VICTOR GRUDTNER BOELL

IDENTIFICAÇÃO DE TRÁFEGO MALICIOSO EM REDES SDN COM TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL

LAVRAS – MG

2021

VICTOR GRUDTNER BOELL

IDENTIFICAÇÃO DE TRÁFEGO MALICIOSO EM REDES SDN COM TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL

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

Prof. DSc. Luiz Henrique Andrade Correia
Orientador

Prof. DSc. Erick Galani Maziero
Coorientador

LAVRAS – MG
2021

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

Boell, Victor Grudtner.
Identificação de Trafego Malicioso em Redes SDN com Técnicas de Inteligência Computacional / Victor Grudtner Boell. - 2021.
55 p. : il.

Orientador(a): Luiz Henrique Andrade Correia.
Coorientador(a): Erick Galani Maziero.
Dissertação (mestrado profissional) - Universidade Federal de Lavras, 2021.
Bibliografia.

1. Rede definida por software. 2. Segurança. 3. Inteligência computacional. I. Andrade Correia, Luiz Henrique. II. Maziero, Erick Galani. III. Título.

VICTOR GRUDTNER BOELL

IDENTIFICAÇÃO DE TRÁFEGO MALICIOSO EM REDES SDN COM TÉCNICAS DE INTELIGÊNCIA COMPUTACIONAL

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

APROVADA em 28 de Abril de 2021.

Prof. DSc. Hermes Pimenta de Moraes Júnior	UFLA
Prof. DSc. Alex Borges Vieira	UFJF
Prof. DSc. Arthur de Miranda Neto	UFLA

Prof. DSc. Luiz Henrique Andrade Correia
Orientador

Prof. DSc. Erick Galani Maziero
Co-Orientador

LAVRAS – MG
2021

Dedico esse trabalho a Deus.

AGRADECIMENTOS

Agradeço a Deus pelo dom da vida. À minha família, meus amigos e aos meus orientadores.

"O sentido de uma pessoa, coisa ou situação, não pode ser dado. Tem que ser encontrado pela própria pessoa - mas não dentro dela, porque isto iria contra a lei da autotranscedência do existir humano."

(Viktor Frankl)

RESUMO

A quantidade de ataques cibernéticos vem aumentando e tornando-se cada vez mais complexos. Assim, se faz necessário adotar medidas de segurança para a proteção dos ativos de informática. Para tomar uma decisão sobre como solucionar um problema ou monitorar um determinado equipamento de rede é necessário uma grande quantidade de informações. Essas informações podem ser obtidas do tráfego de dados e dos equipamentos da rede e, mesmo com as informações obtidas, é possível que os ataques mais sofisticados passem despercebidos aos olhos dos administradores mais experientes e das medidas de segurança como *firewalls* e antivírus. A tecnologia de redes definidas por software (SDN - *Software Defined Network*) permite ter uma visão completa da rede. Além disso, o processamento do tráfego pode ser feito por meio de um módulo externo ao controlador para criar as regras que definem os caminhos dos pacotes na rede. Este trabalho pretende utilizar as características da tecnologia SDN para monitorar a rede a fim de obter dados dos equipamentos e classificar o comportamento do tráfego anômalo utilizando inteligência artificial (IA - Inteligência Artificial). O resultado do processamento será usado para criar regras para o controlador da rede mitigar os problemas de segurança.

Palavras-chave: Rede definida por software. segurança. inteligência computacional

ABSTRACT

The amount of cyber attacks is increasing and becoming more and more complex. Thus, it is necessary to adopt security measures to protect computer assets. To make a decision on how to solve a problem or monitor a particular network equipment, a great deal of information is needed. This information can be attributed to data transfer points and network equipment, even with the information obtained, it is possible that the most sophisticated data go unnoticed in the eyes of more experienced administrators and security measures such as *firewalls* and antivirus. (SDN - *Software Defined Network*) technology gives you a complete view of your network. In addition, traffic processing can be done through a module external to the driver to create rules that define the paths of packets in the network. This work intends to use as characteristics of SDN technology to monitor the network in order to obtain equipment data and classify the behavior of anomalous traffic using artificial intelligence (AI - Artificial Intelligence). The result of the processing will be used to create rules for the network controller to mitigate security issues.

Keywords: Software-defined network. safety. computational intelligence

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A área de redes de computadores é bem desenvolvida e consolidada nos protocolos e equipamentos de rede. Alterar uma determinada infraestrutura para trazer novas melhorias por causa do crescente volume de usuários exige um grande esforço e consumo de recursos. Assim, surge como uma alternativa para a alteração da infraestrutura de rede a Rede Definida por Software (SDN - *Software Defined Network*) (FOUNDATION, 2012b), a qual é uma evolução para área de redes de computadores. Essa evolução está no fato de implementar o plano de controle separado do plano de dados, no qual o processamento do tráfego é feito no controlador da rede e os equipamentos de rede fazem apenas o *link* de comunicação (Scott-Hayward; Natarajan; Sezer, 2016). Na prática isso permite fazer o controle dos tráfegos da rede em um computador distinto dos equipamentos de interconexão entre as redes. Dessa forma, é possível ter uma maior quantidade de dados passando entre os *links*, pois grande parte do processamento dos fluxos que definem as rotas da rede não fica mais nesses equipamentos (SOKOLOV et al., 2014).

O protocolo Openflow foi desenvolvido para fazer a comunicação entre o *switch* e o controlador da rede (MCKEOWN et al., 2008). Pelo protocolo Openflow são enviados comandos que manipulam as *flow tables*, essas tabelas ficam dentro do *switch* Openflow e armazenam as regras de fluxos, determinando se os mesmos serão bloqueados ou não. Os comandos que são enviados pelo controlador ou *switch* Openflow servem para deletar, atualizar e adicionar regras nas tabelas. Além dessas funções, o Openflow permite adquirir estatísticas da rede direto dos *switches* que suportam esse protocolo. Assim, de posse dessas informações é possível implementar programas auxiliares ao plano de controle para processar e disponibilizar novas informações para os administradores da rede (FOUNDATION, 2012b).

Em relação à rede de computadores tradicional, a SDN possui algumas vantagens, a saber, visão global da rede, mecanismo de autorrecuperação, criação de regras condicionais e uma maior capacidade de controle do fluxo (Dabbagh et al., 2015). Entretanto, a implementação da SDN possui falhas que permitem ataques ao plano de dados, plano de controle e plano de aplicação, como ataques de negação de serviço, comprometimento do controlador e ataque de "homem no meio" (Liu et al., 2019), (Scott-Hayward; Natarajan; Sezer, 2016), (Dabbagh et al., 2015). Portanto, é necessário dar ênfase na segurança da SDN para não comprometer seus usuários.

Os objetivos da área de segurança da informação são garantir a disponibilidade, integridade, autenticidade, confidencialidade e determinação de responsabilidade na utilização dos ativos de informática ou nos dados gerados pelos mesmos durante a sua manipulação ou desenvolvimento (BROWN; STALLINGS, 2017). A violação de algum desses aspectos pode trazer prejuízos para empresas ou pessoas, assim, empregar esforços para que nenhum dos aspectos acima citados sejam violados é de

vital importância para uma empresa. Uma maneira de melhorar a segurança da informação é por meio de coleta de dados dos meios computacionais de forma automática para serem analisados. Porém, um grande volume de dados pode tornar essa tarefa de análise impossível para as pessoas e um método para contornar esse problema é utilizar ferramentas de análise automática dos dados.

A análise de redes de computadores se concentra em utilizar métodos para obter informações sobre o fluxo de dados entre as máquinas ou servidores, os dados obtidos são processados para obter um mapa conceitual da rede ou a possível identificação de anomalias no tráfego (NETO; LACERDA; BOELL, 2015). Para tal tarefa, podem ser usadas técnicas de Aprendizado de Máquina (ML - *Machine Learning*) que implementam algoritmos para análise de grandes volumes de dados para obter novas informações (padrões) que não eram visualizadas por meio da análise dos dados brutos. É importante observar que os resultados da análise devem emitir o menor número de falsos positivos e um maior número de verdadeiros positivos (NETO, 2017).

No tráfego da rede, uma anomalia pode ser qualquer coisa que fuja do padrão de uso normal da rede. Porém, existem diversos casos que podem ser classificados como anomalias e que não necessariamente são ataques à rede, como por exemplo um *switch* com defeito. Um ataque comum hoje em dia é o de negação de serviço distribuído (DDoS - *Distributed Denial of Service*), no qual a rede ou aplicação fica temporariamente indisponível para os seus usuários (Nagpal et al., 2015). O autor em (Lim et al., 2014), mostra um esquema de proteção contra ataques DDoS causados por *botnets*[1], no qual controlador monitora textitswitces Openflow para redirecionar as requisições para uma aplicação avaliá-la. Se a requisição for verdadeira ela retorna para o controlador que envia para os *switches* o endereço do servidor, caso contrário a requisição é bloqueada.

Outro exemplo de uso da SDN para monitorar o comportamento de uma rede é descrito em (HADIANTO; PURBOYO, 2018), no qual foi simulado uma rede infectada pelo trojan Zeus para analisar uma *botnet*. As conexões monitoradas mostraram um padrão no comportamento da conexão e no tamanho dos pacotes de rede. Quando uma máquina está infectada por um vírus, os dados capturados por meio de *firewalls* ou antivírus instalados localmente na máquina, podem estar manipulados para esconder a sua presença. A SDN permite redirecionar o tráfego da máquina infectada para ser analisado sem a interferência do *malware*, caso o tráfego fosse capturado no host infectado (GAO et al., 2018).

[1] Uma *botnet* é uma rede privada de máquinas formada por um vírus de computador, e essas máquinas executam comandos enviados por um centro de controle que monitora a *botnet*.

1.1 Definição do Problema

O problema abordado neste trabalho é identificar os tráfegos maliciosos gerados com ferramentas de testes de segurança de redes entre os hosts dentro de uma SDN, e encerrar essa comunicação por meio de regras criadas automaticamente pelo controlador com base nos resultados de um modelo de ML.

1.2 Motivação

Identificar se um tráfego de rede é malicioso ou não, é crucial para o bom funcionamento e a segurança dos usuários de uma determinada rede de computadores. Para isso, a classificação do tráfego tem um papel fundamental nesta tarefa. O volume e a variedade de características que existem em um tráfego de rede se torna uma vantagem na utilização de técnicas de ML para classificar o tráfego da rede. Combinando as características da SDN e técnicas de ML permite criar regras automaticamente pelo controlador SDN e mitigar as anomalias do tráfego.

1.3 Hipóteses

Este trabalho visa responder as seguintes questões:

1. Dado um alto volume de tráfego de uma SDN é possível detectar anomalias em tempo hábil?
2. O controlador consegue definir regras para bloquear as anomalias sem prejudicar o tráfego da rede?
3. Os *datasets* existentes e disponíveis representam os atuais ataques nas SDNs?
4. As pesquisas atuais permitem identificar a variabilidade dos ataques atuais nas redes?

1.4 Objetivos

O objetivo principal deste trabalho, é analisar um conjunto de técnicas de ML e avaliar qual melhor se aplica ao problema de análise do tráfego de rede dentro da SDN, para identificar anomalias e bloquear suas ações por meio de regras criadas automaticamente pelo controlador SDN. As regras são criadas com base nos resultados de classificação de um modelo de ML acoplado ao controlador SDN. Os objetivos específicos estão divididos nos tópicos a seguir.

1. Identificar as características e os parâmetros mais importantes de uma SDN para o monitoramento de redes.

2. Avaliar os métodos de ML supervisionados e não supervisionados mais adequados ao problema de análise de tráfego malicioso de rede.

3. Implementar um software para capturar o tráfego de rede e criar um *dataset* para utilização das técnicas de inteligência computacional.

4. Gerar modelos com os métodos de ML utilizando o *dataset*.

5. Acoplar ao controlador de rede o melhor modelo de ML para classificar o tráfego da rede e gerar regras de fluxos com base nesse resultado.

1.5 Organização do trabalho

Este trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os principais conceitos usados para o desenvolvimento desta pesquisa. São descritos conceitos sobre SDN, *switch*, controladores de SDN, ML, segurança da informação, análise do tráfego de rede e os trabalhos relacionados usados como base desta pesquisa. O Capítulo 4 apresenta a metodologia para execução desta pesquisa, no qual se descreve os equipamentos utilizados, o simulador Mininet, a arquitetura da rede, a forma de captura do tráfego de redes, a construção da base de dados (dataset), os ataques a rede, a análise dos resultados, o cronograma e as considerações finais. O Capítulo 5 apresenta os resultados dos experimentos. O capítulo 6 mostra a conclusão desta pesquisa e as propostas de continuidade.

2 REFERENCIAL TEÓRICO

2.1 Referencial teórico

Esta seção mostra uma descrição dos principais conceitos e técnicas abordadas neste trabalho. Também são mostrados os trabalhos relacionados que fundamentam a elaboração desta pesquisa, ajudando identificar as diferenças e contribuições deste trabalho.

2.2 Redes Definidas por Software

Uma rede de computador tradicional é composta por dois ou mais dispositivos conectados entre si, usando os mesmos conjuntos de protocolos para a comunicação e um meio físico para a troca de mensagens. Os equipamentos de conexão entre esses dispositivos podem ser *switches* ou roteadores, no qual armazenam as informações necessárias das rotas da rede (TANENBAUM, 2011). A SDN é uma nova modalidade de arquitetura que permite organizar as rotas da rede de uma forma mais flexível, obter dados estatísticos das conexões de rede, ter o controle centralizado da rede e uma visão global. Essas características são obtidas por meio da separação do plano de dados do plano de controle, permitindo que o processamento das rotas e as estatísticas dos fluxos sejam feitos em um hardware externo aos *switches* e roteadores da rede, simplificando suas funções. Com o uso do protocolo OpenFlow também é possível fazer a comunicação do controlador da rede com os elementos de rede (MCKEOWN et al., 2008).

A ONF (*Open Network Foundation*) propôs uma arquitetura SDN simplificada, a qual é dividida em três camadas, sendo elas o plano de controle, o plano de dados e o plano de aplicação. Entre as camadas existem interfaces e protocolos para fazer a comunicação entre as mesmas, também sendo divididas em três temos a D-CPI (Data-Controller Plane Interface), a A-CPI (*Application-Controller Interface*) e a MI (*Managment Interface*) (Mendiola et al., 2017). De acordo com o trabalho (Mendiola et al., 2017) e (FOUNDATION, 2014), cada uma dessas interfaces desempenha as seguintes funções:

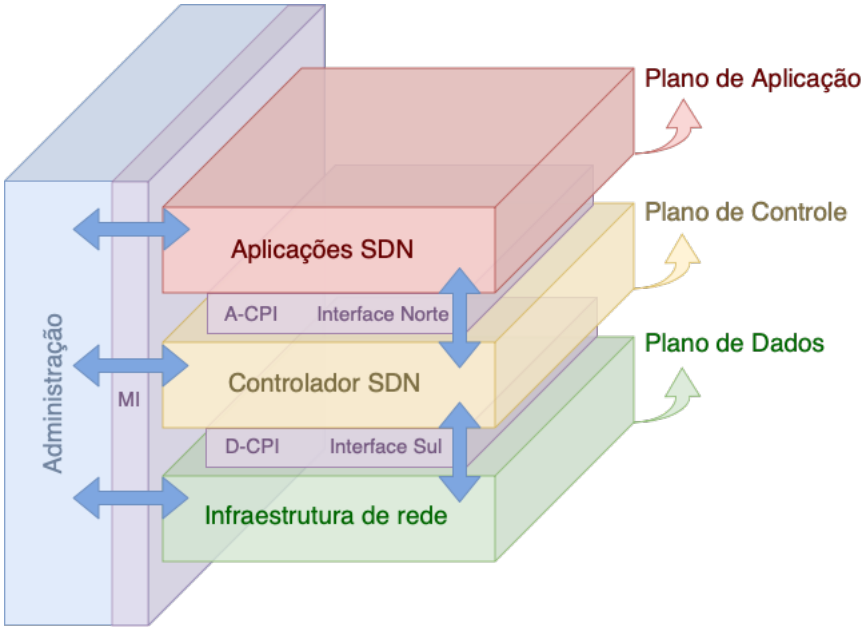
- **D-CPI:** Os protocolos dessa interface fazem a comunicação entre os planos de dados e controle, no qual os elementos da rede provêm um conjunto de recursos e operações que podem ser executadas com uso de protocolos no plano de dados. São rápidos ao ponto de ativar e desativar os circuitos do plano de dados em milissegundos.

- **A-CPI:** Os protocolos dessa interface fazem a comunicação entre os planos de controle e aplicação. Eles fornecem uma abstração dos elementos da rede e um meio de manipulá-los.

- **MI:** Os protocolos dessa interface fazem a comunicação e administração dos três planos da arquitetura SDN.

No caso da divisão da arquitetura em planos, a Figura 2.1 mostra um exemplo simplificado dessa divisão da arquitetura SDN, sendo composta pelas aplicações dentro do plano de aplicações, o controlador dentro do plano de controle e a infraestrutura de rede dentro do plano de dados. De acordo em (FOUNDATION, 2013), cada componente tem a seguinte função:

Figura 2.1 – Arquitetura SDN simplificada.



Fonte: Adaptado de (FOUNDATION, 2013).

- **Plano de Aplicações:** É a camada onde se encontram as aplicações escritas em alto nível para processarem os dados vindos do controlador. Nessa camada se concentra a lógica da SDN, conhecendo parcialmente a arquitetura da rede e o seu comportamento.
- **Plano de Controle:** É a camada entre as aplicações SDN e a infraestrutura de rede, essa camada fornece dados para aplicações formando uma visão abstrata da rede, também pode fornecer dados estatísticos da rede. Uma SDN pode possuir um ou mais controladores que podem estar organizados de forma hierárquica.
- **Plano de Dados:** É a camada onde se encontra os caminhos lógicos da rede que foram definidos no controlador, esses caminhos ficam armazenados na memória do *switch* Openflow e podem existir um ou mais caminhos por dispositivo.
- **Interface Sul:** É uma interface entre o controlador e o plano de dados, ela é responsável por prover um controle sobre as operações de *forwarding*, anúncio, reportar estatísticas e notificações de eventos. Nessa interface se encontra o D-CPI.

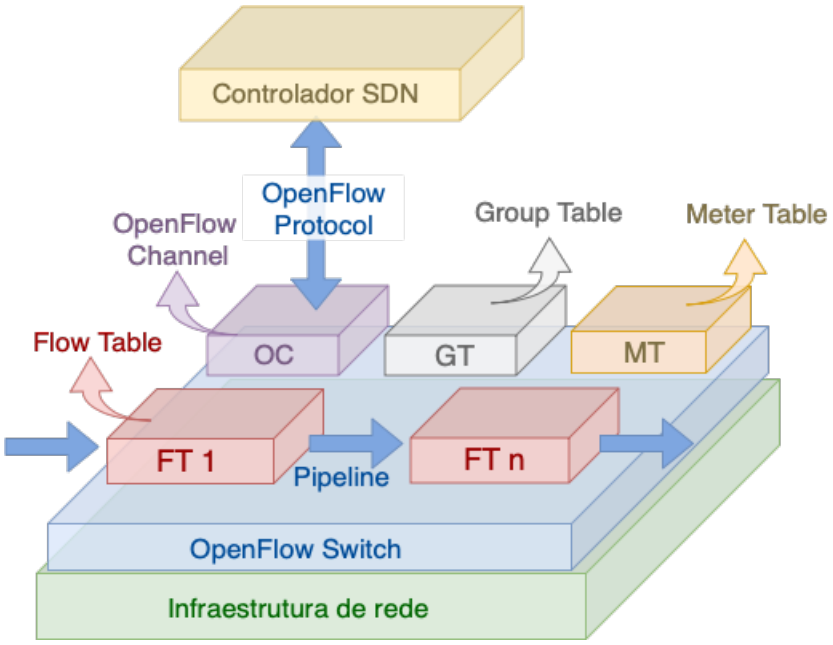
- **Interface Norte:** É uma interface entre a camada de aplicação e o controlador, ela provê uma visão abstrata da rede para a camada de aplicação e comunicação com o controlador por meio do protocolo OpenFlow. Nessa interface se encontra o A-CPI.

O protocolo usado para fazer a comunicação entre os planos e as interfaces com o controlador da rede é o Openflow. Ele atua na interface D-CPI e permite fazer a requisição dos fluxos armazenados no *switch* Openflow, coletar os dados estatísticos de cada fluxo e definir as rotas entre um ou mais equipamentos da rede (Mendiola et al., 2017), (FOUNDATION, 2014).

2.3 Switch Openflow

O *switch* Openflow é um equipamento que possui mecanismos para o funcionamento do protocolo Openflow, entretanto, existem dois tipos de *switche*, o Openflow puro e o híbrido. No Openflow puro os pacotes de rede são processados dentro do *pipeline* de execução do Openflow. No híbrido os pacotes de rede podem ser processados dentro do *pipeline* do Openflow ou fora dele como em um *switch* normal (FOUNDATION, 2012a). A Figura 2.2 mostra um esquema simplificado de um *switch* Openflow, no qual é composto por uma ou mais *flow table*, uma *group table*, uma *meter table* e um ou mais *OpenFlow channel* (FOUNDATION, 2012a).

Figura 2.2 – Esquema simplificado de um *switch* openflow.



Fonte: Adaptado de (FOUNDATION, 2012a).

- **Flow Table:** Fica armazenada dentro do switch e ela contém regras dos fluxos, no qual os pacotes que vão entrado no switch são comparados com essas regras, caso tenha alguma corres-

pondência uma ação é tomada, caso contrário o pacote é enviado ao controlador para tomar uma decisão após algum determinado processamento.

- **Group Table:** É um conjunto de *Flow Tables* usadas para processar fluxos ou pacotes de uma maneira mais complexa e que não poderia ser feito apenas com uma *Flow Table*.
- **Meter Table:** É uma tabela onde se armazenam os dados do monitoramento de um fluxo da rede. Esses dados são constituídos de número de pacotes e são usados para calcular o QoS (*Quality of Service*) da rede.
- **OpenFlow Channel:** É o canal de comunicação entre o *switch* e o controlador da rede usando o protocolo OpenFlow, por esse canal o controlador envia comandos para salvar, deletar e atualizar as regras da *flow table*.

2.4 Controladores

Os controladores de uma SDN ficam no plano de controle, que é uma camada entre o plano de dados e aplicação, são por meio deles que é feito a comunicação entre os equipamentos de rede e as aplicações SDN. O controlador é responsável por encaminhar comandos para os equipamentos de rede usando o protocolo Openflow, assim é possível fazer o gerencimento de cada nó da rede.

Existem diversos tipos de controladores, por exemplo o NOX, foi o primeiro controlador desenvolvido para a SDN e surgiu junto com o protocolo Openflow. Ele foi desenvolvido em C++, é rápido, permite entrada e saída assíncronas (NOX, 2012). O *OpendDayLight* é um controlador modular muito utilizado, foi desenvolvido em Java, possui interface gráfica e suporte para outras tecnologias como *OpenStack* (PROJECT., 2016). Outro exemplo é o controlador Ryu (RYU, 2012), foi desenvolvido em Python, possui uma interface REST (*Representational State Transfer*), suporta o protocolo Openflow 1.0 até 1.5 e *Nicira Extensions*.

Apesar de existirem outros controladores SDN, neste trabalho será usado o controlador Ryu, por ter sido desenvolvido em Python, permite a integração com bibliotecas de ML também em Python. Os controladores *multithreads* ou distribuídos como *OpendDayLight* ou NOX, apresentam melhores resultados em relação ao desempenho ao se comparar com o Ryu (ZHU et al., 2019). Entretanto, o Ryu mostrou ter mais estabilidade no gerenciamento e comunicação com os switchs nos testes de segurança desenvolvidos em (SHALIMOV et al., 2013). Também em (SHALIMOV et al., 2013), o autor cita que o Ryu permite uma rápida prototipação e desenvolvimento de técnicas e software em relação a outros controladores. Como o controlador Ryu suporta versões mais novas do protocolo Openflow, a adaptação deste trabalho para estudar o mesmo problema com outras versões do protocolo Openflow se torna mais fácil.

2.5 Aprendizado de Máquina

O Aprendizado de Máquina (ML - *Machine Learning*), é uma subárea da IA que tem como objetivo a geração de agentes (modelos) inteligentes por meio da detecção de padrões em dados. O ML busca resolver problemas de diversos tipos de predição, com diversos tipos de dados. Se a predição for discreta, tem-se a tarefa de classificação. Caso seja contínua, tem-se a regressão. Em ML, desenvolve-se diversos algoritmos, baseados em diversas técnicas, como estatística (Nanda et al., 2016), probabilidades (Prakash; Priyadarshini, 2018) e biologia (Lai et al., 2019).

Um exemplo de método estatístico é o Naive-Bayes, no qual realiza-se a classificação dos dados com base em um conhecimento anterior, obtido a partir dos dados de treinamento, fazendo a combinação da probabilidade de um evento ocorrer com base na probabilidade de um evento que já ocorreu (Nanda et al., 2016). Já na classe dos algoritmos bio-inspirados, que buscam mimetizar o funcionamento do cérebro humano, existe o algoritmo MLP (*Multi Layer Perceptron*), no qual tem-se vários neurônios organizados em camadas, tendo uma camada de entrada, uma ou mais camadas ocultas e a camada de saída (BEALE, 1997). Cada neurônio calcula o somatório dos dados de entrada e usa o resultado em uma função de limiar na camada de saída para ajustar os limites do resultado (Lai et al., 2019).

Além dos diferentes tipos de algoritmos de ML ainda existe a classificação quanto ao método de execução: os algoritmos supervisionados e os não supervisionados. O método supervisionado consiste na inserção de dados que servem para induzir o treinamento do algoritmo, tal que ocorra a indução para o resultado esperado. Essa indução é por meio do dado classificado quanto ao resultado esperado. Dessa forma, os problemas que esse método abrange são os problemas de classificação e regressão. O método não supervisionado trabalha com dados sem rótulos que os classificam. Dessa forma, o algoritmo agrupa os dados em *clusters* distintos por meio de cálculos de similaridades, assim, também pode ser usado para classificação, agregação e redução da dimensão dos dados (RUSSEL, 2013).

Antes do processamento dos algoritmos de ML,é necessário fazer o pré-processamento dos dados identificando se estão padronizados, normalizados, balanceados ou com poucas amostras, pois na etapa de treinamento o algoritmo pode ter problemas no tempo gasto para o processamento, *overfitting* ou *underfitting*, sem desconsiderar os ajustes do hiperparâmetros dos algoritmos de ML. No problema de *overfitting* o algoritmo decora as amostras de dados, seja em treino ou teste, por causa disso não consegue generalizar para novas entradas de dados. No *underfitting* o algoritmo não consegue acertar as amostras de treinamento, ou seja, não aprende efetivamente e não consegue generalizar para novas entradas de dados (YING, 2019).

2.6 Algoritmos de ML

Dentre os diversos tipos de algoritmos usados no ML, serão explicados de forma sucinta os que foram escolhidos para este trabalho, os quais são: *Naive Bayes*, *Multi Layer Perceptron*, *Random Forest*, *Decision Tree*, *Support Vector Machine*, *K-nearest Neighbors algorithm*, *K-means* e *Gradient Boosting*.

- **Naive Bayes:** O algoritmo de *Naive Bayes* é baseado na teoria da probabilidade de Bayes (HARTIGAN, 1983), em que se tem a finalidade de calcular a probabilidade que uma amostra desconhecida pertença a uma das classes possíveis, ou seja, predizer a classe mais provável com base em um conhecimento anterior do dado, fazendo a combinação da probabilidade de um evento ocorrer com base na probabilidade de um evento que já ocorreu (Nanda et al., 2016). Esse tipo de predição é conhecida como predição estatística na qual o valor de um atributo é independente do valor de outros atributos.

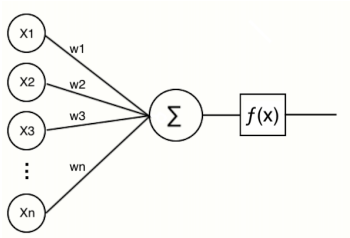
  A formula do *Naive Bayes* em 2.1 mostra que **P(a|b)** é a probabilidade posterior da classe **a**, dado que **b** que são características da instância em análise a ser classificada. **P(a)** é a probabilidade original da classe **a**, **P(b|a)** é a probabilidade posterior dos atributos **b** dado o preditor **a** e **P(b)** é a probabilidade do conjuntos de características que se acontecerem resultam na classe **a**. Um problema desse algoritmo é a falta de amostras de um determinado atributo, tornando a probabilidade correspondente ou a posteriori como zero. Isso, no entanto, pode ser corrigido com o estimador de Laplace (HE; DING, 2007).

$$P(a|b) = \frac{P(b|a)\,P(a)}{P(b)} \tag{2.1}$$

  A ideia principal é que dado um evento **A** e um evento **B**, a probabilidade da ocorrência de um evento não depende apenas da relação dos eventos **A** e **B**, mas também da probabilidade da ocorrência de cada evento.

- **Multi Layer Perceptron:** Uma Rede Neural Artificial (RNA - *Artificial Neural Network*) é um exemplo de classe de algoritmos bioinspirados, nesse caso, o funcionamento do cérebro, especificamente do neurônio cerebral (LUNGER, 2013). O primeiro modelo funcional foi o perceptron, criado em 1959 por Frank Rosenblatt (ROSENBLATT, 1958), sendo similar a um classificador linear que funciona somando as entradas X multiplicadas por um peso W, o resultado desse somatório passa pela função de limiar f(x), classificando em 0 ou 1 a saída. A Figura 2.3, mostra o exemplo de um neurônio do tipo perceptron.

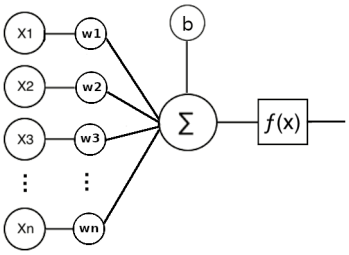
Figura 2.3 – Modelo de neurônio perceptron.



Fonte: Do autor (2021).

O MLP *Multi layer perceptron* é uma evolução do modelo perceptron, no qual agora existem camadas de neurônios e cada camada está totalmente conectada a outra, e o mínimo de camadas de uma MLP são três: uma para entrada de dados, uma oculta e uma de saída dos dados (BEALE, 1997), a Figura 2.5 mostra um exemplo simplificado de um MLP. O neurônio matemático da MLP é mais complexo que o perceptron, pois além das entradas X multiplicadas pelo peso W associado existe um valor de *bias* que ajusta a direção da função de ativação e funções de ativação diferentes. A Figura 2.4, mostra um exemplo desse neurônio matemático.

Figura 2.4 – Modelo de neurônio matemática de uma MLP.

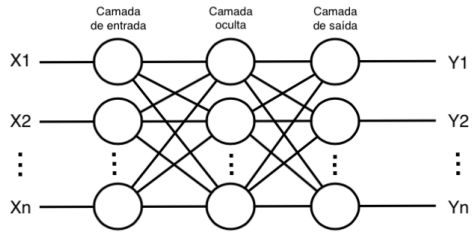


Fonte: Do autor (2021).

O resultado da soma de uma camada se torna a entrada da camada seguinte em uma única direção, a cada iteração os valores de W e o bias são ajustados por meio dos resultados do gradiente descendente e *backpropagation*, essas funções calculam o erro e indicam se os ajustes de W precisam ser positivos, negativos e a magnitude dos ajustes, juntos com a taxa de aprendizado. Assim, a rede neural consegue aprender dos dados e memorizar esse aprendizado nos pesos associados a cada neurônio. Por meio de um processo iterativo de ajustes aplicados aos seus pesos no treinamento, o aprendizado ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

- **Decision Tree:** A árvore de decisão é um método de classificação de eurística gulosa que realiza a indução das regras de classificação dos dados. Esse método constrói uma estrutura de árvore que permite visualizar as regras, iniciando pelo nó raiz até as folhas (BREIMAN et al., 2017). Os nós são os atributos, as folhas são as respostas, as arestas são os valores usados para dividir

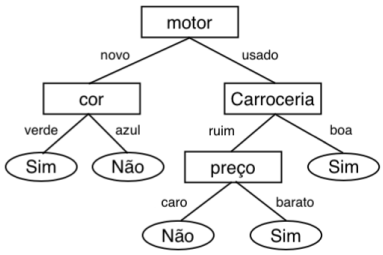
Figura 2.5 – Modelo de RNA do tipo MLP.



Fonte: Do autor (2021).

o caminho entre outros nós até as folhas, formando os caminhos de decisão, a Figura 2.6 mostra um exemplo de árvore de decisão para compra de carros.

Figura 2.6 – Exemplo de árvore de decisão para compra de um carro.



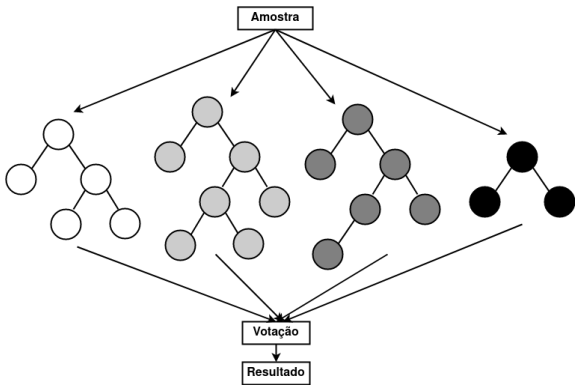
Fonte: Do autor (2021).

A decisão de qual atributo será usado como nó ou a sua posição dentro da árvore pode levar em consideração o ganho de informação (entropia) ou índice Gini. A entropia é a medida de incerteza nos dados e o ganho de informação é a diminuição da entropia, assim, a entropia é calculada antes e depois das partições dos atributos e escolha do nós, os atributos que apresentarem o maior ganho de informação são usados para particionar os dados. O índice Gini é usado para medir o grau de heterogeneidade dos dados, os valores dos seus coeficientes ficam entre 0 e 1, quanto mais próximo de 0 menor a heterogeneidade e quanto mais próximo de 1 maior a heterogeneidade.

- **Random Forest:** O *Random Forest* é um algoritmo da classe de métodos ensemble, os métodos ensemble funcionam pela execução de vários algoritmos usando partes diferentes do *dataset*, ao termino da execução combinam os modelos gerados para realizar a classificação (DIETTERICH, 2000). O *Random Forest* funciona selecionando amostras aleatóriamente do *dataset* para criar um novo *dataset*, no qual pode conter amostras repetidas. Depois cria uma árvore de decisão selecionando alguns atributos aleatoriamente do novo *dataset*. Os passos anteriores são repetidos, assim gerando novos *datasets* e novas árvores de decisão com outros atributos, ao final destes processos é selecionado o melhor conjunto de árvores para criar o modelo (Tin Kam Ho, 1995).

A validação dos modelos é feita pela classificação das amostras que não foram usadas durante a etapa de construção das árvores, deste modo, o melhor modelo é o que apresentar maior acurácia na validação. Na classificação de uma nova amostra são usadas todas as árvores geradas de um modelo, a classe que obtiver mais resultados é selecionada como a classe da nova amostra. Essa combinação de vários modelos e sem dependência entre eles e do mesmo tipo de algoritmo é conhecido como *bagging* (Wang; De Lin, 2007). Um problema desse método é o seu alto custo computacional por causa da geração do grande número de árvores de decisão, além disso, ele torna mais complexa a interpretabilidade do modelo que é obtida na árvore de decisão. A Figura 2.7, mostra um esquema simplificado do *Random Forest*.

Figura 2.7 – Esquema simplificado do Random Forest.

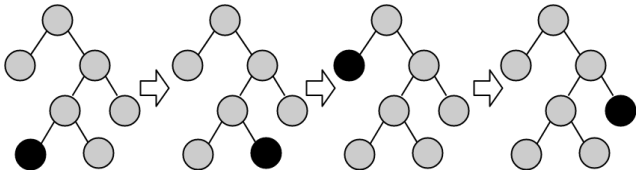


Fonte: Do autor (2021).

- **Gradient Boosting:** O *gradient boosting* é uma técnica do métodos ensemble, sendo usado para classificação e regressão, essa técnica também permite utilizar árvores de decisão internamente. Semelhante ao *Random Forest*, ele também cria diversos modelos e no final combina todos os modelos para obter um melhor resultado. Entretanto, o *gradient boosting* usa a técnica de *boosting* e não de bagging, deste modo os modelos são gerados sequencialmente e dependentes um do outro (FRIEDMAN, 2000), a Figura 2.8 mostra um esquema simplificado deste algoritmo.

  O seu funcionamento na classificação consiste em criar uma predição com base na classe ao qual as amostras pertencem, o valor dessa predição é convertido em probabilidade por uma função logística e usado para criar um novo atributo de resíduo para cada amostra. Assim, se usa uma árvore de decisão completa para classificar as amostras em relação ao resíduo e não a classe inicial. Os novos valores de predição são calculados com o valor das folhas das árvores somado a uma taxa de aprendizado. As próximas árvores de decisão a serem geradas utilizam o valor predito anteriormente e o valor da classe para gerar os novos valores de resíduos a serem classificados. Todos esses passos são repetidos até que se encontre o menor valor de resíduo ou se atinja o número de árvores de decisão definido a serem geradas.

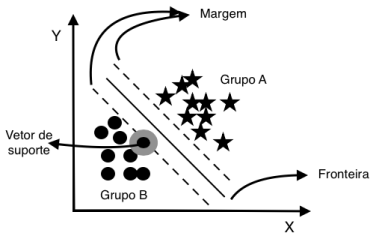
Figura 2.8 – Esquema simplificado do Gradient Boosting.



Fonte: Do autor (2021).

- **Support Vector Machine:** A SVM (*Support Vector Machine*) é um método de aprendizado supervisionado que pode ser usado para classificação ou regressão, funciona com dados linearmente separáveis ou não linearmente separáveis (CORTES; VAPNIK, 1995). O seu funcionamento difere dependendo dos tipos de dados, em dados linearmente separáveis ele classifica as amostras e calcula diversos hiperplanos para separar as classes, além da reta é considerado uma margem entre a reta e as classes, assim a reta que melhor divide as classes é a que possui a maior margem (Sanjaa; Chuluun, 2013). Os dados que ficam em cima da margem são chamados de vetores de suporte. A Figura 2.9, mostra um exemplo do funcionamento da SVM com dados linearmente separáveis.

Figura 2.9 – SVM em dados linearmente separáveis.

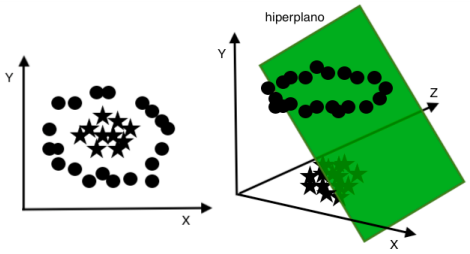


Fonte: Do autor (2021).

Quando o processo de classificação lida com dados não linearmente separáveis a SVM utiliza o artifício de *kernel trick*, no qual adiciona mais uma dimensão nos dados, mudando o seu plano e os cálculos serão para encontrar um hiperplano que melhor divida as classes. O *kernel trick* é feito por meio de uma função que pode ser linear, polinomial, sigmoidal ou Gaussiana. A Figura 2.10 mostra um exemplo do funcionamento da SVM com dados não lineares.

- **K-nearest Neighbors Algorithm:** O KNN é um método de classificação de objetos semelhantes por meio do cálculo da distância de uma amostra usando seu vetor de características, em relação ao número de amostras vizinhas (GUO et al., 2004). A classificação de uma nova amostra se dá pelo o cálculo da distância dessa amostra e seleciona-se a sua classe analisando a classe mais numerosa mais próxima. A Figura 2.11 mostra um exemplo no qual uma amostra ainda não classificada será da classe numerosa mais próxima para o valor de K=2.

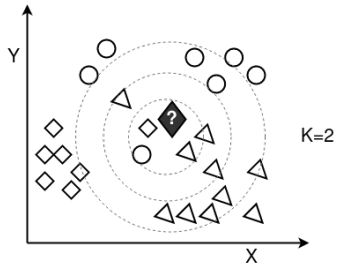
Figura 2.10 – SVM em dados não linearmente separáveis.



Fonte: Do autor (2021).

Existem várias formas de se calcular a distância entre as amostras e cada uma possui uma característica (CHOMBOON et al., 2015), por exemplo: distância Manhattan que é usada quando se tem atributos numéricos, sendo simétrica ela trata todas as dimensões como iguais, porém é sensível a outliers. Distância de Hamming que é usada em dados categóricos, no qual verifica se os atributos são iguais se a distância for 0 e 1 caso contrário. Distância de Minkowsky que é uma generalização da distância euclidiana. Um problema do KNN é a escolha do melhor valor de K, pois se for muito pequeno causa o problema de overfiting e ser for muito grande causa o problema de underfitting, prejudicando a sua performance, sua capacidade de classificação e generalização (BATISTA; SILVA, 2009).

Figura 2.11 – KNN com K=2 com uma amostra ainda não classificada.

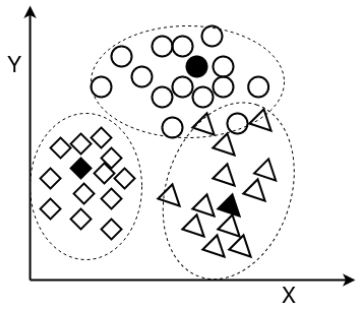


Fonte: Do autor (2021).

- **K-means:** O *K-means* é um algoritmo de clusterização não supervisionado, no qual consiste em criar *clusters* de dados pela proximidade das amostras em relação a uma determinada centroid. A execução desse algoritmo inicia-se com a seleção de um número **K** definido pelo usuário, dentro do espaço de dados são selecionados *K* pontos que serão as *centroids* inciais. Para cada amostra de dados é feito o cálculo da sua distância em relação aos *K* valores, pode ser usado a distância euclidiana para esta tarefa, assim o **K** mais próximo da amostra é o *cluster* a qual ela pertence. Após esses passo é calculado o valor médio para cada *cluster* para encontrar novas *centroids*. Todas as etapas anteriores são repetidas até que as *centroids* não mudem mais de valores. A Figura 2.12, mostra um exemplo de clusterização para o valor de k=3.

Um dos problemas deste algoritmo é encontrar o valor de **K** ideal, um meio de resolver esse problema é por meio do *Elbow Method*. O *Elbow Method* consiste em realizar diversas execuções do *K-means* variando o valor de **K**, assim o valor de **K** que apresentar uma variância baixa e com poucas alterações para outros valores de **K** é o ideal. Outro problema deste método é a falta da capacidade de gerar um modelo que possa ser salvo e executado em outros ambientes.

Figura 2.12 – Exemplo de clusters formados com o K-means.



Fonte: Do autor (2021).

2.7 Segurança da Informação

A segurança da informação está associada a cinco princípios básicos que são: confidencialidade, integridade, disponibilidade, autenticidade e determinação de responsabilidade. A violação de qualquer um destes princípios pode vir a causar prejuízos, assim é necessário desenvolver mecanismos de segurança para a proteção dos ativos de informática e os dados produzidos pelos usuários (BROWN; STALLINGS, 2017).

A confidencialidade de um dado se refere que o mesmo só pode ser acessado por pessoas autorizadas. Esse princípio pode ser alcançado com uso da criptografia de chave simétrica, no qual uma única chave é usada para criptografar e descriptografar o dado. Assim, basta compartilhar a chave com as pessoas autorizadas. Existem diversos tipos de algoritmos de criptografia que variam no seu princípio de funcionamento sendo por fluxo ou bloco, número de operações e tamanho da chave criptográfica (BROWN; STALLINGS, 2008). Diferente da confidencialidade, na autenticidade o objetivo é definir quem é o proprietário de um dado, para isso pode ser usado o mecanismo de chave pública e privada. Nesse método um usuário criptografa o dado com sua chave privada e disponibiliza a chave pública para descriptografar e garantir que o dado pertence ao dono da chave privada (BROWN; STALLINGS, 2017).

Na integridade o objetivo é identificar se um determinado dado sofreu alguma alteração, seja pela ação de um agente autorizado ou em trânsito na rede. A integridade é obtida por meio de algoritmos de *hash*, no qual é uma função que recebe um dado de entrada e retorna um valor único usado

para identificar o mesmo. Caso tenha alguma alteração no dado a sua *hash* também terá o seu valor alterado. A comparação entre as *hashs* que vai definir se houve ou não alteração do dado, entretanto não adianta comparar se não há uma *hash* inicialmente calculada (BROWN; STALLINGS, 2008).

A disponibilidade de um recurso se refere se o mesmo está acessível quando é necessário, entretanto, os mecanismos empregados para se ter a acessibilidade podem variar. Caso o recurso seja um site, é necessário um servidor com capacidade de processar as requisições seja por meio de redundância ou um link de internet que suporte muitos acessos, mesmo assim o site estaria sujeito a um ataque de DDoS (YARIMTEPE; DALKILIC; OZCANHAN, 2015). A determinação de responsabilidade tem o objetivo de identificar uma entidade envolvida em um processo, de forma que seja possível reconstruir seus passos e atribuir unicamente a essa entidade, assim, os sistemas devem manter um *log* de ações dos usuários para posterior análise forense (VELHO, 2016).

Para violar qualquer um desses princípios e comprometer os dados de uma empresa existem diversos meios de ataques, eles variam de acordo com os mecanismos de segurança implementados e as características dos ativos de informática. Um servidor interno, com uma interface web externa, pode estar sujeito a exploração por meio de injeção de código, exposição de dados sensíveis, mecanismos de autenticação fracos, componentes vulneráveis e má configuração (FOUNDATION, 2019). É comum ocorrerem falhas humanas durante os processos executados dentro de uma empresa, no qual os funcionários sofrem ataques de engenharia social, induzindo-os a executarem programas maliciosos sem que saibam ao abrirem anexos dos e-mails ou por meio de *links* maliciosos.

Além dos problemas acima, é cada vez mais comum o uso de malwares para ataques específicos e sofisticados, que é o caso dos ataques com *ramsowares* e *cryptominers* maliciosos. Os *ramsowares* atacam fazendo o sequestro dos dados por meio da criptografia dos mesmos, e a chave para descriptografar só é entregue após o pagamento de um valor em dinheiro ao seu criador. Já os *cryptominers* maliciosos consomem os recursos das máquinas infectadas para minerar criptomoedas (KASPERSKY, 2019 (accessado 17, 08, 2019)).

O autores em Dabbagh et al. (2015) mostram a existência de falhas de segurança na SDN que permite que ocorram ataques de DDoS e ocultação de tráfego malicioso por meio de criptografia até o comprometimento total da rede caso o controlador esteja comprometido por meio de um ataque. O autores em Mohammadi et al. (2017), descrevem os ataques de negação de serviço, especificamente *Route Spoofing* e *Resource Exhaustion*. O ataque de *Route Spoofing* consiste em forjar rotas alternativas para os pacotes de rede podendo derrubar as conexões entre as rede diferentes. O ataque de *Resource Exhaustion* consiste em consumir todos os recursos do plano de dados e do controlador da rede, deixando fora do ar.

Assim, todos os ataques se esforçam para burlar mecanismos de segurança como *firewalls*, antivírus e até mesmo as características estruturais de uma rede, deste modo é necessário desenvolver novos métodos para combater esses ataques e novas ferramentas de segurança. Uma alternativa é o uso da análise do tráfego de rede para identificar padrões de ataques dentro da rede, alertando os administradores para combaterem os ataques.

2.8 Análise do Tráfego de Rede

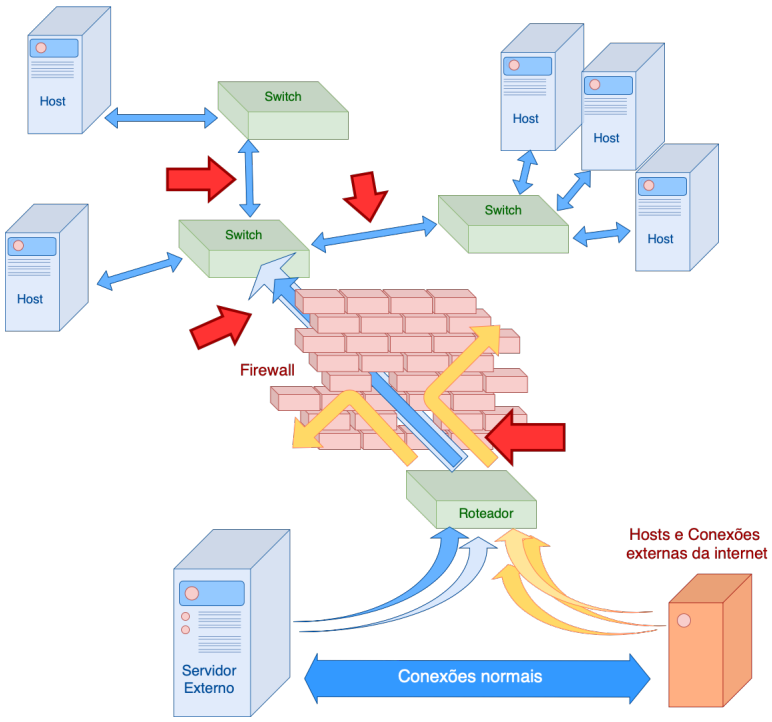
A análise do tráfego de rede é feita por meio da captura de pacotes e classificação dos mesmos, o objetivo é identificar padrões que mostrem alguma informação. Essa abordagem pode ser usada para identificar equipamentos defeituosos ou agentes maliciosos. Apesar de parecer simples, capturar e classificar o tráfego de rede é muito complexo, pois existem diversas características e detalhes que influenciam na captura e nos resultados. A captura do tráfego precisa ser feita de modo que não interfira significativamente no funcionamento da rede, prejudicando o mínimo possível, isso pode ser feito por meio do espelhamento de uma porta do switch. Entretanto, o equipamento que vai capturar e analisar precisa dar uma resposta do processamento em um curto espaço de tempo, senão os resultados obtidos já não podem ser mais usados.

Outro detalhe importante a ser considerado são os protocolos de rede a serem analisados, pois cada protocolo tem um objetivo e funcionamento distinto. Por exemplo, o protocolo TCP (POSTEL, 1981) é usado na camada de transporte, orientado a conexão, faz checagem de erros e é lento em relação a outros protocolo. Outro exemplo é protocolo UDP (POSTEL, 1980) que é usado na camada de transporte, mas não é orientado a conexão e não faz checagem de erros, assim acaba sendo mais rápido que o TCP. O protocolo TCP é mais utilizado em serviços que necessitam de uma segurança no envio dos pacotes de rede, como transporte de arquivos, onde a falta de dados corromperia o arquivo [3]. Já o protocolo UDP é mais utilizado em serviços de *stream multimídia*, no qual a perda de alguns pacotes não afetaria muito a qualidade, já que o objetivo é a velocidade do serviço (TANENBAUM, 2011).

O posicionamento do equipamento para a captura do tráfego da rede também influencia nos resultados. A localização precisa ser escolhida com base em um objetivo na captura do tráfego e quais partes da rede se deseja monitorar. A Figura 2.13 mostra um exemplo de uma rede com duas sub redes protegidas por um *firewall*, nessa mesma figura está destacado alguns pontos que podem ser usados para a captura do tráfego. Essa captura pode ser feita em toda a rede dentro da área de proteção do *firewall* ou fora dela, também pode ser feito em partes específicas da rede.

Existem ferramentas específicas para a capturar e analisar o tráfego de rede com o objetivo de prevenir ou detectar intrusões, tais ferramentas são conhecidas como IPS (*Intrusion Prevention*

Figura 2.13 – Exemplo de arquitetura de rede.



Fonte: Do autor (2021).

System) e IDS (*Intrusion Detection System*). O funcionamento é por meio da captura e processamento do tráfego de rede em busca de comportamento malicioso. O IPS previne a ação maliciosa bloqueando a conexão e o IDS apenas detecta. Segundo os autores em (Tidjon; Frappier; Mammar, 2019) existem quatro métodos para detecção de anomalias, os quais são:

- **Detecção por assinatura:** Nesse método são usadas assinaturas que classificam os tipos de anomalias que podem existir, entretanto, possui a falha de não detectar novos ataques que não fazem parte do banco de assinaturas.
- **Detecção por anomalia:** Nesse método são usados modelos que determinam o comportamento normal da rede, e caso ocorra alguma anomalia ela vai ser identificada por se destacar do comportamento normal da rede.
- **Detecção por multiagente:** Nesse método são utilizados múltiplos agentes de detecção espalhados dentro da rede, trabalhando de forma centralizada, colaborativa e cooperativa para identificar ataques de múltiplos estágios.
- **Deteção híbrida:** Nesse método há a combinação dos métodos anteriores e algoritmos de IA para aumentar a acurácia dos resultados.

Como exemplo de ferramentas que utilizam uma ou mais técnicas descritas acima, existem o Snort (CISCO, 2019) e Suricata (FOUNDATION, 2019). Segundo os autores Albin e Rowe (2012), o

Suricata e o Snort funcionam de forma semelhante na captura do tráfego e processamento dos pacotes por meio de regras pré-definidas para identificar padrões de ataques, e podem ser usados como IDS e IPS.

O crescimento no uso das redes convencionais é um problema para as soluções de análise de rede por causa do grande volume de dados gerados. Dessa forma, uma SDN possui a vantagem de ter plano de controle separado do plano de dados, como consequência o processamento dos pacotes e decisões de rotas ficam fora dos *switchs* removendo um *overhead* nos equipamentos, deixando os *switchs* apenas como meio de conexão entre os equipamentos e *hosts* (Sokolov et al., 2014). Além disso, essa separação de funções permite a construção de módulos auxiliares para o processamento do tráfego para gerar novas informações para tomada de decisão para criação de novas regras para o controlador (Sokolov et al., 2014).

3 TRABALHOS RELACIONADOS

O protocolo Openflow permite o desenvolvimento de ferramentas de análise de rede, como em (Henni; Hadjaj-Aoul; Ghomari, 2016), no qual os autores descrevem o desenvolvimento de um framework para monitoramento do tráfego SDN. Um problema encontrado é a captura de dados equilibrando eficiência e confiabilidade dos dados. A estratégia adotada foi fazer a requisição das estatísticas da rede em intervalos fixos aos switchs Openflow e calcular a média dos resultados. Outro exemplo de uso do Openflow pode ser visto em (Gangwal; Conti; Gaur, 2017), no qual o autor desenvolveu um framework usando as características da rede SDN como separação entre os planos de dados, fluxo e estatísticas da rede. O objetivo deste framework é a descoberta da arquitetura da rede em tempo real, por meio do monitoramento e informações das tabelas dos switchs usando o protocolo OpenFlow. Apesar disso, esses autores (Gangwal; Conti; Gaur, 2017), não avaliam o comportamento em uma rede com a variação do número de switchs ou máquinas emuladas, também não analisam o comportamento do framework com a rede sofrendo algum tipo de ataque ou ruídos que possam ocorrer.

Outro exemplo de manipulação dos recursos que o SDN e o protocolo Openflow é descrito em (Kim; Shin, 2017), no qual os autores explicam o funcionamento do ataque de inundação de link (LFA - *Link Flood Attack*). O ataque tenta exaurir a banda da rede por meio de várias requisições feitas nos links onde se convergem as conexões (gargalo da rede) por meio *bots*, no geral é uma variação de DDoS, porém de difícil identificação. Para combater o ataque de LFA foi usado *honeypots* que imitam dispositivos reais e vulneráveis, a localização dele na rede é baseado no consumo e latência da rede calculados por meio das estatísticas fornecidas pelo controlador da rede SDN. Os autores de (Kim; Shin, 2017), não exploraram outras arquiteturas de rede e outros tipos fluxos além do UDP, para analisar a eficácia da estratégia de proteção contra LFA em outros cenários.

Existem trabalhos de detecção de anomalias em redes SDN que não usam técnicas de ML, os autores em (Garg; Garg, 2015) e o trabalho (Jeong et al., 2014), são exemplos. Em (Garg; Garg, 2015) desenvolveu um algoritmo próprio que analisa a agregação dos links na rede. Os autores em (Jeong et al., 2014) virtualizaram uma rede para redirecionar o tráfego para ser analisado pelo IDS Suricata. Apesar de não contribuir no problema de ML, esses trabalhos ilustram que a capacidade de análise da rede está ligado na forma de amostragem dos pacotes, já quem em redes de alto desempenho o uso de um IDS pode ser inviável. A forma de amostragem de pacotes de rede dos trabalhos (Garg; Garg, 2015) e (Jeong et al., 2014) foi seletiva de tempo em tempo.

Além do volume do tráfego há o problema de alta complexidade dos dados causando um gargalo no tempo de processamento, mesmo com as facilidades de obter dados estatísticos dos switchs Openflow de uma rede SDN (Thupae et al., 2018). Além dessas observações iniciais, os autores em Thupae et al. (2018) faz uma seleção e classificação das técnicas de ML enumerando os principais

algoritmos utilizados para análise de tráfego de rede, dividindo os algoritmos em supervisionados e não supervisionados. O trabalho Thupae et al. (2018) mostra que usando os algoritmos supervisionados se obteve uma acurácia de 90%-94%. Enquanto isso, métodos não supervisionados obteve uma acurácia de 90%-97%, porém não faz nenhuma relação com os tipos de ataques estudados em cada trabalho.

O trabalho desenvolvido por (Nanda et al., 2016), avalia a capacidade de predição de um *host* ser atacado usando C4.5, BayesNet, Decision Table e Naive-Bayes. Para isso foram usados os dados do projeto Longtail[1] que contém informações de ataques de força bruta. O *BayesNet* apresentou o melhor resultado em relação aos outros algoritmos, assim o *host* que for identificado com maior probabilidade de ser atacado deve ser imediatamente bloqueado para evitar maiores danos. Entretanto, o trabalho não menciona se esse método foi efetivamente usado ou se o mesmo método poderia ser usado para outros tipos de ataques.

O trabalho apresentado em (Comaneci; Dobre, 2018), é um exemplo da combinação de métodos supervisionado e não supervisionado, no qual utiliza os dados *inter-packet arrival time*, *packet size*, *packet count* e *flow tuple* obtidos de um switch Openflow. Cada amostra é classificada por um modelo de árvore de decisão C.45, entretanto, se o resultado dessa classificação não for conclusivo é usado o algoritmo *K-means* para agrupar a amostra identificando-a se faz parte da classe de anomalia ou não. Os autores em (Comaneci; Dobre, 2018), não explora outros tipos de ataques, também não utilizam *switchs* reais para avaliar o comportamento dos algoritmos diante de possíveis ruídos da rede

Outro exemplo do uso de redes neurais para classificação se encontra em (JANKOWSKI; AMANOWICZ, 2015), onde os autores simularam uma rede vulnerável monitorada por um switch Openflow. Os ataques realizados foram classificados segundo (DHANABAL; SHANTHARAJAH, 2015), separando em DoS, Probe, R2L e U2R. Para a classificação foi usado o Mapa de Kohonem, separando os pacotes capturados nos referidos grupos. Segundo os autores, os resultados obtidos não foram satisfatórios, além de não terem usado esse método para gerar regras no controlador automaticamente para bloquear novos ataques.

Um exemplo de combinação de métodos diferentes é o trabalho (Abubakar; Pranggono, 2017), no qual o switch Openflow é responsável por redirecionar parte do tráfego para ser analisado pelo Snort, em conjunto foi implementado uma rede neural que classifica os dados vindos do switch. Essa combinação foi necessária, pois os ataques que não são identificados pelo Snort poderiam ser classificados na rede neural. Entretanto, os autores não mencionam se foram criadas regras no controlador para verificar se há o efetivo bloqueio de novos ataques ou dos mesmos, também não utilizou outros algoritmos de ML para comparar os resultados na questão do tempo de resposta da identificação da anomalia.

[1] LongTail Log Analysis @ Marist College / Today's Data All SSH Ports - http://longtail.it.marist.edu/honey

Os autores em Santos da Silva et al. (2016) desenvolveram um framework para análise, detecção, classificação e mitigação de anomalias da rede. Os pacotes capturados passam por algoritmos supervisionados para identificar alguma anomalia, caso não seja identificado são usados algoritmos não supervisionados pra agrupar os fluxos da rede. Como resultados são obtidos três tipos de classes como anomalia, normal e desconhecido. Entretanto os autores não exploraram outros algoritmos de ML, apenas o KNN e SVM. Também não variaram os tipos de ataques e não usaram *switchs* reais para avaliar os possíveis ruídos no tráfego que poderiam ocorrer.

O trabalho (ELSAYED; LE-KHAC; JURCUT, 2020) tem o objetivo de criar e validar um *dataset*, deste modo configuraram uma rede virtual para realizar diversos ataques e captura do tráfego SDN gerado, depois validaram com técnicas de ML o *dataset*. Apesar disso, o trabalho não explora a combinação com equipamentos reais para gerar o tráfego e nem analisa a eficácia dos modelos com novos tráfegos. Já em (AHMAD et al., 2020), foi criado um dataset somente com ataques de DoS em um ambiente virtual, também analisaram os resultados com técnicas de ML. Porém, o trabalho não explora outros tipos de ataques de DoS, outras técnicas de ML e não detalham muito sobre dataset criado.

Com base nos trabalhos observados até o momento, nota-se oportunidades de estudos mais aprofundados sobre o tema de detecção de anomalias em redes SDN, apesar do grande volume de trabalhos. Alguns focam apenas no aspecto da arquitetura da rede, outros apenas nos algoritmos usados para detecção, alguns até chegam explorar modalidades diferentes como a combinação do ML e ferramentas prontas como Snort e Suricata. Além disso, a área de pesquisa com SDN e ML carece de *datasets* e ferramentas específicas para criar os *datasets*. A Tabela 3.1 mostra um resumo dos principais trabalhos usados como base teórica deste projeto em comparação com o que este projeto pretende desenvolver.

A Tabela 3.1 o campo "Título" é o nome do trabalho desenvolvido. O campo "Objetivo" é o objetivo que a pesquisa pretendia realizar. O campo "Desenvolvido" é mostrado se foi criado algum framework ou aplicação que possa ser usada para comprovar os resultados da pesquisa ou em ambientes reais. O campo "ML" mostra os métodos de ML usados para analisar o tráfego de rede, quando há muitos métodos foi colocado apenas a classificação como supervisionado e não supervisionado. O campo "Traces" é com relação as bases de dados, se foram usadas bases prontas ou criadas. O campo "Abordagem" é com relação a estratégia usada para execução da pesquisa. O campo "Virtualização" é com relação ao uso de algum equipamento de rede físico como um switch Openflow ou se foi totalmente virtualizado. O campo "Testes e Ataques" é com relação aos tipos de ataques realizados e se foi feito o uso da rede normal combinando com os ataques.

Tabela 3.1 – Comparação dos principais trabalhos relacionados a este projeto.

Título	Objetivo	Desenvolvimento	ML	Traces	Abordagem	Virtualização	Testes e Ataques	Ano
Sphinx: Detecting Security Attacks in Software-Defined Networks.	deteção de ataques.	Framework.	Não.	Base própria.	Usa algoritmos desenvolvidos específicos para os problemas.	Sim.	arp poisoning, fake topology, controler DOS, network DOS, TCAM auxaustion, switch blackhole, syn flood e misturou.	2015
Intrusion Detection in Software Defined Networks with Self-organized Maps.	Deteção de intrusos na SDN usando mapa de kohonem.	Usa Mapa Kohonem para avaliar o tráfego de rede semelhante ao KDD99 feito na mão.	Sim., mapa de kohonem.	Base própria usando o esquema de KDD.	Criam uma rede, monitora com switch SDN e atacam um host e avaliam o resultado.	Sim.	DOS, Scan(Probe), User2Root, Remote2Local e tráfego normal.	2015
Predicting Network Attack Patterns in SDN using Machine Learning Approach.	Predição de padrões de ataques usando ML com base em ataques anteriores.	Não desenvolveu nenhuma ferramenta.	Sim, KNN, SVM, c45, decision tree.	LongTail dataset (SSH brute force).	Treinou o algoritmo LongTail dataset e testou a eficácia na identificação de novos ataques aos hosts mais suscetíveis.	Sim.	Brute force.	2016
A Simulation Study of SDN Defense Against Botnet Attack Based on Network Traffic Detection.	Identificação de botnets por meio da rede SDN (os recursos que ela oferece).	Nada, apenas estudo do tráfego de rede por meio de SDN	Não.	Não.	Analisa o tráfego de uma rede SDN com máquinas infectadas por malware.	Não.	Bootnet.	2018
Software-Defined Firewall: Enabling Malware Traffic Detection and Programmable Security Control.	Deteção de tráfego malicioso na rede, mesmo que um malware tenha manipulado a rede do host.	Um software que pega os dados de vários pontos da estrutura de TI (host e rede).	SVM.	Base própria capturando os tráfegos dos malwares.	Usar diversos módulos para obter informações, gerar um BD e o que não estiver nele usar o SVM para refinar a classificação.	Não.	Utilização de malwares diferentes para analisar o tráfego.	2018
Securing Networks Using SDN and Machine Learning.	Deteção de anomalias de rede combinando métodos supervisionados e não supervisionados.	Não desenvolveu nenhuma ferramenta.	C.45 e Kmeans	Base própria	Passa no supervisionado, não encontrou passa no não supervisionado. De alguma forma ou de outra ele classifica um fluxo.	Sim.	Diversos ataques. Misturou com o uso normal da rede.	2018
Detection of Flow Based Anomaly in Openflow Controller: Machine Learning Approach in Software Defined Networking.	Deteção de anomalias de rede usando ML (NSL_KDD)	Não desenvolveu nenhuma ferramenta.	Random Forest	Usa parte dos dados da NSL_KDD.	Usar ML para analisar os dados de rede, apenas algumas características da NSLKDD e compara com outra técnicas para análise de rede.	Não.	Identificação de anomalias usando algumas características do NSL_KDD.	2018
Evaluation of Machine Learning Techniques for Security in SDN	Avaliar técnicas de ML contra ataques de DDoS/DoS contra o controlador.	Não desenvolveu nenhuma ferramenta.	SVM, Naive Bayes, Decision Tree, Regressão Logística.	Base própria.	Gerar um dataset com ataques de DoS e analisar com ML se é possível identificar esses ataques.	Sim.	Gerou um tráfego para imitar o ataque de DDoS e analisou o dataset com ML.	2020
InSDN	Gerar um dataset para ser usado por pesquisadores.	Não desenvolveu nenhuma ferramenta.	Knn, MLP, SVM, Random Forest, Decision Tree, Naive Bayes e Adaboost.	Base própria.	Configurar uma estrutura de SDN e gerar um dataset a partir de ataques. Analisar o dataset por meio de ML.	Sim.	Foi gerado tráfego normal e com ataques.	2020
Identificação de Tráfego Malicioso em Redes SDN com Técnicas de Inteligência Computacional.	Identifica tráfego malicioso e gerar regras no controlador automaticamente para bloquear os ataques.	Desenvolvimento de uma ferramenta para gerar datasets e analisar o tráfego de rede usando modelos de ML.	Knn, K-means, MLP, SVM, Random Forest, Decision Tree, Naive Bayes e Gradient Boosting.	Base própria.	Gerar um datasets com o tráfego SDN do switch Openflow. Gerar modelos de ML. gerar regras automaticamente com controlador usando um modelo de ML para classificar o tráfego da rede.	Sim.	Foi gerado tráfego normal, com ataques e misturado. Foram criado modelos de ML com base no tráfego gerado para classificar novas entradas no ambiente virtualizado e real.	2021

Fonte: Do autor (2021).

4 METODOLOGIA

Este projeto aborda a aplicação de técnicas de ML para analisar o tráfego de uma SDN em busca de anomalias. Os resultados das análises são usados para criar as regras automaticamente no controlador para bloquear as anomalias da rede. A execução desta pesquisa será feita em duas etapas distintas:

- **Etapa 1:** Por falta de *datasets* específicos de anomalias de tráfego de SDN, feito a captura do tráfego dentro da SDN para criar um *dataset*. O *dataset* gerado será usado para treinar os algoritmos de ML e obter os modelos mais adequados de acordo com os seus resultados.
- **Etapa 2:** Os modelos escolhidos serão usados para classificar um novo tráfego dentro da SDN, como um meio de validação dos mesmos. Assim, obtendo-se o melhor modelo para o problema de classificação de anomalias.

A execução dos cenários serão feitas dentro do mesmo ambiente criados com o Mininet. A abordagem de criar um *dataset*, treinar um comitê de algoritmos de ML, obter os melhores modelos e validar os mesmos com dados totalmente novos, permite identificar quais são os atributos mais relevantes do *dataset*, quais os algoritmos mais adequados ao problema de classificação e se os mesmos resultados obtidos durante a validação possuem alguma relação com os resultados do treinamento.

4.1 Simulador Mininet

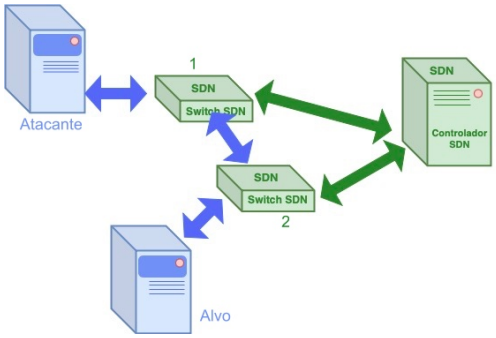
O Mininet é um simulador de rede que permite criar *hosts* virtuais, *switches*, *switches* Openflow e conexões entre eles, permitindo a configuração de arquiteturas de redes virtuais. Possui diversas vantagens como a possibilidade de criar várias topologias de rede usando sua interface de programação em Python, possui um bom desempenho por utilizar bibliotecas nativas do Linux e também por executar em um ambiente seguro e virtualizado com *chroot* (MININET, 2019). Por causa das características do Mininet ele foi usado na criação do ambiente de testes virtualizado, esse ambiente foi simulado em um computador com processador Intel(R) Core(TM) i7-4790 CPU 3.60GHz com 10GB de memória RAM e um armazenamento de 1TB em SSD, o sistema operacional usado foi um Linux com kernel 5.4.

4.2 Arquitetura da rede

A rede usada nesta pesquisa foi construída usando o simulador Mininet. A rede consiste de dois *switches* interligados. No *switch* 1 está conectado o host atacante com o *Kali Linux* instalado, no *switch* 2 está conectado o *Metasploitable* atuando como um servidor vulnerável. A Figura 4.1 mostra

essa arquitetura. O controlador conectado nos *switches* foi construído com o framework Ryu, usando o protocolo Openflow 1.3 para fazer a comunicação.

Figura 4.1 – Arquitetura da rede de teste.



Fonte: Do autor (2021).

4.3 Geração dos Tráfegos de Rede

Os tráfegos de rede desta etapa foram gerados de modo que abranjam alguns tipos ataques que ocorrem na internet, por meio do levantamento de informações com *scanners* de rede, *scanners* de vulnerabilidades, exploração das falhas nas aplicações instaladas, negação de serviço (DoS - *Denial of Service*) e configurações mal feitas. O objetivo desses ataques é gerar um tráfego malicioso com técnicas de ataques com ferramentas mais atualizadas, de forma que o *dataset* exponha os algoritmos de ML aos ataques reais.

A ferramenta usada para escanear a rede é o *Nmap*[1], ela funciona enviando diversos pacotes de redes e analisa a resposta do servidor para identificar qual o serviço ou porta aberta está funcionando. Para scanear em busca de vulnerabilidades será usada a ferramenta *NESSUS*[2], ela mantém uma base de dados com as principais falhas públicas na internet e funciona enviando os comandos necessários para explorar as falhas. Para scanear em busca de vulnerabilidades web será usada a ferramenta *Nikto*[3], ela funciona fazendo diversas requisições no site com padrões de ataques a aplicações web em busca de vulnerabilidades.

Nos ataques de DoS será usado a ferramenta *T50*[4], ela funciona fazendo o envio em grande quantidade de diversos tipos de pacotes de rede, os quais variaram entre protcolos GRE, DCCP, TCP, UDP, IP, ICMP, IGMP, TCP, EGP, RIP, DCCP, RSVP, IPSEC, EIGRP e OSPF. Já a ferramenta *SlowHTTPTest*[5] funciona de forma diferente, ela realiza várias conexões em um servidor web, porém,

[1] Nmap - https://nmap.org
[2] NESSUS - https://www.tenable.com
[3] Nikto - https://github.com/sullo/nikto
[4] T50 - https://sourceforge.net/projects/t50/
[5] SlowHTTPTest -https://github.com/shekyan/slowhttptest

ela mantém essas conexões ativas até que o servidor não tenha mais recursos para alocar. A exploração de vulnerabilidade web foi realizada com a ferramenta *Sqlmap*[6], o seu funcionamento é por meio da manipulação de parâmetros enviados a um servidor web para explorar falhas de *SQL injection*.

O *dataset* também contém o tráfego de rede normal que foi gerado por meio do acesso as páginas do servidor, upload de arquivos com tamanhos diferentes e o download de dados do servidor. E o último tipo de tráfego gerado foi por meio da mistura dos tráfegos maliciosos e normal, no qual foi executado as ferramentas de ataques junto com o acesso do servidor. Assim, cada tráfego diferente gerado se tornou uma classe do atributo "tipo"do *dataset*, como mostrado na Tabela 4.1.

Por questão de praticidade serão usadas distribuições *Linux* pré-configuradas para realizar os ataques e como servidor vulnerável. Para o ataque será a distribuição *Kali Linux*[7], no qual ela foi desenvolvida para ser usada por profissionais de segurança da informação por possuir diversas ferramentas pré-configuradas. Para o servidor será usado o *Metasploitable 2*[8], essa distribuição foi desenvolvida para realizar o estudo de teste de intrusão, contendo diversos serviços vulneráveis para serem explorados.

4.4 Captura do Tráfego e Criação do Dataset

O tamanho do *dataset* criado para esta pesquisa contém 16.000 amostras de dados, 26 atributos, sendo 8 classes distintas de tráfego balanceados em 2000 amostras para cada classe, as classes são *tráfego Nessus*, *Tráfego Nikto*, *Tráfego Normal*, *Tráfego Sqlmap*, *Tráfego Slowhttptest*, *Tráfego T50*, *Tráfego Nmap* e *Tráfego Misturado*. A Figura 4.2, mostra uma parte do *dataset* que foi criado.

Figura 4.2 – Amostra do *dataset* sem o pré-processamento.

```
1  tempo,mac,arp,ipv4_qtd,ipv4_frame,icmp,tcp_qtd,tcp_frame,udp_qtd,udp_frame,dump2,dump3,dump4,in_port,dp,pk_co
   unt,byte_count,duration_flow,rx_packets,rx_bytes,rx_errors,tx_packets,tx_bytes,tx_errors,duration_port,tipo
2  23:00:40,"08:00:27:84:ce:c9",0,0,0,0,1,2003,0,0,0,0,0,3,2,5629,3355513,9,0,0,0,11524,7066302,0,264
3  23:00:40,"08:00:27:63:a8:a9",0,0,0,0,1,63,0,0,0,0,0,5,2,6510,1846061,9,14336,3619190,0,12638,7139046,0,348
4  23:00:52,"08:00:27:84:ce:c9",0,0,0,0,1,516,0,0,0,0,0,3,2,6513,4159807,9,0,0,0,19388,12105991,0,276
5  23:00:52,"08:00:27:63:a8:a9",0,0,0,0,1,63,0,0,0,0,0,5,2,7220,2146389,9,23012,6160865,0,20500,12178584,0,360
6  23:01:04,"08:00:27:63:a8:a9",0,0,0,0,1,63,0,0,0,0,0,2,1,7272,1988774,9,0,0,0,8698,2339616,0,12
7  23:01:04,"08:00:27:84:ce:c9",0,0,0,0,1,2003,0,0,0,0,0,5,1,6397,4030702,10,7589,4689546,0,8699,2339686,0,12
8  23:01:04,"d8:cb:8a:30:01:fa",0,2,64.0,0,0,0,0,0,0,0,6,1,315,23371,10,1302,1294989,0,0,0,0,10
9  23:01:16,"08:00:27:84:ce:c9",0,0,0,0,1,1354,0,0,0,0,0,3,2,6434,4067903,9,0,0,0,7877,4897478,0,12
10 23:01:16,"08:00:27:63:a8:a9",0,0,0,0,1,63,0,0,0,0,0,5,2,7326,1888928,9,8997,2419189,0,7876,4897387,0,12
11 23:01:28,"08:00:27:84:ce:c9",0,0,0,0,1,282,0,0,0,0,0,3,2,5639,3383052,9,0,0,0,15161,9323150,0,24
12 23:01:28,"08:00:27:63:a8:a9",0,1,64.0,0,1,63,0,0,0,0,0,5,2,6532,1800213,9,17349,4779581,0,15156,9320699,0,24
13 23:01:40,"08:00:27:84:ce:c9",0,0,0,0,2,2071,0,0,0,0,0,3,2,6644,4134727,9,0,0,0,23035,14307719,0,36
```

Fonte: Do autor (2021).

Criar um *dataset* é uma tarefa difícil e complexa, pois além do contexto em que a criação se encaixa, nesse caso sobre a segurança computacional, também é necessário analisar quais os dados que podem ser capturados. Assim, foi construído uma ferramenta *open-source* usando a biblioteca Ryu para criar os *datasets* de SDN, caso necessário esta ferramenta pode ser alterada para pegar outros

[6] sqlmap - http://sqlmap.org
[7] Kali Linux - https://www.kali.org/
[8] Metasploitable 2 - https://sourceforge.net/projects/metasploitable/

atributos da rede, além dos descritos na Tabela 4.1. A construção do *dataset* foi feito com as amostras capturadas no ambiente virtualizado, porém, a forma de gerar uma amostra é dividida em duas etapas:

- **1º- Coleta Inicial:** É feito a captura dos dados de um fluxo que entrou no *switch* e que não tenham regras na *flow table* para tratar esse fluxo, também é coletado os dados estatísticos da porta em que o fluxo entrou. As informações que são coletadas do novo fluxo variam de acordo com os tipos de protocolos, sendo eles o *arp*, *tcp*, *udp*, *icmp* e os erros no processamento de cada um desses protocolos.

- **2º- Coleta Final:** É feito uma nova captura após 10 segundos de espera, mas somente dos dados estatísticos por porta do *switch*. Em seguida é calculada a diferença da captura inicial e a captura dessa segunda etapa, assim obtendo o valor estatístico real dentro do intervalo de tempo. A associação dos dados da captura inicial e final para cada amostra é feita pelo endereço MAC do computador, porta e *switch* no qual o computador está fazendo a conexão. Após essa captura as *flow tables* são limpas para se gerar novas estatísticas.

As informações mais detalhadas dos atributos coletados de cada um dos fluxos e transformados em amostras podem ser visualizadas na Tabela 4.1. Entretanto, parte dos atributos foram calculados pelo *switch* e a outra parte é capturado no início de um novo fluxo. O Atributo 1 é calculado quando o fluxo é salvo no arquivo do *dataset*. Os atributos capturados no início de cada novo fluxo que chega no controlador são de 2 a 13. Os atributos calculados automaticamente pelo *switch* são de 14 a 25, no qual foi feito a diferença entre uma captura inicial e final em um intervalo de tempo para adicionar ao *dataset*. O atributo 26 foi adicionado manualmente após os tráfegos serem gerados como descrito no tópico 3.4.

Tabela 4.1 – Atributos que compõe as amostras dos fluxos capturados na SDN

Nº	Atributo	Significado do atributo
1	tempo	Hora, minuto e segundo, em que os dados da amostra foram calculados e adicionado no *dataset*.
2	mac	Endereço MAC do host associado ao fluxo.
3	arp	Quantidade de pacote ARP recebido pelo controlador.
4	ipv4_qtd	Quantidade de pacotes IPV4 recebido pelo controlador.
5	ipv4_frame	Média do TTL dos pacotes IPV4 pelo número de pacotes.
6	icmp	Quantidade de pacotes ICMP recebido pelo controlador.
7	tcp_qtd	Quantidade de pacotes TCP recebidos pelo controlador.
8	tcp_frame	Média do tamanho da janela do pacote pelo número de pacotes
9	udp_qtd	Quantidade de pacotes UDP recebidos pelo controlador.
10	udp_frame	Média do tamanho dos pacotes pelo número de pacotes.
11	dump2	Registro da quantidade de erros ao capturar o pacote ARP. É considerado um erro quando o controlador não identifica o pacote.
12	dump3	Registros da quantidade de erros ao capturar os pacotes IPV4 e ICMP. É considerado um erro quando o controlador não identifica qualquer um dos pacotes.
13	dump4	Registros da quantidade de erros ao capturar os pacotes TCP e UDP. É considerado um erro quando o controlador não identifica qualquer um dos pacotes.
14	in_port	Porta do switch Openflow associada ao fluxo da amostra
15	dp	Identificador do Switch
16	pk_count	Número de pacotes do fluxo
17	byte_count	Quantidade de bytes do fluxo
18	duration_flow	Tempo de duração do fluxo
19	rx_packets	Número de pacotes recebidos
20	rx_bytes	Quantidade de bytes recebidos
21	rx_errors	Número de erros ao receber
22	tx_packets	Número de pacotes enviados
23	tx_bytes	Quantidade de bytes enviados
24	tx_errors	Número de erros ao enviar
25	duration_port	Tempo de duração do fluxo em segundos
26	tipo	Classe do tráfego gerado e esse atributo não é adicionado automaticamente pelo controlador, sendo necessário intervenção manual. As classes são: 1=Nessus, 2=Nikto, 3=Nmap, 4=Normal , 5=slowhttptest, 6=sqlmap, 7=t50 , 8=Misturado.

Fonte: Do autor (2021).

4.5 Pré-processamento e Exploração do Dataset

Apesar do número de amostras estarem balanceadas pelo número de classes, isso não significa que os dados não precisem de um tratamento ou padronização. Nessa etapa foi realizada uma análise

exploratória do *dataset* para identificar os tipos dos dados, formato dos dados, atributos zerados e distribuição das amostras.

Por causa da quantidade de atributos escolhidos do tráfego da rede, foi utilizado o método de correlação de *Spearman* (SPEARMAN, 1904) para identificar se os atributos possuem o problema de correlação. Se dois atributos possuem alta correlação a alteração do valor em um deles vai influenciar no valor do outro atributo, por causa desse comportamento é que foi removido alguns atributos em uma das divisões do *dataset*. Os atributos com alta correlação identificados são o 4, 5, 6, 7, 8, 9, 10, 16, 19 e 22 da Tabela 4.1. Para resolver o problema da alta correlação foram removidos os atributos 4, 9, 16, 19 e 22, da Tabela 4.1, em uma das execuções dos algoritmos de ML.

Para analisar o comportamento e performance dos algoritmos de ML na execução com um *dataset* menos complexo, foi selecionado todas as amostras da classe normal para compor a classe normal e a mesma quantidade de amostras do restante das classes para compor a classe de anomalia. Por meio dos resultados das técnicas do pré-processamento o *dataset* foi dividido de três modos distintos para serem processadas com ML:

1. **Dataset completo -** Nehuma amostra ou atributo foram retirados.

2. **Dataset incompleto -** Foram retirados os atributos 4, 9, 16, 19 e 22, para evitar o problema de correlação entre atributos identificados pela correlação de *Spearman*.

3. **Dataset duas classes -** Foram escolhidas as 2000 amostras da classe de tráfego normal e 2000 amostras de forma estratificada das outras classes do tráfego para compor uma nova classe de tráfego anormal. As classes são do atributo 26 da Tabela 4.1.

Para a execução com o método não supervisionado k-means, foi usado técnicas de PCA (*Principal Component Analysis*) com o *Elbow Method*. A finalidade é identificar o melhor número de componentes gerados pelo PCA para ser usado na clusterização com o algoritmo K-means. Além disso, analisar se há alguma relação entre os clusters e os conjuntos de amostras classificadas manualmente. Pesquisas como (SEBASTIÁN et al., 2016) e (FäRBER et al., 2010), usam métodos não supervisionados para criar *labels* em amostras de dados e comparar se possuem relação com a classificação manual. Depois de todos esses processos iniciou-se a etapa de processamento dos algoritmos de ML.

4.6 Processamento dos Algoritmos de ML

Os algoritmos usados nesta pesquisa são o MLP, SVM, *Decision Tree*, *Random Forest*, *Naive Bayes*, *Gradient Boosting*, *K-means* e KNN, sendo provenientes da biblioteca *Sickt Learning* (PEDREGOSA et al., 2011). Para cada uma das divisões do *dataset*, os algoritmos foram executados com a

técnica de Hiper parametrização implementada na função *GridSearchCV* [9] e com *Cross-validation* da própria função. A Tabela 4.2, mostra os valores usados na técnica de hiper parametrização *GridSearchCV*.

Tabela 4.2 – Hiperparâmetros dos algoritmos de ML

Algoritmos	**Valores dos Hiperparâmetros**
Naive Bayes	var_smoothing (intervalo de 0 a -9 em escala logarítmica)
MLP	activation=('identity', 'logistic','tanh', 'relu'), solver=('lbfgs', 'sgd', 'adam'), alpha=(0.0001, 0.001, 0.01), learning_rate=('constant', 'invscaling', 'adaptative'), max_iter=(30, 60, 90, 120, 150), hidden_layer_sizes=(2,3,4,5)
Árvore de Decisão	criterion=('gini', 'entropy'), max_depth=(4,6,8,12,15), min_samples_split=(5,10,20,30,40,50), max_leaf_nodes=(2,3,4,5,6,7,8,9,10)
Random Forest	criterion=('gini', 'entropy'), max_depth=(6,8,12,15,20,30,40), min_samples_split=(5,10,20,30,40,50), min_samples_leaf=(10,20,30,40,50), max_leaf_nodes=(2,3,4,5,6,7,8,9,10), bootstrap=(True, False)
SVM	kernel=("linear", "poly", "rbf", "sigmoid"), C=(0.1, 0.5, 1, 5, 10, 40, 70), gamma=(1,0.1,0.01,0.001)
KNN	n_neighbors=(1,2,3,4,5,7,12,19,25,33,45,56,70), weights=("distance", "uniform"), metric=("euclidean", "manhattan")
Gradient Boosting	loss=('deviance','exponential'), learning_rate=(0.01, 0.05, 0.1, 0.15), n_estimators=(50,100,150,200), subsample=(0.5, 1.0, 1.5), criterion=('friedman_mse', 'mse'), n_iter_no_change=(5,10,15), tol=(0.01, 0.03, 0.05)

Fonte: Do autor (2021).

4.7 Análise dos resultados

O análise do desempenho e resultados dos algoritmos de ML será feita em duas etapas diferentes. A primeira delas é o cálculo da acurácia (ACC), em que utiliza as informações de verdadeiro positivo (TP - *True Positive*), verdadeiro negativo (TN - *True Negative*), falso positivo (FP - *False Positive*) e falso negativo (FN - *False NEgative*), para fazer o seu cálculo como mostrado na Fórmula 4.1. O uso da acurácia será aplicado em cada execução dos algoritmos de ML em cada divisão do *dataset*, e com isso, obter os três melhores modelos baseados na sua acurácia.

[9] GridSearchCV - https://scikit-learn.org/stable/modules /generated/sklearn.model_selection.GridSearchCV.html

$$ACC = \frac{TP+TN}{TP+TN+FP+FN} \tag{4.1}$$

Ainda na primeira etapa de análise, foi usado o teste estatístico *Dietterich* 5*x*2 *paired T-test* nos modelos de ML, com o objetivo de confirmar se o modelo com a mais alta acurácia é estatisticamente diferente de outros modelos. Os valores *T-statistics* obtidos do teste *Dietterich* 5*x*2 *paired T-test* seguem a distribuição T e considera 5 graus de liberdade, em que a hipótese nula é a de que dois determinados modelos de ML têm a mesma performance. Assim, se o *P-value* obtido é menor que o valor de α=0.05 rejeitamos a hipótese nula e consideramos que os modelos são diferentes (RASCHKA, 2018). Os resultados do teste estatístico para cada divisão do *dataset* foi registrado em tabelas para uma melhor visualização.

A segunda etapa é para fazer a escolha do melhor modelo dentre os melhores modelos gerados na primeira parte, esse processo será pelo número de classes identificadas por cada modelo com novos tráfegos da rede. Os novos tráfegos serão gerados da mesma forma que foram gerados os dataset. Os modelos serão acoplados ao controlador SDN que vai gerar regras de bloqueios do tráfego com base nos resultados das classificações destes modelos. O algoritmo da figura 4.3 mostra como é feito a classificação e criação da regra. A regra de bloqueio é construída com o endereço MAC do host de origem e todas as porta de saída do *switche* em que foi descoberto o tráfego malicioso, depois é enviada ao *switche* para bloquear o fluxo.

Figura 4.3 – Geração de Regras Automáticas pelo Controlador

Algorithm 1: Geração de Regras Automáticas pelo Controlador

```
procedure GERARREGRAS(AmostraDeTrafego);
    ClasseResultado = ModeloMLClassifica(AmostraDeTrafego);
    if ClasseResultado == ClasseMaliciosa then
        Gerar regra de bloqueio;
        Enviar regra para o switche;
end
```

Fonte: Do autor (2021).

Expor os modelos acoplados ao controlador a novos tráfegos de rede, permite identificar qual é o método de divisão do dataset e modelo, mais adequado para o problema de classificar as anomalias da SDN.

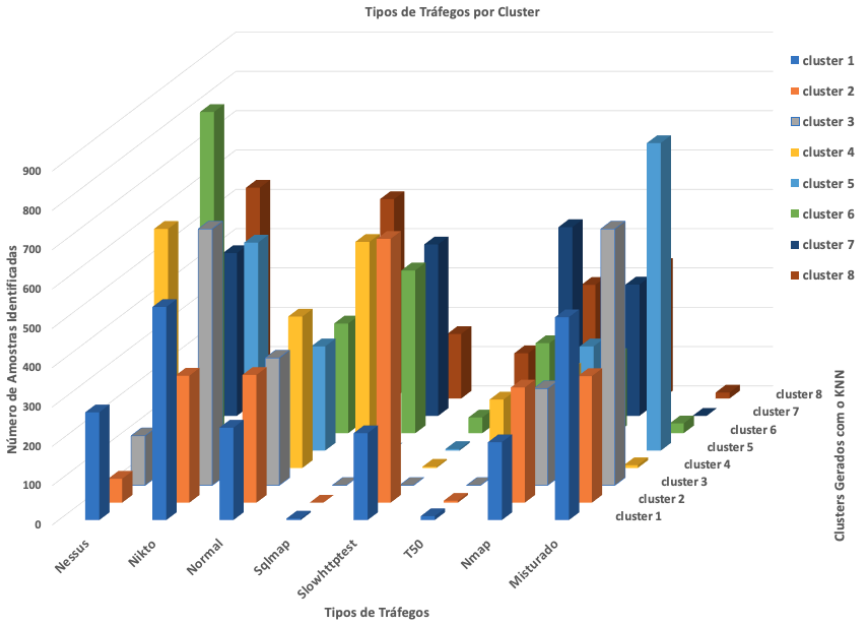
5 RESULTADOS

Esta seção mostra os resultados obtidos das execuções dos algoritmos de ML e a validação dos modelos acoplados ao controlador. Os resultados serão apresentados de acordo com método de execução do algoritmo e divisão do *dataset*. Para avaliar a performance dos algoritmos, os experimentos foram executados em várias rodadas de treinamento, variando o tamanho dos dados de treino e teste.

5.1 Método Não Supervisionado com Dataset Completo

Foram criados os clusters com o K-means usando o valor de K igual ao número de classes do *dataset* por possuir baixa *inertia* pelos resultados do PCA com *Elbow Method*. Após esse processo, as amostras foram agrupadas por cluster e identificadas pela sua classe. Esse resultado pode ser observado na Figura 5.1. Os tráfegos de rede, anômalos ou não, possuem semelhanças por usarem as mesmas estruturas de protocolos e padrões, isso dificulta a tarefa de clusterização e classificação dos tráfegos. Os resultados do método não supervisionado reforçaram essa ideia de semelhança dos tráfegos ao apresentar clusters não homogêneos. Deste modo, não se mostra como um bom método de classificação. Esse mesmo comportamento foi observado no processamento nas outras formas de divisão do *dataset* usando o método não supervisionado.

Figura 5.1 – Clusters formados pelo K-means e discriminados pelos tipos de tráfegos por cluster

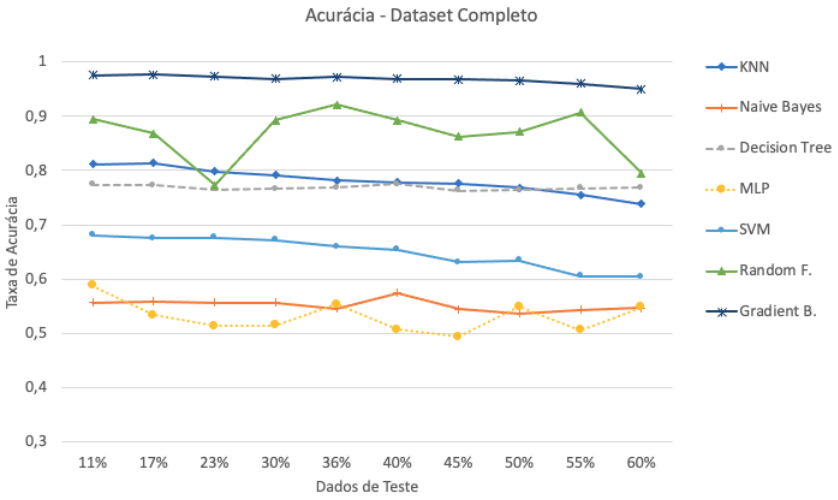


Fonte: Do autor (2021).

5.2 Método Supervisionado com Dataset Completo

Esse experimento utilizou o dataset sem remover atributos ou classes, no qual gerou o modelo mais complexo. A comparação dos resultados deste experimento com os demais, no geral, mostra a segunda melhor performance. O algoritmo *Gradient Boosting* apresentou uma acurácia em torno de 97%, sendo melhor que os outros algoritmos. A Figura 5.2 mostram os resultados e A Tabela 5.1 confirma estatisticamente que o *Gradient Boosting* é diferente dos outros modelos, por causa dos valores de P (*P-value*) serem muito menores que o valor de α=0.05, chegando ao ponto de que nem 4 casas decimais mostrarem eles. O Random Forest classifica uma amostra pela maioria de votos e usa um comitê de Decision Tree internamente, por isso uma melhor performance que a Decision Tree. O KNN tem boa performance com poucos dados, diferente do SVM e MLP, que precisam de maiores quantidades de dados no treinamento. O Naive Bayes não possui boa performance quando os atributos são mutualmente dependentes.

Figura 5.2 – Resultados dos algoritmos de ML usando o *dataset* completo.



Fonte: Do autor (2021).

Tabela 5.1 – Resultado do teste de *Dietterich 5x2 paired T-test* com o dataset Completo.

Divisão dos dados	KNN		Naive		MLP		SVM		Árvore D.		Random F.	
	T	P	T	P	T	P	T	P	T	P	T	P
11	-33,3018	0,0000	-23,5046	0,0000	-14,8227	0,0000	-34,5577	0,0000	-13,3204	0,0000	-11,7386	0,0000
17	-31,6756	0,0000	-22,7081	0,0000	-14,3364	0,0000	-32,5816	0,0000	-13,9932	0,0000	-7,2318	0,0007
23	-41,3868	0,0000	-12,8869	0,0000	-8,9204	0,0002	-35,2328	0,0000	-13,6513	0,0000	-13,6918	0,0000
30	-38,3868	0,0000	-20,4313	0,0000	-14,4865	0,0000	-34,8789	0,0000	-13,9217	0,0000	-13,0233	0,0000
36	-38,1325	0,0000	-23,8757	0,0000	-10,2584	0,0000	-33,6738	0,0000	-14,2750	0,0000	-13,1575	0,0000
40	-28,3138	0,0000	-13,7885	0,0000	-9,1271	0,0000	-36,3852	0,0000	-13,2103	0,0000	-10,6814	0,0000
45	-31,6965	0,0000	-23,2910	0,0000	-20,7683	0,0000	-35,4105	0,0000	-14,4880	0,0000	-12,5140	0,0000
50	-35,5790	0,0000	-24,4191	0,0000	-4,2412	0,0000	-31,0376	0,0000	-13,7758	0,0000	-15,2311	0,0000
55	-31,0845	0,0000	-21,3319	0,0000	-14,7575	0,0000	-31,8101	0,0000	-13,5606	0,0000	-14,7928	0,0000
60	-31,7317	0,0000	-17,2583	0,0000	-17,9977	0,0000	-36,8534	0,0000	-12,9006	0,0000	-18,0760	0,0000

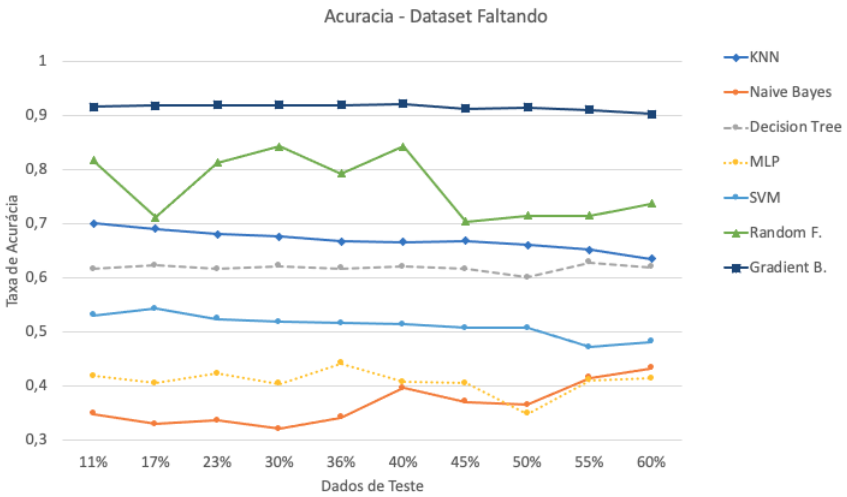
Fonte: Do autor (2021).

5.3 Método Supervisionado com Dataset incompleto

Os resultados deste experimento em comparação com os outros experimentos foram os piores resultados. Apesar disso, é possível fazer a conjectura de que os atributos selecionados para compor o *dataset* são relevantes, pois a remoção de alguns deles diminuiu o desempenho de classificação dos algoritmos de ML, mesmo com o problema de correlação entre alguns atributos.

O algoritmo *Gradient Boosting* apresentou os melhores resultados dentro os outros algoritmos, com a acurácia em torno de 90%. A Figura 5.3 mostra os resultados dos experimentos e na Tabela 5.2 e confirma estatisticamente que o *Gradient Boosting* é diferente dos outros modelos, por causa dos valores de P (*P-value*) serem muito menores que o valor de α=0.05, chegando ao ponto de que nem 4 casas decimais mostrarem eles. O *Random Forest* teve melhor desempenho que o *Decision Tree*, por usar um conjunto de *Decision Tree* durante a classificação. O SVM e MLP, obtiveram menor desempenho, o que poderia ter sido melhor com um maior número de amostras. O KNN sendo um modelo simples obteve melhores resultados que o MLP, SVM, *Decision Tree* e *Naive Bayes*. Por causa da dependência dos dados o *Naive Bayes* obteve os piores resultados.

Figura 5.3 – Resultados dos algoritmos de ML usando o *dataset* incompleto.



Fonte: Do autor (2021).

5.4 Método Supervisionado com Dataset de duas classes

A acurácia de todos os algoritmos explorados neste cenário foram melhores em comparação com os resultados obtidos nas outras divisões do *dataset*. Um forte indício destes resultados é que um menor número de classes torna o problema de classificação menos complexo. O Gradient Boosting obteve os melhores resultados ao se comprar com os outros algoritmos, em torno de 98% de acurácia, mostrado na Figura 5.4. Além disso, a Tabela 5.3 confirma estatisticamente que o Gradient Boosting

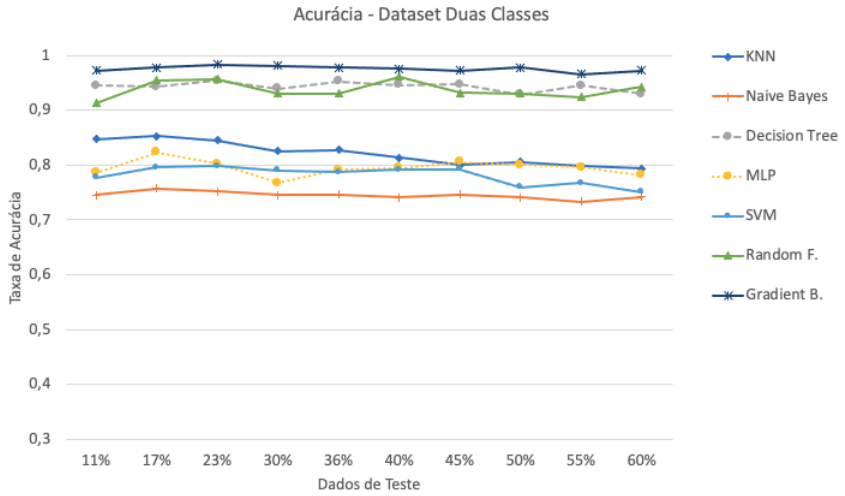
Tabela 5.2 – Resultado do teste de*Dietterich 5x2 paired T-test* com o dataset Incompleto.

Divisão dos dados	KNN		Naive		MLP		SVM		Árvore D.		Random F.	
	T	P	T	P	T	P	T	P	T	P	T	P
11	-50,7754	0,0000	-14,9596	0,0000	-20,1209	0,0000	-48,2133	0,0000	-45,0532	0,0000	-7,2658	0,0007
17	-43,2871	0,0000	-23,9230	0,0000	-31,5621	0,0000	-43,9667	0,0000	-43,7969	0,0000	-10,5150	0,0001
23	-31,2052	0,0000	-16,0221	0,0000	-25,6740	0,0000	-130,9202	0,0000	-33,1133	0,0000	-19,5304	0,0000
30	-37,2346	0,0000	-15,9333	0,0000	-18,6789	0,0000	-76,8345	0,0000	-49,7593	0,0000	-10,5407	0,0001
36	-39,3629	0,0000	-16,2477	0,0000	-24,3729	0,0000	-104,9898	0,0000	-61,1124	0,0000	-6,1937	0,0016
40	-58,6538	0,0000	-13,1367	0,0000	-26,9109	0,0000	-97,3688	0,0000	-43,3565	0,0000	-8,7044	0,0003
45	-57,9416	0,0000	-16,2405	0,0000	-18,8267	0,0000	-103,2068	0,0000	-66,0710	0,0000	-15,8023	0,0000
50	-40,9891	0,0000	-15,2688	0,0000	-18,8785	0,0000	-80,3288	0,0000	-42,8834	0,0000	-24,9865	0,0000
55	-40,1138	0,0000	-15,6055	0,0000	-34,4724	0,0000	-124,3416	0,0000	-78,8400	0,0000	-11,6777	0,0000
60	-39,4627	0,0000	-15,8942	0,0000	-14,6241	0,0000	-79,2704	0,0000	-97,5437	0,0000	-15,3221	0,0000

Fonte: Do autor (2021).

é diferente dos outros modelos, por causa dos valores de P (*P-value*) serem menores que o valor de α=0.05, chegando ao ponto de que nem 4 casas decimais mostrarem eles. Por causa da baixa complexidade e similaridade interna o Random Forest e o Decision Tree obtiveram resultados similares. O MLP e o SVM obtiveram resultados similares que poderiam ter sido melhores com mais amostras de dados. Apesar do menor número de classes, o Naive Bayes ainda possui os piores resultados comparados aos outros algoritmos.

Figura 5.4 – Resultados dos algoritmos de ML usando o *dataset* de duas classes.



Fonte: Do autor (2021).

5.5 Validação dos modelos

O tráfego gerado para cada uma das classes foi menor do que o usado no treinamento dos modelos, sendo equivalente a 761 amostras de cada classe do *dataset*. Obter bons resultados com um modelo de ML no treinamento não significa que o mesmo terá um bom desempenho durante uma validação. A Tabela 5.4, mostra o resultado das classificações. Os modelos "Completo"e "Incom-

Tabela 5.3 – Resultado do teste de *Dietterich 5x2 paired T-test* com o dataset Duas Classes.

Divisão	KNN		Naive		MLP		SVM		Árvore D.		Random F.	
dos dados	T	P	T	P	T	P	T	P	T	P	T	P
11	-7,4151	0,0007	-28,4087	0,0000	-12,5307	0,0000	-27,9956	0,0000	-6,6007	0,0012	-3,7239	0,0136
17	-5,7343	0,0022	-27,1528	0,0000	-5,7058	0,0023	-30,8679	0,0000	-6,9553	0,0009	-9,5668	0,0002
23	-6,0006	0,0018	-28,3406	0,0000	-6,0636	0,0017	-69,7607	0,0000	-6,7337	0,0011	-4,9810	0,0041
30	-6,0435	0,0017	-27,8513	0,0000	-9,9371	0,0001	-29,6052	0,0000	-9,3214	0,0002	-4,3293	0,0075
36	-5,7421	0,0022	-25,8479	0,0000	-6,8902	0,0009	-32,3668	0,0000	-11,6976	0,0000	-5,8663	0,0020
40	-7,9863	0,0005	-29,2951	0,0000	-5,4765	0,0027	-28,0539	0,0000	-8,7389	0,0003	-1,1681	0,2954
45	-7,9595	0,0005	-30,0791	0,0000	-17,4175	0,0000	-29,8363	0,0000	-8,3189	0,0004	-2,5974	0,0484
50	-8,8766	0,0003	-26,4614	0,0000	-8,5954	0,0003	-34,9076	0,0000	-7,2234	0,0007	-6,6086	0,0011
55	-9,0049	0,0002	-25,3052	0,0000	-12,4284	0,0000	-31,0794	0,0000	-7,0238	0,0009	-3,9049	0,0113
60	-5,1743	0,0035	-25,3554	0,0000	-10,3870	0,0001	-35,7004	0,0000	-7,7011	0,0005	-2,9440	0,0321

Fonte: Do autor (2021).

pleto"mostram que, apesar das altas taxas de acurácia, os mesmos não conseguiram discriminar os tipos de tráfegos. O modelo "Duas Classes"conseguiu identificar a maior parte dos tráfegos maliciosos, chegando a 100% para algumas classes. Além disso, conseguiu uma alta taxa de acerto para o tráfego normal ao se comprar com os outros modelos.

Tabela 5.4 – Resultado da classificação durante a etapa de validação.

Tráfego Gerado	Modelo-Completo	Modelo-Incompleto	Modelo-Duas Classes
Nessus	70%	31%	99%
Nmap	60%	2%	76%
Slowhttptest	4%	4%	100%
T50	5%	5%	100%
Nikto	54%	29%	100%
Sqlmap	64%	53%	98%
Normal	25%	39%	84%
Misturado	79%	67%	85%

Fonte: Do autor (2021).

5.6 Resposta das Hipóteses

No fim desta pesquisa obteve-se as seguintes respostas às hipóteses levantadas no começo deste trabalho:

1. Dado um alto volume de tráfego de uma SDN é possível detectar anomalias em tempo hábil?

   (a) Sim, entretanto, a capacidade de processamento está condicionado aos equipamentos utilizados nos testes e esta pesquisa não deu foco nos testes de capacidade de processamento de grandes volumes de tráfego.

2. O controlador consegue definir regras para bloquear as anomalias sem prejudicar o tráfego da rede?

   (a) Sim, entretanto, ao utilizar um modelo de ML para classificar o tráfego da rede para definir se é necessário bloqueá-lo ou não pelo controlador, a sua eficácia em bloquear uma anomalia está condicionada a acurácia do modelo de ML.

3. Os *datasets* existentes e disponíveis representam os atuais ataques nas SDNs?

   (a) Até o momento desta pesquisa não foi encontrado um *dataset* com ataques atualizados, por isso houve a necessidade de criar um com ataques mais atualizados.

4. As pesquisas atuais permitem identificar a variabilidade dos ataques atuais nas redes?

   (a) Até o momento desta pesquisa e considerando os trabalhos analisados não, pois em muitos trabalhos os *datasets* eram antigos, não tinham a comparação entre os modelos de ML, não utilizaram métodos estatísticos para complementar os resultados obtidos, não validaram os modelos de ML gerados ou a combinação desses pontos levantados.

6 CONCLUSÃO

A análise do tráfego de rede é uma técnica indispensável para identificar anomalias da rede. Este trabalho analisou o problema de classificação de anomalias de tráfego dentro de uma SDN usando técnicas de ML. Por causa da falta de *datasets*, foi implementado um com os pacotes do tráfego dentro da SDN, gerado com ferramentas de segurança e os acessos das aplicações do servidor alvo.

Foi analisado um comitê de algoritmos de ML diferentes para se obter o melhor modelo na etapa de treinamento, no qual foi usado para classificar novos tráfegos dentro de uma SDN, por meio regras geradas automaticamente pelo controlador SDN. O melhor modelo foi com o algoritmo de *Gradient Boosting*, com a taxa de acurácia de até 98% durante o treinamento e até 100% de acerto na classificação de tráfego malicioso. Entretanto, alguns modelos não tiveram bom desempenho durante a validação, apesar das boas taxas de acurácia.

Apesar de resultados satisfatórios, deve se considerar as características inerentes ao problema de análise do tráfego de rede. Os agentes maliciosos utilizam a mesma estrutura que os agentes normais, e altas taxas de acurácia nem sempre significam que se tenham bons modelos. Por exemplo, um dos problemas da classificação de tráfegos dentro da SDN é que após a criação de uma regra para liberar um fluxo, o conteúdo do fluxo pode se tornar um ataque e passar despercebido por algum mecanismo de proteção, enquanto a regra de liberação estiver ativa. A próxima seção apresenta as propostas de continuação desta pesquisa.

6.1 Proposta de Continuidade

Como proposta de continuidade podem ser executados as seguintes ações:

1. Alterações do número de parâmetros que são capturados pela ferramenta de captura do tráfego e construção do *dataset*.
2. Criação de um *dataset* com *switchs* reais e sua posterior análise com ML.
3. Alterar o ambiente de dados para funcionar com uma maior carga na rede e avaliar a tolerância do software de captura e criação do *dataset*.
4. Testar outras arquiteturas de rede.
5. Diversificar mais os tipos de ataques e a quantidade de *hosts* a serem atacados.
6. Adicionar outros tipos de pacotes de rede.
7. Explorar novos *frameworks* de SDN para construção do controlador e *dataset*.

REFERÊNCIAS

Abubakar, A.; Pranggono, B. Machine learning based intrusion detection system for software defined networks. In: **2017 Seventh International Conference on Emerging Security Technologies (EST)**. [S.l.: s.n.], 2017. p. 138–143. ISSN 2472-7601.

AHMAD, A. et al. Evaluation of machine learning techniques for security in sdn. 09 2020.

ALBIN, E.; ROWE, N. A realistic experimental comparison of the suricata and snort intrusion-detection systems. In: . [S.l.: s.n.], 2012. p. 122–127. ISBN 978-1-4673-0867-0.

BATISTA, G. E. A. P. A.; SILVA, D. F. How k-nearest neighbor parameters affect its performance. 2009.

BEALE, E. F. R. **Handbook of neural computation**. [s.n.], 1997. 1-436 p. ISBN 9781420050646. Disponível em: <https://www.worldcat.org/title/handbook-of-neural-computation/oclc/646213711>.

BREIMAN, L. et al. **Classification And Regression Trees**. [S.l.: s.n.], 2017. 1-358 p. ISBN 9781315139470.

BROWN, L.; STALLINGS, W. **Criptografia e segurança de redes: Princípios e Práticas**. 4. ed. [S.l.]: Pearson Prentice Hall, 2008. ISBN 9788576051190.

BROWN, L.; STALLINGS, W. **Segurança de Computadores: Princípios e Práticas**. Elsevier Editora Ltda., 2017. ISBN 9788535264500. Disponível em: <https://books.google.com.br/books?id=y2DcAwAAQBAJ>.

CHOMBOON, K. et al. An empirical study of distance metrics for k-nearest neighbor algorithm. p. 280–285, 01 2015.

CISCO. **Snort**. [S.l.], 2019. Disponível em: <https://www.snort.org>.

Comaneci, D.; Dobre, C. Securing networks using sdn and machine learning. In: **2018 IEEE International Conference on Computational Science and Engineering (CSE)**. [S.l.: s.n.], 2018. p. 194–200.

CORTES, C.; VAPNIK, V. Support-vector networks. **Mach. Learn.**, Kluwer Academic Publishers, USA, v. 20, n. 3, p. 273–297, set. 1995. ISSN 0885-6125. Disponível em: <https://doi.org/10.1023/A:1022627411411>.

Dabbagh, M. et al. Software-defined networking security: pros and cons. **IEEE Communications Magazine**, v. 53, n. 6, p. 73–79, June 2015. ISSN 0163-6804.

DHANABAL, L.; SHANTHARAJAH, S. P. A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. In: . [S.l.: s.n.], 2015.

DIETTERICH, T. Ensemble methods in machine learning. **Multiple Classifier Systems: First International Workshop, MCS 2000, Lecture Notes in Computer Science**, p. 1–15, 01 2000.

ELSAYED, M.; LE-KHAC, N.-A.; JURCUT, A. Insdn: A novel sdn intrusion dataset. **IEEE Access**, 09 2020.

FOUNDATION, O. I. S. **Suricata**. [S.l.], 2019. Disponível em: <https://suricata-ids.org>.

FOUNDATION, O. N. **OpenFlow Switch Specification**. [S.l.], 2012. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf>.

FOUNDATION, O. N. **Software-Defined Networking: The New Norm for Networks**. [S.l.], 2012. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>.

FOUNDATION, O. N. **SDN Architecture Overview**. [S.l.], 2013. Disponível em: <https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/SDN-architecture-overview-1.0.pdf>.

FOUNDATION, O. N. **SDN Architecture**. [S.l.], 2014. Disponível em: <https://www.opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf>.

FOUNDATION, T. O. **OWASP Top 10 - 2017**. [S.l.], 2019. Disponível em: <https://www.owasp.org/images/7/72/OWASP_Top_10-2017_\%28en\%29.pdf.pdf>.

FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. **Annals of Statistics**, v. 29, p. 1189–1232, 2000.

FäRBER, I. et al. On using class-labels in evaluation of clusterings. 01 2010.

Gangwal, A.; Conti, M.; Gaur, M. S. Panorama: Real-time bird's eye view of an openflow network. In: **2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)**. [S.l.: s.n.], 2017. p. 204–209.

GAO, S. et al. Software-defined firewall: Enabling malware traffic detection and programmable security control. In: . [S.l.: s.n.], 2018. p. 413–424.

Garg, G.; Garg, R. Detecting anomalies efficiently in sdn using adaptive mechanism. In: **2015 Fifth International Conference on Advanced Computing Communication Technologies**. [S.l.: s.n.], 2015. p. 367–370. ISSN 2327-0659.

GUO, G. et al. Knn model-based approach in classification. 08 2004.

HADIANTO, R.; PURBOYO, T. A simulation study of sdn defense against botnet attack based on network traffic detection. **ARPN Journal of Engineering and Applied Sciences**, v. 13, p. 3489–3494, 05 2018.

HARTIGAN, J. A. Bayes theory. In: ____. **Bayes Theory**. New York, NY: Springer-Verlag New York, 1983. p. XII, 146. ISBN 978-1-4613-8244-7. Disponível em: <https://doi-org.ez26.periodicos.capes.gov.br/10.1007/978-1-4613-8242-3>.

HE, F.; DING, X. Improving naive bayes text classifier using smoothing methods. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 703–707, 2007.

Henni, D.; Hadjaj-Aoul, Y.; Ghomari, A. Probe-sdn: A smart monitoring framework for sdn-based networks. In: **2016 Global Information Infrastructure and Networking Symposium (GIIS)**. [S.l.: s.n.], 2016. p. 1–6. ISSN 2150-329X.

JANKOWSKI, D.; AMANOWICZ, M. Intrusion detection in software defined networks with self-organized maps. v. 2015, p. 3–9, 01 2015.

Jeong, C. et al. Scalable network intrusion detection on virtual sdn environment. In: **2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)**. [S.l.: s.n.], 2014. p. 264–265.

KASPERSKY. **KSN Report: Ransoware and malicious cryptominers 2016-2018**. [S.l.], 2019 (accessado 17, 08, 2019). Disponível em: <https://media.kasperskycontenthub.com/wp-content/uploads/sites/58/2018/06/27125925/KSN-report_Ransomware-and-malicious-cryptominers_2016-2018_ENG.pdf>.

Kim, J.; Shin, S. Software-defined honeynet: Towards mitigating link flooding attacks. In: **2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)**. [S.l.: s.n.], 2017. p. 99–100. ISSN 2325-6664.

Lai, Y. et al. F1ow-based anomaly detection using multilayer perceptron in software defined networks. In: **2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)**. [S.l.: s.n.], 2019. p. 1154–1158.

Lim, S. et al. A sdn-oriented ddos blocking scheme for botnet-based attacks. In: **2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)**. [S.l.: s.n.], 2014. p. 63–68. ISSN 2165-8528.

Liu, Y. et al. A survey: Typical security issues of software-defined networking. **China Communications**, v. 16, n. 7, p. 13–31, July 2019.

LUNGER, G. F. **Inteligência Artificial**. 6. ed. Rua Nelson Francisco, 26, CEP 02712-100, São Paulo - SP: Pearson Prentice Hall, 2013. ISBN 978-8581435503.

MCKEOWN, N. et al. Openflow: Enabling innovation in campus networks. **Computer Communication Review**, v. 38, p. 69–74, 04 2008.

Mendiola, A. et al. A survey on the contributions of software-defined networking to traffic engineering. **IEEE Communications Surveys Tutorials**, v. 19, n. 2, p. 918–953, Secondquarter 2017.

MININET. **Mininet**. [S.l.], 2019. Disponível em: <http://mininet.org>.

Mohammadi, R. et al. Practical extensions to countermeasure dos attacks in software defined networking. In: **2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2017. p. 1–6.

Nagpal, B. et al. Ddos tools: Classification, analysis and comparison. In: **2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)**. [S.l.: s.n.], 2015. p. 342–346.

Nanda, S. et al. Predicting network attack patterns in sdn using machine learning approach. In: **2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)**. [S.l.: s.n.], 2016. p. 167–172.

NETO, H. S. **Sistema de detecção de intrusão em redes de computadores com técnicas de inteligência computacional**. Dissertação (Mestrado) — Universidade Federal de Lavras, Lavras, 01 2017.

NETO, H. S.; LACERDA, W.; BOELL, V. grudtner. Reconhecimento de intrusão em redes de computadores utilizando pybrain. In: . [S.l.: s.n.], 2015.

NOX. **NOX**. 2012. Disponível em: <https://github.com/noxrepo/nox>.

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

POSTEL, J. **User Datagram Protocol**. [S.l.], 1980. <http://www.rfc-editor.org/rfc/rfc768.txt>. Disponível em: <http://www.rfc-editor.org/rfc/rfc768.txt>.

POSTEL, J. **Transmission Control Protocol**. [S.l.], 1981. Disponível em: <http://www.rfc-editor.org/rfc/rfc793.txt>.

Prakash, A.; Priyadarshini, R. An intelligent software defined network controller for preventing distributed denial of service attack. In: **2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)**. [S.l.: s.n.], 2018. p. 585–589.

PROJECT., O. A. L. F. C. **OpenDaylight**. 2016. Disponível em: <https://www.opendaylight.org/>.

RASCHKA, S. Model evaluation, model selection, and algorithm selection in machine learning. **arXiv preprint arXiv:1811.12808**, 2018.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. **Psychological review**, v. 65 6, p. 386–408, 1958.

RUSSEL, P. N. S. **Inteligência Artificial**. [S.l.]: Elsevier Editora Ltda., 2013. ISBN 9788535237016.

RYU. **Ryu SDN controller**. 2012. Disponível em: <https://osrg.github.io/ryu/>.

Sanjaa, B.; Chuluun, E. Malware detection using linear svm. v. 2, p. 136–138, 2013.

Santos da Silva, A. et al. Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In: **NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium**. [S.l.: s.n.], 2016. p. 27–35. ISSN 2374-9709.

Scott-Hayward, S.; Natarajan, S.; Sezer, S. A survey of security in software defined networks. **IEEE Communications Surveys Tutorials**, v. 18, n. 1, p. 623–654, Firstquarter 2016. ISSN 1553-877X.

SEBASTIÁN, M. et al. Avclass: A tool for massive malware labeling. In: MONROSE, F. et al. (Ed.). **Research in Attacks, Intrusions, and Defenses**. Cham: Springer International Publishing, 2016. p. 230–253. ISBN 978-3-319-45719-2.

SHALIMOV, A. et al. Advanced study of sdn/openflow controllers. In: . [S.l.: s.n.], 2013.

SOKOLOV, V. et al. A network analytics system in the sdn. In: . [S.l.: s.n.], 2014. p. 1–3.

Sokolov, V. et al. A network analytics system in the sdn. In: **2014 International Science and Technology Conference (Modern Networking Technologies) (MoNeTeC)**. [S.l.: s.n.], 2014. p. 1–3.

SPEARMAN, C. The proof and measurement of association between two things. **The American Journal of Psychology**, University of Illinois Press, v. 15, n. 1, p. 72–101, 1904. ISSN 00029556. Disponível em: <http://www.jstor.org/stable/1412159>.

TANENBAUM, A. S. **Redes de computadores**. 5. ed. Rua Nelson Francisco, 26, CEP 02712-100, São Paulo - SP: Pearson Prentice Hall, 2011. ISBN 978857605924.

Thupae, R. et al. Machine learning techniques for traffic identification and classifiacation in sdwsn: A survey. In: **IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society**. [S.l.: s.n.], 2018. p. 4645–4650. ISSN 2577-1647.

Tidjon, L. N.; Frappier, M.; Mammar, A. Intrusion detection systems: A cross-domain overview. **IEEE Communications Surveys Tutorials**, p. 1–1, 2019. ISSN 1553-877X.

Tin Kam Ho. Random decision forests. v. 1, p. 278–282 vol.1, 1995.

VELHO, J. A. **Tratado de Computação Forense**. 1. ed. Av. Marechal Rondon, 473, Jd. Chapadão - 13070-172, Campinas-SP: Millennium Editora, 2016. ISBN 9788576253358.

Wang, Y.; De Lin, C. Learning by bagging and adaboost based on support vector machine. v. 2, p. 663–668, 2007.

YARIMTEPE, O.; DALKILIC, G.; OZCANHAN, M. Ddos prevention techniques. 05 2015.

YING, X. An overview of overfitting and its solutions. **Journal of Physics: Conference Series**, v. 1168, p. 022022, 02 2019.

ZHU, L. et al. **SDN Controllers: Benchmarking Performance Evaluation**. 2019.