UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uso de algoritmo de aprendizagem de máquina não-supervisionado para prevenção da formação de redes *botnet*.

Dissertação de Mestrado

Gabriel de Carvalho Arimatéa

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gabriel de Carvalho Arimatéa

Uso de algoritmo de aprendizagem de máquina não-supervisionado para prevenção da formação de redes *botnet*.

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. Ciência da Computação.

Orientador(a): Admilson de Ribamar Lima Ribeiro

São Cristóvão – Sergipe

2021

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE

Arimatéa, Gabriel de Carvalho
A699u Uso de algorítmo de aprendizagem de máquina para prevenção da formação de redes *botnet* / Gabriel Carvalho Arimatéa ; orientador Admilson de Ribamar Lima Ribeiro. - São Cristóvão, 2021.
78 f. : il.

Dissertação (mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2021.

1. Computação. 2. Internet das coisas. 3. Inteligência artificial. 4. Algorítmos computacionais. 5. Computadores – Medidas de segurança. I. Ordonez, Edward David Moreno orient. II. Título.

CDU 004

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: GABRIEL DE CARVALHO ARIMATÉA

Em 03 dias do mês de dezembro do ano de dois mil e vinte, com início às 14h00min, realizou-se na Sala virtual https://meet.jit.si/DefesaMestradoGabrielCarvalhoArimatea2020. A Sessão Pública de Defesa de Dissertação de Mestrado do candidato **GABRIEL DE CARVALHO ARIMATÉA**, que desenvolveu o trabalho intitulado: **"Uso de algoritmo de aprendizagem de máquina não-supervisionado para prevenção da formação de redes botnet",** 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. **Jean Carlos Teixeira de Araújo** (UFAPE) , **Prof. Dr. Anderson Clayton Alves Nascimento (**University of Washington**)** e, em seguida, ao Prof. Dr. **Rubens De Souza Matos Junior** (PROCC/UFS) . Após as discussões, a Banca Examinadora reuniu-se e considerou o 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",*03 de dezembro de 2020.

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

Prof. Dr. Rubens De Souza Matos Junior
(PROCC/UFS)
Examinador Interno

Prof. Dr. Anderson Clayton Alves Nascimento
(University of Washington)
Examinador Externo à instituição

Prof. Dr. Jean Carlos Teixeira de Araújo
(UFAPE)
Examinador Externo à instituição

Gabriel de Carvalho Arimatéa
discente

Agradecimentos

Primeiramente agradeço a Deus por todas as oportunidades e pessoas boas que colocaste em minha vida. E mesmo tendo dificuldades, sempre me apareceram formas de contornar, além de força e coragem para superar os obstáculos.

Gostaria de agradecer a toda minha família (pai, mãe e irmã) por serem minha base e inspiração para ter conseguido alcançar tudo que alcancei até hoje e que conseguiremos alcançar daqui pra frente. Tudo que conquistei até hoje, foi graças a vocês. Eu amo vocês.

A Verena, você foi essencial para mim desde que entrou na minha vida, seja me apoiando, debatendo assuntos do meu mestrado que você se esforçou pra entender (mesmo sendo totalmente fora da sua área). Obrigado também por recarregar minhas energias sempre que eu precisava. Espero que continuemos sendo esses parceiros imbatíveis. Te amo.

Ao professor Admilson, que me orientou e acreditou na proposta, sempre dando sugestões de como melhorar o trabalho e sempre disposto a ajudar, muito obrigado por tudo. Aos professores Anderson, Jean e Rubens agradeço muito por terem aceitado participar da banca. E não só isso, por terem dado sugestões e críticas pensando sempre no melhor para este trabalho.

Por fim, aos colegas de mestrado (Hugo Menezes, João Ribeiro e Caio Meneses) que me ajudaram nesses dois anos de luta e aos meus amigos que de alguma forma, foram essenciais para eu concluir esse trabalho.

Muito Obrigado!

Resumo

A *Internet* das Coisas tem se tornado cada vez mais importante por sua aplicabilidade em vários ecossistemas embarcados do cotidiano. Entretanto, os dispositivos destes sistemas apresentam várias restrições de *hardware* e sua segurança vem sendo negligenciada. Consequentemente, *malwares* formadores de *botnets* tem aproveitado os fracos esquemas de segurança nestes dispositivos. Esta dissertação avalia o uso de quatro algoritmos não-supervisionados que utilizam *data stream* para detectar a formação de *botnets* na borda da rede. Os algoritmos foram escolhidos após revisão da literatura por serem mais leves e, portanto, considerados mais adequados para a implantação em cenários com maiores restrições. Foram utilizados algoritmos de pré-processamento para melhorar a eficiência e a qualidade dos resultados. Foi também utilizado um conjunto de dados que considerou o fluxo de dados gerados por nove dispositivos inteligentes e com duas variantes de *malwares*: *Mirai* e *Bashlite*. Foram realizados testes qualitativos para validar o resultado das classificações de cada algoritmo, além de resultados referentes a variações de processadores e memória para verificar qual o perfil mínimo de dispositivo necessário para executar de forma adequada. Após avaliações qualitativas e de performance, os resultados obtidos mostram que algoritmos como *BIRCH*, *DenStream* e *DStream* são opções viáveis para detectar dados maliciosos que trafegam na formação da *botnet*. Tendo acurácias médias entre 96% e 98%, necessitando de poucas amostras por dispositivos e tempo de análise de amostras de 300 milissegundos em um *Raspberry Pi Zero W*, sendo um dispositivo com menor capacidade computacional e mais próximo de uma aplicação em um cenário da *Internet* das Coisas.

Palavras-chave: Internet das Coisas. *Botnet*. Aprendizagem de Máquina. Segurança.

Abstract

The Internet of Things has become more important due to its applicability to many embedded systems ecosystems in daily use. However, those systems' devices have several hardware constraints and neglected security. Consequently, botnets malwares have taken advantage of poor security schemas on such devices. This dissertation evaluates the use of four unsupervised machine learning algorithms using data streams to detect botnet formation on the network edge. The algorithms were chosen after a literature review for being less demanding, being more adequate to implement in more restricted environments. To increase the efficiency and quality of results, two processing algorithms were also used. It was used a dataset generated by nine smart objects and with two infection variants: Mirai and Bashlite. Qualitative experiments were made to assess the classification results of each algorithm and also to evaluate the results after varying processing and memory resources changes to verify a minimal configuration to a device properly execute the algorithms. After qualitative and performance evaluations, the results showed that algorithms such as BIRCH, DenStream, and DStream are viable choices to detect malicious data that are sent in botnet formation. Those algorithms have an average accuracy between 96% and 98%, needing few samples per device and sample analysis response time of 300 milliseconds in a Raspberry Pi Zero W, being a constrained device and much similar to an application in an Internet of Things scenario.

Keywords: Internet of Things. Botnet. Machine Learning. Security.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AMWR	*Adaptive Moving Window Regression*
AUC	*Area Under de Curve*
BIRCH	*Balanced Iterative Reducing and Clustering using Hierarchies*
CEP	Processador de Evento Complexo
CFG	*Control Flow Graph*
CFT	*Cluster Features Tree*
CF	*Cluster Features*
CMC	*Core-Micro-Clusters*
CNN	*Convolutional Neural Network*
DBScan	*Density-based Spatial Clustering of Applications With Noise*
DDR	*Double Data Rate*
DDoS	*Distributed Denial of Service*
DNS	*Domain Name System*
DVR	*Digital Video Recorder*
FCBF	*Fast Correlation-Based Filter*
FDR	Taxa de Falsa Descoberta
FNR	Taxa de Falsos Negativos
FOR	Taxa de Falsa Omissão
FPGA	*Field-Programmable Gate Array*
GPU	*Unidade de Processamento Gráfico*
IDC	*International Data Corporation*
IoT	*Internet of Things*
LGC	*Local and Globle Consistency*
LOF	*Local Outlier Factor*

NNTree	*Neural Network Tree*
NPV	Valor Preditivo Negativo
PCA	*Principal Component Analysis*
PPV	Valor Preditivo Positivo
PUF	Funções Fisicamente NãoClonáveis
RAM	*Random Access Memory*
S-MAB	*Sequential Multi-Armed Bandits*
SDN	*Software Defined Networks*
SDWN	*Software Defned Wireless Network*
SOC	*System On Chip*
SVM	*Support Vector Machine*
TF-IDF	Frequência do Termo–Inverso Da Frequência nos Documentos
TNR	Taxa de Verdadeiros Negativos
TNR	Taxa de Verdadeiros Negativos
TPR	Taxa de Verdadeiros Positivos
VAT	*Visual Assessment of Tendency*
WSN	*Wireless Sensor Network*

Sumário

1 Introdução

1.1 Apresentação Geral

A Internet das Coisas (ou *IoT*) é uma subárea que está em crescimento devido a sua natureza e utilização. Para alguns autores, a Internet das Coisas consiste em redes de objetos inteligentes, sendo eles sensores, atuadores, celulares, entre outros que interagem e cooperam entre si por um endereçamento único para um determinado propósito (ATZORI; IERA; MORABITO, 2010). Este pode ser automação residencial, industrial, cidades inteligentes, entre outros.

Com o surgimento da *IoT* e sua proposta de ubiquidade, a maneira como as redes tradicionais atuam não atendem de forma adequada a nova realidade. Isso se dá pela dificuldade na manutenção da qualidade associada com o enorme fluxo de dados. Desta maneira, conceitos como *Edge*, *Fog* e *Cloud Computing* são utilizados para estruturar a rede de forma mais adequada onde a *IoT* será inserida (SHI et al., 2016).

Porém, com o crescimento rápido da *IoT*, questões quanto a segurança dos dispositivos e da rede a qual estes fazem parte começaram a ser levantadas. Uma das ameaças que já foram utilizadas são os ataques de negação (*DDoS*) através da formação de *botnets*, como o ataque *Mirai* e *Bashlite* levantados em outros trabalhos (MARZANO et al., 2018; KAMBOURAKIS; KOLIAS; STAVROU, 2017). Este tipo de abordagem aproveita da grande quantidade de objetos inteligentes e menor segurança destes em uma rede embarcada, e faz com que isso se torne uma *botnet*, rede de dispositivos que serve como uma plataforma para promoção ataques de negação de serviço.

1.2 Motivação

Há uma estimativa que até o final de 2020 existirão mais de 24 bilhões de dispositivos inteligentes conectados, espalhados em áreas como domótica, transporte, medicina, dentre outros,

causando um impacto econômico de trilhões de dólares (GUBBI et al., 2013). Uma previsão da importância da *IoT* até 2020 apontada por Mario Morales da *International Data Corporation* (*IDC*) pode ser visto na Figura 1.

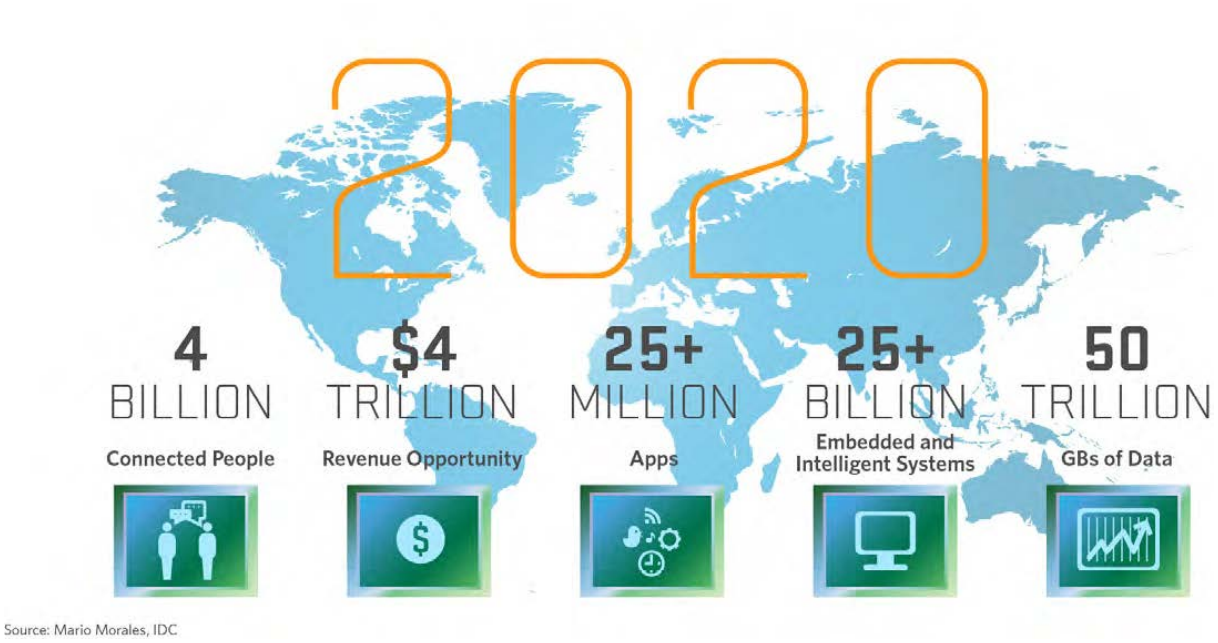


Figura 1 – Importância da Internet das Coisas em 2020.

Fonte: Mario Morales, IDC.

A contribuição da *IoT* para a sociedade, devido ao seu espalhamento em diversas áreas, irá permitir uma vida mais cômoda, prática e segura. As redes embarcadas permeiam por diversas áreas, como setor elétrico (*smart grids*), automação (industrial e residencial), cidades inteligentes, saúde (monitoramento remoto), entre outros (VASSEUR; DUNKELS, 2012).

Grandes empresas tem criado assistentes de voz, como o *Google Assistant* pela *Google* e *Alexa* pela *Amazon*. Com isso, mais objetos inteligentes para automação residencial são criados a cada ano para interagir com essas assistentes. Um estudo disponibilizado pela Ilumeo[1] em 2020 mostra que:

- 48% da população já utiliza algum assistente virtual;
- 47% da população usa algum aparelho controlado por voz;
- 61% da população gostaria de usar mais aparelhos eletrônicos controlado por voz.

[1] Disponível em: <https://ilumeo.com.br/assistentes-de-voz>

1.3 Justificativa

Apesar da motivação levantada na seção 1.2, com esse crescimento, vem a preocupação com a segurança dos dispositivos. Pois, devido a sua ubiquidade, caso não haja um nível de segurança aceitável, haverão diversas formas de ataques que impactarão em áreas vitais da sociedade, além do risco de danificar os dispositivos e a própria rede.

Uma das preocupações levantadas é nas formações das redes *botnet*. Estas tem como objetivo formar uma rede de dispositivos que servem de plataforma para ataques que podem ser *spamming*, roubo de credenciais bancárias, *click fraud*, *DDoS*, entre outros (DIETZ et al., 2018).

Em uma busca bibliográfica previamente realizada, não foram encontrados muitos trabalhos que estejam preparados para um cenário dinâmico. Assim, não há indícios de que esteja sendo considerada a evolução dos ataques. Dessa forma, em geral, vem sendo utilizadas abordagens tradicionais que exigem supervisão externa. Aliado a isso, a maior parte dos trabalhos vem se desenvolvendo para cenários não muito próximos ao dos ambientes com menor capacidade computacional. Muitos deles ainda colocam a segurança para a *cloud*, o que aumenta o tempo de resposta para ameaças a rede.

1.4 Objetivos

1.4.1 Objetivo Geral

O objetivo deste trabalho é verificar a viabilidade do uso de algoritmos de aprendizagem de máquina não-supervisionado voltado para ambientes com menor capacidade computacional. O algoritmo idealmente deve possuir menor demanda computacional e que trabalhe com fluxo de dados da rede (*data streams*), para se adequar ao cenário restrito das redes embarcadas. Este também deverá identificar o início de formações de redes *botnets*.

1.4.2 Objetivos Específicos

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

- Levantar algoritmos leves de aprendizagem de máquina não-supervisionados que podem ser utilizados para identificar formação de redes *botnets*;
- Verificar, por experimentação em ambientes controlados, a avaliação do algoritmo em cenários existentes de ataque utilizando medidas como acurácia, tempo de resposta e adaptação a novos cenários;
- Apurar o perfil mínimo necessário para que o dispositivo dedicado consiga executar o algoritmo.

1.5 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está estruturado em capítulos e seções, 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, com a revisão de literatura relacionada ao tema proposto;
- Capítulo 3 - Trabalhos relacionados: demonstra os trabalhos correlatos com a revisão de literatura adotada (Revisão Sistemática), bem como são apresentados os resultados dessa revisão e o processo de sumarização dos trabalhos estudados;
- Capítulo 4 - Algoritmos de aprendizagem de máquina avaliados: trata sobre os algoritmos de aprendizagem, além de demonstrar o funcionamento dos algoritmos adotados nas experimentações;
- Capítulo 5 - Metodologia: apresenta a metodologia utilizada para as experimentações, como *dataset*, arquitetura utilizada e algoritmos de pré-processamento;
- Capítulo 6 - Experimentação e análise qualitativa: exposição dos resultados obtidos na experimentação qualitativa e a avaliação dos resultados;
- Capítulo 7 - Experimentação e análise de desempenho: exposição dos resultados obtidos na experimentação referente a performance e a avaliação dos resultados;
- Capítulo 8 - Conclusão: conclusão do trabalho, avaliando se há aplicabilidade e pontos observados.

2

Fundamentação Teórica

Neste capítulo são levantados os paradigmas e características dos temas abordados. Para tanto, foi consultada a literatura referente ao tema. Dessa forma, foi possível o conhecimento inicial necessário para este trabalho, no que diz respeito a redes para sistemas embarcados, *edge computing*, *botnet* e aprendizagem de máquina.

2.1 Redes para Sistemas Embarcados

Existem diversas definições para um sistema embarcado. Para Vasseur e Dunkels (2012) qualquer sistema que possua um microprocessador é considerado um sistema embarcado exceto computadores pessoais (*desktops*), *laptops*, ou qualquer outro equipamento facilmente identificado como um computador. Já White (2012) define como um sistema computadorizado construído para uma aplicação específica. Entretanto, pode ser resumido em características de funcionalidade única, recursos limitados, reativos e de tempo real (VAHID; GIVARGIS, 2010).

Os sistemas embarcados quando tem seu uso voltado para o cotidiano (também conhecido como computação pervasiva), permite que pessoas e sistemas interajam entre si de forma mais facilitada e de forma independente, trazendo mais conforto para a população. As áreas do cotidiano podem variar entre várias, como automotiva ou aviação, saúde, produtos eletrônicos e *Smart Homes* e telecomunicações (FRIEDRICH, 2009).

As redes de sistemas embarcados debatidas neste trabalho são as redes de sensores sem fio (*Wireless Sensor Network*, ou *WSN*) e a Internet das Coisas.

2.1.1 *Wireless Sensor Network*

As *WSN* consistem em uma rede de dispositivos que contém sensores, onde estes percebem o ambiente. Tais dispositivos estão interconectados através de uma rede, permitindo a transmissão da informação para que seja alcançado o destino desejado (VASSEUR; DUNKELS,

2012). As principais vantagens da rede de sensores sem fio são a economia de energia, o baixo custo e a melhor detecção (ZHAO, 2004).

Em geral esses sensores podem captar diversos tipos de dados como humidade, temperatura, pressão, movimento, velocidade, entre outros. Estes sensores podem ser utilizados para aplicações militares como parte do *C4ISRT*, responsável por processos como comunicação, inteligência, reconhecimento, entre outros (AKYILDIZ et al., 2002).

Pode também ser utilizado para aplicações ambientais como controle de inundações e queimadas, além da melhoria no plantio e pecuária. Na área da saúde pode facilitar o processo de sensoriamento para acompanhamento de doenças e tratamentos. E por fim, há o uso residencial e industrial para controle de elementos como temperatura, verificação de processos automatizados, etc (AKYILDIZ et al., 2002).

2.1.2 Internet das Coisas

O termo "Internet das Coisas"foi utilizado pela primeira vez em 1999 por Kevin Ashton, em palestra para a *Procter & Gamble*. Este era utilizado para designar uma rede de controle de suprimentos (ASHTON, 2009). Ashton acredita que dando sentido aos computadores através de sensores, pode-se controlar qualquer tipo de processo, reduzindo assim custos e tempo de reparos.

Porém, em 2005, a *International Telecommunications Union* redefiniu o que seria a Internet das Coisas, como sendo:

> Conectar objetos de maneira sensorial e inteligente a partir da combinação do desenvolvimento tecnológico na identificação de objetos ("*tagging things*"), redes de sensores sem fio ("*feeling things*"), sistemas embarcados ("*thinking things*") e nanotecnologia ("*shrinking things*") (SUNDMAEKER et al., 2010).

A *IoT* é formada a partir de dispositivos de sistemas embarcados. No entanto, para serem considerados objetos inteligentes, esses devem possuir quatro componentes básicos: dispositivo de comunicação (*tagging things*), microcontrolador (*thinking things*), conjunto de sensores e atuadores (*feeling things*), e fonte de energia (VASSEUR; DUNKELS, 2012). A relação entre eles pode ser visualizada na Figura 2.

Seu campo de uso é similar ao da *WSN*, porém ainda há alguns problemas referentes a padronizações, como intercomunicação, mudança na rede atual para comportar a quantidade enorme de dispositivos e dados trafegados, segurança, privacidade, integridade de dados, entre outros.

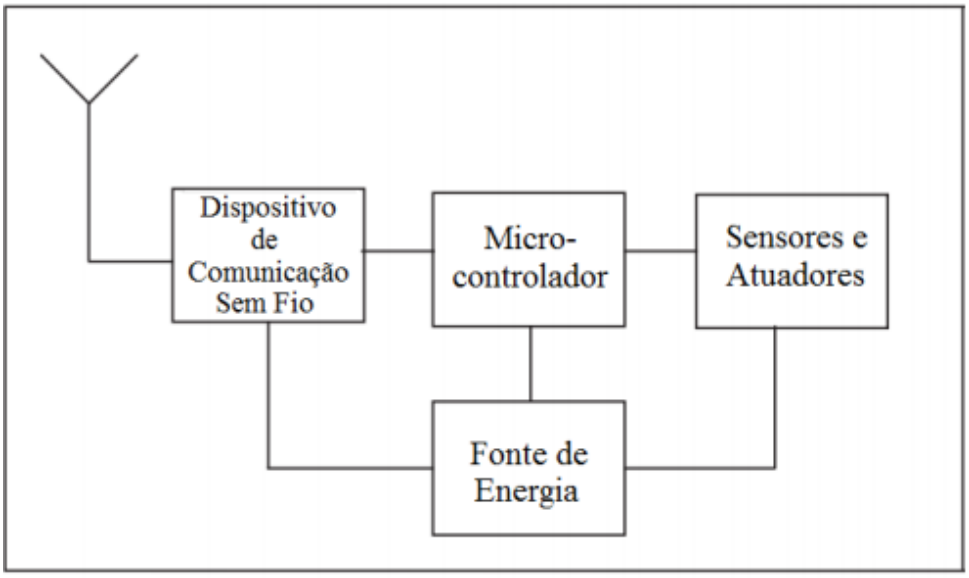


Figura 2 – Dispositivos de uma rede de sensores sem fio.

Fonte: Elaborada pelo autor.

2.2 *Edge* e *Fog Computing*

A Internet das Coisas, utilizando conceitos como *Cloud Computing*, *Fog Computing* e *Edge Computing*, organiza seu fluxo de dados afim de lidar com sua quantidade massiva de dados. Em 2012 a Cisco propôs a criação da *Fog Computing* com o intuito de lidar com o aumento de latência e o crescimento de dados enviados para processamento na *Cloud* devido a mudança para um paradigma de computação em nuvem. Esta alteração permitiria trazer parte do processamento mais perto do dispositivo, evitando assim que procedimentos mais simples necessitassem trafegar até a nuvem para poder serem computados.

Segundo Mahmud, Kotagiri e Buyya (2017), Al-Qamash et al. (2018), esses benefícios incluem um menor tráfego de dados e um armazenamento mais distribuído, além de diminuir consideravelmente a latência. Entretanto, com a chegada da *IoT* e o crescimento ainda mais acelerado do fluxo de dados, os mesmos problemas que existiam com a *Cloud* passaram a ser um problema possível na *Fog*, fazendo com que se fosse proposto a *Edge Computing* (AL-QAMASH et al., 2018).

Esta camada surgiu como uma forma de maximizar o ganho obtido com a *Fog Computing*, fazendo com que cada camada acima receba apenas o necessário, diminuindo o fluxo de dados e a latência de forma significativa. Este paradigma também propõe que cada dispositivo haja como produtor e consumidor de dados, onde contidos em uma mesma rede local, compartilhe informações mais eficientemente e com menor preocupação com vazamento de dados (AL-QAMASH et al., 2018).

Os conceitos de *Cloud Computing*, *Fog Computing* e *Edge Computing* acabam se diferenciado em relação a proximidade da rede com a Internet. A camada mais próxima do

dispositivo inteligente é a *Edge Computing*, responsável pelo processamento do dispositivo. Logo em seguida vem *Fog Computing* encarregada da infra-estrutura e conexão entre os objetos inteligentes (SHI et al., 2016). Esta abordagem promove uma melhor distribuição do processamento e, consequentemente, melhor escalabilidade (BUYYA; DASTJERDI, 2016). As camadas de *Cloud*, *Fog* e *Edge* e seus componentes podem ser vistos na Figura 3.

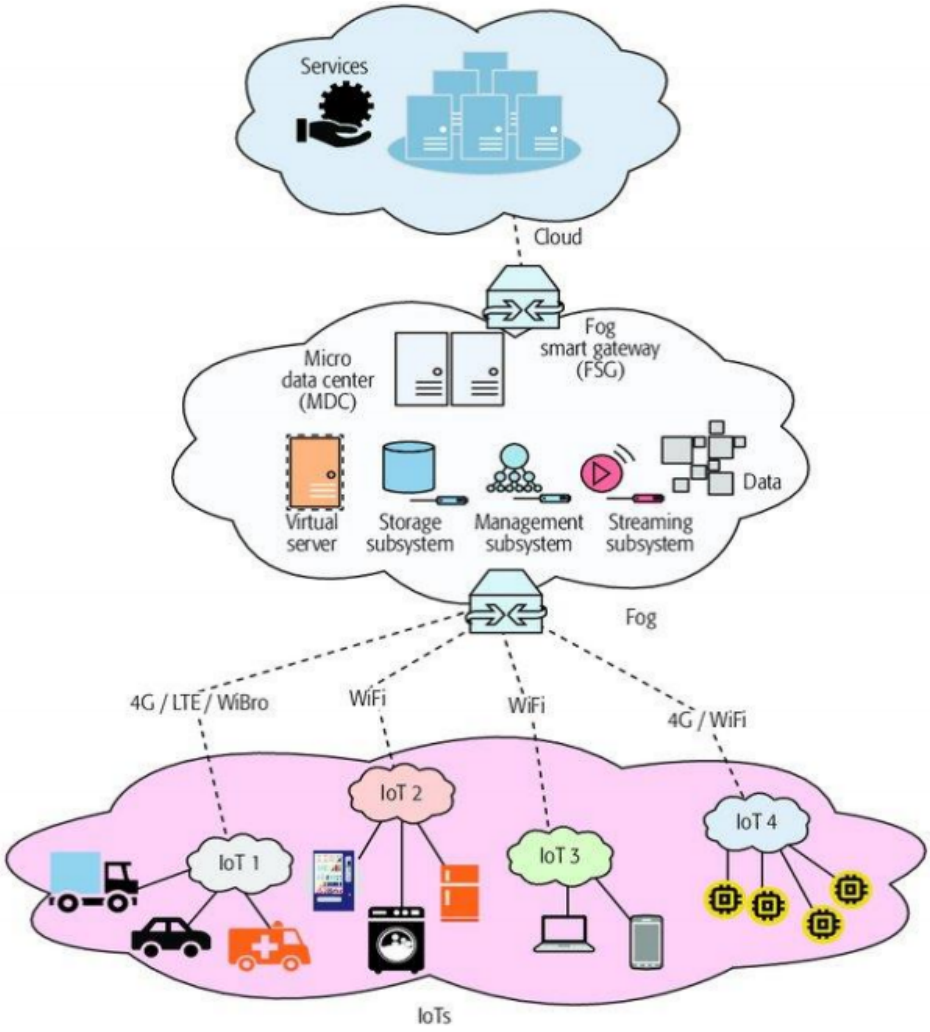


Figura 3 – *Edge*, *Fog* e *Cloud Computing*.

Fonte: Shafik e Mostafavi (2019).

Esta abordagem transfere o processamento feito na Internet (conhecida como "*cloud*") para a borda da rede, conhecida como "*edge*". Com isso, é permitido que a computação seja iniciada mais rapidamente, havendo uma diminuição da latência e do tráfego de dados na rede.

Porém existem desafios que pretendem ser abordados usando a computação de borda. Dois dos principais problemas são o gerenciamento do serviço que, indo para a borda, consegue ser mais distribuído e com menor sobrecarga. Entretanto, para que seja considerado confiável, é necessário garantir as seguintes características: priorização de demandas, expansibilidade e isolamento (AL-JANABI; SHEHAB, 2019).

Outro desafio é a segurança e privacidade. Isto acontece por que há muitos dispositivos de rede que se mantém com usuários e senhas padrões por falta de conhecimento, além da limitação de recursos que naturalmente redes como a *WSN* e a *IoT* já possuem. Assim, esse desafio é latente e alvo de estudo pela comunidade científica para investigar essas implicações (AL-JANABI; SHEHAB, 2019).

2.3 Ataque Distribuído de Negação de Serviço e *Botnets*

Os ataques distribuídos de negação de serviço (ou *DDoS*) são investidas que visam esgotar os recursos de uma máquina, levando a interrupção de serviços. Esses podem ser direcionados a servidores ou dispositivos de rede (MARZANO et al., 2018). Apesar de serem bastante conhecidos, com o surgimento da *WSN* e da *IoT*, ataques como o *Mirai* tem levantado questionamentos sobre a segurança destes tipos de rede. Isso porque elas são compostas por dispositivos mais simples e em geral com esquemas de segurança mais fracos devido as suas limitações.

Os passos para a formação de uma *botnet* passam por i) Identificação de portas de acesso para dispositivos conectados a rede; ii) Ataques de força bruta para ganhar acesso a esses dispositivos; iii) Terminar possíveis concorrentes ao controle do dispositivo; iv) Criar canal para comando e controle (C&C) com o com o dispositivo líder; v) Executa *scripts* maliciosos; vi) Espalhamento pela rede procurando por novos dispositivos; vii) Lançamento de ataques ou outras ações maliciosas (DIETZ et al., 2018). O ciclo de vida de uma *botnet* descritos anteriormente podem ser vistos na Figura 4.

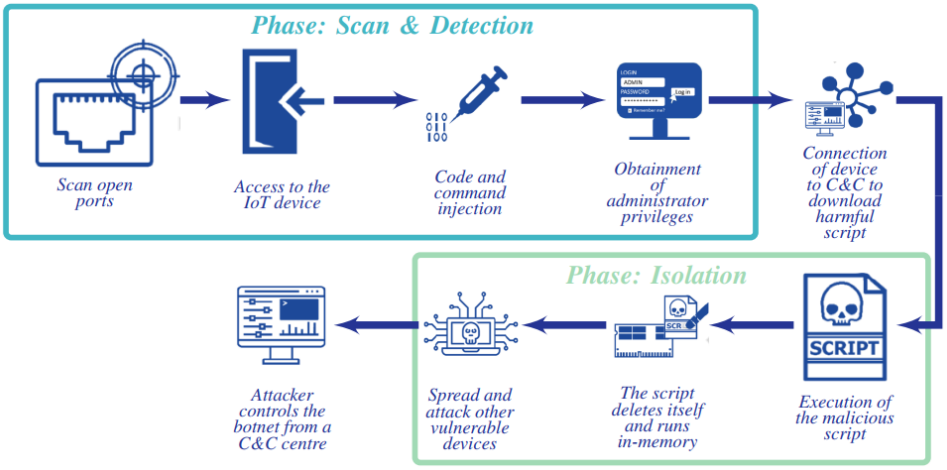


Figura 4 – Ciclo de vida para uma rede *botnet*.

Fonte: Dietz et al. (2018)

2.3.1 *Mirai*

O *Mirai*, descoberto em Agosto de 2016, forma redes *botnets* para ataques *DDoS* a partir de dispositivos como *Digital Video Recorders* (*DVRs*), câmeras WebIP e servidores Linux com baixa segurança (KAMBOURAKIS; KOLIAS; STAVROU, 2017). A dispersão do ataque começa com o dispositivo inicial se conectando a um *IP* público, fazendo um escaneamento *Telnet*. Logo em seguida há uma tentativa de autenticação nos dispositivos encontrados, usando as portas e credenciais comuns de fabricantes. Uma vez autenticado, aquele dispositivo servirá como fonte para distribuir requisições nos próximos ataques de *DDoS*.

O *Mirai* ficou amplamente conhecido em setembro de 2016, sua primeira aparição, onde gerou um fluxo de dados de 620 Gbps no site de consultoria de *Brian Krebs*. Este ataque foi seguido no mesmo mês por um outro com um fluxo de dados de 1,1 Tbps no servidor francês OVH. Em outubro do mesmo ano, após anunciar uma *botnet* de mais de 400.000 dispositivo, foi feito um ataque ao provedor Dyn inutilizando sites como *Twitter*, *Netflix*, *GitHub* e *Reddit* por várias horas (KOLIAS et al., 2017).

2.3.2 *Bashlite*

O *Bashlite* é um dos predecessores do *Mirai* (MARZANO et al., 2018). A estrutura básica do ataque é igual, entretanto o *Bashlite*, na sua versão original, necessita do uso de extensões, tais como: ferramenta para que *bots* da *botnet* consigam procurar por mais dispositivos vulneráveis; utilização de *DNS* para ataque e; envio de comandos usando protocolo binário compactado, para dificultar a sua descoberta. Estas extensões estão presentes nativamente no *Mirai*.

Apesar de ser menos conhecido que o *Mirai*, estatísticas mostram que esse *malware* já contaminou mais de 1 milhão de dispositivos (principalmente câmeras e *DVRs*), podendo lançar ataques com um fluxo maior que 400 Gbps. E devido ao vazamento de parte do seu código fonte em 2015, este foi utilizado para criar o próprio *Mirai* (ANGRISHI, 2017).

3

Trabalhos Relacionados

Nesta seção, serão discutidos trabalhos de pesquisa para ambientes de sistemas embarcados, *IoT* e *WSN* no âmbito de segurança para identificar o que vem sendo feito para aumentar a segurança nestes ambientes restritos. Para a seleção do trabalho, foram escolhidos os artigos que ou usavam aprendizagem de máquina ou que tratavam de segurança em ambientes restritos.

3.1 Trabalhos de Pesquisas Relacionados

O trabalho de Endler, Silva e Haeusler (2017) propôs um processador de evento complexo (*CEP*) voltado para um cenário de *IoT*, com um refinador de dados do evento, dando uma classificação semântica e atribuindo um fato associado ao dado. Este trabalho consiste em um pré-processamento para os dados que chegam do *stream* para melhorar o resultado de um posterior algoritmo de aprendizagem de máquina (o que é sugerido como trabalho futuro pelos autores). Os dados disponibilizando podem ser usados como características úteis pela técnica de aprendizagem escolhida. Porém não foram feitos experimentos para comprovar a eficácia do método.

A pesquisa de Kanoun, Tekin e Atienza (2016) transforma a abordagem de *Multi-Armed Bandits* com um único *stream* de dados para um cenário de múltiplos *streams*, sendo aproveitado para aprendizagem no cenário de *Big Data*. Foi desenvolvida uma aplicação e testada a qualidade dos resultados, recurso alocado e o uso destes em comparação com o algoritmo *S-MAB*. O algoritmo proposto perde em performance para o *S-MAB*, mas consume até 4% menos recursos.

O artigo desenvolvido por Zhao (2005) propõe o *NNTrees*, criando uma árvore de decisão onde os nós não terminais são redes neurais especializadas para processamento de *streams*. A abordagem é testada em bases de dados tradicionais adaptadas para *streams*, e estas são finitas e não muito grandes (segundo o próprio autor), implantando diversas redes neurais (em uma *Unix Workstation*). O autor cita que a atualização das redes neurais são feitas com menos épocas e com

um menor número de parâmetros para onerar menos o sistema, obtendo performance similar ao C4.5 na acurácia com menor custo de recurso, mas com maior tempo computacional.

O trabalho de Amza e Cristea (2011) introduz um sistema de detecção de intrusão fazendo uma abordagem múltipla entre um motor de comparação de padrões e uma rede neural funcionando em paralelo. Este trabalho busca melhorar a eficiência para redes de computadores. O algoritmo foi avaliado utilizando um *dataset* inicialmente com 1.940 dados e posteriormente com 38.711 exemplos. Foram considerados o número de falso-positivos (0,03%), taxa de detecção (99,3%) e tempo de processamento (2 minutos e 20 segundos para um conjunto de dados de 100.000). Porém, como os autores comentam, há um custo associado à comparação de padrões quando utilizado *datasets* grandes pela necessidade de iterar sobre todos os dados. Com os resultados obtidos, os autores concluem que a junção das técnicas funciona melhor do que usar apenas uma delas separadas.

Agarwal et al. (2016) sugere o uso de combinações de técnicas para melhorar a detecção de atividades em uma residência, voltado para um cenário de *IoT* e *Smart Home*. Os autores utilizam um *ensemble* de técnicas de aprendizagem de máquinas (*Random Forest*, *XGBoost* e *k-Nearest Neighbors*), montando dois níveis de combinações. São utilizados 6 instâncias de algoritmos de aprendizagem, além da combinação do resultado de 5 destas, submetendo os dados a uma *TF-IDF*. Foram avaliados a predição do algoritmo com a atividade real, conseguindo um *Area Under de Curve* (*AUC*) de 0,869.

O projeto de Dey et al. (2016) abordou o uso de *Random Forest* em *stream* para identificar a ocupação de salas a partir de medições do nível do CO_2 para um cenário de *Wireless Sensor Network* (*WSN*) e *IoT*. Os autores mostram que houve uma eficácia boa (acurácia de acima de 71%, chegando até a 91%), mas não foi levantado o tempo e custo de processamento, também não sendo especificado em qual dispositivo foi feito o treinamento.

Apesar do foco do trabalho de Kapoor e Dhavale (2016) ser na parte de obtenção e seleção de características, são aplicados os métodos de *Naive Bayes*, *Support Vector Machine* (*SVM*) e *Random Forest*. A abordagem utilizada de extrair o *Control Flow Graph* (*CFG*) e *OpCodes* associando com Separação Binormal se mostrou eficaz em classificar melhor o *malware* em grupos, ao invés de uma separação binária (benigno/maligno). Obteve-se resultados melhores do que algoritmos tradicionais como *Naive Bayes*, *SVM* e *Random Forest*.

A sugestão apresentada por Roopaei, Rad e Jamshidi (2017) seria o uso de *Deep Learning - Convolutional Neural Network* (*CNN*). Como apontado no artigo, seria implantado utilizando uma arquitetura específica para ser realizada na *cloud*, levando o processamento e memória alocada para os servidores, apenas alimentando os nós com os resultados.

Donovan et al. (2018) propõe uma arquitetura para aprendizagem de máquina para *IoT* para um ambiente mais industrial (o qual chama de Indústria 4.0). Esta propõe uma arquitetura que envia para as *fogs* para montagem dos modelos de *Predictive Modelling Markup Language*

baseados naqueles já existentes na *cloud*, a serem utilizados e depois repassa para os nós da rede. Porém, apesar de ser um método que atualiza o nó com o modelo já calculado, o processo de atualização destes dependem de um ambiente externo (no artigo foi usado o *Amazon Web Services*). Foram feitos testes para avaliar o tempo de execução e número de falhas de conexão dos nós com a *fog* e desta com a *cloud*. Os resultados mostram que há um aumento na latência, que pode chegar a 99,4%, e no número de falhas de conexão, de até 6,6%, quando a quantidade de conexões aumenta devido a comunicação com a *cloud*.

Li (2014) pesquisou o uso de *Pattern Query Language* com *Local and Globle Consistency* (*LGC*) e *SVM* para classificar o *stream* de dados para uma *WSN* distribuída. Foram coletados 4.500 dados de sensores e verificado a acurácia, obtendo 76,65%. Porém os próprios autores salientam a necessidade ter todos os padrões que deseja se identificar previamente (visto que tanto o *LGC* quanto o *SVM* são métodos supervisionados), limitando a aplicação. Além disso, há a necessidade do retreinamento periódico, tanto do *LGC* quanto do *SVM*, para readaptação.

A pesquisa desenvolvida por Bhattacharyya, Katramatos e Yoo (2018) consiste na criação de um *framework* para as *Software Defined Networks* (*SDN*). É mais um caso onde os dados fluem primeiro para um *data center* antes de irem para os nós, servindo como a fonte de segurança. Mas como os autores citam, existem desvantagens na introdução de latência na rede, sendo demonstrada nos testes realizados, onde há um aumento praticamente exponencial a partir de 1.000 pacotes simultâneos. Além disso, abordagens que dependem exclusivamente de uma *fog* ou nó externo já foram mencionadas como problemáticas em outros trabalhos. A tentativa de paralelização do processo utilizando *FPGAs* ou *GPUs* foi proposto como trabalho futuro, tentando diminuir o tempo de processamento, onerando menos a latência.

O trabalho de Afghah et al. (2018) sugere o uso em *SDN* (neste caso, mais especificamente na *Software Defined Wireless Network* ou *SDWN*), utilizando as Funções Fisicamente Não-Clonáveis (*PUF*) baseadas nas memórias resistivas. Para isso, ele propõe um protocolo de troca de informações e faz as devidas análises sobre as medidas de segurança tomadas para as principais ameaças em cada etapa. Assim como o trabalho de Bhattacharyya, Katramatos e Yoo (2018), todo o fluxo passa por um sistema intermediário, aqui chamado de *Control Plane*.

Uma abordagem levantada por Schmidt, Kountanis e Al-fuqaha (2014) seria utilizar um algoritmo imuno-inspirado. Este trabalho parte de levantamentos prévios (MOORE; ZUEV, 2005; ALSHAMMARI; ZINCIR-HEYWOOD, 2009; SINGH; AGRAWAL, 2011), onde os autores chegam a conclusão que o melhor resultado para classificação de um *stream* de dados seria utilizando árvore de decisão (pelos artigos, o C4.5). A partir desses dados, é comparado com o algoritmo imuno-inspirado proposto. Para isso é utilizada a base de dados presente no trabalho de Moore e Zuev (2005), com o algoritmo de pré-processamento *Fast Correlation-Based Filter* (*FCBF*). O *FCBF* é utilizado para reduzir de 249 características para as 11 que mais impactam no processamento além de normalizar os dados, tentando tornar o algoritmo mais eficiente com uma perda pouco significativa na eficácia final. Este trabalho foi testado com o código utilizando

Python em uma máquina com um processador *Intel i5* com 1,8 GHz e 4 GB de memória RAM. Os autores citam a viabilidade para utilizar este algoritmo em ambientes *IoT* pela sua capacidade de generalização boa em *datasets* pequenos com resultados similares a algoritmos como *SVM* e *Naive Bayes*. Entretanto, também é discutido que o algoritmo é mais lento que estes dois últimos, e que para uma acurácia boa, é necessário um número maior de anticorpos.

Uma melhoria sobre o algoritmo de *Visual Assessment of Tendency* (*VAT*) (Bezdek; Hathaway, 2002) foi apresentada por Kumar et al. (2016) para melhorar a iteratividade e adaptabilidade para um cenário de *data stream*. Esta melhoria permite o incremento e decremento de elementos dos agrupamentos a partir da evolução natural do fluxo de dados, além de possibilitar a detecção de anomalias. Entretanto, esta abordagem exige um modelo inicial dos agrupamentos, necessitando um conhecimento prévio dos dados, o que dificulta sua adaptabilidade para cenários mais aleatórios.

O trabalho de Axenie e Bortoli (2018) propõe um método para processamento e gerenciamento de dado voltado para aprendizagem de máquina. Os requisitos buscados pelos autores foram baixa latência, alta vazão, alocação de memória fixa e flexibilidade. Características estas fundamentais para ambientes restritos, baseadas na observação dos autores onde, em geral, o custo computacional é alto. Isto se dá pelo recálculo constante de grandes quantidades de dados e pelo enorme alocamento de memória para essa quantidade massiva de dados. Os autores então propõem um algoritmo baseado em janela deslizante, armazenando apenas ocorrências de um dado evento, o que permite uma menor estrutura considerando apenas características importantes. Estas são extraídas por uma camada anterior antes de ser feito a parte responsável pela janela. Não é um sistema que utiliza aprendizagem de máquina em si, mas faz um trabalho de pré-processamento para melhorar a eficiência, diminuindo a latência e aumentando a vazão de dados.

Akbar et al. (2015b), Akbar et al. (2015a), Akbar et al. (2017) partem da abordagem de janela de Axenie e Bortoli (2018), mas incluindo *CEP* com aprendizagem de máquina, montando uma abordagem composta para tratamento de *stream*. Os autores indicam que o uso de *CEP*, devido a sua natureza distribuída de processamento, faz dele um ótimo candidato para analisar o fluxo de dados em ambientes *IoT*. Porém como os autores lembram, o problema da *CEP* para um ambiente reativo é o fato de que essa abordagem é imprevisível, visto que em geral o ideal é prever o evento ao invés de simplesmente reconhecê-lo após ocorrido. Para resolver esse problema os autores propõem um algoritmo supervisionado adaptativo de aprendizagem de máquina com a janela (*Adaptive Moving Window Regression* ou *AMWR*) para antever os dados que chegarão no fluxo, melhorando a capacidade de preditiva da *CEP*. Isto auxilia no problema da falta de escalabilidade e dificuldade em lidar com diversas fontes de fluxo de aprendizagem de máquina. Tal abordagem possibilita uma taxa de erro que varia, segundo os resultados levantados, entre 10,59% e 15,5%.

Salehi et al. (2015) sugerem que a detecção de *outliers* locais vem sendo pouco explorado,

sendo o foco maior nos *outliers* globais, quando se trata do uso de *stream* como fonte de dado. O trabalho citado pelos autores, que feito por Pokrajac, Lazarevic e Latecki (2007), lida de fato com detecção local de *outliers* e tem como problema precisar do conhecimento de todos os dados prévios para detectar de forma eficaz. Segundo Salehi et al. (2015), isto tornaria o algoritmo impraticável além de ficar sujeito a erros por dados muito antigos. Para solucionar este problema, os autores propõem um algoritmo de aproximação. Pensando em sistemas com limitações de memória, o algoritmo armazena as pontuações ao invés dos dados, para a partir destas, recalcular com dados novos. Isto remove a necessidade de ter os dados antigos que possam prejudicar a acurácia do sistema devido a aproximação realizada, sendo a acurácia e uso de memórias melhores do que o algoritmo usado para comparação, o iLOF (POKRAJAC; LAZAREVIC; LATECKI, 2007).

O algoritmo de *FlockStream* criado por Spezzano e Vinci (2015) foi proposto com o objetivo de lidar com o grande volume de dados usando *stream*, porém com o enfoque em sistemas com limitações de memória. Além disso, trata-se um algoritmo de não supervisionado onde não se faz necessário uma classificação prévia para o treinamento dos modelos gerados. O ambiente utilizado pelos autores para construir o *FlockStream* foi uma rede *WSN*. Também é citado no artigo o algoritmo *DenStream* (CAO et al., 2006), por ter similaridades em vários conceitos utilizados, diferenciando apenas pela forma de inicialização e a abordagem do agrupamento. Os experimentos foram feitos utilizando uma máquina com Intel(R) Core(TM)2 6600 com 2 GB de memória RAM. O resultado apresentado compara os *clusters* gerados e os compara com o *DenStream*, obtendo agrupamentos similares.

O algoritmo proposto por Kourtellis e Bifet (2016) parte do bom desempenho em atividades de classificação de dados de árvores de decisão (também citado nos trabalhos de Moore e Zuev (2005), Alshammari e Zincir-Heywood (2009), Singh e Agrawal (2011)), além da facilidade de interpretação humana por criar regras facilmente legíveis. A partir disso, utilizam o método *Hoeffding Tree* (DOMINGOS; HULTEN, 2000), aplicando em um cenário distribuído, é proposto o algoritmo *Vertical Hoeffding Tree*. Este permitiria aproveitar, segundo os autores, a eficiência de se trabalhar com *stream*, lidando com a eficiência do processamento distribuído. O experimento faz um comparativo de algumas abordagens utilizadas no método sendo testadas em uma máquina com Intel Xeon X5650 (24 núcleos com *clock* de 2.67 GHz) e 64GB de memória. Os resultados mostram que a acurácia não é alterada com o paralelismo, e que a vazão da informação processada aumenta. A acurácia também é verificada no trabalho de Elkhoukhi (2018), que utilizou o mesmo algoritmo para, a partir de um sensoriamento (umidade, temperatura, luz, sensor de presença e nível de CO_2), estimar o número de pessoas em um determinado ambiente.

O trabalho proposto por Singh et al. (2013) parte de um cenário de uma rede *WSN* para utilizar algoritmos de aprendizagem (*CluStream* e *SVM*) para detecção de incêndios em florestas. Como os autores citam, o *CluStream* é utilizado na primeira fase do algoritmo para lidar com o enorme fluxo de dados da rede de forma distribuída. Os agrupamentos iniciais gerados são

enviados para uma unidade central, onde, utilizando *SVM*, faz a predição ou detecção de incêndios. Não são apresentados resultados práticos, sendo apenas levantado a análise de complexidade, onde é mostrado pelos autores que o principal algoritmo responsável pela complexidade final da abordagem é o *SVM*, devido a leveza do *CluStream*.

Por fim, a proposta de Lu et al. (2018) apresenta o *P-DenStream*, versão paralelizável do *DenStream*. O *DenStream* possui a capacidade de classificação utilizando *data streams*, mas com um custo associado a parte inicial do processo que consiste na criação de *core-micro-clusters* (*CMC*). Lu et al. (2018) então sugerem a paralelização da formação dos *CMC*, para posterior utilização do *DenStream* para formação dos agrupamentos finais. Foi avaliado utilizando dados sintéticos e reais da utilização de táxis de uma cidade, sendo observado a pureza dos agrupamentos finais e tempo para processamento. Nos resultados, mostrou-se uma pureza maior do que 85%, com o algoritmo sendo considerado pelos autores como eficiente na detecção de *outliers* e com tempo de resposta inferior a 200 ms em um cenário com 23.000 registros. Resultado positivos quando comparados aos 10 segundos obtidos pelo *DenStream* para a mesma quantidade.

3.2 Comparativo Entre os Trabalhos Relacionados

Os trabalhos de Zhao (2005), Amza e Cristea (2011), Kumar et al. (2016), Kapoor e Dhavale (2016), Kanoun, Tekin e Atienza (2016), Roopaei, Rad e Jamshidi (2017), Axenie e Bortoli (2018), Afghah et al. (2018), Bhattacharyya, Katramatos e Yoo (2018) não tinham como cenário um ambiente restrito, e com isso não haviam preocupação com as limitações impostas como memória e processamento. O trabalho de Donovan et al. (2018) apesar de ser um trabalho voltado para um ambiente próximo (*fog computing*), não utilizou nem aprendizado de máquina nem faz levantamento sobre segurança, tendo pouco impacto para o trabalho aqui proposto. Com isto, os trabalhos revistos podem ser visualizados na Tabela 1.

Dey et al. (2016), mostra que houve uma eficácia boa, mas não foram levantados tempo e custo de processamento, também não sendo especificado em qual dispositivo foi feito o treinamento. Por usar *Random Forest*, irá formar diversas árvores para combinação, o que pode onerar os dispositivos.

Agarwal et al. (2016) utiliza 6 instâncias de algoritmos de aprendizagem, além da combinação do resultado de 5 destas e de submeter os dados a uma *TF-IDF*, demandando muito das plataformas. Além disso não há uma menção sobre o uso destas técnicas para segurança.

O trabalho de Endler, Silva e Haeusler (2017) não lida com a classificação em benigno ou maligno, servindo como um pré-processamento para os dados que chegam do *stream*. O algoritmo visa melhorar o resultado de uma técnica de aprendizagem de máquina, disponibilizando mais dados que podem ser usados como características úteis pelo método escolhido.

Li (2014) conclui no fim do trabalho que é a aplicação é limitada por ser necessário ter

todos os padrões que se deseja identificar previamente, visto que tanto o *LGC* quanto o *SVM* são métodos supervisionados. Além disso, o retreinamento periódico para esses algoritmos são um gasto computacional grande para o dispositivo.

A abordagem levantada por Schmidt, Kountanis e Al-fuqaha (2014) não verifica a viabilidade em um ambiente restrito, apesar dos autores citarem que existe a possibilidade de utilização deste algoritmo em ambientes *IoT* pela sua capacidade de generalização boa em *datasets* pequenos. Os autores também citam que o algoritmo é mais lento que algoritmos como *SVM* e *Naive Bayes*, e que para uma acurácia boa, é necessário um número maior de anticorpos, que em ambientes restritos pode onerar recursos de processamento e memória.

O *AMWR* proposto em Akbar et al. (2015b), Akbar et al. (2015a), Akbar et al. (2017) é um algoritmo supervisionado que, pelo menos inicialmente necessite da rotulação dos dados iniciais, permitindo que o algoritmo, após essa fase consiga se adaptar aos novos dados. Além disso, não foram avaliados tempo de resposta, consumo de memória e processamento, avaliando apenas dados previstos com dados reais. Entretanto os autores citam um comportamento quase em tempo real.

O trabalho de Salehi et al. (2015) para detecção de *outliers* locais avaliou do consumo de memória pela preocupação com uma *WSN*, por mais que não tenha sido verificado em um cenário restrito. Porém não foram analisados gasto de processamento, vazão de dados e consumo de memória, como mencionado pelos autores.

O *FlockStream* proposto por Spezzano e Vinci (2015) possui diversas similaridades conceituais com o *DenStream* o que, considerando o trabalho de Lu et al. (2018), sinalizam que o *DenStream* e o *FlockStream* tenham um uso adequado para cenários restritos. Mas em ambos não foram feitas considerações do uso destes algoritmos para segurança e no caso do *FlockStream*, não houveram experimentações em cenários restritos apresentados pelos autores.

O algoritmo supervisionado *Vertical Hoeffding Tree* proposto por Kourtellis e Bifet (2016) e utilizado no trabalho de Elkhoukhi (2018) mostrou resultados bons com o paralelismo e em ambientes restritos. (ELKHOUKHI, 2018) realizou experimentações em uma rede de sensores para detecção de ocupação, sendo fora do escopo de segurança.

O trabalho de Singh et al. (2013) utiliza uma composição dos algoritmos *CluStream* na parte distribuída e *SVM* na parte centralizada. Entretanto não houve experimentação prática, sendo realizada apenas a verificação teórica da complexidade do algoritmo final. Por fim, o algoritmo *P-DenStream* proposto por Lu et al. (2018) fez uma variação no *DenStream*, permitindo sua paralelização e distribuindo melhor o custo operacional. No entanto para um cenário restrito, não foi avaliado uso de memória e processamento.

Com este trabalho pretende-se dar um enfoque maior na segurança, trazendo a aprendizagem de máquina, para os ambientes mais restritos, como *IoT* e *WSN*, sem onerar os recursos dos dispositivos da rede e que se adapte a futuras ameaças que o algoritmo consiga detectar.

Ano (Autor)	**Machine Learning**	**Segurança**	**Ambiente**	**Avaliação**
(SINGH et al., 2013)	*CluStream e SVM*		*WSN*	Análise de complexidade
(SCHMIDT; KOUNTANIS; AL-FUQAHA, 2014)	Algoritmo imuno-inspirado	**X**	*IoT*	Simulação; Comparação com SVM e Naive Bayes
(LI, 2014)	*LGC* e *SVM*		*WSN*	Acurácia e tempo
(SALEHI et al., 2015)	Algoritmo para *local outlier*		*WSN*	taxa de detecção e uso de memória
(AKBAR et al., 2015b)	Pré-processamento		*IoT*	Análise entre predição e dados reais
(AKBAR et al., 2015a)	Pré-processamento		*IoT*	Análise entre predição e dados reais
(SPEZZANO; VINCI, 2015)	*FlockStream*		*IoT*	Comparação dos agrupamentos com *DenStream*
(Dey et al., 2016)	*Random Forest*		*IoT*, *WSN*	Análise entre predição e dados reais
(AGARWAL et al., 2016)	*Random Forest*, *XGBoost*, *kNN* e *ensembles*		*IoT*	Análise entre predição e dados reais
(KUMAR et al., 2016)	*VAT*, *iVAT*			Evolução de agrupamentos usando *data stream*
(KOURTELLIS; BIFET, 2016)	*Vertical Hoeffding Tree*		*IoT*	Comparação com o *CHT*
(ENDLER; SILVA; HAEUSLER, 2017)	Pré-processamento		*IoT*	
(AKBAR et al., 2017)	Pré-processamento		*IoT*	Análise entre predição e dados reais
(ELKHOUKHI, 2018)	*Vertical Hoeffding Tree*		*IoT*	Análise da acurácia
(LU et al., 2018)	*P-DenStream*		*IoT*	Pureza e tempo de resposta

Tabela 1 – Comparação entre os trabalhos relacionados.

Fonte: Elaborada pelo autor.

4

Algoritmos de aprendizagem de máquina avaliados

Este capítulo aborda sobre os algoritmos de aprendizagem de máquina, tratando sobre os possíveis candidatos levantados para tratar sobre detecção da formação da *botnet*. Todos esses algoritmos se baseiam em agrupamentos e são voltados para fluxo de dados com alguma possibilidade de detecção de *outlier* (SILVA et al., 2013). Além disso, todos os algoritmos possuem uma fase de aprendizagem (fase *offline*) e a fase de análise de novas amostras (fase *online*).

4.1 Aprendizagem de Máquina

A aprendizagem de máquina é uma subárea da Inteligência Artificial que vem sendo utilizada em diversas outras com a tentativa de automatizar e melhorar a classificação de dados. Na área de segurança, existem estudos que visam verificar a viabilidade do uso para assegurar a rede de ameaças externas (ALSHAMMARI; ZINCIR-HEYWOOD, 2009; MOORE; ZUEV, 2005; SINGH; AGRAWAL, 2011).Muitos destes têm considerado estratégias supervisionadas de aprendizagem de máquina.

Esta técnica costuma ter taxas de acurácia e eficiência boas para cenários conhecidos e devidamente categorizados (MOORE; ZUEV, 2005). Entretanto, para um cenário onde os ataques sofrem modificações, técnicas supervisionadas necessitam de novos conjuntos de dados categorizados com a inclusão dos novos ataques para se adaptar ao cenário vigente. Uma alternativa seria a utilização de técnicas de aprendizagem não-supervisionadas, permitindo que o próprio sistema consiga separar em grupos distintos tráfego de dados similares entre si.

O levantamento feito por Silva et al. (2013) traz os principais algoritmos não-supervisionados que foram desenvolvidos para um cenário de de agrupamento utilizando *data streams*. Como citado pelo autor, a maioria dos algoritmos de aprendizagem de máquina partem de um cenário onde a quantidade de dados trabalhados são finitos. Mas, para a mineração de *data streams*, a

quantidade de dados que chegam para serem processados podem ser considerados praticamente infinito.

Algoritmos como *BIRCH*, *CluStream*, *D-Stream*, *DenStream*, entre outros, já levam em consideração esse cenário. Isso permite o descarte dos dados após o processamento e assim limita o armazenamento de dados antigos, o que favorece a vazão dos dados que é variante. Destes algoritmos, dois já foram utilizados em cenários restritos como *WSN* e *IoT*: o *DenStream* (LU et al., 2018) e *CluStream* (SINGH et al., 2013).

Os algoritmos *BIRCH*, *CluStream*, *D-Stream*, *DenStream* utiliza para o agrupamento dois algoritmos principais: o *k-means* para o *BIRCH* e *CluStream*; o *DBScan* para o *D-Stream* e *DenStream* (SILVA et al., 2013).

O *k-means* estabelece o posicionamento de uma quantidade previamente conhecida de centróides, pontos onde se esperam os melhores agrupamentos para o conjunto de dados. A partir dos centróides, o algoritmo calcula a distância radial entre os pontos do *dataset* e os centróides, afim de determinar qual agrupamento tem uma menor distância, tendo assim uma maior correlação. Por se valer da distância radial, este algoritmo tende a favorecer distribuições mais globulares, obtendo resultados inferiores com distribuições menos uniformes (SILVA et al., 2013).

Já o *DBScan* se baseia na densidade de um conjunto de dados para formar os *clusters*. Apesar de ainda se basear na distância entre dois dados para considera-los similares, a possibilidade de junção de grupos menores em um grupo maior o torna melhor adaptado para distribuições menos uniformes. Além disso, a parametrização da quantidade de amostras para se formar um agrupamento, lhe permite a detecção de *outliers*.

Realizado o agrupamento, registros fora dos grupos formados são considerados anômalos. Muitos algoritmos apenas descartam esses *outliers*, sendo considerados como anomalias. Porém, existem os algoritmos de detecção de novidade, que ao perceber que existem muitos dados anômalos com características semelhantes, cria um novo grupo, conhecido como grupo novidade. Algoritmos como *MINAS*, *OLINDDA* e *ECSMiner* são exemplos de algoritmos de detecção de novidade (FARIA; GAMA; CARVALHO, 2013).

4.2 *BIRCH*

Balanced Iterative Reducing and Clustering using Hierarchies (*BIRCH*) foi proposto por Zhang, Ramakrishnan e Livny (1996) para ser um algoritmo não supervisionado de agrupamento para grandes conjuntos de dados, voltado para *data mining*. A criação deste algoritmo partia de duas premissas básicas: lidar com a memória finita e muito menor que o conjunto de dados e diminuir o tempo de entrada e saída de dados. Ambas características são desejáveis para o cenário de detecção de formação de *botnet*, onde a premissa da memória faz com que ele lide

melhor com o fluxo de dados e o menor tempo de entrada e saída de dados contribui para um menor tempo de resposta.

Para conseguir atender as premissas, para cada *cluster* formado, são geradas suas características o que permite que não se precise reavaliar os dados novamente na fase *online*, melhorando o uso de memória. Além disso, por considerar os dados de forma local em relação à suas características e, apesar de utilizar o *k-means*, ele assume que os agrupamentos não são uniformemente preenchidos, considerando a densidade daquele grupo, o que permite uma melhor adaptação a dados menos uniformes (ZHANG; RAMAKRISHNAN; LIVNY, 1996).

Em sua fase *offline* o algoritmo constrói a árvore de agrupamentos (conhecida como *Cluster Features Tree* ou *CFT*) a partir dos grupos (conhecido como *Cluster Features* ou *CF*) para os dados passados. Além disso, baseado no parâmetro de limiar (*threshold*) o algoritmo pode decidir por juntar ou separar os *CFTs*. O conceito das *CFTs* foram utilizados em outros algoritmos, entre eles o *CluStream*, sendo renomeado para *micro-cluster* (SILVA et al., 2013). Os agrupamentos gerados pelo *BIRCH* em comparação às diversas distribuições de dados podem ser visto na Figura 5.

Figura 5 – Agrupamentos gerados pelo *BIRCH*.

Fonte: Elaborada pelo próprio autor baseado no site do *Sci-kit Learn*[1].

Na sua fase *online*, o algoritmo recebe novos dados e verifica a necessidade de atualização da sua árvore. Há também um passo adicional chamado de agrupamento global, onde a partir de uma quantidade definida de espaço, os dados com menor densidade na árvore são eliminados por serem considerados anômalos.

4.3 *CluStream*

O *CluStream* proposto por Aggarwal et al. (2003) parte do conceito de *micro-cluster* (similar as *CFs* do BIRCH) e de *macro-cluster*. Os *macro-clusters* seriam agrupamentos de *micro-clusters* para formação de um grupo maior e mais coeso de dados, que possuem características mais próximas.

Um dos enfoques deste algoritmo é a formação dos *micro-clusters* a partir de sumarizações da características, ao invés do dado em si. Isto permite que uma quantidade menor de dado seja

[1] Disponível em: <https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html>

armazenada para a manutenção daquele *micro-cluster* e de uma representação mais agnóstica em relação ao dado em si. Outro enfoque é a evolução dos agrupamentos com o decorrer do tempo. Por ser um algoritmo voltado para fluxo de dados, sua fase *online* é focada na manutenção e atualização dos *micro-cluster*.

Em sua fase *offline*, o algoritmo executa o *k-means* para obter os agrupamentos iniciais de *micro-clusters*. Posteriormente agrupa os *micro-clusters* em grupos maiores, construindo assim os *macro-clusters*. O *k-means* é um dos algoritmos mais leves e eficientes para agrupamentos, mas tem seus *clusters* mais globulares. Isto pode fazer ele ser mais sensível a distribuição de dados, podendo classificar de forma errônea um dado quando seu *dataset* possui um formato mais arbitrário, como pode ser visto na Figura 6.

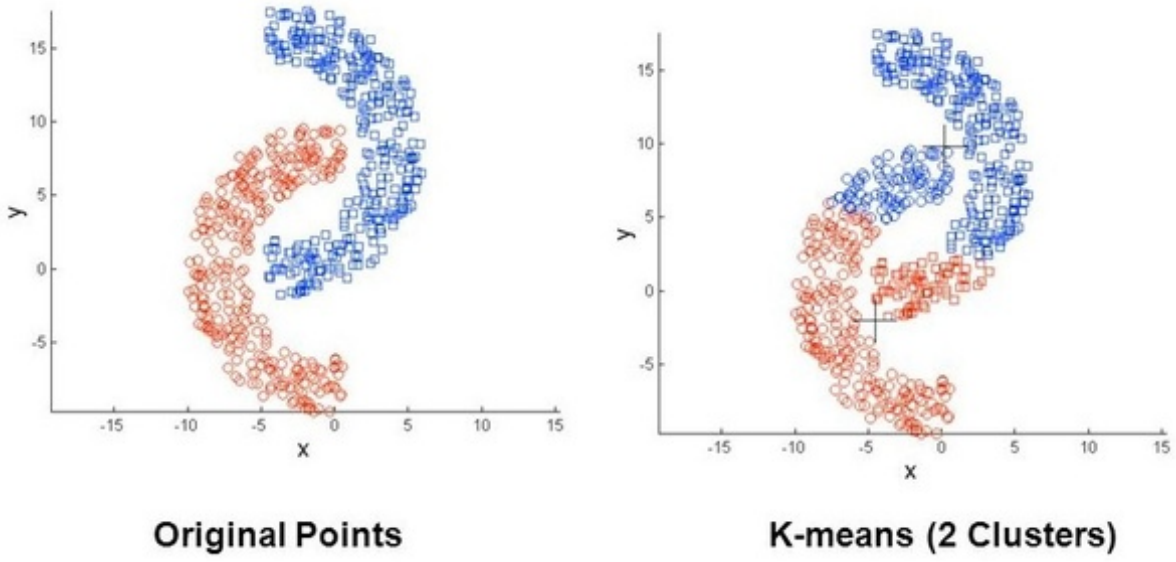


Figura 6 – Agrupamentos gerados pelo *k-means*

Fonte: Quora [2].

Associado a isso, o algoritmo para inicializar precisa que seja disponibilizado previamente a quantidade de agrupamentos a serem formados. Isso demanda um conhecimento técnico prévio, ou várias tentativas com faixas de quantidades aceitáveis para tentar encontrar um número de *clusters* adequado, através de heurísticas como método do cotovelo, máximo valor da silhueta, entre outros.

Além disso, ele também é sensível a inicialização dos centróide. Isto faz com que seja necessário uso de heurísticas para tentar achar a melhor posição dos centroides, evitando que se caia num mínimo local. Este é problema considerado NP-difícil. Um desses exemplos de uso de heurística é o *k-means*++, proposto por Arthur e Vassilvitskii (2007), que baseado na distribuição das distâncias entre os pontos em relação a possíveis centróides, escolhe os melhores

[2] Disponível em: <https://www.quora.com/What-are-the-weaknesses-of-the-standard-k-means-algorithm-aka-Lloyds-algorithm>.

resultados. Entretando, isso faz com que ele perca mais tempo na escolha dos centróides, mas convirja mais rápido no agrupamento.

Na sua fase *online*, a cada amostra nova o algoritmos encaixa o dado em algum dos *micro-clusters* existentes e, periodicamente, refaz a construção dos *macro-clusters*, podendo unir ou separar *micro-clusters* e gerar novos *macro-clusters*, se adaptando. Este processo de reconstrução dos *macro-clusters* é chamado de *Global Clustering*.

4.4 D-Stream

D-Stream foi proposto por Chen e Tu (2007) para lidar com a condição de algoritmos como o *CluStream*. Este, por ser baseado no *k-means*, tem um tratamento pobre com relação a dados anômalos e tem agrupamentos em formatos globulares. Além disso, há necessidade de conhecer previamente a quantidade de agrupamentos. Para isso, o algoritmo parte do conceito de grade, mapeando a área de distribuição de dados em um grid com divisão de tamanhos iguais. Cada divisão contém todos os dados que estão na faixa designada por aquele pedaço da rede.

O outro conceito utilizado para finalizar os agrupamentos é a densidade (similar a algoritmos como *BIRCH* e *DenStream*). Isto permite ao algoritmo uma resposta melhor a formatos arbitrários do conjunto de dados em relação ao formato globular do *CluStream*.

Em sua fase de aprendizagem, o algoritmo forma a grade para poder calcular as densidades. A partir da grade definida, verifica-se pedaços do grid que sejam densos suficientes para formação de agrupamentos. Partes densas que sejam vizinhas são unidas em um único grupo. O processo de agrupamento utilizando a grade e a junção de *clusters* baseado na densidade pode ser visto na Figura 7.

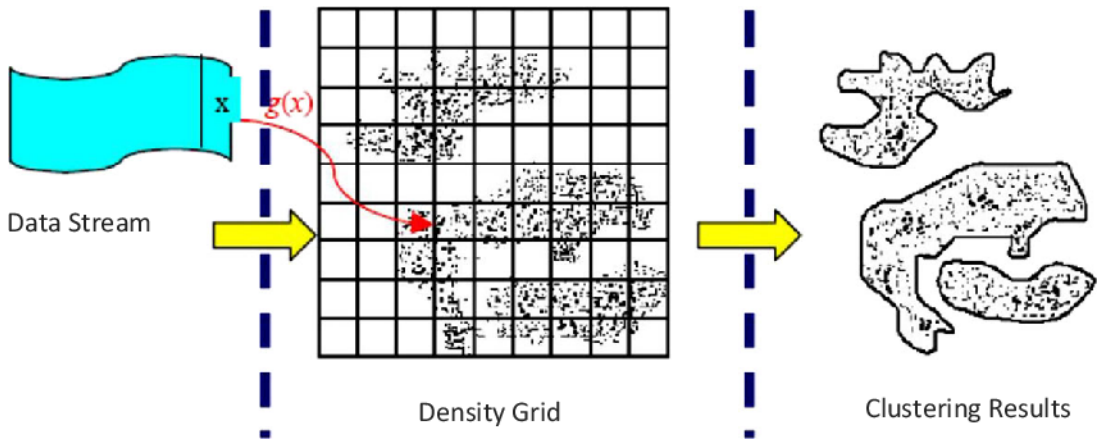


Figura 7 – Agrupamentos gerados pelo *D-Stream*

Fonte: Chen e Tu (2007).

Na fase *online* o algoritmo recebe novas amostras e insere nas áreas correspondentes para suas características. Além disso, são verificadas áreas no grid com baixas densidades

(denominadas esporádicas) para remoção, sendo este o tratamento adotado para retirada de dados anômalos.

4.5 *DenStream*

O *DenStream* foi proposto por Cao et al. (2006) como um algoritmo de aprendizagem não-supervisionado especializado em mineração de dados. Devido ao cenário para o qual ele foi proposto, uma de suas principais preocupações eram a adaptação a evolução do fluxo de dados. Por ser voltado para o *data stream*, outra preocupação é o uso de memória visto que os dados trafegados através dos fluxos tendem a ser quantidades massivas, podendo ser consideradas praticamente infinitos (SILVA et al., 2013).

Para a parte de aprendizagem, é realizado um agrupamento baseado no *Density-based Spatial Clustering of Applications With Noise* (*DBScan*) em busca dos primeiros agrupamentos. Seus parâmetros principais são:

- *epsilon*: a distância máxima para dois pontos serem considerados similares;
- *minPoints*: a quantidade mínima de pontos similares para se iniciar um *cluster*.

Uma melhoria proposta por Ozkok (2017) ao *DBScan* faz com que se tenha uma estimativa do melhor valor de *epsilon* através do parâmetros *minPoints*. Isso faz com que o algoritmo seja mais adaptável, buscando o melhor *epsilon* para os dados passados para o algoritmo. Esta estimativa é feita utilizando as distâncias obtidas entre todos os dados utilizados na fase agrupamento. O *epsilon* é atribuído a menor distância entre pontos que supere a média das distâncias somada com o desvio padrão.

Por o *DBScan* (e por consequência, o *DenStream*) ser um algoritmo baseado em densidade, ele é indiferente a forma que os dados estão espalhados, se preocupando apenas com a densidade e não com a forma do grupo. Isto faz com que o algoritmo seja mais facilmente adaptável a cenários com dados um pouco mais destoantes. Os agrupamentos gerados para diferentes distribuições, destacando assim sua adaptabilidade e sua detecção de *outliers*, pode ser vista na Figura 8.

Figura 8 – Agrupamentos gerados pelo *DBScan*.

Fonte: Elaborada pelo próprio autor baseado no site do *Sci-kit Learn*[3].

Após a formação dos *clusters* a partir do *DBScan*, as amostras novas que chegam são verificadas para avaliar se podem ser incorporadas em algum dos grupos existentes. Se não, são marcados como anômalos e armazenados por um tempo. O algoritmo permite que caso existam vários *outliers* similares seja criado um novo grupo, denominado *outlier's clusters*, o que permite a adaptação a novos cenários

[3] Disponível em: <https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html>

5 Metodologia

Este capítulo apresenta a metodologia usada para fazer a experimentação do projeto, descrevendo o *dataset*, a arquitetura desenvolvida para os testes, os algoritmos utilizados para pré-processamento e o fluxo de execução dos algoritmos utilizados na experimentação.

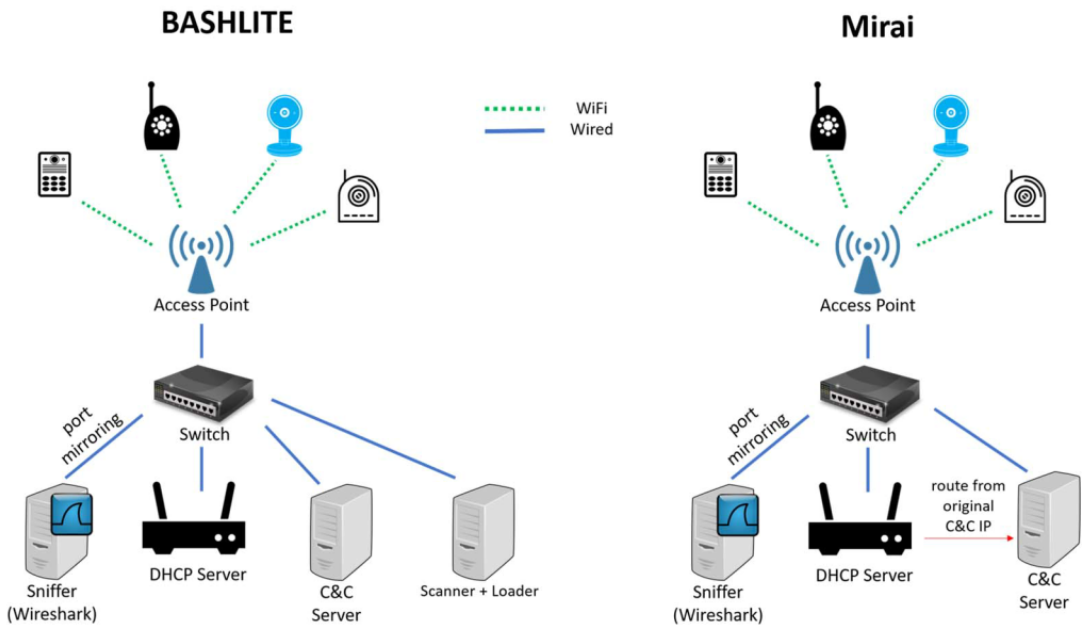
5.1 *Dataset* utilizado

Para avaliar o desempenho dos algoritmos propostos no cenário de *IoT* desejado, foi utilizado o conjunto de dados fornecido pelo trabalho de Meidan et al. (2018). Neste foi gerado um *dataset* de nove dispositivos diferentes sendo expostos a até duas variantes de infecções de *botnets*: *Mirai* e *Bashlite*. As quantidades de amostras obtidas podem ser vistas na Tabela 2.

	Amostras		
Dispositivo	Benigno	Mirai	Bashlite
Danmini Doorbell	49.548	107.685	29.849
Ennio Doorbell	39.100	0	28.120
Ecobee Thermostat	13.113	43.192	27.494
Philips B120N/10 Baby Monitor	175.240	103.621	27.859
Provision PT-737E Security Camera	62.154	96.781	29.297
Provision PT-838 Security Camera	98.514	97.096	28.397
Samsung SNH 1011 N Webcam	52.150	0	27.698
SimpleHome XCS7-1002-WHT Security Camera	46.585	45.930	27.825
SimpleHome XCS7-1003-WHT Security Camera	19.528	43.674	28.572

Tabela 2 – Infecções em cada dispositivo. Adaptado de Meidan et al. (2018).

Para conseguir capturar essas amostras, os autores construíram uma arquitetura, visto na Figura 9, devidamente isolada e coletou as amostras benignas a partir da ferramenta *Wireshark*. Após a coleta das amostras benignas, os dispositivos foram aos poucos contaminados com o *malware* específico e coletado as amostras devidamente separadas e classificadas.



Fonte: Meidan et al. - N-baiot: Network-based detection of iotbotnet attacks using deep autoencoders.

Figura 9 – Arquitetura proposta por Meidan et al. (2018)

Apesar de possuir diversos ataques utilizando essa infecções, foi considerado apenas os dados do ataque descrito no trabalho de Meidan et al. (2018) como "*scan*", responsável por tentar infectar outros dispositivos da rede, visto que o objetivo dos algoritmos é impedir a formação da *botnet*.

Baseado nisso, o *dataset* possui dados sobre o tráfego normal dos nove dispositivos, além de dezesseis conjuntos de dados de tentativas de infecção dos elementos da rede. Com isso possuímos um conjunto de dados com 555.932 amostras benignas (41%) e 793.090 amostras de tentativas de infecção (59%), distribuídas entre 537.979 da variante *Mirai* (67,83%) e 255.111 da variante *Bashlite* (32,17%).

Os dados foram gerados usando quatro características do *data stream*, visto na Tabela 3. A partir desses dados, foram utilizados transformações matemáticas, gerando assim 23 atributos.

Valor	Estatística	Dado agregado	Quantidade
Tamanho do Pacote *Outbound*	Média, Variância	IP origem, MAC-IP, Canal, *Socket*	8
Pacote	Número	IP origem, MAC-IP, Canal, *Socket*	4
Jitter	Média, Variância e Número	Canal	3
Tamanho do Pacote	Magnitude, Raio, Covariância, Coeficiente Correlacional	Canal, *Socket*	8

Tabela 3 – Conjunto de características do *dataset*.

Fonte: Meidan et al. (2018).

Estas características foram fornecidas para o extrator de características da ferramenta Kitsune (MIRSKY et al., 2018), o que permitiu gerar dados a partir da janela com decaimento,

usando como fator de decaimento 5 possíveis valores (5, 3, 1, 0,1 e 0,01), gerando assim o total de 115 características.

5.2 Arquitetura desenvolvida para os testes

Para viabilizar o uso do *dataset* do trabalho de Meidan et al. (2018), foi montado uma arquitetura hipotética baseado na plataforma de captura criada para obter o *dataset*. Essa arquitetura possibilitou utilizar os dados fornecidos simulando o cenário similar ao que foi utilizado no experimento original.

O conjunto de dados que fica disponível a partir do *sniffer* utilizando a ferramenta *Wireshark*, é disponibilizado num servidor *Redis*. Este é utilizado como cache de dados e serviço de mensageria para só então poder ser consumido pela aplicação. O uso do cache e do serviço de mensageria permite que sejam guardados dados enquanto não são processados pelos algoritmos. Isto permite que efetivamente todo novo dado seja devidamente analisado e rotulado.

Além disso, serviços de mensageria permitem também que, quando se chegue novos dados para serem processados, as aplicações registradas para consumir sejam automaticamente notificadas. A adição do servidor *Redis* constitui a mudança para a arquitetura hipotética e permite a utilização do *dataset*, podendo ser vista na Figura 10.

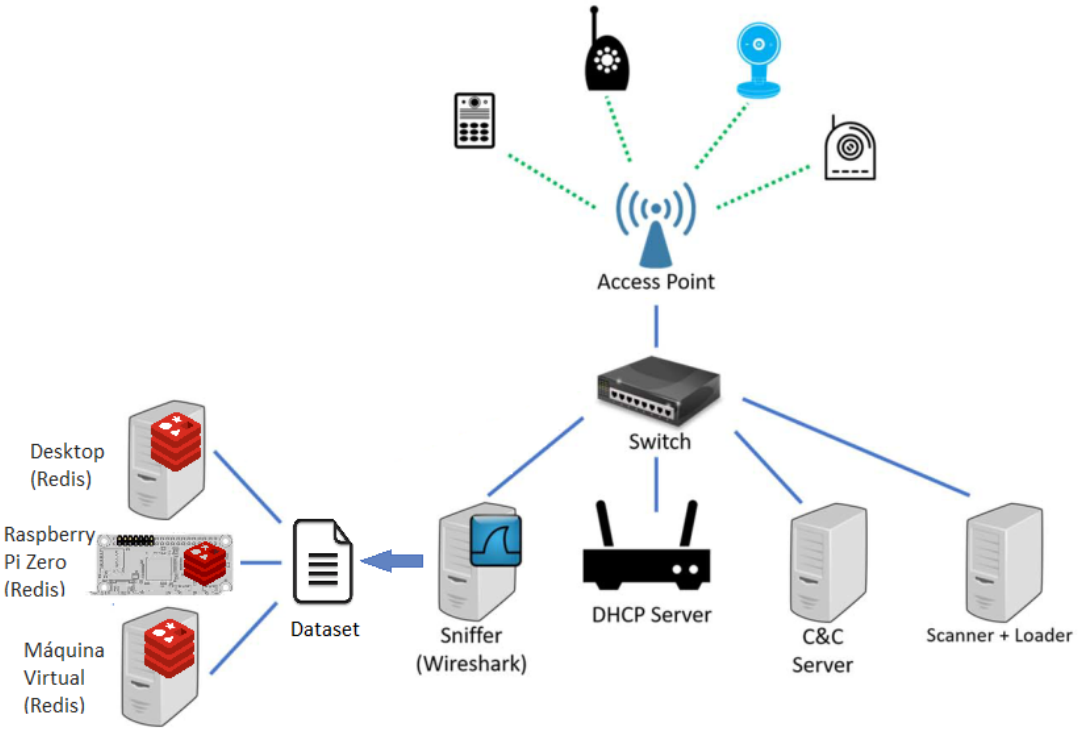


Figura 10 – Arquitetura proposta nesse trabalho.

Fonte: Elaborada pelo autor.

5.3 Pré-processamento

Para melhorar a acurácia geral do processo, os dados foram pré-processados, tanto antes do treinamento, quanto antes de repassar para o modelo obtido. Isto permite que as informações processadas sejam mais representativas e concisas, melhorando o desempenho por evitar que dados correlatos ou de menor importância sejam repassados para o algoritmo, assim como permite que os atributos importantes do dado sejam realçados. Os algoritmos utilizados neste experimento foram o *Principal Component Analysis* e o *Local Outlier Factor*.

5.3.1 *Principal Component Analysis*

O *Principal Component Analysis* (*PCA*) foi criado por Pearson (1901) e tem como objetivo diminuir a dimensionalidade de um dado, identificando variáveis correlacionadas e transformando em características não correlacionadas através de transformações matemáticas. A diminuição de dimensionalidade é interessante por permitir que variáveis correlacionadas não sejam utilizadas pelo algoritmo na aprendizagem, tornando o algoritmo mais leve e com menor processamento. Quando há uma correlação entre variáveis, o comportamento destas é previsível em algum nível, por existir uma dependência entre estas. Logo o mesmo tipo de aprendizagem que é obtido com uma variável também obtido com qualquer outra correlata, sendo assim uma repetição desnecessária.

Outro aspecto interessante são as transformações matemáticas realizadas nos dados. Com isto, há uma extração melhor das informações, permitindo que dados que influenciam de forma significativa tenham maior peso e aqueles menos significativos tenham menor peso. Com isso, é possível diminuir a quantidade de características usadas de forma considerável, favorecendo mais uma vez a leveza e menor processamento dos algoritmos. Um exemplo de transformação matemática seria a rotação do eixo cartesiano para buscar as maiores variações em um conjunto de dado, como visto na Figura 11.

Este algoritmo foi utilizado tanto no pré-processamento dos dados de treinamento quanto na análise de novas amostras (fase *online*) da aprendizagem de máquina, diminuindo das 115 características presentes no conjunto de dados apresentados na seção 5.1 para uma única característica. Isto permite que seja recebido apenas a melhor característica, tanto para treino quanto para análise. Isto permite um melhor agrupamento final e um menor custo de processamento.

5.3.2 *Local Outlier Factor*

O *Local Outlier Factor* (*LOF*) foi proposto no trabalho de Breunig et al. (2000) e tem como objetivo encontrar dados anômalos em *datasets* multidimensionais considerando a

[1] Disponível em: <https://www.visiondummy.com/2014/05/feature-extraction-using-pca/>.

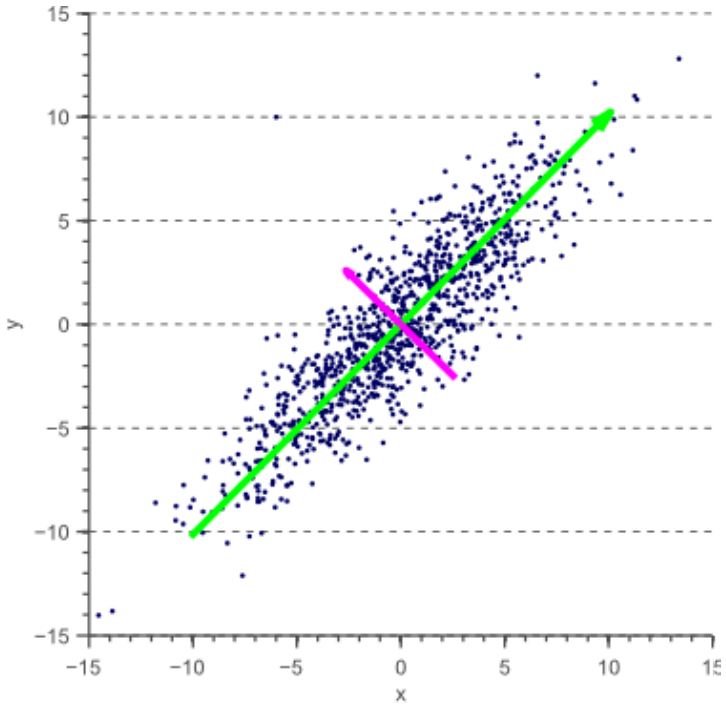


Figura 11 – Demonstração gráfica do *PCA*

Fonte: *Feature extraction using PCA - Computer vision for dummies* [1].

densidade dos agrupamentos dos dados. O conceito de *outlier* usado pelo autor para construção do algoritmo é:

> Um *outlier* é uma observação que desvia tanto das outras observações que levantam suspeitas de ter sido geradas por um mecanismo diferente (HAWKINS, 1980).

Geralmente algoritmos que se baseiam em densidade tem respostas melhores para conjunto de dados adversos por não ser sensível à distribuição geométrica dos dados. Esse tipo de característica faz com que o algoritmo tenha uma resposta mais consistente pra qualquer tipo de dado inserido, tendo uma resposta melhor para cenários imprevistos.

O *LOF* foi o segundo passo do pré-processamento sendo apenas utilizado para eliminar qualquer possibilidade de anomalias nos dados benignos na parte de treinamento (*offline*) dos algoritmos. Ao extrair de forma mais consistente as características comuns do *stream* de dados de cada dispositivo, foram eliminadas amostras que possuam características muitos distintas. Um exemplo pode ser visto na Figura 12 onde, ao eliminar os dados anômalos, é visto que restam apenas dois agrupamentos bem definidos.

O principal parâmetro para o *LOF* é o *minPts*, mesmo parâmetro utilizado em algoritmos como *DBScan* e o *DenStream*, responsável por sinalizar quantos dados próximos são necessários para criar um agrupamento. É proposto por (BREUNIG et al., 2000) uma heurística para melhor

[2] Disponível em: <https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html>.

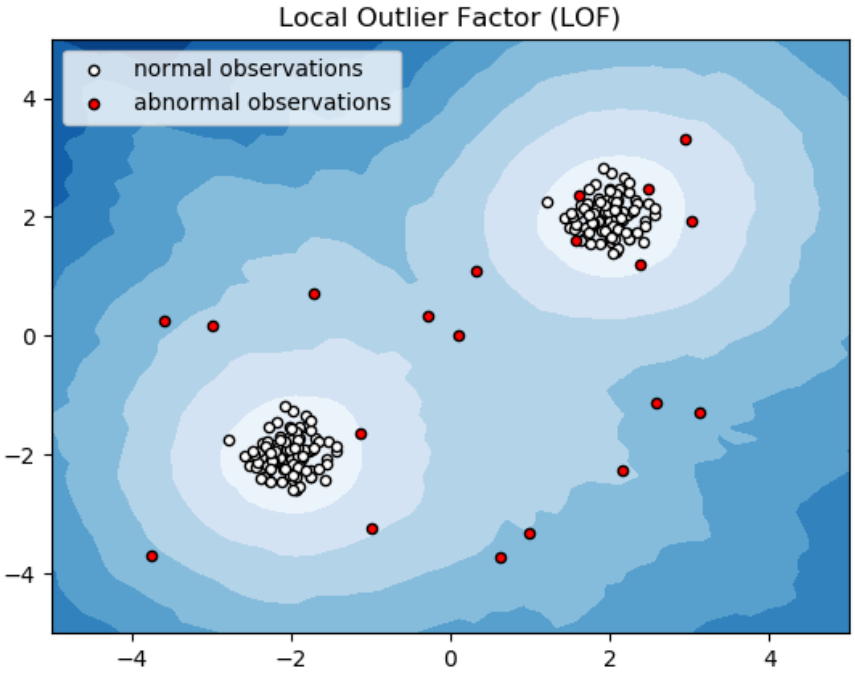


Figura 12 – Demonstração gráfica do *LOF*

Fonte: *Outlier detection with Local Outlier Factor (LOF) - Sci-kit SKLearn*[2].

estimar uma faixa de valores possíveis, e a partir desta faixa, testar para todos os valores possíveis e ver qual retorna o melhor resultado.

5.4 Fluxo de execução dos algoritmos

Para este trabalho, todos os algoritmos serão utilizados como algoritmos de detecção de anomalia, ou seja, os algoritmos verificam se o novo dado pode ser encaixado em algum grupo pré-existente e o classifica em normal ou anômalo. Os primeiros *clusters* serão considerados os agrupamentos do comportamento normal da rede e, a partir destes, qualquer dado anômalo é considerado malicioso.

O fluxo inicia obtendo as amostras necessárias para o treinamento, sendo estes dados pré-processados com o *LOF* e o *PCA*. Finalizando a parte *offline*, o algoritmo a ser avaliado gera os *clusters* referentes ao tráfego normal da rede. Para a parte *online*, enquanto novas amostras são fornecidas ao algoritmo, onde será aplicado o *PCA* e avaliará se o dado é anômalo, sendo assim considerado malicioso, ou normal, sendo considerado benigno. O fluxograma da execução dos algoritmos pode ser visto na Figura 13.

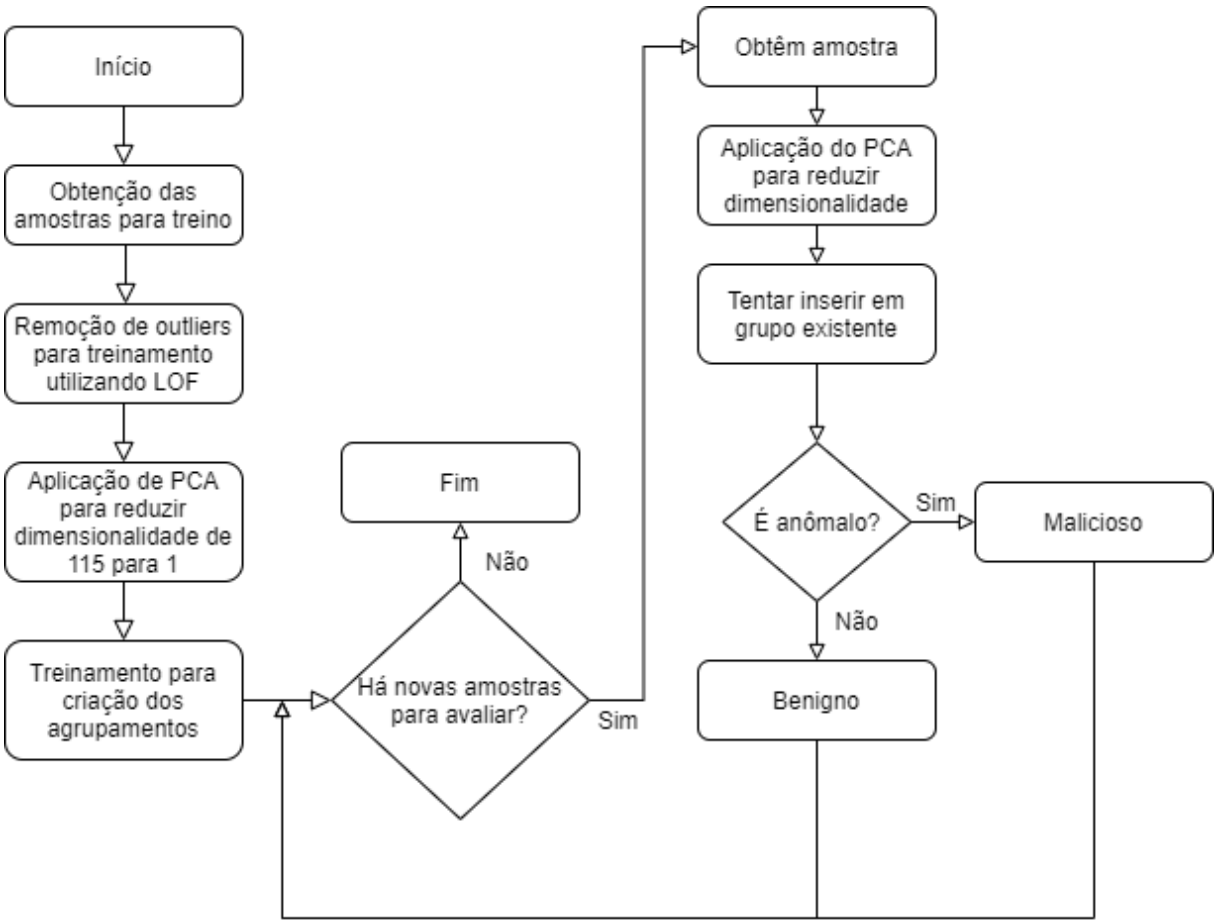


Figura 13 – Fluxo de execução dos algoritmos.

Fonte: Elaborada pelo autor.

6

Experimentação e análise qualitativa

Este capítulo apresentará a experimentação proposta para análise qualitativa dos algoritmos *BIRCH*, *CluStream*, *DenStream* e *DStream* e a análise dos resultados obtidos. Estas comparações visam determinar o quão eficaz cada algoritmo é em determinar se um dado é malicioso ou benigno.

6.1 Descrição do experimento

Para essa experimentação foi utilizado um dispositivo desktop com um processador *i5 5200U* com 4 *threads* operando a 2,2 GHz, 16 GB de *RAM DDR3* com sistema operacional *Windows 10*. Este dispositivo foi escolhido para remover qualquer restrição dos algoritmos, visto que o objetivo deste experimento é verificar a qualidade do resultado, e não o seu comportamento diante cenários restritos. A arquitetura montada para o experimento pode ser visto na Figura 14, onde o *Redis* disponibiliza o conjunto de dados para processamento dos algoritmos no próprio dispositivo.

Para esta experimentação, foram executadas 40 rodadas com amostras aleatórias escolhidas a partir do *dataset* do trabalho de Meidan et al. (2018) descrito na seção 5.1. Foram considerados todos os nove dispositivos disponibilizados e os 16 ataques presentes.

Foram fornecidos para treinamento dos algoritmos 100 amostras benignas para os quatro algoritmos, todos tendo o mesmo conjunto de dados para treino. Com os modelos prontos, foi fornecido um conjunto balanceado de amostras benignas e maliciosas para avaliar os algoritmos. Os quatro modelos recebiam o mesmo dado por vez.

A quantidade de rodadas para o primeiro experimento foi estabelecida considerando que todos os dispositivos teriam uma quantidade de amostras balanceada de benignos e maliciosos, permitindo um total de 15.701 amostras avaliadas por rodadas. Como os dados são distribuídos em dois grupos (normais e anômalos), pode-se assumir uma distribuição de Bernoulli.

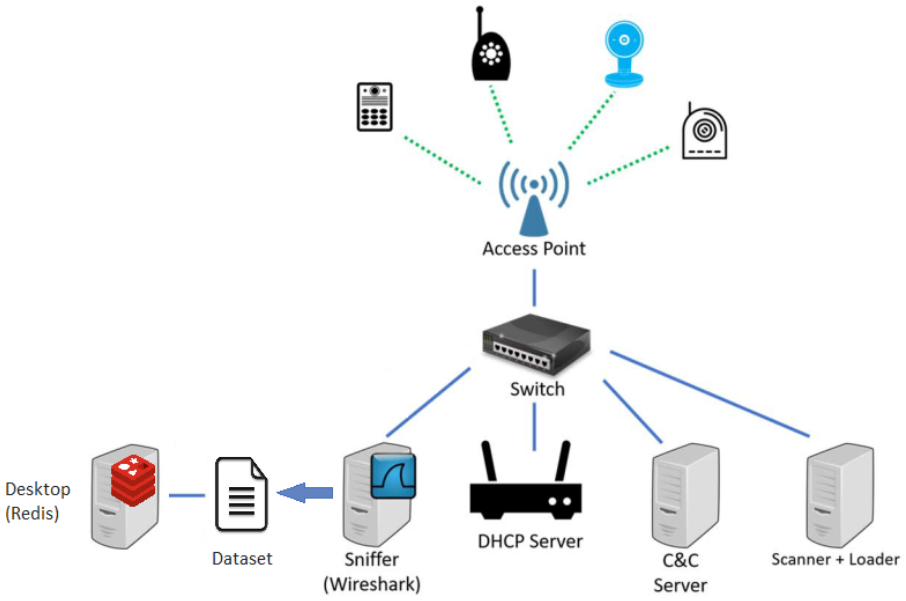


Figura 14 – Arquitetura utilizada na experimentação qualitativa.

Fonte: Elaborada pelo autor.

Executando 40 rodadas com as amostras sendo escolhidas aleatoriamente, foram avaliados 628.040 dados, onde de 1.349.025 amostras disponibilizadas no *dataset*, obtêm-se uma margem de erro aproximada de 0,16% com um nível de confiança de 99%. A fórmula para obtenção da margem de erro (EVANS; ROSENTHAL, 2009) pode ser vista na Equação 6.1.

$$e = Z \cdot \sqrt{\frac{p \cdot (1-p)}{n}} = 2,575 \cdot \sqrt{\frac{0,41 \cdot 0,59}{628.040}} \approx 0,16\% \tag{6.1}$$

O objetivo ideal seria bloquear todo dado maligno a permitir o fluxo benigno. Para avaliar o experimento, considerando dados benignos como positivos e dados malignos como negativos, medidas como taxa de verdadeiros positivos e verdadeiros negativos são importantes.

A partir destas e de medidas complementares como taxa de falso positivos e negativos, foi utilizado a matriz de confusão para avaliar melhor o resultado, permitindo avaliar dados como acurácia. Com isso, as seguintes métricas foram levantadas:

- Taxa de verdadeiros positivos ou sensibilidade (TPR): avalia quantos dos dados benignos foram identificados pelos algoritmos como benigno. Quanto maior a taxa, melhor a identificação dos benignos pelos algoritmos;
- Taxa de verdadeiros negativos ou especificidade (TNR): avalia quantos dos dados maliciosos foram identificados pelos algoritmos como malicioso. Quanto maior a taxa, melhor a identificação dos maliciosos pelos algoritmos;
- Taxa de falsos positivos (FPR): avalia quantos dos dados maliciosos foram identificados pelos algoritmos como benigno. Quanto menor a taxa, menor a chance de um dado

malicioso ser classificado como benigno;

- Taxa de falsos negativos (FNR): avalia quantos dos dados benignos foram identificados pelos algoritmos como maliciosos. Quanto menor a taxa, menor a chance de um dado benigno ser classificado como malicioso;
- Valor preditivo positivo ou precisão (PPV): avalia quantos dos dados benignos identificados pelos algoritmos eram de fato benignos. Quanto maior o valor, maior a chance de um dado classificado como benigno ser efetivamente benigno;
- Taxa de falsa descoberta (FDR): avalia quantos dos dados benignos identificados pelos algoritmos eram maliciosos. Quanto menor o valor, menor a chance de um dado classificado como benigno ser dado malicioso;
- Taxa de falsa omissão (FOR): avalia quantos dos dados malignos identificados pelos algoritmos eram benignos. Quanto menor o valor, menor a chance de um dado classificado como maligno ser um dado benigno;
- Valor preditivo negativo (NPV): avalia quantos dos dados maliciosos identificados pelos algoritmos eram de fato maliciosos. Quanto maior o valor, maior a chance de um dado classificado como malicioso ser efetivamente malicioso;
- Acurácia: Combinação de acerto dos algoritmos, considerando todos os verdadeiros positivos e negativos em relação a população total. Quanto maior a acurácia, mais exato o algoritmo é;
- *F1 score*: Valor entre 0 e 1, resultante da combinação do PPV e sensibilidade. Quanto maior o resultado, melhor o algoritmo.

6.2 Análise das matrizes de confusão

Apesar do *CluStream* ser rápido e eficiente por se basear na distância radial, ele não apresenta bons resultados em distribuições globulares de dados (CAO et al., 2006; NIXON; SEDKY; HASSAN, 2019). Esta limitação o torna pouco eficaz em distribuição mais aleatórias e pouco adaptativo para um *data stream* por sempre depender que o fluxo se mantenha numa distribuição circular ou elipsoide.

O *CluStream* foi o algoritmo que teve o pior desempenho neste conjunto de dados, tendo uma acurácia na média inferior a 50%, com um *F1 score* próximo de 0,1, se mostrando inadequado para esse tipo de cenário. A matriz de confusão pode ser vista na Tabela 4 e sua dispersão em comparação com o *BIRCH*, *DenStream* e *DStream* na Figura 15.

Outra questão evidenciada, agora a partir do gráfico de caixa visto na Figura 15, é o seu problema de mínimo-local devido a aleatoriedade da inicialização dos centróides iniciais. Este

		Valores Reais			
		Positivo	Negativo		
Predição	Positivo	23.352	79.299	PPV: 22,75%	FDR: 77,25%
	Negativo	294.288	231.101	FOR: 56,01%	NPV: 43,97%
		TPR: 7,35%	FPR: 25,55%	Acurácia: 40,52%	F1: 0,11
		FNR: 92,65%	TNR: 74,45%		

Tabela 4 – Matriz de confusão do CluStream a partir de todas as 40 rodadas.

Fonte: Elaborada pelo autor.

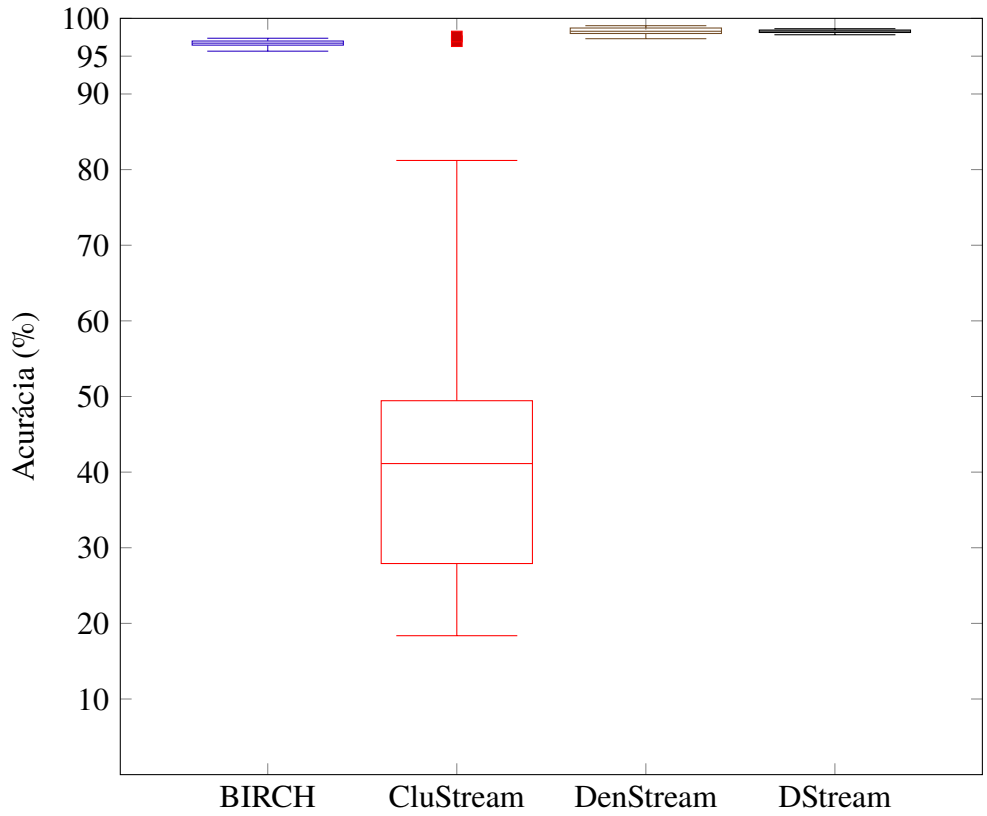


Figura 15 – *Box plot* a partir da acurácia de todas as rodadas.

Fonte: Elaborada pelo autor.

se dá pelo *CluStream* se basear no *K-means*, amplamente conhecido, categorizado como um problema NP-difícil (Garey; Johnson; Witsenhausen, 1982; ALOISE et al., 2009; MAHAJAN; NIMBHORKAR; VARADARAJAN, 2009). Pelo gráfico, é possível ver que houveram rodadas onde o algoritmo conseguiu se encaixar em um mínimo global e conseguiu resultados acima de 95%. Porém por ocorrerem poucas vezes, estes são considerados resultados anômalos.

Já para os algoritmos que se baseiam em densidade tiveram características interessantes para cada um deles. O *BIRCH* foi o que teve a melhor taxa de TNR e FPR de todos os algoritmos, confirmada também pelo alto PPV e baixo FDR. Isto mostra que, dentre todos os algoritmos, foi o que possuiu maior capacidade de, corretamente, caracterizar um dado malicioso. Porém, este possuiu a menor taxa de acurácia (96,71%), tendo assim a maior taxa dos algoritmos baseados

em densidade de falsos negativos (6,33%) e uma taxa menor de verdadeiros positivos (93,67%).

		Valores Reais			
		Positivo	Negativo		
Predição	Positivo	297.534	555	PPV: 99,81%	FDR: 0,19%
	Negativo	20.106	309.845	FOR: 6,09%	NPV: 93,91%
		TPR: 93,67%	FPR: 0,18%	Acurácia: 96,71%	F1: 0,97
		FNR: 6,33%	TNR: 99,82%		

Tabela 5 – Matriz de confusão do *BIRCH* a partir de todas as 40 rodadas.

Fonte: Elaborada pelo autor.

Já o *DenStream* foi o algoritmo que possuiu a maior média de acurácia entre todos (98,31%), como visto na Tabela 6. Este também teve as maiores acurácias considerando individualmente as rodadas como visto na Figura 16. O algoritmo é baseado no *DBScan* e foi aplicado a melhoria proposta no Ozkok (2017) de cálculo do *epsilon* visto na seção 4.5. Isso permitiu pra ele uma melhor adaptação ao cenário de amostras aleatórias do conjunto de dados.

		Valores Reais			
		Positivo	Negativo		
Predição	Positivo	309.652	2.632	PPV: 99,16%	FDR: 0,84%
	Negativo	7.988	307.768	FOR: 2,53%	NPV: 97,47%
		TPR: 97.49%	FPR: 0.85%	Acurácia: 98,31%	F1: 0,98
		FNR: 2,51%	TNR: 99,15%		

Tabela 6 – Matriz de confusão do *DenStream* a partir de todas as 40 rodadas.

Fonte: Elaborada pelo autor.

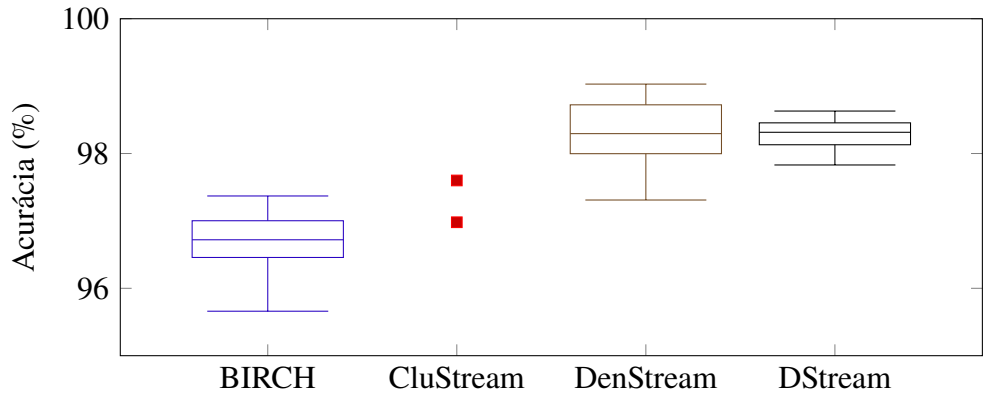


Figura 16 – *Box plot* entre 95% e 100%.

Fonte: Elaborada pelo autor.

Por fim, o *DStream* tem uma acurácia levemente inferior ao *DenStream* (98,29%). Entretanto, a diferença é de apenas 0,02%, com faixas próximas de falsos positivos e negativos também. Considerando a margem de erro deste experimento, pode-se levar a conclusão de que

estes algoritmos possuem desempenho similares. Este dado também é corroborado por possuírem *F1 score* similares. Sua matriz de confusão pode ser vista na Tabela 7.

		Valores Reais			
		Positivo	Negativo		
Predição	Positivo	310.329	3.438	PPV: 98,9%	FDR: 1,1%
	Negativo	7.311	306.962	FOR: 2,33%	NPV: 97,67%
		TPR: 97,7%	FPR: 1,11%	Acurácia: 98,29%	F1: 0,98
		FNR: 2,3%	TNR: 98,89%		

Tabela 7 – Matriz de confusão do *DStream* a partir de todas as 40 rodadas.

Fonte: Elaborada pelo autor.

Entretanto, a dispersão observada é menor que a do *DenStream*, podendo ser vista no gráfico disposto na Figura 16. Isso faz com que a taxa de acerto seja mais constante e previsível para o *DStream* devido a sua boa adaptabilidade à aleatoriedade dos dados, por mais que o *DenStream* tenha resultados superiores em algumas rodadas pontuais. As tabelas com os dados que originaram os gráficos e matrizes de confusão apresentados durante esta seção estão presentes no anexo A.

6.3 Análise comparativa

Como observado para o *CluStream*, obtendo valores de acurácia média 40,52% e de *F1 score* 0,11, este algoritmo se mostra pouco adequado para a distribuição de dados do cenário proposto.

O *BIRCH* teve as melhores taxas de TNR, FPR, PPV e FDR, todas essas relacionadas a detecção de dados maliciosos, considerando estes como *outliers*. Entretanto por ter obtido a segunda menor acurácia, a frente apenas do *CluStream*, levanta a hipótese de que a aplicação deste algoritmo em uma rede real pode levar a uma queda na qualidade de serviço da rede, visto que muitos dados benignos seriam perdidos por serem considerados maliciosos.

Quanto aos algoritmos *DStream* e *DenStream*, ambos tiveram resultados similares, com *F1 score* iguais e métricas com valores próximos. O *DenStream* obteve resultados individuais melhores em algumas rodadas, porém a um custo de uma variância maior na acurácia no decorrer de todas as rodadas. Já o *DStream* obteve a menor variância entre todos os algoritmos, tendo assim um resultado de treinamento mais estável independente da aleatoriedade dos dados usados para o treinamento.

7

Experimentação e análise de desempenho

Este capítulo apresentará a experimentação proposta para análise de desempenho dos algoritmos *BIRCH*, *CluStream*, *DenStream* e *DStream* e a análise dos resultados obtidos. Estas comparações visam determinar o quão eficiente cada algoritmo é e como cada algoritmo se porta em cenários de restrições.

7.1 Descrição do experimento

Foram montadas configurações variando parâmetros de processadores e memória para verificar o impacto nos algoritmos. A arquitetura e disposição das configurações utilizadas neste experimento pode ser vista na Figura 17. Foram executados em três configurações diferentes:

1. Desktop *i5 5200U* com 4 *threads* operando a 2,2 GHz, 16 GB de *RAM DDR3* com sistema operacional *Windows 10*;
2. *Raspberry Pi Zero W* com 1 *thread* operando a 1 GHz, 512 MB de *RAM DDR2* com sistema operacional *DietPi*;
3. Máquina Virtual com 1 *thread* operando a 2,2GHz e 256 MB de *RAM DDR3* com sistema operacional *DietPi*;

Para esta experimentação, foram feitas 10 rodadas com amostras aleatórias para verificar velocidade e consumo de recursos de cada algoritmo individualmente. Esta experimentação tem como objetivo verificar o dispositivo mínimo para executar os algoritmos. Foi utilizado o mesmo conjunto de dados do trabalho de Meidan et al. (2018) descrito na seção 5.1. Considerando as 10 rodadas e que cada rodada foram utilizadas as 15.701 amostras, temos um total de 157.010 amostras analisadas. Em um estudo preliminar, foi obtido uma média ($\overline{x}$) de 9,16 ms e um desvio-padrão (s) de 7,78 ms. Com isso, a partir da Equação 7.1 (JAIN, 1991), obtêm-se que o

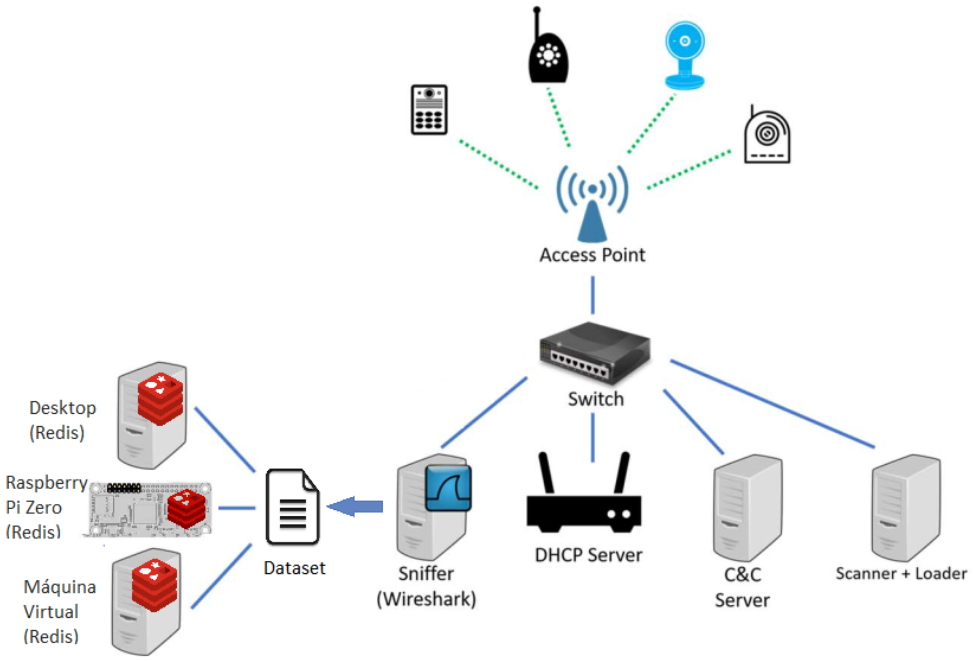


Figura 17 – Arquitetura utilizada na experimentação de desempenho.

Fonte: Elaborada pelo autor.

erro amostral, para um nível de confiança de 99%, é aproximadamente de 0,55%, permitindo um resultado mais confiável.

$$r = \frac{100 \cdot z \cdot s}{\overline{x} \cdot \sqrt{n}} = \frac{100 \cdot 2,575 \cdot 7,78}{9,16 \cdot \sqrt{157.010}} \approx 0,55\% \tag{7.1}$$

Foi utilizado como métrica o tempo gasto para treinamento e avaliação de cada nova amostra. O algoritmo, idealmente, deve ter uma resposta rápida para evitar a tempo que a infecção contamine ou se espalhe pela rede, formando a *botnet*. Portanto, quanto menor o tempo para gerar os modelos e avaliar novas amostras, melhor a adaptação para o uso.

7.2 Análise de desempenho - *Desktop*

Para esta primeira comparação, foi levado em consideração apenas as medições de tempo obtidas no cenário do *desktop*, um ambiente com menor restrição, servindo como base para as comparações feitas na próxima seção.

Para tentar diminuir os problemas do *CluStream*, foi considerado uma faixa de números de agrupamentos possíveis e encontrado o ponto de estabilização na curva da soma quadrática das distâncias. Isto garante localmente a menor quantidade de *clusters* necessários para diferenciar os dados da melhor forma.

Associado a isso, foi utilizado o algoritmo *k-means++* (ARTHUR; VASSILVITSKII,

2007), o qual busca melhorar o posicionamento dos centróides do *k-means*. Estes dois processos acabam aumentando o tempo de treinamento do *CluStream*. Tal tempo é muito superior aos outros, tendo uma média de 6,18 segundos, como visto na Figura 18.

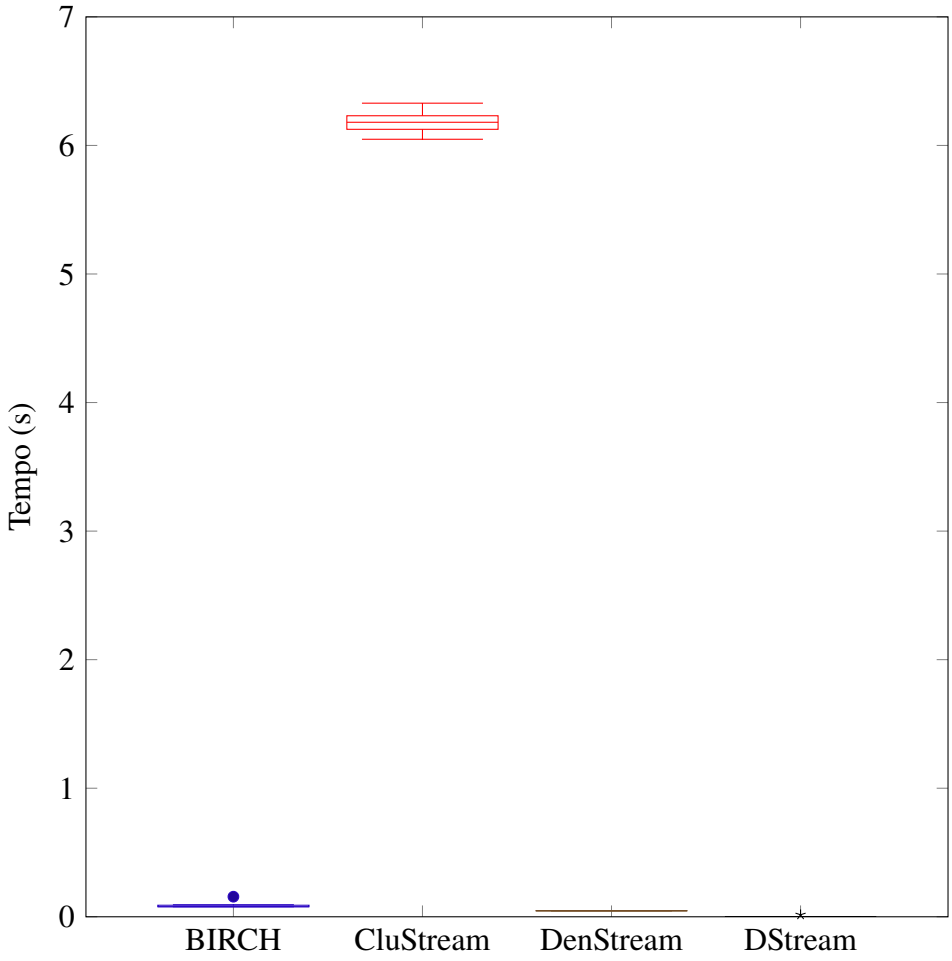


Figura 18 – *Box plot* a partir do tempo reportado de treinamento.

Fonte: Elaborada pelo autor.

Considerando os algoritmos baseados em densidade, todos tiveram tempos médios de treinamento inferiores a 100 ms, devido a necessidade de poucas amostras para treino. Destes, o *DStream* foi o que teve o menor tempo registrado, sendo suficiente para não ser reportado pelo algoritmo a nível de segundos. O *DenStream* teve uma média de treinamento de 46 ms e com baixa variação deste tempo entre as rodadas. Dentre estes, o mais lento foi o *BIRCH*, com uma média de 89 ms. O tempo para treinamento de cada algoritmo baseado em densidade pode ser visto na Figura 19.

Considerando o tempo de avaliação por amostra, o melhor desempenho foi do *DStream* com uma média de 10,40 ms. O *CluStream*, obteve uma média de 10,50 ms, podendo ter seu desempenho considerado similar ao do *DStream*.

O *BIRCH* obteve uma média de 11,45 ms por amostra avaliada. O mais lento foi o *DenStream* com uma média de 12,07 ms. Podemos ver ainda que a diferença entre o *DStream* e o

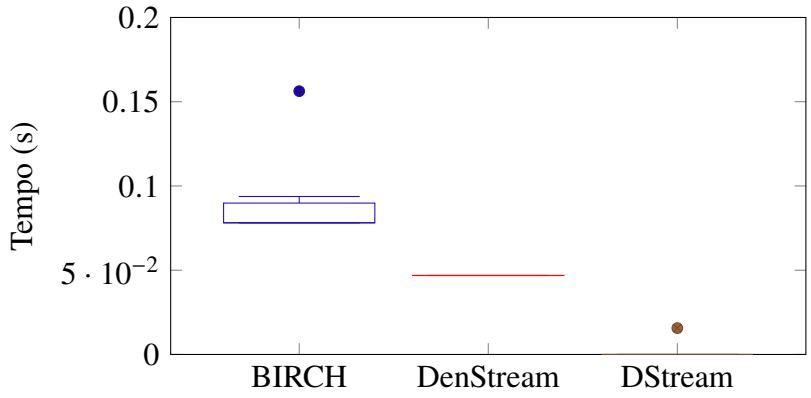


Figura 19 – *Box plot* a partir do tempo reportado de treinamento para os algoritmos baseados em densidade.

Fonte: Elaborada pelo autor.

DenStream por amostra foi de 1,67 ms, um aumento de 16% no tempo por amostra. O tempo de avaliação por amostra para cada algoritmo pode ser visto na Figura 20.

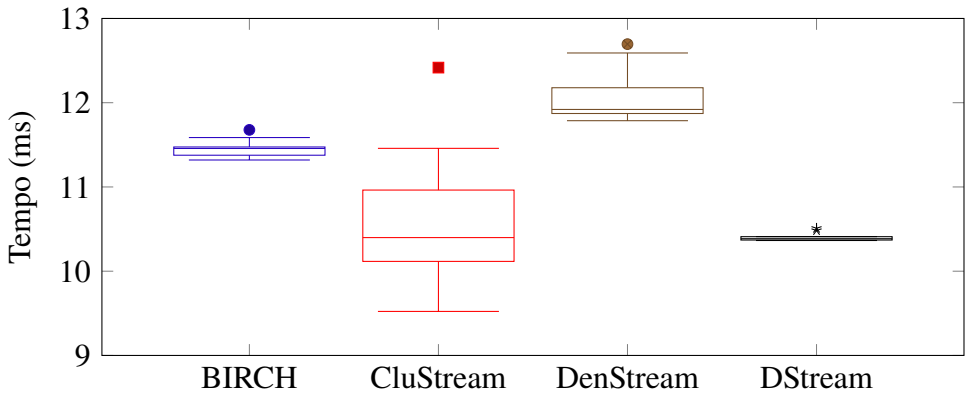


Figura 20 – *Box plot* a partir do tempo médio de avaliação por amostra.

Fonte: Elaborada pelo autor.

7.3 Impacto no tempo em relação à restrição em processamento e memória

Para esta análise, foram feitas as comparações com as configurações planejadas para verificar o efeito de mudanças de processamento e memória RAM disponível. Esta comparação é feita para poder avaliar os impactos com intuito de tentar verificar quais as condições ideais mínimas para executar de forma efetiva.

Todos os algoritmos apresentaram características similares como:

1. O treinamento e análise de cada amostra sempre foi mais lento na *Raspberry Pi Zero*;

2. O tempo de treinamento foi menor na configuração *desktop*;
3. O tempo de avaliação das amostras foi menor na configuração da máquina virtual;
4. Os tempos de treinamento e análise de cada amostra foi próximo entre a configuração do *desktop* e a máquina virtual.

Para o *CluStream*, a lentidão em relação ao treinamento devido ao tempo para estabilização da curva da soma do erro quadrático, apresentado na seção 7.2 se repetiu em todas as configurações. O gráfico comparativo para as configurações do *CluStream* pode ser visto na Figura 21.

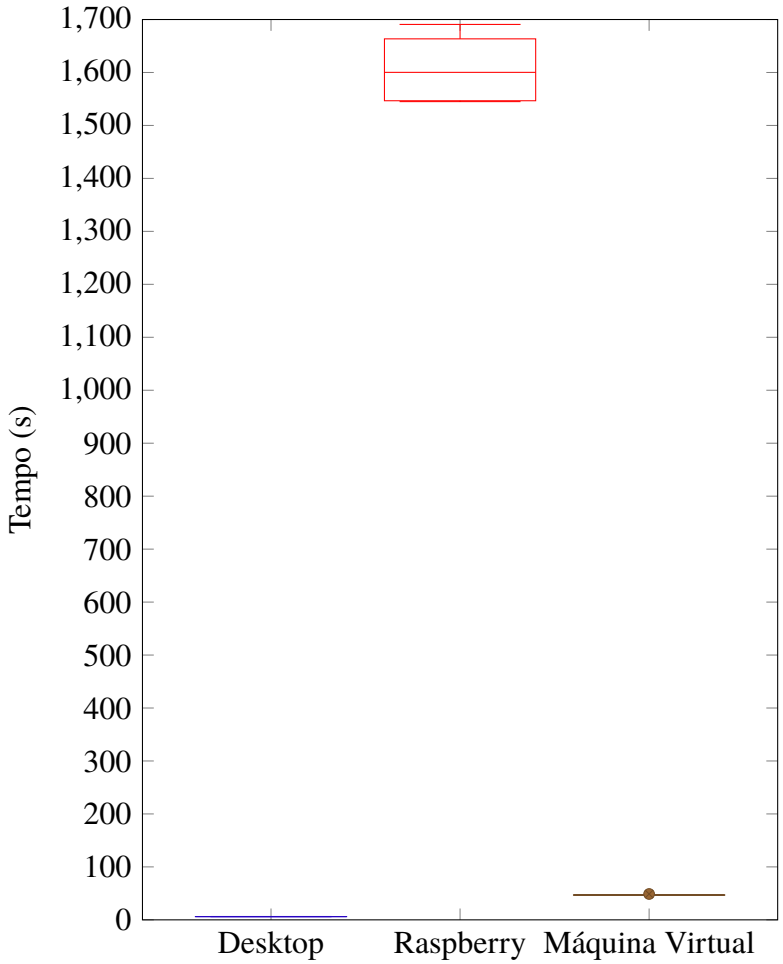


Figura 21 – *Box plot* do tempo de treinamento para o *CluStream* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

O aumento do tempo médio de treinamento entre o *desktop* (6,18 s) e a *Raspberry Pi Zero* (1.607,43 s) foi de 25.916%. Já a comparação entre o tempo médio do *desktop* e a máquina virtual (47,04 s) representou um aumento de 661,3%, visto na Figura 22.

Com relação a análise da amostra, o aumento médio do tempo de treinamento entre a máquina virtual (7,97 ms) e a *Raspberry Pi Zero* (324,58 ms) foi de 3.970,71%, como mostrado

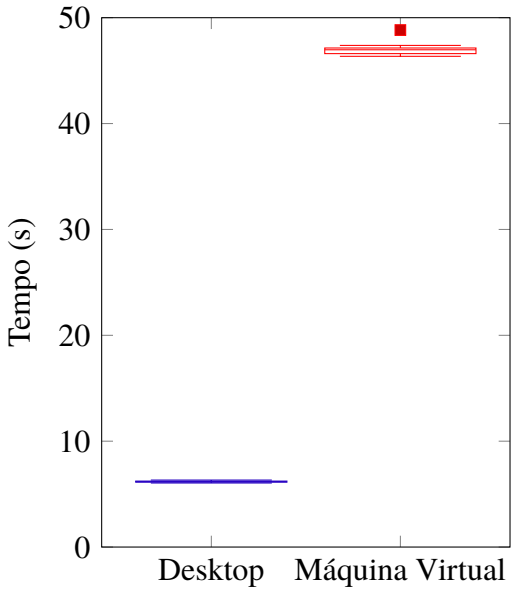


Figura 22 – *Box plot* do tempo de treinamento para o *CluStream* considerando a variação entre *desktop* e a máquina virtual.

Fonte: Elaborada pelo autor.

na Figura 23. Já a comparação entre a máquina virtual e o *desktop* (10,58 ms) representou um aumento de 32,71%, como pode ser visto na Figura 24.

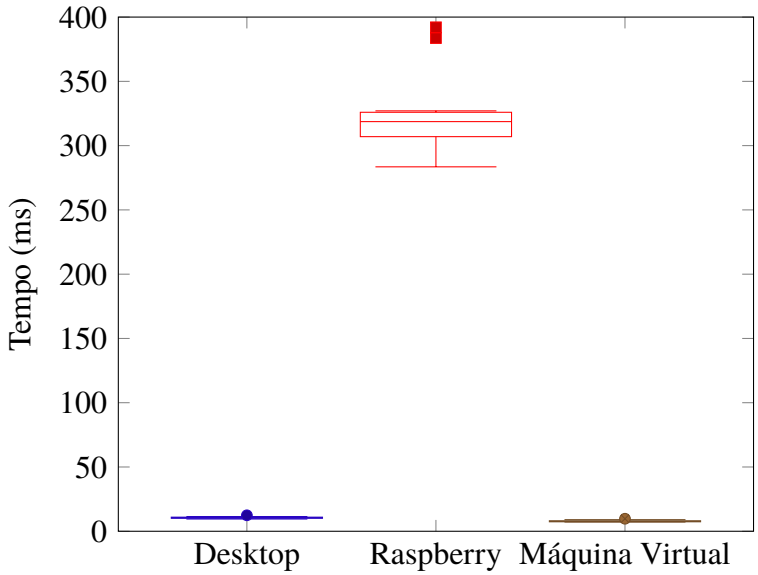


Figura 23 – *Box plot* do tempo de avaliação de cada amostra para o *CluStream* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

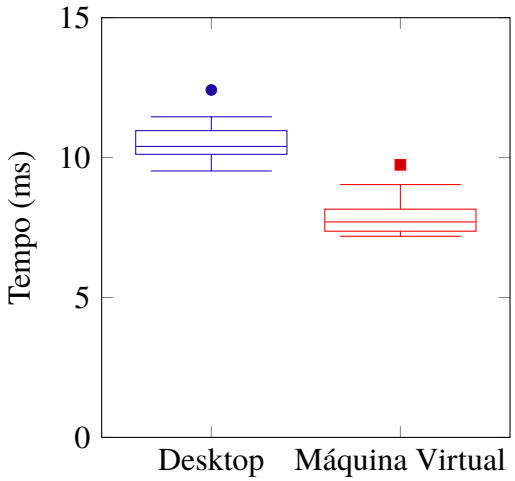


Figura 24 – *Box plot* do tempo de avaliação de cada amostra para o *CluStream* considerando a variação entre *desktop* e a máquina virtual

Fonte: Elaborada pelo autor.

O algoritmo *BIRCH* também apresentou grande aumento percentual na parte de treinamento. Apesar desse aumento, os tempos são mais aceitáveis que o *CluStream*, considerando que o treinamento foi feito com 100 amostras. O gráfico comparativo para as configurações utilizadas com o algoritmo *BIRCH* pode ser visto na Figura 25.

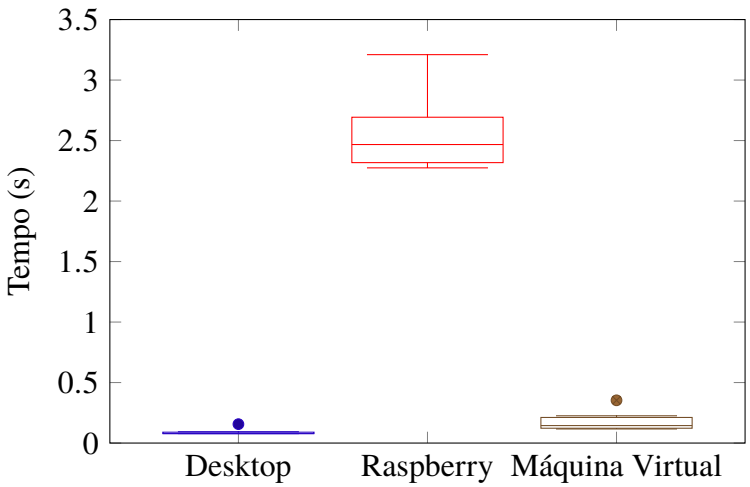


Figura 25 – *Box plot* do tempo de treinamento para o *BIRCH* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

O aumento do tempo médio de treinamento entre o *desktop* (0,09 s) e a *Raspberry Pi Zero* (2,55 s) foi de 2.760,99%. Já a comparação entre o tempo médio do *desktop* e a máquina virtual (0,17 s) representou um aumento de 96,34%.

Com relação a análise da amostra, o aumento médio do tempo de treinamento entre a máquina virtual (7,51 ms) e a *Raspberry Pi Zero* (302,20 ms) foi de 3.923,84%. Já a comparação

entre a máquina virtual e o *desktop* (11,45 ms) representou um aumento de 52,51%. O tempo para cada configuração para o *BIRCH* pode ser visto na Figura 26.

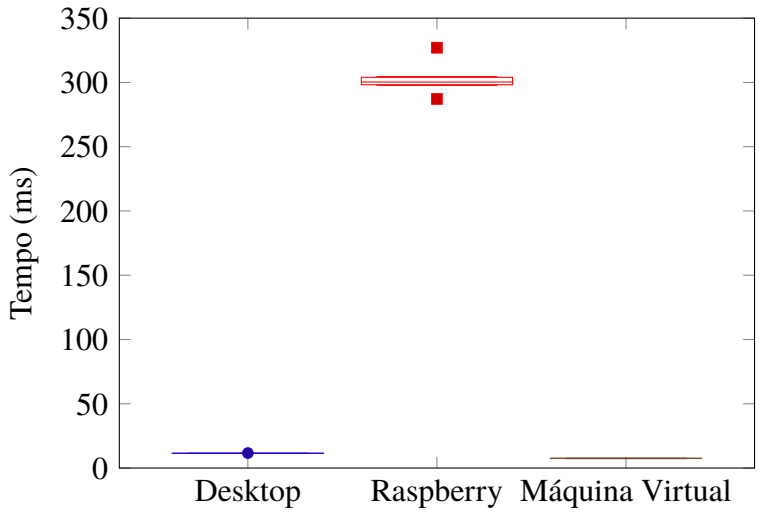


Figura 26 – *Box plot* do tempo de avaliação de cada amostra para o *BIRCH* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

O algoritmo *DenStream* também houve um aumento percentual grande na parte de treinamento, superior ao *BIRCH* e inferior ao *CluStream*. O gráfico comparativo para as configurações do *DenStream* pode ser visto na Figura 27.

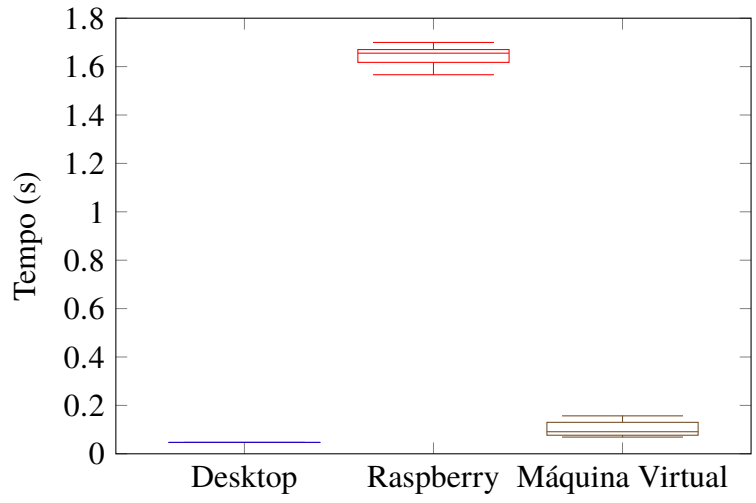


Figura 27 – *Box plot* do tempo de treinamento para o *DenStream* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

O aumento do tempo médio de treinamento entre o *desktop* (0,05 s) e a *Raspberry Pi Zero* (1,65 s) foi de 3.412,01%. Já a comparação entre o tempo médio do *desktop* e a máquina virtual (0,10 s) representou um aumento de 118,75%.

Com relação a análise da amostra, o aumento médio do tempo de treinamento entre a máquina virtual (7,28 ms) e a *Raspberry Pi Zero* (299,17 ms) foi de 4.010,26%. Já a comparação entre a máquina virtual e o *desktop* (12,07 ms) representou um aumento de 65,82%. O tempo para cada configuração para o *DenStream* pode ser visto na Figura 28.

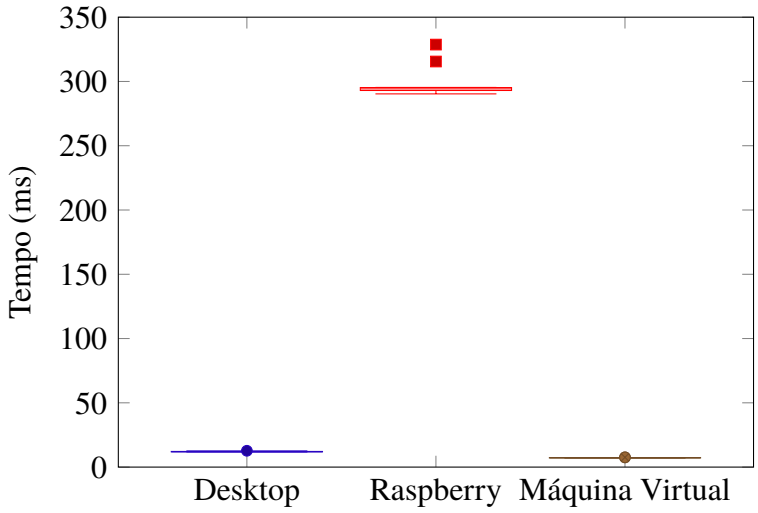


Figura 28 – *Box plot* do tempo de avaliação de cada amostra para o *DenStream* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

O *DStream* também apresentou um aumento percentual grande na parte de treinamento. Entretanto, por ter um tempo muito pequeno, o menor dentre todos, o aumento é praticamente imperceptível quando comparado aos demais algoritmos. O gráfico comparativo para as configurações utilizadas com o algoritmo *DStream* pode ser visto na Figura 29.

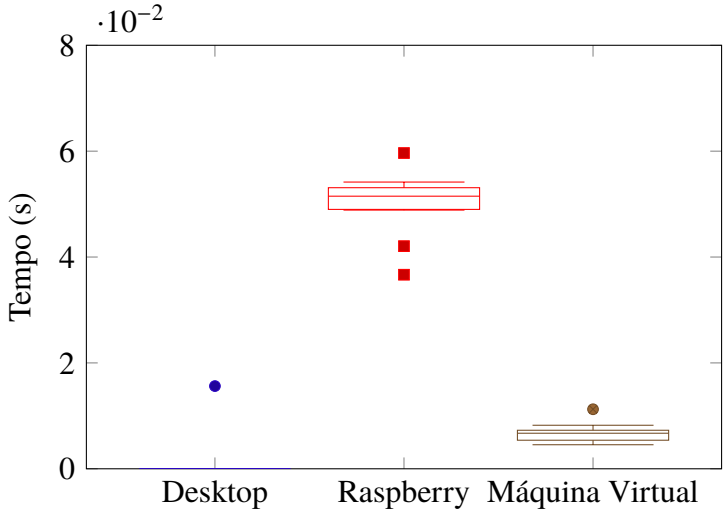


Figura 29 – *Box plot* do tempo de treinamento para o *DStream* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

O aumento do tempo médio de treinamento entre o *desktop* (1,6 ms) e a *Raspberry Pi Zero* (50 ms) foi de 3.099,21%. Já a comparação entre o tempo médio do *desktop* e a máquina virtual (6,8 ms) representou um aumento de 333,84%.

Com relação a análise da amostra, o aumento médio do tempo de treinamento entre a máquina virtual (7,06 ms) e a *Raspberry Pi Zero* (283,91 ms) foi de 3.919,46%. Já a comparação entre a máquina virtual e o *desktop* (10,41 ms) representou um aumento de 47,31%. O tempo para cada configuração para o *DStream* pode ser visto na Figura 30.

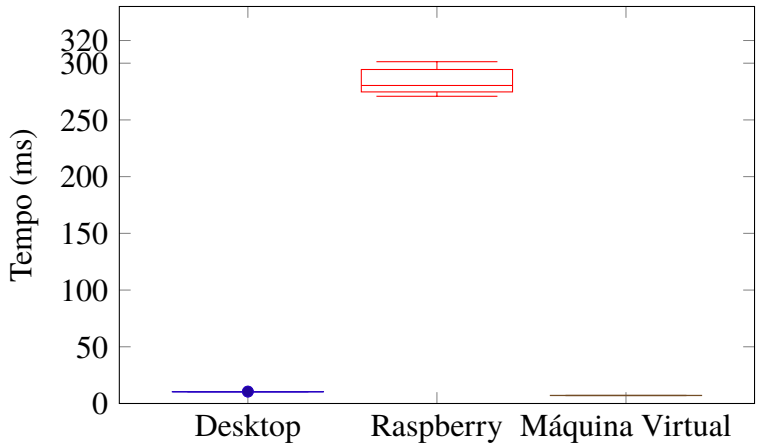


Figura 30 – *Box plot* do tempo de avaliação de cada amostra para o *DStream* considerando a variação para cada configuração.

Fonte: Elaborada pelo autor.

As tabelas com os dados que originaram os gráficos e comparações apresentados durante esta seção estão presentes no anexo B.

7.4 Análise comparativa

Avaliando os algoritmos de forma conjunta, todos os algoritmos conseguiram executar sem nenhuma interrupção por parte de nenhuma das configurações. Algumas características foram comuns a todos os cenários. A primeira característica foi que a quantidades de *threads* teve um impacto muito pequeno no desempenho dos algoritmos. Tal afirmativa se baseia no desempenho do *desktop* e da máquina virtual. Com isso, processadores mais simples de única *thread* parecem ser suficientes para executar todos os algoritmos.

No tocante a memória disponível, a configuração da máquina virtual, mesmo possuindo uma menor quantidade, conseguiu ter desempenho superior às outras configurações na análise de novas amostras. Sendo inclusive superior a *Raspberry Pi Zero* que possue o dobro de memória.

O desempenho do *CluStream* foi inferior aos algoritmos de densidade em relação à fase *offline*. Isso acontece devido à quantidade de iterações necessárias para diminuir o erro da soma quadrática. Por isso, este algoritmo não se faz adequado para cenários restritos. O comparativo

do desempenho do *CluStream* contra os algoritmos baseados em densidade pode ser visto na Figura 31.

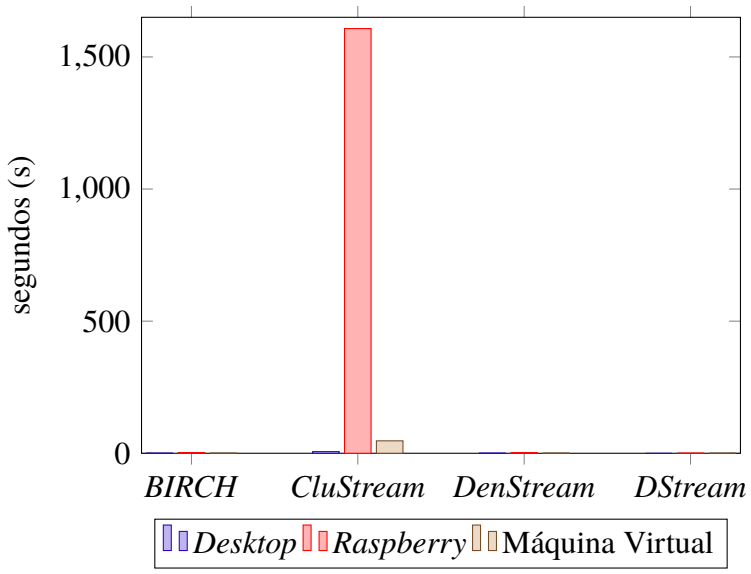


Figura 31 – Comparação da média obtida por cada algoritmo na fase de treinamento em cada configuração.

Fonte: Elaborada pelo autor.

Com relação a análise de novas amostras, todos os algoritmos obtiveram um desempenho similar, como mostrado na Figura 32.

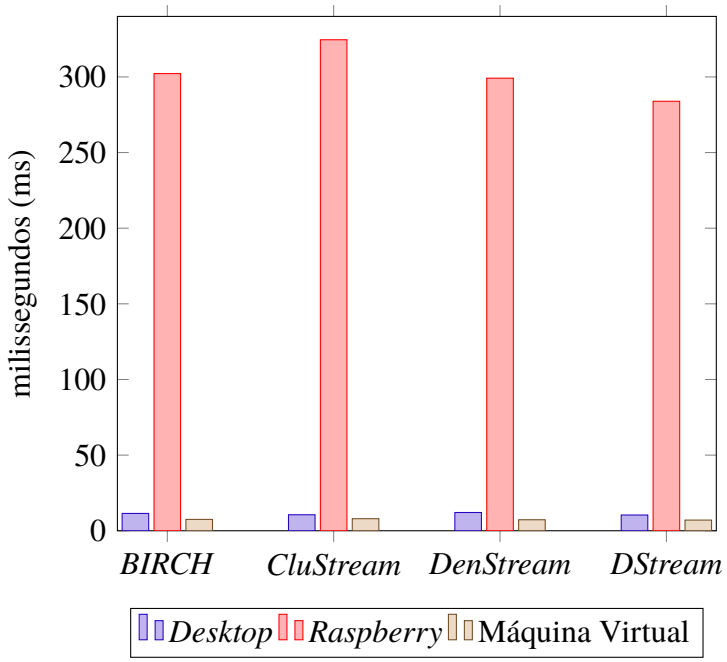


Figura 32 – Comparação da média obtida por cada algoritmo na análise de novas amostras em cada configuração.

Fonte: Elaborada pelo autor.

Já com relação ao tempo de treinamento dos algoritmos baseados em densidade, o *DStream* teve um desempenho muito superior ao restante, como visto na Figura 33, sendo assim o mais adequado para o cenário.

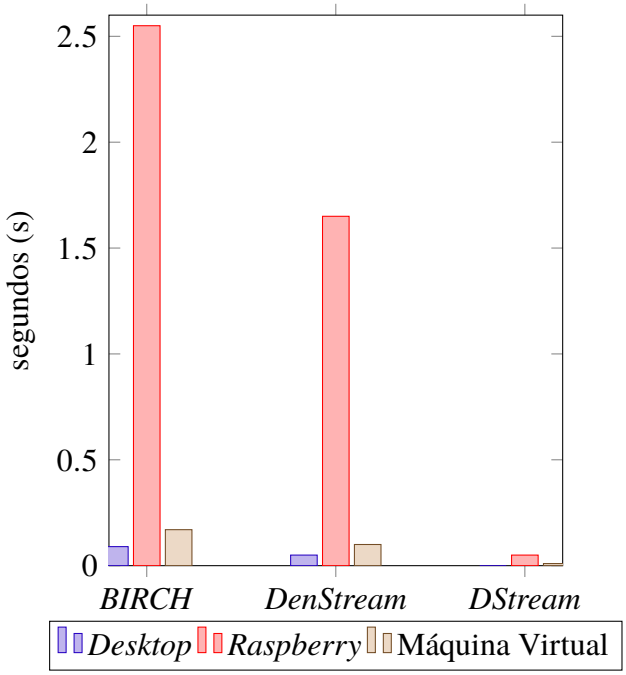


Figura 33 – Comparação da média obtida pelos algoritmos de densidade na fase de treinamento em cada configuração.

Fonte: Elaborada pelo autor.

8
Conclusão

Este trabalho teve como objetivo verificar a viabilidade do uso de algoritmos de aprendizagem de máquina não-supervisionado voltado para ambientes restritos para identificar início de formações de redes *botnets*. Para atender ao objetivo de forma satisfatória o algoritmo deve possuir uma demanda computacional baixa e trabalhar com *data streams*. Esta última condição se deve ao grande volume de dados, sendo considerado praticamente infinito.

Com isto foram definidos como objetivos específicos:

- Levantar algoritmos leves de aprendizagem de máquina não-supervisionados que podem ser utilizados para identificar formação de redes *botnets*;
- Verificar, por experimentação em ambientes controlados, a avaliação do algoritmo em cenários existentes de ataque utilizando medidas como acurácia, tempo de resposta e adaptação a novos cenários;
- Apurar o perfil mínimo necessário para que o dispositivo dedicado consiga executar o algoritmo.

Após um mapeamento sistemático, foram levantados os algoritmos *BIRCH*, *CluStream*, *DenStream* e *DStream*. Destes, apenas o *CluStream* não possui uma abordagem de agrupamento por densidade, e sim por distância radial. Para melhorar a qualidade dos resultados dos algoritmos, foram utilizados dois algoritmos para pré-processamento: o *LOF* para tentar remover dados anômalos do conjunto de dados benignos utilizados para o treinamento e; o *PCA* para reduzir a dimensionalidade dos dados, consumindo menos recursos e favorecendo características mais importantes.

Dos algoritmos selecionados para experimentação, o *CluStream*, justamente por ser baseado em distância radial, apresentou o pior resultado. Teve uma acurácia abaixo de 50%, o

que dificulta a recomendação do seu uso, mesmo após a utilização do método de diminuição da soma quadrática.

Já os algoritmos baseados em densidade tiveram resultados promissores. O *BIRCH* teve o pior desempenho dentre estes, obtendo uma acurácia média de 96,71%, com a taxa mais baixa entre todos de verdadeiros negativos. Foi ainda o que obteve o maior tempo de treinamento. Portanto, seria o indicado para um cenário onde a única característica importante é o bloqueio dos dados maliciosos. A sua taxa para bloqueio de dados benignos foi a mais alta, o que pode impactar na qualidade de serviço da rede.

O *DenStream* obteve, em algumas rodadas de execução, os melhores resultados dentre todas as experimentações feitas. Apesar de ter uma média similar ao *DStream*, ele pode obter resultados superiores com um ajuste preciso dos seus parâmetros, o que faria necessário algum nível de intervenção humana. Entretanto, para esses parâmetros serem automatizados pelo algoritmo, o *DenStream* teve uma variância maior na sua acurácia geral. Associado a isso, este teve o segundo melhor desempenho para treinamento, perdendo apenas para o *DStream*, mas obteve o maior tempo para análise das amostras novas.

Por fim, o *DStream* foi o que obteve o melhor resultado, visto que apresentou uma acurácia alta, com média similar ao do *DenStream*, porém com uma menor variância entre as rodadas. Assim, apresentou-se como um algoritmo mais confiável a nível de acurácia para um cenário sem intervenção humana. Além disso este teve o menor tempo de treinamento e perdeu apenas para o *CluStream* em tempo de análise de novas amostras. Portanto dos 4 algoritmos, para um processo automatizado, o *DStream* tem uma acurácia menos variável e um treinamento muito rápido.

Já quanto ao consumo de recursos, foi verificado que todos conseguiram executar em dispositivos com uma única *thread* e com 256 MB de RAM, tendo assim um baixo impacto computacional em dispositivos *System On Chip* (SOC).

8.1 Dificuldades e Limitações

Algumas dificuldades e limitações foram encontradas ao decorrer do trabalho. A primeira foi a inutilização da Raspberry Pi B 3+ que seria utilizada como quarta configuração para poder executar uma melhor comparação entre as configurações pensadas devido a um curto-circuito.

Além disso, não se conseguiu de forma efetiva, obter a medição da memória utilizada apenas pelo algoritmo. Este dado permitiria uma melhor análise da quantidade mínima de memória livre em um sistema para poder executar os algoritmos, resultando em aproximações.

Outra limitação encontrada foi a falta da análise do consumo de energia por limitação de tempo da pesquisa. Esta é uma característica importante para cenários restritos, principalmente *IoT*. Tendo em vista que muito desses dispositivos não possuem uma fonte infinita de energia, executando na sua maioria a base de baterias ou outras fontes alternativas, como energia solar.

8.2 Trabalhos Futuros

Nesta seção são levantados algumas possibilidades de trabalhos futuros para dar sequência e melhorar os resultados obtidos:

- Para tentar mitigar as dificuldades e limitações levantadas, a avaliação da memória de forma mais precisa permitiria um levantamento mais preciso do dispositivo mínimo;
- Uso de mais configurações para melhor avaliar o impacto de outros fatores, como frequências, sistemas operacionais e otimizações fornecidas por fabricantes para suas plataformas;
- Migrar para um ambiente de *SDN*, permitindo um *test bed* mais fácil de controle e avaliar de fato se estes algoritmos também podem ser utilizados para a segurança das redes definidas por *software*;
- Implantar estes algoritmos em um cenário real, similar ao feito no trabalho que originou o *dataset* (MEIDAN et al., 2018), retirando o uso do conjunto de dados e avaliando seu resultado em um cenário real.

8.3 Publicações Relacionadas

Nesta seção são apresentadas as contribuições relacionadas a dissertação.

8.3.1 Trabalhos Aceitos

a *Detecting IoT Botnet Formation using Data Stream Clustering Algorithms - In Proceedings of the 16th International Conference on Web Information Systems and Technologies - Volume 1: DMMLACS*, Novembro 3-5, 2020, ISBN 978-989-758-478-7, p. 395-402. DOI: 10.5220/0010180903950402[1]- Qualis A4

8.3.2 Trabalhos Submetidos

a Detecção de formação de Botnet para cenários de IoT usando algoritmos de agrupamento para data stream. Submetido para revista Informação & Tecnologia.

b *Evaluation of Unsupervised Machine Learning Algorithms Data Stream to Prevent Botnet Formation*. Possível publicação para revista *Journal of Information Security and Applications* - Qualis A2.

[1] Disponível em: <https://www.scitepress.org/PublicationsDetail.aspx?ID=byivQLSkQnY=&t=1>

Referências

AFGHAH, F. et al. A reram physically unclonable function (reram puf)- based approach to enhance authentication security in software defined wireless networks. *International Journal of Wireless Information Networks*, Springer US, 2018. ISSN 1572-8129. Disponível em: <https://doi.org/10.1007/s10776-018-0391-6>. Citado 2 vezes nas páginas 26 e 29.

AGARWAL, B. et al. Enrichment of machine learning based activity classification in smart homes using ensemble learning. *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing*, 2016. Citado 3 vezes nas páginas 25, 29 e 31.

AGGARWAL, C. C. et al. A framework for clustering evolving data streams. In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. [S.l.]: VLDB Endowment, 2003. (Vldb '03), p. 81–92. ISBN 0127224424. Citado na página 34.

AKBAR, A. et al. Context-aware stream processing for distributed iot applications. *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015. Citado 3 vezes nas páginas 27, 30 e 31.

AKBAR, A. et al. Predicting complex events for pro-active iot applications. *IEEE World Forum on Internet of Things*, p. 0–5, 2015. Citado 3 vezes nas páginas 27, 30 e 31.

AKBAR, A. et al. Predictive analytics for complex iot data streams. *Ieee Internet Of Things Journal*, X, n. X, p. 1–12, 2017. Citado 3 vezes nas páginas 27, 30 e 31.

AKYILDIZ, I. F. et al. Wireless sensor networks: A survey. *Comput. Netw.*, Elsevier North-Holland, Inc., Usa, v. 38, n. 4, p. 393–422, mar. 2002. ISSN 1389-1286. Disponível em: <https://doi.org/10.1016/S1389-1286(01)00302-4>. Citado na página 19.

AL-JANABI, S.; SHEHAB, A. Edge computing: Review and future directions. 09 2019. Citado 2 vezes nas páginas 21 e 22.

AL-QAMASH, A. et al. Cloud, fog, and edge computing: A software engineering perspective. In: . [S.l.: s.n.], 2018. p. 276–284. Citado na página 20.

ALOISE, D. et al. Np-hardness of euclidean sum-of-squares clustering. *Machine Learning*, v. 75, n. 2, p. 245–248, 2009. Citado na página 49.

ALSHAMMARI, R.; ZINCIR-HEYWOOD, A. N. *Machine learning based encrypted traffic classification: Identifying SSH and Skype*. 2009. Disponível em: <https://ieeexplore.ieee.org/document/5356534/>. Citado 3 vezes nas páginas 26, 28 e 32.

AMZA, C.; CRISTEA, V. Hybrid network intrusion detection. *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, p. 503–510, 2011. Citado 2 vezes nas páginas 25 e 29.

ANGRISHI, K. *Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV) : IoT Botnets*. 2017. Citado na página 23.

ARTHUR, D.; VASSILVITSKII, S. K-means++: The advantages of careful seeding. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. Usa: Society for Industrial and Applied Mathematics, 2007. (Soda '07), p. 1027–1035. ISBN 9780898716245. Citado 2 vezes nas páginas 35 e 54.

ASHTON, K. *That 'Internet of Things' Thing*. RFID Jounral, 2009. Disponível em: <https://www.rfidjournal.com/articles/view?4986>. Citado na página 19.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, 2010. ISSN 1389-1286. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1389128610001568>. Citado na página 14.

AXENIE, C.; BORTOLI, S. Starlord : Sliding window temporal accumulate-retract learning for online reasoning on datastreams. *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Ieee, p. 1115–1122, 2018. Citado 2 vezes nas páginas 27 e 29.

Bezdek, J. C.; Hathaway, R. J. Vat: a tool for visual assessment of (cluster) tendency. In: *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*. [S.l.: s.n.], 2002. v. 3, p. 2225–2230 vol.3. Citado na página 27.

BHATTACHARYYA, S.; KATRAMATOS, D.; YOO, S. Why wait ? let us start computing while the data is still on the wire. *Future Generation Computer Systems*, Elsevier B.V., v. 89, p. 563–573, 2018. ISSN 0167-739x. Disponível em: <https://doi.org/10.1016/j.future.2018.07.024>. Citado 2 vezes nas páginas 26 e 29.

BREUNIG, M. M. et al. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, Association for Computing Machinery, New York, NY, USA, v. 29, n. 2, p. 93–104, maio 2000. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/335191.335388>. Citado 2 vezes nas páginas 42 e 43.

BUYYA, R.; DASTJERDI, A. V. *Internet of things: principles and paradigms*. [S.l.]: Morgan Kaufmann, 2016. Citado na página 21.

CAO, F. et al. Density-based clustering over an evolving data stream with noise. *Proceedings of the 2006 SIAM International Conference on Data Mining*, 2006. Citado 3 vezes nas páginas 28, 37 e 48.

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*. New York, NY, USA: Association for Computing Machinery, 2007. (Kdd '07), p. 133–142. ISBN 9781595936097. Disponível em: <https://doi.org/10.1145/1281192.1281210>. Citado na página 36.

Dey, A. et al. Namatad: Inferring occupancy from building sensors using machine learning. In: *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2016. p. 478–483. Citado 3 vezes nas páginas 25, 29 e 31.

DIETZ, C. et al. Iot-botnet detection and isolation by access routers. *2018 9th International Conference on the Network of the Future (NOF)*, 2018. Citado 2 vezes nas páginas 16 e 22.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, p. 71–80, 2000. Disponível em: <http://portal.acm.org/citation.cfm?doid=347090.347107>. Citado na página 28.

DONOVAN, P. O. et al. A fog computing industrial cyber-physical system for embedded low-latency machine learning industry 4 . 0 applications. *Manufacturing Letters*, Society of Manufacturing Engineers (SME), p. 4–7, 2018. ISSN 2213-8463. Disponível em: <https://doi.org/10.1016/j.mfglet.2018.01.005>. Citado 2 vezes nas páginas 25 e 29.

ELKHOUKHI, H. Sciencedirect sciencedirect towards a real-time occupancy detection approach for smart towards a real-time occupancy detection approach for smart buildings buildings. *Procedia Computer Science*, Elsevier B.V., v. 134, p. 114–120, 2018. ISSN 1877-0509. Disponível em: <https://doi.org/10.1016/j.procs.2018.07.151>. Citado 3 vezes nas páginas 28, 30 e 31.

ENDLER, M.; SILVA, F.; HAEUSLER, E. H. Towards stream-based reasoning and machine learning for iot applications. *Intelligent Systems Conference 2017*, n. September, 2017. Citado 3 vezes nas páginas 24, 29 e 31.

EVANS, M.; ROSENTHAL, J. *Probability and Statistics: The Science of Uncertainty*. W. H. Freeman, 2009. ISBN 9781429281270. Disponível em: <https://books.google.com.br/books?id=plokAAAAQBAJ>. Citado na página 47.

FARIA, E. R.; GAMA, J. a.; CARVALHO, A. C. P. L. F. Novelty detection algorithm for data streams multi-class problems. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. New York, NY, USA: Acm, 2013. (Sac '13), p. 795–800. ISBN 978-1-4503-1656-9. Disponível em: <http://doi.acm.org/10.1145/2480362.2480515>. Citado na página 33.

FRIEDRICH, L. F. A survey on operating system support for embedded systems properties. In: *Anais do Workshop de Sistemas Operacionais, VI (julio 2009)*. [S.l.: s.n.], 2009. p. 2393–2404. Citado na página 18.

Garey, M.; Johnson, D.; Witsenhausen, H. The complexity of the generalized lloyd - max problem (corresp.). *IEEE Transactions on Information Theory*, v. 28, n. 2, p. 255–256, 1982. Citado na página 49.

GUBBI, J. et al. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, v. 29, n. 7, p. 1645–1660, 2013. Citado na página 15.

HAWKINS, D. M. Identification of outliers. 1980. Citado na página 43.

JAIN, R. *The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: Wiley, 1991. I-XXVII, 1-685 p. (Wiley professional computing). ISBN 978-0-471-50336-1. Citado na página 52.

KAMBOURAKIS, G.; KOLIAS, C.; STAVROU, A. The mirai botnet and the iot zombie armies. *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, 2017. Disponível em: <https://cs.gmu.edu/~astavrou/research/TheMiraiBotnetandtheIoTZombieArmies.pdf>. Citado 2 vezes nas páginas 14 e 23.

KANOUN, K.; TEKIN, C.; ATIENZA, D. Big-data streaming applications scheduling based on staged multi-armed bandits. *Ieee Transactions On Computers*, Xx, n. Xx, p. 1–14, 2016. Citado 2 vezes nas páginas 24 e 29.

KAPOOR, A.; DHAVALE, S. Control flow graph based multiclass malware detection using bi-normal separation. *Defence Science Journal*, v. 66, n. 2, p. 138–145, 2016. Citado 2 vezes nas páginas 25 e 29.

KOLIAS, C. et al. Ddos in the iot: Mirai and other botnets. *Computer*, v. 50, p. 80–84, 01 2017. Citado na página 23.

KOURTELLIS, N.; BIFET, A. Vht : Vertical hoeffding tree. *2016 IEEE International Conference on Big Data*, p. 915–922, 2016. Citado 3 vezes nas páginas 28, 30 e 31.

KUMAR, D. et al. Adaptive cluster tendency visualization and anomaly detection for streaming data. *ACM Transactions on Knowledge Discovery from Data*, v. 11, n. 2, p. 1–40, 2016. Citado 3 vezes nas páginas 27, 29 e 31.

LI, F. A pattern query strategy based on semi-supervised machine learning in distributed wsns. *Journal of Information and Computational Science*, v. 11, n. 18, p. 6447–6459, 2014. ISSN 15487741. Citado 3 vezes nas páginas 26, 29 e 31.

LU, J. et al. A parallel approach on clustering traffic data stream based on the density. *2018 Sixth International Conference on Advanced Cloud and Big Data (CBD)*, 2018. Citado 4 vezes nas páginas 29, 30, 31 e 33.

MAHAJAN, M.; NIMBHORKAR, P.; VARADARAJAN, K. The planar k-means problem is np-hard. *WALCOM: Algorithms and Computation Lecture Notes in Computer Science*, p. 274–285, 2009. Citado na página 49.

MAHMUD, R.; KOTAGIRI, R.; BUYYA, R. Fog computing: A taxonomy, survey and future directions. *Internet of Things Internet of Everything*, p. 103–130, 2017. Citado na página 20.

MARZANO, A. et al. The evolution of bashlite and mirai iot botnets. *2018 IEEE Symposium on Computers and Communications (ISCC)*, 2018. Disponível em: <https://homepages.dcc.ufmg.br/~cunha/papers/marzano18iscc-botnets.pdf>. Citado 3 vezes nas páginas 14, 22 e 23.

MEIDAN, Y. et al. N-baiot: Network-based detection of iot botnet attacks using deep autoencoders. *CoRR*, abs/1805.03409, 2018. Disponível em: <http://arxiv.org/abs/1805.03409>. Citado 7 vezes nas páginas 7, 39, 40, 41, 46, 52 e 66.

MIRSKY, Y. et al. *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*. 2018. Citado na página 40.

MOORE, A. W.; ZUEV, D. Internet traffic classification using bayesian analysis techniques. *ACM SIGMETRICS Performance Evaluation Review*, v. 33, n. 1, p. 50, 2005. Citado 3 vezes nas páginas 26, 28 e 32.

NIXON, C.; SEDKY, M.; HASSAN, M. Practical application of machine learning based online intrusion detection to internet of things networks. *2019 IEEE Global Conference on Internet of Things (GCIoT)*, 2019. Citado na página 48.

OZKOK, F. O. A new approach to determine eps parameter of dbscan algorithm. *International Journal of Intelligent Systems and Applications in Engineering*, v. 4, n. 5, p. 247–251, 2017. Citado 2 vezes nas páginas 37 e 50.

PEARSON, K. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, v. 2, n. 11, p. 559–572, 1901. Citado na página 42.

POKRAJAC, D.; LAZAREVIC, A.; LATECKI, L. J. Incremental local outlier detection for data streams. *2007 IEEE Symposium on Computational Intelligence and Data Mining*, 2007. Citado na página 28.

ROOPAEI, M.; RAD, P.; JAMSHIDI, M. Deep learning control for complex and large scale cloud systems deep learning control for complex and large scale cloud systems. *Intelligent Automation & Soft Computing*, Taylor & Francis, v. 8587, n. June, p. 1–3, 2017. ISSN 1079-8587. Disponível em: <http://doi.org/10.1080/10798587.2017.1329245>. Citado 2 vezes nas páginas 25 e 29.

SALEHI, M. et al. Local outlier detection for data streams in sensor networks : Revisiting the utility problem invited paper. *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, n. April, p. 7–9, 2015. Citado 4 vezes nas páginas 27, 28, 30 e 31.

SCHMIDT, B.; KOUNTANIS, D.; AL-FUQAHA, A. Artificial immune system inspired algorithm for flow-based internet traffic classification. *IEEE 6th International Conference on Cloud Computing Technology and Science Artificial*, 2014. Citado 3 vezes nas páginas 26, 30 e 31.

SHAFIK, W.; MOSTAFAVI, S. Fog computing architectures, privacy and security solutions. v. 24, p. 1–14, 07 2019. Citado na página 21.

SHI, W. et al. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, v. 3, n. 5, p. 637–646, 2016. ISSN 23274662. Citado 2 vezes nas páginas 14 e 21.

SILVA, J. A. et al. Data stream clustering: A survey. *ACM Comput. Surv.*, Acm, New York, NY, USA, v. 46, n. 1, p. 13:1–13:31, jul. 2013. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2522968.2522981>. Citado 4 vezes nas páginas 32, 33, 34 e 37.

SINGH, K.; AGRAWAL, S. *Comparative analysis of five machine learning algorithms for IP traffic classification*. 2011. Disponível em: <https://ieeexplore.ieee.org/abstract/document/5958481/>. Citado 3 vezes nas páginas 26, 28 e 32.

SINGH, Y. et al. Distributed event detection in wireless sensor networks for forest fires. *2013 UKSim 15th International Conference on Computer Modelling and Simulation*, 2013. Citado 4 vezes nas páginas 28, 30, 31 e 33.

SPEZZANO, G.; VINCI, A. Pattern detection in cyber-physical systems. *Procedia - Procedia Computer Science*, Elsevier Masson SAS, v. 52, p. 1016–1021, 2015. ISSN 1877-0509. Disponível em: <http://dx.doi.org/10.1016/j.procs.2015.05.096>. Citado 3 vezes nas páginas 28, 30 e 31.

SUNDMAEKER, H. et al. Vision and challenges for realizing the internet of things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 04 2010. Citado na página 19.

VAHID, F.; GIVARGIS, T. *Embedded system design: a unified hardware / software introduction*. [S.l.]: John Wiley & Sons, 2010. Citado na página 18.

VASSEUR, J.-P.; DUNKELS, A. *Interconnecting smart objects with IP: the next Internet*. [S.l.]: Morgan Kaufmann Publishers/Elsevier, 2012. Citado 3 vezes nas páginas 15, 18 e 19.

WHITE, E. *Making embedded systems: Design Patterns for Great Software*. [S.l.]: OReilly, 2012. Citado na página 18.

ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: An efficient data clustering method for very large databases. In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1996. (Sigmod '96), p. 103–114. ISBN 0897917944. Disponível em: <https://doi.org/10.1145/233269.233324>. Citado 2 vezes nas páginas 33 e 34.

ZHAO, G. *Wireless sensor networks: an information processing approach*. [S.l.]: Ubsd., 2004. Citado na página 19.

ZHAO, Q. Learning with data streams – an nntree based approach. In: ENOKIDO, T. et al. (Ed.). *Embedded and Ubiquitous Computing – EUC 2005 Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 519–528. ISBN 978-3-540-32296-2. Citado 2 vezes nas páginas 24 e 29.

Anexos

ANEXO A – Tabela contendo resultados qualitativos.

DenStream					
TP	FP	TN	FN	Acertos	Erros
7741	62	7698	200	15439	262
7705	50	7710	236	15415	286
7647	39	7721	294	15368	333
7526	7	7753	415	15279	422
7880	115	7645	61	15525	176
7655	26	7734	286	15389	312
7714	63	7697	227	15411	290
7735	73	7687	206	15422	279
7887	106	7654	54	15541	160
7773	66	7694	168	15467	234
7695	48	7712	246	15407	294
7685	34	7726	256	15411	290
7651	30	7730	290	15381	320
7681	44	7716	260	15397	304
7631	22	7738	310	15369	332
7709	67	7693	232	15402	299
7620	26	7734	321	15354	347
7745	61	7699	196	15444	257
7644	24	7736	297	15380	321
7891	129	7631	50	15522	179
7714	46	7714	227	15428	273
7794	93	7667	147	15461	240
7754	63	7697	187	15451	250
7806	92	7668	135	15474	227
7622	22	7738	319	15360	341
7879	128	7632	62	15511	190
7739	78	7682	202	15421	280
7857	97	7663	84	15520	181
7817	83	7677	124	15494	207
7745	60	7700	196	15445	256
7890	104	7656	51	15546	155
7524	3	7757	417	15281	420
7872	120	7640	69	15512	189
7523	1	7759	418	15282	419
7864	118	7642	77	15506	195
7877	127	7633	64	15510	191
7887	99	7661	54	15548	153
7647	40	7720	294	15367	334
7790	69	7691	151	15481	220
7836	97	7663	105	15499	202

Tabela 8 – Resultado qualitativo do *DenStream*.

BIRCH					
TP	FP	TN	FN	Acertos	Erros
7396	11	7749	545	15145	556
7450	22	7738	491	15188	513
7506	26	7734	435	15240	461
7265	5	7755	676	15020	681
7441	22	7738	500	15179	522
7466	7	7753	475	15219	482
7481	11	7749	460	15230	471
7482	15	7745	459	15227	474
7425	11	7749	516	15174	527
7436	9	7751	505	15187	514
7352	6	7754	589	15106	595
7497	19	7741	444	15238	463
7368	4	7756	573	15124	577
7341	6	7754	600	15095	606
7435	10	7750	506	15185	516
7507	16	7744	434	15251	450
7347	3	7757	594	15104	597
7461	17	7743	480	15204	497
7423	9	7751	518	15174	527
7440	25	7735	501	15175	526
7456	5	7755	485	15211	490
7530	24	7736	411	15266	435
7460	14	7746	481	15206	495
7444	24	7736	497	15180	521
7505	17	7743	436	15248	453
7422	14	7746	519	15168	533
7481	18	7742	460	15223	478
7497	16	7744	444	15241	460
7456	13	7747	485	15203	498
7535	29	7731	406	15266	435
7375	12	7748	566	15123	578
7282	2	7758	659	15040	661
7387	10	7750	554	15137	564
7511	17	7743	430	15254	447
7564	36	7724	377	15288	413
7390	7	7753	551	15143	558
7476	5	7755	465	15231	470
7448	23	7737	493	15185	516
7423	6	7754	518	15177	524
7373	9	7751	568	15124	577

Tabela 9 – Resultado qualitativo do *BIRCH*.

DStream					
TP	FP	TN	FN	Acertos	Erros
7768	81	7679	173	15447	254
7798	84	7676	143	15474	227
7807	102	7658	134	15465	236
7656	49	7711	285	15367	334
7731	79	7681	210	15412	289
7812	98	7662	129	15474	227
7798	104	7656	143	15454	247
7820	120	7640	121	15460	241
7757	75	7685	184	15442	259
7768	85	7675	173	15443	258
7725	85	7675	216	15400	301
7789	82	7678	152	15467	234
7734	86	7674	207	15408	293
7709	81	7679	232	15388	313
7809	91	7669	132	15478	223
7760	117	7643	181	15403	298
7672	61	7699	269	15371	330
7796	86	7674	145	15470	231
7747	81	7679	194	15426	275
7736	92	7668	205	15404	297
7767	69	7691	174	15458	243
7785	99	7661	156	15446	255
7769	86	7674	172	15443	258
7758	91	7669	183	15427	274
7803	93	7667	138	15470	231
7740	93	7667	201	15407	294
7731	91	7669	210	15400	301
7779	83	7677	162	15456	245
7806	90	7670	135	15476	225
7754	77	7683	187	15437	264
7738	76	7684	203	15422	279
7653	52	7708	288	15361	340
7740	90	7670	201	15410	291
7813	87	7673	128	15486	215
7775	101	7659	166	15434	267
7753	101	7659	188	15412	289
7740	66	7694	201	15434	267
7805	124	7636	136	15441	260
7746	71	7689	195	15435	266
7682	59	7701	259	15383	318

Tabela 10 – Resultado qualitativo do *DStream*.

CluStream					
TP	FP	TN	FN	Acertos	Erros
2	1274	6486	7939	6488	9213
8	4884	2876	7933	2884	12817
4	0	7760	7937	7764	7937
23	3	7757	7918	7780	7921
0	3375	4385	7941	4385	11316
20	2823	4937	7921	4957	10744
0	1288	6472	7941	6472	9229
8	4830	2930	7933	2938	12763
1	1256	6504	7940	6505	9196
0	1285	6475	7941	6475	9226
3	168	7592	7938	7595	8106
5	4834	2926	7936	2931	12770
2	1343	6417	7939	6419	9282
0	1295	6465	7941	6465	9236
24	2790	4970	7917	4994	10707
1	1233	6527	7940	6528	9173
7524	57	7703	417	15227	474
9	0	7760	7932	7769	7932
16	2803	4957	7925	4973	10728
0	3377	4383	7941	4383	11318
2	0	7760	7939	7762	7939
4	0	7760	7937	7764	7937
32	2792	4968	7909	5000	10701
0	1322	6438	7941	6438	9263
10	4857	2903	7931	2913	12788
1	1311	6449	7940	6450	9251
0	3381	4379	7941	4379	11322
0	1271	6489	7941	6489	9212
7924	2934	4826	17	12750	2951
1	0	7760	7940	7761	7940
0	3384	4376	7941	4376	11325
29	7	7753	7912	7782	7919
4	0	7760	7937	7764	7937
8	4878	2882	7933	2890	12811
5	4844	2916	7936	2921	12780
24	2525	5235	7917	5259	10442
0	3404	4356	7941	4356	11345
6	0	7760	7935	7766	7935
0	3383	4377	7941	4377	11324
7652	88	7672	289	15324	377

Tabela 11 – Resultado qualitativo do *CluStream*.

ANEXO B – Tabela com resultados do teste de performance

Birch		CluStream		DenStream		DStream	
Offline (s)	Amostra (ms)	Offline (s)	Amostra (ms)	Offline (s)	Amostra (ms)	Offline (s)	Amostra (ms)
0,15626	11,37568	6,12536	9,52257	0,04685	12,25084	0,00000	10,40718
0,07814	11,47119	6,04798	9,53747	0,04687	11,88346	0,00000	10,37033
0,07812	11,67752	6,21910	10,16347	0,04687	11,95893	0,00000	10,36985
0,09375	11,37968	6,09410	10,10071	0,04687	11,78655	0,00000	10,50987
0,09374	11,45930	6,25039	10,25605	0,04685	11,93270	0,00000	10,39633
0,07810	11,58639	6,18805	10,54163	0,04689	11,90741	0,00000	10,41058
0,07812	11,32002	6,23477	10,80036	0,04689	11,82403	0,01562	10,48349
0,07814	11,34286	6,12537	11,01735	0,04687	12,59037	0,00000	10,36433
0,07812	11,45421	6,32867	12,41676	0,04685	11,86806	0,00000	10,36606
0,07814	11,47559	6,17236	11,45829	0,04687	12,69530	0,00000	10,37483

Tabela 12 – Resultado de performance para *Desktop*.

Birch		CluStream		DenStream		DStream	
Offline (s)	Amostra (ms)	Offline (s)	Amostra (ms)	Offline (s)	Amostra (ms)	Offline (s)	Amostra (ms)
3,21004	303,93902	1617,17180	285,24274	1,69452	328,67862	0,05965	280,40421
2,69847	326,97663	1583,29602	283,49814	1,61445	295,03944	0,05289	288,37112
2,33124	299,70178	1544,98790	306,83459	1,69986	315,51717	0,05259	296,47196
2,31305	297,79271	1545,53787	315,40094	1,56637	292,12419	0,03666	270,84001
2,34313	304,52986	1663,95606	321,94918	1,65153	294,54930	0,05316	275,08756
2,31220	300,94745	1546,35499	307,41760	1,61338	295,35591	0,05416	274,56253
2,73135	303,96869	1662,09784	392,05241	1,65943	290,33371	0,04938	301,31828
2,59112	297,70041	1690,74001	327,04616	1,65938	292,81717	0,04887	298,13580
2,67587	299,34250	1547,43974	322,62919	1,62680	293,42746	0,04206	273,57348
2,27436	287,14675	1672,70098	383,69424	1,67412	293,89928	0,05040	280,32173

Tabela 13 – Resultado de performance para *Raspberry Pi Zero*.

Birch		CluStream		DenStream		DStream	
Offline (s)	Amostra (s)	Offline (s)	Amostra (s)	Offline (s)	Amostra (s)	Offline (s)	Amostra (s)
0,35324	7,66848	48,83122	7,34156	0,09659	7,57528	0,00689	7,01905
0,12059	7,44974	46,87010	7,18874	0,13523	7,32524	0,00822	6,94850
0,12137	7,43454	47,06759	7,27634	0,07295	7,08780	0,00532	7,05452
0,22543	7,37305	47,38228	7,45090	0,15678	7,20073	0,01125	7,13923
0,11649	7,67275	46,40113	7,68121	0,08570	7,07684	0,00741	6,96251
0,14065	7,69354	46,65034	7,72023	0,13935	7,34347	0,00510	7,14964
0,17189	7,30487	46,59138	9,03524	0,07938	7,19945	0,00454	7,16207
0,22490	7,43217	46,35199	8,13316	0,07589	7,18311	0,00658	7,03091
0,14812	7,51297	47,07137	8,16494	0,06909	7,58363	0,00685	7,14862
0,12595	7,56136	47,16102	9,74234	0,11429	7,21169	0,00561	7,01858

Tabela 14 – Resultado de performance para Máquina Virtual.