DISSERTAÇÃO DE MESTRADO PROFISSIONAL

FRAMEWORK PARA DETECÇÃO DE ATAQUES DOS EM DISPOSITIVOS IOT, UTILIZANDO ABORDAGENS DE APRENDIZADO DE MÁQUINAS

Felipe Barreto de Oliveira

Programa de Pós-Graduação Profissional em Engenharia Elétrica
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

FRAMEWORK PARA DETECÇÃO DE ATAQUES DOS EM DISPOSITIVOS IOT, UTILIZANDO ABORDAGENS DE APRENDIZADO DE MÁQUINAS

Felipe Barreto de Oliveira

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Georges Daniel Amvame Nze, Ph.D, FT/UnB _____
Orientador

Prof. Fábio Lúcio Lopes de Mendonça, Ph.D, FT/UnB _____
Examinador Interno

Prof. Laerte Peotta de Melo, Ph.D _____
Examinador Externo

Prof. Rafael Rabelo Nunes, Ph.D, FACE/UnB _____
Suplente

FICHA CATALOGRÁFICA

OLIVEIRA, FELIPE BARRETO
FRAMEWORK PARA DETECÇÃO DE ATAQUES DOS EM DISPOSITIVOS IOT, UTILIZANDO ABORDAGENS DE APRENDIZADO DE MÁQUINAS [Distrito Federal] 2023.
xvi, 67 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2023).
Dissertação de Mestrado Profissional - Publicação PPEE.MP.042 - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Sistema de Detecção de Intrusão para IoT	2. Aprendizado de Máquina
3. Detecção próximo ao Tempo Real	4. Ataques DoS
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

OLIVEIRA, F.B. (2023). *FRAMEWORK PARA DETECÇÃO DE ATAQUES DOS EM DISPOSITIVOS IOT, UTILIZANDO ABORDAGENS DE APRENDIZADO DE MÁQUINAS*. Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 67 p.

CESSÃO DE DIREITOS

AUTOR: Felipe Barreto de Oliveira
TÍTULO: FRAMEWORK PARA DETECÇÃO DE ATAQUES DOS EM DISPOSITIVOS IOT, UTILIZANDO ABORDAGENS DE APRENDIZADO DE MÁQUINAS.
GRAU: Mestre em Engenharia Elétrica ANO: 2023

Felipe Barreto de Oliveira
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter me concedido forças para conseguir chegar até aqui. A Ele toda honra, glória e poder para todo sempre. Como está escrito “Os que confiam no Senhor serão como o monte de Sião, que não se abala, mas permanece para sempre” (Salmos 125:1), posso afirmar que essa palavra se cumpriu na minha trajetória nesses anos de universidade. Agradeço também aos meus pais que sempre me apoiaram em todos os momentos, sendo estes de dificuldade ou de alegria, bem como a minha irmã que sempre me fez acreditar que isso seria possível. Devo citar aqui meus agradecimentos aos docentes dessa universidade, pois com o empenho e dedicação exercidos tornam esta uma referência nacional. Em especial agradeço ao professor, e companheiro em cristo, Georges Daniel que me apoiou e ajudou nessa jornada e ao meu amigo e companheiro Luiz Augusto pela força e ajuda. Por último, agradeço a PGFN pela ajuda, tanto financeira quanto profissional, sendo a mesma a maior patrocinadora desta pesquisa.

RESUMO

A Internet das Coisas (IoT) é um dos paradigmas mais importantes dos últimos anos, pois a sua principal característica é a possibilidade de fundir o mundo real com o mundo virtual, utilizando o conceito das "coisas". Por um lado, apresenta uma grande comodidade no nosso cotidiano, revolucionando a comunicação entre pessoas e objetos. Por outro lado, as vulnerabilidades apresentadas e os ataques que têm ocorrido indicam que esta tecnologia continua a ser uma expectativa para o futuro em diversa empresas, submergindo, assim, os benefícios que nos poderia proporcionar. Neste trabalho, propomos um framework composto de um sistema de detecção de intrusão em tempo real para dispositivos IoT, onde os ataques DoS serão detectados, identificados e classificados, seguindo a literatura atual. Para isso, é utilizado técnicas de aprendizado de máquinas para identificar ataques, através de anomalias ocorridas no monitoramento de dispositivos IoT na suíte ELK com o *plugin* Wazuh. O primeiro resultado experimental com o dataset NSL-KDD mostra a eficiência da nossa proposta, com 91,90% de acurácia, 0,9217 de precisão, 0,9190 de *recall* e 0,9168 de F1-score. O segundo resultado experimental com o ataque DoS de *syn flood* em tempo real, criado pelo metasploit, mostra uma acurácia de 99,89%, uma precisão de 1,0000, um *recall* de 0,9953 e um *F1-score* de 0,9977.

ABSTRACT

The Internet of Things is one of the most important paradigms of the last years, because its main characteristic is the possibility of merging the real world with the virtual world, using the concept of "things". On the one hand, it presents a great convenience in our daily lives, revolutionizing the communication between people and objects. On the other hand, the vulnerabilities presented and the attacks that have occurred indicate that this technology remains an expectation for the future, thus submerging the benefits it could provide us. In this paper, we propose a framework for real time intrusion detection system in IoT devices, where the DoS attacks will be detected, identified, and classified, following the present literature. For this purpose, machine learning is used to identify attacks through anomalies that occurred in monitoring IoT devices on the ELK suite with the Wazuh plugin. The first experimental result with the NSL-KDD dataset show our proposal's efficiency, with 91.90% accuracy, 0.9217 precision, 0.9190 recall, and 0.9168 F1-score. The second experimental result with real time syn flood attack, created by metasploit, show accuracy of 99,89%, precision of 1.0000, recall of 0.9953, F1-Score of 0.9977.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

IoT	*Internet of Things*
IDC	*International Data Corporation*
DoS	*Denial of Service*
IDS	*Intrusion Detection System*
NSL−KDD	*Network security laboratory-knowledge discovery in databases*
IRF	*Improved Random Forest*
ELK	*Elasticsearck, Logstash and Kibana*
DDoS	*Distributed Denial of services*
CNN	*Convolutional Neural Network*
PCA	*Principal Component Analysis*
LSTM	*Long Short Term Memory*
KNN	*K-Nearest Neighbors*
RNN	*Recurrent Neural Network*
TLS	*Transport Layer Security*
DNN	*Deep Neural Networks*
SVD	*Singular Value Decomposition*
R2L	*Remote to local*
U2R	*User to Root*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
ICMP	*Internet Control Message Protocol*
AWS S3	*Amazon Simple Storage Service*
AMQP	*Advanced Message Queuing Protocol*
MQTT	*Message Queuing Telemetry Transport*
GNS3	*Graphical Network Simulator 3*
LTS	*Long-Term Support*
CSV	*Comma separated values*
MLops	*Machine Learning Operations*
GST	*Generalized Suffix Tree*
U2R	*User-to-Root*
R2L	*Remote-to-Local*
HIDS	*Hybrid Intrusion Detection System*
SVM	*Support Vector Machine*
PADS	*Position Aware Distribution Signature*
SDN	*Software Define Network*
XGBC	*eXtreme Gradient Boosting Classifier*

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Há décadas, a sociedade vem vivenciando grandes avanços tecnológicos, desde inovações na forma de comunicação pessoal, há redes distribuídas e não centralizadas de informações como a internet. Atualmente, a internet das coisas (*Internet of Things - IoT*) é uma das tecnologias mais emergentes e com o maior potencial em nossa sociedade, onde possui um uso multidisciplinar com o alcance e inovações em diversas áreas. Em todo mundo, conforme o estudo [1], os gastos relacionados com IoT passaram de 745 Bilhões de dólares em 2020, com expectativa de atingir 1,1 trilhões em 2023 (como pode ser observado na Figura 1.1). Os gastos dos consumidores com dispositivos IoT em casas inteligente no mundo, segue a mesma tendência, pois, conforme a Figura 1.2, os valores aumentarão de 50 bilhões de dólares em 2015 para mais de 150 bilhões em 2025. No Brasil, segundo as projeções da FGV (Fundação Getúlio Vargas) [22], o setor IoT cresce cerca de 40% ao ano, com previsões para atingir uma representatividade de cerca de 11 bilhões de dólares em 2025. Como estimativa também, a International Data Corporation (IDC Brasil) projeta que os gastos com IoT devem chegar a 1,6 bilhões de dólares ao longo de 2022 [23], sendo esse valor 17,6% maior do que a de 2021. Deste modo, segundo Ericsson [24], em 2050 haverá cerca de 24 bilhões de dispositivos interconectados através do IoT, desde aplicações na área médica (*healthcare*) até revoluções industriais (Industria 4.0). Entusiastas acreditam que esse sistema dominará a internet que conhecemos atualmente, pois tudo estará conectado em todos os lugares possíveis, possuindo como grande alavancador a tecnologia 5G.

Como toda e qualquer tecnologia emergente, o IoT apresenta diversos desafios, sendo os mais críticos aqueles que se referem à segurança, pois, haverá inúmeros dispositivos que, portarão informações pessoais de cada indivíduo, sendo imprescindível a proteção desses dados. Os desafios são diversos (como mostra a Figura 1.3), tais como:

- Integridade dos dados.
- Falta de capacidade em criptografia.
- Problemas com privacidade.
- Ausência de um *framework* padrão.
- Automação não orquestrada.
- Dificuldade em atualização.
- Falta de investimento em segurança.

Por esses fatores, é possivel observar que as soluções convencionais para a segurança de dados, presente em nosso cotidiano, não são eficientes para essa tecnologia [3]. De acordo com Khanam et al [25], com a

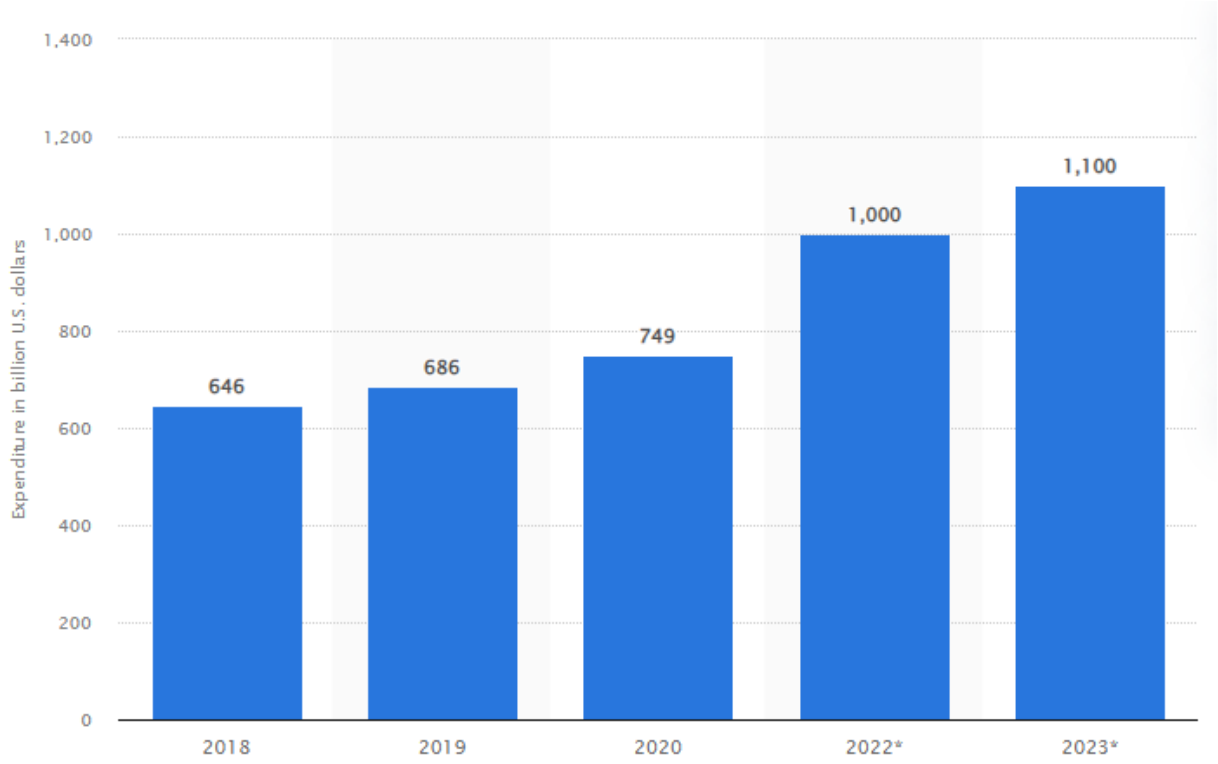

Figura 1.1: Predição de gastos relacionados ao IoT no mundo entre 2018 à 2023 [1]. O eixo das abscissas indica os anos correspondentes e o eixo das ordenadas indica os valores a serem gastos em bilhões de dólares.

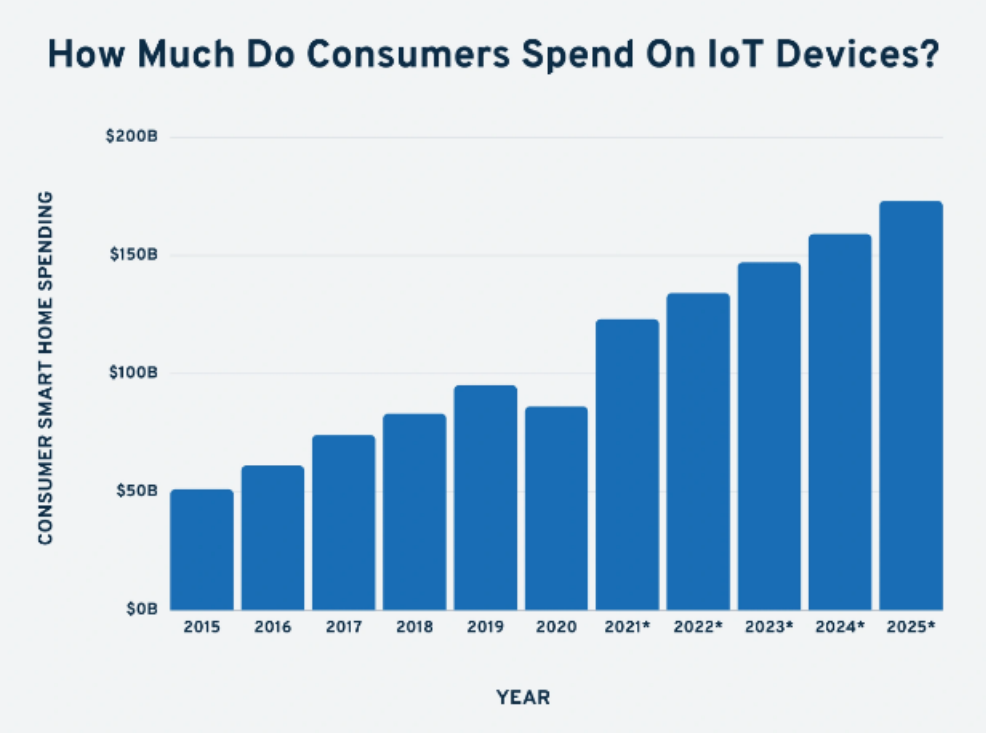

Figura 1.2: Predição dos gastos dos consumidores em dispositivos IoT no mundo entre 2015 à 2025 [2]. O eixo das abscissas indica os anos correspondentes e o eixo das ordenadas indica os valores a serem gastos em bilhões de dólares.

crescente demanda pelo IoT, surge conjuntamente a necessidade de obtenção de mais segurança, visto que a aplicação é composta por uma estrutura bastante vulnerável. Para Farooq et al [26], o IoT é uma grande revolução tecnológica que atualizou a infraestrutura atual da internet, onde todos os objetos presentes na arquitetura são exclusivamente capazes e onipresentes entre si, porém, além de ser uma grande expectativa para futuro, há um enorme potencial de desastre visando segurança.

Figura 1.3: Principais desafios de segurança presentes no IoT [3].

Dentre todos os desafios relacionados à segurança no IoT e seus ataques relacionados (conforme apresentado na Figura 1.4), a camada de aplicação apresenta-se como uma das principais vulnerabilidades desse sistema, pois, segundo Khanam et al [25], ela se destaca por possuir uma série de funcionalidades que fazem parte do cerne do sistema, tais como: monitorações, alertas, controles, otimizações e gerenciamentos. Por sua real importância, ela é alvo de muitos ataques, nos quais são exploradas suas vulnerabilidades através de vírus e malwares [27] [28], spyware [29], flooding [26], spoofing [30], code injection [31], message forging [32], entre outros presentes.

A detecção de intrusão por meio de monitoramento é uma técnica bem difundida em sistemas tradicionais, onde apresenta-se como peça fundamental em gerenciamento de redes de um sistema, entretanto essa técnica costuma ser utilizada como alerta de eventuais acidentes, sendo assim apenas uma etapa inicial para a segurança de dados. No âmbito do IoT, vale a pena ressaltar que as soluções presentes no mercado são generalistas, desprezando, assim, características fundamentais da rede IoT, essenciais para uma detecção imediata de um ataque.

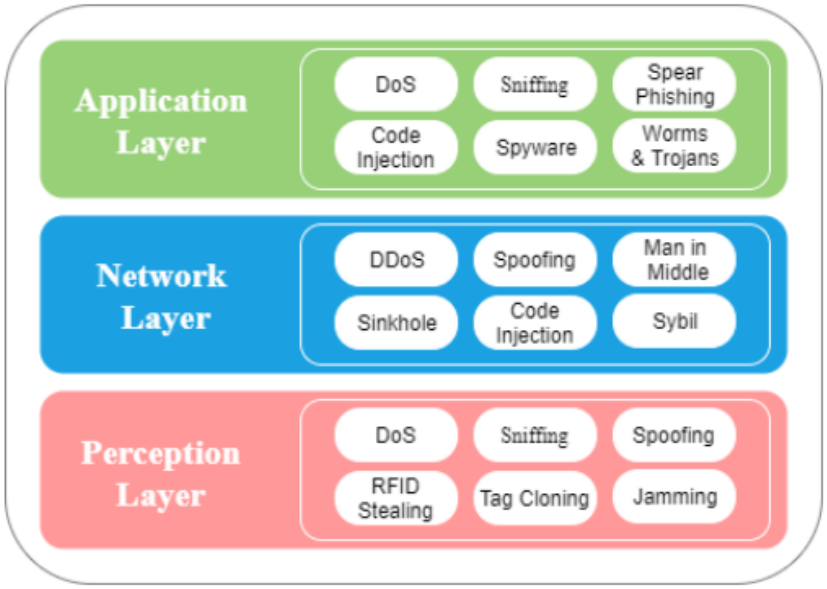

Figura 1.4: Principais ataques em camadas do IoT [4].

1.2 OBJETIVOS

Este trabalho pretende apresentar um *framework* completo para a detecção de intrusão de ataques DoS próximo ao tempo real, aplicando soluções relacionadas à segurança cibernética na camada de aplicação, desenvolvido para ambientes IoT, através de técnicas de aprendizado de máquina.

1.3 OBJETIVOS ESPECÍFICOS

1. Identificar quais os melhores filtros para o armazenamento de dados de cada pacote.
2. Identificar quais são as características do método de ataque DoS na camada de aplicação em sistemas IoT.
3. Encontrar as melhores *features*, para o sistema, presentes no Dataset utilizado.
4. Encontrar o modelo de aprendizado de máquina como melhores métricas para o sistema.
5. Comparar aplicação desenvolvida com as já existentes utilizando diversas métricas, proporcionando generalidades.

1.4 METODOLOGIA

A metodologia utilizada consistiu em um pesquisa quantitativa, na qual foi baseada em números e gráficos para chegar a um resultado plausível, de forma a comparar com os resultados já presentes na lituratura atual. Vale ressaltar que, para chegar nestes resultados numéricos, foram executados experimentos utili-

zando provas de conceitos (PoC), detalhadas neste documento. O passo a passo metodológico está descrito a seguir.

- Realizar uma revisão bibliográfica detalhada acerca de vulnerabilidades na camada de aplicação em sistemas IoT, selecionando a mais crítica atualmente.
- Realizar uma revisão bibliográfica detalhada acerca de soluções já apresentadas, utilizando aprendizado de máquina.
- Criar um protótipo de *framework* utilizando modelos de aprendizado de maquina, de modo a identificar e especificar ataques à dispositivos IoT.
- Criar, em um emulador de redes computacionais, um cenário de modo a ser validado o sistema (utilizando a ferramenta ELK com o plugin Wazuh para monitorar o sistema).
- Apresentar uma modelo de integração entre o sistema de monitoramento e o *framework* desenvolvido.
- Executar experimentos, de modo a validar o *framework* pelas métricas apresentadas na literatura.

1.5 CONTRIBUIÇÕES DO TRABALHO

Este trabalho apresenta as seguintes contribuições:

- A redução de 41 para 17 *features* do dataset NSL-KDD através de métodos de correlação de variáveis, mantendo os valores similares nas principais métricas da literatura.
- A criação de um formato de modelo de log próprio para alertas no Wazuh Manager.
- A criação de decodificadores personalizados na plataforma Wazuh, para a identificação do novo modelo de log criado.
- Adaptação da captura dos dados em tempo real (através de códigos desenvolvidos e registrados com programa de computado no INPI (BR512023001038-3)), transformando em conexões estabelecidas, convertendo-as em dados similares as encontrados nos principais datasets da literatura.
- Emulação do envio e recebimento do dados de sensores pelo sistema Raspbian para testes em Real-Time.
- A criação de um sistema próximo ao tempo real para a identificação de ataques DoS, utilizando o NSL-KDD como dataset de treino.
- Valores expressivos das principais métricas, quando comparados com trabalhos anteriores, no reconhecimento de ataques DoS no dataset de teste do NSL-KDD utilizando o método *Random Forest* (XGBClassifier).

1.6 ORGANIZAÇÃO DO TRABALHO

Esta dissertação está estruturada da seguinte forma: No capítulo 2, serão apresentados e descritos o referencial teórico e os trabalhos correlatos presentes na literatura; No capítulo 3 a metodologia e a arquitetura proposta; No capítulo 4 os experimentos realizados, seus resultados e suas discussões; No capítulo 5 a conclusão do trabalho realizado, suas limitações, bem como os trabalhos futuros.

2 REFERENCIAL TEÓRICO E TRABALHOS CORRELATOS

Neste Capítulo são apresentados os conceitos julgados necessários para entendimento total do presente trabalho, bem como os trabalhos correlatos a este trabalho desenvolvido.

2.1 INTERNET DAS COISAS - IOT

Na literatura, o termo "Internet das Coisas - IoT" foi citado pela primeira vez por Ashton [33]. Segundo ele está frase existiu em sua apresentação na Procter & Gamble (P&G) em 1999. Sua ideia era "Associar a nova ideia de RFID na cadeia de fornecimento da empresa ao tema internet, de forma a chamar a atenção dos executivos". Outras literaturas como Bauer et al. [34], definem como "uma rede mundial de objetos interconectados que são endereçados de forma única, baseada em protocolos de comunicação já existentes". Já segundo Mendonça [35], "Não podemos falar a rigor em uma rede mundial para IoT, nem na possibilidade de um mesmo conjunto de protocolos em todos os casos de IoT". Outros autores como Sundmaeker et al. [36], definem IoT como "uma rede de infraestrutura global dinâmica, na qual dispõe de capacidades de autoconfiguração, baseado em conexões físicas e virtuais de "coisas", que possuem identidades, atributos físicos e interfaces inteligentes". Höller et al. [37] em seu trabalho, ainda enfatiza que "O IoT também se refere à conexão de tais sistemas e sensores à Internet inteira, assim como ao uso de tecnologias gerais da Internet. No longo prazo, é previsto que o ecossistema IoT se tornará equivalente à Internet que temos hoje, permitindo que as coisas e objetos do mundo real se conectem, comuniquem-se e interajam entre si do mesmo jeito que humanos fazem pela *web*". Segundo Oracle [38], o IoT pode ser definido como a tecnologia que "descreve a rede de objetos físicos incorporados a sensores, software e outras tecnologias com o objetivo de conectar e trocar dados com outros dispositivos e sistemas pela internet. Esses dispositivos variam de objetos domésticos comuns a ferramentas industriais sofisticadas. Com mais de 7 bilhões de dispositivos IoT conectados hoje, os especialistas esperam que esse número cresça para 10 bilhões em 2020 e 22 bilhões em 2025". Dessa forma, vemos que o conceito de IoT está distante de ser algo estritamente irrefutável, entretanto, é possível observar que o mesmo se apresenta como um paradigma universal, no qual utiliza-se de tecnologias existentes para a comunicação entre máquinas.

2.2 *MESSAGE QUEUING TELEMETRY TRANSPORT* - MQTT

O *Message Queuing Telemetry Transport* é um protocolo do tipo *publish–subscribe* especificado pela ISO/IEC 20922, tendo seu inicio no ano de 1999. Este tipo de arquitetura de software tem por característica, o envio não direcionado as receptores específicos, onde, diferentemente do convencional, categorizam as mensagens publicadas em classes sem conhecimento dos *subscribes*, chamadas de tópicos. Desta maneira,

os *subscribes* precisam manifestar o interesse em certos tópicos de mensagens para recebê-las. De acordo com [39], o MQTT possui algumas vantagens claras, tais como:

- **Ser Leve e Eficiente:** Os clientes MQTT são muito leves, sendo possível utiliza-los até em microcontroladores. Seus cabeçalhos são curtos, proporcionando eficiência no transporte dos dados.
- **Comunicações Bidirecionais:** Permitem comunicações Bidirecionais entre nuvens de dispositivos.
- **Escalonável:** O MQTT pode ser facilmente escalonável, sendo frequentemente utilizado em dispositivos IoT.
- **Entrega confiável da mensagem:** Possui três níveis de confiabilidade na entrega dos dados.
- **Suporte de uso em redes instáveis:** Possui conexões persistentes que reduzem o tempo de reconexão com o *broker*.
- **Módulo seguro:** Possui a opção de uso com TLS e protocolos de autenticação.

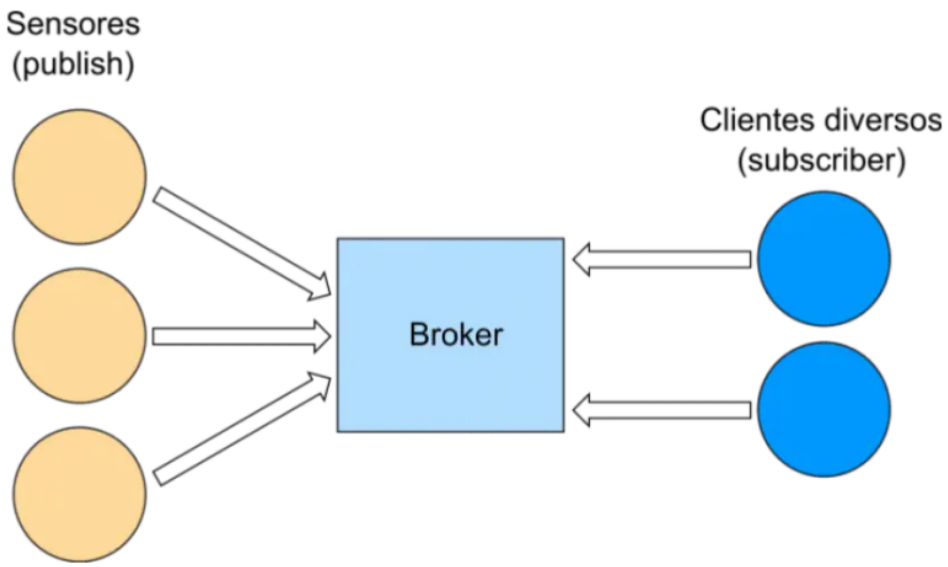

Figura 2.1: Exemplo MQTT [5].

O MQTT possui dois atores, os clientes e o controlador chamado de *broker*. Os clientes possuem duas áreas de atuação, a primeira delas é ser um *publish*, onde por conseguinte, será necessário publicar mensagens em algum tópico. A segunda área de atuação está relacionada em ser um *subscribe*, no qual receberá as mensagens enviadas pelos *publishes*. A comunicação funciona desta forma (conforme apresentado na Figura 2.1), o cliente *publish* envia a mensagem para o *broker* especificando seu tópico. O *broker* armazena a mensagem recebida e envia mensagens para todos os *subscribes* que manifestaram interesse nesse tópico, disponibilizando assim, a mensagem em uma fila especifica. Os clientes *subscribes*, ao receber a mensagem, vão à fila e capturam os dados recebidos.

2.3 TSHARK

O Tshark é um ferramenta para a analise de tráfego de redes. Nele é possível obter uma captura em tempo real de todos os pacotes presentes na rede, além de analisar pacotes em tráfegos previamente capturados. O Tshark possui diversos filtros herdados do wireshark, sendo possível capturar dados específicos de cada pacote na rede. Como saída, o Tshark possui algumas opções, sendo elas a escrita automatica em um arquivo do tipo csv, além de ter a opção de envio direto dos dados para o ELK. A sua versão mais utilizada é chamada de pyshark, onde é possível executar as funções do tshark com comandos python, facilitando assim, o manuseio dos dados capturados.

2.4 BANCO DE DADOS

Banco de dados, segundo Oracle [40], é uma "coleção organizada de informações - ou dados - estruturadas, normalmente armazenadas eletronicamente em um sistema de computador". Estes bancos de dados, geralmente, são gerenciados por um SGBD (Sistema Gerenciador de Banco de Dados), no qual é responsável por varias tarefas, tais como: criar relações entre tabelas, eliminar e copiar arquivos, efetuar consultas nas tabelas, criar e gerir usuários e suas visões, entre outros (veja a Figura 2.2, que mostra como o SGBD funciona no tráfego de dados). Desta forma, existem diversos tipos de SGBDs, onde podemos destacar: os relacionais, os não relacionais, os hierárquicos, os de redes e os orientados a objetos.

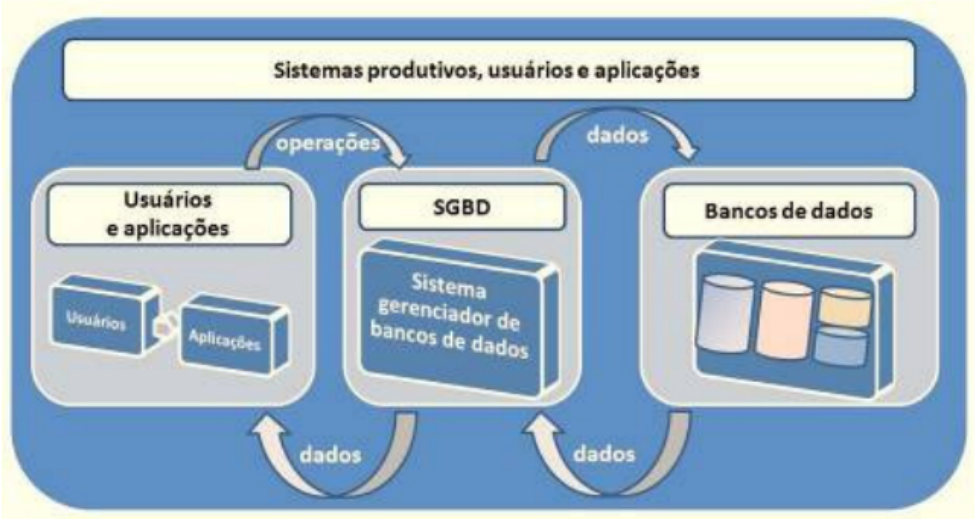

Figura 2.2: Exemplo do funcionamento de um SGBD [6].

2.4.1 Postgres

O Postgres é um SGBD do tipo relacional de código aberto com mais 30 anos de desenvolvimento ativo, onde se tornou um dos SGDB mais confiáveis pela sua robustez, confiabilidade e desempenho. O Postgres, como qualquer SGBD do tipo relacional, modela os dados em formatos de tabelas, utilizando-se de relações, proporcionando acesso, manipulação e recuperação facilitada aos dados armazenados.

2.4.2 Redis

O Redis [41] é um banco de dados do tipo par chave-valor, na qual cada informação armazenada possui uma chave de referência única. Sua estrutura de dados é armazenado em memória, gerando programabilidade, velocidade, persistência e alta disponibilidade.

2.5 SISTEMA DE DETECÇÃO DE INTRUSÃO - IDS

O IDS (do inglês *Intrusion Detection System*) é um sistema de proteção que monitora a rede para identificação de possíveis ataques de intrusos. Thakkar e Lohiya [42], define IDS como uma ferramenta de proteção usada em sistemas de comunicação e informação. Ele é capaz de examinar as atividade da rede entre os dispositivos conectados, gerando assim, alarmes quando é detectado algo fora do padrão esperado, onde, para isso, os IDS analisam os padrões comportamentais do fluxo de dados, dos horários, entre outras *features*. Atualmente, esse sistema é um mecanismo essencial para redes tradicionais, auxiliando na monitoração e segurança delas. Torna-se importante frisar também que, o IDS não é uma ferramenta de diagnostico de segurança, como um antivírus, pois sua função é apenas monitorar e alertar comportamentos suspeitos, desta forma, a eficácia de um IDS em uma rede tradicional está fortemente ligada a qualidade de sua configuração. Andrea et al. [43], por sua vez, destaca que, mesmo sendo uma ferramenta fundamental nas redes tradicionais, a utilização eficiente do IDS em redes IoT é um desafio, dado que essas redes possuem por característica um processamento e armazenamento limitado. A Figura 2.3 mostra a funcionamento de um IDS em uma topologia convencional.

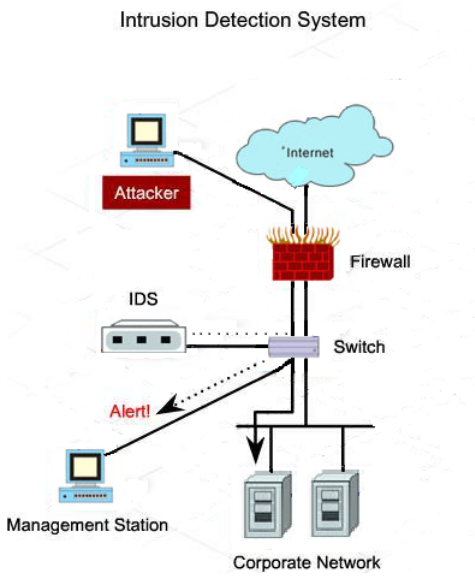

Figura 2.3: Funcionamento de um IDS em uma topologia convencional. ADAPTADA - [7].

2.6 *NETWORK SECURITY LABORATORY-KNOWLEDGE DISCOVERY IN DATABASES* - NSL-KDD

O NSL-KDD [21] é um dataset criado em 2009, onde é utilizado para a detecção de ataques por anomalias, no qual, consiste em um update da versão KDD cup99, visto que resolve alguns problemas existente na versão anterior. Como é possível observar na Figura 2.4, o dataset possui 41 *features* para auxiliar na decisão entre tráfego normal ou tráfego com potencial ataque. Uma das 41 *features* presentes no dataset, existe uma especial chamada *flags*, no qual, possui onze possibilidades de ocorrência. A Figura 2.5 mostra as onze possibilidades. Os seus ataques são classificados em quatro tipos de categorias, são elas: *Denial of service, Surveillance and other probing attacks, Unauthorized access to local superusers, Unauthorized access from a remote machine*. Os ataques presentes estão descritos na Tabela 2.1.

Category I	duration (0–to–54451)	protocol type (1, 2, 3)	service (1–to–70)	src_bytes (0–to–1379963888)	dst_bytes (0–to–309937401)
	Flag (1–to–11)	Land (0, 1)	wrong_fragment (0, 1, 3)	Urgent (0–to–3)	
Category II	Hot (0–to–101)	num_failed_logins (0–to–4)	logged_in (0,1)	num_compromised (0–to–7479)	root_shell (0, 1)
	su_attempted (0, 1)	num_root (0–to–7468)	num_file_creations (0–to–100)	num_shells (0–2)	num_access_files (0–9)
	is_guest_login (0, 1)	is_hot_logins (0, 1)	num_outbound_cmds (0)		
Category III	Count (0–511)	srv_count (0–511)	serror_rate (0–to–1)	srv_serror_rate (0–to–1)	rerror_rate (0–to–1)
	srv_rerror_rate (0–to–1)	same_srv_rate (0–to–1)	diff_srv_rate (0–to–1)	srv_diff_host_rate (0–to–1)	
Category IV	dst_host_count (0–to–255)	dst_host_srv_count (0–to–255)	dst_host_same_srv_rate (0–to–1)	dst_host_diff_srv_rate (0–to–1)	dst_host_same_src_port_rate (0–to–1)
	dst_host_srv_diff_host_rate (0–to–1)	dst_host_serror_rate (0–to–1)	dst_host_srv_serror_rate (0–to–1)	dst_host_rerror_rate (0–to–1)	dst_host_srv_rerror_rate (0–to–1)
	class_label (0–to–1)				

Figura 2.4: *Features* presentes no dataset NSL-KDD [8].

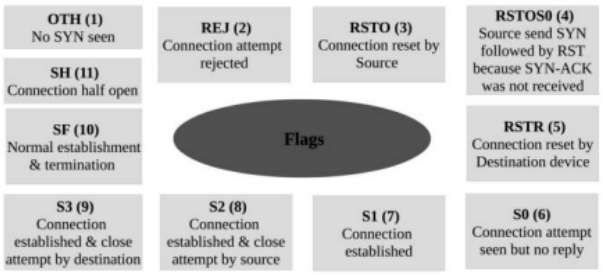

Figura 2.5: Onze possíveis *flags* [8].

O dataset conta com mais de 125 mil exemplos para treino, onde 53,46% são tráfegos normais e 43,07% são tráfegos de ataques. Vale a pena ressaltar que, só os ataques de DoS somam cerca de 36,46% dos dados do dataset. Veja a Tabela 2.2. Atualmente, o NSL-KDD se tornou uma das referências de dataset na área de detecção de ataques por anomalia, haja visto que é utilizado como um dos datasets de avaliação de métricas, de modo a comparar cada solução proposta na literatura.

Tabela 2.1: Lista dos ataques presente no dataset NSL-KDD.

Categoria do ataque	Nome do ataque
Denial of service (DoS)	Apache2,Smurf,Neptune,Back,Teardrop, Pod,Land,Mailbomb,Processtable,UDPstorm
Remote to local (R2L)	WarezClient,Guess_Password,WarezMaster, Imap,Ftp_Write,Named,MultiHop,Phf,Spy, Sendmail,SnmpGetAttack,SnmpGuess, Worm,Xsnoop,Xlock
User to Root (U2R)	Buffer_Overflow, Httptuneel, Rootkit, LoadModule, Perl, Xterm, Ps, SQLattack
Probe (or Surveillance)	Satan, Saint, Ipsweep, Portsweep, Nmap, Mscan

Tabela 2.2: Distribuição dos dados no dataset de treino do NSL-KDD.

KDD dataset	Exemplos Totais	Normal	DoS	R2L	U2R	Probe
KDD train	125,973	53.46%	36.46%	0.79%	0.04%	9.25%
KDD test	22,544	43.07%	33.08%	12.22%	0.89%	10.74%

2.7 APRENDIZADO DE MÁQUINA

Segundo Mitchell [44], aprendizado de máquina (uma das vertentes da inteligência artificial) é um campo de ensino focado em como construir programas de computador que poderão melhorar automaticamente com as experiências passadas. Nisso, vemos que sua ideia visa otimizar critérios de desempenho através de exemplos de dados ou experiencia passada. Desta forma, apresenta-se dois grande campos do conhecimento que fazem parte do aprendizado de máquina, o primeiro deles é o campo estatístico, onde é muito usado para inferência nas amostras de dados. O segundo campo é o da ciência da computação, na qual apresenta algoritmos computacionais eficientes para representar modelos estatístico.

Atualmente, o aprendizado de máquina é subdividido em três grandes grupos de aplicações, são eles:

- **Aprendizado supervisionado:** Neste tipo de aprendizado, o algoritmo, através de exemplos de dados, aprenderá por meios estatísticos a prever os resultados futuros.
- **Aprendizado não-supervisionado:** No aprendizado não-supervisionado, não será apresentados exemplos, pois os algoritmos irão prever os resultados futuros através do reconhecimento de padrões dos dados.
- **Aprendizado por reforço:** Este tipo de aprendizado funciona utilizado a ideia recompensa da ação tomada. Dessa forma, toda ação executada gera uma reação, de modo que o algoritmo irá prever os resultado futuros maximizando cada recompensa em todas as ações.

2.7.1 Random Forest

Random Forest é um método de aprendizado de máquina do tipo supervisionado, onde utiliza a combinação de árvores de decisão. Breiman, em seu trabalho [45], define Random Forest como uma combinação de árvores de decisão, onde cada árvore depende dos valores de vetores aleatórios amostrados independentemente um dos outros, dentre as quais, essas árvores tenham distribuições idênticas em toda a floresta disposta. Ao final haverá uma votação pela maioria, de modo a decidir o resultado final da predição. Veja a Figura 2.6, na qual apresenta o funcionamento do Random Forest.

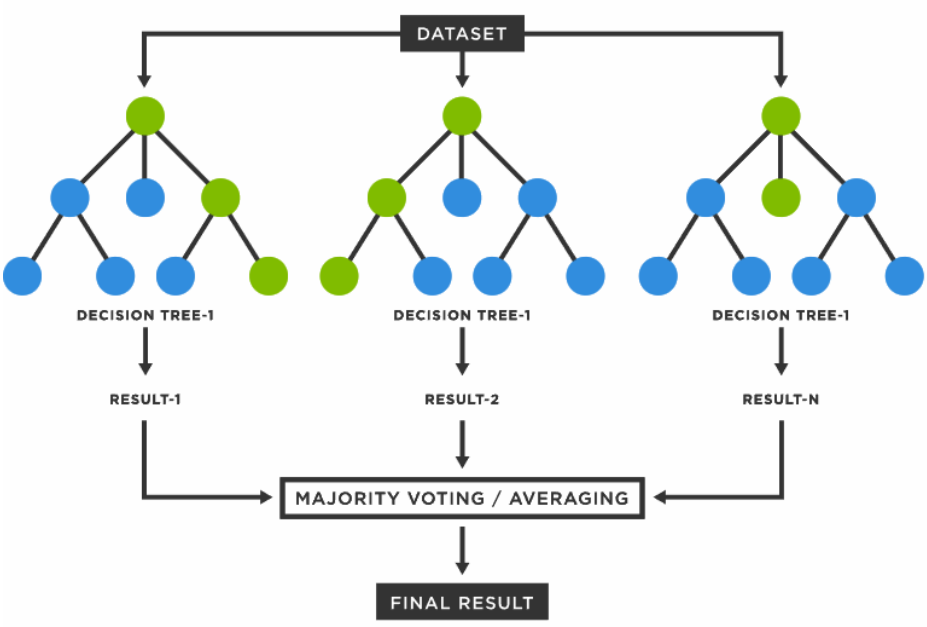

Figura 2.6: Funcionamento Random Forest [9].

2.8 *MACHINE LEARNING OPERATIONS* - MLOPS

MLOps, de acordo com Treveil et al. [46], pode ser definido com processo que auxilia organizações e lideres de negócios a gerar valor a longo prazo e reduzir os riscos relacionados a ciência de dados, aprendizado de máquinas e outras áreas da inteligência artificial, ou seja, em resumo, MLOps é a padronização e o entendimento do gerenciamento do ciclo de vida de modelos de aprendizado de máquina.

O MLOps surge para auxiliar nos desafios atuais presentes no processo de gerenciamento do ciclo de vida de modelos de aprendizado de máquinas, tais como:

- **A existência de muitas dependências envolvidas:** Tanto os dados com os objetivos das organizações envolvidas estão em constante mudança, refletindo no processo de decisão do modelo de aprendizado de máquina.
- **Nem todos falam a mesma língua:** O processo de gerenciamento do ciclo de vida de modelos de aprendizado de máquina envolve diversos profissionais de áreas distintas, tais como: cientista de

dados, time de TI, pessoas da administração, entre outras. Nem todos esses profissionais envolvidos possuem o conhecimento necessário sobre aprendizado de máquina e seu ciclo de vida.

- **Cientistas de Dados não são Engenheiros de Software:** Nem todo cientista de dados possui o conhecimento de gerenciamento de ciclo de vida de softwares, algo bastante explorado na engenharia de software.

Com isso, MLOps apresenta algumas soluções para a resolução dos desafios apresentados, dentre os quais podemos citar:

- Automatização robusta e confiável entre os times envolvidos.
- A ideia da colaboração e a comunicação incremental entre os times.
- Serviço ponto a ponto do ciclo de vida (criações, testes, versões).
- Priorização contínua na entrega de alta qualidade.

2.8.1 MLFlow

MLFlow é uma plataforma de código aberto para o gerenciamento do ciclo de vida de modelos de aprendizado de máquina, utilizando conceito de MLOps. Nessa plataforma é possível criar experimentos, projetos e modelos, sendo que para isso é utilizado quatro componentes: MLFlow Tracking, MLFlow Projects, MLFlow Models, and Models Registry. De acordo com [47], o primeiro componente é responsável por armazenar os logs dos parâmetros relacionados, criar versões do código, gerenciar métricas e os arquivos de saída enquanto o código do modelo de aprendizado de máquina está sendo executado. O segundo componente é responsável por encapsular o código, de modo a ser reutilizável. O terceiro componente encapsula o modelo para ser reutilizável. O quarto, e último componente, armazena e gerencia o ciclo de vida do modelo criado.

2.9 ELK - *ELASTIC STACK*

O *ELK stack* é um conjunto de três ferramentas: o Elasticsearch, o Logstash e o Kibana. Estas três ferramentas foram pensadas para serem usadas em conjunto, representando um solução integrada para a gerência e monitoramento (a Figura 2.7 representa a dinâmica das ferramentas em conjunto). Cada uma das ferramentas será apresentada a seguir.

- **Elasticsearch:** Segundo [48], fundamenta-se como um banco de dados NoSQL, no qual foi baseado no apache Lucene, uma excelente ferramenta de pesquisa de código aberto, distribuível e altamente escalável que foi concebido para ter um desempenho ótimo. Possui uma API REST, na qual permite monitorar e controlar todos os aspectos de uma configuração em cluster. O Elasticsearch oferece implementação simples, máxima flexibilidade, e fácil gerenciamento. Na ELK stack o Elasticsearch

Figura 2.7: Arquitetura ELK stack [10].

é responsável por prover o armazenamento, a pesquisa e a analise de dados de grande e pequeno volume.

- **Logstash:** Logstash é uma ferramenta de linha de produção de dados, sendo que os dados enviados do sistema são ingeridos na entrada do Logstash, onde, normalmente, a sua saída segue direcionada para o Elasticsearch. O Logstsh possui três componentes básico, o primeiro deles é a entrada dos dados, no qual recebe os dados de entrada do ELK stack. O segundo é a filtragem dos dados, onde é possível programar filtros específicos para os dados que passarão na linha de produção. O terceiro é a saída dos dados, sendo necessária especificação para onde será enviado os dados de saída do Logstash. Vala a pena frisar que, o Logstash é independente das outra ferramentas presentes no ELK stack. A Figura 2.8 mostra um exemplo do uso do Logstash em uma arquitetura.

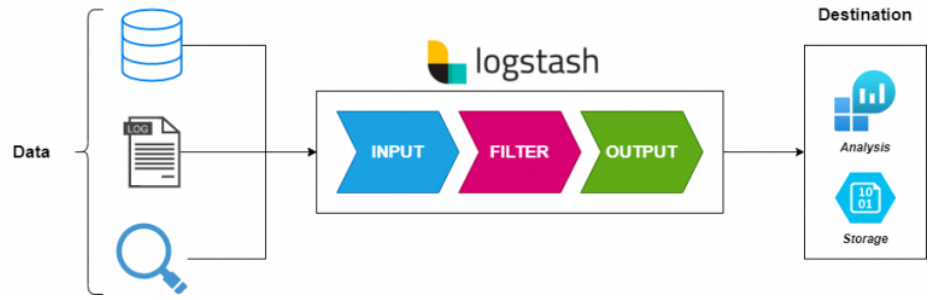

Figura 2.8: Exemplo de arquitetura com o uso do Logstash [11].

- **Kibana:** O kibana é o visualizador de dados presente no ELK stack. Ele é ligado ao Elasticsearch, sendo usado para contemplar os dados, alarmes e métricas do gerenciamento, através do seu *dashboard*. Com ele é possível criar gráficos, tabelas e mapas personalizados com os dados fornecidos pelo banco de dados do Elasticsearch. Dessa forma, é possível gerar valor aos dados, tornando-se um ferramenta essencial para o gerenciamento de sistemas.

2.10 WAZUH

Wazuh é uma plataforma de código aberto para a detecção e monitoramento de ameaças com regras pré-definidas. Conforme descrito por STANKOVIĆ et al. [49], esta ferramenta pode ser utilizada para monitoração de dispositivos como computadores, notebooks, servidores ou dispositivos de rede. O Wazuh possui algumas capacidades específicas, segundo STANKOVIĆ et al. [49]. São elas:

- **Análises de Segurança:** Coleção, agregação, indexação e processamento de dados visando segurança.
- **Detecção de alguns ataques específicos:** O agente Wazuh pode escanear e monitorar o agente para a detecção de malwares e rootkits.
- **Análise de Logs:** O agente Wazuh pode ler arquivos de log, gerando alarmes através de regras pré-estabelecidas.
- **Monitoração de integridade de arquivos:** O Wazuh monitor arquivos de sistema, identificando as mudanças, permissões, donos e atribuições em arquivos que necessitam de atenção.
- **Detector de vulnerabilidade:** O agente Wazuh analisa o dados de inventário e envia as informações para o servidor Wazuh, no qual verifica periodicamente as vulnerabilidades presentes no CVEs (do inglês *Common Vulnerabilities and Exposures*).
- **Avaliação de configuração:** O Wazuh verifica as principais configurações e aplicações de acordo com as politicas e padrões estabelecidos.

2.11 DOCKER

Docker é uma virtualizador de *container* de código aberto que auxilia na criação e administração ambientes. Ele possui uma grande vantagem quando comparado as máquinas virtuais comuns, pois por utilizar uma arquitetura diferente, consegue ser muito mais leve e eficiente. Isso acontece pelo fato do Docker não utilizar a camada de *hypervisor* (veja a Figura 2.9). Para isso o Docker utiliza o conceito de *containers*, onde, por sua vez, pode ser considerado um ambiente que contém um conjunto de processos a serem executados a partir de uma imagem definida.

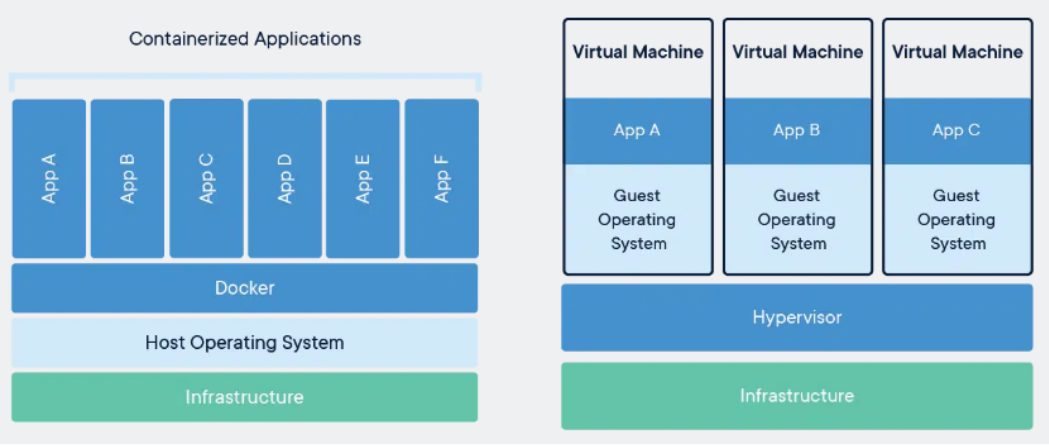

Figura 2.9: Arquitetura convencional dos *containers* à esquerda e das máquinas virtuais à direita. [12].

2.12 *ADVANCED MESSAGE QUEUING PROTOCOL* - AMQP

O AMQP é um protocolo leve criado para a comunicação M2M (do inglês *machine to machine*), no qual foi desenvolvido por John O'Hara pela empresa JPMorgan Chase em Londres no ano de 2003. Segundo NAIK [50], o AMQP é um protocolo de mensagens corporativo desenvolvido para ser confiável, seguro, ter aprovisionamento e interoperabilidade. Este protocolo utiliza a arquitetura *Request/Response*, ou seja, são utilizadas filas entre o *publisher* e o *consumer* para gerenciar o transporte de dado, evitando a arquitetura tradicional de comunicação. Dessa forma, temos que, a arquitetura convencional possui o diálogo síncrono, onde há um tempo de espera entre o envio da mensagem e o recebimento da mesma (a Figura 2.10 mostra a arquitetura convencional). Já com o AMQP, utiliza-se um intermediário que atua entre o remetente e o destinatário evitando o tempo de espera para o recebimento da mensagem (a Figura 2.11 mostra a arquitetura com o AMQP).

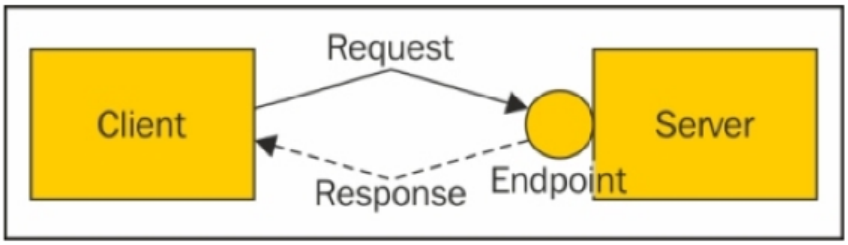

Figura 2.10: Arquitetura convencional de troca de mensagens [13].

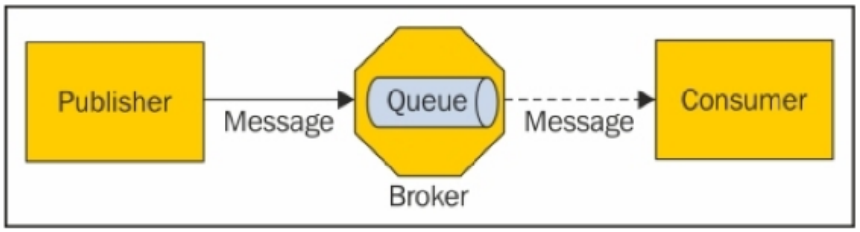

Figura 2.11: Arquitetura de via única utilizando o protocolo AMQP [13].

2.13 TRABALHOS CORRELATOS

A utilização de IDS em redes convencionais é algo bem consolidado na topologia básica de qualquer empresa do mercado atual. Com o grande crescimento do IoT, e suas características peculiares já citadas no Capítulo 1, tornou-se necessária a utilização de métodos de segurança aprimorados para tal fim. Dado isso, observou-se que a utilização de técnicas de monitoramento de segurança, tais como o IDS, adaptadas corretamente, poderia ser uma das ferramentas mais fundamentais para a segurança em sistemas IoT. Com isso, surgiu diversos trabalhos científicos executando provas de conceito, a fim de identificar quais são as adaptações necessárias para tornar o IDS comum em uma ferramenta fundamental na segurança de

sistemas IoT. Esta seção abortará alguns trabalhos presente na literatura que obtiveram sucesso em suas devidas abordagens.

2.13.1 IDS focados em IOT

Nesta subseção será apresentados os principais trabalhos correlatos que desenvolveram IDS pensados em redes IoT. Cada tópico seguinte representa um trabalho correlato.

2.13.1.1 FlowGuard

No trabalho de Jia et al. [14] é apresentado um mecanismo de defesa em dispositivos de borda contra ataques DDoS em sistemas IoT chamado de FlowGuard. Esse sistema foca na detecção, identificação, classificação e mitigação de ataques DDoS em IoT. Segundo os autores, o FlowGuard apresenta três objetivos fundamentais, são eles:

1. Com o FlowGuard, fluxos de inundação maliciosos devem ser detectados e processados imediatamente, sem afetar outros fluxos de rede legítimos [14].
2. Com o FlowGuard, mesmo enfrentando ataques DDoS poderosos, os servidores de borda devem ser capazes de satisfazer os requisitos mínimos de comunicação de rede legítima e não devem parar [14].
3. O FlowGuard deve ser eficiente para comunicações de rede normais e ser atualizável para lidar com ataques *Zero-Day DDoS* [14].

A estrutura de defesa apresentada no projeto é constituída pelo sistema criado presente em servidores de borda do sistema IoT. Essa estrutura de defesa é apresentada na Figura 2.12 (em *Edge Server*), onde, por sua vez, possui dois principais componentes. O primeiro deles é chamado de *Flow Filter*, onde em sintase, utiliza uma tabela de dados para a identificação do possível ataque, levando em consideração os recursos mais relevantes para cada tipo de DDoS específico, de modo a realizar as devidas intervenções imediatas. O segundo componente é chamado de *Flow Handler*, no qual é responsável por examinar os fluxos suspeitos provenientes do *Flow Filter* e identificar e classificar possíveis ataques DDoS. Segundo os autores, para essa identificação e classificação dos ataques foi criado um algoritmo de detecção de ataques DDoS baseado em variações de tráfego, através da elaboração de dois modelos de aprendizado de máquinas, sendo chamados de *DDoS Identification Module* e *DDoS Classification Module*, onde são baseados nos modelos de rede neural chamados CNN e LSTM. Como dataset de treino para o sistema, foi utilizado o CICDDoS2019, provido pelo instituto de segurança cibernética do Canadá (CIC). Este dataset inclui mais de 1 milhão de tráfegos benignos e mais de 30 milhões de tráfegos maliciosos, incluindo ataques DDoS recentes, tais como: NTP, DNS, LDAP, MSSQL, e NetBIOS.

Para testes foi criado um modelo de arquitetura de ataque, na qual é apresentado na Figura 2.13. Para executar os ataques, foi elaborado um experimento utilizando BoNeSi e SlowHTTPTest para disparar diferente tipos de ataques DDoS a uma taxa de 10.000 pacotes por segundo, de modo a esgotar os recursos da vítima, onde foram coletados mais de 1 milhão de pacotes maliciosos. A Tabela 2.3, mostra a tabela apresentada pelos autores, no qual especifica a quantidade de pacotes e fluxos gerados nos ataques efetuados.

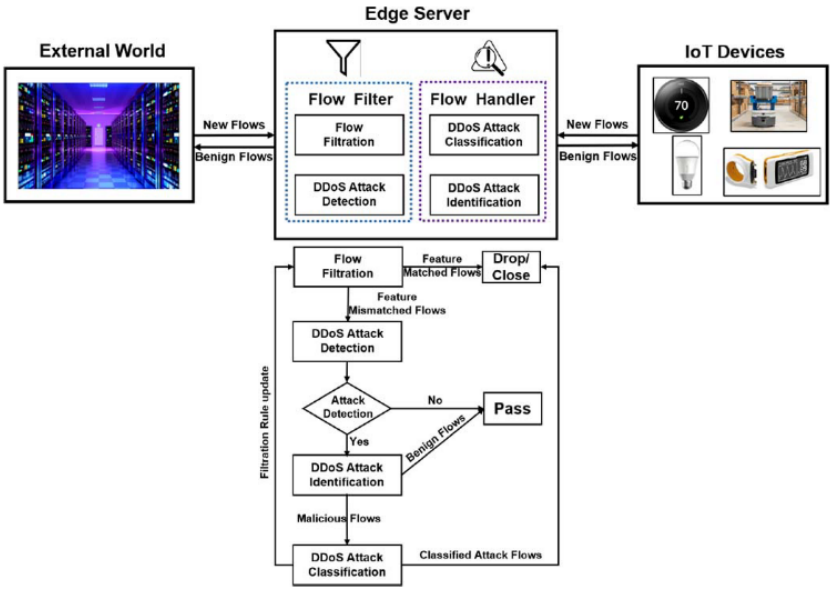

Figura 2.12: Visão geral do funcionamento do FlowGuard [14].

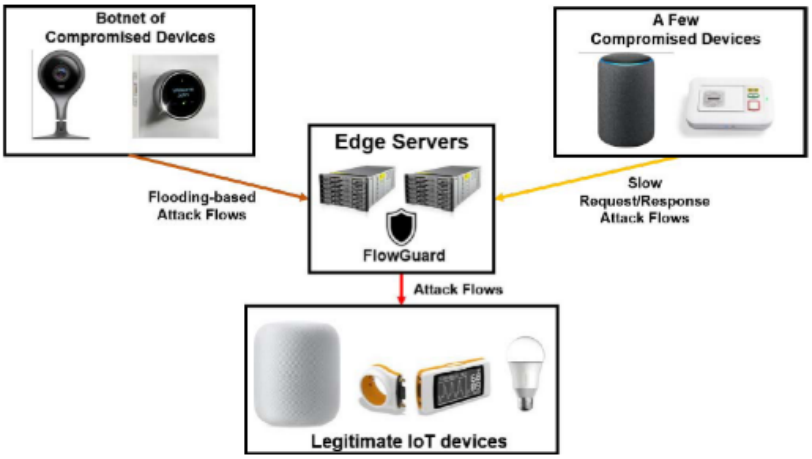

Figura 2.13: Modelo de ameaça da arquitetura criada [14].

Tabela 2.3: Tabela criada pelos autores para especificar a quantidade de pacotes e fluxos envolvidos nos ataques [14].

Attack Types	Total Packets	Total Flows
ICMP Flooding Attacks	1,000,011	999,974
UDP Flooding Attacks	1,000,039	999,997
TCP Flooding Attacks	1,000,061	1,000,000
Slow Headers Attacks	1,003,507	21,087
Slow Read Attacks	1,000,109	22,015
Slow Write Attacks	1,000,085	21,137
Slow Body Attacks	1,002,851	21,535

Foram utilizados como métricas de comparação a acurácia, a precisão, o Recall (Taxa de predições corretas em fluxos malicioso dividido pelo fluxo malicioso total) e o F1-score (média ponderada entre a precisão e o Recall). Com relação aos resultados obtidos, os autores declararam que resultados indicam que a acurácia da identificação utilizando o LSTM chega a 98,9%, o que supera significativamente alguns modelos de aprendizagem bem conhecidos pela maioria dos trabalhos da literatura. A precisão da classificação do CNN proposta é de até 99,9%. Além disso, o modelo criado pelos autores atendem satisfatoriamente aos requisitos de atraso do IoT quando implantados em servidores de borda com poderes computacionais superiores a um computador pessoal.

2.13.1.2 Passban IDS

O trabalho de Eskandari et al. [15] apresenta um sistema de detecção de intrusão chamado Passban, no qual é capaz de proteger os dispositivos IoT diretamente conectados a ele. A solução proposta pode ser instalada diretamente nos *gateways* IoT, diminuído os gastos com servidores potentes. Seus principais objetivos são:

1. Assegurar a proteção de dados perto da fonte de dados IoT. Para isso, foi desenhado um IDS que reside o mais próximo possível dos dispositivos IoT, ou seja, na borda da rede ou no dispositivo de borda (*edge device*) [15].
2. Conceber um IDS capaz de escalar facilmente após a implantação, sendo capaz de gerir sempre novas definições de ameaças, sem exigir atualizações de hardware [15].
3. Fornecer um IDS mais caracterizado por uma taxa de falso-positivos reduzida e uma precisão de detecção satisfatória, uma vez que este último requisito é crucial para que um IDS seja verdadeiramente eficaz [15].

A arquitetura do trabalho apresentado por Eskandari et al. [15] é subdividida em fases. A Figura 2.14 mostra as duas fases presente na arquitetura, onde (a) é a fase responsável pelo treinamento do modelo, e (b) é a fase responsável pela classificação dos ataques na rede. Vemos na Figura 2.14 que, existem quatro blocos nos fluxogramas apresentados, são eles:

- ***Packet Flow Discovery*:** Observa constantemente o tráfego de rede, capturando pacotes brutos e extraindo fluxos de rede [15].
- ***Feature Extraction*:** Calcula estatísticas de fluxo de rede e constrói um conjunto de características utilizadas para treinar o modelo de aprendizagem da máquina. O conjunto de características é normalmente construído com o apoio de um especialista em segurança de rede para assegurar um desempenho optimizado do IDS. O Passban utiliza a ferramenta chamada NetMate, onde ouve simplesmente o tráfego da rede e converte pacotes brutos em fluxos de rede e depois calcula métricas para os fluxos [15].
- ***Train/Load Model*:** A fase de treino do IDS é feita em fluxo normal da rede, ou seja, enquanto o sistema funciona em modo livre de ataque. No final de uma fase de treino, o modelo aprendido é

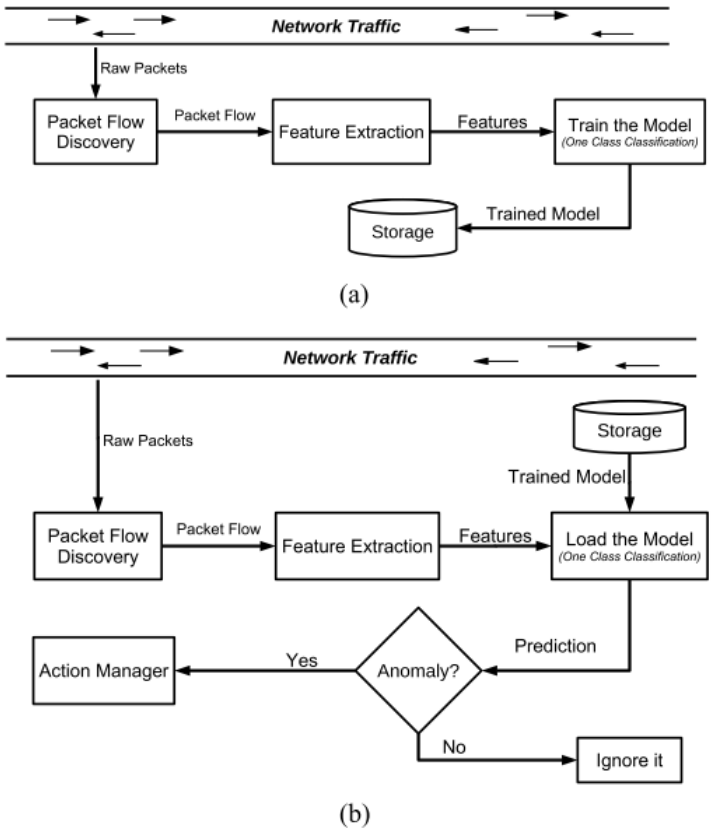

Figura 2.14: Fases presentes no Passban [15].

armazenado/implantado na memória interna do *gateway* para que possa ser utilizado para detectar ataques para o novo tráfego de entrada da rede. A fim de manter o modelo aprendido atualizado, o usuário deve iniciar uma nova sessão de treino sempre que a configuração da rede ser alterada [15].

- ***Action Manager*:** Este módulo é responsável por tomar as ações necessárias de acordo com as decisões geradas pelo modelo aprendido para a detecção de ataques. Alguns exemplos de tais ações são o registo de incidentes de ataque (*logs*), o bloqueio do tráfego de entrada/saída para um dispositivo alvo, a notificação para um determinado usuário sobre o ataque, entre outros [15].

É importante frisar que, o Passban permite a interação do usuário com o IDS, incluindo limpeza e exploração de registos, controle do estado de ativação do IDS e o gerenciamento dos procedimentos de treino do modelo. Para o treinamento do modelo de aprendizado de máquina, foi utilizado o aprendizado não-supervisionado, onde é possível detectar o ataque pelas anomalias geradas nos fluxos da rede. Foram utilizados dois métodos não-supervisionado para comparação de métricas, o primeiro dele é o *Isolantion Forest (iForest)* e o segundo foi o *Local Outlier Factor (LOF)*. Os autores utilizaram um dataset próprio para os treinamentos.

Para os testes foram criados dois cenários distintos, o primeiro deles é apresentado na Tabela 2.4, onde mostra a quantidade de fluxos de ataque e fluxos normais. O dados do segundo cenário são apresentados na Tabela 2.5.

Os resultados obtidos são apresentados nas Tabelas 2.6 e 2.7, onde mostram valores satisfatórios em

Tabela 2.4: Dados dos fluxos do primeiro cenário [15].

Attack Name	#Normal Flows	#Attack Flows
Normal Traffic	3761	0
Port Scanning	148	57
HTTP Brute Force	106	36
SSH Brute Force	870	389
SYN Flood	117	31

Tabela 2.5: Dados dos fluxos do segundo cenário [15].

Attack Name	#Normal Flows	#Attack Flows
Normal Traffic	6845	0
Port Scanning	61	58
HTTP Brute Force	176	45
SSH Brute Force	227	184
SYN Flood	67	16

diversas métricas, podendo até atingir pontuações de *F1-score* superiores a 0,9 em alguns ataques (0,99 no melhor caso e 0,79 no pior caso).

Tabela 2.6: Resultados do primeiro cenário [15].

Attack	*Technique*	*#Normal*	*#Attack*	*FP*	*TP*	*FN*	*TN*	*Precision*	*Recall*	*F1*
Port Scanning	iForest	148	57	1	57	0	147	**0.98**	**1**	**0.99**
	LOF	148	57	10	52	5	138	0.84	0.91	0.87
HTTP Brute Force	iForest	106	36	2	35	1	104	**0.95**	**0.97**	**0.96**
	LOF	106	36	7	35	1	99	0.83	**0.97**	0.89
SSH Brute Force	iForest	870	389	9	370	19	861	**0.98**	**0.95**	**0.96**
	LOF	870	389	7	302	87	863	**0.98**	0.78	0.87
SYN Flood	iForest	117	31	2	27	4	115	**0.93**	**0.87**	**0.9**
	LOF	117	31	5	27	4	112	0.84	0.87	0.85

2.13.1.3 AS-IDS

Otoum e Nayak, em seu trabalho [16], nos apresenta um modelo que combina duas abordagens para detectar ataques conhecidos e desconhecidos em redes IoT, chamado de AS-IDS. A primeira abordagem é a de detecção baseada em anomalias e, a segunda, se refere a detecção baseada em assinatura de ataques. Os principais objetivos propostos por Otum e Nayak [16] são:

1. Conceber uma estrutura híbrida de IDS que possa ultrapassar as questões de detecção baseada em assinatura e detecção baseada em anomalias [16].
2. Reduzir o *overhead* durante a detecção de intrusão pelo *gateway* IoT, através de um processo de filtragem de tráfego [16].
3. Reduzir a dimensionalidade das *features* utilizando processos de *clustering* eficazes antes da detecção de intrusão [16].
4. Superar os desafios na detecção baseada em anomalia, considerando o ambiente relacionado com as métricas estabelecidas [16].

Tabela 2.7: Resultados do segundo cenário [15].

Attack	*Technique*	*#Normal*	*#Attack*	*FP*	*TP*	*FN*	*TN*	*Precision*	*Recall*	*F1*
Port Scanning	iForest	61	58	2	56	2	59	**0.97**	**0.97**	**0.97**
	LOF	61	58	1	25	33	60	0.96	0.43	0.59
HTTP Brute Force	iForest	176	45	1	43	2	175	**0.98**	**0.96**	**0.97**
	LOF	176	45	4	21	24	172	0.84	0.47	0.6
SSH Brute Force	iForest	227	184	3	183	1	224	**0.98**	**0.99**	**0.98**
	LOF	227	184	18	176	8	209	0.91	0.96	0.93
SYN Flood	iForest	67	16	1	11	5	66	**0.92**	**0.69**	**0.79**
	LOF	67	16	2	9	7	65	0.82	0.56	0.67

Em sua arquitetura, o AS-IDS propôs um modelo de IDS que combina a detecção baseada em assinatura e detecção baseada em anomalias. A Figura 2.15 mostra a arquitetura proposta por Otum e Nayak [16].

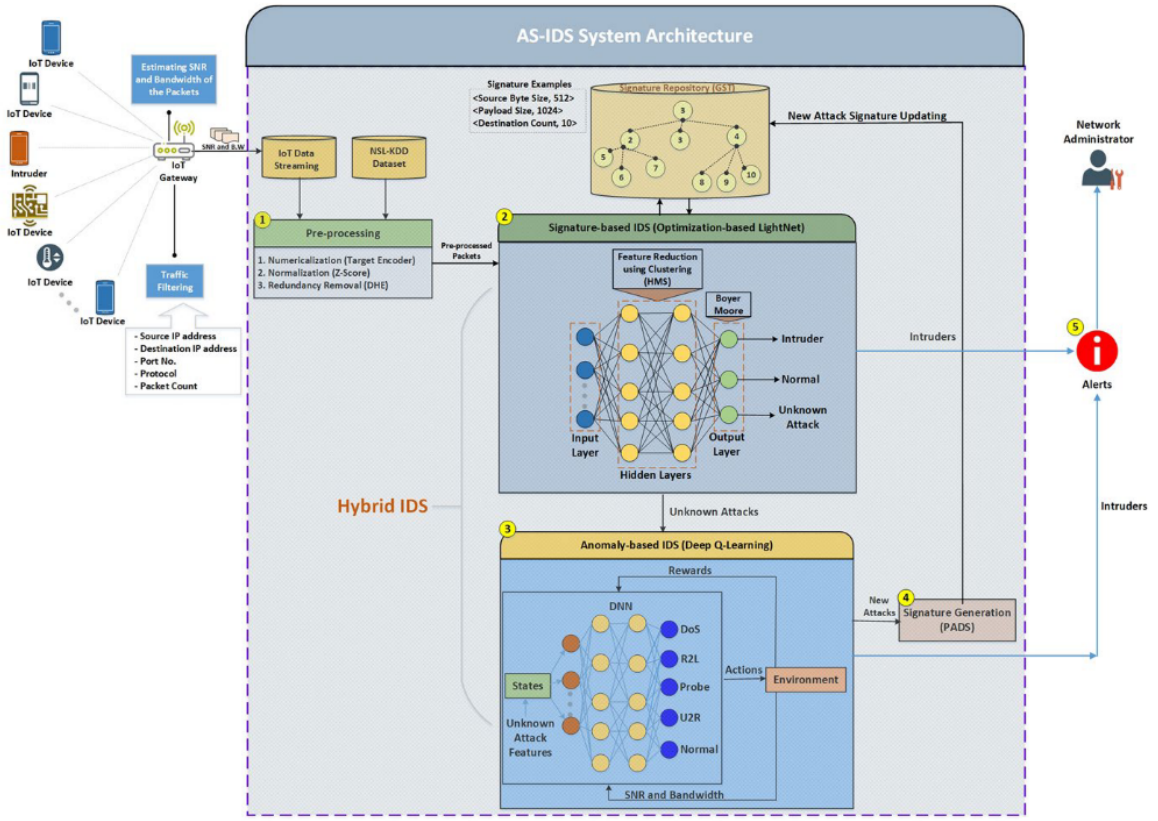

Figura 2.15: Arquitetura do AS-IDS [16].

O modelo apresentado é subdividido em três fases, são elas: filtragem dos dados (*Traffic Filtering*), pré-processamento dos dados (*Preprocessing*) e IDS híbrido (*hybrid IDS*). Cada uma das fases será apresentada com mais detalhe a seguir.

- ***Traffic filtration*:** A primeira fase do sistema proposto é a filtragem do tráfego de entrada. O processo de filtragem é tratado através da validação do endereço IP de origem, endereço IP de destino, números de porta, protocolo e contagem de pacotes. Utilizando estas características, o tráfego chegado é analisado no *gateway*, onde as características anormais são filtradas e os ataques básicos são bloqueados. Para cada pacote não filtrado que passa pelo *gateway* IoT, o SNR e a largura de banda

são computados [16].

- ***Preprocessing*:** Na segunda fase, o pré-processamento do dataset começa por codificar os valores de *strings*, no dataset, em valores numéricos usando o algoritmo de Codificador de Destino (*Target Encoder algorithm*). Por exemplo, o campo de tipo de protocolo tem valores de *strings* TCP, UPD ou ICMP. O algoritmo de Codificador de Destino agrupa os dados por categoria, rastreia o número de ocorrências de cada alvo, e calcula a sua probabilidade, com base no valor médio calculado. Também é executada algumas normalizações de parâmetros e removida redundâncias presentes no dataset [16].

- ***Hybrid IDS*:** O subsistema híbrido IDS tem duas secções principais de processamento que combinam IDS com base em assinatura e IDS com base em anomalias. O IDS baseado na assinatura é executado primeiro para detectar todos os ataques conhecidos, combinando as assinaturas armazenadas. As assinaturas são geradas a partir do algoritmo *Position Aware Distribution Signature* (PADS). A assinatura é mantida no repositório utilizando o *Generalized Suffix Tree* (GST), que pode fazer corresponder as assinaturas num tempo consideravelmente ótimo. No IDS baseado na anomalia, é utilizado uma *deep learning* somado ao Q-learning (método de aprendizado por reforço) que considera SNR e parâmetros de largura de banda, onde classifica os ataques como DoS, Probe, User-to-Root (U2R) ou Remote-to-Local (R2L). Com o Q-learning, o ambiente é aprendido pelos agentes e gera uma matriz Q-table que tem os estados (Características) e as ações (Enviar/Não Enviar Alerta). Contudo, o Q-learning é apenas adequado para ambientes de pequena escala, e o IoT é um ambiente de grande escala. Assim, o Q-learning é combinado com a aprendizagem profunda para se tornar um algoritmo que pode processar vários pacotes de ataque desconhecidos simultaneamente [16].

O dataset utilizado para teste e treino foi o NSL-KDD, sendo assim, não foram executados teste reais, apenas o do próprio dataset. Como forma de comparação, foram utilizadas as métricas de *Recall*, Taxa de alarmes falsos, sensitividade, especificidade e F1-score. Os resultados mostram que a média do *Recall* e da sensitividade alcançadas foram de 96.9%. A especificidade e o F1-score apresentaram valores relevantes, além de proporcionar resultados relativamente baixos de alarmes falsos.

2.13.1.4 *A novel Ensemble of Hybrid Intrusion Detection System for Detecting Internet of Things Attacks*

No trabalho de Khraisat et al. [17] é apresentado um modelo de HIDS (do inglês Hybrid Intrusion Detection System) que combina métodos de aprendizado de máquinas para a detecção de ataques em sistemas IoT. As principais contribuições providas neste trabalho são apresentadas a seguir.

- Desenvolvimento de uma técnica de seleção de *features* baseada no princípio do ganho de informação para selecionar as *features* IoT que resultam na máxima diferença entre todas as aplicações apresentadas [17].

- Desenvolvimento de Sistema Híbrido de Detecção de Intrusão (HIDS) para dispositivos IoT e gateways que utilizam um classificador C5 na primeira fase e uma SVM (*Support Vector Machine*) na

segunda fase para criar uma arquitetura eficaz de maior precisão [17].

A Figura 2.16 mostra a arquitetura proposta no trabalho, onde possui dois estágios relevantes. O primeiro deles é o SIDS, que representa a detecção com base em assinatura, já o segundo AIDS é o estágio referente a detecção com base em anomalia.

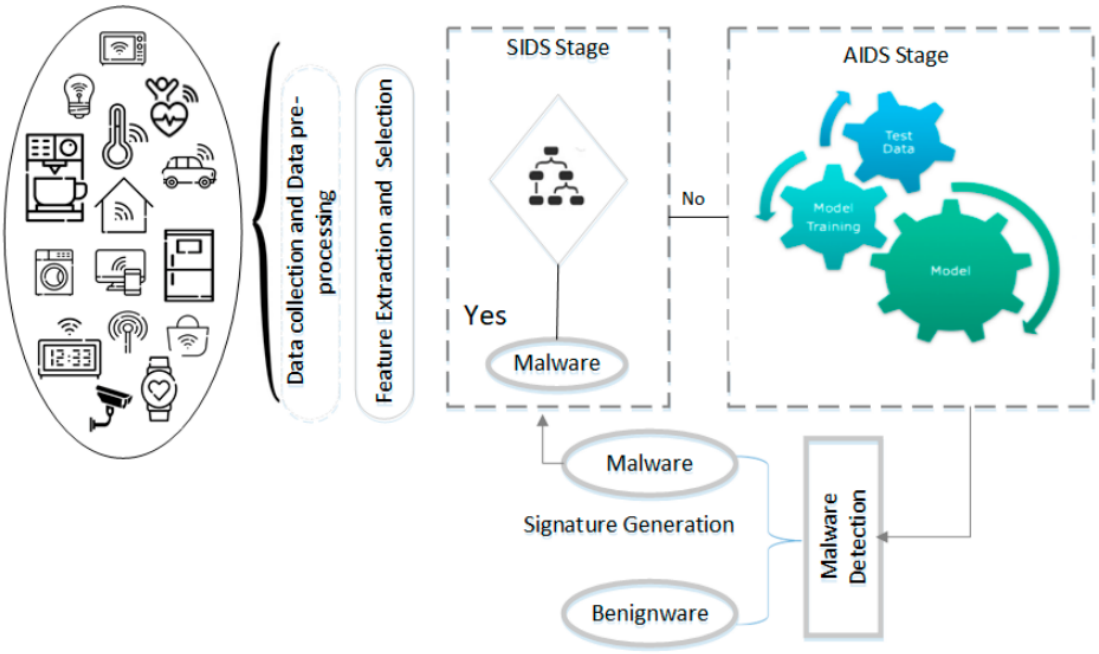

Figura 2.16: Arquitetura proposta [17].

Os autores subdividem sua arquitetura criada em possíveis momentos de funcionamento destintos, são eles:

- ***Feature Selection*:** Uma questão principal entre muitas outras na utilização de IDSs é lidar com muitas características irrelevantes, que podem causar sobrecarga no sistema. É transparente que características redundantes e irrelevantes conduzem frequentemente a uma baixa taxa de detecção. Portanto, o objectivo da seleção de *features* é identificar características significativas que podem ser utilizadas no IDS para detectar vários ataques de forma eficiente. Com as informações essenciais extraídas, as *features* são analisadas tanto para comportamentos normais como anormais para determinar quias são as mais relevantes. Foi aplicado um método de ganho de informação para a seleção de *features*. Os métodos de ganho de informação tiveram um tempo de execução rápido e esta técnica extraiu o conjunto de *features* de melhor desempenho para o tipo particular de modelo [17].

- ***Stage One: SIDS Stage*:** Na fase SIDS, o classificador de árvore de decisão C5 foi utilizado para criar uma árvore de decisão. Uma vez criada uma árvore de decisão, esta pode ser aplicada para detectar outras amostras com sucesso variável, dependendo de quão bem modela o conjunto de dados. A árvore pode então ser aplicada como um conjunto de regras para detectar se uma amostra de teste é um malware ou software benigno. O tráfego desconhecido foi tratado através da correspondência de padrões para determinar se representa atividades normais ou anormais. Se o pedido coincidir com

uma assinatura de ataque da base de dados, dá o alarme de que se trata de uma amostra maliciosa. Se não corresponder, irá para a AIDS [17].

- ***Stage Two: AIDS Stage*:** A fim de reconhecer eficazmente ataques desconhecidos, a produção da fase de SIDS é utilizada para treinar a AIDS para reconhecer atividades anormais. A AIDS, sendo treinada utilizando amostras benignas, deve ser capaz de separar atividades que não parecem ser normais, ou seja, comportamentos anormais exibidos por software do tipo malware. Para treinar a AIDS, foi utilizado o método SVM, que aprende os atributos das amostras benignas sem utilizar qualquer informação da outra classe. Este classificador pode identificar atividades normais com muito mais sucesso, uma vez que os dados de formação de classe normal estão facilmente disponíveis. Portanto, na segunda fase, o comportamento normal é identificado, e qualquer coisa fora do normal é classificado como um ataque de dia zero [17].
- ***Stage Three: Stacking of the Two Stages SIDS*:** O SIDS e o AIDS têm qualidades e deficiências correlativas, por isso, foi proposto a construção de um método híbrido que utiliza um conjunto de ambas as abordagens. No campo de aprendizado de máquinas, as técnicas de *ensemble* (junção de dois métodos de aprendizado de máquina) são utilizadas para melhorar a precisão da previsão [17].

O dataset utilizado pelos autores foi o Bot-IoT, no qual contem os ataques de DDoS, DoS, OS e Service Scan, Keylogging, e Data exfiltration. Todos os testes foram utilizando o próprio dataset, ou seja, não houve teste reais. Os resultados das principais métricas para cada classe de ataque, com a utilização do HIDS, é mostrada na Tabela 2.8.

Tabela 2.8: Resultados das principais métricas utilizando *ensemble* [17].

Class	TP Rate	FP Rate	F-Measure
Normal	0.88	0	0.936
DDoS	0.99	0	0.995
DoS	0.999	0.003	0.997
Reconnaissance	0.993	0.06	0.353
Keylogging	1	0	1

2.13.1.5 *A Comprehensive Analyses of Intrusion Detection System for IoT Environment*

No trabalho de Sicato et al. [18] é apresentado um IDS definido por software, baseado em uma arquitetura de nuvens distribuídas, de modo a prover segurança a dispositivos IoT. As principais contribuições providas neste trabalho são apresentadas a seguir.

- Primeiro, foi esboçado aspectos relevantes das questões de segurança, vulnerabilidades e superfícies em ambientes IoT [18].

- Em segundo lugar, uma discussão abrangente sobre questões em aberto no ambiente IDS em sistemas IoT [18].
- Em último lugar, avaliar a arquitetura proposta e mostrar que é melhor do que os métodos tradicionais [18].

A arquitetura usada no trabalho é apresentada na Figura 2.17. Ela foi subdividida nas camadas IoT já conhecidas, mas dentro da camada fog foi incluída um IDS com aprendizado de máquina para a detecção de ataques.

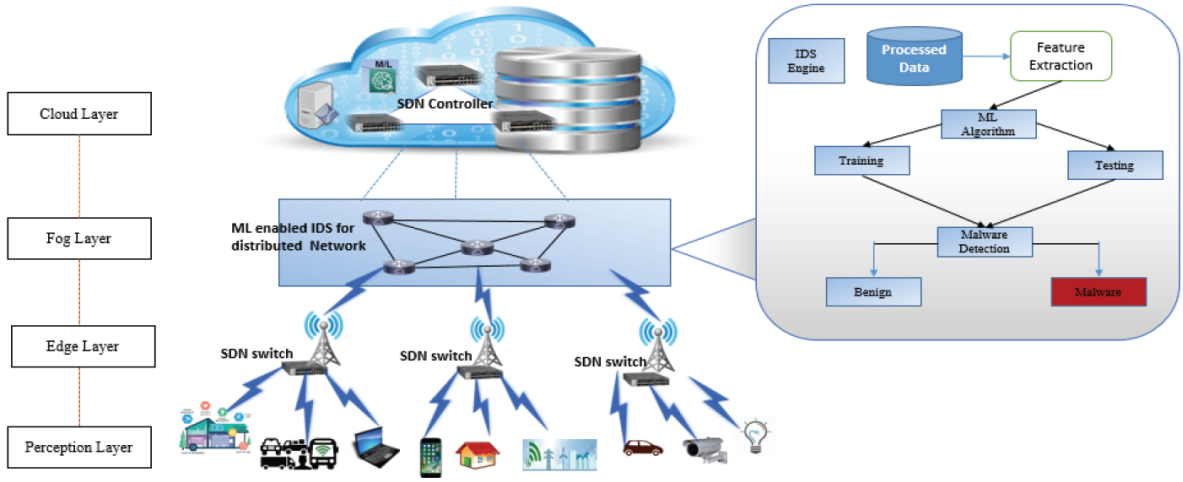

Figura 2.17: Arquitetura do trabalho proposto [18].

Para testes, foram executados experimentos com o dataset NLS KDD utilizando o Ubuntu 18.10, com 6 GB de RAM e 100 GB de espaço no disco rígido no VMware. Para treinar e testar o modelo de aprendizagem de máquinas, foi utilizado o Weka (3.9.3) e TensorFlow, e para SDN, SDN emulador, e MaxiNet [18]. Os resultados apresentados pelo o Weka e pelo TensorFlow são mostrados na Tabela 2.9.

Tabela 2.9: Resultados do trabalho de Sicato et al. [18].

TP rate	FT rate	Precision	Recall	MCC	RDC area	PRC area	Class
0.974	0.012	1.883	0.926	0.920	0.997	0.967	U2R
0.600	0.001	0.999	0.600	0.614	0.999	0.999	R2L
0.095	0.007	1.147	0.095	0.109	0.966	0.261	Probe
0.976	0.000	1.000	0.976	0.840	0.966	0.995	DoS
0.897	0.005	0.792	0.897	0.965	0.994	0.873	Normal
0.953	0.000	0.976	0.953	0.012	0.985	0.762	Unknown

2.13.1.6 OneM2M-IDPS

O trabalho apresentado por Chaabouni et al. [19] apresenta uma proposta de IDS/IPS para a camada de serviço em dispositivos IoT para mensagens oneM2M. Para isso, é utilizado aprendizado de máquinas em ambientes de fronteira (*edge*). O OneM2M-IDPS apresenta os seguintes objetivos:

- A detecção de ameaças IoT quer sejam ataques de segurança ou comportamentos anormais que possam levar ao bug do sistema IoT ou ao seu desligamento. Esta detecção inteligente deve não só identificar ameaças já vistas, mas também ser capaz de aprender ameaças desconhecidas a toda a momento [19].
- A prevenção contra as ameaças detectadas através de ações imediatas e apropriadas, gerando uma prevenção ativa e em tempo real para assegurar a disponibilidade dos serviços. [19].

Para isso, o OneM2M-IDPS possui estratégias bem definida apresentadas na Figura 2.18, onde são subdivididas em bloco. A seguir apresentaremos cada um deles.

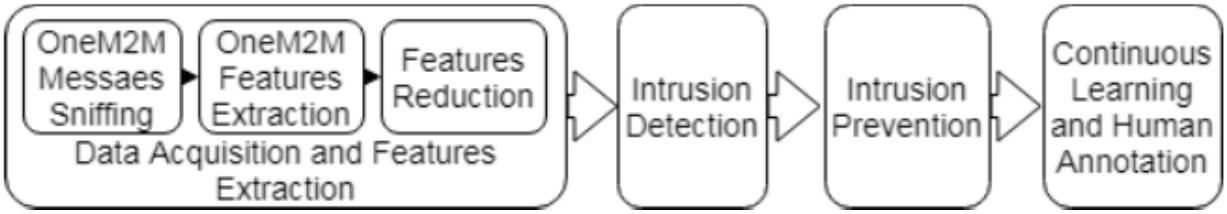

Figura 2.18: Estratégias do OneM2M-IDPS [19].

- ***Data Acquisition and Features Extraction*:** Este módulo capta as mensagens oneM2M a serem analisadas, processadas e enriquecidas. Fareja o fluxo oneM2M composto por um par de mensagens *request and response* como mencionado nas especificações oneM2M. Cada fluxo é composto por parâmetros obrigatórios e opcionais, dependendo da operação solicitada. Depois destes fluxos serem capturados, foi extraído e gerado as características necessárias para a detecção da aprendizagem da máquina. Estas características constituem o GFlow, uma nova abstração para as mensagens oneM2M, com base nos parâmetros obrigatórios. Um GFlow abrange múltiplos fluxos oneM2M com base numa chave composta pelo originador do pedido, o receptor do pedido, o tipo da operação solicitada e o estado da resposta. Para cada fluxo trocado, é gerado um conjunto de GFlows. [19].
- ***Intrusion Detection*:** Este é o módulo IDS para detectar as ameaças. Para garantir uma detecção precisa e leve, é utilizado três níveis de detecção de aprendizagem de máquinas. Em primeiro lugar, o GFlow é analisado numa classificação binária, identificando se os dados de entradas são benignos ou uma ameaça. Se for um GFlow benigno, não é tomada qualquer medida. No entanto, se for uma ameaça ao sistema IoT, o GFlow precisa de ser analisado mais profundamente para identificar o tipo exato da ameaça. Este passo está no segundo nível de detecção, que se baseia numa aprendizagem semi-supervisionada. Esta entrada pode ser classificada como uma das principais classes de ataques de oneM2M apresentadas [19].
- ***Intrusion Prevention*:** O IPS é utilizado para evitar que as ameaças danifiquem o sistema. Para se ter a melhor prevenção contra uma ameaça identificada, é necessária uma assinatura do ataque. Para as ameaças já conhecidas, tais como inundações (*flooding*) e amplificação, as ações de prevenção já estão definidas e associadas aos ataques na implementação do IDPS. Se o GFlow ainda não existir na base de dados de ameaças, então o conjunto de dados será atualizado com este fluxo, será lançado um alerta para captar a atenção do operador humano e será necessário realizar um procedimento de anotação do ataque, bem como a definição da ação de prevenção associada [19].

- ***Continuous Learning and Human Annotation*:** Este módulo produz uma aprendizagem contínua após a anotação humana, onde os modelos de aprendizagem de máquinas do primeiro e segundo níveis são atualizados para levar em consideração as novas ameaças já vistas. Para a requalificação dos modelos ML, é necessário dispor de dados suficientes [19].

Para treino e teste, foi utilizado o oneM2M dataset em dois cenários, com classificação binária de ataque e não ataque e na classificação de multi-classes, identificando, assim, qual seria o ataque sofrido. Como método de aprendizado de máquina foi testado o J48 (presente no Weka) e uma *deep learning* utilizando keras. Os resultados obtidos estão apresentados nas Tabelas 2.10 e 2.11.

Tabela 2.10: Resultados das principais métricas no cenário binário [19].

ML algorithm	Recall (%)	Accuracy (%)	Precision (%)	False Positive Rate (%)	Model Size (Ko)	CPU Training Time (ms)
J48	95.40	**87.81**	**88.90**	34.00	301	**27 490**
DL	**97.41**	86.91	86.61	**13.39**	**18**	596 463

Tabela 2.11: Resultados das principais métricas no cenário multi-classes [19].

ML algorithm	Recall (%)	Accuracy (%)	Precision (%)	False Positive Rate (%)	Model Size (Ko)	CPU Training Time (ms)
J48	**77.07**	**74.98**	**74.80**	**2.31**	3 677	**60 850**
DL	70.18	69.16	73.97	3.16	**33.50**	339 544

2.13.2 IDS para Ataque DoS/DDoS

Nesta subseção será apresentados os principais trabalhos correlatos que desenvolveram IDS para a detecção específica de ataques DoS/DDoS. Cada tópico seguinte representa um trabalho correlato.

2.13.2.1 *A Distributed Denial of Service Attack Detection System using Long ShortTerm Memory with Singular Value Decomposition*

No trabalho de Ugwu et al. [51], foi apresentado um sistema detecção de intrusão focado em reconhecimento de ataques DDoS. Para isso, foi proposta uma solução utilizando os métodos de Long Short Term Memory (LSTM) e Singular Value Decomposition (SVD) (esta última para extração de features). Para o treinamento do modelo, foi utilizado dois datasets o UNSW-NB15 e o NSL-KDD. Como resultado, o sistema criado foi testado nos arquivos de teste de ambos os dataset, no qual geraram uma acurácia de 90,59%, um recall de 0,82 e um F1-Score de 0,88 no NSL-KDD, já no UNSW-NB15 o sistema obteve melhores resultados, gerando uma acurácia de 94,28%, um recall de 0,80 e um F1-Score de 0,83.

2.13.2.2 *Minimizing false positive rate for dos attack detection: A hybrid sdn-based approach*

Latah e Toke [52] propõe uma abordagem de detecção de intrusão híbrida em redes SDN para a detecção de ataques DoS. O trabalho consiste em duas fases, em que a primeira fase é a detecção de intrusão baseada em fluxo na abordagem KNN, e a segunda fase é a detecção de intrusão baseada em pacotes com base em redes neurais. Ambas as fases utilizaram o conjunto de dados NSL-KDD para treinar e testar os modelos, no qual apresentou uma acurácia de 91%, uma precisão de 99%, além de um recall de 0.74 e um F1-score de 0.85.

2.13.3 IDS baseado em Classificação Multiclasses utilizando o NSL-KDD

Nesta subseção será apresentados os principais trabalhos correlatos que desenvolveram IDS para a detecção de ataques diversos utilizando o NSL-KDD. Cada tópico seguinte representa um trabalho correlato.

2.13.3.1 *A Deep Learning Approach for Intrusion Detection Using Recurrent Neural Networks*

Yin et al. [53] em seu trabalho, apresenta um sistema de detecção de intrusão utilizando técnicas de deep learning. Dessa forma, foi desenvolvido uma solução baseada em Recurrent Neural Networks (RNN-IDS). Com isso, segundo os autores suas contribuições gerais são: apresentar um modelo e uma implementação de RNN para IDS, executando testes em cenários de classificação binária e multiclasses; Executar testes com os métodos de aprendizado de máquina comuns; Comparar o desempenho nos dois cenários com os métodos tradicionais de aprendizado de máquina. Para isso, foi utilizado o dataset NSL-KDD para treino (arquivo KDD-Train+), além do arquivo KDD-Test+ para os testes de desempenho do sistema. No treinamento o sistema obteve 99,81% de acurácia utilizando cross-validation no arquivo de treino. No arquivo de teste (KDD-Test+) o sistema obteve 83,28% de acurácia no cenário de classificação binária e 81,29% de acurácia no cenário de classificação multiclasses.

2.13.3.2 *Deep Learning Approach for Intelligent Intrusion Detection System*

Vinayakumar et al. [54] apresenta um sistema de detecção de intrusão baseado em deep learning, demonstrando os melhores valores de parâmetros para este método. Dessa forma, foram testados diversos parâmetros, de modo a obter os melhores valores de métricas para este tipo de sistema, quando utilizado deep neural network (DNN). Para testes, foram utilizados os seis datasets mais prestigiados pela literatura, são eles: KDDCup 99, NSL-KDD, UNSW-NB15, WSN-DS, CICIDS 2017 e Kyoto. Todos estes testes foram executados em dois cenários distintos, classificação binária e classificação multiclasses. Todos os resultados foram comparados com os métodos de aprendizado de máquina tradicionais, sendo que na maioria deles a solução proposta apresentou uma ligeira vantagem.

2.13.4 Conclusões sobre os Trabalhos Relacionados

Os trabalhos apresentados variam em pontos significativos, tais como: metodologia de desenvolvimento, arquitetura, técnica de detecção de ataque, datasets, ferramentas utilizadas, equipamentos, entre outros. A seguir será apresentada a comparação dos trabalhos descritos na seção com o trabalho proposto nessa dissertação, evidenciando os pontos de contribuição em cada um deles.

Em Jia et al. [14], diferentemente do trabalho aqui proposto, não apresenta uma ferramenta de gerenciamento de ativos para gerar alarmes providos da detecção de ataques, o que pode inviabilizar seu uso no dia a dia de um instituição, dado a importância dessa ferramenta atualmente. Outra desvantagem apresentada no trabalho de Jia et al. [14], em comparação com o trabalho proposto, é a utilização de métodos de aprendizado de máquinas sofisticados, tais como: CNN e LSTM, onde torna-se necessários dispositivos com mais capacidade de recursos para um funcionamento ideal do sistema.

Sequencialmente, em dissonância com o trabalho proposto, o projeto apresentado por Eskandari et al. [15] não utiliza dos datasets de referência da literatura, sendo utilizado de um dataset próprio, o que inviabiliza a comparação dos resultados das principais métricas com quaisquer trabalhos correlacionados. Outros pontos a se destacar, é a falta de um sistema de geração automática de alarmes e a não utilização de MLOps para o gerenciamento do ciclo de vida do método treinado, pontos estes, que são abordados pelo trabalho proposto.

Na proposta apresentada por Otoum e Nayak [16], diferentemente do trabalho proposto, não houveram testes do trabalho desenvolvido em tempo real, apresentando assim, os resultados das métricas utilizando o dataset de treino do NSL-KDD. Além disso, não foi muito bem apresentados os resultados obtidos, sendo utilizados algumas métricas incomuns para a literatura. Não houve também a utilização de MLOps para o gerenciamento do ciclo de vida do método treinado, algo executado pelo trabalho proposto.

Como diferença ao trabalho proposto, o projeto apresentado por Khraisat et al. [17] não utiliza testes em tempo real, apresentando apenas testes com o próprio dataset, algo que pode diminuir a eficácia do sistema, dado a imprevisibilidade dos ataques executados em tempo real. Não há também nenhuma plataforma de geração de alarmes automática e nem ferramentas para o gerenciamento do ciclo de vida do método treinado, diferentemente do trabalho proposto.

No trabalho apresentado por Sicato et al. [18], diferentemente do trabalho proposto, não utiliza testes em tempo real (apenas testes utilizando o dataset de treino do NSL-KDD), não possui ferramentas para a geração automática de alarmes de ataque, não há um gerenciamento do ciclo de vida do método treinado, além de apresentar métodos de aprendizado de máquinas sofisticados, gerando, assim, a necessidade de dispositivos com capacidade computacional razoáveis.

Em diferença com o trabalho proposto, o projeto apresentado por Chaabouni et al. [19] não apresenta ferramentas para o gerenciamento dos alarmes criados, o gerenciamento do ciclo de vida do método treinado é altamente dependendo de um ser humano, além de não terem sido realizados testes em tempo real com o sistema proposto, utilizando apenas dados dos dataset escolhido.

As propostas apresentadas por Ugwu et al. [51], Latah e Toke [52], Yin et al. [53] e Vinayakumar et al. [54], se diferenciam do trabalho proposto, pois não são focados na detecção em dispositivos IoT, além de não serem direcionados para a detecção de ataques DoS em tempo real. Apesar de todos utilizarem tecnicas

de aprendizado de máquinas, nem um deles apresentam quaisquer plataformas de geração de alarmes automática e nem ferramentas para o gerenciamento do ciclo de vida do método treinado, diferentemente do trabalho proposto.

Os trabalhos apresentados aqui não são capazes de suprir os seguintes pontos:

- Uma solução completa (*framework*) de um sistema de detecção de intrusão, desde a captura dos dados ao gerenciamento dos alarmes criados.
- A utilização de métodos de aprendizado de máquinas convencionais, utilizando a ferramenta Optuna para a melhor definição dos hiperparâmetros associados.
- A utilização de ferramentas MLOps (MLFlow) para o gerenciamento do ciclo de vida do método treinado.
- Criação de um modelo de log próprio, para o geração automática de alarmes no Wazuh.
- O gerenciamento dos alarmes criados pelo ataques na plataforma ELK com o plugin Wazuh, no qual possibilita a geração de valor dos dados através de gráficos, mapas e *dashboards*.
- Execução de teste com o dataset de treino do NSL-KDD, além de teste com ataques efetuado em tempo real, validando, assim, os dois cenários possíveis.

3 METODOLOGIA

Este capítulo apresentará a metodologia do *framework* desenvolvido, relatando e detalhando a solução proposta para a detecção de ataques DoS em sistemas IoT, utilizando métodos de aprendizado de máquinas para a classificação dos ataques.

3.1 APRESENTAÇÃO DO *FRAMEWORK* PROPOSTO

A arquitetura do *framework* é apresentada na Figura 3.1, na qual é subdividida em quatro módulos centrais: captura de dados, filtragem dos dados, classificação dos dados e visualização dos dados. O primeiro deles é responsável pela a captura de todo o tráfego recebido e enviado pelos dispositivos IoT. O segundo salva os dados capturados no módulo anterior e filtra estes dados, de modo a se tornarem fluxos passíveis de classificação. O terceiro módulo é responsável por classificar os fluxos capturados, através do método de aprendizado de máquina treinado, sendo gerenciado pelo MLFlow. Por fim, temos o último módulo responsável pela a criação automática e gerenciamento dos alarmes. Cada um destes módulo apresentados será explanado com mais detalhes nas seções seguintes.

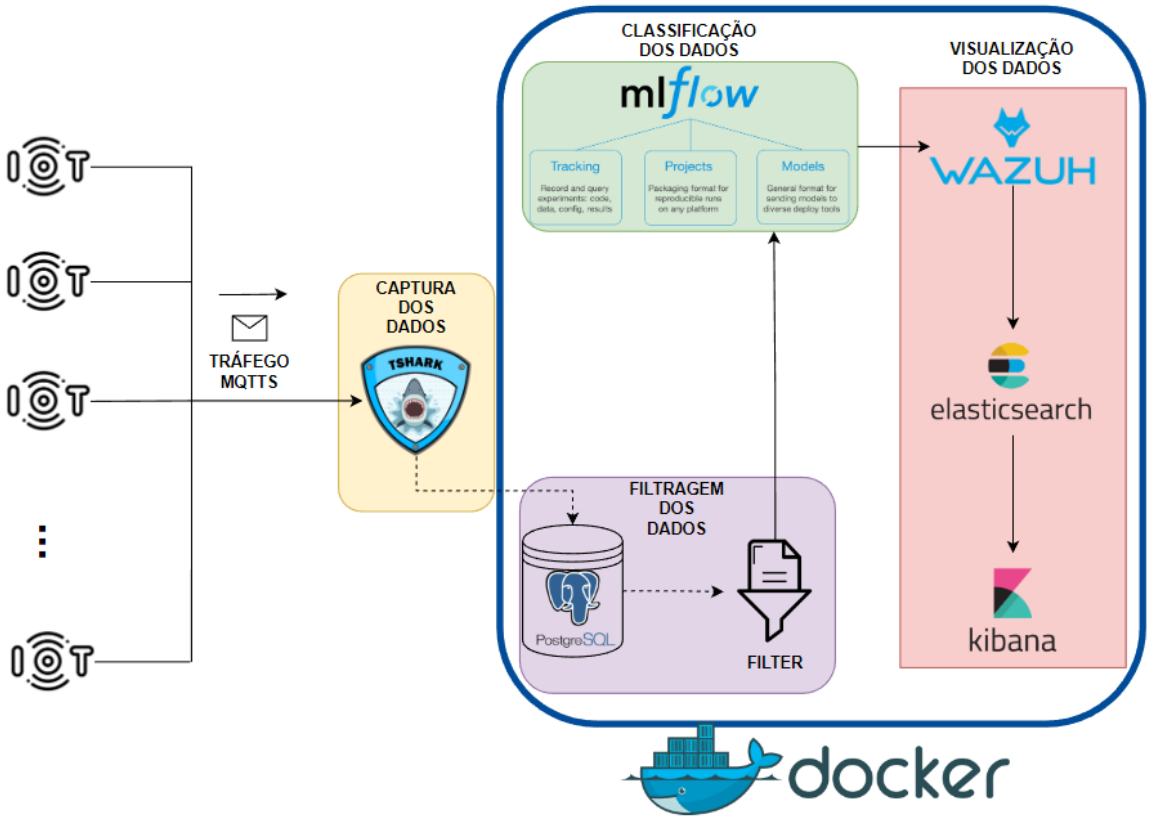

Figura 3.1: Arquitetura do *framework* desenvolvido.

3.2 CAPTURA DOS DADOS

A captura de dados foi implementada através do tshark, onde, utilizando a função *sniff_continuously()*, os pacotes trafegados na redes são capturados e analisados, coletando, assim, as informações necessárias (vale ressaltar a utilização de filtros para selecionar os protocolos desejados). O diagrama da captura dos dados é apresentado na Figura 3.2, no qual podemos perceber que, o sistema foi implementado para capturar apenas pacotes TCP, UDP ou ICMP. Isso se dá pelo fato do dataset NSL-KDD, apresentar apenas esses protocolos em seu arquivo de treino. Desta forma, cada pacote analisado depende do protocolo envolvido, pois existem informações especificas, tais como: Somente os pacotes TCP possuem as flags TCP e o protocolo ICMP não possui porta de destino e nem de origem. Os dados capturados em cada pacote são armazenados em um banco de dados relacional Postgres, para serem tratados, classificados e visualizados, através dos módulos seguintes do *framework*. Os principais dados capturados de cada pacote são: o tempo da coleta do pacote, o endereço e a porta de destino e origem, o *checksum*, as flags TCP (no caso do pacote ser TCP) e os *bytes* do pacote (alcançando, assim, o objetivo especifico número 1, apresentado no capítulo 1). O pseudocódigo referente ao módulo de captura dos dados é apresentado em Algoritmo 1.

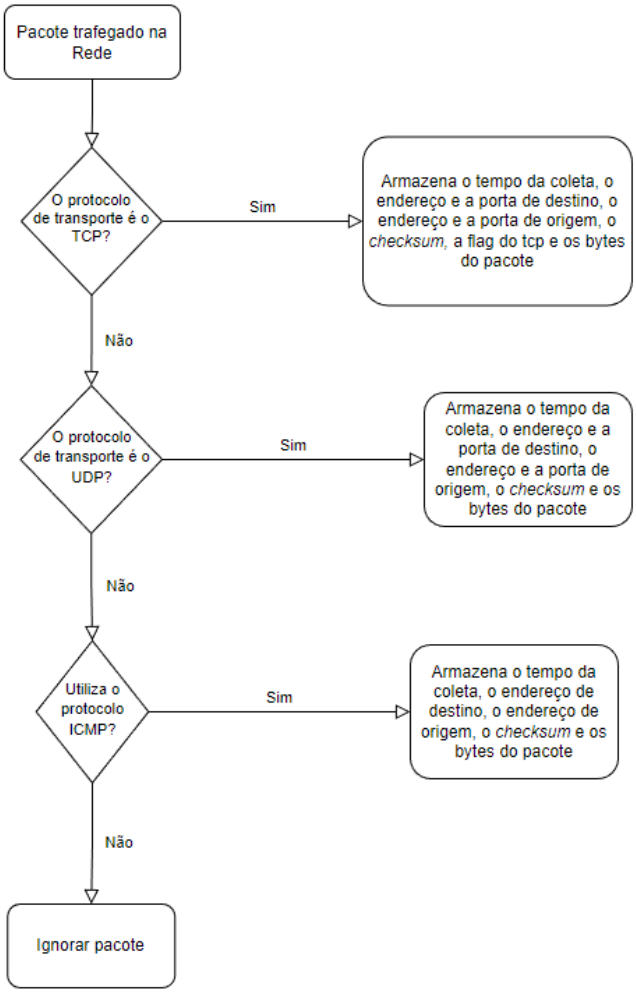

Figura 3.2: Diagrama da captura dos dados.

Algoritmo 1: Pseudocódigo referente a captura dos dados

for $packets$ **to** *sniff_continuously()* **do**
    $localtime \leftarrow packet.sniff\_timestamp$;
    $protocol \leftarrow packet.transport\_layer$;
    $byte \leftarrow packet.ip.len$;
    $src\_addr \leftarrow packet.ip.src$;
    $ip\_checksum \leftarrow packet.ip.checksum$;
    $src\_port \leftarrow packet[protocol].srcport$;
    $dst\_addr \leftarrow packet.ip.dst$;
    $dst\_port \leftarrow packet[protocol].dstport$;
    if $protocol = TCP$ **then**
        Salva no banco de dados com o *template* de protocolo TCP;
    else if $protocol = UDP$ **then**
        Salva no banco de dados com o *template* de protocolo UDP;
    else if $protocol = ICMP$ **then**
        Salva no banco de dados com o *template* de protocolo ICMP;
    else
        Ignora o pacote;
    end
end

3.3 FILTRAGEM DOS DADOS

Para o treinamento de classificação do método de aprendizado de máquina, foi necessária a utilização de um dataset de treino especifico para a detecção de anomalias. O dataset escolhido foi o NSL-KDD que, como apresentado no Capítulo 2, se tornou uma referência na literatura pela sua versatilidade, no qual diversos trabalhos utilizam-se dele para comparação de resultados. O NSL-KDD possui, ao todo, 41 *features* para a predição de quatro classes distintas de ataques. Como o foco da proposta apresentada é o sistema IoT, jugou-se que dentro das classes de detecção de ataques presentes no NSL-KDD, a classe de ataques DoS seria a mais problemática para este tipo de sistema, dado os constantes ataques desta classe ocorridos recentemente, onde, por exemplo, em 2016 o vírus Mirai infectou 65000 dispositivos nas primeiras 20 horas, com o intuito de criar uma botnet, de modo a fazer ataques DDoS a sistemas IoT [55]. Dado isso, nesta proposta foi desconsiderado qualquer outro ataque que seja distinto da classe de ataques DoS, sendo necessário a conversão do dataset para detecção apenas deste tipo de ataques. Para isso, foi necessário mudar os rótulos do dataset, onde para todo ataque não pertencente a classe DoS, foi alterado seu rótulo para conexão normal. Isso foi possível, pois os ataques de DoS somam cerca de 36,46% do dados dos dataset (alcançando, assim, o objetivo especifico número 2, apresentado no capítulo 1).

Pensando em reduzir a quantidade de *features* (obtendo a mínima perda de eficácia), foi aplicada a técnica de correlação de variáveis, no qual utiliza-se de uma ferramenta matemática para a identificar quais são as *features* mais importantes para o sistema. O cálculo da correlação de variáveis retorna um valor entre -1 e 1, onde -1 é a correlação perfeita negativa (variáveis altamente opostas) e 1 e a correlação perfeita positiva (variáveis muito similares). As variáveis abaixo de -0.5 e superiores a 0.5 são consideradas altamente correlacionadas (abaixo de -0.5 alta correlação negativa e acima de 0.5 alta correlação positiva). Essa técnica é muito utilizada em um ramo do aprendizado de máquina, chamado *feature engineering*.

Para isso, foi criado o código, onde, através da função *corr()* do panda, é executado o cálculo de correlação entre todas as *features* do sistema com o rótulo de predição. Dessa forma, é retirado o sinal e capturada todas as *features* com correlação maior que 0.5. Vale a pena ressaltar que, como o NSL-KDD, em sua concepção, não foi criado para ser um dataset de detecção de anomalias em tempo real, foi necessário considerar algumas *features* além das apresentadas pelo método de correlação. Assim, utilizando esses processos, foi possível reduzir de 41 *features* para 17, onde cada um delas pode ser verificada na Tabela 3.1 (alcançando, assim, o objetivo especifico número 3, apresentado no capítulo 1).

Tabela 3.1: As 17 *features* selecionadas.

***Features* Selecionadas**	
src_bytes	dst_bytes
count	srv_count
serror_rate	srv_serror_rate
same_srv_rate	dst_host_count
dst_host_srv_count	dst_host_same_srv_rate
dst_host_serror_rate	dst_host_srv_serror_rate
S0	S1
S2	S3
SF	-

3.3.1 Adequação das conexões em tempo real

Para ser um *framework* efetivo na detecção próxima ao tempo real, foi necessário gerar conexões, pois as *features* são apresentadas em fluxo de pacotes e não em pacotes individuais. Para isso, foi ajustado o agrupamento de pacotes com os mesmos endereços e portas de destino e origem nos últimos dois segundos de captura, gerando assim, conexões estabelecidas. A Figura 3.3 mostra o diagrama que representa esse processo.

As conexões salvas necessitam serem filtradas, de modo a apresentarem as 17 *features* estabelecidas anteriormente, pois com isso, será possível ter uma predição adequada. Para isso, torna-se necessário entendermos como as *features* são capturadas no NSL-KDD. As principais *features* do NSL-KDD são subdivididas em análise de curto prazo e de longo prazo. A primeira delas, referencia as *features* que utilizam as conexões ocorridas nos últimos dois segundos. Já a análise de longo prazo, valida as *features* utilizando as cem últimas conexões estabelecidas. Desta forma, foi estabelecido dois fluxos de adequação de dados, um para cada tipo de análise. Na Figura 3.4 vemos à esquerda o diagrama relacionado a filtragem das *features* de curto prazo, já à direita, temos o diagrama para filtragem das *features* de longo prazo (O Algoritmo 2 mostra o pseudocódigo da filtragem executada). A seguir será apresentada cada uma das 17 *features*, indicando quais são de curto ou longo prazo.

- ***src_bytes*:** Quantidade de *bytes* enviada pelo remetente (não se enquadra como curto ou longo prazo).
- ***dst_bytes*:** Quantidade de *bytes* enviada pelo destinatário (não se enquadra como curto ou longo prazo).

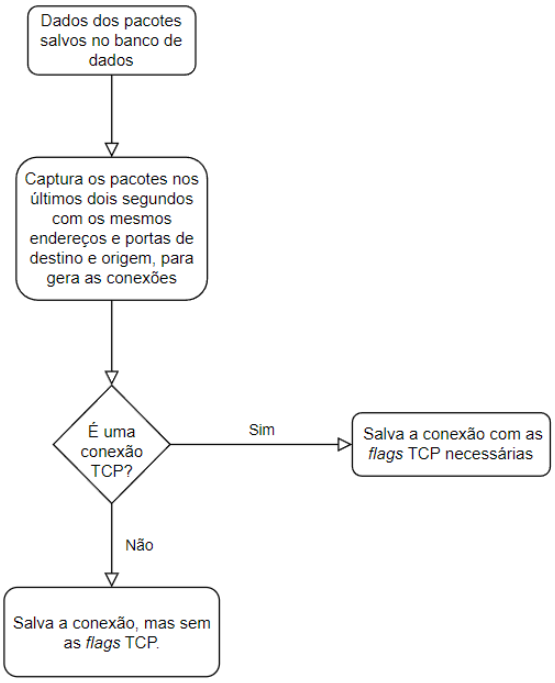

Figura 3.3: Diagrama da geração de conexões.

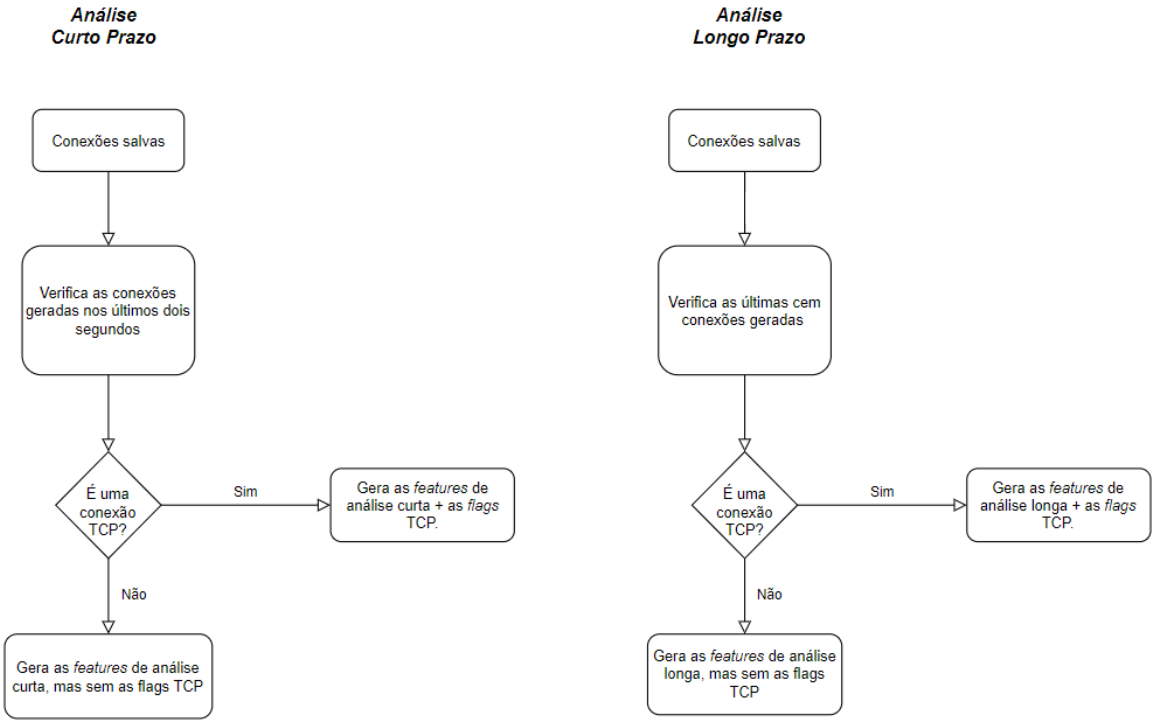

Figura 3.4: Diagrama da filtragem entre as *features* de curto e longo prazo.

Algoritmo 2: Pseudocódigo referente a filtragem dos dados

```
while True do
    Seleciona o próximo pacote com flag SYN no banco ou protocol = UDP ou
     protocol = ICMP;
    Espera 2 segundos;
    if nos últimos dois segundos existem pacotes com o mesmo endereço e porta de origem e
     destino then
        Cria conexão juntando todos os pacotes relacionados;
        Calculada todas as features de curto prazo e atribuí as flags;
        if já temos mais de 100 conexões then
            Calculada todas as features de longo prazo;
        else
            Todas as features de longo prazo são zeradas;
        end
    else
        Todas as features são zeradas, sendo atribuída a flag S0;
    end
    Salva todos os resultados das features calculadas;
end
```

- ***count*:** Número de conexões com o mesmo destinatário da conexão que está sendo analisada nos últimos dois segundos (*feature* de curto prazo).
- ***srv_count*:** Número de conexões com o mesmo serviço (mesma porta) da conexão que está sendo analisada nos últimos dois segundos (*feature* de curto prazo).
- ***serror_rate*:** Porcentagem das conexões que ativaram as *flags* S0, S1, S2 ou S3 entre as conexões agregadas em *count* (*feature* de curto prazo).
- ***srv_serror_rate*:** Porcentagem das conexões que ativaram as *flags* S0, S1, S2 ou S3 entre as conexões agregadas em *srv_count* (*feature* de curto prazo).
- ***same_srv_rate*:** Porcentagem das conexões que tem o mesmo serviço (mesma porta) entre as conexões agregadas em *count* (*feature* de curto prazo).
- ***dst_host_count*:** Número de conexões que possui o mesmo endereço de destino (*feature* de longo prazo).
- ***dst_host_srv_count*:** Número de conexões que possui a mesma porta (*feature* de longo prazo).
- ***dst_host_same_srv_rate*:** Porcentagem das conexões que tem o mesmo serviço (mesma porta) entre as conexões agregadas em *dst_host_count* (*feature* de longo prazo).
- ***dst_host_serror_rate*:** Porcentagem das conexões que ativaram as *flags* S0, S1, S2 ou S3 entre as conexões agregadas em *dst_host_count* (*feature* de longo prazo).
- ***dst_host_srv_serror_rate*:** Porcentagem das conexões que ativaram as *flags* S0, S1, S2 ou S3 entre as conexões agregadas em *dst_host_srv_coun* (*feature* de longo prazo).

- ***S0*:** Tentativa de execução do *three-way handshake*, mas sem resposta de ACK do remetente (não se enquadra como curto ou longo prazo).
- ***S1*:** O *three-way handshake* foi executado com sucesso, mas não houve finalização da conexão (não se enquadra como curto ou longo prazo).
- ***S2*:** O *three-way handshake* foi executado com sucesso e houve uma tentativa de finalização de conexão por parte do remetente sem resposta (não se enquadra como curto ou longo prazo).
- ***S3*:** O *three-way handshake* foi executado com sucesso e houve uma tentativa de finalização de conexão por parte do destinatário sem resposta (não se enquadra como curto ou longo prazo).
- ***SF*:** O *three-way handshake* foi executado com sucesso e finalização ocorrida sem falhas (não se enquadra como curto ou longo prazo).

3.4 CLASSIFICAÇÃO DOS DADOS

Conforme descrito na capítulo 2, o método escolhido para o reconhecimento de ataques foi a detecção por anomalias (dado a sua qualidade em detectar o comportamento da rede), utilizando aprendizado de máquina para tal situação. Dessa forma, tornou-se necessário a definição e o treinamento de um modelo, de modo a apresentar bons resultados em ataques DoS em sistemas IoT, bem como sua adaptação para reconhecimento de ataques em tempo real.

3.4.1 Modelo treinado

Como apresentado na literatura [8] [56], métodos de aprendizado de máquina que utilização árvores de decisão são os que obtiveram melhores resultados nas principais métricas no treino e teste com o NSL-KDD. Desta forma, foi escolhido o uso do método *Random Forest*, pois apresenta-se como uma das melhores opções quando utiliza-se árvore de decisão. Como apresentado no Capítulo 2, o método *Random Forest* combina diversas árvores de decisão simultâneas, gerando um floresta de decisão, assim, no final há uma votação pela maioria para decidir o resultado da predição.

Para o treinamento do modelo, foi utilizada a biblioteca XGBClassifier (eXtreme Gradient Boosting Classifier), onde é um algoritmo de *boosting* baseado em árvores de decisão, sendo uma classe compatível com a API scikit-learn para classificação, disponível para Python. O XGBoost aplica técnicas de regularização para reduzir o *overfitting*. Esta ferramenta é utilizada para treinar árvores de decisão com o auxilio do *gradient-boosted*. A diferença para do treinamento com XGBClassifier para uma biblioteca de treino de *Random Forest* convencional, segundo [57], é apenas no algoritmo de como será treinada. Portanto, é comum utilizar esse método para o treinamento de *Random Forest* em grande escala no Python, dado a sua velocidade e precisão (em [57] é apresentado como utilizar XGBClassifier para *Random Forests*).

Uma das partes mais importantes do treinamento é a definição dos hiperparâmetros dos métodos de aprendizado de máquina, pois, alterações em cada um deles, pode gerar diferenças significativas na qualidade do modelo, ou seja, com pequenas alterações é possível aumentar ou diminuir o erro médio do

método utilizado. Dada a sua grande importância, torna-se necessário gerar um estudo dos melhores valores para cada hiperparâmetro envolvido. Para isso, foi utilizado uma ferramenta chamada Optuna, na qual gerar estudos através de testes, de modo a indicar os melhores valores para cada hiperparâmetro. A Figura 3.5 mostra o funcionamento geral do Optuna, onde inicialmente é necessário definir a quantidade de estudos desejada e qual será a direção da escolha (redução ou maximização), o *range* dos hiperparâmetros a serem testados, além de apontar a função de teste disponibilizada pelo método utilizado, na qual será testada diversas vezes o resultado no conjunto de validação, variando os hiperparâmetros. Cada teste é chamado de *Worker* sendo salvo seu resultado, no final de todos os estudos será apontado os valores dos hiperparâmetros que melhor se adéqua na direção escolhida.

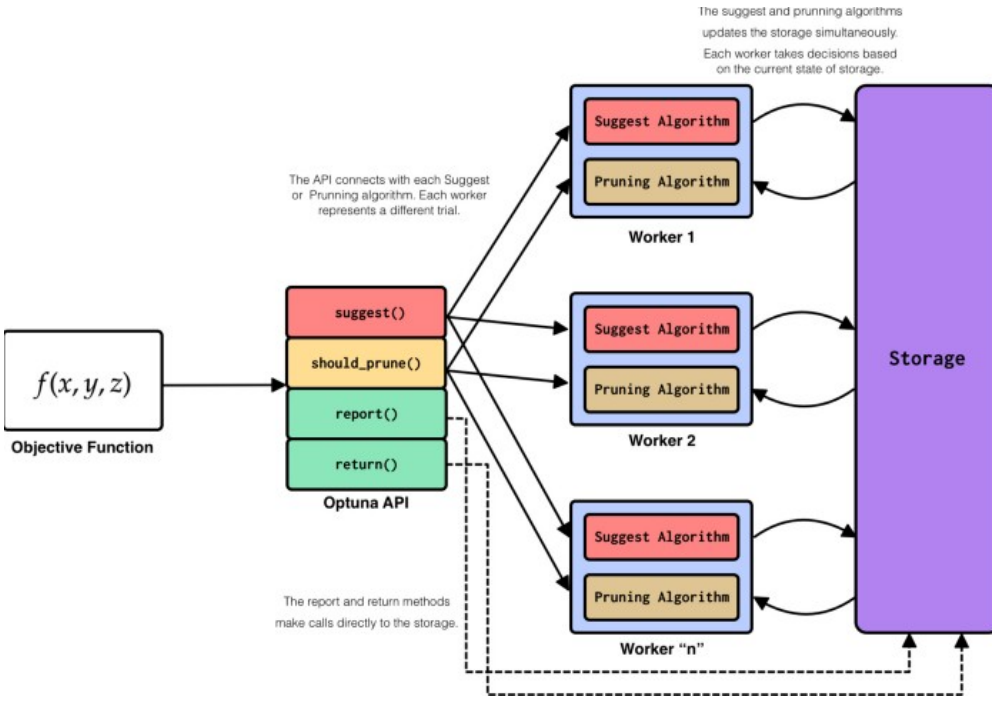

Figura 3.5: Arquitetura do Optuna [20]

Dado o que foi apresentado, foi utilizado a ferramenta Optuna na direção de minimizar o erro quadrático médio (MSE - do inglês, *Mean Square Error*), utilizando 50 estudos para a definição dos hiperparâmetros. O *range* selecionado dos hiperparâmetros é apresentado a seguir.

- ***learning_rate*:** Valor inicial: $1 * 10^{-}2$ -> Valor Final: 0.25.
- ***reg_lambda*:** Valor inicial: $1 * 10^{-}8$ -> Valor Final: 100.
- ***reg_alpha*:** Valor inicial: $1 * 10^{-}8$ -> Valor Final: 100.
- ***subsample*:** Valor inicial: 0.1 -> Valor Final: 1.
- ***colsample_bytree*:** Valor inicial: 0.1 -> Valor Final: 1.
- ***max_depth*:** Valor inicial: 1 -> Valor Final: 7.

Desta forma, foi apresentado pelo Optuna os melhores valores para cada hiperparâmetro envolvido no método de treinamento. Os valores obtidos são apresentados na Tabela 3.2.

Tabela 3.2: hiperparâmetros usados.

hiperparâmetros	Valor
n_estimators	7000
learning_rate	0.03135068104119495
reg_lambda	4.3530763634327134e-08
reg_alpha	1.922393232074839
subsample	0.6059698233437025
colsample_bytree	0.3052219054514836
max_depth	7

Com isso, foi treinado o modelo utilizado com o dataset de treino do NSL-KDD, onde 80% desses dados foi disposto para o treino e 20% para validação e testes, levando em consideração as alterações descritas anteriormente para a detecção exclusiva de ataques DoS. Os resultados gerados apontam que o melhor estudo (o com o menor erro quadrático médio) apresentou um MSE = 0.007938082952966859 (alcançando, assim, o objetivo especifico número 4, apresentado no capítulo 1).

3.4.2 Funcionamento próximo ao tempo real

Para a execução da classificação próxima ao tempo real, era necessário integrar o modelo treinado no fluxo do *framework* apresentado. Dessa forma, foi utilizado o MLOps, no qual define práticas para implementar e manter modelos de aprendizado de máquina em ambientes de produção. Com isso, foi utilizado o MLFlow (conforme apresentado no Capítulo 2), onde inicialmente foi criado um experimento para o treinamento do modelo descrito anteriormente, com os devidos testes. Após isso, o experimento foi transformado na primeira versão do modelo chamado *ddos_attack*, sendo colocado em fase de produção (a Figura 3.6 mostra o modelo criado no MLFlow). A partir desse momento, é possível, através de funções Python, acessar o endpoint do modelo criado, onde para cada amostra enviada é retornada sua predição de acordo com o modelo treinado. Isto só é possível, pois a cada modelo criado no MLFlow gera-se artefatos, no qual estes são suficientes para replicar o modelo treinado todas as vezes que há a necessidade de predições. Estes artefatos gerados precisam ser salvos em banco de dados orientados a objetos, semelhante ao AWS S3. Desta forma, foi utilizada a ferramenta MinIO para simular um banco S3, com o intuito de salvar os artefatos gerados pelo modelo.

Após obtermos o modelo treinado e o MLFlow configurado corretamente, foi necessário gerar estratégias para a detecção de ataques, através dos dados presentes até o momento no *framework*. Para isso, foi criada uma função para a classificação dos dados, onde recebe um *array* de 10 conexões filtradas e seu retorno é o envio dos logs de predição (serão apresentados na próxima seção) para a fila do RabbitMQ, apresentando, assim, quais conexões são possíveis ataques e quais são possíveis conexões normais. A Figura 3.7 mostra o diagrama da função de classificação elaborada, já o Algoritmo 3 apresenta o pseudocódigo desta função criada.

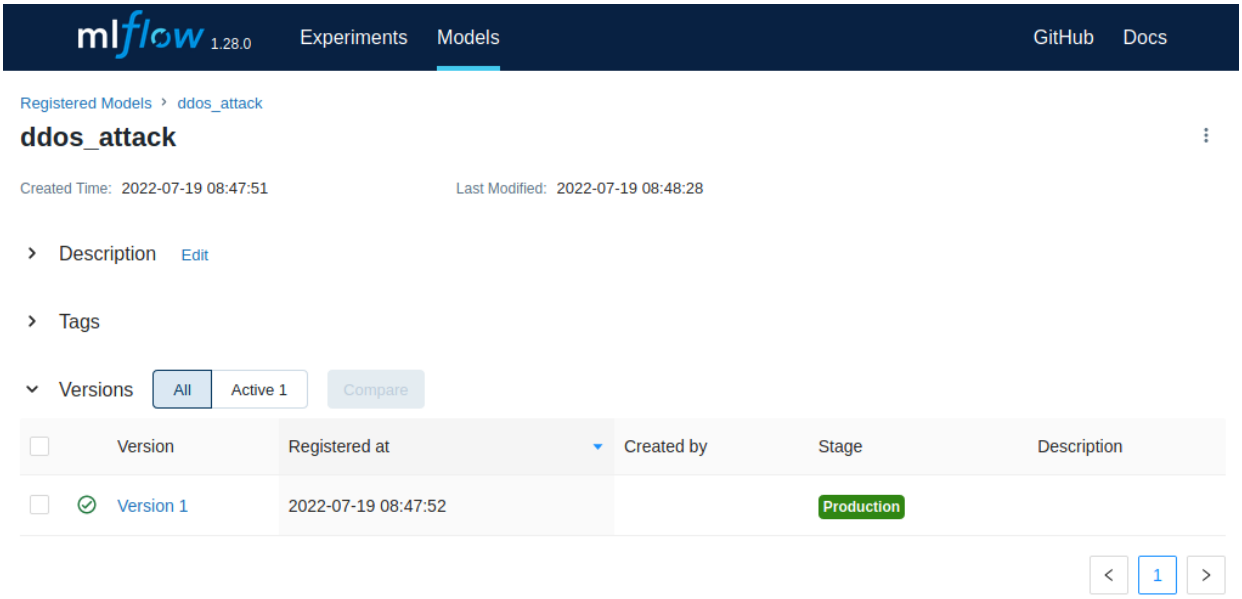

Figura 3.6: Modelo criado no MLFlow.

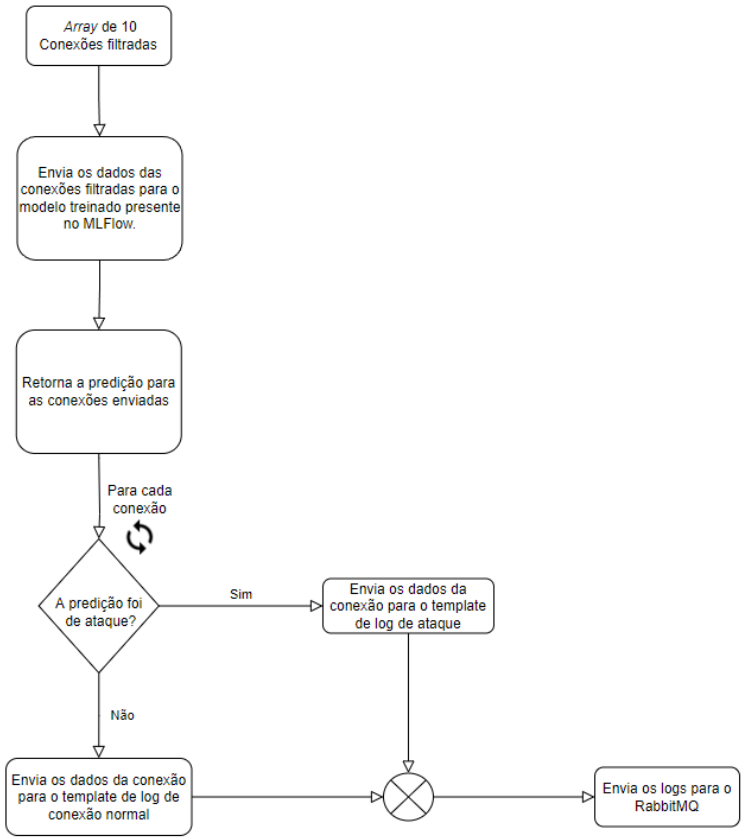

Figura 3.7: Diagrama da função de classificação dos dados.

Algoritmo 3: Pseudocódigo referente a classificação dos dados.

$connections \leftarrow 10\_ultimas\_conexoes\_filtradas$;

/* Início de leitura do modelo */;

$client \leftarrow mlflow.tracking.MlflowClient(tracking\_uri =' IP\_do\_servidor')$;
$uri \leftarrow (client.get\_model\_version\_download\_uri(name =' ddos\_attack', version =' 1')) +' /'$;
$loaded\_model \leftarrow mlflow.sklearn.load\_model(model\_uri = uri)$;

/*Fim da leitura do modelo */;

/*Início da predição e do envio dos dados */;

$predictions \leftarrow loaded\_model.predict(connections)$;

for $prediction$ ***in*** $predictions$ **do**
    if $prediction$ = *'Attack'* **then**
        Coloca os dados no template de ataques;
        Envia para o RabbitMQ;
    else
        Coloca os dados no template de conexão normal;
        Envia para o RabbitMQ;
    end
end

/*Fim da predição e do envio dos dados */;

3.5 VISUALIZAÇÃO DOS DADOS

No último módulo do *framework* desenvolvido, temos a visualização dos dados. Inicialmente, foi criado um modelo de log próprio, para o envio dos dados de classificação. Ele é composto pelas informações de data e hora, nome do dispositivo analisado, endereço ip de origem, protocolo, porta de destino e predição realizada. Dessa forma, o log criado tem esta estrutura: "***Mês Dia Hora:Minuto:Segundo Nome_do_dispositivo*** ml_detection ***_Predição***[***Id_dos_logs***]: Connections from ***Ip_de_origem*** using ***Protocolo_utilizado*** protocol in a port ***Porta_de_destino***, results in a ***Predição*** prediction" (todas as informações em itálico e negrito são as que variam para cada tipo de conexão). Um exemplo do log criado é apresentado na Figura 3.8.

Posteriormente, como apresentado na Figura 3.7, os dados classificados são enviados para o RabbitMQ,

```
Jul 04 14:28:04 osboxes ml_detection_normal[402]: Connection from 192.168.0.46 using UDP protocol in a port 53, results in a Normal prediction
Jul 04 14:28:24 osboxes ml_detection_attack[404]: Connection from 192.168.0.46 using UDP protocol in a port 53, results in a Attack prediction
```

Figura 3.8: Exemplo do modelo de Log criado.

no qual utiliza-se do protocolo AMQP para criar e gerenciar uma fila entre um *publisher* e um *consumer*. No *framework* o *publisher* é o módulo de classificação dos dados, onde envia os logs das predições feitas. O *consumer* é o Logstash (ferramenta de *pipeline* que recebe os dados em sua entrada, filtra-os e direciona os dados para alguma saída específica), sendo que o mesmo recebe os dados da fila AMQP e os escreve e salva constantemente em um arquivo *.log*, chamado de *detection.log*. A Figura 3.9 apresenta os parâmetros de entrada e saída utilizados no Logstash para captura e armazenamento dos dados, onde como entrada temos a fila AMQP do RabbitMQ (chamada de realtime_amqp) e como saída temos o arquivo *detection.log*.

```
input {

        rabbitmq {

                id => "rabbitmq_logs"
                host => "172.16.16.7"
                port => 5672
                vhost => "/"
                queue => "realtime_ampq"
                ack => false

        }
}

output {

  file {

    path => "/var/log/detection.log"
    codec => line { format => "%{message}"}

  }

}
```

Figura 3.9: Parâmetros definidos no Logstash.

Dado isso, o arquivo *detection.log* foi referenciado no agente Wazuh, no qual fará a leitura toda vez que ocorrer alterações no arquivo, para a detecção de alarmes provenientes do sistema. Como foi desenhado um modelo de log próprio, o servidor Wazuh não reconhecerá o modelo de log criado, impossibilitando a geração de alertas de predições. Para que o log seja interpretado, foi necessário criar decodificadores e regras de alarmes personalizadas no servidor Wazuh. Primeiramente, a Figura 3.10 mostra que foi criado dois decodificadores (um para cada tipo de detecção), onde foi necessário apresentar em que local do log criado está as informações que poderão ser filtradas futuramente no Kibana. Dessa forma, a Figura 3.10 mostra que foi referenciado, através de regex, o endereço ip de origem, o protocolo, a porta de destino, e o campo de informação extra (que é a predição do sistema) para cada tipo de decodificador.

Como segunda personalização no servidor Wazuh, foi criado regras de alarmes para cada tipo de detecção apresentada no sistema. A Figura 3.11 mostra as regras criadas, onde em cada uma delas foi indicado o nível de importância de alarme, no qual foi escolhido pelo autor o uso de nível 4 para conexões normais

```
<decoder name="ml_detection_normal">
  <program_name>^ml_detection_normal</program_name>
</decoder>

<decoder name="ml_detection_attack">
  <program_name>^ml_detection_attack</program_name>
</decoder>


<decoder name="ml_detection_normal">
  <parent>ml_detection_normal</parent>
  <regex>Connection from (\d+.\d+.\d+.\d+) using (\w+) protocol in a port (\d+), results in a (\w+) prediction</regex>
  <order>srcip,protocol,dstport,extra_data</order>
</decoder>

<decoder name="ml_detection_attack">
  <parent>ml_detection_attack</parent>
  <regex>Connection from (\d+.\d+.\d+.\d+) using (\w+) protocol in a port (\d+), results in a (\w+) prediction</regex>
  <order>srcip,protocol,dstport,extra_data</order>
</decoder>
```

Figura 3.10: Decodificadores criados no servidor Wazuh.

e nível 9 para conexões de alarmes (as conexões normais receberam nível 4 apenas para fins de teste e coleta de dados, pois este é um dos primeiros níveis que obriga o Wazuh a mostrar o alarme). Ainda na Figura 3.11, podemos observar que foi configurado, também, as descrições a serem apresentadas quando ocorrerem os devidos alarmes.

```
<group name="ml_detection">

  <rule id="100010" level="4">
    <program_name>ml_detection_normal</program_name>
    <description>Connection with Normal Prediction.</description>
  </rule>

  <rule id="100011" level="9">
    <program_name>ml_detection_attack</program_name>
    <description>Connection with Attack Prediction. Attack Alert.</description>
  </rule>


</group>
```

Figura 3.11: Regras de alarmes criadas no servidor Wazuh.

Desta forma, a Figura 3.12 apresenta exemplos de alarmes, na ferramenta kibana, gerados automaticamente pelo servidor Wazuh, através dos dados providos pelo *framework*, onde o primeiro deles é referente a uma conexão normal e o segundo a uma conexão predita como ataque.

	Time	rule.description	rule.level	rule.id
›	Jul 21, 2022 @ 16:31:53.532	Connection with Normal Prediction.	4	100010
›	Jul 21, 2022 @ 16:31:53.487	Connection with Attack Prediction. Attack Alert.	9	100011

Figura 3.12: Exemplos de alarmes criados no kibana pelo servidor Wazuh.

3.6 DETALHAMENTO DA ARQUITETURA DO SERVIDOR CENTRAL

O servidor central foi desenvolvido utilizando a ferramenta Docker, onde diversos serviços foram organizados em *containers* específicos por aplicação (o Logstash e o agente Wazuh são exceções, pois não são *containers*, e sim serviços instalados no próprio servidor central, como apresentado na Figura 3.13). Diante disso, a arquitetura do servidor central foi subdividida em três partes essências para o funcionamento correto do *framework* proposto. Na Figura 3.13, é possível observar com detalhes cada *container* utilizado e suas ligações diretas dentro do servidor, demostrando, assim, o fluxo de funcionamento do *framework*. A seguir será apresentado a subdivisão dos serviços utilizados.

1. **Bancos de dados**

   (a) **Postgresql -** *Container* responsável pela aplicação PostgreSQL, ou seja, um banco relacional que armazena as informações dos pacotes capturados.

   (b) **Redis -** Responsável por armazenar a variável de estado que indica o último pacote que foi analisado e predito corretamente, utilizando o banco de dados Redis.

   (c) **MinIO -** Implementação de um banco de dados responsável por emular um AWS S3. Este *container* é importante, pois está é a maneira correta (segundo a documentação do MLFlow [58]) de armazenar os artefatos do modelo treinado.

2. **Processamento**

   (a) **Main -** Este é o *container* principal, sendo responsável pela filtragem e classificação dos dados.

   (b) **MLFlow -** Responsável pelo modelo treinado, enviando, sempre que solicitado pelo Main, as predições para cada conexão analisada. Este *container* também é responsável pelo gerenciamento do ciclo de vida do modelo de aprendizado de máquina utilizado.

   (c) **RabbitMQ -** Recebe em sua fila os logs das classificações efetuadas pelo Main, armazenando-as para entregar ao *consumer*.

3. **Visualização**

   (a) **RabbitMQ -** Recebe os dados presentes na fila e os envia para o Logstash.

   (b) **Logstash -** Os logs de classificação são inseridos no Logstash, sendo escritos no arquivo *detection.log*.

   (c) **Agente Wazuh -** Tem como função principal verificar constantemente o arquivo de log e enviar suas informações ao servidor Wazuh.

   (d) **Servidor Wazuh -** Responsável pelo servidor Wazuh, no qual analisa os dados enviados pelo agente Wazuh, através dos decodificadores e das regras criadas, gera os alarmes automáticos.

   (e) **ElasticSearch -** Recebe os dados provenientes do servidor Wazuh, armazenando-os para que seja possível criar gráficos, tabelas, *dashboards*, entre outros.

(f) **Kibana -** Este *container* é responsável por apresentar visualmente os alarmes ao administrador, além de proporcionar a visualização dos possíveis gráficos, tabelas e *dashboards* a serem criados.

Pensando na implementação real de todo esse sistema, foi utilizado um servidor central com 8 GB de RAM, 1 processador e aproximadamente 60 GB de armazenamento de disco. O sistema operacional escolhido foi o Ubuntu Server 22.04 LTS.

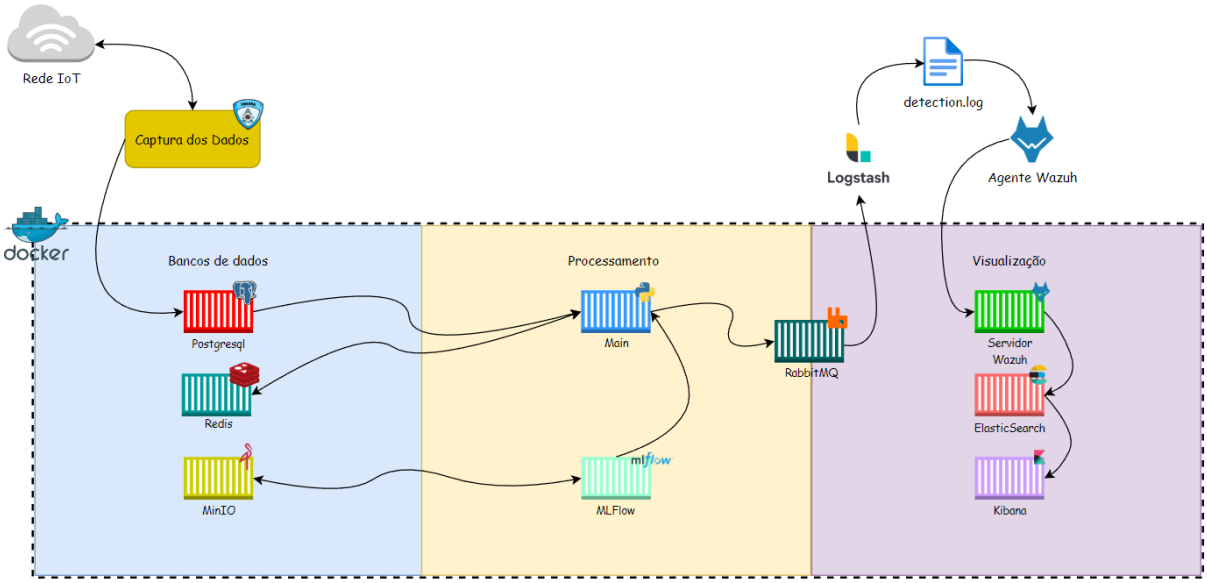

Figura 3.13: Arquitetura interna do servidor.

4 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta os experimentos executados como forma de validação do sistema, bem como os seus resultados envolvidos. Os resultados são baseados nas métricas mais utilizadas na literatura, sendo estes apresentados em forma de tabelas e gráficos. No final, os resultados obtidos serão discutidos, comparando-os com trabalhos correlatos na literatura.

4.1 EXPERIMENTOS

Para validar o *framework* desenvolvido, foram desenhados dois experimentos com diferentes abordagens. O primeiro deles é referente a verificação do desempenho do sistema no arquivo de teste do dataset NSL-KDD. O segundo experimento, tem como objetivo verificar o desempenho do *framework* em situações da vida real, ou seja, através de ataques ocorridos no dia a dia. Cada um deles será explanado, de forma detalhada, a seguir.

4.1.1 Experimento com o arquivo de teste do NSL-KDD

Como passo fundamental, foi adquirido o arquivo de teste do dataset NSL-KDD (disponível em [21]), onde consiste de cerca de mais de 22 mil registros de dados. Dentre esses registos de conexões, mais de 33% são ataques DoS, além de cerca de 43% serem conexões normais e 24% referentes a outros ataques (conforme apresentado na Tabela 2.2). Esses dados são referentes as conexões sem redundância, se formos considerar os pacotes redundantes temos que o arquivo de teste do NSL-KDD, tem cerca de mais de 310 mil pacotes, sendo mais de 88% deles ataques executados (conforme a Figura 4.1). O arquivo de teste do NSL-KDD é, também, uma excelente ferramenta para validação da escolha do método de aprendizado de máquina, pois pode-se utilizar-lo para verificar quais são os melhores métodos com este tipo de sistema, além de poder agregar na seleção dos melhores hiperparâmetros.

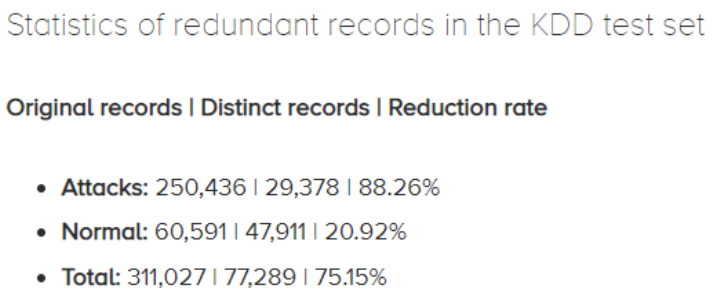

Figura 4.1: Estatística dos registros redundantes no arquivo de teste do NSL-KDD [21].

Inicialmente, foi feita uma filtragem dos dados, similarmente ao apresentado no Capítulo 3. Dessa forma, foi transformado o dataset para identificação apenas de ataques DoS (o Algoritmo 4 apresenta o pseudocódigo utilizado). Além disso, foi executado a redução de *features*, apresentando, assim, apenas as

17 *features* igualmente treinadas no modelo de aprendizado de máquina.

Algoritmo 4: Pseudocódigo referente a transformação do arquivo de teste do NSL-KDD

```
label ← [] ;

dos_attack ← ataques_DoS_presentes_no_dataset;

for index in test_file.label do
    if index = dos_attack then
        label.append(1);
    else
        label.append(0);
    end
end

test_file['label'] ← label;
```

Como segundo passo, e para confirmar que o método *Random Forest* é o melhor para o treinamento desse dataset, foi executado testes com os principais métodos de aprendizado de máquinas com referencia na literatura neste dataset. Os métodos utilizados foram:

- KNN (do inglês, *k-nearest neighbors algorithm*).
- Catboost.
- Adaboost.
- Decision Tree.
- Random Forest.

Para comparação dos modelos apresentados, foi utilizado as métricas de acurácia, precisão, *recall* e *F1-score*. A Tabela 4.1 mostra o resultado obtido, onde podemos verificar que o método *Random Forest* obteve a melhor acurácia, precisão, *recall* e *F1-score* com os dados do arquivo de teste do NSL-KDD, justificando, assim, seu uso como método de treinamento do sistema criado.

Tabela 4.1: Resultado dos principais métodos de aprendizado de máquina usando o arquivo de teste do NSL-KDD.

	Acurácia	Precissão	Recall	F1-score
KNN	0.8950	0.8965	0.8951	0.8923
Catboost	0.9115	0.9151	0.9115	0.9088
Adaboost	0.9122	0.9152	0.9122	0.9097
Decision Tree	0.8967	0.8975	0.8967	0.8942
Random Forest	**0.9190**	**0.9217**	**0.9190**	**0.9168**

4.1.2 Experimento com ataques em tempo real

Neste segundo experimento, foi desenvolvido um cenário para identificar a eficácia do *framework* em situação de ataques em tempo real. Desta forma, utilizando o emulador GNS3 (do inglês, *Graphical Network Simulator 3*) em um servidor Dell PowerEdge R740 (com 32 GB de RAM) disponibilizado pelo laboratório LATITUDE, foi criada uma topologia IoT para os devidos testes. Vale a pena ressaltar que, o GNS3 é um emulador de redes de código aberto que permite a criação de topologias independentes, no qual é possível, através de dispositivos virtuais ou físicos, gerar redes complexas. O GNS3 é largamente utilizado em diversas instituições acadêmicas e empresas (como Exxon, Walmart, AT&T, NASA, entre outras), além de ser popular em exames de certificações profissionais na área de redes. Seu uso mais comum é na criação e execução de provas de conceito.

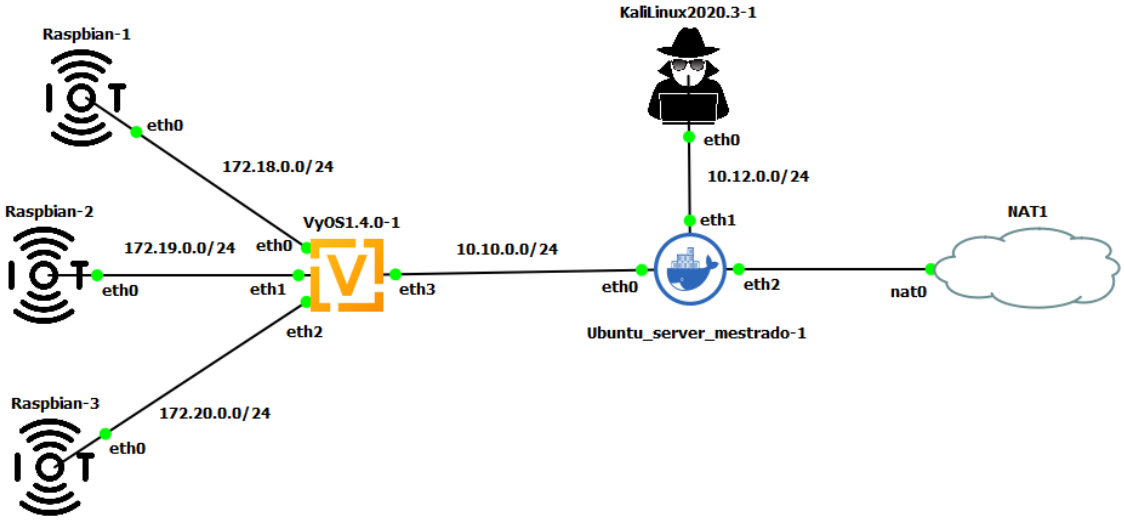

Figura 4.2: Topologia criada para o experimento.

A Figura 4.2 mostra a topologia criada para o experimento, onde podemos observar o uso de sete elementos distintos, onde cada um deles será apresentado a seguir.

- **Raspbian-1:** Emulação de um micro-controlador Raspberry, utilizando o sistema operacional próprio da marca, com o uso de 1 GB de RAM.
- **Raspbian-2:** Emulação de um segundo micro-controlador Raspberry, utilizando o sistema operacional próprio da marca, com o uso de 1 GB de RAM.
- **Raspbian-3:** Emulação de um terceiro micro-controlador Raspberry, utilizando o sistema operacional próprio da marca, com o uso de 1 GB de RAM.
- **VyOS1.4.0-1:** O VyOS é um emulador de roteador de código aberto, amplamente usado para topologias de testes.
- **Ubuntu_server_mestrado_1:** Este é o servidor central, no qual abrange o *framework* desenvolvido em um máquina Ubuntu Server 22.04 LTS, com 8 GB de RAM e 60 GB de armazenamento de disco.

- **KaliLinux2020.3-1:** Máquina virtual com o sistema Kali Linux instalado para a execução de ataques, utilizando 2 GB de memória RAM.
- **NAT1:** Nuvem NAT que prover acesso com a internet externa à topologia.

Podemos observar também, que, na Figura 4.2, há seis redes distintas em toda a topologia. A três primeiras, são referentes a conexão dos dispositivos IoT com o roteador VyOS (172.18.0.0/24, 172.19.0.0/24, 172.20.0.0/24). A quarta é a conexão entre o roteador e o servidor central (10.10.0.0/24), já a quinta é a conexão entre do Kali Linux no servidor central (10.12.0.0/24). Por fim, temos a conexão entre o servidor central e a nuvem NAT (modo DHCP). Vale ressaltar que, o Kali Linux ligado ao servidor central representa algum intruso que consiga ter acesso à rede de fora da rede interna criada.

Pensando em metodologia, a topologia criada visa simular um ataque DoS em um sistema IoT, através de uma prova de conceito. O intuito é, utilizando o Kali Linux, gerar ataques DoS nos *Raspbians*, de forma a avaliar o funcionamento do *framework* desenvolvido. Como primeiro passo, foi instalado e configurado a ferramenta *Mosquitto* entre os dispositivos IoT e o servidor central, para gerar tráfegos MQTT. Na configuração executada, foi adicionado o MQTTS, ou seja, o uso do protocolo MQTT com TLS 1.2. Após as realizada as configurações, foi criado dois scripts em Python. O primeiro é usado para simular a captura e envio dos sensores de informações para o servidor central, e o segundo, para receber e armazenar os dados recebidos. Os Algoritmos 5 e 6 apresentam os pseudocódigos usados. A ideia dos scripts foi simular o uso de dois sensores, um de temperatura e um de umidade. Dado isso, foi configurado todas as redes e as rotas apresentadas na topologia da Figura 4.2.

Ao termino da configuração, foi iniciado o experimento ligando todas as máquinas envolvidas na topologia. Como primeiro passo, foi gerado o tráfego MQTTS pelos dispositivos IoT para o servidor central, utilizando os scripts já apresentados. O segundo passo foi verificar se o Kali Linux tinha conexão com os dispositivos IoT. Dado que a conexão ocorreu, foi utilizada a ferramenta nmap para verificar as portas abertas nos dispositivos IoT, onde identificou-se que a porta do *Mosquitto* (porta 1883) estava aberta. A Figura 4.3 mostra um exemplo do nmap no Raspbian-1.

```
kali@kali:~$ nmap -p1-65535 80 172.18.0.1
Starting Nmap 7.91 ( https://nmap.org ) at 2022-09-30 08:20 EDT
Nmap scan report for 172.18.0.1
Host is up (0.011s latency).
Not shown: 65534 closed ports
PORT     STATE SERVICE
1883/tcp open  mqtt

Nmap done: 2 IP addresses (1 host up) scanned in 18.31 seconds
```

Figura 4.3: Exemplo do nmap executado.

Após a execução do nmap, foi planejado o ataque aos dispositivos IoT, onde dividiu-se em jorradas de 250 pacotes por execução de ataque para não sobrecarregar os recursos dos dispositivos Raspbians. O ataque escolhido foi o de syn_flood, onde já possui uma plataforma exclusiva no metasploit do Kali Linux. A Figura 4.4 mostra a configuração dos ataques executados. Portanto, para o experimento, foi gerado cerca de 33870 conexões no sistema (conexões normais e maliciosas somadas) em mais de 12 horas de conexão ininterrupta. Dentre essas conexões, 7500 são ataques do Kali Linux nos dispositivos IoT. A Figura 4.5

mostra os dados das conexões geradas.

Figura 4.4: Exemplo da configuração dos ataques executados.

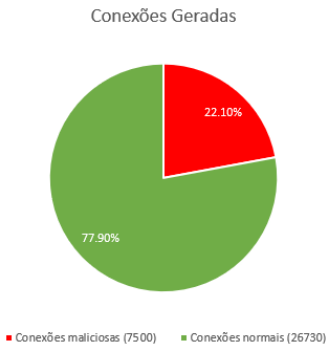

Figura 4.5: Conexões geradas no experimento.

4.1.3 Métricas de desempenho utilizadas nos experimentos

O sistema criado classifica os resultados de forma binária, ou seja, classifica apenas em duas classes (ataque ou não ataque). Dessa forma, existe algumas métricas utilizadas na literatura para este tipo de classificação, quando utiliza-se de aprendizado de máquina como método de seleção de classes. Com isso, as métricas escolhidas foram a acurácia, precisão, *recall* e *F1-score*, pois são as mais empregadas na literatura. Todas essas métricas são baseadas na identificação do universo de possível resultados do sistema. Os dados deste universo é subdividido do seguinte modo:

- **Verdadeiros positivos (VP):** Os ataques que foram corretamente identificados como ataque.

Algoritmo 5: Pseudocódigo referente ao script criado para simular captura e envio dos sensores.

```
count ← 0;

while True do
    Espera 3 segundos;
    if count = 0 then
        humidity_fake_value ← Valor aleatório decimal entre 50 e 100;
        date ← date();
        humidity_data["date"] ← date;
        humidity_data["data"] ← humidity_fake_value;
        humidity_json_data ← json.dumps(humidity_data);

        Publica os dados no Tópico Humidity no MQTT;

        count ← 1;

    else
        temperature_fake_value ← Valor aleatório decimal entre 1 e 30;
        date ← date();
        temperature_data["date"] ← date;
        temperature_data["data"] ← temperature_fake_value;
        temperature_json_data ← json.dumps(temperature_data);

        Publica os dados no Tópico Temperature no MQTT;

        count ← 0;

    end
end
```

Algoritmo 6: Pseudocódigo referente ao script criado para receber e armazenar os dados dos sensores criados.

Function `on_message` $(mosq, obj, msg)$ **:**
  Salvar os dados recebidos;

Function `on_connect` $(mosq, obj, rc, properties = None)$ **:**
  mqttc.subscribe(MQTT_Topic, 0);

Function `on_subscribe` $(mosq, obj, mid, granted\_qos)$ **:**
  pass;

$mqttc \leftarrow mqtt.Client()$;

$mqttc.on\_message \leftarrow on\_message$;

$mqttc.on\_connect \leftarrow on\_connect$;

$mqttc.on\_subscribe \leftarrow on\_subscribe$;

mqttc.connect(MQTT_Broker, int(MQTT_Port), int(Keep_Alive_Interval)) ;

mqttc.loop_forever();

- **Verdadeiros negativos (VN):** Os não ataques que foram corretamente identificados como não ataque.
- **Falsos positivos (FP):** Os não ataques que foram identificados incorretamente como ataque.
- **Falsos negativos (FN):** Os ataques que foram identificados incorretamente como não ataque.

A ideia principal, para que seja um sistema confiável, é obter valores elevados de verdadeiro positivos e verdadeiros negativo, somados a valores mínimos de falso positivos e falsos negativos. A Equação 4.1, mostra a nossa primeira métrica, chamada de acurácia. Nela medimos a proporção de predições corretas (ou seja, verdadeiro positivo e verdadeiro negativo) entre todos os dado presentes no universo. Com isso, podemos identificar o desempenho da qualidade da detecção binária, ou seja, a porcentagem de acerto na detecção.

$$Acurácia = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.1}$$

A segunda métrica amplamente utilizada é a precisão que, como mostrado na Equação 4.2, relaciona os verdadeiros positivos com os falsos positivos, ou seja, dado as amostra classificadas como ataques, qual a proporção de acertos (amostra classificadas corretamente como ataques).

$$Precisão = \frac{VP}{VP + FP} \tag{4.2}$$

O *Recall* é a terceira métrica utilizada, onde, como podemos observar na Equação 4.3, associa as

amostras de verdadeiro positivo com as de falso negativo. Isso ocorre, pelo fato de seu alvo ser relacionar as amostras de ataques, ou seja, dado que são ataques, qual a proporção de amostra que foram corretamente classificadas como ataques.

$$Recall = \frac{VP}{VP + FN} \tag{4.3}$$

Por ultimo, temos a métrica chamada de *F1-score* (Equação 4.4) que, por sua vez, relaciona a precisão com o *recall*. Isso acontece, por haver uma relação de causa e efeito entre a precisão e o *recall*, pois, uma precisão muita alta pode gerar um resultado de *recall* relativamente baixo, e a reciproca é verdadeira, ou seja, um *recall* muito alto pode gerar uma precisão relativamente baixa. Para facilitar a verificação da validação dos modelos de aprendizado de máquina, a literatura utiliza-se da métrica *F1-score*, que simplesmente é a média harmônica entre as métricas de precisão e o *recall*. O que se busca é um alto *F1-score*, onde representa altas taxas de precisão e de *recall* em conjunto.

$$F1\text{-}Score = 2 * \frac{Precisão * Recall}{Precisão + Recall} \tag{4.4}$$

4.2 RESULTADOS

Com o intuito de validar o sistema, foram elaborados dois experimentos distintos. O primeiro deles consiste, usando as métricas mais utilizadas na literatura, em verificar o desempenho do sistema no arquivo de teste do NSL-KDD, onde, atualmente, é uma prática bem comum na literatura. O segundo experimento mostra a qualidade do *framework* desenvolvido em ataques em tempo real, com o funcionamento da captura, classificação e alarmes de ataques ocorridos.

4.2.1 Resultados com o arquivo de teste do NSL-KDD

Conforme descrito na seção de experimentos, julgou-se importante verificar o desempenho do sistema desenvolvido no arquivo de teste do NSL-KDD. De forma a validar os resultados obtidos, foram escolhidos os trabalhos, julgados mais relevantes, presentes na literatura, onde, para a validação da comparação dos resultados encontrados, torna-se importante destacar que, todos os trabalhos descritos abaixo utilizam-se do arquivo de teste do NSL-KDD para obtenção dos dados, além dos resultados serem especificamente para ataques do tipo DoS. Os trabalhos escolhidos são:

- **Yin et al. [53]:** Apresenta um sistema de detecção de intrusão utilizando técnicas de deep learning (recurrent neural networks (RNN-IDS)).
- **Ugwu et al. [51]:** Solução utilizando os métodos de Long Short Term Memory (LSTM) e Singular Value Decomposition (SVD) (esta última para extração de features) para a criação de um IDS focado em ataques DoS.
- **Yusof et al [59]:** Solução que propõe uma seleção adaptativa de *features*, no dataset NSL-KDD,

para ataques DoS.

- **Latah and Toker [52]:** Desenvolvimento de um sistema de detecção de intrusão hibrido, focado em minimizar a taxa de falsos positivos em ataques DoS.
- **Vinayakumar et al. [54]:** Demostração de um sistema de detecção de intrusão baseado em deep learning, encontrando os melhores valores de parâmetros para este método.

Inicialmente, foi comparado a métrica de acurácia do *framework* desenvolvido, com os demais trabalhos (veja a Figura 4.6). Podemos observar que, o sistema proposto, juntamente com os trabalhos de Yusof et al [59] e Vinayakumar et al. [54], possuem os melhores valores referentes a essa métrica.

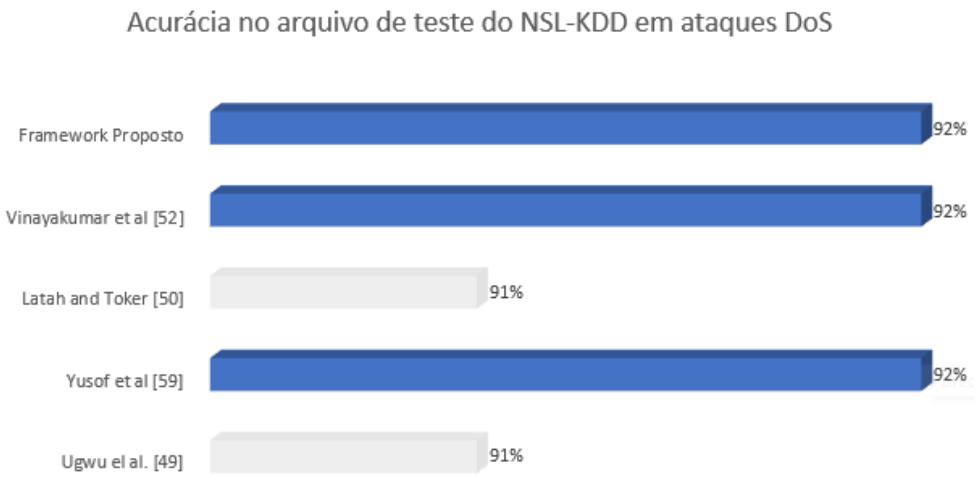

Figura 4.6: Comparação da métrica de acurácia entre os trabalhos citados e a solução desenvolvida.

Como segundo passo, foi comparada a métrica de precisão entre os trabalhos citados e o *framework* criado, no qual os resultados são apresentados na Figura 4.7. Vale ressaltar que, o Latah and Toker [52] é o que obteve o melhor resultado, seguido pelo *framework* proposto.

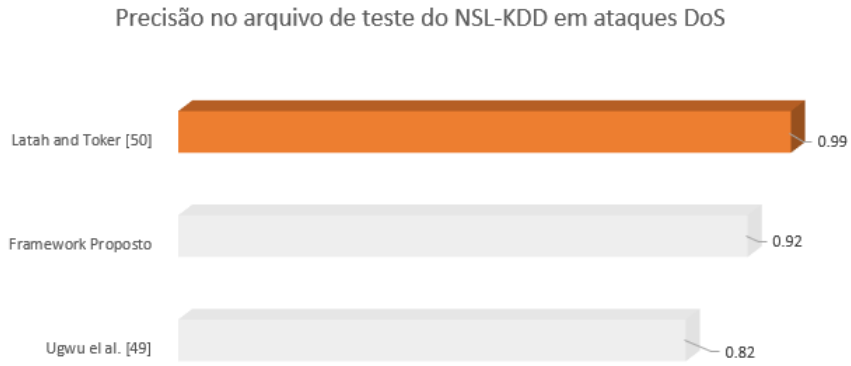

Figura 4.7: Comparação da métrica de precisão entre os trabalhos citados e a solução desenvolvida.

A terceira métrica comparada, foi a métrica de *recall*, onde os resultados são apresentados na Figura 4.8. Podemos verificar que, a solução proposta por Ugwu et al. [51] obteve melhor resultados entre os trabalhos observados, seguido imediatamente pelo *framework* proposto.

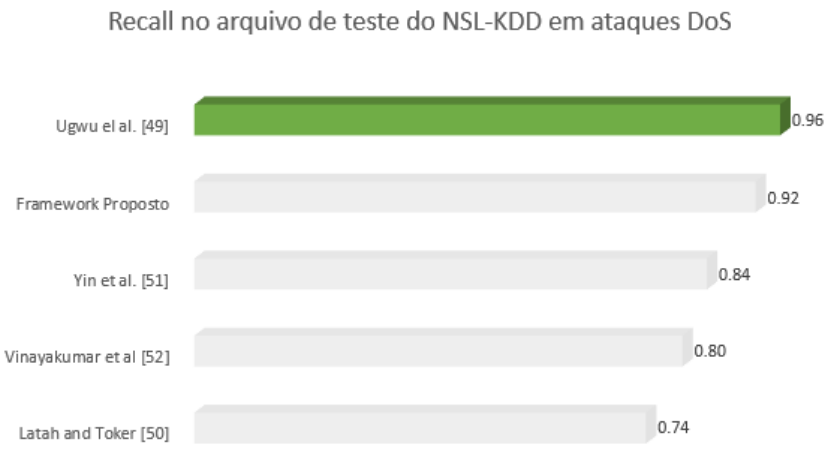

Figura 4.8: Comparação da métrica de *recall* entre os trabalhos citados e a solução desenvolvida.

A última métrica analisada foi a *F1-Score* (veja a Figura 4.9, onde, como destacado anteriormente, é uma média harmonia entre a métrica de precisão e o *recall*. Nesta métrica, o *framework* proposto obteve o melhor resultado, seguido pelo trabalho do Ugwu et al. [51].

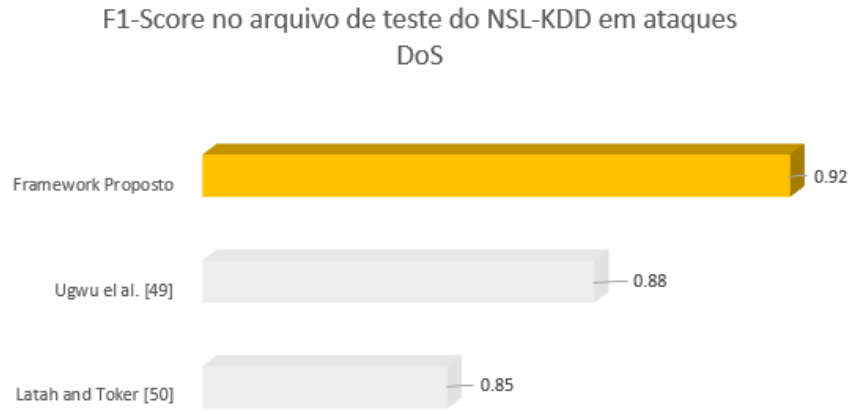

Figura 4.9: Comparação da métrica de *F1-Score* entre os trabalhos citados e a solução desenvolvida.

Para obter uma visão geral, a Tabela 4.2 mostra um resumo das principais métricas utilizadas para as comparações, apresentando, assim, os valores absolutos, com duas casas decimais, de cada um dos trabalhos comparados.

4.2.1.1 Discussão dos resultados

Sobre os resultados alcançados, podemos observar que o *framework* desenvolvido liderou duas das quatro métricas utilizadas na comparação, perdendo apenas nas métricas de precisão e recall. Os trabalhos que mais se aproximaram dos resultados apresentados pelo sistema criado foram o do Latah and Toker [52] e Ugwu et al. [51], onde o primeiro, na métrica de precisão, liderou com certa vantagem, e o segundo liderou na métrica de *recall*. É importante frisar que, o trabalho do Latah and Toker [52] foi focado em

Tabela 4.2: Resumo das métricas das diferentes soluções desenvolvidas que foram testadas com o arquivo de teste do NSL-KDD.

	Acurácia	Precisão	*Recall*	*F1-Score*
Yin et al. [53]	–	–	0.84	–
Ugwu el al. [51]	0.91	0.82	**0.96**	0.88
Yusof et al [59]	**0.92**	–	–	–
Latah and Toker [52]	0.91	**0.99**	0.74	0.85
Vinayakumar et al [54]	**0.92**	–	0.80	–
Framework Proposto	**0.92**	0.92	0.92	**0.92**

diminuir a taxa de falsos positivos, onde traduziu-se em um alto resultado da métrica de precisão, mas como discutido anteriormente, o aumento forçado dessa métrica pode custar uma diminuição no *recall*, algo que ocorreu neste sistema, fazendo com que na média harmônica entre a precisão e o *recall* (*F1-Score*) gerasse um valor relativamente inferior ao sistema desenvolvido. Similarmente, a solução de Ugwu et al. [51] foi projetada para manter alta taxa de detecção (recall), que no fim gerou baixas taxas de precisão, diminuindo, assim, a métrica de F1-Score. Portanto, podemos confirmar que o *framework* apresentado é o mais robusto entre os trabalhos que foram comparados, quando utilizado o arquivo de teste do NSL-KDD para a comparação entre os sistemas (alcançando, assim, o objetivo especifico número 5, apresentado no capítulo 1).

4.2.2 Resultados do experimento em tempo real

Como apresentado anteriormente, foram gerados 33870 conexões ao todo nesse experimento, onde 26370 foram conexões normais e 7500 conexões maliciosas. A Figura 4.10 mostra a matriz de confusão referente ao resultados alcançados, onde temos:

- 7465 conexões classificadas em verdadeiro positivo.
- 35 conexões classificadas em falso negativo.
- 0 conexões classificadas em falso positivo.
- 26370 conexões classificadas em verdadeiro negativo.

Dado isso, podemos observar que foram criados 7465 alarmes corretos de ataque no Wazuh, sendo que 35 conexões passaram ilesa do *framework* desenvolvido. Vale destacar também que, não teve nenhuma conexão normal sendo classificada incorretamente como ataque pelo sistema. Com essas informações, foi possível calcular todas as métricas apresentadas anteriormente, nas quais são referencia na literatura. A Figura 4.11 mostra os resultados das métricas em forma gráfica e a Tabela 4.3 mostra o resumo dos resultados.

4.2.2.1 Discussão dos resultados

Mediante aos resultados encontrados no experimento de ataques em tempo real, podemos observar que

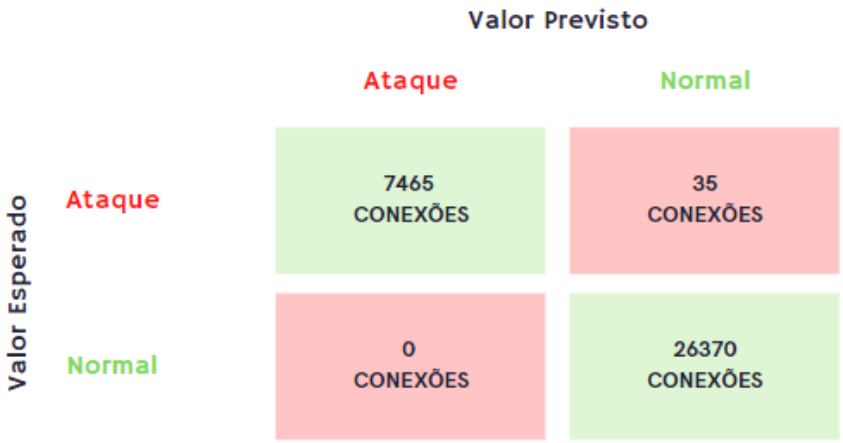

Figura 4.10: Matriz de confusão do experimento realizado.

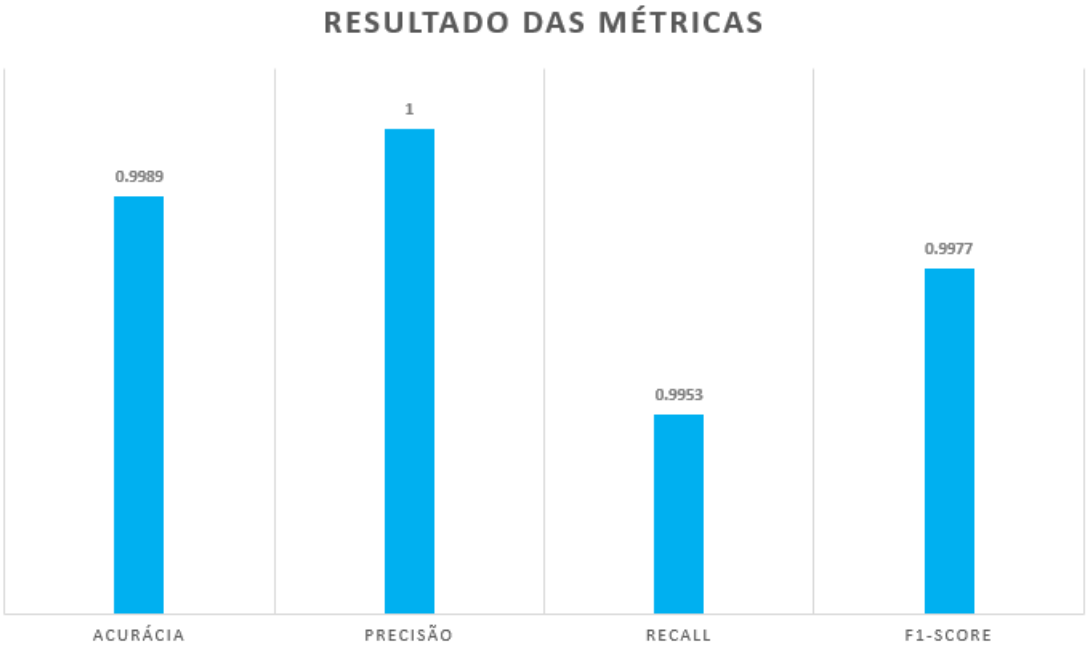

Figura 4.11: Resultado das principais métricas.

Tabela 4.3: Resumo dos resultados das métricas calculadas.

Métricas	**Resultados**
Acurácia	0.9989 (99,89%)
Precisão	1
Recall	0.9953
F1-Score	0.9977

o *framework* desenvolvido se comportou muito bem nesta situação, gerando resultados surpreendentes. Isso se dá pela integração continua de todos as ferramentas utilizadas, desde a captura dos dados através do Tshark à visualização e os alarmes gerados automaticamente pela ferramenta Wazuh combinados com o ELK. Vale a pena ressaltar que, o sistema teve mais dificuldade em reconhecer o que realmente é ataque, quando comparado com a identificação de conexão normais, o que gerou alguns falsos negativos. Isto é explicado pela grande variação que ocorre entre ataques (mudança continua de IPs de origem, ataques rápidos e não periódicos) em máquinas distintas, onde pode gerar variações nesta identificação. Mesmo com isso, o número de conexões classificadas como falso negativo foi muito pequeno comparado ao total de ataques, gerando, assim, uma acurácia, um *recall* e um *F1-Score* elevado.

5 CONCLUSÃO E TRABALHOS FUTUROS

Neste Capítulo apresentaremos a conclusão do trabalho, no qual foi subdividida em duas seções para o completo entendimento. A primeira seção apresentada, diz respeito a visão Geral do trabalho apresentado (Seção 5.1). A segunda aborda as limitações apresentadas na pesquisa e no sistema desenvolvido e os trabalhos futuros a serem executados, visando, assim, a melhora do sistema (Seção 5.2).

5.1 VISÃO GERAL

Este trabalho apresentou um sistema para a detecção de ataques DoS em redes IoT, utilizando metodologias de aprendizado de máquinas para a classificação dos fluxos de dados. Este sistema foi arquitetado para ser uma solução completa, de modo a proporcionar um gerenciamento de alarme dos ataque ocorridos, sendo, assim, modelado desde a captura dos fluxos de dados até a visualização do alarmes de ataques ocorridos.

Os objetivos pretendidos pelo sistema desenvolvido foram:

- Proporcionar uma identificação de ataques eficaz na camada de aplicação, através do gerenciamento de alarmes.
- Gerar uma solução visando redes IoT e suas peculiaridades.
- Apresentar a integração de MLOps e o gerenciamento de ciclo de vida dos métodos de classificação de ataques com aprendizado de máquina.
- Desenvolver um projeto com o mínimo custo computacional possível.
- Fornecer uma solução com fácil implementação e replicação, não dependendo de cenários específicos.

O *framework* apresentado é, em sua totalidade, executado em um servidor central, no qual seu processo de funcionamento foi subdividida em quatro grandes fases de execução (conforme apresentado no Capítulo 3). A primeira delas é a fase denominada captura dos dados, onde o servidor central observar e captura os pacotes TCPs, UDPs e ICMPs que circulam na rede interna, através da ferramenta Tshark. Com o intuito de reduzir todos os dados capturados, foram selecionados apenas os dados com maior relevância, em cada tipo de pacote, para a classificação futura, no qual estes dados foram salvos em um banco de dados relacional Postgres. Os dados selecionados em cada tipo de pacote foram:

- **TCP:** Tempo da coleta do pacote, o endereço e a porta de destino e origem, o *checksum*, as *flags* TCP e os *bytes* do pacote analisado.

- **UDP:** Tempo da coleta do pacote, o endereço e a porta de destino e origem, o *checksum* e os bytes do pacote analisado.
- **ICMP:** Tempo da coleta do pacote, o endereço de destino e origem, o *checksum* e os *bytes* do pacote analisado.

A segunda fase é denominada filtragem dos dados, no qual é responsável por filtrar os dados armazenados no banco de dados Postgres. Este filtro é relacionado ao dataset utilizado, pois os dados capturados necessitam estar no mesmo formato que os dados presentes no dataset, além de serem alterados para classificação binária, ou seja, selecionando ataque apenas os ataques DoS. Para isso, foi gerado conexões com os dados dos pacotes capturados, sendo, assim, calculada as 17 *features* necessárias para a classificação dos ataques. Após a execução da filtragem, os dados são enviados para a terceira fase do sistema. Vale a pena ressaltar que, o dataset originalmente possui 43 *features*, mas foi executado técnicas de *feature engineering* que possibilitaram diminuir as *features* para o sistema desenvolvido.

Na terceira fase temos a classificação do dados, onde que dado colhido nas etapas anteriores é classificados como possível ataque DoS ou não por suas características. Para isso, foi utilizado o método de classificação baseado em árvores binárias chamados Random Forest, no qual foi treinado utilizando o dataset de treino do NSL-KDD, sendo seus hiperparâmetros selecionados pela ferramenta Optuna. Na visão próxima ao tempo real, foi utilizado o Mlflow, onde torna-se possível um classificação rápida e eficiente além da possibilidade de gerenciamento de ciclo de vida do método de classificação. Como última etapa, temos a visualização dos dados, onde foi utilizado o ELK com o incremento do plugin do Wazuh, de forma a gerar alertas automáticos com o ataque fosse detectado. Para isso, foi gerado um modelo único de log, além da criação de regras e decodificadores personalizados no Wazuh.

Foram executados dois experimento distintos para a validação do sistema, onde o primeiro deles foi verificar o desempenho do framework desenvolvido no arquivo de teste do dataset NSL-KDD (KDD-Test+), comparando os resultados obtidos com os presentes na literatura, onde foram alcançados os valores de 91.90% de acurácia, 0.9217 de precisão, 0.9190 de recall e 0.9168 de F1-Score, sendo o sistema mais robusto entre os trabalhos apresentados. O segundo experimento foi desenvolvido com o intuito de validar o sistema em ataques DoS em real-time, onde foi criada uma topologia com os dispositivos IoT, o framework desenvolvido e um invasor (utilizando kali Linux) no emulador GNS3. Dessa forma, foram gerados ataques syn flood (através do metasploit), obtendo, assim, o desempenho do sistema próximo ao tempo real

5.2 LIMITAÇÕES E TRABALHOS FUTUROS

A solução construída se mostrou eficaz, porém ainda há limitações que, através de melhorias, podem fortalecer o desempenho e funcionalidade do sistema, nas quais podem ser exploradas em trabalhos futuros. Como possibilidades de melhorias, temos:

- A utilização de um sistema de detecção de intrusão híbrido, ou seja, não apenas detecção por anomalia, mas a junção de detecção por anomalia com detecções por assinaturas de ataques.

- O número limitado de classe de ataque, podendo ser mais diversificado.
- Diminuição do atraso entre a captura dos dados maliciosos até a geração do seu alarme.
- Treinamento com agregação de diversos dataset, gerando assim, maior generalização na classificação.
- Desenvolvimento de um auto aprendizado do modelo criado, utilizando o gerenciamento do ciclo de vida do método treinado.
- Elaboração de um dataset escalonável, através de big datas, com um grande volume de dados, proporcionando assim, a utilização massiva de técnicas de aprendizado profundo de máquinas.
- Detecção de ataques focados em múltiplas camadas.
- Detecção de ataques iniciados por dispositivos IoT infectados.

REFERÊNCIAS BIBLIOGRÁFICAS

1 VAILSHERY, L. S. *Prognosis of worldwide spending on the Internet of Things (IoT).* Disponível em: <https://www.statista.com/statistics/668996/worldwide-expenditures-for-the-internet-of-things/>. Acesso em: 4/10/2021.

2 HOWARTH, J. *Consumer IoT Spending Stats*. Disponível em: <https://explodingtopics.com/blog/iot-stats>. Acesso em: 29/08/2022.

3 SCOTT. *Current Security Challenges Facing the Internet of Things*. Disponível em: <https://www.colocationamerica.com/blog/security-challenges-of-iot>. Acesso em: 30/08/2022.

4 ALDHAHERI, S.; ALGHAZZAWI, D.; CHENG, L.; ALZAHRANI, B.; AL-BARAKATI, A. Deepdca: Novel network-based detection of iot attacks using artificial immune system. *Applied Sciences*, v. 10, p. 1909, 03 2020.

5 BARROS, M. *MQTT – Protocolos para IoT*. Disponível em: <https://embarcados.com.br/mqtt-protocolos-para-iot/>. Acesso em: 31/08/2022.

6 AFONSO, R.; NOGUEIRA, M. Utilização de sagbd no apoio à sistemas de manufatura. 09 2022.

7 GTA. *O que é e como funciona um IDS?* Disponível em: <https://www.gta.ufrj.br/grad/16_2/2016IDS/conceituacao.html>. Acesso em: 02/09/2022.

8 AHMAD, M.; SHAH, S. Mitigating malicious insider attacks in the internet of things using supervised machine learning techniques. *Scalable Computing*, v. 22, p. 13–28, 04 2021.

9 TIBCO. *O que é uma floresta aleatória?* Disponível em: <https://www.tibco.com/pt-br/reference-center/what-is-a-random-forest>. Acesso em: 02/09/2022.

10 KNOLDUS. *What is the ELK Stack?* Disponível em: <https://medium.com/@knoldus/what-is-the-elk-stack-ad8398dd265e>. Acesso em: 05/09/2022.

11 WICKRAMASINGHE, S. *Logstash 101: Using Logstash in a Data Processing Pipeline*. Disponível em: <https://www.bmc.com/blogs/logstash-using-data-pipeline/>. Acesso em: 05/09/2022.

12 DOCKER. *Use containers to Build, Share and Run your applications*. Disponível em: <https://www.docker.com/resources/what-container/>. Acesso em: 06/09/2022.

13 DOSSOT, D. Rabbitmq essentials. In: ____. [S.l.]: Packt Publishing, 2014. cap. 1, p. 2–3. ISBN 978-1-78398-320-9.

14 JIA, Y.; ZHONG, F.; ALRAWAIS, A.; GONG, B.; CHENG, X. Flowguard: An intelligent edge defense mechanism against iot ddos attacks. *IEEE Internet of Things Journal*, v. 7, n. 10, p. 9552–9562, 2020.

15 ESKANDARI, M.; JANJUA, Z. H.; VECCHIO, M.; ANTONELLI, F. Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. *IEEE Internet of Things Journal*, v. 7, n. 8, p. 6882–6897, 2020.

16 OTOUM, Y.; NAYAK, A. As-ids: Anomaly and signature based ids for the internet of things. *J. Netw. Syst. Manage.*, Plenum Press, USA, v. 29, n. 3, jul 2021. ISSN 1064-7570. Disponível em: <https://doi.org/10.1007/s10922-021-09589-6>.

17 KHRAISAT, A.; GONDAL, I.; VAMPLEW, P.; KAMRUZZAMAN, J.; ALAZAB, A. A novel ensemble of hybrid intrusion detection system for detecting internet of things attacks. *Electronics*, v. 8, n. 11, 2019. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/8/11/1210>.

18 SICATO, J.; SINGH, S. K.; RATHORE, S.; PARK, J. A comprehensive analyses of intrusion detection system for iot environment. *Journal of Information Processing Systems*, v. 16, p. 975–990, 09 2020.

19 CHAABOUNI, N.; MOSBAH, M.; ZEMMARI, A.; SAUVIGNAC, C. A onem2m intrusion detection and prevention system based on edge machine learning. In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. [S.l.: s.n.], 2020. p. 1–7.

20 LóPEZ, F. *OPTUNA: A Flexible, Efficient and Scalable Hyperparameter Optimization Framework*. Disponível em: <https://towardsdatascience.com/optuna-a-flexible-efficient-and-scalable-hyperparameter-optimization-framework-d26bc7a23fff>. Acesso em: 19/09/2022.

21 BRUNSWICK, U. of N. *NSL-KDD dataset*. Disponível em: <https://www.unb.ca/cic/datasets/nsl.html>. Acesso em: 12/07/2022.

22 FGV. *Professor da Direito Rio analisa impacto da 'Internet das Coisas' na sociedade*. Disponível em: <https://portal.fgv.br/noticias/professor-direito-rio-analisa-impacto-internet-coisas-sociedade>. Acesso em: 01/10/2021.

23 IDC-BRASIL. *5G vai movimentar US$ 25 bilhões no Brasil até 2025*. Disponível em: <https://proximonivel.embratel.com.br/5g-vai-movimentar-us-25-bilhoes-no-brasil-ate-2025/>. Acesso em: 29/08/2022.

24 ORACLE. *The Internet of Things (IOT) Technology*. Disponível em: <https://www.ericsson.com/en/internet-of-things>. Acesso em: 30/08/2022.

25 KHANAM, S.; AHMEDY, I. B.; IDRIS, M. Y. I.; JAWARD, M. H.; SABRI, A. Q. B. M. A survey of security challenges, attacks taxonomy and advanced countermeasures in the internet of things. *IEEE Access*, v. 8, p. 219709–219743, 2020.

26 FAROOQ, M.; WASEEM, M.; KHAIRI, A.; MAZHAR, P. A critical analysis on the security concerns of internet of things (iot). *International Journal of Computer Applications*, v. 111, p. 1–6, 02 2015.

27 SHARMEEN, S.; HUDA, S.; ABAWAJY, J. H.; ISMAIL, W. N.; HASSAN, M. M. Malware threats and detection for industrial mobile-iot networks. *IEEE Access*, v. 6, p. 15941–15957, 2018.

28 HAM, H.-S.; KIM, H.-H.; KIM, M.-S.; CHOI, M.-J. Linear svm-based android malware detection for reliable iot services. *Journal of Applied Mathematics*, v. 2014, p. 10, 2014.

29 KAUR, P.; SHARMA, S. Spyware detection in android using hybridization of description analysis, permission mapping and interface analysis. *Procedia Computer Science*, v. 46, p. 794–803, 12 2015.

30 KOH, J.; NEVAT, I.; LEONG, D.; WONG, L. Geo-spatial location spoofing detection for internet of things. *IEEE Internet of Things Journal*, v. 3, p. 1–1, 02 2016.

31 GEORGE, T. K.; JACOB, K. P.; JAMES, R. K. A proposed framework against code injection vulnerabilities in online applications. *Journal of Internet Technology*, v. 20, p. 83–96, 1 2019.

32 GROVER, J.; LAXMI, V.; GAUR, M. Attack models and infrastructure supported detection mechanisms for position forging attacks in vehicular ad hoc networks. *CSIT*, v. 1, p. 261–279, 9 2013.

33 ASHTON, K. That "internet of things"thing. *RFiD Journal*, v. 22, p. 97–114, 01 2009.

34 BAUER, M.; WALEWSKI, J. W. The iot architectural reference model as enabler. In: ____. *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. p. 17–25. ISBN 978-3-642-40403-0. Disponível em: <https://doi.org/10.1007/978-3-642-40403-0_3>.

35 MENDONCA, F. *PROPOSIÇÃO DE UM MODELO DE INTEROPERAÇÃO PEER-TO-PEER PARA INTERNET DAS COISAS – P2PIOT*. Tese (Doutorado) — Universidade de Brasília, 2019.

36 SUNDMAEKER, H.; GUILLEMIN, P.; FRIESS, P.; WOELFFLÉ, S. Vision and challenges for realising the internet of things. In: ____. 1. ed. Brussels, Belgium: European Union, 2010. cap. 3, p. 43. ISBN 978-92-79-15088-3.

37 HÖLLER, J.; TSIATSIS, V.; MULLIGAN, C.; KARNOUSKOS, S.; AVESAND, S.; BOYLE, D. From machine-to-machine to the internet of things. In: ____. [S.l.]: Elsevier Ltd., 2014. cap. 2, p. 10–13. ISBN 978-0-12-407684-6.

38 ORACLE. *O que é IoT?* Disponível em: <https://www.oracle.com/br/internet-of-things/what-is-iot/>. Acesso em: 30/08/2022.

39 MQTT. *Why MQTT?* Disponível em: <https://mqtt.org/>. Acesso em: 31/08/2022.

40 ORACLE. *O Que É um Banco de Dados?* Disponível em: <https://www.oracle.com/br/database/what-is-database/>. Acesso em: 01/09/2022.

41 LABS, R. *Redis*. Disponível em: <https://redis.io/>. Acesso em: 01/09/2022.

42 THAKKAR, A.; LOHIYA, R. Role of swarm and evolutionary algorithms for intrusion detection system: A survey. *Swarm and Evolutionary Computation*, v. 53, p. 100631, 2020. ISSN 2210-6502. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210650219303918>.

43 ANDREA, I.; CHRYSOSTOMOU, C.; HADJICHRISTOFI, G. Internet of things: Security vulnerabilities and challenges. In: *2015 IEEE Symposium on Computers and Communication (ISCC)*. [S.l.: s.n.], 2015. p. 180–187.

44 MITCHELL, T. Machine learning. In: ____. McGraw-Hill, 1997. (McGraw-Hill International Editions), cap. Preface, p. xv. ISBN 9780071154673. Disponível em: <https://books.google.com.br/books?id=EoYBngEACAAJ>.

45 BREIMAN, L. Random forests. *Machine Learning*, v. 45, p. 5–32, 10 2001.

46 TREVEIL, M.; OMONT, N.; STENAC, C.; LEFEVRE, K.; PHAN, D.; ZENTICI, J.; LAVOILLOTTE, A.; MIYAZAKI, M.; HEIDMANN, L. Introducing mlops. In: ____. [S.l.]: O'Reilly Media, 2020. cap. 1, p. 3.

47 MLFLOW. *MLFlow - An open source platform for the machine learning lifecycle*. Disponível em: <https://mlflow.org/>. Acesso em: 20/07/2022.

48 BAJER, M. Building an iot data hub with elasticsearch, logstash and kibana. In: *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. [S.l.: s.n.], 2017. p. 63–68.

49 STANKOVIĆ, S.; GAJIN, S.; PETROVIĆ, R. A review of wazuh tool capabilities for detecting attacks based on log analysis.

50 NAIK, N. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In: *2017 IEEE International Systems Engineering Symposium (ISSE)*. [S.l.: s.n.], 2017. p. 1–7.

51 UGWU, C. C.; OBE, O. O.; POPOOLA, O. S.; ADETUNMBI, A. O. A distributed denial of service attack detection system using long short term memory with singular value decomposition. In: *2020 IEEE 2nd International Conference on Cyberspac (CYBER NIGERIA)*. [S.l.: s.n.], 2021. p. 112–118.

52 LATAH, M.; TOKER, L. Minimizing false positive rate for dos attack detection: A hybrid sdn-based approach. *ICT Express*, v. 6, n. 2, p. 125–127, 2020. ISSN 2405-9595. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405959519303480>.

53 YIN, C.; ZHU, Y.; FEI, J.; HE, X. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, v. 5, p. 21954–21961, 2017.

54 VINAYAKUMAR, R.; ALAZAB, M.; SOMAN, K. P.; POORNACHANDRAN, P.; AL-NEMRAT, A.; VENKATRAMAN, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access*, v. 7, p. 41525–41550, 2019.

55 ANTONAKAKIS, M.; APRIL, T.; BAILEY, M.; BERNHARD, M.; BURSZTEIN, E.; COCHRAN, J.; DURUMERIC, Z.; HALDERMAN, J. A.; INVERNIZZI, L.; KALLITSIS, M.; KUMAR, D.; LEVER, C.; MA, Z.; MASON, J.; MENSCHER, D.; SEAMAN, C.; SULLIVAN, N.; THOMAS, K.; ZHOU, Y. Understanding the mirai botnet. In: . USA: USENIX Association, 2017. (SEC'17), p. 1093–1110. ISBN 9781931971409.

56 LATAH, M.; TOKER, L. Towards an efficient anomaly-based intrusion detection for software-defined networks. *CoRR*, abs/1803.06762, 2018. Disponível em: <http://arxiv.org/abs/1803.06762>.

57 XGBOOST. *Random Forests(TM) in XGBoost*. Disponível em: <https://xgboost.readthedocs.io/en/stable/tutorials/rf.html>. Acesso em: 19/09/2022.

58 MLFLOW. *MLflow Tracking*. Disponível em: <https://www.mlflow.org/docs/latest/tracking.html>. Acesso em: 22/09/2022.

59 YUSOF, A. R.; UDZIR, N. I.; SELAMAT, A.; HAMDAN, H.; ABDULLAH, M. T. Adaptive feature selection for denial of services (dos) attack. In: *2017 IEEE Conference on Application, Information and Network Security (AINS)*. [S.l.: s.n.], 2017. p. 81–84.