UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Detecção de ataques DDoS em ambientes SDN/NFV utilizando algoritmos de aprendizagem de máquina não supervisionados em fluxos de dados

Dissertação de Mestrado

João Ribeiro de Almeida Neto

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

João Ribeiro de Almeida Neto

Detecção de ataques DDoS em ambientes SDN/NFV utilizando algoritmos de aprendizagem de máquina não supervisionados em fluxos de dados

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito final para a obtenção do título de mestre em Ciência da Computação.

Orientador(a): Admilson Ribamar Ribeiro

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
COORDENAÇÃO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da Sessão Solene de Defesa da Dissertação do Curso de Mestrado em Ciência da Computação-UFS.
Candidato: JOÃO RIBEIRO DE ALMEIDA NETO

Em 29 dias do mês de julho do ano de dois mil e vinte um, com início às 14h00min, realizou-se na Sala virtual https://meet.jit.si/defesaJoaoRibeiroProccUfs. A Sessão Pública de Defesa de Dissertação de Mestrado do candidato **JOÃO RIBEIRO DE ALMEIDA NETO**, que desenvolveu o trabalho intitulado: **"*Detecção de ataques DDoS em ambientes SDN/NFV utilizando algoritmos de aprendizagem de máquina não supervisionados em fluxos de dados*",** sob a orientação do Prof. Dr. **Admilson de Ribamar Lima Ribeiro**. A Sessão foi presidida pelo Prof. Dr. **Admilson de Ribamar Lima Ribeiro** (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, Prof. Dr. **Anderson Clay Nascimento** (UW) e em seguida, a Prof. Dr. **Renê Pereira de Gusmão** (PROCC/UFS). Após as discussões, a Banca Examinadora reuniu-se e considerou o(a) mestrando(a) Aprovado "*(aprovado/reprovado)"*. Atendidas as exigências da Instrução Normativa 01/2017/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEPE), Resolução nº 25/2014/CONEPE e da Portaria nº 413 de 27 de maio de 2020 (Banca por videoconferência) que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Cidade Universitária "*Prof. José Aloísio de Campos",* 29 de julho de 2021.

Admilson Ribeiro.
Prof. Dr. Admilson de Ribamar Lima Ribeiro
(PROCC/UFS)
Presidente

Renê P. de Gusmão
Prof. Dr. Renê Pereira de Gusmão
(PROCC/UFS)
Examinador Interno

Anderson C.A. Nascimento
Prof. Dr. Anderson Clay Alves Nascimento
(UW)
Examinador Externo

João Ribeiro de Almeida Neto
João Ribeiro de Almeida Neto
Candidato

Agradecimentos

Gostaria de agradecer ao nosso bom Deus, por me conceder essa oportunidade e me manter saudável e forte durante essa caminhada. Agradeço também a graça alcançada: poder ter meu pai presente no plano terrestre, após uma longa internação por conta da covid-19.

Agradeço imensamente a toda a minha família. Sandra (mãe), Abílio (pai) e Jordy (irmão), por tudo que fizeram por mim durante toda vida. Tenham certeza de que todas as minhas conquistas são compartilhadas com vocês! A minha esposa Thais, por toda parceria de vida. Muito obrigado por todo apoio durante não só esses 2 anos, mas por toda nossa história escrita e que ainda está nos primeiros capítulos.

Ao Prof. Dr. Admilson de Ribamar Lima Ribeiro, pela sua orientação, ensinamentos e disposição em sempre me ajudar. Aos professores Prof. Dr. Anderson Clayton e Prof. Dr. Renê Pereira por terem aceitado participar desta banca e por todas as sugestões e críticas engrandecedoras.

Aos meus colegas de caminhada Layse e Gabriel que me ajudaram bastante durante esses dois anos de mestrado. Por fim, aos meus amigos que, de alguma forma, foram essenciais para conclusão esse trabalho. Em especial, aos meus amigos Artur, Fernando, Denisson, Italo, Alfran e Daniel, por todas as dicas, pelos ensinamentos, pela enorme ajuda, enfim, por tudo o que fizeram por mim durante todo esse período.

A todos, meu muito obrigado!

Resumo

Segundo dados do *Cisco Visual Networking Index (VNI)*, que visa realizar uma previsão realista baseada em vários níveis e fontes de dados reais, estima-se que o número total de ataques *DDoS* a nível global chegue a 14,5 milhões até 2022. Por esse motivo, fica evidente que é imprescindível se proteger de ataques do tipo *DDoS*. Dessa forma, há necessidade de que novas técnicas de proteção sejam desenvolvidas. Além disso, é preciso que as soluções levem em consideração os requisitos de desempenho e escalabilidade. Aliado a isso, ambientes baseados na arquitetura *SDN/NFV* permitem que os administradores de rede detectem e reajam aos ataques *DDoS* com mais eficiência. Isso porque o controle da rede é centralizado e é possível desenvolver recursos de análise de tráfego baseados em software. Esta dissertação analisa a eficiência e efetividade da utilização de algoritmos de aprendizagem de máquina não supervisionados que trabalham com a estratégia de fluxo de dados na detecção de ataques do tipo *DDoS* em ambientes *SDN/NFV*, por meio de uma análise comparativa. Primeiramente, foi realizado um Mapeamento Sistemático da Literatura, o qual serviu de embasamento para a realização de um primeiro experimento. Em seguida, foi realizada uma Revisão Sistemática da Literatura e foram incluídos os trabalhos que utilizassem aprendizagem de máquina não supervisionada na detecção de ataques *DDoS* e que trabalhassem com a estratégia de fluxo de dados, pois, essa característica é inerente ao ambiente *SDN/NFV*. Dessa maneira, os algoritmos escolhidos foram: *BIRCH*, *Mini-batch k-means*, *Clustream*, *StreamKM++*, *DenStream* e *D-Stream*. Após isso, foi montada uma plataforma para a execução do experimento, assim como foi desenvolvido um *dataset* para ser utilizado. Após a realização dos testes, foi realizada uma análise qualitativa e quantitativa sobre os resultados. A análise qualitativa objetivou comparar o quão efetivo são os algoritmos na detecção de ataques *DDoS* e a análise quantitativa visa comparar a eficiência, neste caso, a velocidade de processamento dos algoritmos nessa detecção. Os resultados obtidos mostram os algoritmos *BIRCH*, *Mini-batch k-means*, *Clustream* e *StreamKM++* obtiveram acurácias em torno de 99%, enquanto *DenStream* e *D-Stream* alcançaram acurárias em torno de 79%. O menor tempo total de execução foi do algoritmo *D-Stream*, enquanto o maior tempo foi do *StreamKM++*. Em vista disso, os algoritmos que se destacaram foram *D-Stream* e *Mini-batch k-means*, já que aquele foi o algoritmo mais rápido e este obteve uma acurácia 25,18% maior que *D-Stream*.

Palavras-chave: *SDN*. *NFV*. *DDoS*. Fluxo de dados. Aprendizagem de Máquina. Segurança.

Abstract

According to data from the Cisco Visual Networking Index (VNI), which aims to make a realistic forecast based on various levels and real data sources, it is estimated that the total number of DDoS attacks on a global level will reach 14.5 million by 2022. For this reason, it is essential to protect yourself from DDoS attacks. Thus, there is a need for new protection techniques to be developed. In addition, solutions need to take into account performance and scalability requirements. In addition, environments based on the SDN/NFV architecture allow network administrators to detect and react to DDoS attacks more efficiently. This is because network control is centralized and software-based traffic analysis capabilities can be developed. This dissertation analyzes the efficiency and effectiveness of using unsupervised machine learning algorithms that work with the data flow strategy in the detection of DDoS type attacks in SDN/NFV environments, through a comparative analysis. First, a Systematic Literature Mapping was carried out, which served as a basis for the realization of a first experiment. Then, a Systematic Literature Review was carried out, and works that used unsupervised machine learning to detect DDoS attacks and that worked with the data flow strategy were included, as this characteristic is inherent to the environment. SDN/NFV. Thus, the chosen algorithms were: BIRCH, Mini-batch k-means, Clustream, StreamKM++, DenStream, and D-Stream. After that, a platform was set up to run the experiment, as well as a dataset was developed. After performing the tests, a qualitative and quantitative analysis of the results was performed. The qualitative analysis aimed to compare how effective the algorithms are in detecting DDoS attacks and the quantitative analysis aimed to compare the efficiency, in this case, the processing speed of the algorithms in this detection. The results obtained show that the algorithms BIRCH, Mini-batch k-means, Clustream, and StreamKM++ obtained accuracy around 99%, while DenStream and D-Stream reached accuracy around 79%. The shortest total execution time was for the D-Stream algorithm, while the longest time was for StreamKM++. Because of this, the algorithms that stood out were D-Stream and Mini-batch k-means, since that was the fastest algorithm, and this one obtained an accuracy 25.18% higher than D-Stream.

Keywords: SDN. NFV. DDoS. Data stream. Machine learning. Security.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACID	*Atomicity Consistency Isolation Durability*
AIS	*Artificial Immune System*
ARP	*Address Resolution Protocol*
AWS	*Amazon Web Services*
BIRCH	*Balanced Iterative Reducing and Clustering using Hierarchies*
CAPEX	*Capital Expenditure*
CF	*Clustering Features*
CORE	*Common Open Research Emulator*
CPU	*Central Process Unit*
CUF	*Coburg Utility Framework*
DBSCAN	*Density-based Spatial Clustering of Applications With Noise*
DDoS	*Distributed Denial of Service*
DHCP	*Dynamic Host Configuration Protocol*
DNS	*Domain Name System*
ETSI	*European Telecommunications Standards Institute*
FDR	*False Discovery Rate*
FNR	*False Negative Rate*
FOR	*False Omission rate*
FPR	*False Positive Rate*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IoT	*Internet of Things*
IP	*Internet Protocol*

IPS	*Intrusion Prevention System*
ISP	*Internet Service Provider*
ISG	*Industry Specification Group*
JSON	*JavaScript Object Notation*
MANO	*NFV Management and Orchestration*
NFV	*Network Functions Virtualization*
NFVI	*NFV Infrastructure*
NFVO	*NFV Orchestrator*
NIDS	*Network Intrusion Detection System*
NOS	*Network Operating System*
NOSQL	*Not Only SQL*
NPV	*Negative Predictive Value*
NVI	*Cisco Visual Network Index*
OPEX	*Operational Expenditure*
P2P	*Peer to Peer*
PPV	*Positive Predictive Value*
RAM	*Random Access Memory*
RGCE	*Realistic Global Cyber Environment*
SARNET	*Secure Autonomous Response Network*
SDN	*Software Defined Networking*
SGD	*Stochastic Gradient Descent*
SGU	*Safe Guard Unit*
SMTP	*Simple Mail Transfer Protocol*
SNMP	*Simple Network Management Protocol*
SSD	*Solid-State Drive*
SVM	*Support Vector Machine*

TCP	*Transmission Control Protocol*
TNR	*True Negative Rate*
TPR	*True Positive Rate*
UDP	*User Datagram Protocol*
VIM	*Virtualized Infrastructure Manager*
VNF	*Virtual Network Functions*
VNFC	*VNF Components*
VNFM	*VNF Manager*
VNI	*Cisco Visual Networking Index*
WAN	*Wide Area Network*
WSGI	*Web Server Gateway Interface*

Sumário

1 Introdução

1.1 Apresentação Geral

Muito se tem discutido acerca do crescimento exponencial do tráfego de dados na Internet, em razão da sua utilização como uma plataforma global de comunicação, abarcando um grande número de serviços e de aplicações dinâmicas e heterogêneas. Desse modo, houve também um aumento na quantidade de dados trafegados nessa rede, assim como no consumo de largura de banda (ALSAEEDI; MOHAMAD; AL-ROUBAIEY, 2019). As redes tradicionais, apesar de sua ampla adoção, são complexas e difíceis de gerenciar, pois, o plano de controle e o plano de dados residem nos dispositivos de redes. Essa característica diminui a flexibilidade e a inovação, trazendo limitações e para diminuir essas limitações, a arquitetura *Software Defined Networking (SDN)* ou Redes Definidas por Software foi desenvolvida (KREUTZ et al., 2014). Esse novo paradigma promete simplificar o gerenciamento da rede, assim como melhorar a utilização dos recursos e incentivar a evolução e a inovação em redes tradicionais (ALSAEEDI; MOHAMAD; AL-ROUBAIEY, 2019).

Um dos pilares que definem as redes *SDN*, definidos por Kreutz et al. (2014), é o fato de existir uma separação do plano de dados do plano de controle. Esse desacoplamento, propicia uma maior incidência de problemas de segurança de redes. Existem diversos tipos de problemas, dentre eles, ataques cibernéticos que visam explorar vulnerabilidades. O ataque de negação de serviço distribuído ou *Distributed Denial of Service (DDoS)* é um exemplo disso. Ele é uma forma especializada de ataque que ocorre em um ambiente amplamente distribuído, ou seja, após infectar um exército de vítimas, o atacante coordena de maneira anônima que as vítimas ataquem ao mesmo tempo o alvo. Os ataques *DDoS* são ameaças de rede bastante prejudiciais nos dias de hoje, eles são capazes de causar interrupções em serviços de tecnologia da informação, a fim de tornar os recursos de um sistema indisponível para os seus utilizadores (BAWANY; SHAMSI; SALAH, 2017).

Na direção da segurança de redes, soluções de inspeção de pacotes ou de detecção de intrução podem ser empregadas a fim de minimizar os riscos e tratar as vulnerabilidades como forma de tentativa de resolução desse problema. Dentre as caraterísticas da arquitetura *SDN*, destaca-se a possibilidade de programar a lógica de controle ou de desenvolver aplicativos de software para a rede, como por exemplo um *firewall* ou *Intrusion Detection System* (*IDS*). Para facilitar a programabilidade da rede, podemos integrar a arquitetura *SDN* com a arquitetura *Network Functions Virtualization (NFV)* ou Funções de Rede Virtualizadas. Essa arquitetura objetiva inovar a prestação de serviços e virtualizar funções anteriormente executadas por dispositivos de hardware específicos. A arquitetura *SDN* oferece um ambiente propício para a implantação de *NFV* (ETSI, 2017). Dessa maneira, os conceitos *SDN* e *NFV* são considerados complementares e compartilham o objetivo de acelerar a inovação dentro das redes, permitindo a programação e automação de aplicações para elas (KREUTZ et al., 2014).

1.2 Motivação

De acordo com uma análise recente, os ataques *DDoS* dobraram sua frequência em um período de apenas seis meses (NEWMAN, 2019). Estimativas globais preveem que o número total de ataques *DDoS* possam chegar a 14,5 milhões até 2022, segundo os dados de 2018 do *Cisco Visual Networking Index (VNI)*. Ainda de acordo com o *VNI*, ataques *DDoS* representam 25% do tráfego de um país enquanto ele está ocorrendo (BARNETT JAIN, 2018). Uma pesquisa do *SecureList Kaspersky* mostra que no primeiro trimestre de 2019 o número de ataques *DDoS* aumentou 84%. Além disso, no segundo trimestre de 2019, houve mais ataques *DDoS* de alto nível do que no trimestre anterior. A China e os EUA se classificaram como os dois principais alvos, com 63,8% e 17,5% dos ataques, respectivamente (CRANE, 2019).

Nesse sentido, dentre as técnicas que podem ser utilizadas para detecção de intrusões, a correspondência de assinaturas é amplamente utilizada, além de possuir uma alta precisão (VIGNA; ROBERTSON; BALZAROTTI, 2004). No entanto, essas metodologias não se adaptam a novos cenário de maneira satisfatória, já que há necessidade de atualização frequente das assinaturas. Ademais, desenvolver e manter um banco de dados de assinaturas atualizado pode ser custoso e inviável, visto que, a todo momento novas anomalias de rede são criadas e novas vulnerabilidades são exploradas. Tudo isso acabaria deixando a base da dados muito grande e dificultaria o seu armazenamento e o tempo de processamento das correspondências. Outra técnica utilizada na detecção de intrusões é o reconhecimento de padrões, que visa classificar os dados com base em informações estatísticas. Os padrões que serão classificados são geralmente grupos de observações ou parâmetros. Portanto, os métodos de reconhecimento de padrões podem ser mais adequados à classificação de tráfego da rede. Para isso, pode-se utilizar métodos de aprendizagem de máquina a fim de construir modelos adequados à classificação de tráfego (YUAN et al., 2010).

1.3 Justificativa

A partir da motivação levantada na seção 1.2, fica evidente a necessidade de se proteger de ataques do tipo *DDoS* dada a sua relevância. Assim, há necessidade de que novas formas de proteção sejam desenvolvidas e que tenham a característica de levar em consideração os requisitos de desempenho e escalabilidade.

Dessa maneira, ambientes baseados na arquitetura *SDN/NFV* permitem que os administradores de rede detectem e reajam aos ataques *DDoS* com mais eficiência. Isso porque, o controle da rede é centralizado e é possível desenvolver recursos de análise de tráfego baseados em software (BHUSHAN; GUPTA, 2019). Assim, há diminuição de custos com a compra de equipamentos, aumento da escalabilidade, aumento da elasticidade e aumento na rapidez de implementação e implantação de novos recursos (BONFIM; DIAS; FERNANDES, 2019) (YI et al., 2018).

Aliado a isso, utilizando a aprendizagem de máquina é possível desenvolver aplicações que detectem anomalias na rede, dentre elas ataques do tipo *DDoS*. A aprendizagem de máquina possui diferentes técnicas e a não supervisionada se destaca neste contexto, pois, possui a característica de não necessitar de um supervisor (SURESH; ANITHA, 2011). Essa técnica pode ser uma alternativa viável, já que o ambiente de rede é altamente dinâmico e ela permite que o próprio sistema consiga separar o tráfego de rede em categorias distintas com adição de novos dados. Ademais, em redes de computadores há a presença de fluxos de dados, que são conjuntos contínuos de dados de tamanho potencialmente ilimitado. Por isso, é necessário que os algoritmos de aprendizagem de máquina utilizados levem em consideração essas restrições e trabalhem com a estratégia de fluxos de dados.

Neste sentido, em uma busca bibliográfica previamente realizada, foram encontrados poucos trabalhos que trataram da detecção de ataques *DDoS* utilizando algoritmos de aprendizagem de máquina não supervisionados que utilizem a estratégia de fluxo de dados em ambientes *SDN/NFV*.

1.4 Objetivos

1.4.1 Objetivo Geral

O objetivo geral deste trabalho é analisar a eficiência e efetividade da utilização de algoritmos de aprendizagem de máquina não supervisionados em ambientes *SDN/NFV* na detecção de ataques do tipo *DDoS*, por meio de uma análise comparativa. Além disso, os algoritmos devem trabalhar com a estratégia de fluxo de dados para se adequarem ao cenário dinâmico presente no ambiente *SDN/NFV*.

1.4.2 Objetivos Específicos

Para alcançar o objetivo principal, foram definidos os seguintes objetivos específicos:

- Levantar e analisar algoritmos de aprendizagem de máquina não supervisionados que trabalhem com fluxo de dados e que possam ser utilizados na detecção de ataques *DDoS*;
- Definir uma arquitetura *SDN/NFV* e que utilize microsserviços;
- Implementar o módulo de detecção de ataques *DDoS*;
- Simular cenários de ataque *DDoS* no ambiente *SDN/NFV* proposto;
- Analisar e avaliar qualitativa e quantitativamente os algoritmos utilizando métricas como acurácia, *f1-score*, matriz confusão, tempo de treinamento e tempo de avaliação.

1.5 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos, que são:

- Capítulo 1 - Introdução: apresenta as definições preliminares da literatura, problemática, argumentações e hipóteses sobre o tema, além dos objetivos;
- Capítulo 2 - Fundamentação teórica: expõe a contextualização teórica relacionada ao tema proposto;
- Capítulo 3 - Trabalhos relacionados: demonstra os trabalhos correlatos com a revisão de literatura adotada (Mapeamento Sistemático e Revisão Sistemática), bem como são apresentados os resultados dessa revisão e a síntese do processo de sumarização dos trabalhos estudados;
- Capítulo 4 - Algoritmos de aprendizagem de máquina avaliados: apresenta os algoritmos de aprendizagem de máquina utilizados nas experimentações;
- Capítulo 5 - Metodologia: apresenta a metodologia utilizada para as experimentações, desde o *dataset* utilizado até a arquitetura desenvolvida;
- Capítulo 6 - Experimentação e análise dos resultados: exposição dos resultados obtidos na experimentação e avaliação qualitativa e quantitativa dos resultados;
- Capítulo 7 - Conclusão: conclusão do trabalho, contribuições, dificuldades e limitações, trabalhos futuros e publicações.

2
Fundamentação Teórica

Neste capítulo são abordados conceitos, arquiteturas, características e paradigmas a respeito dos temas abordados. Dessa maneira, foi realizada uma consulta a literatura específica e foi possível modelar o conhecimento inicial necessário para este trabalho no que diz respeito a ambientes *SDN/NFV*, microsserviços, ataque de negação de serviço distribuído e aprendizagem de máquina.

2.1 SDN

As redes de computadores atuais podem ser divididas em três planos com relação a suas funcionalidades: plano de dados, plano de controle e plano de gerenciamento. O plano de dados corresponde aos dispositivos de rede, como por exemplo *switches* e roteadores, os quais são responsáveis por realizar os devidos encaminhamentos de dados de maneira eficiente. Já o plano de controle é representado pelos protocolos que são utilizados pelos dispositivos de encaminhamento. Por fim, o plano de gerenciamento inclui os serviços de software, como, por exemplo, uma aplicação de gerenciamento de rede. Além disso, as politicas de rede são definidas no plano de gerenciamento, o plano de controle aplica essas políticas e o plano de dados executa o encaminhamento de dados de acordo com a política. As redes *IP* tradicionais possuem os planos de dados e de controle fortemente acoplados, incorporados no mesmo dispositivo de rede, como também, toda a estrutura é descentralizada. Essas características foram importantes para a arquitetura da Internet, pois, dessa maneira foi possível garantir a resiliência da rede. O resultado foi uma arquitetura rígida e complexa de se gerenciar e controlar. Essas duas características são amplamente responsáveis por promover uma arquitetura verticalmente integrada, onde a inovação é difícil (KREUTZ et al., 2014).

Para dar suporte ao gerenciamento de rede alguns fabricantes criaram soluções especializadas e proprietárias em hardware, sistemas operacionais e aplicações de controle. Dentre

elas, se destacam: *firewall*, *Intrusion Prevention System (IPS)*, *IDS*, programas de inspeção de pacotes, entre outros. Essas soluções são comumente chamadas de *middleboxes* e apesar de implementarem serviços e funcionalidades importantes, acabam aumentando a complexidade da rede, assim como aumentam o custo para desenvolver e manter as infraestruturas. Dessa maneira, os operadores da rede acabam necessitando de habilidades especializadas para conseguir trabalhar com essas diferentes soluções (KREUTZ et al., 2014).

As redes *SDN* são um paradigma de rede emergente. Segundo Kreutz et al. (2014), existem quatro pilares que definem uma rede *SDN*:

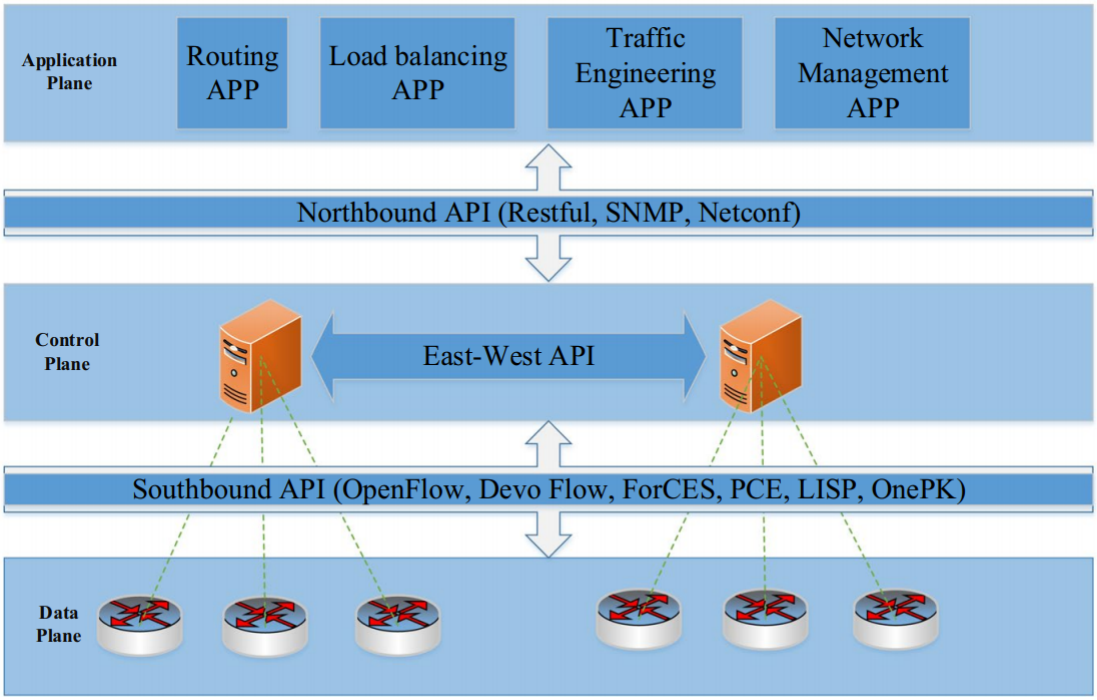
- Desacoplamento entre o plano de dados e plano de controle. O controle é retirado dos dispositivos de rede, os quais se tornam simples encaminhadores de pacotes;
- As decisões de encaminhamento são orientadas ao fluxo;
- Controle lógico da rede é movido para uma entidade externa centralizada. Essa entidade é um software executado em servidores não-especializados e é chamada de controlador *SDN* ou *Network Operating System (NOS)*;
- A rede é programável, os programas desenvolvidos executam no controlador *SDN* e se comunicam com o plano de dados.

O conceito *SDN* pode ser definido por três abstrações fundamentais: encaminhamento, distribuição e especificação. A abstração de encaminhamento pode habilitar qualquer que seja o comportamento desejado pelas aplicações da rede, ou seja, o programa de controle executa seu trabalho, enquanto esconde os detalhes do hardware subjacente. O protocolo *Openflow* é uma realização desta abstração e pode ser comparado a um *drive* de dispositivo em um sistema operacional (KREUTZ et al., 2014). Esse protocolo tem o objetivo de padronizar a comunicação entre o plano de controle e o plano de dados. Ele descreve como os aplicativos podem utilizar e modificar a tabela de fluxos de diferentes dispositivos encaminhadores (LARA; KOLASANI; RAMAMURTHY, 2013). A abstração de distribuição possui o objetivo de transformar o controle distribuído em um problema logicamente centralizado. Para isso, é necessário existir uma camada de distribuição comum, a qual reside no controlador *SDN*. Por fim, a abstração de especificação deve permitir uma aplicação de rede expressar o desejo de toda a rede sem ser responsável por implementar esse comportamento em si (KREUTZ et al., 2014).

As redes *SDN* tem o intuito de quebrar a forte integração vertical existente nas redes tradicionais. Essa ruptura divide a rede em controle lógico, controle centralizado da rede e plano de dados. No geral, a arquitetura *SDN* é divida em três camadas: aplicação, controle e plano de dados. Na camada de plano de dados residem os equipamentos de redes, os quais passam a ser somente encaminhadores. Na camada de controle reside a estratégia da rede, ou seja, as decisões e as ações do ingresso de fluxo em cada equipamento de rede. Por fim, na camada de

aplicação residem as aplicações de rede como por exemplo *firewall*, balanceamento de tráfego, etc (SHIRMARZ; GHAFFARI, 2020). Esta divisão pode ser vista na Figura 1.

Figura 1 – Arquitetura *SDN*



Fonte: Shirmarz e Ghaffari (2020, p. 3).

2.2 NFV

Para que seja possível desenvolver novos serviços e novas funcionalidades em redes tradicionais, é necessário que os administradores de rede comprem diferentes equipamentos e de diferentes fabricantes. Cada equipamento é responsável por uma parte do processamento de tráfego e cada equipamento requer estratégias de gerenciamento específicas. Dessa maneira, para esses ambientes, a gestão e a configuração sofrem dificuldades para serem gerenciados de maneira efetiva. A exigência de permitir tal flexibilidade na configuração e na implantação de novas funções de rede deve ser atendida em novos modelos de negócios e novas arquiteturas. Consequentemente há uma maior despesa com a aquisição de bens ou *Capital Expenditure (CAPEX)* e com custos operacionais ou *Operational Expenditure (OPEX)* (BONFIM; DIAS; FERNANDES, 2019).

De acordo com o *European Telecommunications Standards Institute (ETSI)*, a arquitetura *NFV* é responsável por separar as funcionalidades de rede, que antes eram disponibilizadas através de hadwares, como por exemplo, *firewall*, balanceadores de carga e *IDS* e oferecê-las através de serviços virtualizados. O *ETSI* é um órgão de normalização regional reconhecido mundialmente

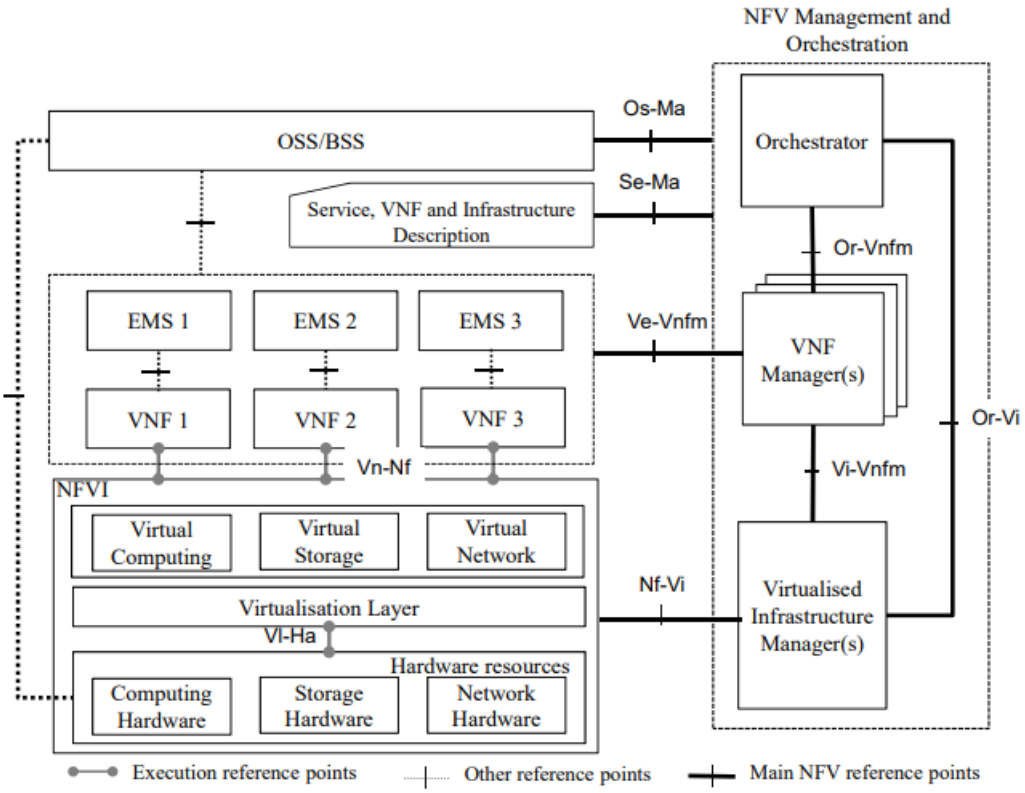
que trata de telecomunicações, radiodifusão e outras redes e serviços de comunicações eletrônicas. Esses serviços são decompostos em *VNFs* ou *Virtual Network Functions* e são executados em servidores de uso geral. Com isso, o *NFV* promove a implantação mais flexível dessas funções de rede. Nesse ambiente, os novos serviços não requerem nova infraestrutura de hardware, mas simplesmente a instalação do software e são demandados sempre que necessário (SOUSA et al., 2019).

Dentre os benefícios de se utilizar a arquitetura *NFV*, segundo Bonfim, Dias e Fernandes (2019), destacam-se:

- Flexibilidade na alocação de funções de rede em hardwares de uso geral;
- Rápida implementação e implantação de novos serviços de rede;
- Suporte a múltiplas versões do serviço e também a múltiplos cenários;
- Redução do *CAPEX* por conta de um melhor gerenciamento de gasto energético;
- Automação dos processos operacionais e melhoria da eficiência e redução do *OPEX*.

A Figura 2 ilustra a arquitetura *NFV*, a qual compreende três grandes blocos. O primeiro deles é o *Virtual Network Function (VNF)*, que é a virtualizaçao de determinada funcionalidade e que pode operar independentemente das outras funcionalidades. Em determinados casos um *VNF* pode ser divido em algumas sub-funções chamadas de *VNF Components (VNFCs)*. O segundo bloco é chamado de *NFV Infrastructure (NFVI)*, que compreende todos os hardwares e softwares necessários para implantação, operação e monitoração dos *VNFs*. Para isso, ela necessita de uma camada de virtualização para abstrair os recursos de rede necessários. Por fim, o terceiro, é o *NFV Management and Orchestration (MANO)*. Ele compreende três componentes: o *Virtualized Infrastructure Manager (VIM)* que é responsável por gerenciar e controlar a interação do *VNFs* com os recursos físicos, o *VNF Manager (VNFM)* que é responsável por gerenciar o ciclo de vida do *VNF* e o *NFV Orchestrator (NFVO)*, que é responsável pela realização de serviços de rede no NFVI, gestão global de recursos e gerenciamento de políticas (ETSI, 2014).

Figura 2 – Arquitetura *NFV*



Fonte: ETSI (2014, p. 3).

2.3 Microsserviços

No âmbito da arquitetura de software, a arquitetura baseada em microsserviços possui a ideia principal de se construir serviços pequenos, independentes e especializados na resolução de um único problema dentro de um ecossistema de uma aplicação. Ou seja, essa arquitetura tem uma abordagem de dividir um aplicativo em um conjunto de pequenos serviços independentes, e cada serviço é executado em seu próprio processo independente (NAMIOT; SNEPS-SNEPPE, 2014).

A arquitetura de microsserviços se tornou o principal estilo arquitetônico escolhido pela indústria para softwares orientados a serviços. Os serviços que os compõe são pequenos e leves e são construídos para executarem uma função de negócio coesa. Dentre os benefícios que essa arquitetura promove destacam-se: agilidade no desenvolvimento, resiliência, escalabilidade, confiabilidade, manutenibilidade, separação de obrigações e facilidade de implantação (ALSHUQAYRAN; ALI; EVANS, 2016).

Segundo Wolff (2016), os microsserviços possuem algumas características ou fatores, dentre eles:

- Tamanho do serviço. Como o próprio nome diz, os microsserviços devem ser necessariamente pequenos;
- Modularização. Times de desenvolvimento que trabalham com modularização são mais capazes de adotarem a arquitetura de microsserviços;
- Comunicação distribuída. Já que, cada microsserviço é executado em seu próprio processo;
- Arquitetura deve ser sustentável;
- Facilidade na troca de serviços. Um microsserviço deve ser fácil de ser trocado por outro sempre que possível;
- Suas transações devem possuir as características *ACID*: atomicidade, consistência, isolamento e durabilidade. A atomicidade possibilita que a transação seja executada por completo ou não seja executada. A consistência promove que os dados sejam consistentes tanto antes quanto depois da execução da transação. O isolamento propicia que uma transação não influencie na execução da outra. Por fim, a durabilidade indica a permanência dos dados daquela transação após a execução dela.

2.4 *DDoS*

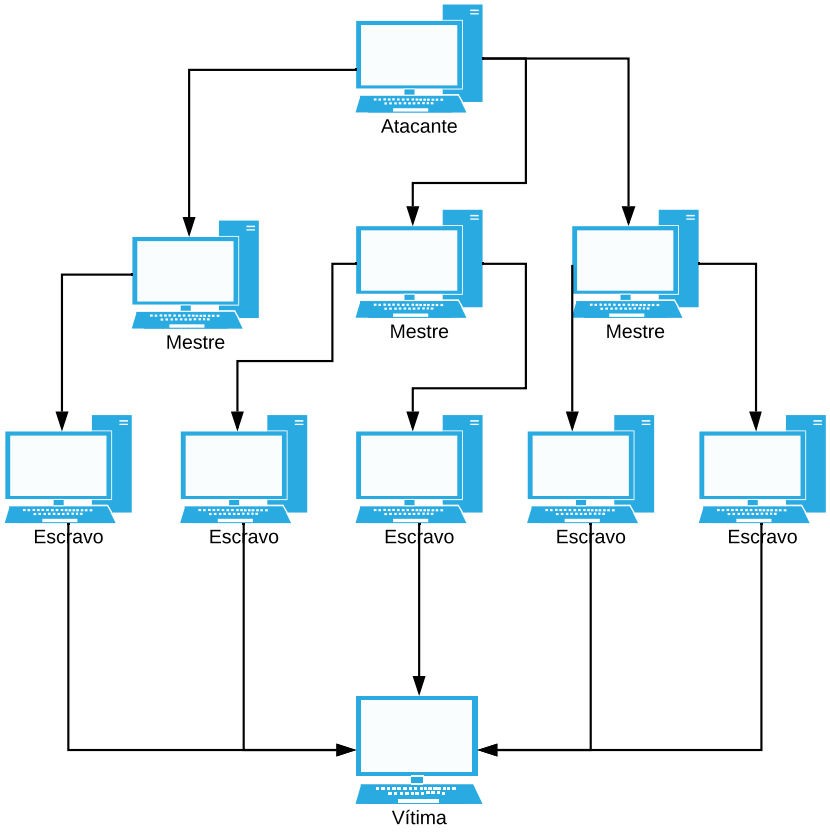
Os ataques do tipo *Denial of Service (DoS)* são definidos como casos em que um invasor impede que usuários legítimos acessem recursos, consumindo todo o serviço disponível, ou seja, a estratégia por trás desse ataque é conseguir negar o uso de serviços degradando os seus recursos e colocando a sua disponibilidade em risco. Quando o atacante utiliza várias fontes para realizar o ataque, ou seja, o ataque passa a ser distribuído, ele se torna um *DDoS* (KALKAN; GÜR; ALAGÖZ, 2016).

No geral, pacotes do ataque *DDoS* não mostram nenhuma característica óbvia que consiga diferenciar um fluxo malicioso de um fluxo legítimo. A estrutura desse ataque é simples e utiliza o conceito de muitos para um, ou seja, muitos atacante para uma só vítima. Pelo fato da estrutura da Internet ser simples, pois segue o modelo *best-effort* ou melhor esforço, a sua complexidade é baixa. O núcleo da Internet não possui autenticação para os pacotes *IPs* entregues. Além disso, os roteadores do núcleo não possuem mecanismos de rastreamento de pacotes, devido ao grande tráfego corrente. Dessa maneira, essas limitações fornecem aos invasores a oportunidade de permanecerem ocultos durante a execução do ataque *DDoS* (MAHJABIN et al., 2017).

Ataques *DDoS* podem ser motivados por benefícios econômicos e financeiros, vingança, crença ideológica, desafio intelectual e disputas ou guerras cibernéticas. Um ataque *DDoS* possui três fases e os seguintes componentes: atacante, mestres, escravos e vítima. Na primeira fase, o atacante gasta bastante tempo para infectar várias máquinas que serão os mestres. Por sua vez, os mestres controlam outras máquina dentro da estrutura do ataque, as chamadas

máquinas escravas. A criação do exército de máquinas mestres normalmente é feito de maneira automatizada utilizando um escaneador de vunerabilidades. Após isso, o código malicioso instalado pelo atacante nas máquinas mestres é utilizado para infectar mais máquinas e recrutá-las para o exército. As máquinas escravas são diretamente controladas pelas máquinas mestres e indiretamente controladas pelo atacante através das máquinas mestres, conforme pode ser visto na Figura 3 (MAHJABIN et al., 2017).

Figura 3 – Componentes do ataque *DDoS*.



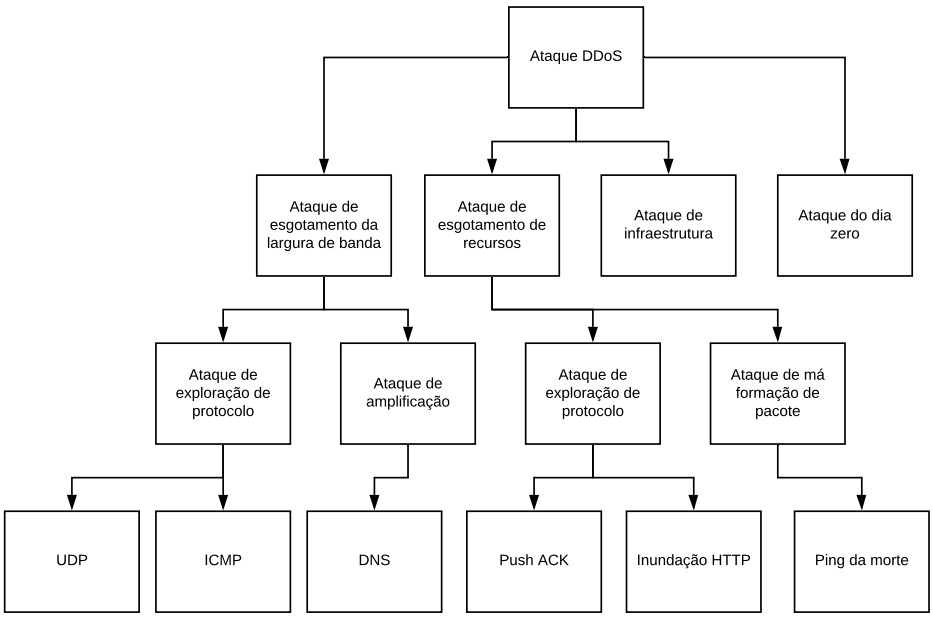
Fonte: Adaptado de Mahjabin et al. (2017, p. 8).

A segunda fase inicia com um número suficiente de máquinas infectadas compondo o exército de ataque. Esse exército se chama *botnet*. Nesta fase, o atacante transfere todas as informações necessárias, como códigos e comandos, para as máquinas mestres, que por sua vez enviam esses códigos as máquinas escravas. Na terceira fase, a fase final, o atacante comanda o exército para iniciar o ataque. O ataque ocorre de forma distribuída e com isso um grande tráfego de informações é gerado com o intuito de inundar o sistema da vítima (MAHJABIN et al., 2017).

Os ataques *DDoS* podem ser categorizados quanto aos seus mecanismos, são eles: ataques de esgotamento da largura de banda, ataques de esgotamento de recursos, ataques de infraestrutura e ataques do dia zero (MAHJABIN et al., 2017). O objetivo dos ataques de esgotamento da largura de banda é conseguir consumir toda a largura de banda da rede do sistema da vítima.

Ataques de *UDP flood* e *ICMP flood* são exemplos de ataques dessa categoria, como pode ser visto na Figura 4.

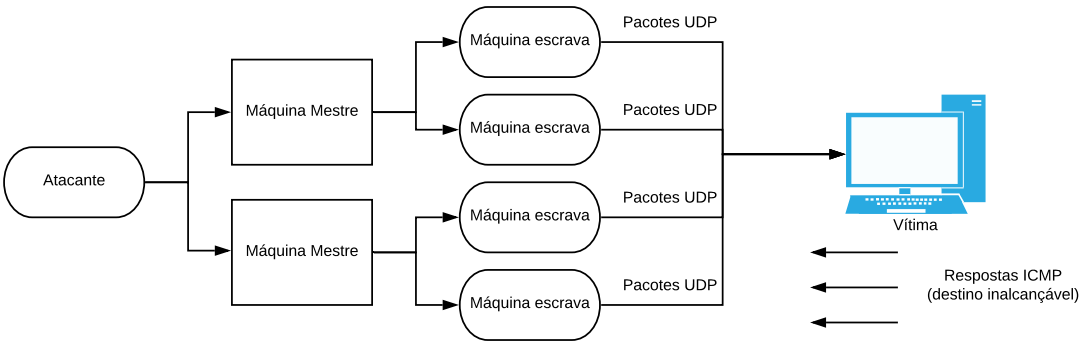
Figura 4 – Tipos de ataque *DDoS*.



Fonte: Adaptado de Mahjabin et al. (2017, p. 8).

No ataque do tipo *UDP Flood*, o atacante emite para as máquinas mestre o comando de ataque com o endereço da vítima, a duração do ataque, os métodos e outras instruções. Por sua vez, as máquinas mestres enviam as instruções para as máquinas escravas que começam a enviar pacotes *UDP* para a vítima com um *IP* falsificado no endereço de fonte. Assim que a vítima recebe esses pacotes, ela envia a confirmação para o endereço *IP* de origem. Porém, não recebe resposta alguma, já que o *IP* de origem é falsificado. Isso é feito por um número de máquinas escravas grande e por isso acontece uma inundação do sistema e toda a largura de banda é consumida, causando a falha do sistema (SINGH; JUNEJA, 2010). A Figura 5 ilustra como esse mecanismo funciona.

Figura 5 – Mecanismo de ataque *UDP flood*.



Fonte: Adaptado de Singh e Juneja (2010, p. 8).

O objetivo dos ataques de esgotamento de recursos é inundar a fim de conseguir travar algum recurso do sistema, como por exemplo memória e *CPU*. O ataque *TCP SYN* se aproveita das limitações do protocolo *TCP* para manter o máximo de conexões semi-abertas possível a fim de consumir os recursos de computação disponíveis na máquina da vítima (KUMAR; PETANA, 2008).

Dos ataques *DDoS*, o ataque à infraestrutura pode ser considerado o tipo mais catastrófico de ataque. O objetivo dele é danificar elementos cruciais da rede. Assim, não apenas segmenta a largura de banda da rede, mas também consome os recursos, como memória e *CPU*, do sistema da vítima. Um exemplo deste ataque é o ataque de infraestrutura ao *DNS (Domain Name System)*, especialmente os *DNSs* raízes. Este ataque é batante perigoso, pois, são os principais pontos do serviço de nomes hierárquico e eles fornecem serviços a todos os usuários da Internet (MAHJABIN et al., 2017).

O ataque do dia zero é um ataque cibernético que explora uma vulnerabilidade que ainda não foi divulgada publicamente. Por isso, quase não há defesa contra um ataque do tipo dia zero e enquanto a vulnerabilidade permanece desconhecida, o software afetado não pode ser corrigido (BILGE; DUMITRAŞ, 2012).

2.5 Aprendizagem de máquina

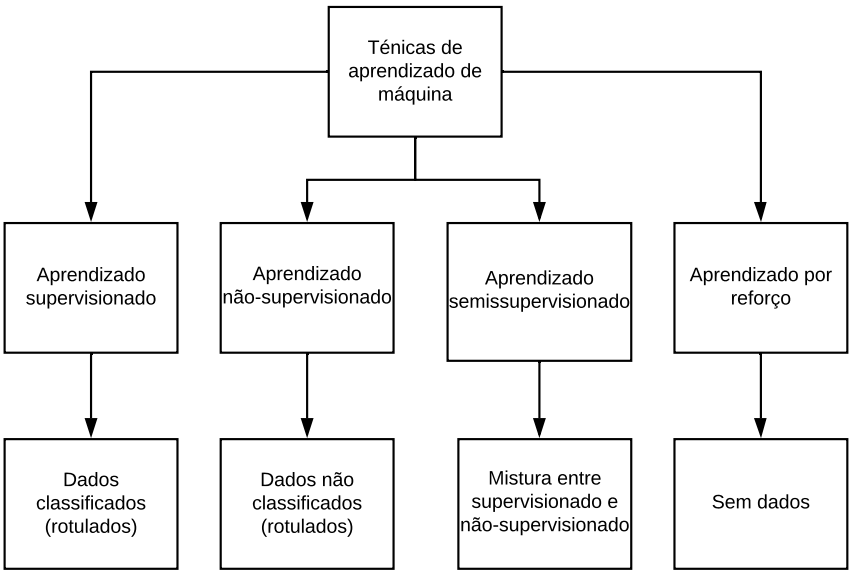
A aprendizagem de máquina é um ramo da inteligência artificial que tem como objetivo permitir que máquinas executem seus trabalhos com habilidade utilizando software inteligente. Os métodos estatísticos de aprendizagem constituem a espinha dorsal dos softwares inteligentes utilizados para desenvolver a inteligência nas máquinas (MOHAMMED; KHAN; BASHIER, 2016).

O conceito de aprendizagem de máquina é uma consequência natural da interseção de Ciência da Computação e Estatística. A Ciência da Computação se concentrou principalmente

em como programar de fato os computadores e a aprendizagem de máquina concentra-se na questão de como fazer com que os computadores se auto programem utilizando alguma estrutura inicial (MITCHELL, 2006).

Existem diferentes técnicas de aprendizagem de máquina, conforme pode ser visto na Figura 6. Na técnica de aprendizagem supervisionada, o objetivo é inferir uma função ou mapeamento para que seja possível predizer certo dado a partir de dados de treinamento rotulados previamente. Essa técnica utiliza supervisores e frequentemente esses supervisores são humanos, mas também podem ser utilizadas outras máquinas. Assim, se determinado dado foi rotulado de maneira errada, o supervisor pode modificar aquele rótulo a fim de melhorar o modelo. Já na técnica de aprendizado não supervisionado, não existe a figura do supervisor, como também, não existem dados rotulados previamente para o treinamento. A ideia desta técnica é encontrar uma estrutura oculta nesses dados (MOHAMMED; KHAN; BASHIER, 2016).

Figura 6 – Técnicas de aprendizagem de máquina.



Fonte: Mohammed, Khan e Bashier (2016, p. 7).

Os algoritmos *k-means, fuzzy c-means e agglomerative clustering* são exemplos de algoritmos de aprendizagem de máquina não supervisionados. Segundo Ghosh e Dubey (2013) o algoritmo *k-means* ou *hard c-means* é um método de particionamento aplicado para analisar e tratar observações dos dados como objetos com base em localizações e distância entre vários pontos de dados de entrada. Particionando os objetos em grupos mutuamente exclusivos (k), o algoritmo posiciona os objetos dentro de cada grupo de maneira que eles permaneçam tão próximos quanto possível uns dos outros, mas, o mais longe possível dos objetos em outros grupos. Cada grupo é caracterizado por seu ponto central, ou seja, centróide. As distâncias utilizadas no agrupamento na maioria das vezes não representam as distâncias espaciais. Em

um conjunto de dados, um número desejado de grupos K e um conjunto de k pontos de partida iniciais, o algoritmo encontra o número desejado de distintos grupos e seus centróides, que é o ponto cujas coordenadas são obtidas por meio do cálculo da média de cada uma das coordenadas dos pontos de amostras atribuídos aos grupos.

Por sua vez, o algoritmo *fuzzy c-means*, introduzido por Bezdek, Ehrlich e Full (1984), foi uma extensão do método de agrupamento *hard c-mean*. Ele é um algoritmo de agrupamento não supervisionado que é aplicado a uma ampla gama de problemas relacionados com a análise de recursos e agrupamento. Ele é amplamente aplicado na agricultura, engenharia, astronomia, química, geologia, análise de imagens, diagnóstico médico, análise de forma e reconhecimento de alvo. Este algoritmo é baseado na teoria *Ruspini Fuzzy* (RUSPINI, 1970) de agrupamento e é utilizado para análise com base na distância entre vários dados de entrada pontos. Os grupos são formados de acordo com a distância entre os pontos de dados e os centros do grupos formados.

Já o algoritmo *agglomerative clustering*, é da família de algoritmos de agrupamento hirárquico, os quais constroem grupos aninhados mesclando pontos de dados de maneira sucessiva. A hierarquia dos agrupamentos pode ser representada como um diagrama de árvore conhecido como dendrograma. Dessa maneira, a parte superior da árvore é um único grupo com todos os pontos de dados, enquanto a parte inferior contém pontos individuais. Este algoritmo executa um agrupamento hierárquico utilizando uma abordagem ascendente, ou seja, cada observação começa em seu próprio grupo, e os grupos são sucessivamente mesclados. O critério de ligação determina a métrica usada para a estratégia de fusão de grupos, dentre eles: *ward*, que minimiza a soma das diferenças quadradas em todos os grupos; a ligação máxima ou completa, que minimiza a distância máxima entre as observações de pares de agrupamentos; a ligação média, que minimiza a média das distâncias entre todas as observações de pares de grupos e a ligação única, que minimiza a distância entre as observações mais próximas de pares de grupos (KAUSHIK; MATHUR, 2014).

No cenário de detecção de ataques em ambientes de rede, pelo fato da quantidade grande de dados e novas técnicas de ataque surgirem a todo momento, o aprendizado não supervisionado pode ser mais interessante de ser utilizado. Porém, nesse cenário existe o conceito de fluxos de dados ou *data streams*, que são grandes sequências e ilimitadas de objetos de dados. Eles são gerados continuamente e a taxas rápidas. Esses fluxos também podem ser vistos em análises meteorológicas e redes de sensores, por exemplo. Dessa maneira, os algoritmos tradicionais de aprendizado não supervisionado se tornam restritos por trabalharem com um número finito de categorias enquanto os que trabalham com fluxos de dados precisam continuamente categorizar os dados (SILVA et al., 2013).

Os fluxos de dados são conjuntos de dados que são muito grandes para estarem na memória principal e por isso, geralmente são armazenados em um dispositivo de armazenamento secundário. Desse ponto de vista, e levando em consideração que o acesso aleatório é bastante custoso (GUHA et al., 2003), realizar varreduras lineares dos dados é o único método de acesso

aceitável em termos de eficiência computacional. A extração de conhecimento potencialmente útil dos fluxos de dados é um desafio. Em um ambiente que trabalhe com fluxos de dados, os algoritmos utilizados devem levar em consideração as seguintes restrições e condições, segundo Silva et al. (2013):

- Os fluxos de dados chegam continuamente;
- Não há controle sobre a ordem na qual os objetos de dados devem ser processados;
- O tamanho de um fluxo é potencialmente ilimitado;
- Os objetos de dados são descartados após serem processados. Na prática, é possível armazenar parte dos dados por um determinado período de tempo, utilizando um mecanismo para descartá-los mais tarde;
- O processo de geração de dados é possivelmente não-estacionário, ou seja, sua distribuição de probabilidades pode mudar com o tempo.

O trabalho de Silva et al. (2013) estudou algoritmos não supervisionados que trabalham com fluxo de dados. A fim de realizar uma análise entre eles, o estudo pontuou algumas métricas de análise dos algoritmos, dentre elas: estrutura de dados, algoritmo de *clustering* e forma do *cluster*. A Tabela 1 relaciona os algoritmos e as algumas dessas métricas.

Tabela 1 – Análise dos algoritmos.

Algoritmo	**Estrutura de dados**	**Algoritmo de *clustering***	**Forma do cluster**
BIRCH	*feature vector*	*k-means*	hiperesfera
CluStream	*feature vector*	*k-means*	hiperesfera
ClusTree	*feature vector*	*k-means/DBSCAN*	arbitrário
D-Stream	*grid*	*DBSCAN*	arbitrário
DenStream	matriz correlacional	*DBSCAN*	arbitrário
DGClus	*feature vector*	*hierarchical clustering*	hiperesfera
OADC	*feature vector*	*k-means*	hiperelipse
Scalable k-means	*feature vector*	*k-means*	hiperesfera
Single-pass k-means	*feature vector*	*k-means*	hiperesfera
Stream	array prototipado	*k-median*	hiperesfera
Stream LSearch	array prototipado	*k-median*	hiperesfera
StreamKM++	conjuto básico de árvore	*k-means*	hiperesfera
SWClustering	array prototipado	*k-means*	hiperesfera

Fonte: Silva et al. (2013).

3

Trabalhos Relacionados

Neste capítulo, identificamos e discutimos trabalhos de pesquisa que abordaram a detecção de ataques *DDoS* em ambientes *SDN/NFV*. Primeiramente, foi realizado um Mapeamento Sistemático da Literatura. Este mapeamento foi necessário para embasar o desenvolvimento de um primeiro experimento, descrito no Capítulo 6 e que buscou estudar algoritmos que não trabalham com a estratégia de fluxo de dados. Assim, como critério de seleção foram incluídos os trabalhos que utilizassem a aprendizagem de máquina como técnica de detecção e mais precisamente que eram baseados no algoritmo *k-means* em ambientes *SDN/NFV*. Após isso, foi realizada uma Revisão Sistemática da Literatura e como critério de seleção foram incluídos os trabalhos que utilizassem aprendizagem de máquina não supervisionada na detecção de ataques *DDoS* e que trabalhassem com a estratégia de fluxo de dados.

3.1 Levantamento Bibliográfico

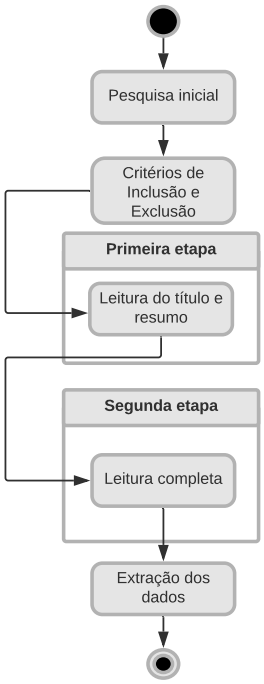
Segundo Keele et al. (2007), um estudo de mapeamento sistemático permite que as evidências num certo domínio sejam plotadas num alto nível de granularidade. Isso permite a identificação de grupos de evidências e a falta de evidências para direcionar o foco de revisões sistemáticas futuras e para identificar áreas onde mais estudos primários precisam ser conduzidos. Além disso, o objetivo principal do mapeamento sistemático é prover uma visão geral de uma área de pesquisa, identificar a quantidade, o tipo e os resultados das pesquisas disponíveis, ou seja, conhecer as freqüências de publicações ao longo do tempo a fim de identificar tendências (PETERSEN et al., 2008). Por esses motivos, o Mapeamento Sistemático da Literatura foi realizado inicialmente, além de servir como base para o desenvolvimento do primeiro experimento que tinha enfoque nos algoritmos que não trabalham com a estratégia de fluxo de dados.

Após realizado o mapeamento e o primeiro experimento, a pesquisa foi melhor definida

e com isso, foi realizado outro levantamento bibliográfico. Ele se deu através de uma Revisão Sistemática da Literatura, a fim de manter um alto grau de rigor científico. Uma Revisão Sistemática da Literatura é um meio de identificar, avaliar e interpretar toda a pesquisa disponível e relevante para determinado tema ou questão de pesquisa específica (KITCHENHAM, 2004).

Para a realização do levantamento bibliográfico utilizou-se as bases de dados *Springer Link* [1], *ACM Digital Library* [2], *IEEE Digital Library* [3], *ISI Web Of Science* [4], *Science Direct* [5] e *Scopus* [6]. Seguiu-se um protocolo rígido e sistematizado dividido em três etapas: execução da busca, primeira etapa de seleção e segunda etapa de seleção. A fim de demonstrar como serão realizadas as etapas da pesquisa, foi elaborado um diagrama para melhor entendimento e visualização da estratégia de seleção dos estudos, conforme pode ser visto na Figura 7.

Figura 7 – Estratégia de seleção de estudos.



Fonte: Próprio autor.

Para a etapa de execução da busca, são realizados dois refinamentos na *string* de busca, com o objetivo de obter resultados mais significativos. A *string* de busca é então utilizada nas fontes selecionadas e os resultados obtidos armazenados na ferramenta online *Parsifal* [7]. Após isso, foi analisado e aplicado aos artigos retornados na busca os critérios de inclusão e exclusão.

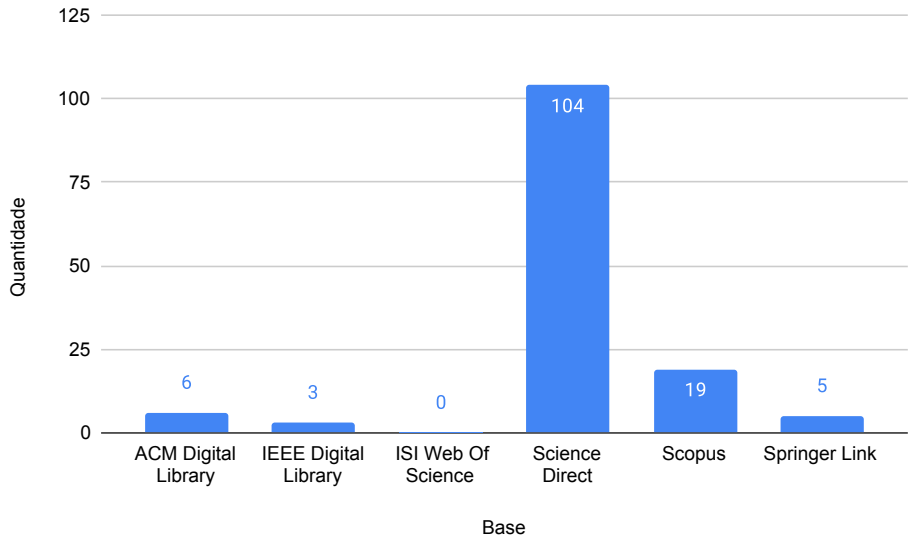
1 <https://link.springer.com/>
2 <https://dl.acm.org/>
3 <https://ieeexplore.ieee.org/Xplore/home.jsp>
4 <https://webofknowledge.com>
5 <https://www.sciencedirect.com/>
6 <https://www.scopus.com/>
7 <https://parsif.al/about/>

Foram definidos como critérios de exclusão: artigos duplicados, artigos que não estão escritos na língua inglesa, estudos não primários, artigos publicados apenas como resumos ou prefácio de periódicos e eventos, falta de disponibilidade do artigo para download, teses, dissertações e monografias de conclusão de curso e artigos fora da área de Ciências da Computação. Logo após, os artigos selecionados passarão para a primeira etapa, a qual é feita uma leitura dos títulos e dos resumos de cada estudo. Após isso, os estudos selecionados passam para a segunda etapa, a qual é realizada a leitura completa dos artigos e a análise dos resultados.

3.2 Mapeamento Sistemático

Para o mapeamento sistemático foi utilizada a seguinte *string*de busca: *(("detection") OR ("detecting") OR ("detect")) AND (("ddos") OR ("distributed denial of service")) AND (("sdn") OR ("software defined network")) AND (("nfv") OR ("network functions virtualization") OR ("networking functions virtualization")) AND (("kmeans") OR ("k-means"))*. A coleta de dados aconteceu no dia 06 de maio de 2020 e não foi definido nenhum limite temporal. Dessa forma, a pesquisa retornou um total de 137 artigos, como pode ser visto na Figura 8, sendo 6 artigos na base *ACM Digital Library*, 3 artigos na base *IEEE Digital Library*, 104 artigos na base *Science Direct*, 19 artigos na base *Scopus*, 5 artigos na base *Springer Link* e não foi retornado nenhum artigo na base *ISI Web Of Science*.

Figura 8 – Artigos levantados no mapeamento.

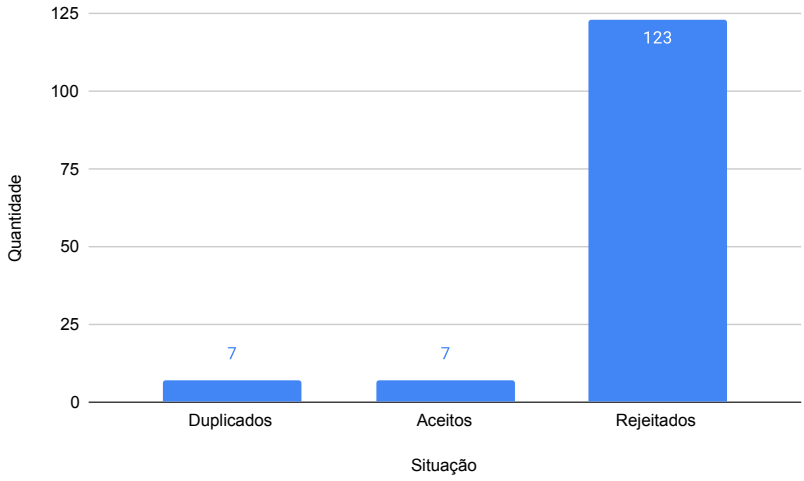


Fonte: Próprio autor.

Primeiramente, foram aplicados os critérios de inclusão e exclusão aos artigos retornados na busca, essa foi a primeira etapa de seleção. Nessa etapa foram excluídos 7 artigos duplicados e

45 artigos que se encaixaram nos outros critérios de exclusão, resultando num total de 52 artigos excluídos, restando 85 artigos. Logo após, os artigos selecionados passaram para a primeira etapa de seleção, nela foi feita uma leitura dos títulos e dos resumos de cada estudo, a fim de detectar artigos fora do escopo da pesquisa. Nesta etapa foram excluídos 14 artigos, restando um total de 71 artigos. Após isso, os estudos selecionados passaram para a segunda etapa, a qual foi realizada a leitura completa dos artigos, nesta etapa foram excluídos 64 artigos, restando 7 artigos, os quais atendem os critérios formalizados no protocolo desta Revisão Sistemática. Do resultado inicial contendo 137 artigos, 7 foram aceitos e 130 foram excluídos, dentre os quais 7 estavam duplicados, conforme pode ser visto na Figura 9.

Figura 9 – Resultado da segunda etapa de seleção do mapeamento sistemático.



Fonte: Próprio autor.

3.2.1 Trabalhos de Pesquisas Relacionados

O projeto de Wei et al. (2018) propõe um sistema de defesa contra ataques *DDoS* chamado *SDN-Oriented Two-Stage DDoS Detection and Defence System Based on Clustering (STDC)*. Ele possui dois estágios, no primeiro o sistema extrai as *features* do fluxo tanto para detectar o ataque quanto para obter a lista de *IPs* atacantes. No segundo estágio, o fluxo passa por regras de filtragem do tráfego de ataque. Ou seja, além de detectar de fato a ocorrência do ataque, este sistema também pode realizar a mitigação do impacto do ataque.

O estudo de Aqil et al. (2017) propôs o *Jaal*, que é um sistema de detecção de intrusão que trabalha diretamente no *Internet Service Provider (ISP)*, ou seja, nos provedores de Internet. O objetivo deste estudo foi desenvolver uma arquitetura que possuísse as seguintes características: pudesse detectar uma grande variedade de ataques e que não fosse necessário copiar e transferir uma quantidade grande de pacotes para uma central analisadora de pacotes. Monitores ficam

espalhados na rede e sumarizam os pacotes, diminuindo drasticamente o tamanho deles. Esses pacotes são enviados a central analisadora que os analisa e ativa os alertas de ataque.

O artigo desenvolvido por Vidal, Orozco e Villalba (2018) propõe uma estratégia de detecção e mitigação de ataques *DDoS* do tipo inundação. A implantação de sensores na rede, os quais se integram em um sistema imunológico artificial ou *Artificial Immune System (AIS)* é principal característica deste trabalho. Esse sistema, o *AIS*, é inspirado em mecanismos de defesa biológica dos seres humanos e diferentemente de propostas de trabalhos semelhantes, os métodos convencionais de inspiração biológica para reconhecimento de padrões não foram aplicados. Ao invés disto, há uma combinação de estratégias, o estudo de variações de entropia e uma adaptação da estratégia de reações imunológicas da biologia para detecção de *DDoS*. Com isso, torna-se possível aplicar contramedidas em tempo real, construindo uma memória imune e o estabelecimento de áreas de quarentena.

Koning et al. (2019) propõem um framework chamado *Secure Autonomous Response Network (SARNET)*, que é um componente autonômico de defesa. Ele foi desenvolvido para existir e trabalhar dentro de um ambiente *VNET*, o qual foi desenvolvido previamente no trabalho Koning et al. (2016). Para realizar esta defesa autônoma, o *framework* possui um agente que recebe dados em tempo real e monitora os dados e estados observáveis da *VNET*. Ele consegue instruir a *VNET* a alterar a infraestrutura de rede virtual quando uma ação é necessária. Por sua vez, a *VNET* fornece ao agente SARNET as informações e as ferramentas necessárias para a defesa autônoma da rede.

Suomalainen et al. (2018) propõem uma estrutura flexível e escalável para monitoramento de segurança, aprendizado de máquina, medição e controle de segurança em redes *5G*. A estrutura permite que diferentes partes avaliem a confiabilidade de uma rede móvel, dentre essas partes destacam-se: provedores de infraestrutura, usuários finais, aplicativos prestadores de serviços terceirizados e organizações de clientes. Além disso, foi proposto um mecanismo de gerenciamento de acordo de nível de serviço a fim de compartilhar informação e conscientizar a respeito da segurança em tempo real em cenários de vários domínios e vários operadores. Ademais, descreveram abordagens para lidar com duas ameaças do *5G*: o rastreamento da localização e ataques de inundação de autenticação causadas por *botnets IoT*.

A pesquisa de Li, Zhao e Li (2017) desenvolve um sistema de detecção de intrusão baseado na arquitetura *5G*. A solução proposta se beneficia da tecnologia *SDN* e *NFV*, integrando módulos de segurança em uma plataforma unificada e dinamicamente invocada, sob gerenciamento e controle centralizado. Além disso, aplica aprendizado de máquina em uma grande quantidade de dados e detecta ataques desconhecidos com base na classificação de fluxo. Foi utilizado o algoritmo *Random Florest* para seleção de *features* e combinado a ele o algoritmo *k-means++* com recursos adaptáveis, *AdaBoost*, para classificação de fluxo. O sistema proposto aumenta a força da proteção de segurança para futuras redes *5G*.

O projeto de Attak et al. (2017) propõe uma solução chamada *SHIELD* que visa oferecer

segurança de TI como um serviço integral de infraestrutura de redes virtuais. Esses serviços podem existir tanto nos provedores, os *ISPs*, quanto em empresas. Os *vNSFs*, ou *Virtual Network Security Functions*, fornecem instâncias do software de segurança, os quais podem ser implantados dinamicamente na infraestrutura de rede desejada. A arquitetura da solução proposta conta com o monitoramento de dados e logs através dos *vNSFs* e a agregação dessas informações coletadas para serem processadas posteriormente por engines intermediárias. Além disso, há possibilidade de adição de novos *vNSFs* para coleta adicional de dados ou combate a ataques, divulgação das informações de forma visual e recomendação de ações necessárias, assim como a possibilidade de adição de novas funcionalidades para segurança e reconfiguração dos controles atuais do sistema.

3.2.2 Análise dos Trabalhos de Pesquisas Relacionados

O projeto de Wei et al. (2018) possui dois estágios, o primeiro, utiliza as *features* do fluxo para detectar o ataque *DDoS* e obter os *IPs* atacantes. Após isso, regras de filtragem são enviadas ao controlador, a fim de mitigar o ataque, não só detectá-lo.

A arquitetura proposta por Aqil et al. (2017) tem como objetivo de estudo as redes em malha, para isso foi projetado monitores, os quais ficam espalhados na rede capturando os pacotes que serão posteriormente classificados. Os pacotes que serão analisados passam por um processo de sumarização e normalização, depois disso os sumários dos pacotes são agregados para que assim possam ser analisados quanto a ocorrência ou não de ataques. Essa solução utiliza utiliza o *Snort* e um *Network Intrusion Detection System (NIDS)* de uso popular para fazer inferências.

A solução proposta por Vidal, Orozco e Villalba (2018) se inspira no sistema imunológico humano, assim como descrito anteriormente. Por isso, nessa arquitetura não existe um componente off-line para realizar a classificação do fluxo, ao invés disso, ele trabalha de maneira on-line. Para isso, existem alguns atores que coexistem dentro segmento de rede que se deseja proteger. Um dos atores é o Detector H, eles são envolvidos na resposta imune inata e na resposta adaptativa. Consequentemente, eles são capazes de reconhecer e bloquear novos ataques, além de participarem da construção da memória imunológica. Outro ator é o Detector A, que são agentes que tem a capacidade de detectar e mitigar os ataques previamente identificados pelo DH assumindo um papel muito importante a resposta adaptativa.

O trabalho de Koning et al. (2019) propõem uma solução completa baseada em um loop de controle que detecta, analisa, decide, responde e aprende. Os autores destacam que o aprendizado de máquina pode ser utilizado na detecção de ataques, porém, foram utilizadas técnicas supervisionadas de aprendizado de máquina. Além disso, não fica claro quais técnicas foram utilizadas. A pesquisa de Attak et al. (2017) também utilizou técnicas supervisionadas de aprendizado de máquina, dentre elas: *Naive Bayes* e *SVM*.

A proposta de Suomalainen et al. (2018) possui como objetivo o estudo do ambiente *5G*

e além disso, trabalha num contexto multidomínio. Além da figura do controlador *SDN*, que já possui a característica de ser um elemento centralizador, existe também a figura do *Event Broker*. Ele é componente que trabalha com a arquitetura de publicação de subscrição e encaminha as informações aos componentes que se inscreveram nesse fluxo de informações em particular. Essa abordagem aumenta a escalabilidade e a flexibilidade da segurança *5G*. Além disso, na análise de dados, após a coleta e a sua normalização, foram utilizados os algoritmos *k-means* e *streaming-k-means*, que é uma modificação do algoritmo *k-means* para trabalhar com fluxo de dados.

A pesquisa de Li, Zhao e Li (2017) foi desenvolvida para o ambiente *5G* e suas peculiaridades. A estratégia de detecção de fluxo malicioso utilizada na arquitetura mostrou resultados bons no tocante a acurácia e menor *overhead*. Porém, o tempo de processamento para um número crescente de fluxos foi alto, como pode ser visto em uma comparação feita pelo próprio, o que não é desejado e pode ocasionar um gargalo de processamento em um ambiente grande.

Os trabalhos revistos foram sumarizados e podem ser visualizados na Tabela 2. Diferente dos trabalhos apresentados, o primeiro experimento (Capítulo 6) pretende avaliar algoritmos de aprendizagem de máquina não supervisionados sendo utilizados na detecção de ataques *DDoS* em ambientes *SDN/NFV* e que não utilizam a estratégia de fluxo de dados.

Tabela 2 – Comparação entre os trabalhos relacionados do mapeamento sistemático.

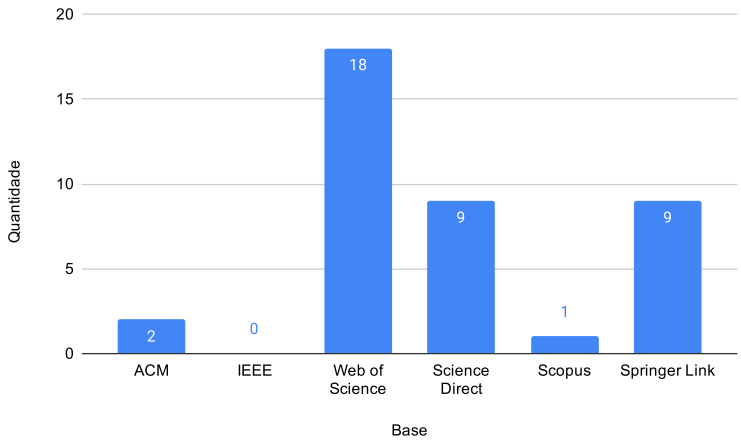
Autor (Ano)	***SDN***	***NFV***	***5G***	***Supervisionado***	***Não-supervisionado***	***Imuno-inspirado***	**Algoritmo**	**Fluxo de dados**
(AQIL et al., 2017)	X	X			X		*k-means++*	
(WEI et al., 2018)	X	X			X		*k-means*	
(VIDAL; OROZCO; VILLALBA, 2018)	X	X			X	X	*k-means++*	
(KONING et al., 2019)	X	X		X			não deixa claro	
(SUOMALAINEN et al., 2018)	X	X	X		X		*k-means* e *streaming-k-means*	X
(LI; ZHAO; LI, 2017)	X	X	X		X		*k-means++*	
(ATTAK et al., 2017)	X	X		X			*Naive Bayes* e *SVM*	

3.3 Revisão Sistemática

Para a Revisão Sistemática, foi utilizada a *string* de busca *("data stream clustering"OR "online clustering") AND ("ddos"OR "distributed denial of service") AND ("detect"OR "detec-*

ting"OR "detection"). A coleta de dados aconteceu no dia 23 de janeiro de 2021, não foi definido nenhum limite temporal. Dessa forma, a pesquisa retornou um total de 39 artigos, como pode ser visto na Figura 10, sendo 2 artigos na base *ACM Digital Library*, 0 artigos na base *IEEE Digital Library*, 18 artigos na base *ISI Web Of Science*, 9 artigos na base *Science Direct*, 1 artigo na base *Scopus* e 9 artigos na base *Springer Link*.

Figura 10 – Artigos levantados revisão sistemática.



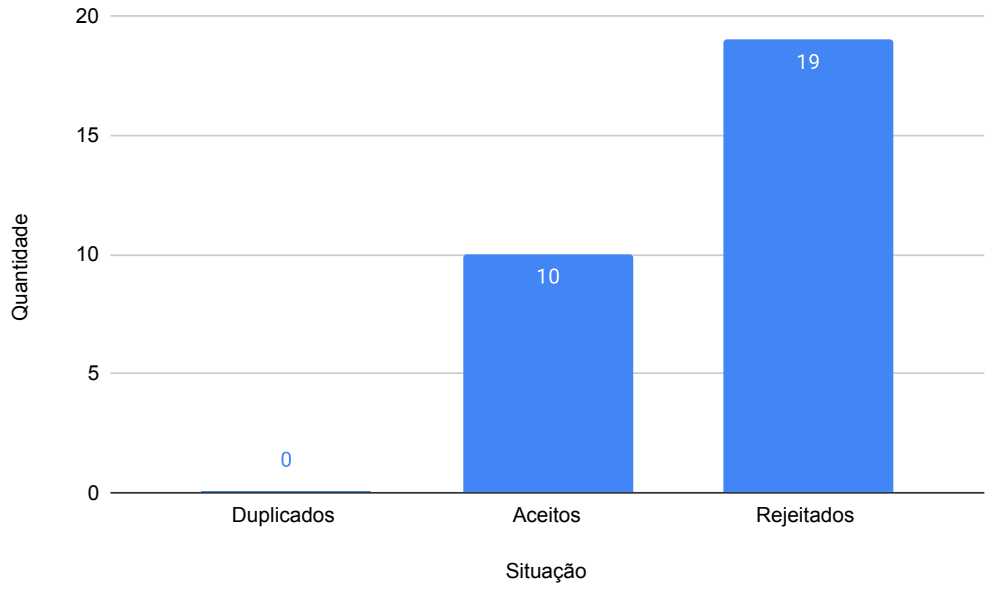
Fonte: Próprio autor.

Na primeira etapa de seleção foram aplicado os critérios de inclusão e exclusão aos artigos retornados na busca. Nessa etapa 10 artigos que se encaixaram nos outros critérios de exclusão e não houve nenhum artigo duplicado, resultando num total de 10 artigos excluídos, restando 29 artigos. Logo após, os artigos selecionados passaram para a primeira etapa de seleção, nela foi feita uma leitura dos títulos e dos resumos de cada estudo, a fim de detectar artigos fora do escopo da pesquisa. Nesta etapa foram excluídos 9 artigos, restando um total de 20 artigos. Após isso, os estudos selecionados passaram para a segunda etapa, a qual foi realizada a leitura completa dos artigos, nesta etapa foram excluídos 10 artigos, restando 10 artigos, os quais atendem os critérios formalizados no protocolo desta Revisão Sistemática. Do resultado inicial contendo 39 artigos, 10 foram aceitos e 29 foram excluídos, dentre os quais nenhum estava duplicado, conforme pode ser visto na Figura 11.

3.3.1 Trabalhos de Pesquisas Relacionados

A arquitetura proposta por Saravanan (2017) apresenta um modelo de detecção de ataques *DDoS* de rede baseado em sistemas *neuro-fuzzy* para formação dos agrupamentos. Esse tipo de sistema consegue detectar eficientemente pacotes sem que haja interrupção na rede. A detecção é realizada por meio de uma técnica que utiliza estatística sobre o fluxo dados de tráfego da rede.

Figura 11 – Resultado da segunda etapa de seleção da revisão sistemática.



Fonte: Próprio autor.

Dessa maneira, há uma melhoria na detecção de anomalias e na agregação de alertas de ataque por meio dos dados do tráfego da rede.

O artigo desenvolvido por Škrjanc et al. (2017) apresenta uma nova abordagem de solução baseada em técnicas de aprendizado de máquina não supervisionadas para o monitoramento da *darknet* utilizando algoritmos de agrupamento on-line. Esta abordagem utiliza sistemas não supervisionados em evolução para detecção dos ataques cibernéticos. Além disso, utilizam a distância de Mahalonobis de maneira generalizada para obter uma forma mais compacta de *clusters*.

O estudo desenvolvido por Patil, Krishna e Kumar (2020) propôs uma nova estrutura de detecção de ataques *DDoS* de maneira distribuída e em tempo real, baseada na plataforma *Spark* [8] e chamada *S-DDoS*. O *Spark* é uma *API* de código aberto desenvolvida pela *Apache* [9] e responsável por realizar processamento distribuído, utilizada para analisar grandes volumes de dados em tempo real. Além disso, utilizam também *Apache Flume* [10], que é um serviço distribuído, confiável e disponível para coletar, agregar e mover com eficiência grandes quantidades de dados.

Zolotukhin et al. (2016) propõem melhorar a técnica desenvolvida em um trabalho anterior para detecção de ataques *DDoS* baseada em anomalias. Essa técnica é focada na camada de aplicação e em protocolos que utilizam criptografia. Ela extrai padrões de comportamento

[8] https://spark.apache.org/
[9] https://www.apache.org/
[10] https://flume.apache.org/

normais e detecta amostras que se desviam significativamente desses padrões. A melhoria proposta permite atualizar o modelo de comportamento normal do usuário sempre que um novo fluxo do tráfego da rede estiver disponível para análise.

O projeto de Zhou et al. (2018) propõe um sistema on-line de monitoramento de tráfego da Internet para detectar ataques *DDoS* baseado em aprendizado de máquina e na plataforma de *big data Spark*. A solução consegue processar uma enorme quantidade dados de tráfego de forma eficiente, a fim de monitorar o status da rede em tempo real. Mesmo se houver alguma falha, o sistema não para seu processamento, ou seja, mesmo em situações de erro o sistema não aborta todo o processo de monitoramento.

A pesquisa desenvolvida por Zolotukhin e Hämäläinen (2018) propõe uma solução de detecção de ataque *DDoS* que reside na camada de aplicação *TCP/IP*. A arquitetura proposta pode ser aplicada e utilizada juntamente com protocolos seguros que criptografam os dados das conexões de rede, mas sem necessidade de descriptografá-los. Nesta solução foram utilizados algoritmos de agrupamento de fluxo de dados, os quais foram escolhidos para este cenário por conseguirem permitir a atualização contínua do modelo dentro das restrições de memória e tempo.

Ivannikova, Zolotukhin e Hämäläinen (2017) propõem uma técnica de detecção que extrai padrões normais de comportamento do usuário e detecta anomalias. Além disso, aprimoraram o algoritmo de detecção proposto em trabalhos anteriores. Dessa maneira, a arquitetura desenvolvida fornece uma solução para detectar ataques *DDoS* na camada de aplicação *TCP/IP* em ambientes de nuvem baseados em *SDN*.

O estudo desenvolvido por Campbell, Kiran e Wala (2020) explora o uso de algoritmos não supervisionados na classificação informações de tráfego de rede. Dessa maneira, o objetivo deste estudo é resolver os problemas de segurança em redes *WAN* utilizando aprendizado de máquina não supervisionado. Para isso, foi utilizado como base o algoritmo *k-means* e o método gaussiano. Para isso, foram desenvolvidos novos métodos para reconhecer anomalias, estimando a distância dos pontos para *cluster* e estudando as informações de densidade.

Ring et al. (2017) propõem um novo conjunto de ferramentas chamado *CUF (Coburg Utility Framework)*. Este conjunto é baseado em uma arquitetura flexível e oferece uma ampla gama de algoritmos de mineração de dados. Este trabalho aborda vários aspectos que podem contribuir para vencer o desafio de identificar incidentes significativos ambientes que trabalham com fluxos de dados, como por exemplo as rede de computadores.

A pesquisa de Guerrieri e Montresor (2012) desenvolveu o algoritmo *DS-means (Distributed Streams Means)*, que é uma combinação de várias técnicas conhecidas para resolver problemas de aprendizagem de máquina do tipo agrupamento que utilizam a estratégia de fluxo de dados. O algoritmo utiliza uma versão distribuída do algoritmo *k-means*

3.3.2 Análise dos Trabalhos de Pesquisas Relacionados

O trabalho de Saravanan (2017) utiliza dados de um *ISP*, que são organizações que ofereçem serviços de acesso, participação ou utilização da Internet. Além disso, o algoritmo utilizado foi o *neuro-fuzzy clustering* e o ambiente utilizado para estudo foi o simulador de rede denominado *ns-2 (Network Simulator 2)* [11].

O estudo de Škrjanc et al. (2017) utiliza dados do *National Institute of Information and Communications (NITC)* [12]. Este é o principal instituto de pesquisa de informações e comunicações nacional do Japão. A missão do *NICT* é realizar pesquisa e desenvolvimento na área de tecnologia da informação e comunicação. Além disso, o algoritmo estudado foi o *cauchy possibilistic clustering (eCauchy)*.

O ambiente utilizado no trabalho de Patil, Krishna e Kumar (2020) foi o *Common Open Research Emulator (CORE)* [13], que é uma ferramenta para construção de redes virtuais. O *CORE* constrói uma representação de uma rede de computadores real que roda em tempo real, ao contrário da simulação, onde modelos abstratos são usados. A emulação de execução ao vivo pode ser conectada a redes físicas e roteadores. Ele fornece um ambiente para a execução de aplicativos e protocolos reais, aproveitando as ferramentas fornecidas pelo sistema operacional Linux. Além disso, essa solução também utiliza o *framework Apache Spark* e o algoritmo *k-means*.

Já no trabalho de Zolotukhin et al. (2016) o ambiente escolhido para desenvolvimento do estudo foi o *Realistic Global Cyber Environment (RGCE)* [14] e o algoritmo *weighted fuzzy c-means* . Ele reúne o mundo global realista e ambientes reais de organização em uma nuvem privada isolada. Além disso, combina técnicas de virtualização, dispositivos físicos e sistemas específicos de negócios. Também é possível criar ambientes personalizados para as necessidades específicas, como por exemplo pesquisa ou treinamento.

A pesquisa desenvolvida por Zhou et al. (2018) também utilizou como apoio a arquitetura *Apache Spark*, mais precisamente a solução *Kafka* [15]. Ela é uma plataforma de *streaming* de eventos distribuída e de código aberto usada por milhares de empresas em *pipelines* de dados de alto desempenho, análise de streaming e integração de dados e aplicativos. Os algoritmos utilizados na solução foram: *naive bayes*, *logistic regression* e *decision tree*. Além disso, todo o software foi executado na *AWS (Amazon Web Services)* [16], que é uma plataforma de serviços de computação em nuvem oferecida pela *Amazon*.

A proposta de Zolotukhin e Hämäläinen (2018) utiliza uma rede virtual, porém os autores não especificaram qual arquitetura, ferramenta ou ambiente foi utilizado para o desenvolvimento do estudo. Além disso, os algoritmos estudados foram *k-means*, *k-medoids* e *fuzzy c-means*.

[11] https://www.isi.edu/nsnam/ns/index.html
[12] https://www.nict.go.jp/en/
[13] https://www.nrl.navy.mil/Our-Work/Areas-of-Research/Information-Technology/NCS/CORE/
[14] https://jyvsectec.fi/cyber-range/overview/
[15] https://kafka.apache.org/
[16] https://aws.amazon.com/pt/

A solução proposta por Ivannikova, Zolotukhin e Hämäläinen (2017) utilizou a arquitetura *SDN* como base. Para isso, utilizaram a plataforma *Opendaylight* [17], que é uma plataforma e um controlador *SDN* de código aberto e modular utilizado para personalizar e automatizar redes de qualquer escala. Além disso, foi executada na plataforma *Openstack* [18], uma plataforma *open source* que utiliza recursos virtuais agrupados para criar e gerenciar nuvens públicas e privadas. As ferramentas disponíveis na plataforma abrangem serviços essenciais de *cloud computing*: processamento, rede e armazenamento. Por fim, os algoritmos utilizados foram *k-means*, *CURE* e *streaming k-means*.

O estudo de Campbell, Kiran e Wala (2020) objetivou o estudo de redes *Wide Area Network (WAN)*. Ele utilizou como ambiente os *ISPs* e executou uma análise em dados extraídos de três data centers reais de janeiro até maio de 2020. Os dados foram coletados utilizando ferramentas como os protocolos *Simple Network Management Protocol (SNMP)*, *sflow* e *Netflow*. Dessa maneira, foi considerado padrões de utilização normais para dias de semana e também para os finais de semana. Os algoritmos utilizados no estudo foram *k-means* e *GMM*.

A pesquisa desenvolvida por Ring et al. (2017) utilizou o ambiente *OpenStack* jutamente com a ferramenta *Open vSwitch* [19] para desenvolver a solução. Ela integrou os algoritmos *CluStream* e *DenStream (CUF)*.

Guerrieri e Montresor (2012) propõem o algoritmo *DS-means*. Eles utilizaram como ambiente de estudo um software que faz um simulação *Peer to Peer (P2P)* chamado de *PeerSim*. O algoritmo proposto é uma versão distribuída do algoritmo *k-means* e pode ser utilizado em um *ISP*, por exemplo, sem a necessidade de compartilhar os dados de seus usuários.

Após a análise dos trabalhos relacionados, constatou-se que os trabalhos Saravanan (2017), Škrjanc et al. (2017), Patil, Krishna e Kumar (2020), Zolotukhin et al. (2016), Zhou et al. (2018), Zolotukhin e Hämäläinen (2018), Campbell, Kiran e Wala (2020), Ring et al. (2017) e Guerrieri e Montresor (2012) utilizaram simulação de rede como tipo ambiente. Outros trabalhos como Patil, Krishna e Kumar (2020) e Zhou et al. (2018) utilizaram o ambiente *Apache Spark* e apenas um trabalho Ivannikova, Zolotukhin e Hämäläinen (2017) usou o ambiente *SDN* em seus experimentos e nenhum dos trabalhos utilizou a arquitetura *NFV* em seus experimentos. Os trabalhos revistos foram sumarizados e podem ser visualizados na Tabela 3. Diferente dos trabalhos apresentados, este trabalho pretende avaliar quantitativamente e qualitativamente algoritmos de aprendizagem de máquina não supervisionados que trabalhem com a estratégia de fluxo de dados na detecção de ataques *DDoS* em ambientes *SDN/NFV*.

[17] https://www.opendaylight.org/
[18] https://www.openstack.org/
[19] https://www.openvswitch.org/

Tabela 3 – Comparação entre os trabalhos relacionados da revisão sistemática.

Autor (Ano)	**Algoritmo**	***Big-data (Apache)***	**Ambiente**
(SARAVANAN, 2017)	*Neuro-fuzzy*	Não	*nS-2 Simulator*
(ŠKRJANC et al., 2017)	*eCauchy*	Não	Dados coletados da *darknet*
(PATIL; KRISHNA; KUMAR, 2020)	*K-means com Apache Spark*	*Apache Flume*	Emulador *CORE*
(ZOLOTUKHIN et al., 2016)	*Weighted Fuzzy C Means*	Não	*RGCE Cyber Range*
(ZHOU et al., 2018)	*Naive Bayes, Logistic Regression* e *Decision Tree*	*Kafka*	*AWS*
(ZOLOTUKHIN; HÄMÄLÄINEN, 2018)	*k-means, k-medoid* e *fuzzy c means*	Não	Rede virtual
(IVANNIKOVA; ZOLOTUKHIN; HÄMÄLÄINEN, 2017)	*K-Means*, *CURE* e *streaming k-Means*	Não	*SDN*
(CAMPBELL; KIRAN; WALA, 2020)	*K-means* e *GMM*	Não	*WAN*
(RING et al., 2017)	*CUF (CluStream e DenStream)*	Não	*OpenStack*
(GUERRIERI; MONTRESOR, 2012)	*DS-means*	Não	*P2P*

4

Algoritmos de aprendizagem de máquina avaliados

Este capítulo aborda os algoritmos de aprendizagem de máquina estudados. Todos eles são algoritmos de agrupamento. A escolha pela utilização de algoritmos não supervisionados se dá por eles não necessitarem de uma rotulação prévia dos dados. Dessa forma, não é preciso possuir conhecimento prévio dos dados que se quer classificar e com isso, pretende-se melhorar a adaptação a novos cenários. Foi escolhido estudar algoritmos trabalhassem com a estratégia de fluxo de dados, pois, a natureza intrínseca do ambiente *SDN/NFV* é de se trabalhar com fluxos de dados.

Segundo Silva et al. (2013), os algoritmos de agrupamento baseados em fluxo de dados podem ser divididos em duas etapas principais: etapa de abstração de dados, também conhecida como etapa ou fase on-line e a etapa de agrupamento, também conhecida como etapa ou fase off-line. A etapa de abstração on-line resume o fluxo de dados com a ajuda de estruturas de dados particulares para lidar com as restrições de espaço e memória que o cenário possui. Essas estruturas de dados resumem o fluxo a fim de preservar o significado dos objetos originais sem a necessidade de armazená-los de fato. Entre as estruturas de dados comumente implementadas destacam-se: *feature vector*, *coreset tree* e *data grid*.

Depois de realizar a etapa de abstração de dados, os algoritmos obtêm uma partição de dados por meio de uma etapa de agrupamento off-line. Para isso, o componente off-line utiliza alguns parâmetros a fim de obter uma compreensão rápida dos grupos gerados a partir fluxo de dados. Dentre os parâmetros de entrada estão as estatísticas dos resumos do fluxo de dados. Dessa maneira, é possível utilizar os algoritmos de agrupamento tradicionais com o propósito de encontrar partição de dados sobre os resumos, cujo tamanho é relativamente pequeno em comparação com todo o fluxo de dados (GHESMOUNE; LEBBAH; AZZAG, 2016).

Silva et al. (2013), destaca que a forma do *cluster* estará diretamente relacionada ao algoritmo de agrupamento que será utilizado. O algoritmo *k-means* gera *clusters* hiperesféricos, por exemplo, já o algoritmo *DBSCAN* permite a descoberta de *clusters* de formato arbitrário. Este

trabalho avaliou os seguintes algoritmos: *BIRCH*, *Mini-batch k-means*, *CluStream*, *StreamKM++*, *DenStream* e *D-Stream*.

4.1 *BIRCH*

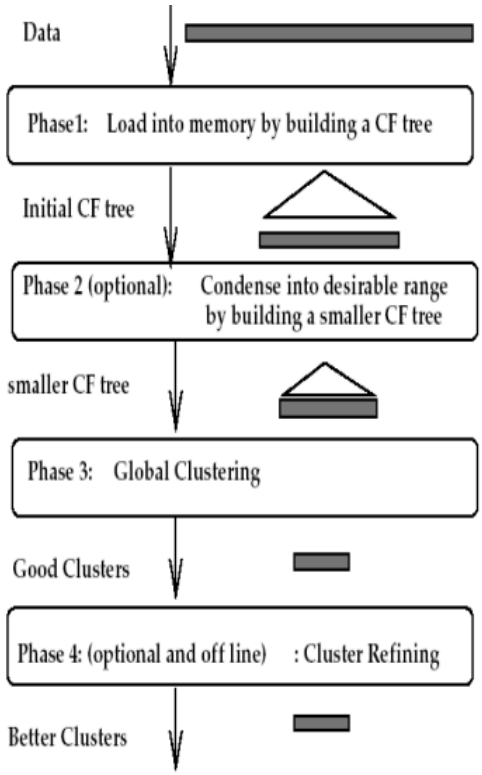
Zhang, Ramakrishnan e Livny (1996) desenvolveram um método de agrupamento de dados chamado *Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)*. Eles foram motivados em evidenciar uma possibilidade de encontrar padrões úteis em grandes massas de dados multidimensionais, levando em consideração restrições de memória e custos computacionais de entrada e saída. O algoritmo *BIRCH* agrupa incrementalmente e dinamicamente as métricas multidimensionais dos pontos de dados para tentar produzir o agrupamento de melhor qualidade com os recursos disponíveis (tempo e memória). Além disso, foi o primeiro algoritmo de agrupamento proposto na área de banco de dados para lidar com ruído de forma eficaz.

O *BIRCH* utiliza as estruturas de dados chamadas de *Clustering Feature (CF) e CF Tree*. O *CF* é uma sumarização tripla, ou seja, três informações são mantidas a respeito do *cluster*. Dados N pontos de dados d-dimensional em um agrupamento: $\vec{X_i}$ onde i = 1, 2, ..., N, o *CF* é um vetor definido conforme a tripla: $CF = (N, \vec{LS}, SS)$. N é o número de pontos de dados dentro do grupo, $\vec{LS}$ é a soma linear dos N pontos de dados, ou seja, $\sum_{i=1}^{N} \vec{X_i}$. Por fim, o SS é a soma quadrada dos N pontos de dados, ou seja, $\sum_{i=1}^{N} \vec{X_i}^2$. Pode-se pensar em um grupo como um conjunto de pontos de dados, mas o vetor *CF* é armazenado como resumo. Este resumo não é apenas eficiente, já que armazena menos dados do que todos os pontos de dados no cluster, mas também é preciso. Isso porque, ele é suficiente para calcular todas as medidas necessárias para fazer as decisões de agrupamento.

Ademais, a *CF Tree* é uma árvore de altura balanceada do tipo B+. Ela possui dois parâmetros, o fator de ramificação B e limiar T. Cada nó não folha contém no máximo B entradas da forma $[CF_i, child_i]$, onde i = 1,2, ..., B e $child_i$ é um ponteiro a seu i-ésimo nó filho, e CF_i, é o *CF* do subagrupamento representado por este filho. Portanto, um nó não folha representa um agrupamento formado por todos os subagrupamentos representados por suas entradas. Um nó folha contém no máximo, L entradas, cada uma da forma $[CF_i]$, onde i = 1, 2, ... L. O tamanho da árvore é uma função de T, quanto maior T for, menor é a árvore. A *CF Tree* será construída dinamicamente à medida que novos objetos de dados são inseridos. Ela é uma representação muito compacta do conjunto de dados porque cada entrada em um nó folha não é um único dado ponto, mas um subgrupo que absorve muitos pontos de dados com diâmetro abaixo do limite T específicado.

A Figura 12 apresenta a visão geral do algoritmo *BIRCH*. A tarefa principal da fase 1 é varrer todos os dados e construir a *CF Tree* utilizando a memória disponível. A fase 2 e 4 são opcionais. Já na fase 3, é onde ocorre o agrupamento global das folhas da árvore, ou seja, a fase 1 é etapa on-line e a fase 3 é a etapa off-line do algoritmo.

Figura 12 – Visão geral do algoritmo *BIRCH*



Fonte:(ZHANG; RAMAKRISHNAN; LIVNY, 1996).

4.2 *Mini-batch k-means*

Sculley (2010) propôs modificações no popular algoritmo não supervisionado de agrupamento *k-means* e desenvolveu o algoritmo *Mini-batch k-means*. O objetivo das modificações era atender aos requisitos dos usuários de aplicações web, dentre eles: latência, escalabilidade e dispersão. Para isso, desenvolveu-se uma otimização para o algorimo *k-means* original para que ele trabalhasse em minilotes. Isso reduziu o custo computacional em comparação ao clássico algoritmo em lote, como também, conseguiu-se produzir soluções significativamente melhores.

O algoritmo *k-means* clássico em lote possui um custo computacional alto para grandes conjuntos de dados, exigindo tempo de computação de $O(kns)$, onde n é o número de exemplos e s é o número máximo de elementos diferentes de zero. Bottou e Bengio (1995) proporam um Algoritmo de Gradiente Estocástico ou *Stochastic Gradient Descent (SGD)* on-line que calculou o passo do gradiente descendente em um exemplo de cada vez. Enquanto o *SGD* converge rapidamente em grandes dados conjuntos, ele encontra soluções de qualidade inferior do que o algoritmo de lote devido ao ruído estocástico. A motivação por trás do desenvolvimento do novo algoritmo *mini-batch k-means* é que os minilotes tendem a ter menor ruído do que exemplos individuais em *SGD*, permitindo a convergência para melhores soluções. Além disso, não sofrem aumento do custo computacional quando os conjuntos de dados aumentam com redundância

exemplos.

4.3 *CluStream*

O algoritmo *Clustream* foi proposto por Aggarwal et al. (2003) e pode ser dividido em dois componentes: on-line e off-line. A fase on-line corresponde a coleta dos dados estatísticos e onde são gerados os *micro-clusters*, os quais mantêm as informações estatísticas sobre a localidade dos dados. Eles são definidos como uma extensão temporal do *cluster feature vector* utilizado também no algoritmo *BIRCH*. A propriedade de adição dos *micro-clusters* os torna uma escolha natural para problemas de fluxo de dados. Além disso, os *micro-clusters* são armazenados em *snapshots*, que é o estado de um sistema em um determinado momento, ao mesmo tempo que seguem um padrão piramidal. Este padrão fornece uma compensação eficaz entre os requisitos de armazenamento e a capacidade de recuperar as estatísticas dos resumos.

Um *micro-cluster* é um conjunto de pontos d-dimensional X_{i1} ... X_{in} com as marcações de tempo T_{i1} ... T_{in}, definido como a tupla ($C\vec{F}2^x$, $C\vec{F}1^x$, $CF2^t$, $CF1^t$, n), em que $C\vec{F}2^x$ e $C\vec{F}1^x$ correspondem a um vetor de d entradas. Para cada dimensão d, $C\vec{F}2^x$ corresponde a soma quadrada, $C\vec{F}1^x$ corresponde ao somatório dos valores, $CF2^t$ corresponde a soma dos quadrados das marcas de tempo, $CF1^t$ corresponde ao somatório das marcas de tempo e n é o número de pontos de dados.

Os *micro-clusters* gerados pelo algoritmo servem como uma representação estatística intermediária que tem condições de ser armazenada de uma forma eficiente mesmo para um grande volume de fluxo de dado. O componente off-line do *Clustream* não utiliza o mesmo volume de dados que a fase on-line, mas sim, as estatísticas armazenadas e resumidas dos *micro-clusters*. Para obter os agrupamentos iniciais dos *micro-clusters* o *Clustream* utiliza o algoritmo *k-means*, após isso, agrupa os *micro-clusters* em grupos maiores chamados de *macro-clusters*. Esse processo é chamado de *Global Clustering* e permite que cada nova amostra encontre um respectivo *micro-cluster*. Além disso, o algoritmo pode refazer a composição dos *macro-clusters*, sendo possível a união ou separação de *micro-clusters* e consequentemente criação de novos *macro-clusters*.

4.4 *StreamKM++*

Ackermann et al. (2012) desenvolveram um novo algoritmo chamado de *Streamkm++*. Ele é baseado no algoritmo *k-means* e utiliza pontos provenientes de fluxos de dados em um espaço euclidiano. Além disso, ele calcula uma pequena amostra ponderada do fluxo de dados e utiliza o algoritmo *k-means++*, proposto por Arthur e Vassilvitskii (2006). Para isso, foi necessário desenvolver uma nova maneira para construção do *coreset*. Em geral, o *coreset* é um pequeno conjunto ponderado de pontos que se aproxima do conjunto original de pontos de

entrada em relação a um determinado problema de otimização. O foco do trabalho de Ackermann et al. (2012) é propor um *coreset* para problemas de agrupamento euclidiano usando o *k-means* e que seja adequado para dados com grandes dimensões. Até a publicação do trabalho deles, as abordagens existentes eram baseadas em cálculos de *grids* e geravam *coresets* com dimensões exponenciais (FRAHLING; SOHLER, 2005). Por sua vez, os *grids* são uma maneira de sumarizar os dados de um fluxo de dados e particionar o espaço n-dimensional disponível em células de uma malha utilizando densidade. Mesmo que a inicialização *(seeding)* do *k-means++* funcionasse bem para dados com grandes dimensões, uma construção de *coreset* com base nesta abordagem se mostrou ser mais promissora.

A fim de implementar a construção de *coresets* de forma eficiente, desenvolveu-se uma nova estrutura de dados chamada de *coreset tree*. Ela armazena pontos de forma que se possa realizar uma amostragem adaptativa e rápida, semelhante ao *k-means++*. A vantagem dessa nova abordagem é que o tempo de execução é significativamente menor em comparação ao tempo de execução do *k-means++* original. Aliado a isso, utilizaram a técnica padrão utilizada em processamento de fluxo de dados chamada *merge-and-reduce* (HAR-PELED; MAZUMDAR, 2004). Depois de processar todo o fluxo de entrada, aplica-se o algoritmo *k-means++* nas amostras e para se obter agrupamento final, utiliza-se o algoritmo *k-means*.

O algoritmo funciona da seguinte maneira: extrai-se um pequeno conjunto de *coresets* de tamanho m, que é um parâmetro fixo, do fluxo de dados usando a técnica de *merge-and-reduce*. Esta técnica organiza os dados em um pequeno número de amostras, cada uma representando $2^i m$ pontos de entrada, onde i é inteiro. Quando duas amostras representarem o mesmo número de pontos de entrada, é necessário fazer a união deles (*merge*). Após isso, cria-se uma nova amostra a partir dos dados unidos (*reduce*). Na etapa de redução dos dados, utiliza-se a implementação desenvolvida da *coresets tree*. Por fim, sobre os *coresets* gerados, pode-se utilizar qualquer algoritmo baseado no *k-means*, a fim de se obter os agrupamentos finais.

4.5 *DenStream*

O algoritmo *DenStream* foi proposto por Cao et al. (2006) e objetiva ser uma nova abordagem de algoritmo não supervisionado de agrupamento que trabalha em um fluxo de dados contínuo. Similar ao *Clustream*, este algoritmo também possui duas fases: a fase on-line, a qual é realizada a manutenção dos *micro-clusters* e a fase off-line, responsável por gerar os *clusters* finais, sob demanda do usuário. Na fase on-line são utilizadas duas estruturas de dados denominadas de *p-micro-cluster* (*potential c-microclusters*), que são os microgrupos pontenciais e *o-micro-cluster* (*outlier micro-clusters*), que são os microgrupos de *outliers* canditados. O microgrupo *p-micro-cluster* para um tempo t, para um grupo de pontos próximos p_{i1} ... p_{in}, com as respectivas marcas de tempos T_{i1} ... T_{in} é definido como $\vec{CF}^1$, $\vec{CF}^2$, ω. O valor $\vec{CF}^1$ corresponde soma linear ponderada e definido como $\vec{CF}^1 = \sum_{j=1}^{N} \int (t - T_{ij}) p_{ij}$. O valor $\vec{CF}^1$

corresponde soma quadrada ponderada e definido como $\vec{CF}^2 = \sum_{j=1}^{N} \int (t - T_{ij}) p_{ij}^2$. Por fim, ω é o peso e é definido por $\omega = \sum_{j=1}^{N} \int (t - T_{ij})$, onde $\omega \geq \beta\mu$ e $0 < \beta \leq 1$ (β é o limiar para determinação dos *outliers*).

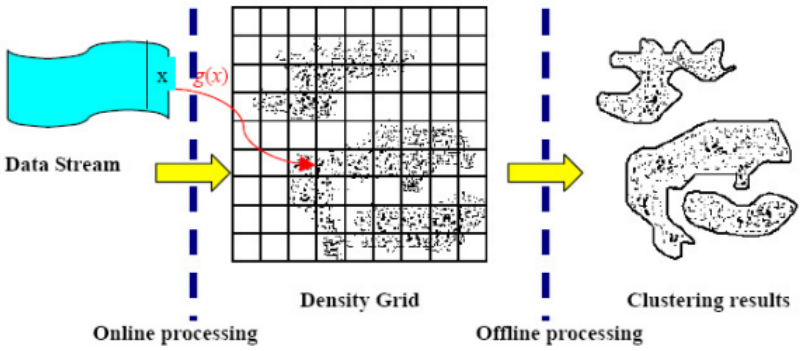
Na fase on-line, cada microgrupo possui um peso associado, utilizando como base a densidade das amostras. Existe um limite para este peso, e se por acaso for ultrapassado ele passa a ser considerado do microgrupo dos *outliers* (*o-micro-cluster*). O algoritmo recebe cada nova amostra e tenta adicioná-la ao *p-micro-cluster* mais próximo, para isso, utiliza-se a distância da amostra para os centros de cada microgrupo. Após isso, os parâmetros do microgrupo escolhido para a amostra são atualizados. Vale ressaltar que também é possível a criação de um novo microgrupo do tipo *o-micro-cluster*, assim como a transformação de um *o-micro-cluster* em *p-micro-cluster*.

Os microgrupos criados na fase on-line capturam a área de densidade dos fluxos de dados. Porém, para se obter grupos significativos, precisamos aplicar algum algoritmo de agrupamento para obter o resultado final. Para isso, na etapa off-line, o *DenStream* utiliza uma variante do algoritmo *DBSCAN* (ESTER et al., 1996), que é aplicada no conjunto *p-micro-clusters* a fim de se obter o resultado final do agrupamento. Cada *p-micro-cluster* c_p é considerado como um ponto virtual localizado no centro de c_p com peso ω. A variante do algoritmo DBSCAN inclui dois parâmetros denominados: *epsilon* e *minPoints*. *Epsilon* é a distância máxima para que dois pontos sejam considerados do mesmo grupo e *minPoints* é a quantidade mínima de pontos similares para se definir um grupo.

4.6 *D-Stream*

O algoritmo *D-Stream* foi proposto por Chen e Tu (2007) e também possui uma estrutura que agrupa as entradas do fluxo de dados utilizando uma abordagem baseada em densidade. Além disso, este algoritmo possui dois components, o on-line e o off-line. Porém, diferentemente do *DenStream* o componente on-line mapeia cada registro de dados de entrada em uma malha *(grid)*. Por fim, o componente off-line calcula a densidade da malha e agrupa as os dados da grade com base na densidade, a ilustração deste processo pode ser vista na Figura 13. O algoritmo adota uma técnica de decaimento de densidade para capturar as mudanças dinâmicas de um fluxo de dados e explorando as relações intrínsecas existentes entre o fator de decaimento, os dados densidade e a estrutura dos grupos.

Figura 13 – Visualização do uso de malha de densidade.



Fonte:(CHEN; TU, 2007).

Chen e Tu (2007) assumem que os dados de entrada possuem dimensões d e cada registro de dados de entrada é definido dentro do espaço $S = S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot ... \cdot S_d$, onde S_i é definido como o espaço de i dimensões. Para o algoritmo *D-Stream* o espaço dimensional d é dividido em malhas de densidade S. Para cada dimensão, seu espaço S_i, $i = 1, \cdots, d$ é dividido em partições p_i como $S_i = S_i, 1 \cup S_i, 2 \cup ... \cup S_i, p_i$. Então, o espaço de dados S é particionado em malhas de densidade, definido como $N = \prod_{i=1}^{d} p_i$. Para uma malha de densidade g que é composta por $S_{1,j_1} \times S_{2,j2} \times ... \times S_{d,j_d}$, onde $j = 1, ..., p_i$ denota-se que $g = (j_1, j_2, ..., j_d)$. Um registro de dados $x = (x_1, x_2, ..., x_d)$ pode ser mapeado para uma malha de densidade $g(x)$ da seguinte maneira: $g(x) = (j_1, j_2, ..., j_d)$, onde $x_i \in S_{i,j_i}$. Para cada registro de dado x atribuímos um coeficiente de densidade que diminui com o passar do tempo x. Na verdade, se x chegar no tempo t_c, define-se o carimbo de tempo como $T(x) = t_c$ e seu coeficiente de densidade como sendo $D(x,t) = \lambda^{t-T(x)} = \lambda^{t-t_c}$, onde $\lambda \in (0,1)$ é uma constate denominada de fator de decaimento.

5 Metodologia

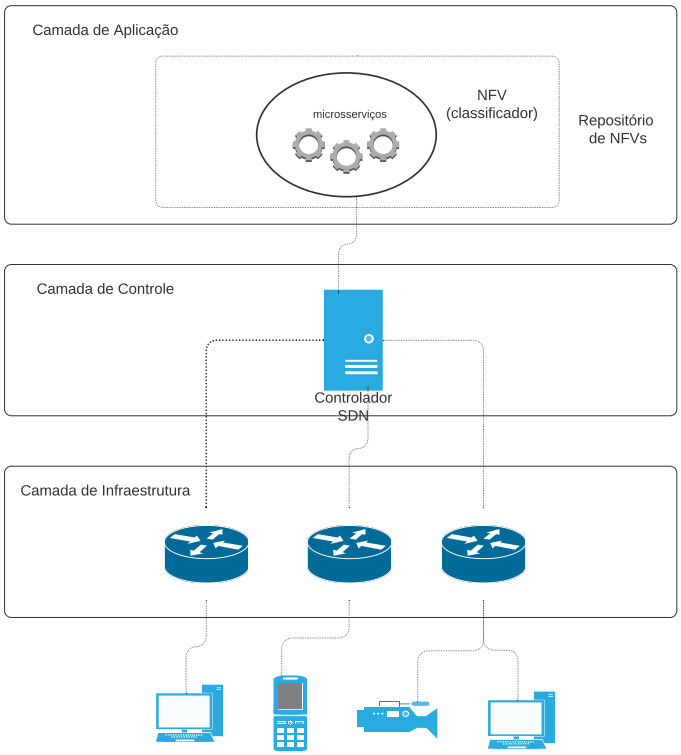
Este capítulo apresenta a metodologia utilizada para realizar a experimentação deste projeto. Dessa maneira, foi descrita a arquitetura, a plataforma, o dataset e as métricas utilizadas e desenvolvidas na experimentação.

5.1 Arquitetura *SDN/NFV*

Conforme discutido no Capítulo 3 há poucos trabalhos que avaliam o uso da aprendizagem de máquina em ambientes *SDN/NFV* e utilizam algoritmos de aprendizagem não supervisionado com estratégias de fluxos de dados. À vista disso, este trabalho se propõe a avaliar qualitativa e quantitativamente o uso da aprendizagem de máquina não supervisionada utilizando estratégias de fluxo de dados com o enfoque na detecção de ataques *DDoS* em um ambiente *SDN/NFV*. A escolha pela utilização de algoritmos não supervisionados se dá por eles não necessitarem de uma rotulação prévia dos dados. Dessa forma, não é preciso possuir conhecimento prévio dos dados que se quer classificar e com isso, pretende-se melhorar a adaptação à novos cenários. A natureza intrínseca do ambiente *SDN/NFV* é trabalhar com fluxos de dados e por isso a escolha por estudar algoritmos que utilizem a estratégia de fluxo de dados.

Como mencionado no Capítulo 2, a arquitetura *SDN* oferece um ambiente favorável a implementação de *NFV*. Dessa maneira, a arquitetura desenvolvida neste trabalho reúne os dois conceitos e também o conceito de microsserviços, conforme pode ser visto em Figura 14. A arquitetura *SDN* ficará responsável pelo gerenciamento e controle da rede, enquanto a arquitetura *NFV* ficará responsável pela virtualização de elementos de rede. Isso acarretará em uma ambiente mais flexível, de mais fácil customização e atualização.

Figura 14 – Arquitetura SDN/NFV.



Fonte: Próprio autor.

5.2 Plataforma

O hardware utilizado nos experimentos foi um notebook com processador *Intel Core i7-4500U*, 8 GB de *RAM DDR3* e 240 GB de armazenamento do tipo *Solid-State Drive (SSD)*. O hipervisor executado sobre este hardware foi o VirtualBox [1] na versão 6.0 e uma máquina virtual chamada *VM01* com sistema operacional *Ubuntu 16.04*, configurada com 3 GB de *RAM*, 1 *CPU (Central Process Unit)* e 20 GB de armazenamento. Para o ambiente *SDN*, foi utilizado o *POX Controller* [2], um controlador *SDN* de código aberto e com interface escrita em Python. O controlador *POX* oferece uma maneira eficiente de implementar o protocolo *OpenFlow* [3], que é um protocolo de comunicação entre o controlador e os dispositivos de rede (KAUR; SINGH; GHUMMAN, 2014). O *OpenFlow* foi desenvolvido para padronizar a comunicação baseada em software entre os *switches* e o controlador na arquitetura *SDN* (MCKEOWN et al., 2008). O experimento foi simulado utilizando a ferramenta Mininet [4], que emula uma rede para o ambiente *SDN*. Ela foi desenvolvida por pesquisadores da *Stanford University* e permite a criação de *hosts*, *switches*, controladores e links virtuais. Também foi utilizada a plataforma

1 https://www.virtualbox.org/
2 https://noxrepo.github.io/pox-doc/html/
3 https://opennetworking.org/
4 http://mininet.org/

Docker Container [5]. Ela facilita o desenvolvimento e gerenciamento de contêiner virtualizados e torna possível empacotar software ou um ambiente completo em um contêiner. Isso o torna um software portátil para qualquer tipo de host que tenha o *Docker* instalado. Além disso, acarreta em um melhor processamento, menor tempo de inicialização e economia de recursos. Há também a possibilidade de fazer upload de vários contêineres simultaneamente, consumindo menos recursos de hardware físico ou virtual (MOUAT, 2015). Para possibilitar o uso do conceito de *NFV*, utilizou-se software *Containernet* (PEUSTER; KARL; ROSSEM, 2016). Ele permite a adição e a remoção de contêineres de uma rede emulada em tempo de execução, o que não é possível somente com o *Mininet*. Além disso, é possível instalar softwares sob demanda, sem a necessidade de instalar novos equipamentos, como também, desenvolver e testar serviços na mesma infraestrutura, o que reduz custos e aumenta a velocidade de desenvolvimento (BONFIM; DIAS; FERNANDES, 2019).

5.3 *Dataset*

Em buscas prévias, não se encontrou nenhum um conjunto de dados (*dataset*) específico para o ambiente *SDN*, por esse motivo, foi criado um *dataset* sintético para a execução dos experimentos.

Toda decisão de encaminhamento do *OpenFlow* é feita utilizando as tabelas de decisão definidas pelos fluxos. No *OpenFlow* na versão 1.0, a tabela de fluxo é dividida em 24 características (HELLER, 2009). Dentre elas, 14 foram escolhidas para o desenvolvimento do *dataset*, por caracterizarem de maneira satisfatória o estado da rede e a ocorrência do ataque *DDoS*, são elas: *packet_count, byte_count, duration_sec, port, duration_nsec, nw_dst, dl_src, nw_proto, nw_tos, tp_dst, tp_src, dl_dst, nw_src and in_port*. A descrição de cada uma deles pode ser vista na Tabela 4.

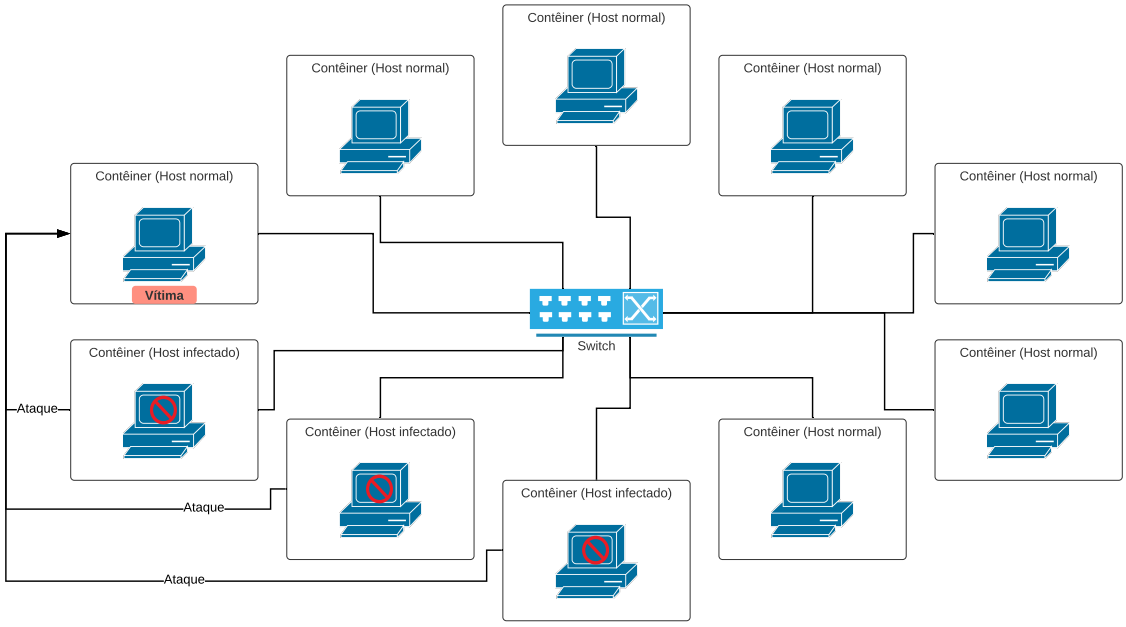
Tabela 4 – *Features* escolhidas.

Feature	**Descrição**
packet_count	Número de pacotes em um fluxo
byte_count	Número de *bytes* em um fluxo
duration_sec	Tempo que o fluxo ficou ativo em segundos
duration_nsec	Tempo que o fluxo ficou ativo em nanosegundos
port	Porta *TCP* da origem
nw_dst	Endereço *IP* de destino
dl_src	Endereço *Ethernet* da origem
nw_proto	protocolo *IP* ou 8 bits inferiores do *ARP opcode*
nw_tos	*IP ToS (Type Of Service)*
tp_dst	porta de destino *TCP/UDP*
tp_src	porta de origem *TCP/UDP*
dl_dst	endereço de destino *Ethernet*
nw_src	endereço de origem *IP*
in_port	porta de entrada do *switch*

[5] https://www.docker.com/

O *dataset* foi desenvolvido a partir dos dados coletados das tabelas de fluxo do controlador *POX* de uma topologia criada com o *Mininet*. A topologia continha 10 *hosts*, um *switch*, dentre eles 7 *hosts* eram normais e 3 *hosts* eram infectados, confome pode ser visto na Figura 15. Dos *hosts* normais 1 era a vítima e a coleta de dados durou 2 horas. Além disso, é necessário criar de tráfego de rede e para isso foi utilizada a ferramenta *API Scapy* [6]. Ela é uma ferramenta poderosa que possui interface em *Python* e é focada na manipulação de pacotes de rede. Com ela é possível forjar ou decodificar pacotes de um grande número de protocolos e enviá-los através da rede. Por isso, foi criado um *script* para gerar tráfego normal e para gerar tráfego de ataque, ambos aleatórios. Para o tráfego normal, foram utilizados os protocolos *TCP*, *UDP*, *ICMP*, *DNS*, *HTTP*, *ARP*, *IP*, *SMTP* e *DHCP*. Para o tráfego de ataque, foi utilizado o tipo de ataque de inundação *UDP*. Isso resultou em um conjunto de dados com um total de 59.674 amostras, dentre elas 39.778 com tráfedo normal e 19.896 com tráfego de ataque, ou seja, 66,66% das amostras são de tráfego normal e 33,34% são de tráfego de ataque.

Figura 15 – Topologia para criação do *dataset*.



Fonte: Próprio autor.

5.4 Métricas

As métricas de desempenho utilizadas para avaliar os resultados experimentais obtidos baseiam-se na matriz de confusão. Ela pode ser definida formalmente como uma matriz de valores inteiros não negativos $[c_{ij}]$ com $i = 1...N$, $j = 1...N$, onde N é o número de classes ou

[6] https://scapy.net/

agrupamentos. O elemento c_{ij} é o número de objetos escolhidos da classe i que foram atribuídos a classe j durante os testes. A soma de todos os elementos c_{ii}, ou seja, os elementos na diagonal principal da matriz, é o valor de todos os objetos classificados corretamente. Assim, todos os elementos fora da diagonal e diferentes de zero, por outro lado, indicam os erros cometidos (SUSMAGA, 2004). Um exemplo de uma matriz confusão pode ser visto na Tabela 5.

Tabela 5 – Exemplo de matriz confusão.

	Predição			
TPO	Positivos	Negativos		
AP	**TP**	**FN**	TPR	FNR
AN	**FP**	**TN**	FPR	TNR
ACC	PPV	FOR	$F1$	
	FDR	NPV		

A População Total ou TPO é igual a $TPO = AP + AN$, onde AP são os Positivos Reais, que é o total de positivos na População Total, ou seja, amostras de tráfego normal. Já AN são os Negativos Reais, que é o total de negativos na População Total, ou seja, amostras de tráfego de ataque. Além disso, o TP são os Verdadeiros Positivos e referem-se as amostras corretamente classificadas como normais. Já o TN são os Verdadeiros Negativos, ou seja, as amostras corretamente classificadas corretamente como ataque. Por sua vez, o FP se refere aos Falsos Positivos e são as amostras incorretamente classificadas como normais e o FN são os Falsos Negativos, aquelas que foram incorretamente classificadas como ataque.

Além disso, o Valor Preditivo Positivo ou PPV é quantidade de amostras positivas identificadas pelos algoritmos como de fato positivas, ou seja, tráfego normal. O NPV representa o Valor Preditivo Negativo, que é quantidade de amostras negativas identificadas pelos algoritmos como de fato negativas, ou seja, tráfego de ataque. A Taxa de Falsa Omissão ou FOR avalia quantos dos dados de ataque identificados eram normais. Já a Taxa de Falsa Descoberta ou FDR avalia quantos dos dados normais identificados eram de ataque.

Ademais, a Proporção de Positivos Corretos ou TPR mede a proporção de positivos que foram identificados corretamente como positivos e a Proporção de Negativos Corretos ou TNR mede a proporção de negativos que foram identificados corretamente como negativos. A Taxa de Falsos Negativos ou FNR avalia quantas amostras normais foram identificadas como de ataque. Já a Taxa de Falsos Positivos ou FPR avalia quantas amostras de ataque foram identificadas como normais. Além do mais, a acurácia ou ACC é total de acertos considerando o total de observações e o *f1-score* ou $F1$ é a média harmônica entre precisão e *recall* e indica a detecção correta geral sobre as amostras de teste. As métricas de desempenho utilizadas são definidas como:

$$PPV = \frac{TP}{TP + FP} \tag{5.1}$$

$$NPV = \frac{TN}{FN + TN} \tag{5.2}$$

$$FOR = 1 - NPV \tag{5.3}$$

$$FDR = 1 - PPV \tag{5.4}$$

$$TPR = \frac{TP}{AP} \tag{5.5}$$

$$TNR = \frac{TN}{AN} \tag{5.6}$$

$$FNR = 1 - TPR \tag{5.7}$$

$$FPR = 1 - TNR \tag{5.8}$$

$$F1 = \frac{2 * PPV * TPR}{PPV + TPR} \tag{5.9}$$

$$ACC = \frac{TP + TN}{TPO} \tag{5.10}$$

6

Experimentação e análise dos resultados

Este capítulo apresentará a primeira experimentação, desenvolvida a partir do Mapeamento Sistemático descrito no Capítulo 3. Após isso, apresentará a segunda experimentação e a análise qualitativa e quantitativa dos resultados obtidos para algoritmos *BIRCH*, *Mini-batch k-means*, *CluStream*, *StreamKM++*, *DenStream* e *D-Stream*.

6.1 Experimentação com algoritmos que não utilizam fluxo de dados

A primeira experimentação foi realizada com o intuito de analisar dois algoritmos de aprendizado de máquina não supervisionados, o *fuzzy c-means* e o *k-means*, trabalhando na detecção de ataques *DDoS* do tipo *UDP Flood*. Esse experimento estudou as implementações clássicas desses algoritmos, ou seja, o processamento deles não utilizava a estratégia de fluxo de dados.

A primeira experimentação é composta por três experimentos e para isso foi utilizada a mesma plataforma física e virtual descrita no Capítulo 5. Não se encontrou nenhum *dataset* específico para o ambiente *SDN*, sendo assim, foi desenvolvido um componente para o *POX Controller* com o intuito de gerar o *dataset* sintético utilizado nestes experimentos. O componente armazenou as estatísticas para um fluxo de dados atual em um arquivo *.csv* e para gerar tráfego normal e malicioso aleatório, foi utilizada a ferramenta *Scapy* para criar um script de geração de tráfego. Para o tráfego normal foram utilizados os protocolos *TCP*, *UDP* e *ICMP* e para o tráfego de ataque foi utilizado o ataque *DDoS UDP flood*. Dessa maneira, as informações de 4 *hosts* foram monitoradas durante 30 minutos. Com isso, o *dataset* desenvolvido continha 8.820 amostras, sendo 6.284 de tráfego normal e 2.536 de tráfego malicioso. Os experimentos foram simulados utilizando as ferramentas *Mininet*, *Docker Container* e *Containernet*.

Assim, três experimentos foram realizados: predição de dados utilizando o *dataset* de teste

em um ambiente não *SDN*, detecção em um ambiente *SDN* e detecção em um ambiente *SDN/NFV* com microsserviços. Nos dois últimos experimentos, o monitoramento e gerenciamento de rede foram feitos com base no pressuposto de que a rede já estaria comprometida, ou seja, a *botnet* já havia sido formada e os *hosts* já teriam sido infectados.

6.1.1 Experimento A

Esse experimento foi focado nos dois algoritmos *fuzzy c-means* e *k-means*. Para apoiá-los, foi utilizada a biblioteca do *scikit-learn* [1], uma ferramenta simples e eficiente para a análise e predição de dados. O algoritmo *k-means* está presente nativamente nele, mas, para usar o *fuzzy c-means*, foi necessário instalar um pacote *Python* utilizando a ferramenta *pip* [2] (DIAS, 2019). A massa de dados de treinamento possuía um total de 4.410 amostras do *dataset* original, ou seja, metade do *dataset* original foi usado para treinamento. Assim, o conjunto de dados de treinamento teve 3.142 amostras com tráfego normal e 1.268 amostras com tráfego malicioso. Para provar a eficácia dos algoritmos, foi utilizada a outra metade do conjunto de dados original, chamado de *dataset* de teste. Entre os recursos existentes no *dataset* elaborado, foram escolhidos os seguintes: *packet_count*, *byte_count*, *duration_sec*, *duration_nsec*, *tp_src* e *tp_dst*. A descrição de cada um deles pode ser vista na Tabela 6. Os dados foram pré-processados utilizando o pacote *sklearn.preprocessing*, porque, em geral, os algoritmos de aprendizado de máquina se beneficiam da padronização dos dados. O método utilizado neste experimento foi o *MinMaxscaler*, que é um método de pré-processamento que transforma recursos escalando cada um em um intervalo específico. Além disso, ele dimensiona e converte cada recurso individualmente, de modo que pertença ao intervalo especificado no conjunto de treinamento, por exemplo, entre zero e um.

Tabela 6 – *Features* escolhidas

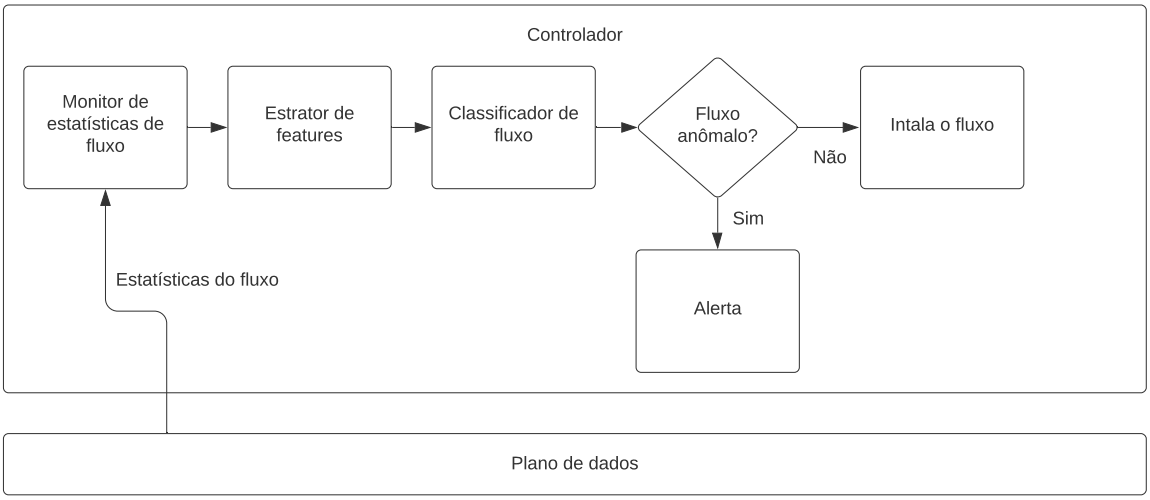
Feature	**Descrição**
packet_count	Número de pacotes de um fluxo
byte_count	Número de *bytes* de um fluxo
duration_sec	Tempo que o fluxo está ativo em segundos
duration_nsec	Tempo que o fluxo está ativo em nanosegundos
tp_src	Porta *TCP* remetente
tp_dst	Porta *TCP* destinatária

6.1.2 Experimento B

Nesse experimento, foi utilizando somente o ambiente *SDN* e foi criado um componente para o *POX Controller* para executar a classificação de fluxo, conforme ilustrado na Figura 16. Através do monitor de estatísticas, as *features* do fluxo são extraídas e passadas para o classificador. Depois disso, o classificador organizará o fluxo como normal ou anômalo. Se o fluxo estiver normal, ele será instalado e funcionará normalmente; caso contrário, o alerta de fluxo anômalo será acionado.

[1] https://scikit-learn.org/stable/
[2] https://pypi.org/project/pip/

Figura 16 – Componente *POX* proposto.

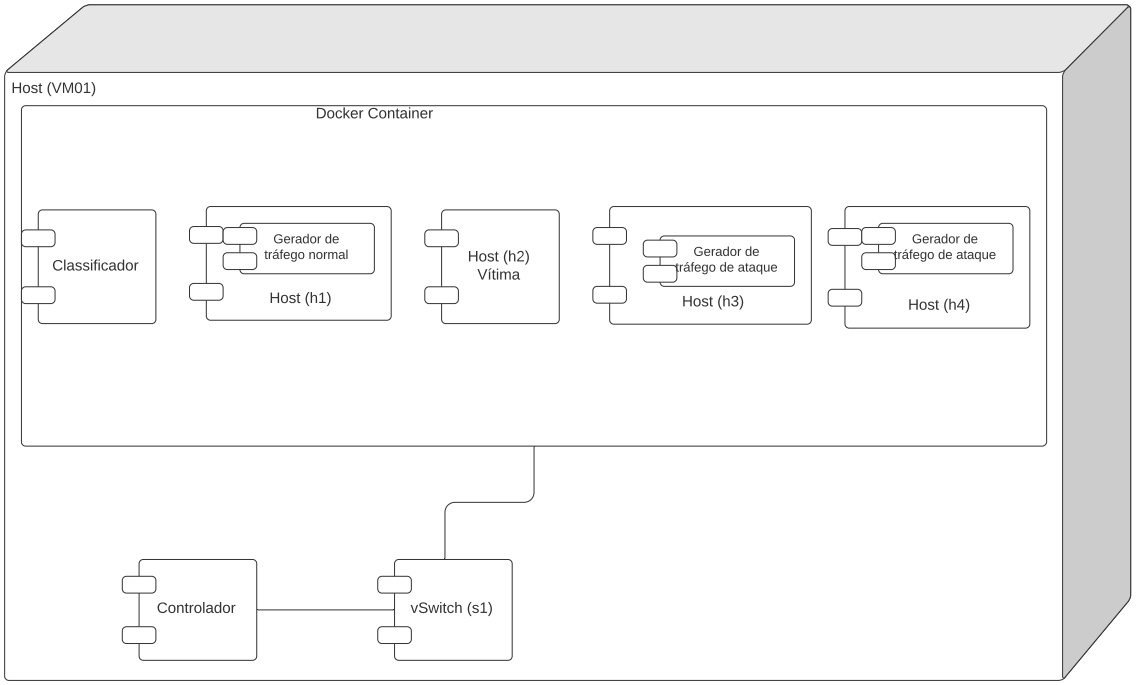


Fonte: Próprio autor.

6.1.3 Experimento C

Neste experimento foi utilizado o ambiente *SDN/NFV*, implementando microsserviços. A Figura 17 apresenta um diagrama de componentes, que é uma visão estática de como o sistema foi implementado e quais componentes são utilizados para detectar ataques *DDoS* do tipo *UDP flood*. O ambiente consiste em um *host* da *VM01*, um controlador *POX* que é executado diretamente no *host* da *VM01* e em um ambiente em contêiner, onde os serviços de rede virtual serão executados. Para isso, foi necessário instalar a ferramenta *Containernet*, que inclui as ferramentas *Docker Container* e *Mininet*. No *Docker Container*, foram criados os seguintes microsserviços de rede virtual: o classificador de fluxo, um *host* de rede com gerador de tráfego normal e um *host* de rede com gerador de tráfego de ataque. Para esse experimento, foram criados quatro *hosts* de rede, o *host* h1, que gerava tráfego de rede normal entre todos os *hosts*; o *host* h2 que foi a vítima e recebeu os ataques *DDoS* e os *hosts* h3 e h4, que geraram o ataque para h2.

Figura 17 – Diagrama de componentes.



Fonte: Próprio autor.

6.1.4 Resultados

As métricas usadas para analisar os algoritmos *fuzzy c-means* e *k-means* nos experimentos foram: *precision*, *recall*, *f1-score* e *support*, presentes no pacote *sklearn.metrics* da biblioteca *scikit-learn*. Cada uma das métricas é descrita na Tabela 7.

Tabela 7 – Métricas estudadas.

Métrica	**Descrição**
Precision	tp / (tp + fp), tp é o número de verdadeiros positivos e fp o número de falsos positivos. A precisão é intuitivamente a capacidade do classificador de não rotular como positiva uma amostra negativa.
Recall	tp / (tp + fn), tp é o número de verdadeiros positivos e fn o número de falsos negativos. O recall é intuitivamente a capacidade do classificador de encontrar todas as amostras positivas.
F1-score	Pode ser interpretado como uma média ponderada da precisão e recuperação
Support	Número de ocorrências de cada classe em y_true.

Além disso, foram utilizadas as seguintes métricas: acurácia e tempo de execução do algoritmo. Acurácia é a razão de acertos sobre o número total de pontos e indica uma performance geral do modelo. Abaixo está a fórmula de cálculo da acurária, Equação 6.1, onde Ac é a acurária calculada, $ClassfC$ é o número de classificações corretas e $TAmostras$ é o número total de

amostras. Para calculá-la, foi utilizado o método *accuracy_score*, também presente no pacote *sklearn.metrics*. Para medir o tempo de execução, foi utilizado o módulo *time* do *Python*.

$$Ac = \frac{ClassfC}{TAmostras} \tag{6.1}$$

Para os experimentos A, B e C, o pré-processamento do *dataset* de teste foi realizado utilizando o *MinMaxScaler*. Após o uso do modelo criado, os algoritmos obtiveram os mesmos resultados, constantes na Tabela 8.

Tabela 8 – Métricas para os algoritmos *fuzzy c-means* e *k-means*.

	Precision	***Recall***	***F1-score***	***Support***
0	1.00	1.00	1.00	3141
1	1.00	1.00	1.00	1269
Avg/ Total	1.00	1.00	1.00	4410

Ambos algoritmos apresentaram uma taxa de acurácia de 99,98%, ou seja, isso revelou que ambos possuem acurácia satisfatória na detecção desse tipo de ataque, conforme pode ser visto na Tabela 9. Para o cenário do Experimento A, o algoritmo *fuzzy c-means* obteve 0,61 segundos de tempo de execução, enquanto o algoritmo *k-means* obteve 0,44 segundos. Dessa maneira, o algoritmo *k-means* foi 0,17 segundos ou 27,86% mais rápido que o *fuzzy c-means*. Já para a Experimento B, o tempo de execução do algoritmo *fuzzy c-means* foi de 1,25 segundos, enquanto o algoritmo *k-means*, foi de 1,09 segundos. O algoritmo *k-means* provou ser 0,16 segundos mais rápido, ou 12,8% mais rápido. O tempo de execução nesse cenário pode ter aumentado devido à execução do método de classificação que competiu pelo processamento de recursos na máquina com o processo do controlador *POX*, uma vez que a classificação ocorreu no controlador. Para o Experimento C, o tempo de execução do algoritmo *fuzzy c-means* foi de 0,03 segundos e do *k-means* foi de 0,02 segundos. O algoritmo *k-means* acabou sendo 0,01 segundos mais rápido, o que torna sua execução 33,33% mais rápida. O tempo de execução neste cenário pode ter sido bastante reduzido pelo fato de a execução do método de classificação ter sido realizada em um contêiner, ou seja, um processo específico.

Tabela 9 – Acurácia e tempo de execução dos algoritmos.

Experimento	**Acurácia (*fuzzy c-means*)**	**Tempo de execução (*fuzzy c-means*)**	**Acurácia (*k-means*)**	**Tempo de execução (*k-means*)**
A	99,98%	0,61	99,98%	0,44
B	99,98%	1,25	99,98%	1,09
C	99,98%	0,03	99,98%	0,02

6.1.5 Considerações sobre a primeira experimentação

Durante o experimento, foram encontrados alguns desafios e, entre eles, notamos a impossibilidade de classificar todos os fluxos sequencialmente no ambiente do Experimento C, que usava o conceito de *NFV* e microsserviços. Isso se deve ao fato de que a chamada ao

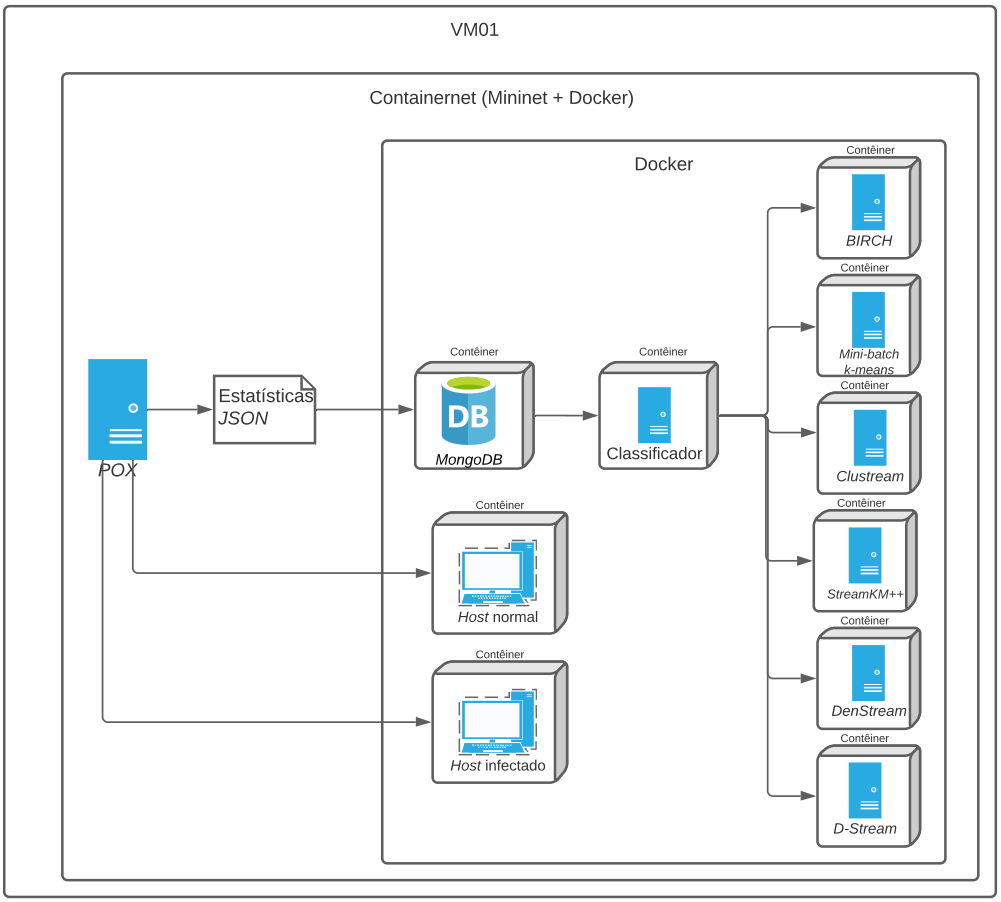
classificador ocorre por meio do método *POST/HTTP*. Para isso, foi utilizada uma biblioteca *Python* chamada *Request* [3], para fazer a solicitação, enquanto no lado do servidor usamos o *framework Flask* [4], que é uma estrutura da Web escrita em *Python* e baseada na biblioteca *WSGI* [5]. Ao fazer chamadas para o *endpoint* do classificador de maneira sequencial e ininterrupta, a cada fluxo que chegava ao controlador, a *API requets* entra em colapso e interrompe o trabalho. Ou seja, é como se o ataque *DDoS* também estivesse afetando a biblioteca. Assim, para a realização do experimento, foi necessário classificar apenas algumas amostras dos fluxos. Os experimentos mostraram-se eficazes na detecção de ataques de *DDoS* do tipo *UDP flood*, utilizando os algoritmos *fuzzy c-means* e *k-means*. Por fim, há necessidade de se estudar algoritmos que utilizem a estratégia de fluxos de dados, pois eles trabalham de maneira mais satisfatória com os recursos restritos de memória.

6.2 Experimentação com algoritmos que utilizam fluxo de dados

A experimentação foi realizada com o objetivo de analisar qualitativa e quantitativamente os algoritmos *BIRCH*, *Mini-batch k-means*, *CluStream*, *StreamKM++*, *DenStream* e *D-Stream*. Diferentemente da primeira experimentação, este experimento estudou algoritmos que utilizam a estratégia de fluxo de dados na detecção de ataques *DDoS* do tipo inundação *UDP* no ambiente *SDN/NFV* e implementando microsserviços. A plataforma utilizada neste experimento é a mesma descrita no Capítulo 5, composta pela máquina virtual *VM01* juntamente com as ferramentas *Mininet*, *Docker Container* e *Containernet*. No *Docker Container*, foram criados dois modelos de *hosts* em forma de contêiner, um *host* comum ou normal e outro infectado pela *botnet*. Para a realização da análise quantitativa do experimento foram criados dinamicamente 15 *hosts*, sendo 5 infectados, 10 normais e 1 vítima do ataque *DDoS*. A arquitetura desenvolvida pode ser vista na Figura 18.

[3] https://docs.python-requests.org/en/master/
[4] https://flask.palletsprojects.com/en/2.0.x/
[5] https://wsgi.readthedocs.io/en/latest/

Figura 18 – Arquitetura desenvolvida para o experimento.



Fonte: Próprio autor.

Além disso, foram criados os seguintes microsserviços de rede virtual em forma de contêineres: banco de dados, Classificador de fluxo e algoritmos de aprendizado de máquina não supervisionados. O fluxo das informações se inicia com a extração das estatísticas do fluxo de dados do Controlador *POX* utilizando o componente desenvolvido e descrito na subseção 6.1.3. As estatísticas do fluxo são extraídas em formato *JSON (JavaScript Object Notation)*. Após isso, elas são gravadas em um contêiner contendo o banco de dados *MongoDB* [6]. Ele é um banco de dados de propósito geral poderoso, flexível e escalável. Como também, é um banco de dados do tipo *NoSQL (Not Only SQL)*, não relacional e orientado a documento. Dessa maneira, é possível utilizar documentos ou *arrays* possibilitando a representação de relacionamentos hierárquicos complexos utilizando somente um registro (CHODOROW, 2013). Conceitualmente os documentos utilizados podem ser de diferentes tipos, dentre eles: *JSON*, *BSON*, *XML* e *BLOB* (GYŐRÖDI et al., 2015). Este tipo de banco de dados provê uma solução com altas taxas de transferências, escalabilidade horizontal e que podem ser executados em hardwares de propósito geral, características necessárias para o ambiente proposto (STRAUCH; SITES; KRIHA, 2011).

[6] https://www.mongodb.com/

Após isso, o Classificador que lê do banco de dados *MongoDB* a amostra a ser avaliada e depois executa o pré-processamento nela. Para o pré-processamento, foi utilizada a reescala de dados *MinMaxScaler* do *scikit-learn*, a mesma técnica utilizada no primeiro experimento. Por fim, o Classificador chama o algoritmo de aprendizado de máquina para classificar a amostra como normal ou de ataque. Para os algoritmos *BIRCH* e *Mini-batch k-means* foi utilizado a implementação disponível na biblioteca *scikit-learn*. Já para os algoritmos *Clustream* e *StreamKM++*, foi utilizado um pacote chamado *clusopt-core* [7] e para os algoritmos *DenStream* e *D-Stream* foram utilizadas implementações dos algoritmos originais. Vale ressaltar que a fim de executar todos os algoritmos igualitariamente e se obter uma avaliação mais precisa, cada amostra lida do banco de dados era enviada paralelamente para todos os algoritmos estudados neste trabalho para ser processada.

6.3 Análise de resultados

Esta seção apresentará a análise qualitativa e quantitativa dos resultados obtidos.

6.3.1 Análise qualitativa

A análise qualitativa visa comparar o quão efetivos são os algoritmos na detecção de ataques *DDoS*. Para realizar esta análise, foi utilizado o *dataset*, mais precisamente dois subgrupos dele e as métricas descritas no Capítulo 5. O primeiro subgrupo do *dataset* é utilizado para realizar o treinamento do algoritmo. Este subgrupo possui 17.902 amostras, o que corresponde a 30% do número de amostras totais e é utilizado na fase on-line dos algoritmos. Já o segundo subgrupo é utilizado para realizar a avaliação do algoritmo e é chamado de grupo de teste. Este subgrupo possui 5.968 amostras, o que corresponde a 10% do número de amostras totais e é utilizado na fase off-line dos algoritmos.

Desse modo, como o *dataset* está dividido em 66,66% de tráfego normal e 33,34% de tráfego de ataque, obtêm-se uma margem de erro com um nível de confiança de 99% igual a $e = 2.575 \cdot \sqrt{\frac{0{,}3334 \cdot 0{,}6666}{5.968}} \approx 1,5\%$. Segundo Evans e Rosenthal (2009), a margem de erro corresponde a Equação 6.2, sendo e a margem de erro, Z o valor crítico de confiança respectivo a um nível de confiança de 99%, p a porcentagem de amostras de ataque e n o total amostras.

$$e = Z \cdot \sqrt{\frac{p \cdot (1-p)}{n}} \tag{6.2}$$

Conforme pode ser visto na Tabela 10 o algoritmo *BIRCH* obteve 99,98% de acurácia e *f1-score* igual a 0,99. Além disso, obteve um *PPV* de 99,97%, um *NPV* de 100%, um *FOR* de 0%, um *FDR* de 0,02%, um *TPR* de 100%, um *TNR* de 99,94%, um *FNR* de 0% e um *FPR* de 0,05%

[7] https://github.com/giuliano-oliveira/clusopt

Tabela 10 – Matriz confusão para o algoritmo *BIRCH*.

	Predição			
$TPO = 5968$	Positivos	Negativos		
$AP = 3974$	3974	0	$TPR = 100\%$	$FNR = 0\%$
$AN = 1994$	1	1993	$FPR = 0,05\%$	$TNR = 99,94\%$
$ACC = 99,98\%$	$PPV = 99,97\%$	$FOR = 0\%$	$F1 = 0,99$	
	$FDR = 0,02\%$	$NPV = 100\%$		

Conforme pode ser visto na Tabela 11 o algoritmo *Mini-batch k-means* obteve 99,11% de acurácia e *f1-score* igual a 0,99. Além disso, obteve um *PPV* de 99,97%, um *NPV* de 97,45%, um *FOR* de 2,54%, um *FDR* de 0,02%, um *TPR* de 98,69%, um *TNR* de 99,94%, um *FNR* de 1,3% e um *FPR* de 0,05%.

Tabela 11 – Matriz confusão para o algoritmo *Mini-batch k-means*.

	Predição			
$TPO = 5968$	Positivos	Negativos		
$AP = 3974$	3922	52	$TPR = 98,69\%$	$FNR = 1,3\%$
$AN = 1994$	1	1993	$FPR = 0,05\%$	$TNR = 99,94\%$
$ACC = 99,11\%$	$PPV = 99,97\%$	$FOR = 2,54\%$	$F1 = 0,99$	
	$FDR = 0,02\%$	$NPV = 97,45\%$		

Conforme pode ser visto na Tabela 12 o algoritmo *Clustream* obteve 99,91% de acurácia e *f1-score* igual a 0,99. Além disso, obteve um *PPV* de 99,49%, um *NPV* de 99,84%, um *FOR* de 0,15%, um *FDR* de 0,05%, um *TPR* de 99,92%, um *TNR* de 99,89%, um *FNR* de 0,07% e um *FPR* de 0,1%.

Tabela 12 – Matriz confusão para o algoritmo *Clustream*.

	Predição			
$TPO = 5968$	Positivos	Negativos		
$AP = 3974$	3971	3	$TPR = 99,92\%$	$FNR = 0,07\%$
$AN = 1994$	2	1992	$FPR = 0,1\%$	$TNR = 99,89\%$
$ACC = 99,91\%$	$PPV = 99,49\%$	$FOR = 0,15\%$	$F1 = 0,99$	
	$FDR = 0,05\%$	$NPV = 99,84\%$		

Conforme pode ser visto na Tabela 13 o algoritmo *StreamKM++* obteve 99,89% de acurácia e *f1-score* igual a 0,99. Além disso, obteve um *PPV* de 99,49%, um *NPV* de 99,79%, um *FOR* de 0,2%, um *FDR* de 0,05%, um *TPR* de 99,89%, um *TNR* de 99,89%, um *FNR* de 0,1% e um *FPR* de 0,1%.

Tabela 13 – Matriz confusão para o algoritmo *StreamKM++*.

	Predição			
$TPO = 5968$	Positivos	Negativos		
$AP = 3974$	3970	4	$TPR = 99,89\%$	$FNR = 0,1\%$
$AN = 1994$	2	1992	$FPR = 0,1\%$	$TNR = 99,89\%$
$ACC = 99,89\%$	$PPV = 99,49\%$	$FOR = 0,2\%$	$F1 = 0,99$	
	$FDR = 0,05\%$	$NPV = 99,79\%$		

Conforme pode ser visto na Tabela 14 o algoritmo *DenStream* obteve 79,05% de acurácia e *f1-score* igual a 0,86. Além disso, obteve um *PPV* de 76,07%, um *NPV* de 100%, um *FOR* de 0%, um *FDR* de 23,92%, um *TPR* de 100%, um *TNR* de 37,31%, um *FNR* de 0% e um *FPR* de 62,68%.

Tabela 14 – Matriz confusão para o algoritmo *DenStream*

	Predição			
$TPO = 5968$	Positivos	Negativos		
$AP = 3974$	3974	0	$TPR = 100\%$	$FNR = 0\%$
$AN = 1994$	1250	744	$FPR = 62,68\%$	$TNR = 37,31\%$
$ACC = 79,05\%$	$PPV = 76,07\%$	$FOR = 0\%$	$F1 = 0,86$	
	$FDR = 23,92\%$	$NPV = 100\%$		

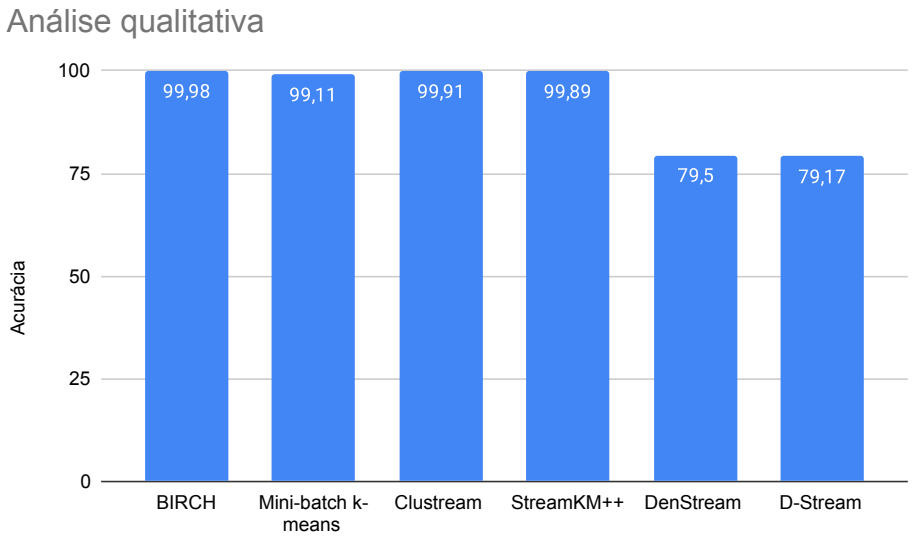
Conforme pode ser visto na Tabela 15 o algoritmo *D-Stream* obteve 79,17% de acurácia e *f1-score* igual a 0,86. Além disso, obteve um *PPV* de 76,17%, um *NPV* de 100%, um *FOR* de 0%, um *FDR* de 23,82%, um *TPR* de 100%, um *TNR* de 37,66%, um *FNR* de 0% e um *FPR* de 62,33%.

Tabela 15 – Matriz confusão para o algoritmo *D-Stream*

	Predição			
$TPO = 5968$	Positivos	Negativos		
$AP = 3974$	3974	0	$TPR = 100\%$	$FNR = 0\%$
$AN = 1994$	1243	751	$FPR = 62,33\%$	$TNR = 37,66\%$
$ACC = 79,17\%$	$PPV = 76,17\%$	$FOR = 0\%$	$F1 = 0,86$	
	$FDR = 23,82\%$	$NPV = 100\%$		

Os algoritmos *BIRCH*, *Mini-batch k-means*, *Clustream* e *StreamKM++* alcançaram um ótimo desempenho com relação a acurácia, 99,98%, 99,11%, 99,91% e 99,89%, respectivamente, conforme pode ser visto na Figura 19.

Figura 19 – Comparativo de acurácias.



Fonte: Próprio autor.

Dessa maneira, todos os algoritmos se mostraram bastante adequados para este tipo de problema e este cenário. Porém, considerando a margem de erro calculada de 1,5%, pode-se concluir que estes algoritmos possuem acurácia similar. Já os algoritmos *DenStream* e *D-Stream* obtiveram um resultado menos expresivo, 79,05% e 79,17%, respectivamente. Por isso, estes algoritmos se mostram menos adequados para este tipo de problema e este cenário, porém, possuem valores ainda relevantes que se aproximam de 80% de acurácia.

Além disso, os algoritmos *BIRCH*, *Mini-batch k-means*, *Clustream* e *StreamKM++* obtiveram bons valores para *TPR*, *TNR*, *PPV*, *NPV* e *f1-score*, conforme pode ser visto na Figura 20. Considerando a margem de erro calculada de 1,5%, concluí-se que eles possuem resultados similares, com exceção do algoritmo *Mini-batch k-means*.

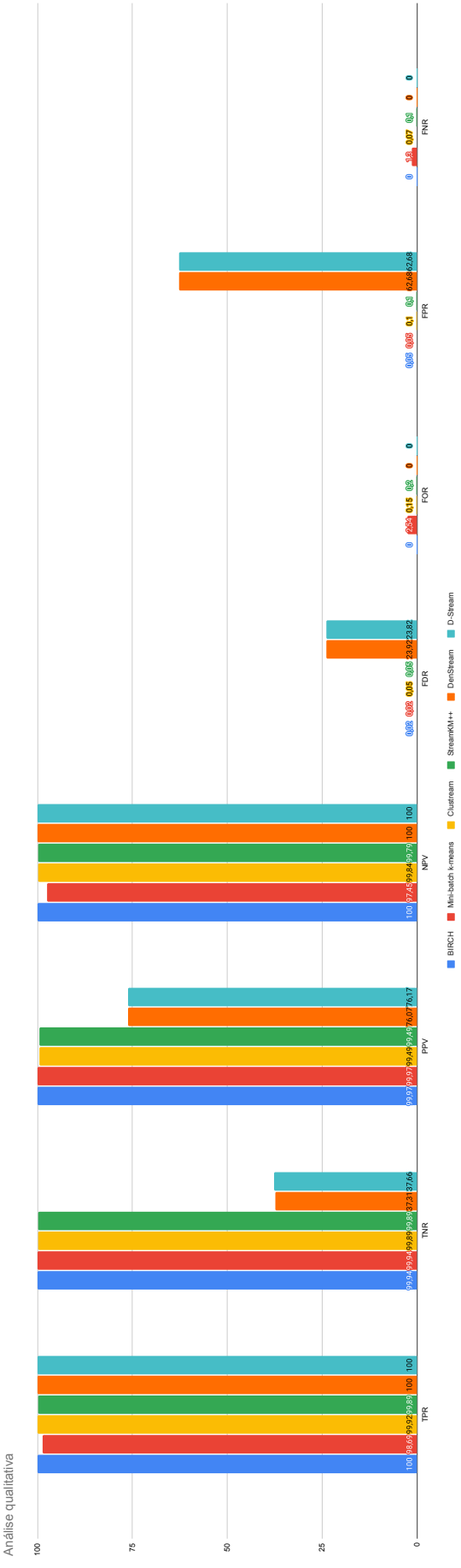


Figura 20 – Comparativo de métricas qualitativas.

Fonte: Próprio autor.

Dessa maneira, também concluí-se que todos possuem uma alta capacidade de caracterizar corretamente os dados tanto como normais quanto como de ataque. Porém, o algoritmo *Mini-batch k-means* obteve *NPV* igual a 97,45% e *FOR* igual a 2,54%, valores que ultrapassam a margem de erro. Isso denota que ele possui uma capacidade inferior de caracterizar as amostras de ataque corretamente. Por fim, os algoritmos *DenStream* e *D-Stream* obtiveram valores altos de falsos positivos, confirmados com piores valores para *FDR*, *FPR* e *TNR*, o que denota uma capacidade menor de caracterizar as amostras de ataque corretamente.

6.3.2 Análise quantitativa

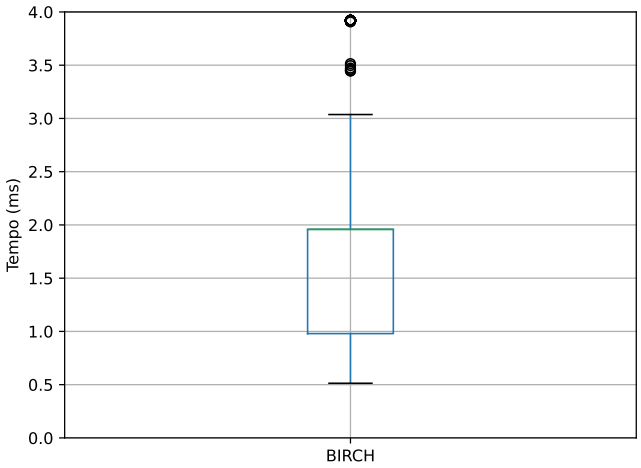
A análise quantitativa visa comparar a velocidade de processamento dos algoritmos na detecção de ataques *DDoS*. Serão analisadas duas vertentes: a velocidade de treinamento de cada amostra, ou seja, a velocidade da fase on-line dos algoritmos e a velocidade de avaliação de cada amostra, ou seja, a velocidade da fase off-line. Para esta experimentação, foi utilizada a plataforma e a arquitetura descrita no Capítulo 5 e foram criados 15 contêineres de *hosts* sendo 10 normais gerando tráfego normal, dentre eles 1 vítima e 5 infectados gerando tráfego de ataque.

Dessa maneira, considerando os 6 algoritmos e que para cada um deles foram utilizadas 53.706 amostras para a fase on-line (treinamento) e 5.968 amostras para a fase off-line (avaliação), temos um total de 358.044 amostras analisadas. Segundo Jain (1990), a partir da Equação 6.3 é possível se obter o erro amostral, onde r é o erro amostral, z é o valor crítico de confiança, s é o desvio padrão, $\bar{x}$ é a média e n o espaço amostral. Assim, obteu-se média $\bar{x}$ igual a $1,01ms$ e um desvio-padrão s igual a $2,31ms$. A partir disso, para um nível de confiança de 99%, obtêm-se o erro amostral $r = \frac{100 \cdot 2,575 \cdot 2,31}{1,01 \cdot \sqrt{358.044}} \approx 0,54\%$.

$$r = \frac{100 \cdot Z \cdot s}{\bar{x} \cdot \sqrt{n}} \tag{6.3}$$

A mediana do tempo de treinamento para o algoritmo *BIRCH* foi $1,9ms$, conforme pode ser visto na Figura 21.

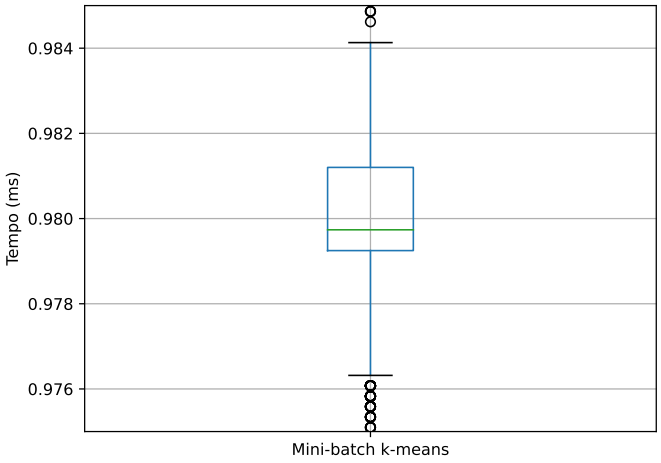
Figura 21 – *Box plot* do tempo de treinamento para o algoritmo *BIRCH*.



Fonte: Próprio autor.

A mediana do tempo de treinamento para o algoritmo *Mini-batch k-means* foi aproximadamente $0,98ms$, conforme pode ser visto na Figura 22.

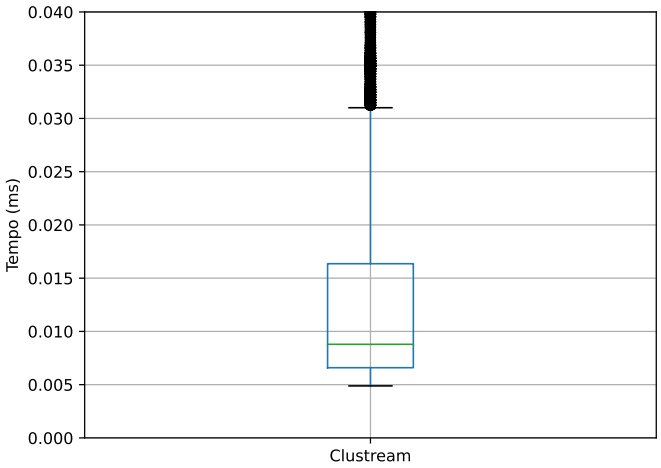
Figura 22 – *Box plot* do tempo de treinamento para o algoritmo *Mini-batch k-means*.



Fonte: Próprio autor.

A mediana do tempo de treinamento para o algoritmo *Clustream* foi $0,09ms$, conforme pode ser visto na Figura 23.

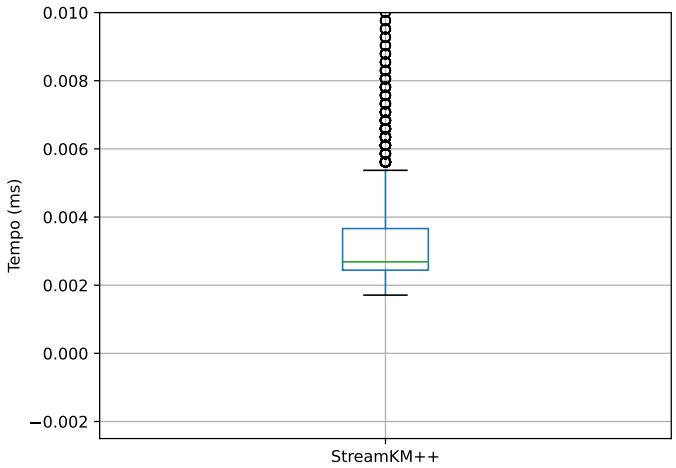
Figura 23 – *Box plot* do tempo de treinamento para o algoritmo *Clustream*.



Fonte: Próprio autor.

A mediana do tempo de treinamento para o algoritmo *StreamKM++* foi aproximadamente $0,003ms$, conforme pode ser visto na Figura 24.

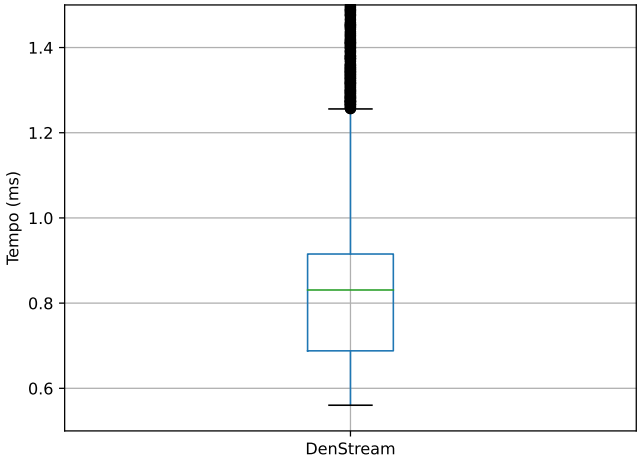
Figura 24 – *Box plot* do tempo de treinamento para o algoritmo *StreamKM++*.



Fonte: Próprio autor.

A mediana do tempo de treinamento para o algoritmo *DenStream* foi $0,82ms$, conforme pode ser visto na Figura 25.

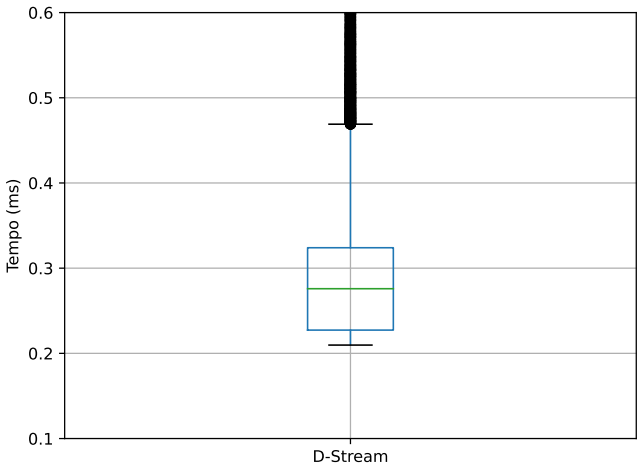
Figura 25 – *Box plot* do tempo de treinamento para o algoritmo *DenStream*.



Fonte: Próprio autor.

A mediana do tempo de treinamento para o algoritmo *D-Stream* foi $0,28ms$, conforme pode ser visto na Figura 26.

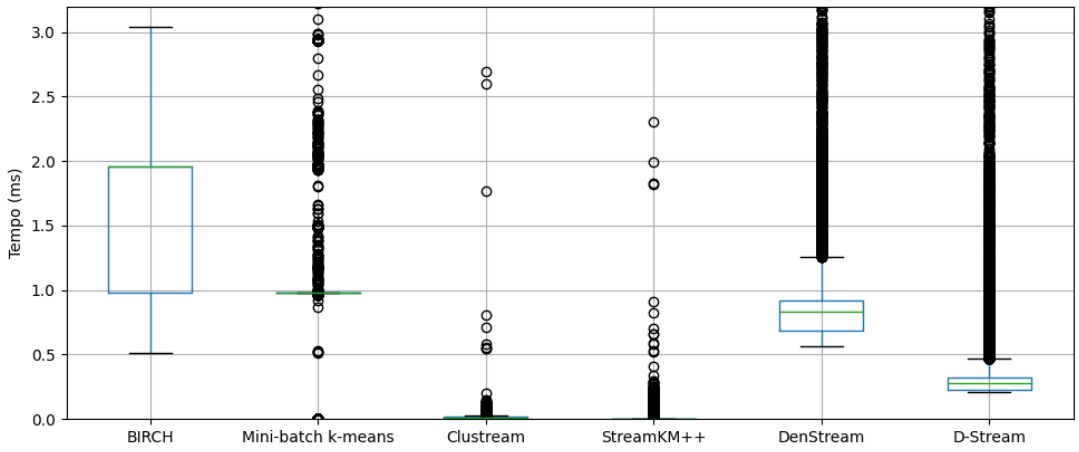
Figura 26 – *Box plot* do tempo de treinamento para o algoritmo *D-Stream*.



Fonte: Próprio autor.

Dentre eles, o algoritmo que obteve o pior tempo de treinamento foi o algoritmo *BIRCH* com o valor de $1,9ms$, conforme pode ser visto na Figura 27. O segundo pior tempo de treinamento foi do algoritmo *Mini-batch k-means*. O tempo de treinamento do *BIRCH* foi 193,87% maior que o do *Mini-batch k-means*. Por sua vez, o algoritmo com o menor tempo de treinamento foi o *StreamKM++*, seguido do *Clustream*. Enquanto o *StreamKM++* utiliza a estrutura de dados *coreset tree*, o *Clustream* utiliza a estrutura de dados *feature vector*. O tempo de treinamento do *Mini-batch k-means* foi 32.366,66% maior que o do *StreamKM++*, enquanto o do *Clustream* foi 3.000% maior que o do *StreamKM++*. Por fim, o tempo de treinamento do *DenStream* foi 27.333,33% e o do *D-Stream* foi 9.333,33% maior que o do *StreamKM++*. Além disso, o *DenStream* utiliza a estrutura de dados *grid*, enquanto o *D-Stream* utiliza a estrutura de dados *feature vector*.

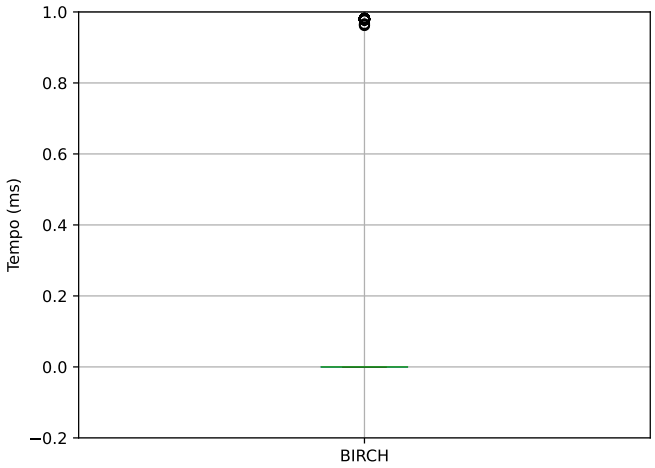
Figura 27 – Comparativo de velocidades de treinamento.



Fonte: Próprio autor.

A mediana do tempo de avaliação para o algoritmo *BIRCH* foi aproximadamente $0,0ms$, conforme pode ser visto na Figura 28.

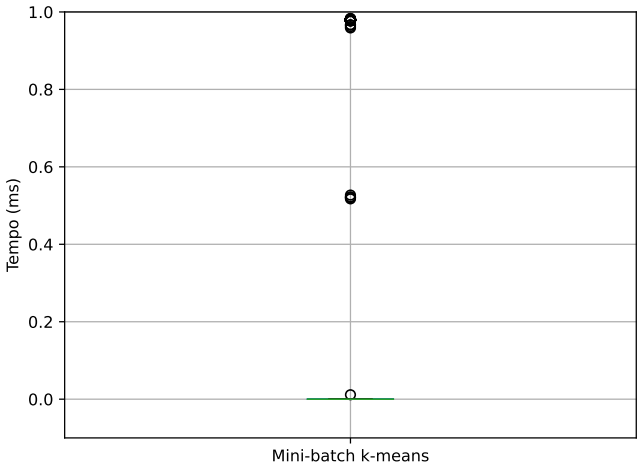
Figura 28 – *Box plot* do tempo de avaliação para o algoritmo *BIRCH*.



Fonte: Próprio autor.

A mediana do tempo de avaliação para o algoritmo *Mini-batch k-means* foi aproximadamente $0,0ms$, conforme pode ser visto na Figura 29.

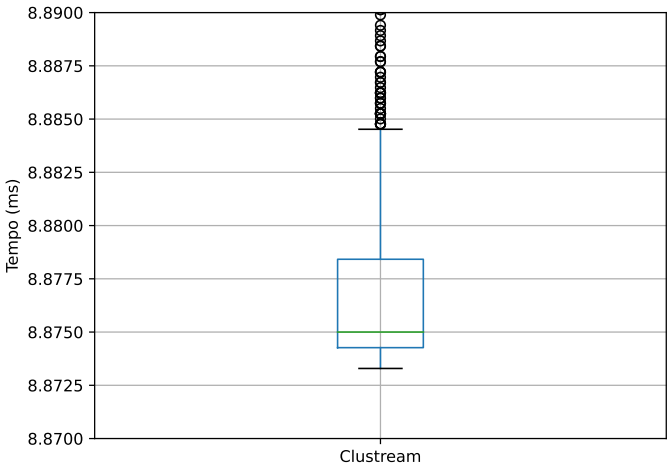
Figura 29 – *Box plot* do tempo de avaliação para o algoritmo *Mini-batch k-means*.



Fonte: Próprio autor.

A mediana do tempo de avaliação para o algoritmo *Clustream* foi $8,87ms$, conforme pode ser visto na Figura 30.

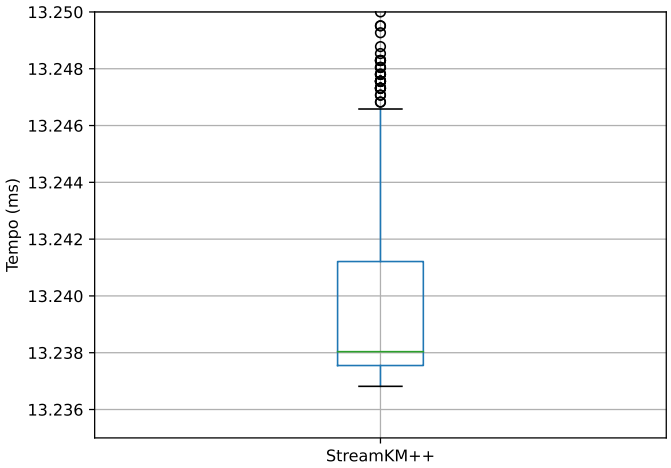
Figura 30 – *Box plot* do tempo de avaliação para o algoritmo *Clustream*.



Fonte: Próprio autor.

A mediana do tempo de avaliação para o algoritmo *StreamKM++* foi $13,23ms$, conforme pode ser visto na Figura 31.

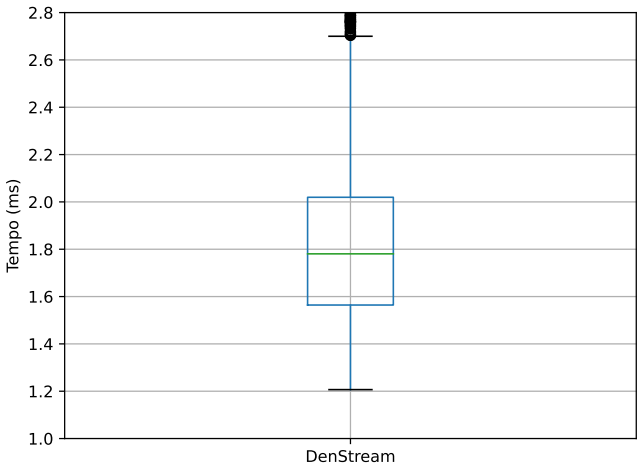
Figura 31 – *Box plot* do tempo de avaliação para o algoritmo *StreamKM++*.



Fonte: Próprio autor.

A mediana do tempo de avaliação para o algoritmo *DenStream* foi $1,79ms$, conforme pode ser visto na Figura 32.

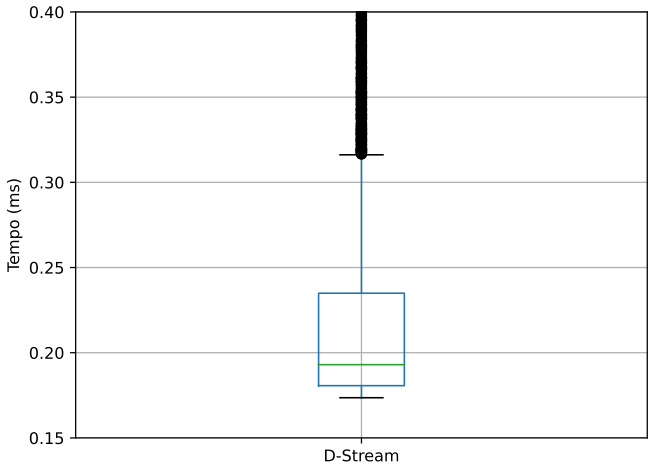
Figura 32 – *Box plot* do tempo de avaliação para o algoritmo *DenStream*.



Fonte: Próprio autor.

A mediana do tempo de avaliação para o algoritmo *D-Stream* foi 0, 19ms, conforme pode ser visto na Figura 33.

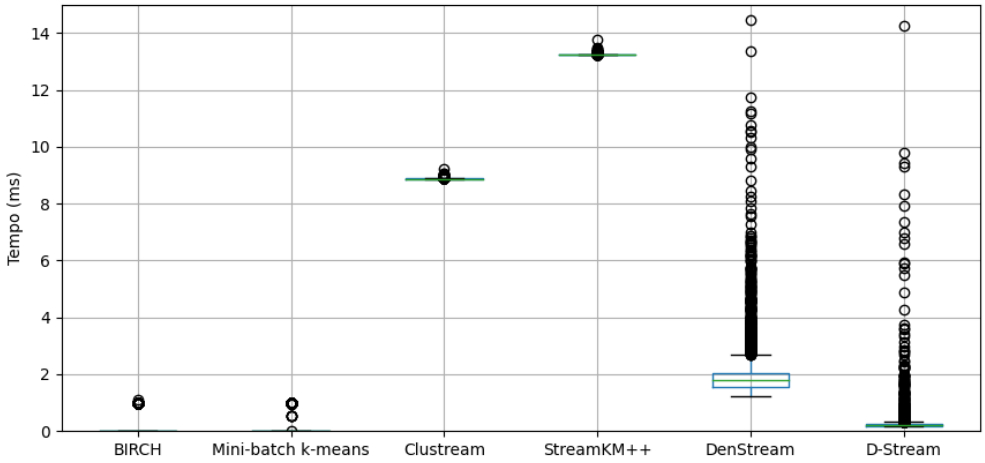
Figura 33 – *Box plot* do tempo de avaliação para o algoritmo *D-Stream*.



Fonte: Próprio autor.

O algoritmo que obteve o melhor tempo de avaliação por amostra foi o *BIRCH*, conforme pode ser visto na Figura 34. Ele utiliza o algoritmo não supervisionado chamado *Agglomerative Clustering* para realizar o agrupamento na fase off-line.

Figura 34 – Comparativo de velocidades de avaliação.



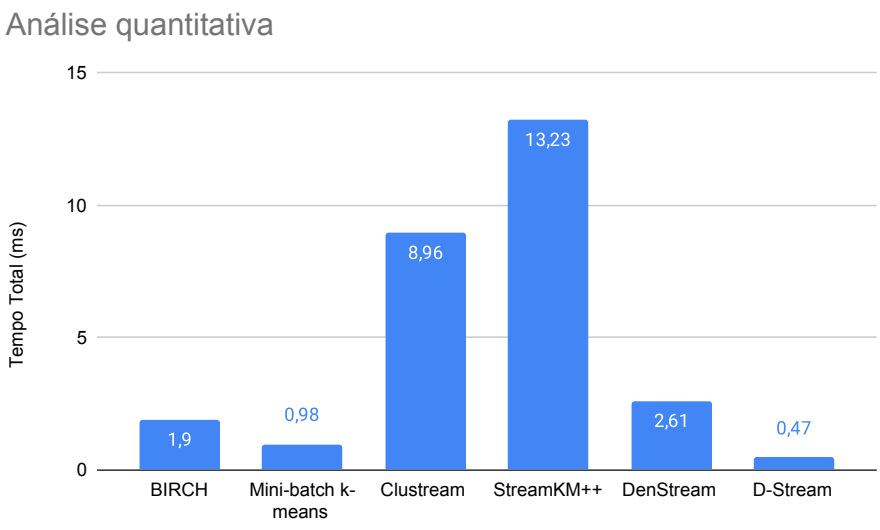
Fonte: Próprio autor.

Logo após o *BIRCH*, em empate, veio o algoritmo *Mini-batch k-means*, que utiliza o algoritmo *k-means* para realizar o agrupamento na fase off-line. Ambos algoritmos obtiveram tempo de avaliação igual a 0,0*ms*. Isso se deu, porque para o experimento foram utilizadas as implementações desses dois algoritmos disponíveis na biblioteca *scikit-learn*. O método responsável pela fase on-line chamado *partial_fit*, possui uma chamada a outro método responsável pelo agrupamento global dos *clusters* (fase off-line). Assim, no momento da predição os *clusters* já estavam formados.

Além disso, o algoritmo *StreamKM++* que obteve o pior tempo de avaliação por amostra, seguido do *Clustream*. Ambos utilizam o algoritmo *k-means++*, uma versão modificada do original, para realizar o agrupamento na fase off-line. O tempo de avaliação do *StreamKM++* foi 149,15% maior que o do *Clustream*. Por fim, os algoritmos *DenStream* e *D-Stream* utilizam o algoritmo *DBSCAN* para realizar o agrupamento na fase off-line. Eles obtiveram os melhores tempos de avaliação depois do *BIRCH* e do *Mini-batch k-means*. O tempo de avaliação do *DenStream* foi 495,53% menor que o do *Clustream*, enquanto o tempo de avaliação do *D-Stream* foi 4.668,42% menor que o do *Clustream*.

Por sua vez, avaliando o tempo total, ou seja, os tempos de treinamento e avaliação somados, o algoritmo *D-Stream* obteve o melhor desempenho com um tempo total de 0,47*ms*, como pode ver visto na Figura 35. O segundo melhor desempenho foi do algoritmo *Mini-batch k-means* com um tempo total de 0,98*ms*, que representa um aumento de 208,51% em relação ao *D-Stream*. Como também, o tempo do *BIRCH* foi de 1,9*ms*, o que representa um aumento de 193,87% em relação ao *Mini-batch k-means*. Ou seja, o algoritmo *Agglomerative Clustering* se mostrou menos performático em comparação ao *k-means*.

Figura 35 – Comparativo de velocidades totais.



Fonte: Próprio autor.

Por fim, o *DenStream* obteve um tempo total de 2,61*ms*, o que representa um aumento de 555,31% em relação ao *D-Stream*. Como ambos ambos são baseados no *DBSCAN*, a estrutura de dados *grid* utilizada pelo *D-Stream* se mostrou mais eficiente. Além disso, os dois maiores tempos totais foram dos algoritmos *Clustream* e *StreamKM++*, com tempos de 8,96*ms* e 13,23*ms*, respectivamente. Ambos utilizam o algoritmo *k-means++* e o tempo total maior pode ser explicado pelo fato que o algoritmo se baseia na distribuição das distâncias entre os pontos em relação a possíveis centróides e escolhe os melhores resultados. Ou seja, há algumas iterações realizadas no início do algoritmo, que têm o intuito de otimizar o posicionamento dos centróides, podendo resultar em uma perda maior de tempo pelo algoritmo na escolha desses centróides.

7

Conclusão

O objetivo principal deste trabalho foi analisar a eficiência e efetividade da utilização de algoritmos de aprendizagem de máquina não supervisionados na detecção de ataques *DDoS* do tipo *UDP flood* em ambientes *SDN/NFV*, por meio de uma análise comparativa. Além disso, os algoritmos deveriam trabalhar com a estratégia de fluxo de dados para se adequarem ao cenário de dinâmico presente no ambiente *SDN/NFV*. A fim de se alcançar o objetivo principal, foram definidos os seguintes objetivos específicos:

- Levantar e analisar algoritmos de aprendizagem de máquina não supervisionados que trabalhem com fluxo de dados que possam ser utilizados na detecção de ataques *DDoS*;
- Definir uma arquitetura *SDN/NFV* e que utilize microsserviços
- Implementar o módulo de detecção de ataques *DDoS*;
- Simular cenários de ataque *DDoS* no ambiente *SDN/NFV* proposto;
- Analisar e avaliar qualitativa e quantitativamente os algoritmos utilizando métricas como acurácia, *f1-score*, matriz confusão, tempo de treinamento e tempo de avaliação.

Primeiramente foi realizado um Mapeamento Sistemático da Literatura. Dessa maneira, foram incluídos os trabalhos que utilizassem a aprendizagem de máquina não supervisionada como técnica de detecção dos ataques *DDoS* e também que utilizassem o ambiente *SDN/NFV*. Além disso, a solução deveria ser baseada no algoritmo não supervisionado *k-means*. Este mapeamento foi necessário para embasar o desenvolvimento de um primeiro experimento objetivando o estudo de algoritmos que não utilizam a estratégia de fluxo de dados. Em seguida, foi realizada uma Revisão Sistemática da Literatura e como critério de seleção foram incluídos os trabalhos que utilizassem aprendizagem de máquina não supervisionada na detecção de ataques *DDoS*. Além disso, era necessário que os algoritmos trabalhassem com a estratégia de fluxo de

dados, pois, essa característica é inerente ao ambiente *SDN/NFV*. Após isso, foram levantados os algoritmos *BIRCH*, *Mini-batch k-means*, *Clustream*, *StreamKM++*, *DenStream* e *D-Stream*.

Dentre os algoritmos estudados, o *BIRCH*, o *Mini-batch k-means*, o *Clustream* e o *StreamKM++* alcançaram desempenho qualitativo alto, com acurácia em torno de 99%. Além disso, obtiveram também valores baixos de falsos positivos e falsos negativos. Já os algoritmos *DenStream* e *D-Stream* alcançaram acurácia similar em torno de 79%, porém, obtiveram um valor expressivo de falsos positivos. Essa característica pode trazer problemas para o ambiente, já que fluxos de ataque não seriam detectados corretamente.

Por outro lado, avaliando o tempo total, ou seja, os tempos de treinamento e avaliação somados, o algoritmo *D-Stream* obteve o melhor desempenho com um tempo total de 0,47*ms*. O segundo melhor desempenho foi do algoritmo *Mini-batch k-means* com um tempo total de 0,98*ms*, que representa um aumento de 208,51% em relação ao *D-Stream*. O tempo total do algoritmo *BIRCH* foi 193,87% maior que o tempo do *Mini-batch k-means*. Ou seja, o algoritmo *Agglomerative Clustering* obteve uma performance pior em comparação ao *k-means*. Já o *DenStream* alcançou um tempo total 555,31% maior que o tempo do *D-Stream*. Os dois são baseados no *DBSCAN*, porém, a estrutura de dados *grid* do *D-Stream* se mostrou mais eficiente. Além do mais, os algoritmos *Clustream* e *StreamKM++* obtiveram os maiores tempos totais. Eles utilizam o algoritmo *k-means++* e os melhoramentos de posicionamento dos centróides realizados no algoritmo original *k-means* podem ter degradado o desempenho. Portanto dos 6 algoritmos analisados, os algoritmos que se destacaram foram o *D-Stream* e o *Mini-batch k-means*. Para escolher o melhor algoritmo é necessário levar em consideração o ambiente inserido. Se as restrições de tempo de processamento forem mais importantes em detrimento da acurácia, seria mais interessante o uso do *D-Stream*, caso contrário, o *Mini-batch k-means*.

7.1 Contribuições

As principais contribuições deste trabalho foram: a definição de uma arquitetura *SDN/NFV* com microsserviços, a implementação de um módulo de detecção de ataques *DDoS* para ambientes *SDN/NFV* e apresentar a viabilidade da utilização de alguns algoritmos de aprendizagem de máquina não supervisionados que trabalham com fluxos de dados na detecção de ataques *DDoS* através de uma análise quantitativa e qualitativa. Outras contribuições deste trabalho foram, desenvolver um ambiente simulado que aplicasse as arquiteturas *SDN/NFV* e microsserviços com o intuito de validar os resultados com experimentos.

7.2 Dificuldades e limitações

Algumas dificuldades e limitações foram encontradas ao decorrer do trabalho. Primeiramente não se achou nenhum *dataset* que fosse específico do ambiente *SDN*. Por esse motivo, foi

necessário desenvolver um *dataset* sintético, ou seja, não foi possível trabalhar com dados reais.

Além disso, houve limitação no uso dos algoritmos *BIRCH* e *Mini-batch k-means*, pois, a implementação da biblioteca *scikit-learn* trazia o método de agrupamento global da fase off-line acoplado ao método da fase on-line. Isso acabou influenciando os resultados do tempo de avaliação das amostras.

Por fim, houve limitação na quantidade de contêineres on-line ao mesmo tempo, pois, o hardware disponível era um notebook pessoal. Com isso, não se conseguiu testar o ambiente com uma quantidade de dispositivos similar a de um ambiente real.

7.3 Trabalhos Futuros

Nesta seção são levantados alguns possíveis trabalhos futuros, com o intuito de dar prosseguimento a pesquisa, trazendo novos resultados ou melhorando os resultados atuais:

- Implementar os algoritmos *BIRCH* e *Mini-batch k-means* para se conseguir medir os tempos de treinamento e avaliação mais precisamente;
- Utilizar um *dataset* com dados reais ao invés de utilizar dados sintéticos;
- Implantar a arquitetura desenvolvida em um ambiente real com dados reais;
- Medir o uso de memória de todo parque computacional, incluindo o controlador *SDN*;
- Medir o uso de *CPU* de todo parque computacional, incluindo o controlador *SDN*;
- Implementar testes de carga e estresse na arquitetura desenvolvida.

7.4 Publicações

Nesta seção são apresentadas as publicações que contribuiram e estão relacionadas com a dissertação.

7.4.1 Trabalhos Submetidos

- *DDoS Detection in SDN/NFV Environment Using Unsupervised Data Stream Algorithms*. Possível publicação para a conferência *IEEE ISCC* 2021 - Qualis A3;
- *DDoS Attack Detection With Unsupervised Data Stream Algorithms in SDN/NFV Environments*. Possível publicação para a revista *IEEE Transactions on Network and Service Management* - Qualis A2.

7.4.2 Trabalhos Publicados

- *Comparative Analysis Between the K-means and Fuzzy c-means Algorithms to Detect UDP Flood DDoS Attack on a SDN/NFV Environment* - In Proceedings of the 16th International Conference on Web Information Systems and Technologies - Volume 1: Novembro 4, 2020, ISBN 978-989-758-478-7, p. 105-112. DOI: 10.5220/0010176201050112 - Qualis A4.

Referências

ACKERMANN, M. R. et al. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, ACM New York, NY, USA, v. 17, p. 2–1, 2012. Citado 2 vezes nas páginas 44 e 45.

AGGARWAL, C. C. et al. A framework for clustering evolving data streams. In: ELSEVIER. *Proceedings 2003 VLDB conference*. [S.l.], 2003. p. 81–92. Citado na página 44.

ALSAEEDI, M.; MOHAMAD, M. M.; AL-ROUBAIEY, A. A. Toward adaptive and scalable openflow-sdn flow control: A survey. *IEEE Access*, IEEE, v. 7, p. 107346–107379, 2019. Citado na página 12.

ALSHUQAYRAN, N.; ALI, N.; EVANS, R. A systematic mapping study in microservice architecture. In: IEEE. *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. [S.l.], 2016. p. 44–51. Citado na página 20.

AQIL, A. et al. Jaal: Towards network intrusion detection at isp scale. In: *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. [S.l.: s.n.], 2017. p. 134–146. Citado 3 vezes nas páginas 31, 33 e 34.

ARTHUR, D.; VASSILVITSKII, S. *k-means++: The advantages of careful seeding*. [S.l.], 2006. Citado na página 44.

ATTAK, H. et al. Shield: Securing against intruders and other threats through an nfv-enabled environment. In: *Guide to Security in SDN and NFV*. [S.l.]: Springer, 2017. p. 197–225. Citado 3 vezes nas páginas 32, 33 e 34.

BARNETT JAIN, A. K. *Cisco Visual Networking Index (VNI). Complete Forecast Update, 2017–2022*. [S.l.], 2018. Disponível em: <https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/1213-business-services-ckn.pdf>. Acesso em: 01 jul. 2020. Citado na página 13.

BAWANY, N. Z.; SHAMSI, J. A.; SALAH, K. Ddos attack detection and mitigation using sdn: methods, practices, and solutions. *Arabian Journal for Science and Engineering*, Springer, v. 42, n. 2, p. 425–441, 2017. Citado na página 12.

BEZDEK, J. C.; EHRLICH, R.; FULL, W. Fcm: The fuzzy c-means clustering algorithm. *Computers & geosciences*, Elsevier, v. 10, n. 2-3, p. 191–203, 1984. Citado na página 26.

BHUSHAN, K.; GUPTA, B. B. Distributed denial of service (ddos) attack mitigation in software defined network (sdn)-based cloud computing environment. *Journal of Ambient Intelligence and Humanized Computing*, Springer, v. 10, n. 5, p. 1985–1997, 2019. Citado na página 14.

BILGE, L.; DUMITRAŞ, T. Before we knew it: an empirical study of zero-day attacks in the real world. In: *Proceedings of the 2012 ACM conference on Computer and communications security*. [S.l.: s.n.], 2012. p. 833–844. Citado na página 24.

BONFIM, M. S.; DIAS, K. L.; FERNANDES, S. F. Integrated nfv/sdn architectures: A systematic literature review. *ACM Computing Surveys (CSUR)*, v. 51, n. 6, p. 1–39, 2019. Citado 4 vezes nas páginas 14, 18, 19 e 50.

BOTTOU, L.; BENGIO, Y. Convergence properties of the k-means algorithms. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1995. p. 585–592. Citado na página 43.

CAMPBELL, S.; KIRAN, M.; WALA, F. B. Unsupervised anomaly detection in daily wan traffic patterns. In: SPRINGER. *Smoky Mountains Computational Sciences and Engineering Conference*. [S.l.], 2020. p. 240–256. Citado 3 vezes nas páginas 37, 39 e 40.

CAO, F. et al. Density-based clustering over an evolving data stream with noise. In: SIAM. *Proceedings of the 2006 SIAM international conference on data mining*. [S.l.], 2006. p. 328–339. Citado na página 45.

CHEN, Y.; TU, L. Density-based clustering for real-time stream data. In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2007. p. 133–142. Citado 2 vezes nas páginas 46 e 47.

CHODOROW, K. *MongoDB: the definitive guide: powerful and scalable data storage*. [S.l.]: "O'Reilly Media, Inc.", 2013. Citado na página 60.

CRANE, C. *The 15 Top DDoS Statistics You Should Know In 2020*. [S.l.], 2019. Disponível em: <https://cybersecurityventures.com/the-15-top-ddos-statistics-you-should-know-in-2020/>. Acesso em: 06 jun. 2020. Citado na página 13.

DIAS, M. L. D. *fuzzy-c-means: An implementation of Fuzzy C-means clustering algorithm.* 2019. Disponível em: <https://github.com/omadson/fuzzy-c-means>. Citado na página 55.

ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Kdd*. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231. Citado na página 46.

ETSI. *Network functions virtualisation (NFV)—Architectural framework*. [S.l.]: ETSI GS NFV, 2014. v. 002 V1.2.1. Citado 2 vezes nas páginas 19 e 20.

ETSI, N. *Network functions virtualisation (NFV); Network Operator Perspectives on NFV priorities for 5G*. [S.l.]: Technical report. http://portal.etsi.org/NFV/NFV_White_Paper_5G.pdf, 2017. Citado na página 13.

EVANS, M. J.; ROSENTHAL, J. S. *Probability and statistics: The science of uncertainty*. [S.l.]: W.H. Freeman Company, 2009. ISBN 978-1429224628. Citado na página 61.

FRAHLING, G.; SOHLER, C. Coresets in dynamic geometric data streams. In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. [S.l.: s.n.], 2005. p. 209–217. Citado na página 45.

GHESMOUNE, M.; LEBBAH, M.; AZZAG, H. State-of-the-art on clustering data streams. *Big Data Analytics*, BioMed Central, v. 1, n. 1, p. 1–27, 2016. Citado na página 41.

GHOSH, S.; DUBEY, S. K. Comparative analysis of k-means and fuzzy c-means algorithms. *International Journal of Advanced Computer Science and Applications*, Citeseer, v. 4, n. 4, 2013. Citado na página 25.

GUERRIERI, A.; MONTRESOR, A. Ds-means: distributed data stream clustering. In: SPRINGER. *European Conference on Parallel Processing*. [S.l.], 2012. p. 260–271. Citado 3 vezes nas páginas 37, 39 e 40.

GUHA, S. et al. Clustering data streams: Theory and practice. *IEEE transactions on knowledge and data engineering*, IEEE, v. 15, n. 3, p. 515–528, 2003. Citado na página 26.

GYŐRÖDI, C. et al. A comparative study: Mongodb vs. mysql. In: IEEE. *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. [S.l.], 2015. p. 1–6. Citado na página 60.

HAR-PELED, S.; MAZUMDAR, S. On coresets for k-means and k-median clustering. In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. [S.l.: s.n.], 2004. p. 291–300. Citado na página 45.

HELLER, B. Openflow switch specification. 2009. Citado na página 50.

IVANNIKOVA, E.; ZOLOTUKHIN, M.; HÄMÄLÄINEN, T. Probabilistic transition-based approach for detecting application-layer ddos attacks in encrypted software-defined networks. In: SPRINGER. *International Conference on Network and System Security*. [S.l.], 2017. p. 531–543. Citado 3 vezes nas páginas 37, 39 e 40.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: john wiley & sons, 1990. Citado na página 66.

KALKAN, K.; GÜR, G.; ALAGÖZ, F. Filtering-based defense mechanisms against ddos attacks: A survey. *IEEE Systems Journal*, IEEE, v. 11, n. 4, p. 2761–2773, 2016. Citado na página 21.

KAUR, S.; SINGH, J.; GHUMMAN, N. S. Network programmability using pox controller. In: *International Conference on Communication, Computing & Systems (ICCCN'2014)*. [S.l.: s.n.], 2014. p. 134–138. Citado na página 49.

KAUSHIK, M.; MATHUR, B. Comparative study of k-means and hierarchical clustering techniques. *International journal of software and hardware research in engineering*, v. 2, n. 6, p. 93–98, 2014. Citado na página 26.

KEELE, S. et al. *Guidelines for performing systematic literature reviews in software engineering*. [S.l.], 2007. Citado na página 28.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado na página 29.

KONING, R. et al. Interactive analysis of sdn-driven defence against distributed denial of service attacks. In: IEEE. *2016 IEEE NetSoft Conference and Workshops (NetSoft)*. [S.l.], 2016. p. 483–488. Citado na página 32.

KONING, R. et al. Measuring the efficiency of sdn mitigations against attacks on computer infrastructures. *Future Generation Computer Systems*, Elsevier, v. 91, p. 144–156, 2019. Citado 3 vezes nas páginas 32, 33 e 34.

KREUTZ, D. et al. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, Ieee, v. 103, n. 1, p. 14–76, 2014. Citado 4 vezes nas páginas 12, 13, 16 e 17.

KUMAR, S.; PETANA, E. Mitigation of tcp-syn attacks with microsoft's windows xp service pack2 (sp2) software. In: IEEE. *Seventh International Conference on Networking (icn 2008)*. [S.l.], 2008. p. 238–242. Citado na página 24.

LARA, A.; KOLASANI, A.; RAMAMURTHY, B. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, IEEE, v. 16, n. 1, p. 493–512, 2013. Citado na página 17.

LI, J.; ZHAO, Z.; LI, R. Machine learning-based ids for software-defined 5g network. *IET Networks*, IET, v. 7, n. 2, p. 53–60, 2017. Citado 2 vezes nas páginas 32 e 34.

MAHJABIN, T. et al. A survey of distributed denial-of-service attack, prevention, and mitigation techniques. *International Journal of Distributed Sensor Networks*, SAGE Publications Sage UK: London, England, v. 13, n. 12, p. 1550147717741463, 2017. Citado 4 vezes nas páginas 21, 22, 23 e 24.

MCKEOWN, N. et al. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 38, n. 2, p. 69–74, 2008. Citado na página 49.

MITCHELL, T. M. *The discipline of machine learning*. [S.l.]: Carnegie Mellon University, School of Computer Science, Machine Learning . . . , 2006. v. 9. Citado na página 25.

MOHAMMED, M.; KHAN, M. B.; BASHIER, E. B. M. *Machine learning: algorithms and applications*. [S.l.]: Crc Press, 2016. Citado 2 vezes nas páginas 24 e 25.

MOUAT, A. *Using Docker: Developing and Deploying Software with Containers*. [S.l.]: "O'Reilly Media, Inc.", 2015. Citado na página 50.

NAMIOT, D.; SNEPS-SNEPPE, M. On micro-services architecture. *International Journal of Open Information Technologies*, v. 2, n. 9, p. 24–27, 2014. Citado na página 20.

NEWMAN, S. Under the radar: the danger of stealthy ddos attacks. *Network Security*, Elsevier, v. 2019, n. 2, p. 18–19, 2019. Citado na página 13.

PATIL, N. V.; KRISHNA, C. R.; KUMAR, K. S-ddos: Apache spark based real-time ddos detection system. *Journal of Intelligent & Fuzzy Systems*, IOS Press, n. Preprint, p. 1–9, 2020. Citado 4 vezes nas páginas 36, 38, 39 e 40.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. [S.l.: s.n.], 2008. p. 1–10. Citado na página 28.

PEUSTER, M.; KARL, H.; ROSSEM, S. van. Medicine: Rapid prototyping of production-ready network services in multi-pop environments. In: *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. [S.l.: s.n.], 2016. p. 148–153. Citado na página 50.

RING, M. et al. A toolset for intrusion and insider threat detection. In: *Data Analytics and Decision Support for Cybersecurity*. [S.l.]: Springer, 2017. p. 3–31. Citado 3 vezes nas páginas 37, 39 e 40.

RUSPINI, E. H. Numerical methods for fuzzy clustering. *Information Sciences*, Elsevier, v. 2, n. 3, p. 319–350, 1970. Citado na página 26.

SARAVANAN, K. Neuro-fuzzy-based clustering of ddos attack detection in the network. *International Journal of Critical Infrastructures*, Inderscience Publishers (IEL), v. 13, n. 1, p. 46–56, 2017. Citado 4 vezes nas páginas 35, 38, 39 e 40.

SCULLEY, D. Web-scale k-means clustering. In: *Proceedings of the 19th international conference on World wide web*. [S.l.: s.n.], 2010. p. 1177–1178. Citado na página 43.

SHIRMARZ, A.; GHAFFARI, A. Performance issues and solutions in sdn-based data center: A survey. *The Journal of Supercomputing*, Springer, p. 1–49, 2020. Citado na página 18.

SILVA, J. A. et al. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 46, n. 1, p. 1–31, 2013. Citado 3 vezes nas páginas 26, 27 e 41.

SINGH, A.; JUNEJA, D. Agent based preventive measure for udp flood attack in ddos attacks. *International Journal of Engineering Science and Technology*, v. 2, n. 8, p. 3405–3411, 2010. Citado 2 vezes nas páginas 23 e 24.

ŠKRJANC, I. et al. Evolving cauchy possibilistic clustering and its application to large-scale cyberattack monitoring. In: IEEE. *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. [S.l.], 2017. p. 1–7. Citado 4 vezes nas páginas 36, 38, 39 e 40.

SOUSA, N. F. S. de et al. Network service orchestration: A survey. *Computer Communications*, Elsevier, v. 142, p. 69–94, 2019. Citado na página 19.

STRAUCH, C.; SITES, U.-L. S.; KRIHA, W. Nosql databases. *Lecture Notes, Stuttgart Media University*, v. 20, p. 24, 2011. Citado na página 60.

SUOMALAINEN, J. et al. Security awareness in software-defined multi-domain 5g networks. *Future Internet*, Multidisciplinary Digital Publishing Institute, v. 10, n. 3, p. 27, 2018. Citado 3 vezes nas páginas 32, 33 e 34.

SURESH, M.; ANITHA, R. Evaluating machine learning algorithms for detecting ddos attacks. In: SPRINGER. *International Conference on Network Security and Applications*. [S.l.], 2011. p. 441–452. Citado na página 14.

SUSMAGA, R. Confusion matrix visualization. In: *Intelligent Information Processing and Web Mining*. [S.l.]: Springer, 2004. p. 107–116. Citado na página 52.

VIDAL, J. M.; OROZCO, A. L. S.; VILLALBA, L. J. G. Adaptive artificial immune networks for mitigating dos flooding attacks. *Swarm and Evolutionary Computation*, Elsevier, v. 38, p. 94–108, 2018. Citado 3 vezes nas páginas 32, 33 e 34.

VIGNA, G.; ROBERTSON, W.; BALZAROTTI, D. Testing network-based intrusion detection signatures using mutant exploits. In: *Proceedings of the 11th ACM conference on Computer and communications security*. [S.l.: s.n.], 2004. p. 21–30. Citado na página 13.

WEI, S. et al. Stdc: A sdn-oriented two-stage ddos detection and defence system based on clustering. In: IEEE. *2018 17th IEEE International Conference on Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. [S.l.], 2018. p. 339–347. Citado 3 vezes nas páginas 31, 33 e 34.

WOLFF, E. *Microservices: flexible software architecture*. [S.l.]: Addison-Wesley Professional, 2016. Citado na página 20.

YI, B. et al. A comprehensive survey of network function virtualization. *Computer Networks*, Elsevier, v. 133, p. 212–262, 2018. Citado na página 14.

YUAN, R. et al. An svm-based machine learning method for accurate internet traffic classification. *Information Systems Frontiers*, Springer, v. 12, n. 2, p. 149–156, 2010. Citado na página 13.

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, ACM New York, NY, USA, v. 25, n. 2, p. 103–114, 1996. Citado 2 vezes nas páginas 42 e 43.

ZHOU, B. et al. Machine-learning-based online distributed denial-of-service attack detection using spark streaming. In: IEEE. *2018 IEEE International Conference on Communications (ICC)*. [S.l.], 2018. p. 1–6. Citado 4 vezes nas páginas 37, 38, 39 e 40.

ZOLOTUKHIN, M.; HÄMÄLÄINEN, T. Data stream clustering for application-layer ddos detection in encrypted traffic. In: *Cyber Security: Power and Technology*. [S.l.]: Springer, 2018. p. 111–131. Citado 4 vezes nas páginas 37, 38, 39 e 40.

ZOLOTUKHIN, M. et al. Weighted fuzzy clustering for online detection of application ddos attacks in encrypted network traffic. In: *Internet of things, smart spaces, and next generation networks and systems*. [S.l.]: Springer, 2016. p. 326–338. Citado 4 vezes nas páginas 36, 38, 39 e 40.