmatch.tp_src
ARPprotosrc	ICMPcodigo	PSrx_over_err	FSduration_nsec	FSnw_proto	CALCrx_packets	CALCidle_timeout	CALCmatch.dl_vlan
ARPprotodst	ICMPid	PStx_dropped	FSpriority	FSnw_tos	CALCrx_frame$_e$*rr*	CALCcookie	CALCmatch.in_port
IPprotocol	ICMPsequencia	PSrx_packets	FSidle_timeout	FStp_dst	CALCrx_bytes	CALCtable_id	CALCactionsmax_len
IPsrcip	TCPlenpkt	PSrx_frame$_e$*rr*	FScookie	FStp_src	CALCtx_errors	CALCpacket_count	CALCactionstype
IPdstip	TCPsrcport	PSrx_bytes	FStable_id	FSdl_dst	CALCrx_crc_err	CALChard_timeout	CALCactionsport

Fonte: do autor (2023).

Quando prontos, os *datasets* puderam ser utilizados na construção dos modelos de aprendizado de máquina que serviram para classificar os fluxos nos experimentos.

4.2 Algoritmos e modelos de aprendizado de máquina

Para a análise dos fluxos foram utilizados algoritmos de aprendizado de máquina, auxiliado pelo projeto *Scikit-learn*, um módulo *Python* que integra uma ampla gama de algoritmos de aprendizado de máquina de última geração para problemas supervisionados e não supervisionados de média escala. Este pacote se concentra em levar o aprendizado de máquina para não especialistas usando uma linguagem de alto nível de uso geral (PEDREGOSA et al., 2011).

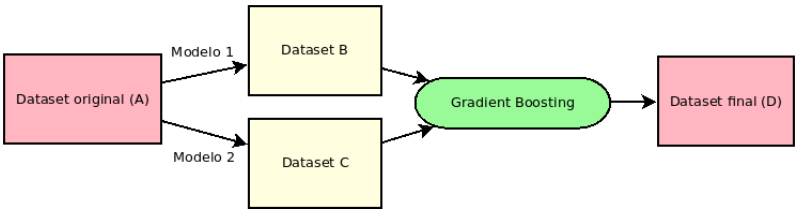
Devido à existência de várias opções de algoritmos, foi importante que aqueles selecionados fossem de natureza distinta, usando técnicas diferentes para processar os dados. Também foi considerada a natureza do problema, a classificação dos fluxos que trafegam na SDN, além de outras variáveis, como o tempo hábil para a construção dos modelos e testes exaustivos de muitas opções, a comparação com trabalhos relacionados e quais algoritmos utilizaram. Os algoritmos utilizados neste trabalho foram o *Gradient Boosting*, *Decision Tree*, *Support Vector Machine* e *Naïve Bayes*. Esses algoritmos de aprendizado de máquina utilizados são baseados em modelo supervisionado.

Após a geração dos *datasets*, foram feitos refinamentos e engenharia de atributos nos dados, identificando seus tipos, valores, erros de captura, distribuição, padrões. Também foi realizado pré-processamento dos *datasets* para os algoritmos processarem melhor os dados, com troca de rótulos por números, definição de limites, discretização, aplicação de normalização ou padronização. Foram realizados testes quanto à aplicação de padronização ou normalização na preparação dos dados, visando colocá-los em um intervalo de valores comuns, para a construção dos modelos. Os resultados foram

semelhantes, não havendo relevância na escolha de uma em detrimento da outra, mas para haver critério, os experimentos foram feitos com padronização. Também, para a construção dos modelos, foram realizados testes empíricos para definir o percentual de treinamento e teste. Os melhores resultados foram obtidos a partir da proporção de 60% dos dados para treinamento e 40% para teste, considerando custos computacionais e tempo de construção dos modelos. As características dos algoritmos utilizados são descritas a seguir.

O *Gradient Boosting* (GB) é uma técnica de aprendizado de máquina usada em problemas de regressão e classificação, que produz um modelo de previsão na forma de um *ensemble* de modelos de previsão, como o *Decision Tree*. O objetivo dos métodos *ensemble* é combinar as previsões de vários modelos construídos com um determinado algoritmo de aprendizado, a fim de melhorar a generalização/robustez em um único modelo. Esta técnica (GB) constrói o modelo baseado em outros métodos de *boosting*, e os generaliza, permitindo a otimização de uma função de perda diferenciável arbitrária (DEVELOPERS, 2020). A Figura 4.1 ilustra de maneira bem simples a ideia geral de método *ensemble*, no qual classificadores fracos tratam um *dataset* original de forma independente e, posteriormente, entra a técnica de *ensemble* do GB trabalhando nos *datasets* anteriores, gerando um conjunto de dados final.

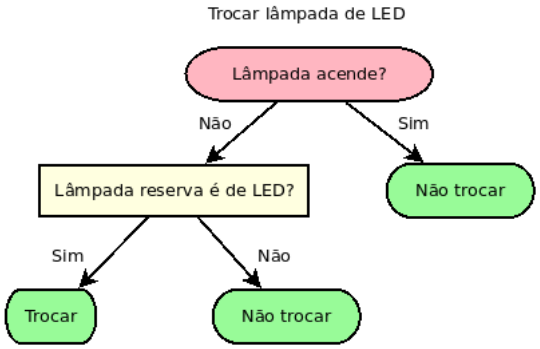
Figura 4.1 – Ideia geral de método *ensemble* com *gradient boosting*



Fonte: do autor (2023).

Árvores de decisão (*Decision Trees*) são um método de aprendizado supervisionado não paramétrico usado para classificação e regressão. O objetivo é criar um modelo que preveja o valor de uma variável de destino, aprendendo regras de decisão simples inferidas das características dos dados. Uma árvore pode ser vista como uma aproximação constante por partes. Algoritmos de florestas aleatórias (*Random Forest*) podem usar como base árvores de decisão. Em florestas aleatórias, cada árvore no conjunto é construída a partir de uma amostra retirada com substituição do conjunto de treinamento. Além disso, ao dividir cada nó durante a construção de uma árvore, a melhor divisão é encontrada em todos os recursos de entrada ou em um subconjunto aleatório de tamanho do máximo de características (DEVELOPERS, 2020). A Figura 4.2 ilustra um processo de decisão simples de uma *decision tree* para trocar uma lâmpada de LED.

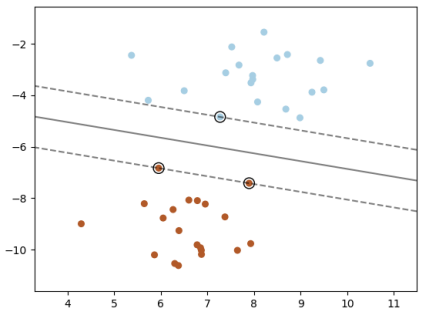
Figura 4.2 – Processo de decisão simples de uma *decision tree*



Fonte: do autor (2023).

Os algoritmos do tipo *Support Vector Machines* (SVMs) são conjuntos de métodos de aprendizado supervisionado usados para classificação, regressão e detecção de *outliers*. Há algumas vantagens no uso de SVMs, como a eficácia em espaços dimensionais elevados e ainda ser eficaz nos casos em que o número de dimensões é maior que o número de amostras. Usa um subconjunto de pontos de treinamento na função de decisão (chamados de vetores de suporte), portanto, também é eficiente em termos de memória. Também há desvantagens em seu uso, como não fornecer estimativas de probabilidade diretamente, mas calculá-las usando uma validação cruzada quíntupla (*five-fold cross-validation*) cara (DEVELOPERS, 2020).

Figura 4.3 – Vetores de suporte em SVM



Fonte: obtido de <https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation>. Acesso em: janeiro, 2023.

A Figura 4.3 mostra a função de decisão para um problema linearmente separável, com três vetores de suporte (pontos de treinamento).

O *Naïve Bayes* (NB) é um algoritmo de aprendizado supervisionado baseado na aplicação do teorema de Bayes com a suposição "ingênua" de independência condicional entre cada par de *features*, dado o valor da variável de classe. Apesar de suas suposições aparentemente simplificadas, os classi-

ficadores NB funcionam bem em muitas situações do mundo real, como a classificação de documentos e filtragem de *spam*. Eles requerem uma pequena quantidade de dados de treinamento para estimar os parâmetros necessários (DEVELOPERS, 2020). A equação 4.1 ilustra o teorema de *Naïve Bayes*, em que o algoritmo homônimo é baseado, onde P é a probabilidade, h é uma hipótese e o uma observação. Por exemplo, $P(h|o)$ significa a probabilidade de uma hipótese dada uma observação.

$$P(h|o) = \frac{P(o|h)P(h)}{P(o)} \tag{4.1}$$

Uma síntese do processo de geração dos modelos é apresentada a seguir.

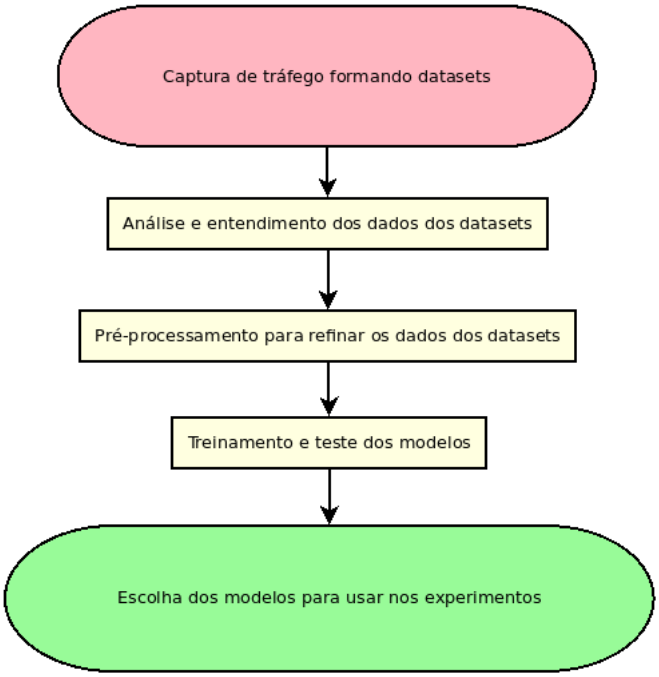
a) Foram, primeiramente, gerados os *datasets* capturando os dados de fluxos a partir do controlador;
b) A seguir, os *datasets* foram analisados e combinados com auxílio da ferramenta *Jupyter Notebook* (JUPYTER, 2023), para entender os dados, seus tipos, valores, erros na captura, distribuição, visualizar padrões, engenharia de atributos;
c) Ocorreu o pré-processamento dos *datasets*, para que os algoritmos de aprendizado pudessem processar melhor esses dados - troca de *labels* por números, definição de limites, discretização, aplicação de padronização ou normalização;
d) Foram feitos o treinamento e teste dos modelos para coletar as métricas de desempenho com auxílio da ferramenta *ML flow* (PROJECT, 2022);
e) Escolha dos modelos que apresentaram as melhores métricas e uma proporção de treinamento e teste viável do ponto de vista de tempo de treinamento, além da natureza do algoritmo usado em cada modelo.

A Figura 4.4 ilustra os passos descritos acima.

4.3 Ferramentas

Além dos *datasets*, foi necessário utilizar ferramentas diversas durante a execução deste projeto. Algumas auxiliaram no monitoramento do ambiente e dos experimentos, enquanto outras foram indispensáveis ao executar os ataques para realizar os testes que permitiram obter os resultados deste trabalho. Para tráfego malicioso, foi utilizada a ferramenta que simula ataques com *botnets* chamada BoNeSi (GOLDSTEIN, 2015). Através dela foram simulados vários tipos de ataques, utilizando os protocolos de rede TCP, UDP e ICMP, gerando então, tráfego malicioso para a coleta de dados. A seguir, são descritas as ferramentas utilizadas no projeto.

Figura 4.4 – Processo de criação de modelos



Fonte: do autor (2023).

4.3.1 Wireshark

O Wireshark é um famoso analisador de protocolos que permite uma visão ao nível microscópico do que acontece em uma rede de computadores. Pode inspecionar centenas de protocolos e realiza captura em tempo real das informações dos pacotes (Wireshark Foundation, 2021). O analisador de pacotes Wireshark foi usado para obter o IP de origem e destino, protocolo, tamanho, entre outros dados dos pacotes via API pcap.

4.3.2 Ostinato

O *software* gerador de tráfego Ostinato (OSTINATO, 2021) foi usado para gerar tráfego legítimo na construção de alguns *datasets*. O gerador tem suporte para diversos protocolos, como ARP, IPv4, IPv6, TCP, UDP, ICMPv4, ICMPv6, HTTP, etc.

4.3.3 BoNeSi

BoNeSi ou DDoS Botnet Simulator é uma ferramenta para simular tráfego de *botnet* em ambiente de teste, destinada ao estudo dos efeitos de ataques DDoS. O simulador é capaz de gerar tráfego de ataques de inundação ICMP, UDP e TCP (HTTP). Podem ser configurados parâmetros como taxa,

volume de dados, endereços IP das fontes, URLs, dentre outros (GOLDSTEIN, 2015). Foi usada para gerar tráfego malicioso tanto na criação dos *datasets*, quanto nos experimentos de detecção dos ataques.

4.3.4 T50

T50 é uma ferramenta designada para testes de stress, capaz de injetar altos volumes de pacotes de vários protocolos, incluindo ICMP, TCP e UDP. Permite simular ataques DoS e DDoS, como *SYN Flood* (PISSARA, 2021). Foi usada para auxiliar na geração de tráfego malicioso na construção de alguns *datasets* e também na geração de tráfego legítimo nos experimentos de detecção, a partir de um *script*.

4.4 Métricas de avaliação

Para validar os resultados dos algoritmos de aprendizado de máquina, foram usadas métricas de desempenho, a partir de taxas de verdadeiros positivos (TP), verdadeiros negativos (TN), falsos positivos (FP) e falsos negativos (FN). As métricas usadas para calcular o desempenho de detecção de ataques nesse trabalho são as seguintes:

a) *Accuracy* - porcentagem de classificação de tráfegos legítimos e ataques verdadeiros sobre o total de tráfego;

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{4.2}$$

b) *Precision* - apresenta o volume de tráfego predito como ataque que realmente é ataque sobre o total de fluxos classificados como ataque;

$$Precision = \frac{TP}{TP+FP} \tag{4.3}$$

c) *Recall* - apresenta o percentual de volume de tráfego predito como ataque dividido pelo total de tráfego malicioso ocorrido;

$$Recall = \frac{TP}{TP+FN} \tag{4.4}$$

d) *F-score* - considera tanto a *accuracy* quanto o *recall* para calcular o desempenho da classificação.

$$F-score = \frac{2TP}{2TP+FP+FN} \tag{4.5}$$

A seguir, são descritos os experimentos em que são gerados tráfego legítimo e malicioso para a classificação, por modelos de aprendizado de máquina, a fim de identificar quais fluxos são oriundos de ataques de DDoS e quais não.

5 GERANDO ATAQUES DE DDOS

Neste capítulo, são descritos os cenários e a topologia utilizada durante os experimentos de ataques e classificação dos fluxos, bem como os detalhes desses experimentos.

5.1 Cenários

Os experimentos foram executados em uma rede SDN montada com equipamentos reais, visando gerar resultados relevantes e sem as limitações de um cenário virtualizado. Foi utilizado um *switch*-OF Extreme Summit x440-48t de 52 portas, CPU *Single Core* de 500 MHz com o sistema operacional ExtremeXOS 16.2.3.5. A Figura 5.1 mostra *switches* do tipo mencionado, sendo os dois mais inferiores na pilha os x440-48t. Além do *switch*, foram utilizados oito computadores para formar a rede, distribuídos entre os planos de uma rede SDN. Um dos computadores utilizados não participou da fase de construção dos *datasets*. Esta máquina foi adicionada somente durante os experimentos, com o intuito de observar se haveria alguma diferença em relação às outras máquinas, já conhecidas no ambiente e utilizadas na construção dos *datasets*. Não foram observadas diferenças, o controlador conseguiu reconhecer a máquina nova e a rede foi operada normalmente com a adição deste computador.

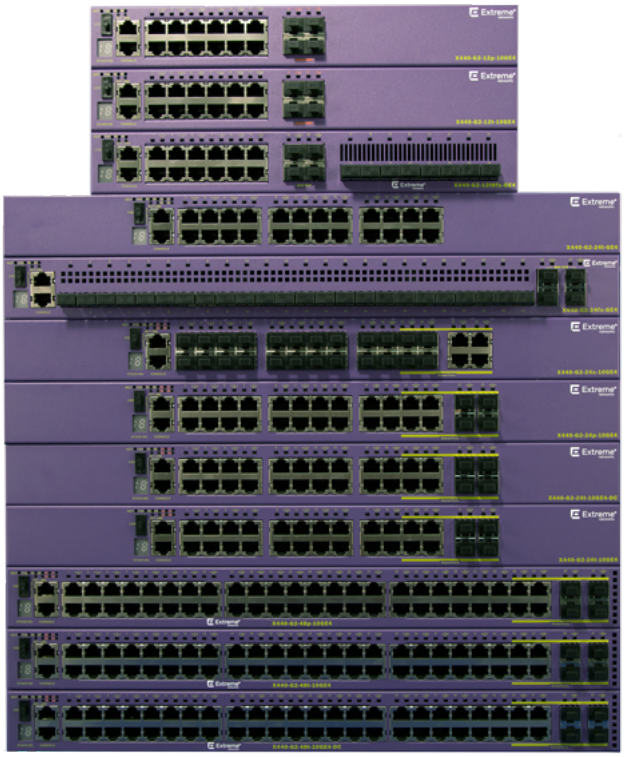
Neste trabalho, foi utilizado o POX (POX, 2021a), que é um controlador OF (versão 1.0) centralizado, de código aberto e escrito em *Python*, para uso geral. Enquanto outros controladores são mais usados em redes de produção, o POX é muito popular para prototipação e pesquisa, devido à sua estrutura simples, visando prover um ambiente eficiente e fácil para experimentação e testes (NOMAN; JASIM, 2020).

Foram executados cinquenta e dois experimentos, usando duas metologias distintas para a geração de tráfegos: (i) a primeira, com 48 experimentos, gera um tráfego imutável (*TI*) em um curto período; (ii) e a segunda, com 4 experimentos, gera um tráfego variável (*TV*) ao longo de um tempo maior. Sendo assim, temos dois cenários: um com tráfego definido e imutável e, o outro, com mudanças no tráfego durante a sua execução. Todos foram realizados em uma mesma topologia, variando o algoritmo de aprendizado de máquina selecionado para classificar as entradas de fluxos.

5.1.1 Topologia

A topologia de rede montada com um *switch*-OF, continha duas VLANs. Em uma VLAN estavam conectadas duas máquinas, sendo uma executando o controlador POX e, a outra, o servidor de aprendizado de máquina, com a finalidade de classificar os fluxos. A solução de detecção e classificação do tráfego fica em um servidor separado do controlador, para evitar que os recursos desse último

Figura 5.1 – Família de *switches* Extreme Summit x440



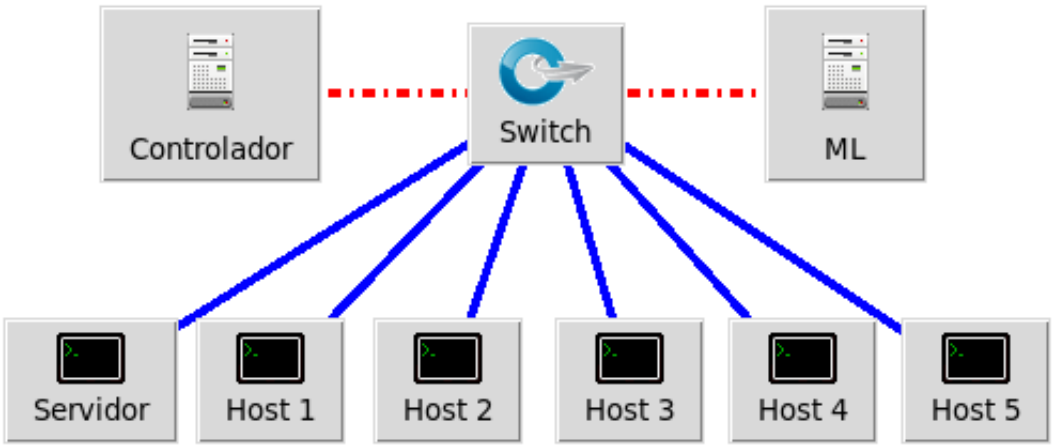
Fonte: obtido de <https://www.extremenetworks.com/product/x440-g2/>. Acesso em: junho, 2021.

sejam sobrecarregados, potencialmente causando a inoperabilidade do Plano de Controle e, consequentemente, de toda a rede. Na outra VLAN, foram conectados os outros seis computadores, sendo um deles o servidor alvo e, os outros, *hosts* responsáveis por gerar tráfego, seja legítimo ou oriundo de ataques.

A Figura 5.2 ilustra a topologia previamente descrita, destacando nas linhas pontilhadas em vermelho, as conexões da VLAN utilizada pelo controlador e pelo servidor de *machine learning*. As linhas contínuas azuis destacam as conexões da outra VLAN, em que acontecem as operações no plano de dados da SDN. O controlador deve ficar em uma VLAN separada do plano de dados, por questões de segurança. O servidor de ML precisa ficar na mesma VLAN que o controlador, pois precisam se comunicar para que a classificação dos fluxos opere.

Durante as operações na rede, os *hosts* têm seu tráfego, legítimo ou malicioso, endereçado ao servidor alvo, enviado à tabela de fluxos do *switch*-OF por meio do protocolo *Openflow*. Por sua vez, o *switch*-OF envia os fluxos novos ao controlador, esperando que este responda com uma regra de fluxo, orientando como tratar esse tráfego. O controlador, ao receber os fluxos, encaminha-os para o servidor de *machine learning* por meio de um *gateway*, que repassa os dados ao algoritmo responsável por classificar

Figura 5.2 – Topologia usada nos experimentos de classificação



Fonte: do autor (2023).

cada entrada em: malicioso ou legítimo. O classificador grava em um arquivo de *log* os resultados da classificação. Enquanto isso, o controlador responde ao *switch*, que encaminha o tráfego ao alvo.

A Figura 5.3 mostra o laboratório real, com os equipamentos usados em destaque. Nota-se o *switch* de cor roxa, apoiado em cima de algumas das máquinas, que serviram como *hosts*, servidores e controlador.

Figura 5.3 – Foto do laboratório onde foram executados os experimentos

Fonte: do autor (2023).

5.1.2 Protocolos explorados

Os autores Banjar et al. (2014) fazem uma comparação entre as camadas dos modelos OSI, TCP/IP e da arquitetura SDN, como mostrado na Figura 5.4. Como os experimentos neste trabalho são focados no Plano de Controle de uma SDN, os tráfegos maliciosos gerados nos experimentos foram baseados nos protocolos TCP (camada 4 OSI), UDP (camada 4 OSI) e ICMP (camada 3 OSI), paralelamente abordando o Plano de Controle da SDN. Estes três protocolos também são popularmente explorados em ataques de DDoS, facilitando a decisão de utilizá-los.

Figura 5.4 – Comparação dos modelos OSI, TCP/IP e SDN

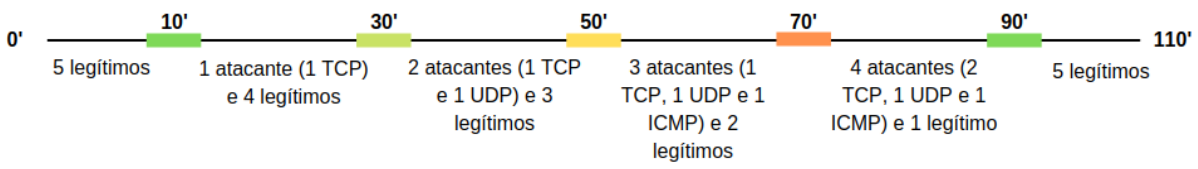
<table>
<tr><th>Network Layers</th><th>OSI Model</th><th>TCP/IP</th><th>SDN</th></tr>
<tr><td>Layer 7</td><td>Application</td><td rowspan="3">Application</td><td rowspan="3">Application</td></tr>
<tr><td>Layer 6</td><td>Presentation</td></tr>
<tr><td>Layer 5</td><td>Session</td></tr>
<tr><td>Layer 4</td><td>Transport</td><td>Transport</td><td rowspan="3">Control Layer</td></tr>
<tr><td>Layer 3</td><td>Network</td><td>Internet</td></tr>
<tr><td>Layer 2</td><td>Data Link</td><td rowspan="2">Network Access</td></tr>
<tr><td>Layer 1</td><td>Physical</td><td>Physical</td></tr>
</table>

Fonte: (BANJAR et al., 2014).

5.1.3 Experimentos

No Cenário *TV*, em que ocorreram mudanças em tempo real no tráfego, cada um dos quatro experimentos foi executado de forma contínua, alterando propositalmente o tipo de tráfego enviado pelos *hosts*. As alterações foram equivalentes para cada experimento, tornando a comparação justa entre os resultados obtidos a partir de cada algoritmo. Em cada experimento, foi executado um algoritmo de aprendizado de máquina para realizar a classificação dos fluxos. Os experimentos tiveram duração de 1 hora e 50 minutos para cada algoritmo. A Figura 5.5 ilustra a linha do tempo com as ocorrências durante os experimentos.

Figura 5.5 – Linha do tempo dos experimentos com alteração de tráfego em tempo real



Fonte: do autor (2023).

Ao iniciar a execução, todos os cinco *hosts* começam a enviar requisições de tráfego legítimo ao servidor. A partir de um *script* desenvolvido para emular tráfego de usuários, é iniciada a injeção de pacotes usando vários protocolos diferentes, upload de arquivos e uma variação na formação desse tráfego, alterando de tempo em tempo o tipo de requisição. Dez minutos após o início, o *Host* 1 interrompe o envio de tráfego legítimo e começa a simular um ataque DDoS (protocolo TCP) contra o servidor, pela ferramenta BoNeSi, enquanto os demais *hosts* continuam a enviar tráfego legítimo. Na marca de 30 minutos, o *Host* 2 também troca o envio de tráfego legítimo por ataque (protocolo UDP), assim ficando dois atacantes e três legítimos. Aos 50 minutos de experimento, o *Host* 3 também começa a atacar (protocolo ICMP), deixando três atacantes e dois legítimos. Com 70 minutos, o *Host* 4 introduz tráfego malicioso (protocolo TCP). Tem-se vinte minutos com quatro máquinas infligindo tráfego malicioso e uma legítimo, quando, aos 90 minutos, todas elas cessam os ataques e retornam a aplicar fluxos legítimos. O *Host 5* não ataca em nenhum momento, funcionando como uma variável de controle para a análise do comportamento da classificação. A experiência se encerra na marca de 110 minutos.

No Cenário *TI*, diferentemente do anterior, os experimentos foram feitos com tráfego imutável do início ao fim, em doze experimentos de vinte minutos cada, repetidos para os quatro algoritmos. Cada uma das doze iterações teve uma configuração diferente de tráfego, como mostrado no Quadro 5.1.

Quadro 5.1 – Configuração de tráfegos para os experimentos

Configuração	Ataques de DDoS	Tráfego Normal
1	5 hosts TCP	–
2	5 hosts UDP	–
3	5 hosts ICMP	–
4	5 hosts (3 TCP, 1 UDP, 1 ICMP)	–
5	4 hosts TCP	1 host
6	4 hosts UDP	1 host
7	4 host ICMP	1 host
8	4 hosts (2 TCP, 1 UDP, 1 ICMP)	1 host
9	1 host TCP	4 hosts
10	1 host UDP	4 hosts
11	1 host ICMP	4 hosts
12	–	5 hosts

Fonte: do autor (2023).

A ferramenta usada para realizar os ataques foi a BoNeSi, enquanto o tráfego legítimo foi gerado a partir do *script* desenvolvido para tal finalidade. O objetivo destes experimentos foi verificar como cada algoritmo se comportava em testes estáticos e configurações imutáveis de tráfego.

Ao final de cada um dos cinquenta e dois experimentos, de ambos os cenários (*TV* e *TI*), foi gerado um arquivo de *log* contendo os resultados da classificação dos fluxos. Com os dados de cada

classificação, foi possível realizar uma comparação das métricas dos algoritmos, assim como outros detalhes observados durante o andamento da experimentação, apresentados no capítulo a seguir.

6 RESULTADOS

Neste capítulo, serão apresentados os resultados dos cenários desenvolvidos e discutidos os detalhes de cada atividade experimental realizada a fim de classificar os fluxos.

6.1 Cenário com alterações no tráfego durante os experimentos (*TV*)

A seguir são mostrados e discutidos os resultados dos experimentos que tiveram duração de uma hora e cinquenta minutos, com alterações propositais na geração de tráfego oriundo dos *hosts*.

A Figura 6.1, apresenta quatro gráficos tridimensionais, onde são mostradas as classificações dos fluxos de cada uma das cinco máquinas e do servidor alvo, ao longo dos experimentos. Os números no **eixo Z** representam as máquinas da topologia pelas respectivas **portas físicas** do *switch*-OF às quais estão conectadas. Na porta 9 estava conectado o servidor alvo, enquanto nas outras estavam os *hosts* que geravam tráfego. As máquinas conectadas às portas 10, 17, 18 e 25 iniciavam os experimentos gerando tráfego legítimo e alternavam para ataques ao longo do tempo, como ilustrado na Figura 5.5. A máquina conectada à porta 19 sempre enviou apenas tráfego legítimo. No **eixo Y** tem-se a **classificação** dos fluxos, onde 0 significa legítimo e 1, malicioso. O **eixo X** apresenta o **tempo** do experimento.

Tabela 6.1 – Resultado percentual das métricas

Algoritmo	Accuracy	Precision	Recall	F-score
GB	59,88	43,79	96,57	60,26
DT	70,73	12,32	3,71	5,71
SVM	57,10	26,05	17,78	21,14
NB	80,15	60,60	81,63	69,50

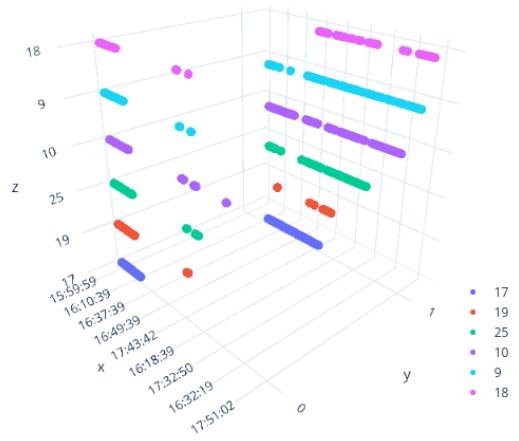
Fonte: do autor (2023).

Os gráficos apresentados na Figura 6.2 mostram o percentual de acertos (TP e TN) e erros (FP e FN) na classificação dos fluxos pelos algoritmos. E na Tabela 6.1, são mostrados os percentuais de cada métrica aplicada aos resultados obtidos pelos algoritmos.

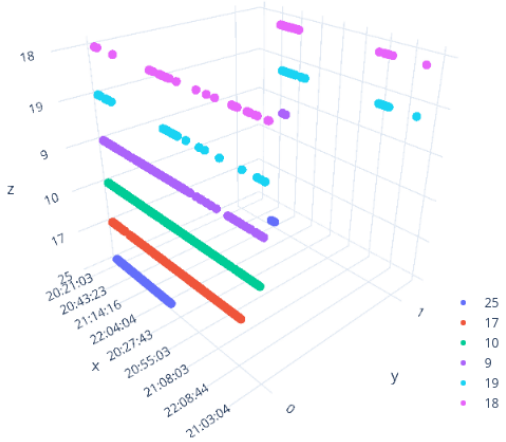
Gradient Boosting - GB

Apesar de ser um algoritmo robusto, o GB não obteve os melhores resultados, tendo um número alto de FP (326). Isso significa, que se fosse aplicado dessa forma em um sistema de defesa contra ataques DDoS, ele bloquearia os fluxos maliciosos, mas prejudicaria o fluxo dos usuários legítimos da rede. A falha do GB nesse experimento acontece depois de dez minutos, quando entram os primeiros fluxos de ataques, pois a partir desse momento, o algoritmo classifica erroneamente praticamente todos os fluxos seguintes como maliciosos. A Tabela 6.1 apresenta um alto valor de *Recall* (96,57%), devido

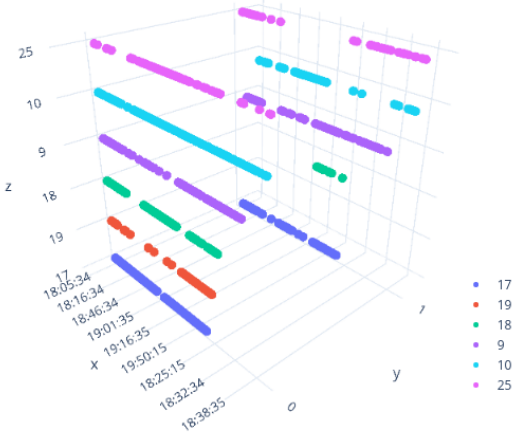
Figura 6.1 – Classificação dos fluxos, por porta física do switch, durante o tempo do experimento.



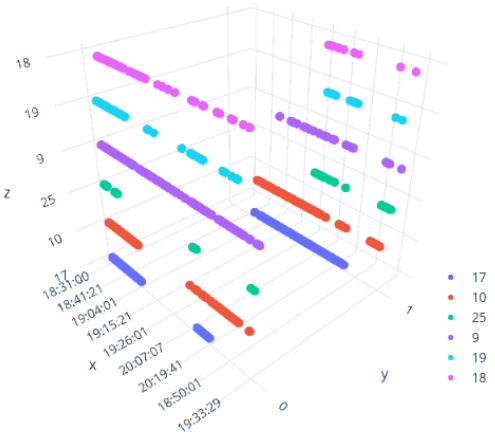
((a)) Gradient Boosting



((b)) Decision Tree



((c)) Support Vector Machine



((d)) Naïve Bayes

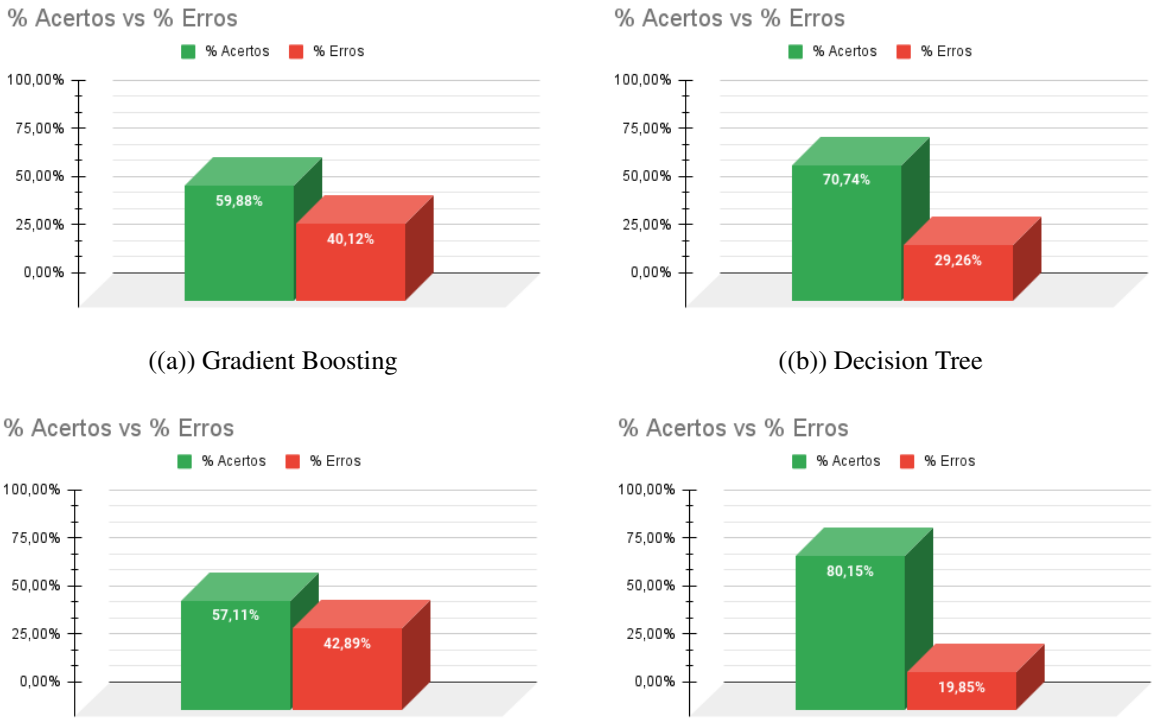
Fonte: do autor (2023).

ao baixo número de FN (9). Para a métrica *Accuracy*, o GB alcançou o valor de 59,88%, o terceiro melhor resultado quando comparado aos outros algoritmos, devido aos seus valores de TP (254) e TN (246). As métricas *Precision* (59,88%) e *F-score* (60,26%) do GB foram inferiores apenas aos valores do algoritmo NB.

***Decision Tree* - DT**

No experimento com o algoritmo DT, ocorreu algo semelhante ao que foi observado no com o GB, mas dessa vez, com alto valor de FN (233) e com um FP de 64. Mesmo com vários ataques entrando em cena, o algoritmo classificou a maioria dos fluxos como legítimos. Ainda, após todas as máquinas atacantes injetarem pacotes maliciosos, o DT classificou todos esses fluxos como legítimos. As métricas

Figura 6.2 – Percentual de acertos e erros na classificação dos fluxos.



((a)) Gradient Boosting

((b)) Decision Tree

((c)) Support Vector Machine

((d)) Naïve Bayes

Fonte: do autor (2023).

Precision, *Recall* e a taxa de *F-score* se mostraram as piores, quando comparadas aos outros algoritmos, com valores de 12,32%, 3,71% e 5,71%, respectivamente (Tabela 6.1). Isso se deve ao alto número de FN. A métrica *Accuracy* obtida pelo DT foi de 70,73%, a segunda melhor entre todos os algoritmos, devido aos números de TP (9) e TN (709).

***Support Vector Machine* - SVM**

Ao utilizar o SVM como algoritmo de classificação, diferentemente dos anteriores, não houve um momento em que marca o início de uma sequência longa de erros. Houve bastante alternância entre as classes 0 (legítimo) e 1 (malicioso) durante todo o experimento. No entanto, esse algoritmo obteve uma taxa de acertos de 57,11% com valores de TP (74) e TN (661). O algoritmo SVM foi inferior em todas as métricas, quando comparado aos algoritmos GB e NB, mas foi superior ao DT em todas as métricas, exceto na *Accuracy*. Esse algoritmo obteve consideráveis números de FN (342) e FP (210), contra um baixo número de TP, acarretando baixas taxas nas métricas analisadas.

***Naïve Bayes* - NB**

No experimento de classificação de fluxos usando o algoritmo NB, assim como na ocasião em que o SVM foi usado, pôde ser observada maior alternância entre as classes 0 e 1. Mas, diferentemente do anterior, o NB apresentou resultados satisfatórios, se mostrando a melhor opção entre as quatro analisadas. No NB o número de FP (156) ficou abaixo do de TP (240), além do número de FN (54) ter sido baixo em relação ao de TN (608). Assim, ao calcular as métricas, os resultados do NB foram superiores aos algoritmos anteriores, exceto à métrica de *Recall* do algoritmo GB, que possui baixo número de FN (9).

O gráfico em três dimensões do experimento com o NB, apresentado na Figura 6.1(d), mostra uma distribuição mais consistente com as ocorrências dos tráfegos, no eixo Y, em comparação com os outros algoritmos. Apesar de ser um algoritmo mais simples que os demais, uma vez que suas suposições de independência condicional entre suas *features* são aparentemente simplificadas (DEVELOPERS, 2020), o NB obteve os melhores resultados neste cenário. O gráfico da Figura 6.2(d) mostra o percentual de fluxos classificados corretamente (80,15%) e erroneamente (19,85%) pelo algoritmo NB.

6.2 Cenário com tráfego imutável durante os experimentos (*TI*)

Adiante são apresentados e discutidos os resultados das experiências com duração de vinte minutos, sem alterações no tráfego durante cada experimento.

***Gradient Boosting* - GB**

Os resultados dos cálculos das métricas dos experimentos de tráfego imutável utilizando o algoritmo GB são apresentados a seguir. Na Tabela 6.2 estão os resultados dos doze experimentos, com os valores das quatro métricas calculadas.

Ao ver os resultados dos cálculos das métricas, podemos observar que, quando há mais máquinas injetando tráfego legítimo do que ataques, o GB apresenta resultados baixíssimos ou nulos. Isso acontece porque a taxa de FN foi muito alta nesses experimentos em que há ocorrência de tráfego legítimo. A taxa de tráfego legítimo gerada pelos computadores pode ter sido alta o suficiente para confundir o algoritmo, que acaba interpretando esse tráfego como ataque.

Tabela 6.2 – Resultados das métricas calculadas para cada experimento dos algoritmos no Cenário *TI*

	GB				DT				SVM				NB			
	Accuracy	**Precision**	**Recall**	**F-score**	**Accuracy**	**Precision**	**Recall**	**F-score**	**Accuracy**	**Precision**	**Recall**	**F-score**	**Accuracy**	**Precision**	**Recall**	**F-score**
1	**0,92**	0,92	**1**	**0,96**	0,56	0,88	0,59	0,71	0,80	**0,98**	0,81	0,89	0,91	0,91	**1**	0,95
2	**0,91**	0,91	**1**	**0,95**	0,56	0,86	0,62	0,72	0,82	**0,98**	0,84	0,90	**0,91**	0,91	**1**	**0,95**
3	**0,92**	**0,92**	**1**	**0,96**	0,54	0,90	0,58	0,70	0,76	**0,98**	0,77	0,87	0,87	0,87	**1**	0,93
4	0,88	0,88	**1**	0,94	0,25	0,75	0,28	0,41	0,88	**0,98**	0,90	0,93	**0,93**	0,93	**1**	**0,97**
5	**0,91**	0,91	**1**	**0,95**	0,28	0,71	0,32	0,43	0,86	**1**	0,86	0,92	0,88	0,88	**1**	0,94
6	0,72	0,72	**1**	0,84	0,15	0,59	0,16	0,25	0,75	**0,94**	0,78	0,85	**0,92**	0,91	**1**	**0,96**
7	0,83	0,83	**1**	0,91	0,54	0,76	0,65	0,70	0,80	**0,96**	0,83	0,89	**0,89**	0,89	**1**	**0,94**
8	**0,93**	0,93	**1**	**0,96**	0,65	0,94	0,67	0,78	0,86	**0,98**	0,87	0,92	0,92	0,92	**1**	**0,96**
9	0,32	0,32	**1**	0,49	0,62	indef	0	indef	0,43	0,06	0,18	0,08	**0,72**	**0,54**	0,81	**0,65**
10	0,32	0,32	**1**	0,48	0,61	indef	0	indef	0,52	0,18	0,60	0,28	**0,69**	**0,53**	0,76	**0,63**
11	0,30	0,30	**1**	0,46	0,53	indef	0	indef	0,44	0,07	0,21	0,11	**0,68**	**0,54**	0,64	**0,58**
12	0	0	0	0	**1**	indef	indef	indef	0,81	0	indef	indef	**1**	indef	indef	indef

Fonte: do autor (2023).

Decision Tree - DT

Os resultados dos cálculos das métricas dos experimentos de tráfego imutável utilizando o algoritmo DT são apresentados a seguir. Na Tabela 6.2 estão os resultados dos doze experimentos, com os valores das quatro métricas calculadas.

Quando calculadas as métricas para o DT, os resultados são piores que os apresentados para o GB. Observando a Tabela 6.2, da mesma forma que no GB, quando há mais máquinas enviando tráfego legítimo do que ataques, os resultados pioram. Nesse caso, algumas métricas não puderam ser calculadas (indef), pois os números de TP e FP foram zero nesses experimentos com mais máquinas legítimas.

Support Vector Machine - SVM

Os resultados dos cálculos das métricas dos experimentos de tráfego imutável utilizando o algoritmo SVM são apresentados a seguir. Na Tabela 6.2 estão os resultados dos doze experimentos, com os valores das quatro métricas calculadas.

Como pode ser observado na Tabela 6.2, os resultados do SVM foram melhores que os do DT na maioria dos casos. Apenas no experimento em que há apenas tráfego legítimo, algumas métricas não puderam ser calculadas, devido ao número zero de FN e TP. Para os experimentos de tráfego misto, o SVM tem resultados semelhantes aos do GB, exceto quando é calculado o *recall*. No geral, o SVM se sai melhor, quando comparado aos resultados do DT.

Naïve Bayes - NB

Os resultados dos cálculos das métricas dos experimentos de tráfego imutável utilizando o algoritmo NB são apresentados a seguir. Na Tabela 6.2 estão os resultados dos doze experimentos, com os valores das quatro métricas calculadas.

Ao observar a Tabela 6.2, onde são apresentados os resultados dos experimentos com o algoritmo NB, podemos interpretar que seus números foram melhores que os apresentados nos outros três algoritmos. No experimento em que há apenas tráfego legítimo sendo injetado, não foi possível calcular as três métricas. No entanto, isso acontece devido ao número de FN e FP ter sido zero, o que significa que o algoritmo classificou com 100% de precisão o tráfego ali analisado.

Considerações finais

Ao final de todos os experimentos e análise dos resultados, foi possível notar que o algoritmo NB obteve melhores resultados que os outros três. Tanto no cenário *TV*, quanto no *TI*, o algoritmo NB foi mais consistente ao classificar corretamente o tráfego malicioso em meio ao tráfego legítimo e vice-versa. No cenário *TI*, os outros algoritmos apresentam alguns números altos, no entanto, isso ocorre ora pela alta presença de FP, ora por muitos FN, que acabam mascarando os resultados. Ainda assim, nota-se que o NB consegue números mais consistentes, principalmente quando o tráfego é misto de malicioso e legítimo. Para o problema abordado nesse projeto, mesmo sendo um algoritmo de construção mais simples, o NB conseguiu ter maior assertividade de TP, FP, TN e FN do que os outros algoritmos abordados. Consequentemente, os resultados calculados pelas métricas aplicadas a esse algoritmo foram melhores que dos outros três.

A seguir é apresentado o capítulo de conclusão deste trabalho, fazendo um apanhado geral e discutindo alguns pontos dos resultados obtidos.

7 CONCLUSÕES E TRABALHOS FUTUROS

Durante o desenvolvimento deste trabalho, a partir do estudo de trabalhos referenciados em capítulos anteriores, foi possível caracterizar alguns pontos para a criação de cenários e experimentos relevantes, visando contribuir com avanços na área de pesquisa para o problema de ataques de DDoS em SDN.

O Cenário *TV* explora o limite dinâmico de pacotes por tempo, pois os ataques variam de intensidade no decorrer do tempo dos experimentos. Também foram explorados ataques de DDoS de múltiplas fontes, visto que foram utilizados vários *hosts* para aplicar os ataques, além da ferramenta BoNeSi já simular os ataques distribuídos. Outra lacuna de pesquisa explorado foi o de que as soluções de segurança deveriam ser implantadas externas ao controlador, para não exigir mais recursos do mesmo. Desta forma, a solução de classificação dos fluxos foi implementada em um *host* separado do controlador.

Diferentes autores apontaram que um controlador SDN pode ser um ponto único de falha e como os ataques de DDoS podem comprometê-lo indiretamente, a partir da inundação de tráfego no plano de dados. Estas características foram abordadas com os ataques direcionados a um servidor no plano de dados, mas inundando o *switch*-OF para o mesmo retransmitir todos os fluxos oriundos dos ataques ao controlador.

No Cenário *TI*, foram aplicados experimentos em que havia maior intensidade de tráfego legítimo do que malicioso, para observar como ocorreria a classificação por parte da solução de aprendizado de máquina. Ainda sobre tipos de ataques, foram explorados os protocolos TCP, UDP e ICMP, direcionados às camadas de rede e transportes do modelo OSI, que têm correlação com o plano de controle da arquitetura SDN.

O problema abordado neste trabalho foi detectar tráfego malicioso oriundo de ataques de DDoS em redes do tipo SDN e seus controladores. Foram criados e disponibilizados *datasets* com dados de tráfego de DDoS em SDN. Vários experimentos foram realizados, com ataques de DDoS em uma SDN real, com *switch*-OF físico. Com a solução implementada, onde se tem apenas duas classes (legítimo e malicioso), dados os atributos de tráfego capturado nos *datasets*, somados à forma como foram feitos cenários e realizados os experimentos, nota-se que um algoritmo de aprendizado de máquina mais simples e popular (NB) conseguiu melhores resultados, identificando mais assertivamente os fluxos legítimos e maliciosos.

A partir desse projeto, será possível realizar um próximo passo, a criação de regras de defesa no controlador para mitigar os ataques. Para isso, além da configuração descrita nesta pesquisa, se faz necessária a programação das regras, para serem ativadas pelo controlador no *switch*-OF, orientando o que fazer com cada fluxo, rejeitando os ataques e permitindo os legítimos.

REFERÊNCIAS

ALADAILEH, M. A. et al. Detection techniques of distributed denial of service attacks on software-defined networking controller–a review. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 143985–143995, 2020.

BANJAR, A. et al. Analysing the performance of the openflow standard for software-defined networking using the omnet++ network simulator. In: **2014 Asia Pacific Conference on Computer Aided System Engineering International Conference on Wireless Networks and Mobile Communications (APCASE)**. [S.l.]: IEEE, 2014.

BENABBOU, J.; ELBAAMRANI, K.; IDBOUFKER, N. Security in OpenFlow-based SDN, opportunities and challenges. **Photonic Network Communications**, Springer Science and Business Media LLC, v. 37, n. 1, p. 1–23, oct 2018.

CARVALHO, L. F. et al. An ecosystem for anomaly detection and mitigation in software-defined networking. **Expert Systems with Applications**, Elsevier BV, v. 104, p. 121–133, aug 2018.

CHICA, J. C. C.; IMBACHI, J. C.; BOTERO, J. F. Security in SDN: A comprehensive survey. **Journal of Network and Computer Applications**, Elsevier BV, v. 159, p. 102595, jun 2020.

CONTRIBUTORS, M. P. **Mininet - An Instant Virtual Network on your Laptop (or other PC)**. 2021. Http://mininet.org. Accessed: June, 2021. Disponível em: <http://mininet.org/>.

DEEPA, V.; SUDAR, K. M.; DEEPALAKSHMI, P. **Detection of DDoS Attack on SDN Control plane using Hybrid Machine Learning Techniques**. 2018.

DEVELOPERS scikit-learn. **scikit-learn Machine Learning in Python**. 2020. Acesso em: 12/2020.

FARHADY, H.; LEE, H.; NAKAO, A. Software-defined networking: A survey. **Computer Networks**, Elsevier BV, v. 81, p. 79–95, apr 2015.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. W. The road to SDN: an intellectual history of programmable networks. **Comput. Commun. Rev.**, v. 44, n. 2, p. 87–98, 2014.

FOUNDATION, O. N. **Software-Defined Networking: The New Norm for Networks**. 2012. Https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/. Accessed: June, 2021. Disponível em: <https://opennetworking.org/sdn-resources/whitepapers/software-defined-networking-the-new-norm-for-networks/>.

GOLDSTEIN, M. **BoNeSi**. 2015. Https://github.com/markus-go/bonesi. Accessed: May, 2021. Disponível em: <https://github.com/markus-go/bonesi>.

JAZI, H. H. et al. Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling. **Computer Networks**, Elsevier BV, v. 121, p. 25–36, jul 2017.

JUPYTER, P. **Jupyter Notebook**. 2023. Https://github.com/jupyter/notebook. Accessed: Jan, 2023. Disponível em: <https://github.com/jupyter/notebook>.

KAUR, K.; SINGH, J.; GHUMMAN, N. Mininet as software defined networking testing platform. In: . [S.l.: s.n.], 2014.

KREUTZ, D.; RAMOS, F. M.; VERISSIMO, P. Towards secure and dependable software-defined networks. In: **Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13**. [S.l.]: ACM Press, 2013.

LI, C. et al. Detection and defense of DDoS attack-based on deep learning in OpenFlow-based SDN. **International Journal of Communication Systems**, Wiley, v. 31, n. 5, p. e3497, jan 2018.

MOUSAVI, S. M.; ST-HILAIRE, M. Early detection of DDoS attacks against software defined network controllers. **Journal of Network and Systems Management**, Springer Science and Business Media LLC, v. 26, n. 3, p. 573–591, sep 2017.

NOMAN, H. M.; JASIM, M. N. POX controller and open flow performance evaluation in software defined networks (SDN) using mininet emulator. **IOP Conference Series: Materials Science and Engineering**, IOP Publishing, v. 881, p. 012102, aug 2020. ISSN 1757-899X.

NOVAES, M. P. et al. Long short-term memory and fuzzy logic for anomaly detection and mitigation in software-defined network environment. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, p. 83765–83781, 2020. ISSN 2169-3536.

NSFOCUS. **2020-Mid-Year DDoS Attack Landscape Report**. 2020. Https://nsfocusglobal.com/2020-mid-year-ddos-attack-landscape-report/. Accessed: May, 2021. Disponível em: <https://nsfocusglobal.com/2020-mid-year-ddos-attack-landscape-report/>.

NSFOCUS. **Global DDoS Attack Landscape H1 2022**. 2022. Https://nsfocusglobal.com/wp-content/uploads/2022/09/H1-2022-Global-DDoS-Attack-Landscape-1.pdf. Accessed: January, 2023. Disponível em: <https://nsfocusglobal.com/wp-content/uploads/2022/09/H1-2022-Global-DDoS-Attack-Landscape-1.pdf>.

NUNES, B. A. A. et al. A survey of software-defined networking: Past, present, and future of programmable networks. **IEEE Communications Surveys & Tutorials**, Institute of Electrical and Electronics Engineers (IEEE), v. 16, n. 3, p. 1617–1634, 2014.

OSTINATO. **Ostinato - Traffic Generator for Network Engineers**. 2021. Https://ostinato.org/. Accessed: May, 2021. Disponível em: <https://ostinato.org/>.

PALIWAL, M.; SHRIMANKAR, D.; TEMBHURNE, O. Controllers in SDN: A review report. **IEEE Access**, Institute of Electrical and Electronics Engineers (IEEE), v. 6, p. 36256–36270, 2018.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

PISSARA, F. L. **T50**. 2021. Https://gitlab.com/fredericopissarra/t50. Accessed: May, 2021. Disponível em: <https://gitlab.com/fredericopissarra/t50>.

POX. **POX Controller**. 2021. Https://github.com/noxrepo/pox. Accessed: May, 2021. Disponível em: <https://github.com/noxrepo/pox>.

POX. **POX Manual**. 2021. Https://noxrepo.github.io/pox-doc/html/. Accessed: May, 2021. Disponível em: <https://noxrepo.github.io/pox-doc/html/>.

PROJECT, M. **MLflow Documentation**. 2022. Https://mlflow.org/docs/latest/index.html. Accessed: Jan, 2023. Disponível em: <https://mlflow.org/docs/latest/index.html>.

RIOREY. **Taxonomy of DDoS Attacks**. 2015. Https://www.riorey.com/riorey-ddos-taxonomy. Accessed: June, 2021. Disponível em: <https://www.riorey.com/riorey-ddos-taxonomy>.

SANTOS, R. et al. Machine learning algorithms to detect DDoS attacks in SDN. **Concurrency and Computation: Practice and Experience**, Wiley, v. 32, n. 16, jun 2019.

SAYED, M. S. E. et al. A flow-based anomaly detection approach with feature selection method against DDoS attacks in SDNs. **IEEE Transactions on Cognitive Communications and Networking**, Institute of Electrical and Electronics Engineers (IEEE), v. 8, n. 4, p. 1862–1880, dec 2022.

SECURITY, H. N. **4.83 million DDoS attacks took place in the first half of 2020, a 15% increase**. 2020. Acesso em: 12/2020. Disponível em: <https://www.helpnetsecurity.com/2020/09/30/4-83-million-ddos-attacks-first-half-of-2020/>.

SEN, S.; GUPTA, K. D.; AHSAN, M. M. Leveraging machine learning approach to setup software-defined network(SDN) controller rules during DDoS attack. In: **Proceedings of International Joint Conference on Computational Intelligence**. [S.l.]: Springer Singapore, 2019. p. 49–60.

SHARAFALDIN, I. et al. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In: **2019 International Carnahan Conference on Security Technology (ICCST)**. [S.l.]: IEEE, 2019.

SINGH, J.; BEHAL, S. Detection and mitigation of DDoS attacks in SDN: A comprehensive review, research challenges and future directions. **Computer Science Review**, Elsevier BV, v. 37, p. 100279, aug 2020.

SRIDHARAN, V.; GURUSAMY, M.; LEON-GARCIA, A. **Anomalous Rule Detection using Machine Learning in Software Defined Networks**. [S.l.]: IEEE, 2019.

SUDAR, K. et al. Detection of distributed denial of service attacks in SDN using machine learning techniques. IEEE, jan 2021.

TANG, T. A. et al. Deep learning approach for network intrusion detection in software defined networking. In: **2016 International Conference on Wireless Networks and Mobile Communications (WINCOM)**. [S.l.]: IEEE, 2016.

Wireshark Foundation. **About Wireshark**. 2021. Https://www.wireshark.org/. Accessed: May, 2021. Disponível em: <https://www.wireshark.org/>.

XIA, W. et al. A survey on software-defined networking. **IEEE Communications Surveys & Tutorials**, Institute of Electrical and Electronics Engineers (IEEE), v. 17, n. 1, p. 27–51, 2015.

YAN, Q. et al. Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges. **IEEE Communications Surveys & Tutorials**, Institute of Electrical and Electronics Engineers (IEEE), v. 18, n. 1, p. 602–622, 2016.

APÊNDICE A

Dicionário de dados dos atributos capturados e calculados nos *datasets*.

1. portaSwitch: inteiro - Porta do *switch* em que ocorreu o evento de entrar um pacote de rede e não tem uma regra de fluxo na *flowtable* para tratar esse novo fluxo.

2. macOrigemPacote: *string* - Endereço MAC de origem do pacote de rede que entrou no *switch*. Esse MAC é coletado quando um novo pacote de rede entra no *switch*, não há regras de fluxo para tratar e, é encaminhado para o controlador. É usado para identificar a máquina de origem do fluxo.

3. macDestinoPacote: *string* - Endereço MAC de destino do pacote de rede que entrou no *switch*. Esse MAC é pego quando um novo pacote de rede entra no *switch*, não há regras de fluxo para tratar e, é encaminhado para o controlador. É usado para identificar a máquina destino do fluxo.

4. ARPop: *string* - Tipo de operação que está sendo feita com o protocolo ARP, as operações monitoradas são REQUEST, REPLY, REV_REQUEST e REV_REPLAY. Esse atributo contém o registro de diversas requisições que podem ocorrer para uma mesma máquina. O *parsing* desse campo pode gerar um novo atributo.

5. ARPhwtypw: inteiro - Tipo de *hardware* que se origina o pacote ARP. Cada tipo de equipamento possui um identificador para o tipo de *hardware*. Ex: equipamentos com Ethernet = 0x0001.

6. ARPprototype: inteiro - Tipo de protocolo ARP, é mais comum se utilizar o protocolo IP. Ex: Protocolo IP = 0x0800.

7. ARPhwsrc: *string* - Endereço MAC da origem do pacote ARP.

8. ARPhwdst: *string* - Endereço MAC do destino do pacote ARP.

9. ARPprotosrc: *string* - Endereço IP de origem do pacote ARP.

10. ARPprotodst: *string* - Endereço IP que se destina o pacote ARP.

11. IPprotocol: *string* - Tipo do protocolo com o IP, se é IP+TCP, IP+UDP ou IP+ICMP. Este atributo contém o registro dos tipos de pacotes IP que entraram no controlador. Este atributo combinado com outros pode permitir contar o número de pacotes TCP/UDP em uma conexão. O *parsing* desse campo pode gerar um novo atributo.

12. IPsrcip: *string* - Endereço IP de origem.

13. IPdstip: *string* - Endereco IP de destino.

14. IPversao: inteiro - Número que indica se o protocolo IP está na versão 4 ou 6.

15. IPhl: inteiro - Número que indica o tamanho do cabeçalho do protocolo IP. Esse número varia conforme a quantidade de campos ativos no cabeçalho.

16. IPiplen: inteiro - Tamanho do protocolo IP.

17. IPttl: inteiro - Tempo de vida do protocolo IP. Esse campo registra os TTLs dos fluxos novos que serão adicionados na *flowtable*. Examinar esse campo pode permitir a identificar se os pacotes podem ter sido criados artificialmente por uma ferramenta ou padrão fora do normal. O *parsing* desse campo pode gerar um novo atributo.

18. UDPsrcport: *string* delimitada - Endereço de origem de um pacote UDP. Esse campo armazena as portas que aparecem no fluxo de uma mesma máquina, assim vai conter o registro de mais de uma porta. Examinar as portas registradas nesse campo permite identificar se há uma variação nas portas atacadas, se é apenas uma porta, se são portas altas menos privilegiadas ou portas baixas. O *parsing* desse campo pode gerar um novo atributo.

19. UDPdstport: *string* delimitada - Endereço de destino de um pacote UDP. Esse campo armazena as portas que aparecem no fluxo de uma mesma máquina, assim vai conter o registro de mais de uma porta. Examinar as portas registradas nesse campo permite identificar se há uma variação nas portas atacadas, se é apenas uma porta, se são portas altas menos privilegiadas ou portas baixas. O *parsing* desse campo pode gerar um novo atributo.

20. UDPlen: inteiro - Tamanho do pacote UDP.

21. ICMPtipo: *string* - O tipo do protocolo ICMP, esse campo tem relação quanto as mensagens enviadas por esse protocolo como ECHO_REPLY, REDIRECT, DEST_UNREACH e etc. Esse campo armazena os tipos de mensagens em um mesmo fluxo, assim pode haver mais de um tipo ou repetido dentro desse mesmo campo. Examinar esse campo pode permitir identificar anomalias no tráfego (DoS). O *parsing* desse campo pode gerar um novo atributo.

22. ICMPcodigo: inteiro - Código que identifica se houve algum erro no datagrama ICMP. Esse código não tem relação com o tipo do ICMP. Caso haja muitos erros nos datagramas que podem ser gerados, isso pode ser um indício de um erro na rede ou sendo gerados por uma ferramenta de *fuzzing*. O *parsing* desse campo pode gerar um novo atributo.

23. ICMPid: inteiro - ID do pacote ICMP, esse campo foi adicionado pelo POX.

24. ICMPsequencia: inteiro - Campo usado para fazer o controle da sequência dos pacotes ICMP, esse campo foi adicionado pelo POX.

25. TCPlenpkt: inteiro delimitado - Esse campo registra o tamanho em *bytes* do protocolo TCP. Esse campo pode conter mais de um tamanho por registrar os pacotes que passam em um mesmo fluxo de dados. O *parsing* desse campo pode gerar um novo atributo.

26. TCPsrcport: inteiro delimitado - Esse campo registra as portas de origem do protocolo TCP. Esse campo pode conter mais de uma porta por registrar os pacotes que passam em um mesmo fluxo de dados. O *parsing* desse campo pode gerar um novo atributo.

27. TCPdstport: inteiro delimitado - Esse campo registra as portas de destino do protocolo TCP. Esse campo pode conter mais de uma porta por registrar os pacotes que passam em um mesmo fluxo de dados. O *parsing* desse campo pode gerar um novo atributo.

28. TCPseq: inteiro delimitado - Esse campo registra a sequência dos pacotes TCP que passam em um mesmo fluxo. Se os pacotes apresentarem uma sequência estranha pode indicar um comportamento anômalo na rede. O *parsing* desse campo pode gerar um novo atributo.

29. TCPack: inteiro - Esse campo registra o ACK de resposta enviado pelo servidor.

30. TCPf: *string* - Esse campo registra as *flags* do pacote TCP: SYN=S, ACK=A, FIN=F, RST=R, PSH=P, URG=U, ECN=E, CWR=C. Devido à abundância de fluxos esse campo é passível de *parsing* para gerar novos dados.

31. TCPops - Significa "todas as opções", conforme o código do POX.

32. VLANid - ID da VLAN.

33. VLANpcp - Prioridade de quadros.

34. VLANeth_type - Tipo de VLAN Ethernet.

35. PSrx_over_err: inteiro - Número de pacotes com RX *overrun*.

36. PStx_dropped: inteiro - Número de pacotes descartados no TX.

37. PSrx_packets: inteiro - Número de pacotes recebidos na rede.

38. PSrx_frame_err: inteiro - Número de erros no alinhamento dos *frames*.

39. PSrx_bytes: inteiro - Número de *bytes* recebidos.
40. PStx_errors: inteiro - Número de erros ao transmitir um pacote
41. PSrx_crc_err: inteiro - Número de erros de CRC.
42. PScollisions: inteiro - Número de colisões.
43. PSrx_errors: inteiro - Número de erros recebidos.
44. PStx_bytes: inteiro - Número de *bytes* transmitidos.
45. PSrx_dropped: inteiro - Número de pacotes descartados pelo RX.
46. PStx_packets: - Número de pacotes transmitidos.
47. PSport_no: inteiro - Número da porta.
48. FSduration_nsec: inteiro - Tempo de duração em que o fluxo está ativo na escala de nanosegundos.
49. FSpriority: inteiro - Prioridade da entrada do fluxo, é significativo quando não tem um *match* exato.
50. FSidle_timeout: inteiro - Número de segundos em espera antes da expiração do fluxo.
51. FScookie: inteiro - Identificador usado pelo controlador.
52. FStable_id: inteiro - Identificador de qual tabela a que o fluxo pertence.
53. FSpacket_count: inteiro - Número de pacotes de um fluxo.
54. FShard_timeout: inteiro - Número de segundos antes da expiração do fluxo.
55. FSbyte_count: inteiro - Número de *bytes* de um fluxo.
56. FSduration_sec: inteiro - Duração do fluxo em segundos.
57. FSdl_type: inteiro - Número que identifica o tipo de internet *frame*.
58. FSnw_dst: *string* - Endereço IP do destino.
59. FSdl_vlan_pcp - Prioridade de quadros.
60. FSdl_src: *string* - Endereço MAC de origem do fluxo.
61. FSnw_proto: *string* - Protocolo IP ou os 8 primeiros *bits* de um código ARP.

62. FSnw_tos: inteiro - Serve para identificar o tipo de serviço.
63. FStp_dst: inteiro - Porta TCP/UDP de destino.
64. FStp_src: inteiro - Porta TCP/UDP de origem.
65. FSdl_dst: *string* - Endereço MAC de destino do fluxo.
66. FSdl_vlan: inteiro - Identificador da VLAN.
67. FSnw_src: *string* - Endereço IP de origem.
68. FSin_port: inteiro - Porta de entrada do *switch*.
69. FSmax_len - Tamanho máximo de fluxos.
70. FStype - Tipo de fluxos.
71. FSport - Porta de fluxos.
72. CALCrx_over_err: inteiro - Número de pacotes com RX *overrun*.
73. CALCtx_dropped: inteiro - Número de pacotes descartados no TX.
74. CALCrx_packets: inteiro - Número de pacotes recebidos na rede.
75. CALCrx_frame_err: inteiro - Número de erros no alinhamento dos *frames*.
76. CALCrx_bytes: inteiro - Número de bytes recebidos.
77. CALCtx_errors: inteiro - Número de erros ao transmitir um pacote
78. CALCrx_crc_err: inteiro - Número de erros de CRC.
79. CALCcollisions: inteiro - Número de colisões.
80. CALCrx_errors: inteiro - Número de erros recebidos.
81. CALCtx_bytes: inteiro - Número de bytes transmitidos.
82. CALCrx_dropped: inteiro - Número de pacotes descartados pelo RX.
83. CALCtx_packets: inteiro - Número de pacotes transmitidos.
84. CALCport_no: inteiro - Número da porta.

85. CALCduration_nsec: inteiro - Tempo de duração em que o fluxo está ativo na escala de nanossegundos.
86. CALCpriority: inteiro - Prioridade da entrada do fluxo, é significativo quando não tem um *match* exato.
87. CALCidle_timeout: inteiro - Número de segundos em espera antes da expiração do fluxo.
88. CALCcookie: inteiro - Identificador usado pelo controlador.
89. CALCtable_id: inteiro - Identificador de qual tabela a que o fluxo pertence.
90. CALCpacket_count: inteiro - Número de pacotes de um fluxo.
91. CALChard_timeout: inteiro - Número de segundos antes da expiração do fluxo.
92. CALCbyte_count: inteiro - Número de *bytes* de um fluxo.
93. CALCduration_sec: inteiro - Duração do fluxo em segundos.
94. CALCdl_type: inteiro - Número que identifica o tipo de internet *frame*.
95. CALCdl_vlan_pcp
96. CALCnw_proto: *string* - Protocolo IP ou os 8 primeiros bits de um código ARP.
97. CALCnw_tos: inteiro - Possibilita identificar o tipo de serviço.
98. CALCtp_dst: inteiro - Porta TCP/UDP de destino.
99. CALCtp_src: inteiro - Porta TCP/UDP de origem.
100. CALCdl_vlan: inteiro - Identificador da VLAN.
101. CALCin_port: inteiro - Porta de entrada do *switch*.
102. CALCactionsmax_len - Tamanho máximo de ações.
103. CALCactionstype - Tipo de ações.
104. CALCactionsport - Porta de ações.