UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JARÉLIO GOMES DA SILVA FILHO

DETECÇÃO DE ANOMALIAS NO TRÁFEGO MQTT DE REDES IOT UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA

FORTALEZA

2021

JARÉLIO GOMES DA SILVA FILHO

DETECÇÃO DE ANOMALIAS NO TRÁFEGO MQTT DE REDES IOT UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Emanuel Bezerra Rodrigues

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S58d Silva Filho, Jarélio Gomes da.
Detecção de anomalias no tráfego MQTT de redes IOT utilizando técnicas de aprendizado de máquina / Jarélio Gomes da Silva Filho. – 2021.
66 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Ciências, Curso de Computação, Fortaleza, 2021.
Orientação: Prof. Dr. Emanuel Bezerra Rodrigues.

1. Internet das coisas. 2. Sistemas de detecção de intrusão. 3. MQTT. 4. Aprendizado de máquina. I. Título.

CDD 005

JARÉLIO GOMES DA SILVA FILHO

DETECÇÃO DE ANOMALIAS NO TRÁFEGO MQTT DE REDES IOT UTILIZANDO TÉCNICAS DE APRENDIZADO DE MÁQUINA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Ciência da Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Emanuel Bezerra Rodrigues (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Rafael Lopes Gomes
Universidade Estadual do Ceará (UECE)

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará (UFC)

A Deus. Aos meus pais, Jarélio Gomes e Cláudia Maria.

AGRADECIMENTOS

Ao Prof. Dr. Emanuel Bezerra Rodrigues, pela excelente orientação, suas sugestões, correções e todo apoio que me ajudaram não só neste trabalho, mas também durante o período de graduação e atuação no projeto RSI.

Aos professores participantes da banca examinadora José Neuman de Souza e Rafael Lopes Gomes pelo tempo, pelas valiosas colaborações e sugestões.

A Universidade Federal do Ceará por toda estrutura e apoio necessário para o meu aprendizado.

A minha família, Jarélio Gomes, Cláudia Maria, Jade Cristina, Jairo Cristiano, Joana Maria, Beatriz Gomes e Gabriel Napolião por fazerem parte e estarem juntos comigo em toda essa minha caminhada.

Aos colegas e amigos que fiz na universidade, compartilhando experiências, conhecimentos e tornando o meu processo de formação mais agradável. Especialmente aos meus amigos Franklyn e Samir que entraram nessa jornada de desenvolvimento de monografia comigo.

Ao projeto de extensão Residência em Segurança da Informação (RSI) que além de me proporcionar bastante conhecimento, me deu vários amigos que irei levar para vida.

A todos os professores e funcionários que fizeram parte da minha vida diretamente e indiretamente por meio da universidade, compartilhando todo seu conhecimento e suas vivências.

“Não viemos ao mundo para fazer o que os outros fazem. Não viemos para fazer um pouco melhor o que os outros fazem. Viemos para fazer o que só nós podemos fazer.”

(Trigueirinho)

RESUMO

Tendo em vista o crescimento das redes e dispositivos da internet das coisas e juntamente com eles diversas falhas de seguranças associadas, pesquisa-se sobre técnicas de aprendizado de máquina e sistemas de detecção de intrusão, a fim de tornar seguras as redes e dispositivos que comunicam-se utilizando o protocolo MQTT. Para tanto, é necessário analisar e descrever o processo e o protocolo de comunicação destes dispositivos, descrever aspectos acerca da utilização e funcionamento de sistemas de detecção de intrusão em redes de internet das coisas e descrever técnicas e algoritmos de aprendizado de máquina necessários para a detecção de ataques nas redes. Este trabalho propõe um sistema de segurança para redes IoT em tempo real sob protocolo MQTT utilizando técnicas de aprendizado de máquina para detecção de anomalias e criação de regras para os ataques detectados e um sistema de detecção de intrusão para utilizar as regras e detectar reincidências de ataques com base nelas. O sistema foi desenvolvido sob uma infraestrutura modular, permitindo a modificação e utilização de diferentes sistemas de detecção de intrusão e *brokers*. Como parte da solução, foi utilizada a plataforma nacional Dojot que possui o objetivo de integrar soluções de IoT, permitindo o gerenciamento dos dispositivos, processamento de dados e visualização dos alertas. Os experimentos foram realizados em duas redes distintas por meio de dados coletados a partir delas. Diante disso, verifica-se que as redes e dispositivos da internet das coisas podem comunicar-se de maneira insegura e a integração de algoritmos de aprendizado de máquina e mecanismos de segurança como os sistemas de detecção de intrusão pode ser feita para criar uma camada de segurança às redes e dispositivos.

Palavras-chave: Internet das Coisas. Sistemas de Detecção de Intrusão. MQTT. Aprendizado de Máquina.

ABSTRACT

In view of the growth of internet of things networks and devices and together with them several associated security flaws, research on machine learning techniques and intrusion detection systems in order to make secure networks and devices that communicate using the MQTT protocol. Therefore, it is necessary to analyze and describe the communication process and protocol of these devices, describe aspects about the use and operation of intrusion detection systems in internet networks of things and describe techniques and algorithms of machine learning necessary for the detection of attacks on networks. This work proposes a security system for real-time IoT networks under MQTT protocol using machine learning techniques to detect anomalies and create rules for detected attacks and an intrusion detection system to use the rules and detect recurrences of attacks based on them. The system was developed under a modular infrastructure, allowing the modification and use of different intrusion detection systems and extitbrokers. As part of the solution, the national Dojot platform was used, which has the objective of integrating IoT solutions, allowing device management, data processing and alert visualization. The experiments were carried out in two distinct networks through data collected from them. In view of this, it turns out that internet of things networks and devices can communicate inanely and the integration of machine learning algorithms and security mechanisms such as intrusion detection systems can be made to create a layer of security for networks and devices.

Keywords: Internet of Things. Intrusion Detection Systems. MQTT. Machine Learning.

LISTA DE FIGURAS

Figura 1 – Exemplo de *Isolation Forest* 31
Figura 2 – Exemplo dos efeitos de Inundação e Mascaramento no *Isolation Forest* . . . 32
Figura 3 – *Framework* do *Isolation Forest Algorithm for Streaming Data* (IFASD) . . . 32
Figura 4 – Arquitetura da solução proposta. 46
Figura 5 – Visualização Dojot. 52
Figura 6 – Matriz de Confusão *Isolation Forest* (IF) - Experimento MQTTset. 55
Figura 7 – Matriz de confusão IFASD - Experimento MQTTset. 56
Figura 8 – Matriz de Confusão IF - Experimento Proposta. 57
Figura 9 – Matriz de confusão IFASD - Experimento Proposta. 58
Figura 10 – Visualização dos dispositivos na Página Web. 58
Figura 11 – Alertas gerados na Página Web. 59
Figura 12 – Relação *contamination* e *Area Under Receiver Operating Characteristic Curve* (AUC ROC) 60
Figura 13 – Parâmetro *contamination* Rede MQTTset. 67
Figura 14 – Parâmetro *max_samples* Rede MQTTset.. 67

LISTA DE TABELAS

Tabela 1 – Métricas de Avaliação IF - Experimento MQTTset. 55
Tabela 2 – Métricas de Avaliação IFASD - Experimento MQTTset. 56
Tabela 3 – Métricas de Avaliação IF - Experimento Proposta. 57
Tabela 4 – Métricas de Avaliação IFASD - Experimento Proposta. 58

LISTA DE QUADROS

Quadro 1 – Exemplo de tópicos com uso do curinga '+' 25
Quadro 2 – Exemplo de tópicos com uso do curinga '#' 26
Quadro 3 – Exemplo de tópicos com uso único do curinga '#' 26
Quadro 4 – Quadro comparativo de trabalhos relacionados. 41
Quadro 5 – *Features* contidas nos *datasets*. 66

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Funcionamento do Agente no modo ambos. 47
Código-fonte 2 – Função de leitura e processamento de dados da rede. 48
Código-fonte 3 – Inicialização do agente. 49
Código-fonte 4 – Função de pontuação de anomalias do agente de ML. 50
Código-fonte 5 – Funcionamento do Agente SDI. 51
Código-fonte 6 – Funcionamento dos Dispositivos na Rede Proposta. 53

LISTA DE ABREVIATURAS E SIGLAS

IFASD	*Isolation Forest Algorithm for Streaming Data*
IF	*Isolation Fores*t
AUC ROC	*Area Under Receiver Operating Characteristic Curve*
IoT	*Internet Of Things*
ML	*Machine Learning*
MQTT	*Message Queuing Telemetry Transport*
SDI	Sistemas de Detecção de Intrusão
SPI	Sistemas de Prevenção de Intrusão
RFID	*Radio-frequency identification*
M2M	*Machine-to-machine*
SCADA	*Supervisory Control and Data Acquisition*
QoS	*Quality of Service*/Qualidade de Serviço
JSON	*JavaScript Object Notation*
XML	*Extensible Markup Language*
NIST	*National Institute of Standards and Technology*/Instituto Nacional de Padrões e Tecnologia
LR	*Logistic Regression*
NB	*Gaussian Naive Bayes*
k-NN	*k-Nearest, Neighbours*
SVM	*Support Vector Machine*
DT	*Decision Trees*
RF	*Random Forests*
SO GAAL	*Single-Objective Generative Adversarial Active Learning*
OCSVM	*One-Class Support Vector Machines*
LOF	*Local Outlier Factor*
HST	*Half Space Trees*
ANN	*Artificial Neural Networks*
LSTM	*Long Short-term Memory*
GRU	*Gated Recurrent Unit*

SUMÁRIO

1 INTRODUÇÃO 16
1.1 Objetivos 18
2 FUNDAMENTAÇÃO TEÓRICA 20
2.1 Redes, Dispositivos e Protocolos IoT 20
2.2 Sistemas de Detecção e Prevenção de Intrusão (SDIs/SPIs) 27
2.3 Algoritmos de Aprendizado de Máquina 29
3 TRABALHOS RELACIONADOS 34
3.1 Detecção de Ataques MQTT em redes *Internet Of Things* (IoT) 34
3.2 Detecção de ataques à redes IoT de baixo recurso 36
3.3 Detecção de Anomalias em redes IoT 38
3.4 Análise Comparativa e Considerações Finais 39
4 PROCEDIMENTOS METODOLÓGICOS 42
4.1 Coleta de Dados 42
4.2 Pré-processamento e Seleção de Dados 42
4.3 Definição e Treinamento do Modelo 43
4.4 Configuração do SDI e Criação de Regras 43
4.5 Validação e Análise dos Resultados 44
5 PROPOSTA 46
5.1 Agente Coletor 47
5.2 Agente de *Machine Learning* (ML) 48
5.3 Agente de Detecção de Intrusão (SDI) 50
5.4 *Broker* e Página Web 52
5.5 Dispositivos e Atacante 53
6 EXPERIMENTOS E RESULTADOS 54
6.1 Procedimentos Iniciais 54
6.2 Experimento Rede MQTTset 54
6.3 Experimento Rede Proposta 56
6.4 Discussão dos Resultados 59
7 CONCLUSÕES E TRABALHOS FUTUROS 61
REFERÊNCIAS 62

APÊNDICES 66
APÊNDICE A –QUADRO DE FEATURES CONTIDAS NOS DATASETS DOS EXPERIMENTOS 66
APÊNDICE B –SELEÇÃO DE PARÂMETROS ISOLATION FOREST EXPERIMENTO MQTTSET 67

1 INTRODUÇÃO

O surgimento e avanço da tecnologia nas áreas de eletrônica, sensoriamento, sistemas embarcados e outros, expandiu horizontes e expectativas acerca de quais sistemas do mundo físico poderiam ser integrados à internet. Esses sistemas ou dispositivos começaram a ser chamados de coisas, do inglês *things* dentro da Internet das Coisas, do inglês *Internet of Things - IoT*. Segundo o Gartner (2019), o número de dispositivos pode alcançar cerca de 25 bilhões em 2021.

Estimativas da Palo Alto Networks (2020) indicam que cerca de 30% das interfaces finais de comunicações das empresas à internet são dispositivos IoT. O que mostra o crescimento do uso deles nos meios de comunicação, porém juntamente com esse crescimento surgem diversos problemas, como o de privacidade, segurança dos dados e dispositivos, entre outros.

Um dos maiores exemplos de problemas causados por intermédio de dispositivos IoT aconteceu em 2016 (Beyond Trust, 2016), no qual diversos dispositivos IoT, Websites e serviços onlines como o Twitter, Netflix e Paypal saíram do ar quando ataques de negação de serviço distribuídos (DDoS) foram lançados por meio de uma falha de segurança em alguns sistemas IoT. Além deste, é possível citar casos em falhas de segurança em câmeras, babás-eletrônicas, sistemas de jeeps e entre outros dispositivos criados e inseridos na internet das coisas (ZD Net, 2015).

Pesquisas relacionadas à adoção de medidas de segurança para esses dispositivos e redes estão constantemente sendo feitas. Devido às limitações de recursos, eles tornam-se mais vulneráveis que computadores pessoais (Gendreau; Moorman, 2016), logo essas pesquisas não só visam a segurança, mas também a aplicação em redes com diversas limitações como baixo recurso, processamento, memória e outras.

A utilização de sistemas de detecção de intrusão SDIs (do inglês, *Intrusion Detection Systems* - IDSs) nessas redes vem sendo bastante estudada e implementada (BACE; MELL, 2001; SCARFONE; MELL, 2007; PATEL *et al.*, 2013; LIAO *et al.*, 2013; KASINATHAN *et al.*, 2013; Kasinathan *et al.*, 2013; Gendreau; Moorman, 2016). Eles possuem diferentes abordagens que permitem analisar dados na rede, dados nos dispositivos com o objetivo de detectar possíveis ataques. Por poderem possuir essa abordagem *agent-less*, ou seja, não necessária a aplicação diretamente nos dispositivos são bastante utilizados em redes IoT, tendo em vista os baixos recursos.

No entanto, esses sistemas muitas vezes se propõem a defender apenas ataques

conhecidos, como é exemplo os SDIs baseados em assinaturas, que utilizam dessas regras previamente conhecidas para detectar ataques que se assemelham a elas. Como alternativa para complementar essa deficiência, vários estudos têm sido feitos em cima da implementação de técnicas de aprendizado de máquina, do inglês *Machine Learning* (ML) para estudar os dados e ataques com o objetivo de criar novas regras ou aplicar novas técnicas na defesa desses ataques (HINDY *et al.*, 2020; SHALAGINOV *et al.*, 2019; ALAIZ-MORETON *et al.*, 2019).

Além disso, comumente soluções propostas com métodos de aprendizado de máquina, utilizam dados de ataques já conhecidos, o que pode muitas vezes tornar uma solução com viés para detecção apenas dos ataques treinados. Por esse motivo, alguns estudos se propõem a estudar o comportamento normal da rede ou de anomalias para ser possível a detecção de ataques que se diferenciam do comportamento esperado (ESKANDARI *et al.*, 2020; CIKLABAKKAL *et al.*, 2019; HASAN *et al.*, 2019), esses ataques não conhecidos são popularmente chamados de ataques de *zero-day*.

Contudo, é necessário destacar que as técnicas de aprendizado de máquina comumente utilizadas são feitas de maneira *offline* com o uso de dados previamente coletados de redes, ou seja, quando essas técnicas são implementadas em redes e elas mudam sua infraestrutura ou meios de comunicação, a técnica pode passar a não detectar mais possíveis ataques na nova infraestrutura, sendo necessário uma nova coleta de dados e um novo treinamento. Levando isso em consideração, vários estudiosos estão criando técnicas de aprendizado de máquina para treinamento *online* ou em tempo real via *streaming* de dados (ŽLIOBAITĖ *et al.*, 2013; HOENS *et al.*, 2012).

Igualmente, estudos propõem soluções, algoritmos e técnicas para detecção de anomalias por meio de treinamento *online* ou em tempo real via *streaming* de dados (AHMAD *et al.*, 2017; TAN *et al.*, 2011b; DING; FEI, 2013a).

Logo, diante das informações até aqui dispostas, a proposta deste trabalho é criar uma solução para segurança de redes IoT que utilizam o protocolo *Message Queuing Telemetry Transport* (MQTT) para comunicação, baseada em técnicas de aprendizado de máquina para detecção de anomalias em tempo real por meio de *streaming* de dados e criação automática de regras em um sistema de detecção de intrusão.

Os experimentos foram realizados em duas redes diferentes para validar o funcionamento da proposta em infraestruturas e comunicações distintas. Os dados dessas duas redes foram salvos em datasets, necessários para avaliar os resultados dos experimentos. O primeiro

foi obtido a partir de um trabalho de detecção de ataques em redes MQTT (VACCARI *et al.*, 2020), um segundo dataset foi coletado a partir de uma simulação de rede própria.

Os algoritmos *Isolation Forest* e *Isolation Forest Algorithm for Streaming Data* foram aplicados e avaliados por meio de métricas de avaliação na comparação dos resultados da aplicação da proposta em tempo real e offline.

Dessa forma, a elaboração e conclusão deste trabalho traz como contribuições: (i) A proposta de um sistema de detecção de anomalias em ambientes IoT de distintas infraestruturas, porém utilizando o protocolo MQTT; (ii) A avaliação de dois algoritmos de aprendizado de máquina, um comum em treinos offlines e outro adaptado e criado para streaming de dados; (iii) A avaliação da integração entre técnicas de aprendizado de máquina e sistemas de detecção de intrusão baseado em assinaturas.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é criar uma solução para a segurança de diferentes redes IoT que utilizam o protocolo MQTT em suas comunicações, sendo essa solução capaz de detectar ataques desconhecidos por meio da integração de técnicas de aprendizado de máquina e um sistema de detecção de intrusão baseado em assinaturas.

1.1.2 Objetivos Específicos

- Realizar e expor uma pesquisa bibliográfica acerca dos principais conceitos utilizados, além de apresentar trabalhos relacionados ao objetivo geral esperado.
- Desenvolver um sistema de detecção de anomalias em tempo real das comunicações MQTT em redes IoT por meio de técnicas de Aprendizado de Máquina.
- Comparar a performance de algoritmos de detecção anomalias offline e baseados em streaming de dados.
- Projetar e construir uma infraestrutura de agentes coletores, analisadores e detectores de intrusão para compor um sistema completo de detecção em redes IoT.
- Avaliar o sistema proposto em dados coletados e dados em tempo real de uma rede simulada.

Os capítulos posteriormente dispostos compõem a seguinte ordem: o Capítulo 2

expõe a fundamentação teórica necessária para embasar o trabalho proposto; o Capítulo 3 apresenta os trabalhos relacionados, citando seus estudos, problemas e objetivos e compara-os à proposta apresentada neste trabalho citando as suas principais características; o Capítulo 4 mostra os procedimentos metodológicos necessários para o desenvolvimento deste trabalho; o Capítulo 6 visa expor os resultados e as avaliações dos experimentos realizados, bem como as características técnicas de cada; por fim, o Capítulo 7 conclui este trabalho resumindo os resultados e trabalhos futuros relacionados.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma base de conhecimentos teóricos necessária para o entendimento deste trabalho. As seções iniciais apresentam conceitos acerca das redes, dispositivos e protocolos de comunicação utilizados na internet das coisas, focando em específico no protocolo MQTT, bastante utilizado por essas redes. Em seguida, são mostrados os Sistemas de Detecção de Intrusão (SDI) e os Sistemas de Prevenção de Intrusão (SPI), citando suas diferenças, usos e o motivo para não serem totalmente efetivos na implementação de segurança nas redes. Por fim, são descritos algoritmos de aprendizado de máquina utilizados em conjunto com os SDI e SPI para uma abordagem mais completa de segurança aos dispositivos da internet das coisas.

2.1 Redes, Dispositivos e Protocolos IoT

O termo internet das coisas, tradução de IoT, foi utilizado pela primeira vez em uma apresentação sobre a integração do *Radio-frequency identification* (RFID) com cadeias de suprimentos (ASHTON *et al.*, 2009). Com o tempo, diversas definições surgiram e começaram a ser adaptadas e utilizadas. Segundo o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) (2015) "A IoT é uma rede que conecta "coisas"unicamente identificáveis, estas possuem sensores/atuadores e potenciais capacidades de programabilidade. Através dessas características é possível coletar informações e mudar o estado da "coisa"de qualquer lugar e a qualquer momento".

Os sensores são responsáveis por coletar as informações do mundo físico para serem processadas e utilizadas pelos atuadores para afetar o mundo físico (Moyer, 2019). Devido a baixa capacidade energética e de processamento, os dispositivos IoT necessitam transferir responsabilidades para camadas superiores (Li *et al.*, 2018; Yu *et al.*, 2018), logo é necessário uma estruturação da solução.

Uma boa implementação de uma solução IoT dá-se ao atender a necessidade de comunicação em tempo real, dinâmica demanda por recurso, disponibilidade das aplicações, privacidade dos dados, consumo eficiente de energia, execução das aplicações próximas ao usuário final e acesso a sistemas de nuvens abertos e interoperáveis (MADAKAM *et al.*, 2015).

Dessa maneira, é possível dividir a solução entre três grandes componentes (*Hardware*, *Middleware* e Visualização) permitindo a modularização da estrutura e uma maior facilidade de implementação. O *hardware*, composto de sensores e atuadores comunica-se com o

middleware, responsável por salvar, computar e analisar os dados, com o objetivo de torná-los apresentáveis ao usuário final por meio da visualização (GUBBI *et al.*, 2013).

A intercomunicação entre esses componentes se dá por meio dos protocolos de comunicação (Al-Sarawi *et al.*, 2017). Dentre eles alguns dos mais comuns e utilizados[1] são:

- **IEEE 802.11, Wi-Fi:** Possui versões utilizando frequências de 2.4 GHz e 5 GHz com alcance de aproximadamente 50 metros podendo chegar a 100 metros. Em sua última versão suas taxas de dados podem chegar a 5Gbps (Cisco Systems, Inc., 2020). Tem uma desvantagem de possuir o consumo energético alto em algumas versões, impossibilitando a utilização do padrão por muitas aplicações IoT.
- **Bluetooth:** Utiliza frequência de 2.4 GHz com alcance de 50 metros podendo chegar a 150 metros aproximadamente. Com uma taxa de transferência de até 1 Mbps, é utilizado para enviar baixas quantidades de dados através de produtos pessoais, como *smartwatches*. Sua versão de baixa energia *Bluetooth Low-Energy* (BLE) torna esse padrão muito utilizado por produtos IoT.
- **Zigbee:** Criado pela ZigBee Alliance baseado no padrão IEEE802.15.4 utiliza frequência de 2.4 GHz com alcance de 10 metros podendo chegar a 100 metros. Possui uma taxa de transferência baixa de até 250Kbps, porém opera com baixa energia, alta segurança, robustez e alta escalabilidade. Foi desenvolvido para ser utilizado pela indústria no contexto de baixo requerimento de energia e rede consumidora.
- **LoRaWAN:** Pode utilizar várias frequências com alcance de aproximadamente 2.5km para ambientes urbanos e 15km para ambientes suburbanos. Com uma taxa de transferência de 0.3 até 50Kbps é um protocolo de longa distância otimizado para um baixo consumo de energia e suporte de grandes redes com diversos dispositivos. Um exemplo de uso é na implementação de ruas com iluminação inteligente, as luzes das ruas seriam conectadas em um *LoRa gateway* utilizando o protocolo.
- **SigFox:** Utiliza frequência de 900MHz com alcance de aproximadamente 30 a 50km para ambientes rurais e 3 a 10km para ambientes urbanos. Com taxas de 10 até 1000bps, foi desenvolvido para cobrir grandes áreas com a utilização de aplicações *Machine-to-machine* (M2M).

Como este trabalho tem como foco a segurança de redes e dispositivos IoT que comunicam-se através do protocolo MQTT, a seção seguinte detalha o seu funcionamento para

[1] https://hashstudioz.com/blog/top-iot-communication-protocols-2020/

uma melhor base de entendimento acerca dos passos de leitura e análise dos dados utilizados.

2.1.1 Protocolo MQTT

O protocolo foi inventado em 1999 por Andy Stanford-Clark (IBM) e Arlen Nipper (Cirrus Link) para ter uma perda mínima de bateria e uma mínima largura de banda, com objetivo de conectar óleo-dutos via satélites. Por possuir uma implementação simples, qualidade de serviço na transferência de dado, mínima e eficiente largura de banda, ser agnóstico aos dados e sensível a sessões contínuas, seu foco usual era em sistemas embarcados proprietários, porém foi transferido à casos de usos da internet das coisas (HiveMQ, 2015a).

É um protocolo do tipo cliente/servidor *publish/subscribe* utilizado para o transporte de mensagens. Por ser leve, aberto, simples e fácil de implementar, é utilizado em vários ambientes de comunicação M2M e IoT onde os dispositivos possuem recursos limitados (OASIS, 2014). Por exemplo, é usado na comunicação de sensores à *brokers* via links de satélites, em sistemas *Supervisory Control and Data Acquisition* (SCADA), em sistemas de cuidados médicos e na automação de casas e dispositivos pequenos (OASIS, 2016).

Diante dessas informações, é necessário entender o funcionamento técnico e detalhado do protocolo. As próximas seções abordarão os principais conceitos acerca da estrutura e modo de comunicação do protocolo.

2.1.1.1 Clientes e Brokers/Servidores

Clientes usando o protocolo MQTT podem ser publicadores e inscritos. Em relação a dispositivos podem ser desde um micro controlador a um servidor que roda uma biblioteca *MQTT* e se conecta à um servidor/*broker* na rede. Pelo fato do *MQTT* ser um protocolo leve e de baixo custo de implementação, ele pode ser implementado em dispositivos com baixos recursos, o que o torna muito importante para redes IoT.

Uma das partes mais importantes dos sistemas de publicação e inscrição é o servidor/*broker*, responsável por receber todas as conexões e mensagens de clientes, filtrá-las e enviar para os respectivos clientes inscritos. É papel do servidor também manter sessões de clientes, autenticar e autorizar os clientes e processar o sistema de garantia de entrega e envio de mensagens por meio dos níveis de qualidade de serviço.

A conexão entre um cliente e um servidor é feita através do envio de uma mensagem do tipo *CONNECT* pelo cliente e o envio de um reconhecimento *CONNACK* pelo servidor. A

conexão é estabelecida até que o cliente envie uma mensagem de *DISCONNECT* ou a conexão quebre.

A mensagem de conexão pode possuir os seguintes campos: *ClientId*, *Clean Session*, *Username/Password*, *Will Message* e *Keep Alive*.

- ***ClientId*:** Campo utilizado para identificar o cliente de conexão ao *broker*, deve ser único ou se vazio deve ter a *flag clean session* ativa para o broker aceitar a conexão.
- ***Clean Session*:** Campo utilizado para avisar ao *broker* que o cliente vai realizar uma sessão persistente ou não. Na sessão persistente o *broker* salva todas as inscrições e todas as mensagens perdidas pelo cliente se ele se inscreveu com *Quality of Service/Qualidade de Serviço (QoS)* 1 ou 2. Se a sessão não for persistente o *broker* apaga todas as informações após a desconexão e não salva mensagens.
- ***Username/Password*:** Campo utilizado para autenticação ao broker se ativa. Os dados são passados em texto plano caso o *broker* não tenha implementação de encriptação ou TLS ativo.
- ***Will Message*:** Campo utilizado para notificar outros clientes quando este se desconecta do broker, utilizado em conjunto com um tópico para a mensagem ser enviada.
- ***Keep Alive*:** Campo utilizado para definir a quantidade de tempo máxima que o cliente pode ficar sem enviar uma mensagem para o servidor e vice-versa, a conexão é mantida utilizando mensagens de reconhecimento chamadas PING requests.

Já a mensagem de reconhecimento de conexão enviada pelo broker possui os seguintes campos: *Session Present* e *Return Code*.

- ***Session Present*:** Campo utilizado para avisar ao cliente se ele já possui uma sessão persistente anteriormente ativada, dessa forma o *broker* pode utilizar dados salvos por um mesmo cliente em sessões anteriores.
- ***Return Code*:** Campo utilizado para retornar se a conexão foi realizada com sucesso ou não. Pode assumir valores: (0) se a conexão for feita; (1); se a conexão for recusada por possuir a versão de protocolo não aceita; (2) se o identificador do cliente não for válido; (3) se o servidor não está disponível; (4) se o cliente enviou um usuário ou senha errados; (5) se a conexão não foi autorizada.

2.1.1.2 Publish e Subscribe

O conceito de *publishing* e *subscribing*[2] é uma alternativa a arquitetura tradicional de cliente/servidor na qual o cliente conhece o servidor e se comunica diretamente com ele. Com o modelo de pub/sub o componente que envia a mensagem (*publisher*) é separado do componente que recebe a mensagem (*subscriber*) de modo que eles nunca contactem-se diretamente. Essa comunicação é feita através de um terceiro componente (o *broker*/*server*[3]). O servidor tem a função de receber e filtrar todas as mensagens e distribuí-las corretamente para os *subscribers* (HiveMQ, 2015b).

Esse modelo permite: o desacoplamento de espaço, *publishers* e *subscribers* precisam conhecer apenas o servidor e não uns aos outros; O desacoplamento de tempo, *publishers* e *subscribers* não precisam rodar ao mesmo tempo; O desacoplamento da sincronização, as operações dos componentes não precisam ser interrompidas durante a ação. Dessa maneira a escalabilidade torna-se maior que a do modelo tradicional, visto que as operações de filtragem, processamento e envio das mensagens no servidor podem ser paralelizadas (HiveMQ, 2015b).

No protocolo MQTT o processo de publicação é feito pelo cliente após conectar-se ao servidor. A mensagem é enviada contendo um cabeçalho e seu conteúdo, este de qualquer tipo (*JavaScript Object Notation* (JSON), *Extensible Markup Language* (XML), texto plano e etc), visto que o protocolo é agnóstico ao dado transferido. Uma mensagem do tipo *PUBLISH* pode possuir os seguintes campos: *packetId*, *topicName*, *QoS*, *retainFlag*, *payload* e *dupFlag*.

- ***packetId*:** Identifica unicamente cada mensagem enviada. Necessário para transmissão de pacotes de reconhecimento (*ack packets*) usando nível de QoS acima de 0.
- ***topicName*:** Representa o canal por onde a mensagem vai ser enviada. Detalhes na seção 2.1.1.3.
- ***QoS*:** Possui três níveis (0, 1 e 2) que determinam mecanismos de garantia de envio e recebimento da mensagem. No nível 0 as mensagens são enviadas apenas uma vez, sem garantia de entrega ou reenvio. No nível 1 a mensagem é recebida pelo menos uma vez por meio de reconhecimentos e reenvios. No nível 2 a mensagem é entregue exatamente uma vez por meio de reconhecimentos do cliente e servidor através de um *four-way handshake*.
- ***retainFlag*:** Define se essa mensagem deve ficar retida no servidor no respectivo tópico

[2] Os termos *publish/publisher/publishing* e *subscribe/subscriber/subscribing* não foram traduzidos por serem amplamente utilizados na literatura.

[3] O termo *broker* foi substituído por *server*, logo é possível encontrar os dois termos sendo utilizados (MQTT Community Wiki, 2020).

até um cliente solicitar a última mensagem retida no tópico definido.

- ***payload*:** Conteúdo da mensagem a ser enviada.
- ***dupFlag*:** Indica se a mensagem está sendo reenviada caso o cliente não receba um reconhecimento da mensagem original. Usada apenas em níveis de QoS acima de 0.

No processo de inscrição o cliente envia uma mensagem *SUBSCRIBE* contendo o identificador do pacote (*packetId*) e a lista de tópicos a se inscrever e seus respectivos níveis de QoS. O servidor então envia uma mensagem de reconhecimento contendo o identificador do mesmo pacote e para cada tópico inscrito retorna um código de resultado. Os códigos de resultados podem ser: (0) para sucesso com QoS 0; (1) para sucesso com QoS 1; (2) para sucesso com QoS 2; (128) para falha na inscrição.

Para revogar sua inscrição o cliente pode enviar uma mensagem do tipo *UNSUBSCRIBE* contendo o identificador do pacote e os respectivos tópicos (sem o QoS) que deseja não receber mensagens. A mensagem de reconhecimento da revogação retorna o identificador do pacote contendo a mensagem de *UNSUBSCRIBE*.

2.1.1.3 Tópicos

No MQTT os tópicos são como um canal por onde as mensagens são passadas, logo os *publishers* enviam suas mensagens a tópicos específicos e os *subscribers* se inscrevem neles via *broker* para recebê-las. Cada tópico possui um nome (diferenciando letras maiúsculas e minúsculas) no formato *UTF-8* e é composto por um ou mais níveis separados por uma barra '/' chamada de *topic level separator* (Akamai Technologies, 2020; HiveMQ, 2019).

Um cliente *subscriber* pode se inscrever a mais de um tópico utilizando um tópico do tipo filtro composto por um ou mais caracteres "curingas". Esses caracteres podem ser de nível único ou de multi-níveis.

O curinga '+' é de nível único e utilizado para ser substituído por qualquer outro nível de tópico.

Quadro 1 – Exemplo de tópicos com uso do curinga '+'

Válido	Tópico Exemplo	Tópico Alvo
Sim	casa/**quarto**/temperatura	casa/+/temperatura
Sim	casa/**sala**/temperatura	casa/+/temperatura
Não	casa/sala/**umidade**	casa/+/temperatura
Não	**quintal**/grade/temperatura	casa/+/temperatura

Fonte: Elaborado pelo autor.

O curinga ’#’ é de multi-níveis e é utilizado no final do tópico para representar quaisquer níveis de tópicos após o tópico padrão anterior ao caractere ’#’.

Quadro 2 – Exemplo de tópicos com uso do curinga ’#’

Válido	Tópico Exemplo	Tópico Alvo
Sim	casa/**quarto/temperatura**	casa/#
Sim	casa/**sala/temperatura**	casa/#
Sim	casa/**sala/umidade**	casa/#
Não	**quintal**/grade/temperatura	casa/#

Fonte: Elaborado pelo autor.

Se for especificado apenas o curinga "#"como tópico, o cliente receberá todas as mensagens recebidas pelo servidor, até mesmo as relacionadas aos tópicos de configuração do *broker* citadas na próxima seção.

Quadro 3 – Exemplo de tópicos com uso único do curinga ’#’

Válido	Tópico Exemplo	Tópico Alvo
Sim	**casa/quarto/temperatura**	#
Sim	**casa/sala/temperatura**	#
Sim	**casa/sala/umidade**	#
Sim	**$SYS/broker/clients/connected**	#
Sim	**$SYS/broker/clients/total**	#

Fonte: Elaborado pelo autor.

2.1.1.3.1 *Tópicos de Configuração $*

Tópicos iniciando com $ são reservados para apresentar estatísticas internas do servidor. Esses tópicos não estão incluídos nas mensagens recebidas do tópico de curinga "#"e clientes não podem publicar nesses tópicos. Alguns exemplos de tópicos de configuração do sistema são:

- **$SYS/broker/clients/connected**: Lista a quantidade de clientes conectados ao servido.
- **$SYS/broker/clients/disconnected**: Lista a quantidade de clientes desconectados do servidor.
- **$SYS/broker/clients/total**: Lista a quantidade total de clientes.
- **$SYS/broker/messages/sent**: Lista a quantidade de mensagens enviadas pelo servidor.
- **$SYS/broker/uptime**: Mostra o tempo total de atividade do servidor.

2.2 Sistemas de Detecção e Prevenção de Intrusão (SDIs/SPIs)

Segundo o *National Institute of Standards and Technology*/Instituto Nacional de Padrões e Tecnologia (NIST) (BACE; MELL, 2001; SCARFONE; MELL, 2007) a detecção de intrusão é o processo de monitoramento e análise dos eventos que ocorrem em sistemas ou redes de computadores com o objetivo de encontrar sinais de possíveis intrusões ou incidentes. Essas intrusões podem ser definidas como tentativas de violar mecanismos de segurança ou comprometer os três pilares da segurança de um sistema (confidencialidade, integridade e disponibilidade).

Os sistemas de detecção de intrusão (SDIs) são sistemas de *softwares* ou *hardwares* que automatizam esse processo de detecção de intrusão (BACE; MELL, 2001). Já os sistemas de prevenção de intrusão (SPIs) se diferenciam dos SDIs por terem a capacidade de responder às ameaças detectadas, podendo prevenir que causem danos aos sistemas (SCARFONE; MELL, 2007).

Os três principais componentes que definem o funcionamento de um SDI/SPI são: Ambiente monitorado, Métodos de Detecção e Análise e os Métodos de Resposta.

O ambiente monitorado pode ser definido por três tipos: *Network-based*, *Host-based*, *Application-based* (PATEL *et al.*, 2013; BACE; MELL, 2001). Alguns autores citam outros ambientes como *Wireless*, *Network Behavior Analysis (NBA)* (SCARFONE; MELL, 2007) que estendem os ambientes principais.

***Network-based*:** Monitora o tráfego da rede, segmentos da rede ou dispositivos e analisa a atividade dos protocolos de rede e aplicação para identificar atividades suspeitas.

***Host-based*:** Monitora o comportamento e estado de um sistema específico. Neste ambiente são analisadas as operações de uso dos recursos do sistema monitorado.

***Application-based*:** Monitora dentro do ambiente uma aplicação específica e seus eventos. A análise é feita em cima das saídas de dados da aplicação monitorada, por exemplo, os *logs*.

É importante ressaltar que a combinação das abordagens e a criação de *Mixed IDSs* (LIAO *et al.*, 2013) é uma maneira muito utilizada para aumentar a performance dos sistemas.

Os métodos de detecção e análise fazem parte do principal componente de um SDI/SPI, são eles que irão detectar ou predizer as intrusões através dos dados resultados do monitoramento do sistema, dispositivo ou rede. Existem vários métodos propostos, porém os mais utilizados estão definidos abaixo (PATEL *et al.*, 2013; BACE; MELL, 2001).

- ***Misuse or Signature detection*:** Esse método utiliza padrões conhecidos de comportamentos intrusivos, chamados de assinatura, para predizer e detectar tentativas similares de intrusões.
- ***Anomaly detection*:** Esse método tenta descobrir padrões anormais de comportamentos baseado tanto na diferença das atividades consideradas normais, quanto em padrões de intrusões já conhecidos. As maiores dificuldades em implementar este método, é que se o método não estiver bem implementado, a taxa de falsos positivos gerados pode ser muito alta, visto que muitas vezes o sistema pode detectar padrões normais como padrões de intrusões. Esse método de detecção possui diversas categorias de implementação estudadas (HOANG *et al.*, 2009; ELSHOUSH; OSMAN, 2011), dentre elas algumas são importante destacar como:
  - ***Statistical*:** Com esta técnica o sistema observa a atividade de alguns dados como o uso de CPU dos dispositivos ou número de conexões TCP, criando distribuições e analisando os comportamentos, caso baseado nesses dados de entrada o comportamento da rede se diferencie do esperado, uma anomalia pode ser detectada.
  - ***Machine learning and Data mining based*:** Por meio dessa técnica o sistema através de dados previamente adquiridos, de maneira similar à categoria de técnicas estatísticas, pode analisar, otimizar e aprender os comportamentos da rede. Com novas informações ou alterações nas estruturas dos dados analisados, o sistema pode melhorar analisando e classificando novas intrusões ou comportamentos estranhos.

Os métodos de resposta representam o resultado dos componentes anteriores. Após os passos de monitoramento, detecção e análise, o resultado pode ser dado de forma ativa ou passiva (BACE; MELL, 2001; PATEL *et al.*, 2013), os principais meios utilizados estão citados abaixo.

- ***Active response:*** Os sistemas que possuem respostas ativas, realizam ações quando determinados tipos de intrusão são detectados. Essas ações podem ser implementadas de acordo com diferentes tipos de ambientes e objetivos. Segundo Bace e Mell (2001) existem três categorias de respostas ativas, estas detalhadas abaixo.
  - ***Coletar informações adicionais:*** Após a suspeita de uma intrusão, a ação tomada pelo sistema com esse método é realizar uma coleta de informações do comportamento suspeito com o objetivo de resolver a detecção do ataque.
  - ***Modificar o ambiente:*** O sistema ao tomar essa ação como resposta à intrusão, tenta

prevenir o ataque subsequente por meio de bloqueios na rede, injeção de pacotes para impedir o acesso do atacante e etc.

  - ***Executar ações contra o intruso:*** Esta ação, apesar de correr o risco de ser ilegal, ocorre quando o sistema tenta atacar o intruso para tentar obter informações. É utilizada em pretextos de uso de sistemas críticos e com atribuições legais anteriormente obtidas.

- ***Passive response:*** Nas respostas passivas o sistema apenas provê a informação da detecção para os usuários e administradores do sistema. Essa informação contém os dados obtidos da tentativa de intrusão e pode ser repassada por meio de notificações, alarmes, e-mails. Alguns sistemas preferem a utilização das respostas passivas, por possuírem uma taxa alta de falsos positivos que podem ocasionar instabilidades nas redes ao se utilizar respostas ativas.

2.3 Algoritmos de Aprendizado de Máquina

O aprendizado de máquina é um ramo que foca na construção de aplicações que aprendem com dados e melhoram a precisão de seu objetivo ao longo do tempo sem serem manualmente programadas, mas automaticamente por meio de algoritmos treinados para encontrar padrões nesses dados e criar previsões e/ou tomar decisões (IBM, 2020).

Segundo Dietterich (2003) uma das áreas da aplicação de algoritmos de aprendizado de máquina é no problema de encontrar padrões em dados, necessários para aplicação na melhoria do *marketing* das empresas, na detecção de fraudes e predição de problemas futuros. Além disso, é importante ressaltar a existência de vários tipos de aprendizado de máquina, dentre eles o Supervisionado e Não Supervisionado.

No aprendizado supervisionado o sistema conta com os dados e o resultado da interpretação deles, em outras palavras, dados rotulados, tendo como objetivo predizer novos valores, gerar interpretações em cima de novos dados com base no treinamento anteriormente feito, como predição de riscos e vendas (AITUDE, 2020). Exemplos de algoritmos de aprendizado supervisionado são: *Logistic Regression* (LR), *Gaussian Naive Bayes* (NB), *k-Nearest, Neighbours* (k-NN), *Support Vector Machine* (SVM), *Decision Trees* (DT), *Random Forests* (RF).

No aprendizado não supervisionado o sistema conta apenas com os dados não rotulados. Seu objetivo é explorar os padrões contidos naqueles dados, agrupar os dados, encontrar

deviações como a detecção de anomalias ou alguma informação intrínseca que resulte em uma saída ou interpretação objetiva. Exemplos de algoritmos de aprendizado não supervisionado são: *Single-Objective Generative Adversarial Active Learning* (SO GAAL), IF, *One-Class Support Vector Machines* (OCSVM), *Local Outlier Factor* (LOF).

Geralmente esses métodos tradicionais de aprendizado de máquina funcionam de maneira *offline*, ou de aprendizado em *batch*, no qual o algoritmo é treinado em todo os dados ao mesmo tempo e é implantado logo em seguida para realizar suas tomadas de decisões ou predições, sem realizar qualquer tipo de atualização depois ou ser re-treinado, tendo em vista os custos para isso (HOI *et al.*, 2018). Por esse motivo, essas abordagens tornam-se pouco escaláveis para aplicações do mundo real.

Tendo em vista esse problema, estudos buscam formas de resolvê-lo, uma delas é o aprendizado *online* (BLUM, 1998), método que treina o algoritmo em ordem sequencial, para cada nova instância de dados. Assim, não sendo necessários custos adicionais de re-treino, tornando essa abordagem mais escalável para tarefas de análise do mundo real.

Diversos algoritmos vêm sendo construídos para este fim, tais como: *Half Space Trees* (HST) (TAN *et al.*, 2011a), *LSEARCH* (O'Callaghan *et al.*, 2002). Outros algoritmos que em seu estado de arte são *offline*, ou treinam em *batch* e podem ser utilizados no aprendizado *online* como: IFASD, adaptação do IF ou OCSVM (Bhat; Singh, 2020).

Com o objetivo inicial de fazer uma avaliação entre os dois tipos de aprendizado expostos nesta seção (*online* e *offline*), foram escolhidos algoritmos que possuem suas versões nas duas abordagens, apesar de existirem diversos algoritmos para o aprendizado *online* e outros para o aprendizado *offline*, são poucos os que podem ser utilizados em ambas. Por esse motivo essa pesquisa utiliza os algoritmos IF e IFASD para comparação de suas performances na detecção de anomalias, um sendo utilizado no aprendizado *offline* e o outro no aprendizado *online* respectivamente, porém mantendo, ambos o mesmo algoritmo.

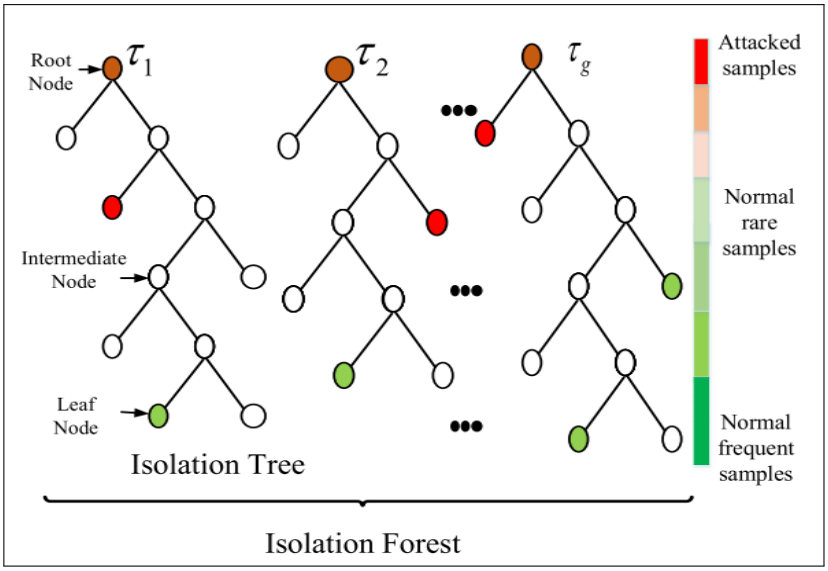
2.3.1 Isolation Forest

O *Isolation Forest* (Liu *et al.*, 2008) é um algoritmo de aprendizado não supervisionado para detecção de anomalias, a base do seu funcionamento é isolar as anomalias em vez de criar padrões normais das instâncias, como seguem as abordagens mais comuns de detecção de anomalias. O método proposto parte de duas propriedades quantitativas: (1) Existem poucas instâncias anômalas nos dados lidos, (2) Esses dados possuem valores muito diferentes dos dados

normais. Sendo possível assim, ser mais fácil isolar os dados anômalos.

O método usa os dados para construir uma floresta com árvores isoladas dividindo as instâncias randomicamente, escolhendo um valor entre o máximo e o mínimo do valor selecionado, até que todas sejam isoladas (Ahmed *et al.*, 2019), então as anomalias são os dados que possuem os menores caminhos em cada sub-árvore. Logo, dada a suscetibilidade da isolação as anomalias ficam próximas à raiz das árvores como demonstra a Figura 1.

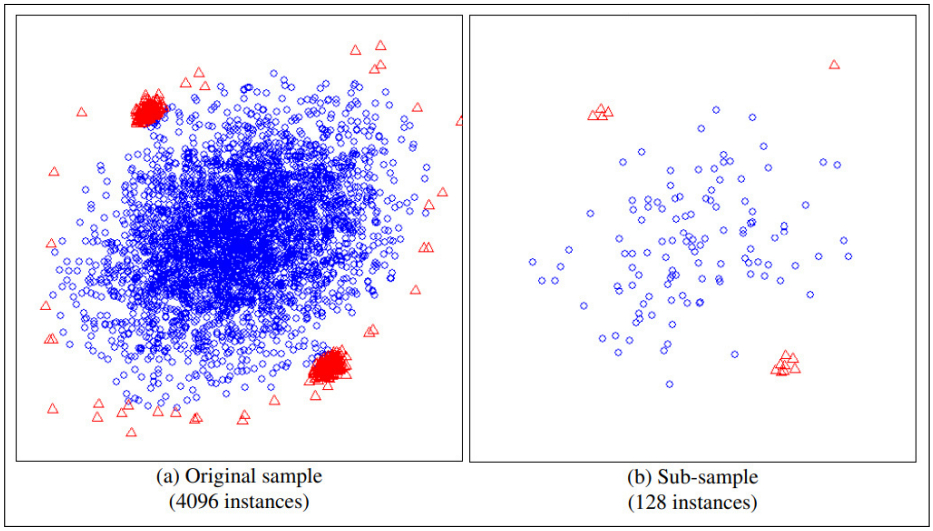
Figura 1 – Exemplo de *Isolation Forest*



Fonte: Ahmed *et al.* (2019)

As principais características desse algoritmo são permitir que pequenas amostras sejam utilizadas sem perder a performance, dado que não são necessárias grandes árvores para isolar dados normais e pequenas amostras podem reduzir efeitos de inundação (rotular dados normais como anomalias) e mascaramento (rotular dados anômalos como normais) como demonstra a Figura 2. Também não utiliza medidas de distância ou densidade, eliminando custos computacionais e requerendo pouco tempo e memória para execução.

Figura 2 – Exemplo dos efeitos de Inundação e Mascaramento no *Isolation Forest*



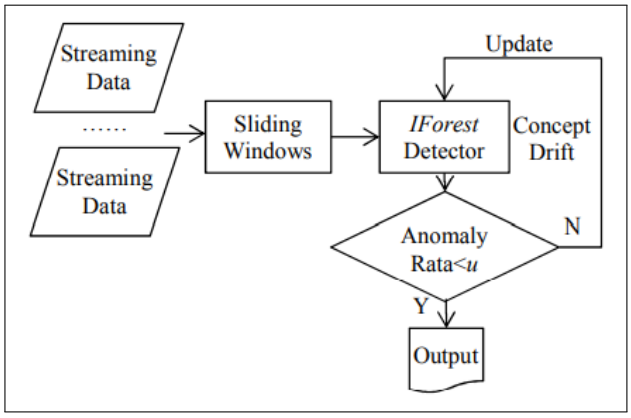
Fonte: Adaptado de Liu *et al.* (2008)

2.3.2 Isolation Forest Algorithm for Streaming Data

O *Isolation Forest Algorithm for Streaming Data* (DING; FEI, 2013b) é um algoritmo criado para resolver problemas comuns de aprendizado *offline* aqui apresentados. Adaptado do IF, tem como objetivo a detecção de anomalias trazendo todas as capacidades que o IF implementa e adicionando a capacidade de detecção *online* ou em *streaming* de dados.

O seu funcionamento é baseado em janelas deslizantes como apresentado na Figura 3. Os dados são inseridos dentro de uma janela deslizante com um tamanho pré-definido. Com esses dados um detector inicial é treinado pelo algoritmo IF. Na fase de testes, todas as instâncias dentro de cada janela deslizante são examinadas pelo detector e são rotuladas como anomalias ou não.

Figura 3 – *Framework* do IFASD



Fonte: Ding e Fei (2013b)

Com os dados rotulados, se a taxa de anomalia dentro da janela deslizante for menor que um limite pré-estabelecido u, então o algoritmo ainda está apto a continuar seu funcionamento. Já se a taxa de anomalia for maior que o limite, significa que o algoritmo está detectando mais anomalias que o esperado e o problema da diferença de domínio pode estar acontecendo, nesse caso o algoritmo precisa ser descartado e um novo precisa ser atualizado e re-treinado com os novos dados.

3 TRABALHOS RELACIONADOS

Esta seção provê de forma resumida estudos relacionados à detecção de intrusão em ambientes IoT. Inicialmente, como foco principal, são apresentados trabalhos que estudam e propõem métodos para detecção de ataques utilizando o protocolo MQTT em redes IoT. De maneira secundária, são apresentados trabalhos relacionados à detecção de ataques focados em redes IoT de baixo recurso e trabalhos com foco em detecção de anomalias nessas redes. Ao final é feita uma discussão comparativa por meio de um quadro, entre as principais características deste trabalho e os trabalhos relacionados.

3.1 Detecção de Ataques MQTT em redes IoT

Foram selecionados três trabalhos que têm como foco principal abordar técnicas para detecção de ataques sob protocolo MQTT em redes IoT. Apesar de Hindy *et al.* (2020), Ciklabakkal *et al.* (2019) e Alaiz-Moreton *et al.* (2019) possuírem a mesma motivação de detectar ataques MQTT à brokers nas redes, eles utilizam ambientes, dados, ataques, algoritmos e métricas de avaliação diferentes.

Hindy *et al.* (2020) simula ataques de força bruta e scans na rede e utiliza seis técnicas e cinco métricas de avaliação para cada algoritmo. Já Ciklabakkal *et al.* (2019) foca em detectar ataques anômalos à brokers que são dispositivos IoT, logo o estudo além de avaliar a eficácia das técnicas, analisa também a eficiência delas. Seu estudo utiliza seis técnicas e duas métricas de avaliação para cada. Por fim, Alaiz-Moreton *et al.* (2019) tem como foco classificar e detectar múltiplos tipos de ataques específicos para MQTT por meio de três técnicas avaliadas por cerca de cinco métricas.

De maneira detalhada, a descrição e principais características como ambiente, ataques, algoritmos, métricas, resultados e objetivos de cada trabalho estão descritas a seguir.

3.1.1 Machine Learning Based IoT Intrusion Detection System: An MQTT Case Study

Hindy *et al.* (2020) estudam e avaliam a performance de seis técnicas de machine learning em um conjunto de dados de ataques baseados no protocolo MQTT. Inicialmente a base de dados é criada por meio de um ambiente com diversos dispositivos simulados, um *broker* centralizado e um atacante. Durante a simulação os dispositivos geram os dados normais de comunicação e o atacante realiza quatro ataques, sendo três genéricos (scan agressivo, scan udp

e brute force ssh) e um específico para MQTT (brute force MQTT).

Com foco na classificação de dados normais e ataques, o estudo analisa seis técnicas: Logistic Regression (LR), Gaussian Naive Bayes (NB), k-Nearest, Neighbours (k-NN) , Support Vector Machine (SVM) , Decision Trees (DT) and Random Forests (RF). As métricas utilizadas para avaliar os resultados foram: Overall Accuracy, Precision, Recall e F1-score.

O estudo visou capturar *features* baseadas no pacote MQTT, de rede de fluxo único e de rede de fluxo duplo, para então enfatizar que os dados baseados no pacote MQTT são suficientes para detectar os três ataques genéricos, porém não para detectar o ataque MQTT visto que há uma dificuldade em diferenciar dados MQTT normais de dados do ataque MQTT. Também foi resultado a importância das *features* de fluxo único e duplo para a detecção do ataque MQTT.

3.1.2 ARTEMIS: An Intrusion Detection System for MQTT Attacks in Internet of Things

Ciklabakkal *et al.* (2019) propõem um sistema de detecção de intrusão para ataques MQTT em redes IoT por meio de algoritmos e técnicas de ML. A estrutura protótipo utilizada para coleta dos dados é composta de sensores conectados a um dispositivo de *gateway* (Raspberry Pi) que envia os dados para um *broker*. Os pacotes de ataques foram gerados pelo *gateway* utilizando a ferramenta MQTT malaria *tool*. O *broker* captura esses pacotes e por meio das técnicas, treina e prediz se aquele pacote contém ataque ou não. Os resultados obtidos são enviados para o usuário por meio de uma aplicação interativa. Segundo os autores a abordagem centralizada no *broker* foi utilizada para que as atividades computacionais sejam minimizadas no *gateway* (Raspberry Pi).

Os dados utilizados para o treinamento consistem de informações a cerca dos protocolos TCP, MQTT e IP contidos nos pacotes. As técnicas avaliadas nesse sistema foram: Autoencoder, Single-Objective Generative Adversarial Active Learning (SO GAAL), Random Forest, Isolation Forest, One-Class Support Vector Machines (OCSVM), and Kmeans Clustering. As métricas de avaliação de performance utilizadas foram: ROC AUC Scores e Accuracy Scores.

O estudo teve como objetivo demonstrar os resultados obtidos por meio de um modelo não treinado com ataques previamente conhecidos, visto que a ferramenta utilizada gerava ataques randomizados, além de gerar o *dataset* contendo dados normais e de ataques MQTT e expor a avaliação de performance dos algoritmos utilizados no objetivo de detecção de ameaças MQTT.

3.1.3 Multiclass Classification Procedure for Detecting Attacks on MQTT-IoT Protocol

Alaiz-Moreton *et al.* (2019) propõem um procedimento de detecção de anomalias sob protocolo MQTT em redes IoT utilizando técnicas de *machine learning*. Para o procedimento de classificação eles construíram uma base de dados utilizando pacotes MQTT. Esses pacotes foram capturados por meio de uma rede de topologia estrela com um *broker* central recebendo os dados de dispositivos, sensores e um atacante. Foram três tipos de ataques gerados pelo atacante, sendo eles: ataque de negação de serviço (DoS) utilizando o programa MQTT Malaria, ataque de Man in the Middle (MitM) interceptando as mensagens entre os sensores e *broker* para modificar os dados e ataques de intrusão ao *broker* por meio do próprio protocolo MQTT.

Os dados gerados pelo ambiente resultaram em um *dataset* para cada ataque, contendo os dados do ataque específico e os dados normais. Antes da classificação, cada *dataset* passou por uma fase de pré-processamento, tendo suas classes balanceadas e melhores features escolhidas, além de serem filtradas e separadas levando em consideração uma mínima redundância e máxima relevância.

Os algoritmos utilizados na classificação foram: LSTM, GRU, e XGBoost. Segundo os autores, os algoritmos LSTM e GRU, que são modelos de redes neurais recorrentes, foram escolhidos devido a importância do tempo e sequenciamento em ataques de redes e o XGBoost devido a estrutura do problema beneficiar o desempenho de métodos de conjuntos hierárquicos, permitindo atingir altas acurácias.

Para a avaliação do algoritmo de XGBoost as métricas utilizadas foram: multiclass logarithmic loss e multiclass classification error rate. Para os algoritmos LSTM e GRU, as métricas utilizadas foram: Categorical Cross Entropy, categorical accuracy e f-score, além de ser utilizado o algoritmo Nadam como otimizador.

3.2 Detecção de ataques à redes IoT de baixo recurso

Diferente dos trabalhos citados na Seção 3.1 cujo foco era detectar ataques direcionados ao *broker* da rede, os trabalhos descritos a seguir têm como objetivo propor técnicas para defender dispositivos IoT de baixo recurso nas redes.

No estudo de Shalaginov *et al.* (2019) é apresentada uma estrutura na qual os dispositivos IoT comunicam-se com o *broker* central para obter um modelo com baixa necessidade de processamento necessário para detectar ataques direcionados a eles. Já Kasinathan *et al.* (2013),

Kasinathan *et al.* (2013) propõem uma infraestrutura conjunta de dispositivos 6LoWPAN e uma sistema de detecção de intrusão *open source* para análise e detecção de ataques de negação de serviço (DoS) direcionados aos dispositivos.

Detalhes e características gerais acerca dos trabalhos estão descritas abaixo.

3.2.1 MEML: Resource-aware MQTT-based Machine Learning for Network Attacks Detection on IoT Edge Devices

Shalaginov *et al.* (2019) demonstram um conceito de re-treinamento, por meio de mensagens MQTT, de métodos de *machine learning* utilizados para detecção de ataques em dispositivos de baixo recurso. O ambiente proposto envolve a utilização de um *broker* central disponível a internet, responsável pela maior parte do processamento, visto que os modelos propostos são todos treinados nele e dispositivos IoT também disponíveis à internet, pela qual conectam-se ao *broker* para trocar informações acerca do modelo e dessa maneira serem capazes de re-treiná-lo com base nos ataques recebidos.

Utilizando dessa abordagem, é possível haver um desacoplamento das técnicas utilizadas, visto que cada dispositivo exposto pode treinar um modelo de domínio específico. A transmissão dos dados é feita utilizando o protocolo MQTT, que possui características de compatibilidade com a comunicação via TCP-IP, independência entre a plataforma e o dispositivo, utiliza mínimos recursos em sua implementação, utiliza um processamento mínimo e possui a comunicação assíncrona.

Os algoritmos testados nesse ambiente foram: Artificial Neural Networks (ANN) e Support Vector Machine (SVM). O *dataset* utilizado foi o NSL-KDD, possuindo alta performance e complexidade reduzida em relação ao KDD Cup 1999, cujo foi adaptado. As avaliações de resultados foram baseadas no uso de memória dos dispositivos, complexidade de tempo e consumo de energia, todos esses atingindo médias baixas o suficiente para a utilização em ambientes de recurso limitado, porém devido à baixa complexidade dos algoritmos e utilização de dados reduzidos. Os autores citam que com a utilização de dados e modelos mais complexos, o modelo de memória deve ser repensado.

3.2.2 DEMO: An IDS Framework for Internet of Things Empowered by 6LoWPAN

Kasinathan *et al.* (2013), Kasinathan *et al.* (2013) propõem um *framework* para detecção de ataques em redes IoT sob protocolo 6LoWPAN. A arquitetura apresentada consiste

de uma rede 6LoWPAN conectada a um Sistema de Detecção de Intrusão (Suricata) e um Sistema de Pentest. Diversos dispositivos são conectados a um roteador que transmite as informações para um gateway entre a rede e a internet. Um módulo dentro da rede chamado *IDS Probe* é responsável por monitorar o tráfego e enviar as informações ao Sistema de Detecção de Intrusão. O sistema de pentest realiza os ataques na rede. Toda essa arquitetura é integrada a um sistema de gerenciamento de incidentes e eventos (SIEM) para monitorar os eventos de ataques ou alertas.

O objetivo principal do sistema é detectar ataques de negação de serviço (DoS) em redes 6LoWPAN por meio das assinaturas geradas e incluídas no Sistema de Detecção de Intrusão que recebe os dados capturados e traduzidos pelo nó *IDS Probe* na rede.

3.3 Detecção de Anomalias em redes IoT

Em adição às seções 3.1 e 3.2 anteriormente apresentadas os trabalhos descritos abaixo têm como objetivo principal a detecção de anomalias ou ataques não previamente conhecidos em redes IoT.

Eskandari *et al.* (2020) propõem um sistema de detecção de anomalias ou ataques não conhecidos que utiliza técnicas de *machine learning* para aprenderem o comportamento normal do ambiente monitorado e alertar qualquer comportamento diferente. O sistema foi desenvolvido com o objetivo de ser instalado em dispositivos IoT. Foram avaliados quatro ataques direcionados à rede e dispositivos. Dois algoritmos foram testados e avaliados por cerca de 3 métricas. Já Hasan *et al.* (2019) simulam um ambiente gerando dados normais e de sete ataques diferentes e utilizam-os para avaliar cinco algoritmos com cerca de cinco métricas.

Detalhes e características gerais acerca dos trabalhos estão descritas abaixo.

3.3.1 Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices

Eskandari *et al.* (2020) apresentam um sistema de detecção de intrusão baseado em anomalias que pode ser instalado e executado em dispositivos IoT. O sistema proposto foi testado em duas infraestruturas, na primeira como um *gateway* IoT por onde passa todo o tráfego do atacante aos dispositivos ou rede e na segunda como um complemento separado que monitora a rede alvo em busca de padrões suspeitos.

O funcionamento do sistema consiste em treinar algoritmos de classificação utili-

zando o tráfego real da rede com o objetivo de entender o seu comportamento normal, dessa forma qualquer comportamento diferente do aprendido pode ser considerado uma anomalia. Os ataques avaliados nos testes foram: Port Scan, HTTP Brute Force, SSH Brute Force e SYN Flood. Os algoritmos utilizados foram: Isolation Forest e Local Outlier Factor.

Para avaliar o funcionamento do sistema foram avaliadas nas duas redes as métricas obtidas pelos algoritmos, sendo elas: Precision, Recall e F1-Score e a utilização de recursos pelo dispositivo IDS, tendo uma boa média quando a taxa de rede não excedia 50 Mb/s de taxa de transmissão.

Os alertas gerados pelo sistema podem ser visualizados por meio de uma interface web interativa do *gateway* utilizado para a comunicação com os dispositivos (AGILE). Nesse caso, o sistema proposto foi adaptado como um módulo adicional do *gateway*.

3.3.2 Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches

Hasan *et al.* (2019) propõem um modelo para detecção de ataques e anomalias em uma insfraestrutura de rede IoT. O *dataset* utilizado foi obtido por meio de uma infraestrutura simulada por um sistema de orquestração de espaço inteligente (DS2OS) (PAHL; AUBET, 2018). Ele contém dados normais da comunicação e ataques de negação de serviço (DoS), Data Type Probing, Malicious Control, Malicious Operation, Scan, Spying e Wrong Setup.

Os algoritmos avaliados foram: Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF) e Artificial Neural Network (ANN). As métricas de avaliação utilizadas foram: accuracy, precision, recall, f1 score e Receiver Operating Characteristic Curve.

Por meio da execução do modelo, o resultado principal obtido pelo estudo mostrou que o algoritmo de Random Forest (RF) teve maior acurácia em relação aos outros modelos, além de ter previsto mais amostras de ataques DoS.

3.4 Análise Comparativa e Considerações Finais

Essa seção expõe uma discussão acerca das diferenças entre os trabalhos apresentados neste capítulo e o trabalho proposto. As principais características são discutidas e demonstradas no Quadro 4. O seguinte símbolo é utilizado para: (N/A) demonstrar uma característica que não

se aplica ao trabalho.

A característica "Rede Monitorada"indica o domínio e a estrutura da rede e dos dados monitorados. Com exceção de Kasinathan *et al.* (2013) que propuseram uma infraestrutura para uma rede 6LoWPAN, todos os outros trabalhos apresentados monitoraram uma rede MQTT centralizada possuindo sensores conectados a um único *broker* ou sistema de *machine learning*.

A característica "Ataques Monitorados"lista os ataques estudados dentro das redes monitoradas. Dos trabalhos apresentados, Shalaginov *et al.* (2019), Kasinathan *et al.* (2013) e Eskandari *et al.* (2020) monitoraram e estudaram ataques genéricos, já Ciklabakkal *et al.* (2019), Alaiz-Moreton *et al.* (2019) e Hasan *et al.* (2019) estudaram ataques específicos do MQTT. Hindy *et al.* (2020) estudaram cerca de 3 ataques genéricos e 1 específico do MQTT. Como destaque, os trabalhos de Eskandari *et al.* (2020) e Ciklabakkal *et al.* (2019) propuseram um sistema para detecção de anomalias, logo os ataques testados foram utilizados como anomalias na avaliação do sistema.

As características "Algoritmos"e "Métricas de Avaliação"listam os algoritmos utilizados nas soluções e métricas de avaliação dos resultados obtidos por elas em cada um dos trabalhos, com exceção de Shalaginov *et al.* (2019) e Kasinathan *et al.* (2013), Kasinathan *et al.* (2013), que avaliaram a performance, uso de energia e memória dos dispositivos nas situações propostas.

As características "*Dataset*"e "Features"mostram quais *datasets* e features foram utilizadas pelos trabalhos na execução de cada algoritmo, visto que nas redes MQTT é possível utilizar dados tanto de fluxo de rede, tcp, ip e do protocolo MQTT.

A última característica "Visualização"mostra qual ferramenta foi utilizada para visualização dos resultados dos experimentos em detecção em redes reais. Como alguns trabalhos se propuseram apenas a coletar os dados das redes e avaliar os algoritmos propostos, essa característica não se aplica a eles.

Logo, este trabalho objetiva criar um sistema de detecção de anomalias automático que se adapte à rede na qual ele é instalado por meio do aprendizado *online* via *streaming* de dados como apresentado no Capítulo 5. Eskandari *et al.* (2020) propõem um trabalho que se aproxima da solução proposta neste, porém a abordagem em *batch* é adotada, logo os dados são coletados e salvos em arquivos para serem utilizados no treinamento do modelo e logo após inseri-lo na rede, tendo custos de re-treino e coleta dos dados sempre que o domínio mudar.

Quadro 4 – Quadro comparativo de trabalhos relacionados.

	Hindy *et al.* (2020)	**Ciklabakkal *et al.* (2019)**	**Alaiz-Moreton *et al.* (2019)**	**Shalaginov *et al.* (2019)**	**Kasinathan *et al.* (2013)**	**Eskandari *et al.* (2020)**	**Hasan *et al.* (2019)**	**Este trabalho**
Rede Monitorada	MQTT Centralizada	MQTT Centralizada	MQTT Centralizada	MQTT Centralizada	6LoWPAN	MQTT Centralizada	MQTT Centralizada	MQTT Centralizada
Ataques Monitorados	3 Genéricos (SSH Bruteforce, Agressive Scan e UDP Scan) e 1 Específico (MQTT Bruteforce)	Ataques Anômalos por meio de Fuzzy MQTT Payloads	3 Específicos (MQTT DoS, MiTM e Instrusion)	1 Genérico (IP SYN Flood)	1 Genérico (DoS)	4 Genéricos (Port Scan, HTTP Brute Force, SSH Brute Force e SYN Flood)	7 Específicos (DoS, Data Type Probing, Malicious Control, Malicious Operation, Scan, Spying e Wrong Setup)	7 Específicos (Flooding DoS, MQTT Publish flood, SlowITe, Malformed data, Brute Force) e Ataques Anômalos
Algoritmos	LR, NB, k-NN, SVM, DT e RF	Autoencoder, SO GAAL, RF, IF, OCSVM e Kmeans Clustering	*Artificial Neural Networks* (ANN) e SVM	*Long Short-term Memory* (LSTM), *Gated Recurrent Unit* (GRU), e XGBoost	N/A	IF e LOF	LR, SVM, DT, RF e ANN	IF e IFASD
Métricas de Avaliação	Accuracy, Precision, Recall e F1-score	ROC AUC e Accuracy	Cross Entropy, Accuracy e F1-score	N/A	N/A	Precision, Recall e F1-Score	Accuracy, Precision, Recall, F1-score e AUC ROC	Accuracy, Precision, Recall, F1-score e AUC ROC
Dataset	*Dataset* próprio	*Dataset* próprio	*Dataset* próprio	NSL-KDD	N/A	*Dataset* próprio	*Dataset* próprio	MQTTset e *Dataset* próprio
Features	34 features TCP/IP e 9 MQTT	25 features TCP/IP e 6 MQTT	28 features TCP/IP e 38 MQTT	9 features TCP/IP	N/A	25 features de Fluxo da Rede	11 features MQTT e 2 de Fluxo da Rede	30 features MQTT
Visualização	N/A	Web Server Próprio	N/A	N/A	Prewikka	Web AGILE	N/A	Dojot

Fonte: Elaborado pelo autor.
Legenda: (N/A), Característica não se aplica ao trabalho.

4 PROCEDIMENTOS METODOLÓGICOS

Inicialmente foi realizada uma pesquisa bibliográfica acerca do funcionamento das redes de internet das coisas e seus protocolos, especificamente o MQTT do qual esta pesquisa teve o maior foco, mecanismos de defesas utilizados nessas redes, como por exemplo, os de detecção e prevenção de intrusão e maneiras de melhorar a segurança provida por esses mecanismos por meio de técnicas de aprendizagem de máquina, tendo em vista a automatização das defesas e reconhecimento de novos ataques. Dessa maneira, o conhecimento adquirido foi necessário para a elaboração da proposta de uma arquitetura de melhoria de segurança nas redes IoT que utilizam o protocolo MQTT em sua comunicação.

Os procedimentos deste trabalho são: Coleta de Dados, Pré-processamento e Seleção de Dados, Definição e Treinamento do Modelo, Configuração do SDI e Criação de Regras e Validação e Análise dos Resultados.

4.1 Coleta de Dados

Para avaliar a proposta em diferentes redes, a coleta de dados consistiu inicialmente em utilizar um *dataset* pré-existente contendo diversas classes de ataques MQTT, estes dados foram gerados em uma simulação de rede no estudo (VACCARI *et al.*, 2020). Também foram coletados dados de comunicação de uma rede própria simulada. A captura de informações da rede foi realizada por um agente coletor próprio programado na linguagem de programação *Python*.

4.2 Pré-processamento e Seleção de Dados

Durante o levantamento de dados da rede, toda a comunicação das camadas de aplicação, transporte e rede é capturada pelo agente, por esse motivo é necessário um pré-processamento dos dados para adequá-los à entrada do modelo proposto. Nesse procedimento foram substituídos valores faltantes e valores textuais por identificadores numéricos únicos, tendo em vista que o algoritmo recebe como entrada valores numéricos, como apresenta a Seção 6.1, explicando os procedimentos anteriormente realizados na execução dos passos de processamento e seleção.

Além do pré-processamento, com o conhecimento do domínio e foco dessa proposta em redes MQTT, foram selecionados dos dados apenas os valores relacionados à camada de

aplicação e comunicação MQTT, com o objetivo de demonstrar que é possível detectar ataques utilizando apenas as informações do protocolo MQTT.

As informações e seus tipos citados anteriormente podem ser vistas a partir do Quadro 5.

4.3 Definição e Treinamento do Modelo

Os algoritmos utilizados foram: IF para o treinamento *offline* e IFASD para o treinamento *online*. O modelo do IFASD é implementado através do *framework* PySAD (YILMAZ; KOZAT, 2020) construído na linguagem de programação *Python*. Já o IF é implementado utilizando o *framework sklearn*.

Para ambos os algoritmos os *datasets* utilizados são divididos em 70% dos dados para treino e 30% dos dados para teste, além disso por se tratar de detecção de anomalias, inicialmente os algoritmos são treinados com todos os dados considerados normais do conjunto de dados de treino sem a utilização dos rótulos.

Para os dois modelos é necessário um parâmetro de contaminação, logo é necessário analisar o *dataset* anteriormente com o objetivo de saber quantos dados anômalos estão contidos, para utilizar na definição do modelo.

4.4 Configuração do SDI e Criação de Regras

Para detecção das ameaças foi utilizado um agente próprio em conjunto com o SDI Suricata, ferramenta *opensource*, *network-based* e *signature-based*, ou seja, opera analisando o tráfego da rede e utilizando das regras definidas em sua configuração para detectar ataques.

As regras são criadas pelo agente utilizando as saídas do modelo e transformando-as no padrão esperado pelo Suricata. Dessa maneira, para novos ataques serão criadas novas regras de maneira que o SDI os identifique e alerte para o usuário ou administrador.

De modo geral, o funcionamento do SDI em conjunto com o modelo tem como objetivo salvar as regras criadas e detectar ataques que ocorreram no passado, tendo em vista que o algoritmo de ML pode mudar de conceito, por esse motivo pode passar a não detectar mais ataques já detectados anteriormente, esses são detectados a partir das regras salvas no SDI.

4.5 Validação e Análise dos Resultados

Para a validação e análise dos resultados dos experimentos, os *datasets* obtidos foram utilizados na execução dos modelos propostos. A avaliação do modelo se deu por meio de cinco métricas: *Precision*, *Recall*, *F1-score*, *Accuracy* e AUC ROC.

Cada uma das métricas utiliza indicadores em seu cálculo, são eles: Falso Positivo, Falso Negativo, Verdadeiro Positivo e Verdadeiro Negativo. Esses indicadores definem a classe dos dados preditos pelo algoritmo, permitindo a avaliação de seus acertos e erros.

Como o objetivo dessa pesquisa é predizer dados anômalos, é importante salientar que anomalias são tratadas como a classe positiva e os dados normais como a classe negativa. Logo os indicadores, para efeito de entendimento, podem ser definidos como:

- **Verdadeiro Positivo (VP):** Anomalia predita e os dados referem-se a uma anomalia.
- **Falso Positivo (FP):** Anomalia predita, porém os dados referem-se a dados normais.
- **Verdadeiro Negativo (VN):** Dado normal predito e os dados referem-se a dados normais.
- **Falso Negativo (FN):** Dado normal predito, porém os dados referem-se a uma anomalia.

Logo, as métricas são descritas da seguinte maneira:

- ***Precision* (P):** Define a quantidade de acertos de uma determinada classe baseado nas predições feitas. A Equação 4.1 define dado as anomalias preditas, a porcentagem de anomalias preditas corretamente. A Equação 4.2 define dado os dados normais preditos, a porcentagem de dados normais preditos corretamente.

$$P = \frac{VP}{VP + FP} \tag{4.1}$$

$$P = \frac{VN}{VN + FN} \tag{4.2}$$

- ***Recall* (R):** Define a quantidade de acertos de uma determinada classe. A Equação 4.3 define dado o total de anomalias contidas nas instâncias, a porcentagem de predições de anomalias corretas. A Equação 4.4 define dado o total de dados normais contidos nas instâncias, a porcentagem de predições de dados normais corretas.

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

$$R = \frac{VN}{VN + FP} \tag{4.4}$$

– ***F1-Score* (F1):** Define uma combinação entre as métricas *Precision* e *Recall*, para serem observadas em conjunto por meio de um único número. Definida pela Equação 4.5.

$$F1 = 2 \cdot \frac{P \times R}{P + R} \tag{4.5}$$

– ***Accuracy* (A):** Define a porcentagem total de predições corretas, tanto de anomalias quanto de dados normais. Definida pela Equação 4.6.

$$A = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.6}$$

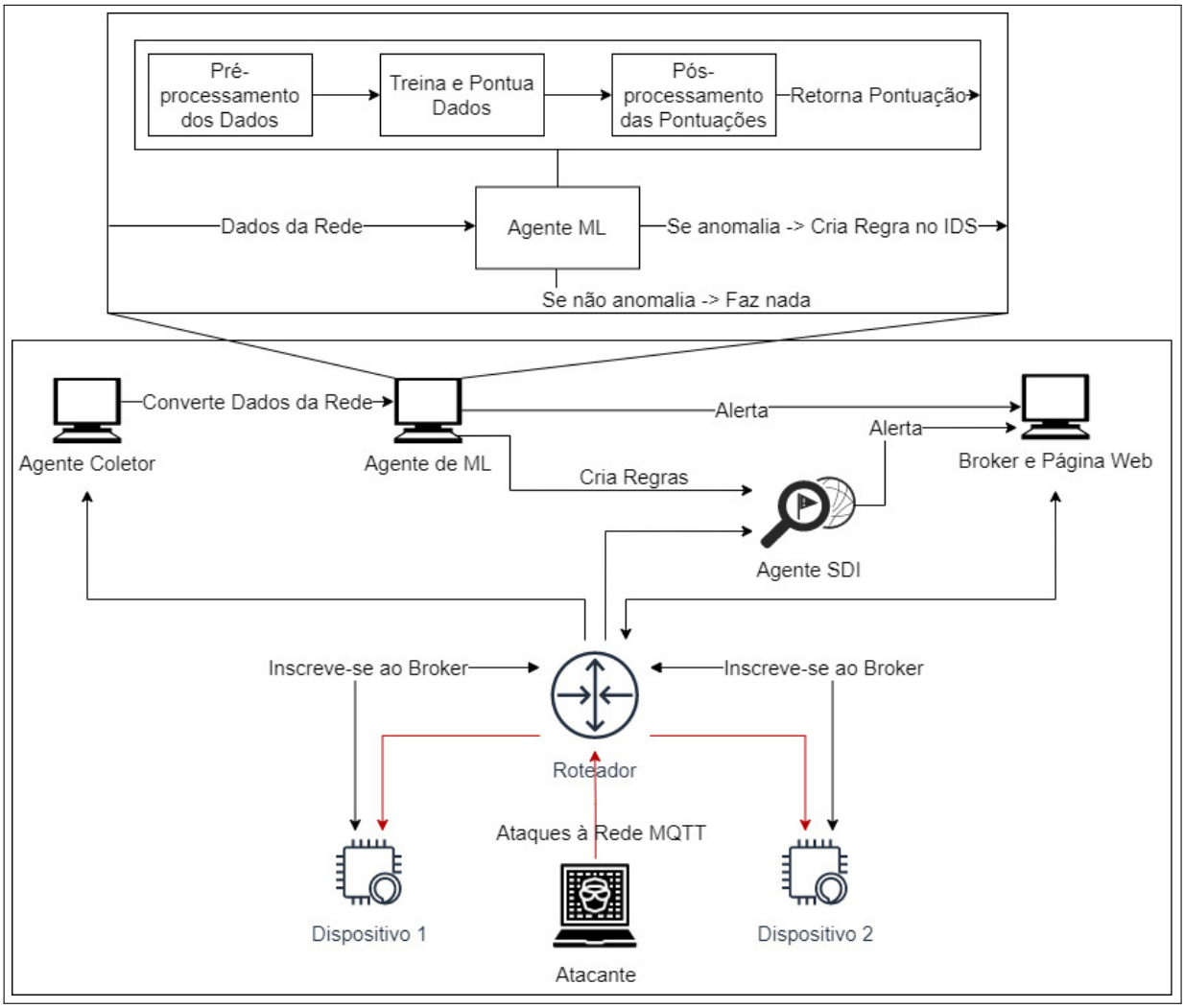
– **AUC ROC:** Área sob a ROC, avalia a performance do algoritmo baseado no limite da relação entre verdadeiros positivos e verdadeiros negativos. Com valores de 0 a 1, sendo: (0) algoritmo prediz anomalias como dados normais e dados normais como anomalias; (0.5) algoritmo não consegue distinguir as duas classes; (>0.5) taxa de performance da predição das duas classes pelo algoritmo (NARKHEDE, 2018).

Na execução da proposta em tempo real o modelo não possui informação de assertividade da predição, por esse motivo a análise do seu funcionamento foi dada por meio da visualização dos alertas gerados pelo SDI em momentos de simulações de ataques na rede, logo é necessário um conhecimento do domínio para avaliação do funcionamento.

5 PROPOSTA

A proposta consiste em uma arquitetura[1], demonstrada na Figura 4 composta de: um agente coletor, um agente de ML e um agente SDI. Além disso é inserida a visualização dos dados, dispositivos da rede e alertas gerados pelo agente SDI e de ML por meio de uma Página Web, com o objetivo de tornar claro o comportamento atual da rede.

Figura 4 – Arquitetura da solução proposta.



Fonte: Elaborado pelo autor.

A solução foi desenvolvida em módulos para permitir que cada agente possa ser instalado em dispositivos diferentes ou todos no mesmo dispositivo, caso necessário. Além disso, como Agente SDI e *Broker* foram utilizadas ferramentas já existentes, logo é possível substituí-las por outras que atendam aos requisitos necessários para as diferentes redes. As

[1] Todo o código desenvolvido para a elaboração da proposta está contido no repositório: https://github.com/jarelio/mqtt_security/.

seções seguintes apresentarão o funcionamento em detalhes de cada um dos módulos propostos.

5.1 Agente Coletor

O Agente Coletor é responsável por ler e processar o tráfego da rede, então inseri-lo em um *dataset*, enviá-lo ao Agente de ML, ou ambos. Seu funcionamento é demonstrado no Código-Fonte 1. Foi programado na linguagem *Python* e as principais bibliotecas utilizadas por ele são: csv, usada para gerar o *dataset*; *socket*: usada para abrir *sockets* e permitir a leitura do tráfego da rede; *pahomqtt*: usada para transformar o agente em um cliente MQTT para enviar os dados para o Agente de ML.

Para comunicar-se com o Agente de ML via conexão cliente/*broker* MQTT são necessários os parâmetros *args.host*, *args.port*, *args.topic*, significando identificador do Agente de ML, porta (padrão: 1883) e tópico (padrão: *network/data*) respectivamente.

```
# Abertura de Socket
conn = socket.socket(socket.PF_PACKET, socket.SOCK_RAW, socket.ntohs(3))
# Cliente MQTT
mqttc = mqtt.Client()
mqttc.connect(args.host, args.port)
mqttc.loop_start()
# Cria CSV inicial
writer, csvfile = create_csv(args.csv_filename)
while True:
    # Le dados da rede
    data = sniff_data(conn)
    if len(data):
        # Transforma os dados para a entrada do modelo
        agent_data = transform_data_to_model_input(data)
        mqttc.publish(args.topic, agent_data)
        # Transforma os dados para inseri-los no CSV
        csv_data = transform_data_to_csv_input(data)
        add_to_csv(csv_data, writer, csvfile)
# Fecha as conexoes ao Agente de ML e do arquivo CSV
mqttc.loop_stop()
mqttc.disconnect()
csvfile.close()
```

Código-fonte 1 – Funcionamento do Agente no modo ambos.

Os pacotes gerados na comunicação são extraídos para serem obtidos os cabeçalhos TCP e IP, além do *payload* e cabeçalho MQTT. Apenas os pacotes com porta fonte/destino 1883 ou 8883 (padrões do MQTT) são utilizados, como demonstra o Código-Fonte 2.

```
def sniff_data(conn)
    # Captura dados da rede
    network_data, _ = conn.recvfrom(65536)
    # Decodifica o cabecalho Ethernet
    ethernet_header_data, ip_data = ethernet_frame(network_data)
    # Decodifica o cabecalho IPv4
    if ethernet_header_data['protocol'] == IPV4_PROTOCOL_HEXNUMBER:
        ip_header_data, tcp_data, tcp_data_length = ip_header(ip_data)
        # Decodifica o cabecalho TCP
        if ip_header_data['protocol'] == TCP_PROTOCOL_NUMBER:
            tcp_header_data, packet_data = tcp_header(tcp_data,
                tcp_data_length)
            # Decodifica os dados MQTT usando a porta padrao
            if tcp_header_data['src_port'] == MQTT_PORT or
                tcp_header_data['dst_port'] == MQTT_PORT:
                 return mqtt_packet(packet_data)
    return ''
```

Código-fonte 2 – Função de leitura e processamento de dados da rede.

5.2 Agente de *Machine Learning* (ML)

O Agente de ML é responsável por receber os dados, fazer o pré-processamento nas *features* categóricas, transformando-as em valores numéricos únicos por meio de um dicionário de palavras e usando-as no treinamento e pontuação do algoritmo, a pontuação é pós-processada por um calibrador e categorizada como anomalia ou não, caso seja, o modelo envia os dados daquele pacote para o Agente SDI para que seja criada uma regra de detecção de ameaça. Também são enviados os dados para o *Broker* para visualização do alerta, tendo em vista que o Agente SDI mantém os alertas para detecção de ataques iguais anteriormente detectados.

Foi programado na linguagem *Python* e a sua comunicação com o Agente Coletor é feita através do próprio protocolo MQTT. O agente foi criado como um *broker* MQTT[2] e se

[2] Ferramenta utilizada para criação do *broker*: https://mosquitto.org/

inscreve no tópico definido pelo usuário (padrão: *network/data*), pelo qual recebe os dados do Agente Coletor. Também possui um cliente *MQTT* que se conecta ao Agente SDI utilizando a porta, *host* e o tópico necessário (padrão: *ml/rules*) para o envio dos resultados do modelo.

O algoritmo utilizado pelo Agente é o IFASD, implementado no *framework* PySAD, os parâmetros utilizados no algoritmo foram os padrões, logo o tamanho adotado de janela deslizante foi de 2048 instâncias de dados, como explanado no funcionamento do algoritmo na seção 2.3.2.

O funcionamento padrão do Agente, como demonstra o Código-Fonte 3 é realizar a coleta dos dados por um tempo inicial pré-definido com o objetivo de treinar o comportamento normal da rede. Nesse momento uma *thread* é executada em paralelo para contabilizar o tempo enquanto o tráfego é recebido e salvo em uma lista em memória, após o tempo finalizar os dados são treinados no modelo que passa a pontuar todos os dados subsequentes.

```
# Inicia o agente
def run(self, target='127.0.0.1', port=1883, topic='network/data'):
    self.connect(target, port, 60)
    self.subscribe(topic, 0)
    self.__countdown_thread.start()
    self.loop_forever()
# Contador para leitura inicial
def __count_down(self, timer):
    while (timer):
        time.sleep(1)
        timer -= 1
    self.__end_timer = True
    self.__init_model(self.__initial_data)
# Inicia o modelo com dados iniciais e o calibrador
def __init_model(self, initial_data):
    if len(initial_data) > 0:
        self.__model = IForestASD(initial_window_X=initial_data)
    else:
        self.__model = IForestASD()
    self.__calibrator = ConformalProbabilityCalibrator(windowed=True,
        window_size=2048)
    self.__model_initialized = True
# Inicia o agente com tempo de 300 segundos e limite de anomalia 40%
if __name__ == "__main__":
```

```
24      agent = MlAgentClient(initial_time=300, anomaly_threshold=0.4)
        agent.run()
```

Código-fonte 3 – Inicialização do agente.

Cada dado pontuado é transformado em uma probabilidade por meio de um calibrador conformal como demonstra o Código-Fonte 4, implementado também no *framework* PySAD, recebe como parâmetros: *windowed True* e *window_size* 2048, mesma quantidade de instâncias utilizadas na definição do modelo. Dessa forma cada dado terá uma probabilidade de ser anomalia, caso ultrapasse o limite pré-definido é caracterizado como anomalia e o modelo envia os dados para a criação da regra no Agente SDI.

```
1       def __score_anomaly(self, payload):
            if not self.__model_initialized: return
3           # Treina modelo com a nova instancia
            self.__model.fit_partial(payload)
5           # Pontua dado
            score = self.__model.score_partial(payload)
7           # Transforma pontuacao em probabilidade
            calibrated_score = self.__calibrator.fit_transform_partial(score)
9           # Cria regra se probabilidade for maior que o limite definido
            if calibrated_score > 1-self.__anomaly_threshold:
11              self.send_data_to_ids(payload)
                self.send_alert_to_broker(payload)
```

Código-fonte 4 – Função de pontuação de anomalias do agente de ML.

5.3 Agente de Detecção de Intrusão (SDI)

O Agente de Detecção de Intrusão é responsável por receber os dados do Agente de ML que indicam uma anomalia e transformá-los em regras para serem utilizadas em um sistema de detecção de intrusão baseado em regras, além disso é responsável também por ler os alertas gerados pelo SDI e enviá-los para o *broker*.

Foi programado na linguagem *Python* e a sua comunicação com o Agente de ML é feita através do próprio protocolo MQTT. O agente foi criado como um *broker* MQTT através

da ferramenta *Mosquitto* e se inscreve no tópico definido pelo usuário (padrão: *ml/rules*), pelo qual recebe os dados do Agente de ML. Também possui um cliente *MQTT* que se conecta ao *broker* utilizando a porta, *host* e o tópico necessário para o envio dos alertas.

O sistema de detecção intrusão utilizado foi o Suricata, sua instalação é bem simples e após instalado já pode ser utilizado. Para serem criadas regras customizadas é preciso inseri-las em um novo arquivo com o formato aceito pelo SDI[3] e reiniciar o serviço para serem utilizadas na detecção.

Nos sistemas linux o caminho para o arquivo de configuração do suricata é **/etc/suricata/suricata.yaml**, logo é necessário adicionar, na opção ***rule-files*** contida nele, o caminho para o arquivo de regras customizadas. Após o arquivo de regras ser populado, o Suricata dispõe do comando **suricatasc -c ruleset-reload-nonblocking**, para atualizar as regras de forma não bloqueante (em segundo plano), sem precisar reiniciar completamente o serviço do Suricata.

A leitura dos alertas para envio ao *broker* dá-se utilizando o arquivo **/var/log/suricata/eve.json**, responsável por gerar os alertas no formato JSON. Logo, tendo em vista a descrição anteriormente dada, o Código Fonte 5 ratifica o funcionamento do Agente SDI.

```
def on_message(self, mqttc, obj, msg):
    payload = msg.payload.decode('utf-8')
    # Cria regra no formato e insere no arquivo de regras
    rule = self.convert_data_to_rule(payload)
    self.add_rule_to_file(rule)
    # Reinicia as regras do Suricata
    self.restart_ids_rules()
if __name__ == "__main__":
    # Inicia Agente SDI e cliente para o broker
    agent = IDSAgent()
    agent.run()
    broker_client = mqtt.Client()
    while True:
        # Le e envia alertas dos logs do Suricata
        data = {'alert': read_alert(args.suricata_logs_filepath)}
        if data['alert']:
            broker_client.publish(args.topic, ujson.dumps(data))
    agent.loop_stop()
    broker_client.loop_stop()
```

[3] Formato de regras aceito pelo Suricata: https://suricata.readthedocs.io/en/latest/rules/intro.html#

Código-fonte 5 – Funcionamento do Agente SDI.

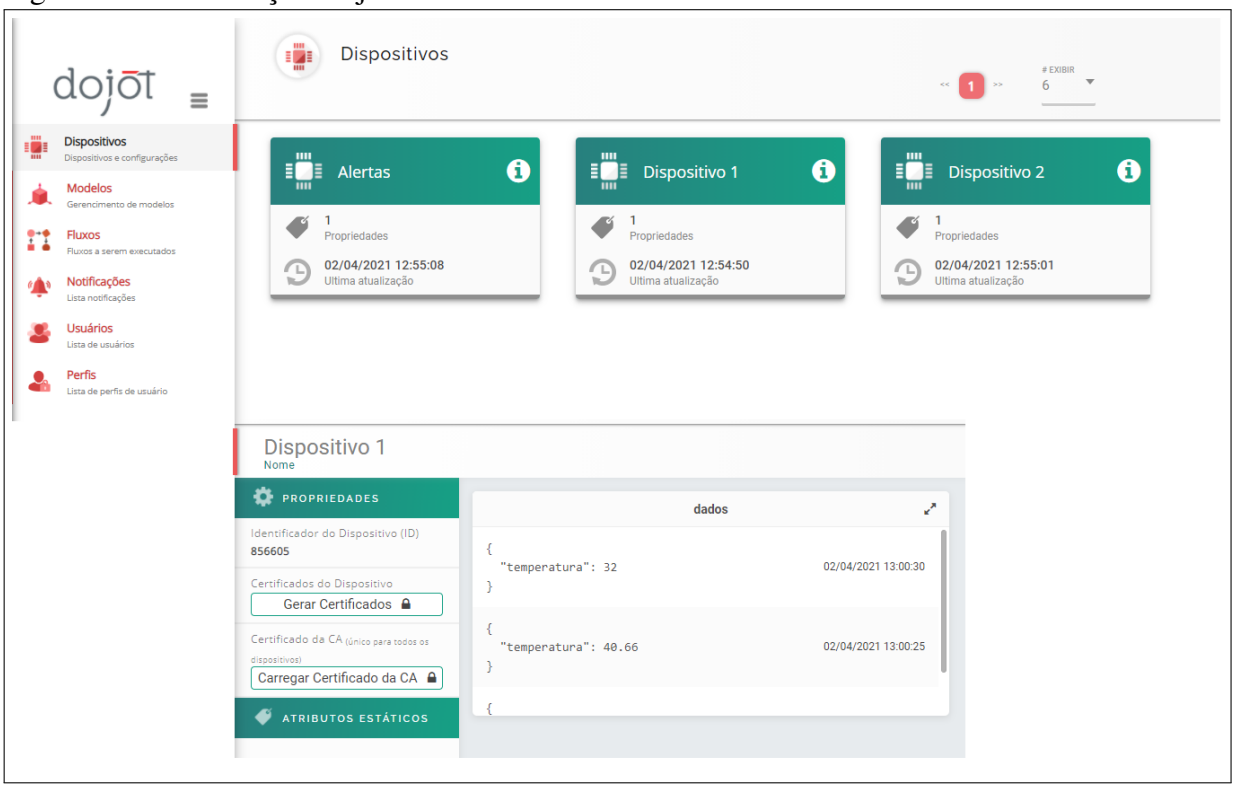
5.4 *Broker* e Página Web

Como a rede proposta possui um ambiente de comunicação MQTT, um *broker* é necessário para receber as informações dos dispositivos, neste trabalho foi escolhido a Dojot[4], uma plataforma *middleware*, *broker*, *open source* desenvolvida no Brasil que facilita o desenvolvimento de soluções no ecossistema IoT.

Possui uma própria arquitetura de micro-serviços que permite o gerenciamento do ciclo de vida dos dispositivos, a construção de fluxos e dados para processamento em tempo real, a persistência dos dados e contém uma interface para acesso dos dados em tempo real.

Neste trabalho foi utilizada para gerenciar a visualização ampla dos dispositivos e alertas gerados no Agente SDI. Sua arquitetura possui um *broker* interno que recebe as mensagens via *MQTT* e insere em dispositivos criados para visualização, como demonstra a Figura 5.

Figura 5 – Visualização Dojot.



Fonte: Elaborado pelo autor.

[4] Link para a plataforma: https://dojot.com.br/

5.5 Dispositivos e Atacante

Os dispositivos foram simulados utilizando dois clientes MQTT programados em *Python* utilizando a biblioteca *paho-mqtt*. O primeiro dispositivo envia palavras aleatórias em um JSON. E o segundo envia uma simulação de localização, também no formato JSON. Os funcionamentos dos dispositivos estão demonstrados no Código-Fonte 6;

```
# Inicia cliente ao broker
client = mqtt.Client()
client.username_pw_set(args.broker_user)
client.connect(args.broker_ip, args.broker_port)
client.loop_start()
# Representacao dos envios dos dois dispositivos
while True:
    # Dispositivo 1
    data = {"dados": {"localizacao": {"x": random.randint(0, 9000), "y":
        random.randint(0, 9000)}}}
    # Dispositivo 2
    data = {"dados": {"palavra": r.get_random_word()}}
    client.publish(args.broker_topic, json.dumps(data))
    time.sleep(3)
```

Código-fonte 6 – Funcionamento dos Dispositivos na Rede Proposta.

Já o atacante foi simulado também como um cliente utilizando a ferramenta Malaria[5], gerando *fuzzy* payloads e dados de conexão anômalos.

[5] https://github.com/etactica/mqtt-malaria

6 EXPERIMENTOS E RESULTADOS

Neste capítulo serão apresentados e discutidos os resultados obtidos nas experimentações realizadas. Inicialmente são descritos os procedimentos realizados em ambos os experimentos. Nas seções seguintes são apresentados as configurações adotadas por cada experimento, suas execuções e resultados.

O primeiro experimento foi executado com o objetivo de comparar os dois algoritmos propostos e o funcionamento do algoritmo usado no Agente de ML, tendo em vista que os dados já foram coletados de outra rede. O segundo experimento foi realizado a partir de uma simulação na rede proposta, logo nele são comparados os dois algoritmos, avaliado o funcionamento do Agente Coletor, Agente de ML, Agente SDI e a visualização dos alertas gerados.

Todos os experimentos reportados aqui foram executados em uma máquina virtual *VMWARE* com 12GB de memória RAM, sistema operacional Lubuntu 20.04 LTS e 4 processadores lógicos e 2 núcleos por processador virtualizados de um Intel Core i7.

6.1 Procedimentos Iniciais

Com os *datasets*, foi necessário um pré-processamento para se adequarem aos modelos propostos. Nos dois foram transformados todos os valores categóricos por numéricos únicos através de um pré-processador de codificação de rótulos. Das *features* indicadas no Quadro 5 as seguintes foram transformadas: *mqtt.conack.flags*, *mqtt.conflags*, *mqtt.hdrflags*, *mqtt.protoname*, *mqtt.msg*.

Além disso, para manter todas as instâncias referentes ao domínio da rede, foi feita uma seleção das *features* relacionadas apenas aos dados da camada de aplicação e comunicação via protocolo MQTT. Por esse motivo, das *features* indicadas no Quadro 5 contidas nos *datasets*, as seguintes foram removidas: *tcp.flags*, *tcp.time_delta*, *tcp.len*.

6.2 Experimento Rede MQTTset

Neste experimento foi utilizado o *dataset* coletado no estudo (VACCARI *et al.*, 2020), ele possui dados de 10 sensores conectados a um *broker* e dados de um atacante realizando cerca de 6 ataques *MQTT* à rede (*Flooding DoS*, *MQTT Publish flood*, *SlowITe*, *Malformed data*, *Brute Force*).

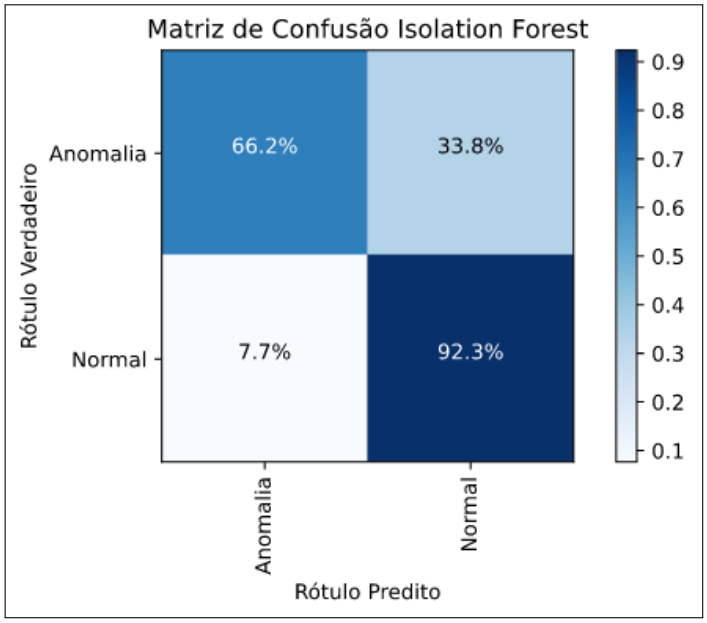
Os resultados obtidos na Tabela 1 e Figura 6 foram providos pelo modelo com o

algoritmo IF utilizando os seguintes parâmetros:

- **max_samples:** 0.2, máximo de amostras que podem ser utilizadas em cada estimador, sendo 100 o número padrão de estimadores.
- **random_state:** 110, número escolhido para definir a pseudo-randomização que o algoritmo faz na escolha de cada *feature*.
- **contamination:** 0.3, taxa de contaminação do *dataset*.
- **n_jobs:** -1, número de *jobs* utilizados em paralelo na execução do algoritmo. -1 significa que todos os *jobs* foram utilizados.

Os parâmetros foram escolhidos com base em uma série de execuções do modelo para avaliar os melhores parâmetros, como indicam as Figuras 13 e 14. Como o algoritmo IF funciona melhor com poucas instâncias e taxa de contaminação baixa, foi escolhido o menor número de *samples* e menor taxa de contaminação para os dois modelos propostos, tendo em vista a performance dos algoritmos com os determinados parâmetros.

Figura 6 – Matriz de Confusão IF - Experimento MQTTset.



Fonte: Elaborado pelo autor.

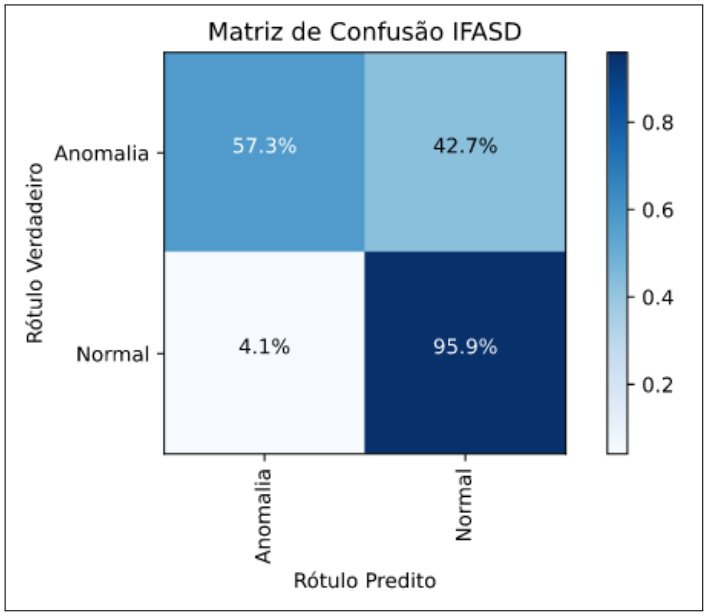
Tabela 1 – Métricas de Avaliação IF - Experimento MQTTset.

Geral		**Dados Normais**			**Anomalias**		
Acurácia Geral	**AUC ROC**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
79.3%	79.3%	73.2%	92.3%	81.7%	89.6%	66.2%	76.2%

Fonte: Elaborado pelo autor.

Já utilizando a mesma taxa de contaminação de 0.3 e o tamanho de janela deslizante de 2048 instâncias, o algoritmo IFASD obteve os resultados visualizados na Figura 7 e Tabela 2.

Figura 7 – Matriz de confusão IFASD - Experimento MQTTset.



Fonte: Elaborado pelo autor.

Tabela 2 – Métricas de Avaliação IFASD - Experimento MQTTset.

Geral		**Dados Normais**			**Anomalias**		
Acurácia Geral	**AUC ROC**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
76.6%	76.6%	69.2%	95.9%	80.4%	93.4%	57.3%	71.0%

Fonte: Elaborado pelo autor.

6.3 Experimento Rede Proposta

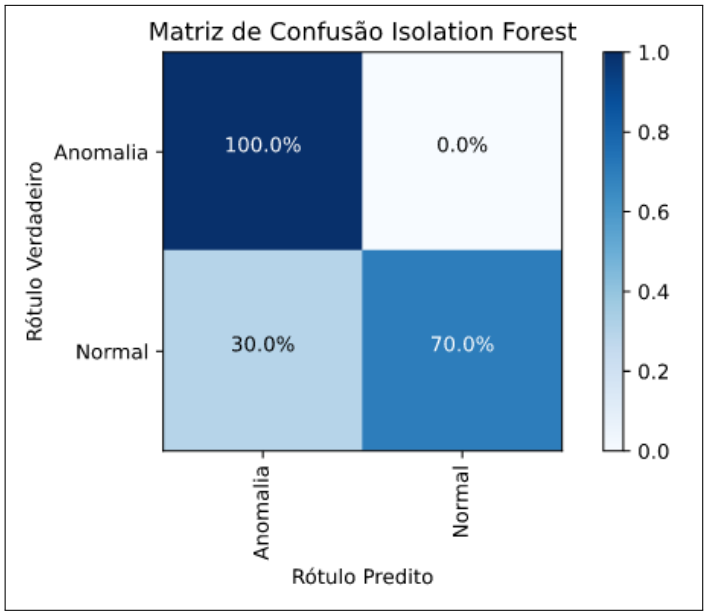
Neste experimento foi realizada uma simulação na rede proposta vide Seção 5 contendo 2 sensores conectados a um *broker* e um atacante realizando ataques. A partir da simulação foi obtido um *dataset* com as *features* demonstradas no Quadro 5, para ser utilizado na avaliação *offline* dos modelos propostos. Também foi possível analisar o funcionamento do Agente SDI na criação das regras por meio dos alertas gerados e visualizados na Página Web.

Os resultados obtidos na Tabela 3 e Figura 8 foram providos pelo modelo com o algoritmo IF utilizando os seguintes parâmetros:

– **max_samples:** 0.2, máximo de amostras que podem ser utilizadas em cada estimador, sendo 100 o número padrão de estimadores.

– **random_state:** 110, número escolhido para definir a pseudo-randomização que o algoritmo faz na escolha de cada *feature*.
– **contamination:** 0.3, taxa de contaminação do *dataset*.
– **n_jobs:** -1, número de *jobs* utilizados em paralelo na execução do algoritmo. -1 significa que todos os *jobs* foram utilizados.

Figura 8 – Matriz de Confusão IF - Experimento Proposta.



Fonte: Elaborado pelo autor.

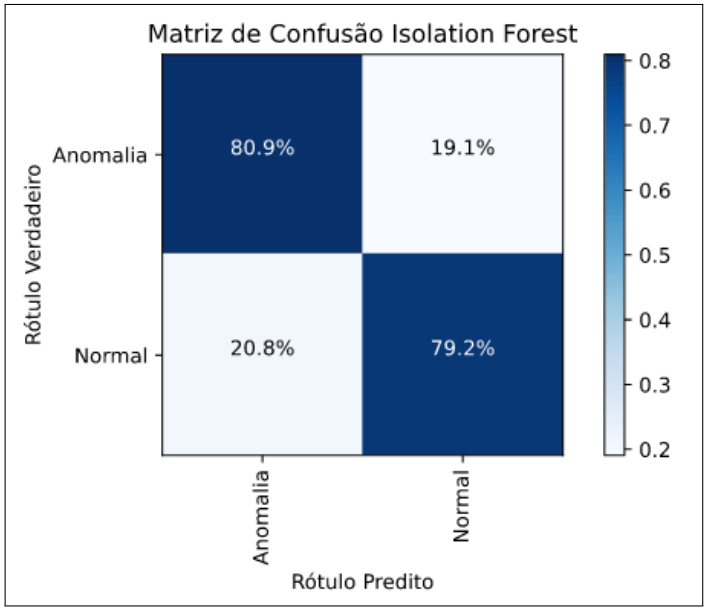
Tabela 3 – Métricas de Avaliação IF - Experimento Proposta.

Geral		**Dados Normais**			**Anomalias**		
Acurácia Geral	**AUC ROC**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
79.2%	85.0%	100.0%	70.0%	82.4%	59.7%	100.0%	74.7%

Fonte: Elaborado pelo autor.

Já utilizando a mesma taxa de contaminação de 0.3 e o tamanho de janela deslizante de 2048 instâncias, o algoritmo IFASD obteve os resultados visualizados na Figura 9 e Tabela 4.

Figura 9 – Matriz de confusão IFASD - Experimento Proposta.



Fonte: Elaborado pelo autor.

Tabela 4 – Métricas de Avaliação IFASD - Experimento Proposta.

Geral		**Dados Normais**			**Anomalias**		
Acurácia Geral	**AUC ROC**	**Precision**	**Recall**	**F1-Score**	**Precision**	**Recall**	**F1-Score**
79.7%	80.1%	90.3%	79.2%	84.4%	63.3%	80.9%	71.0

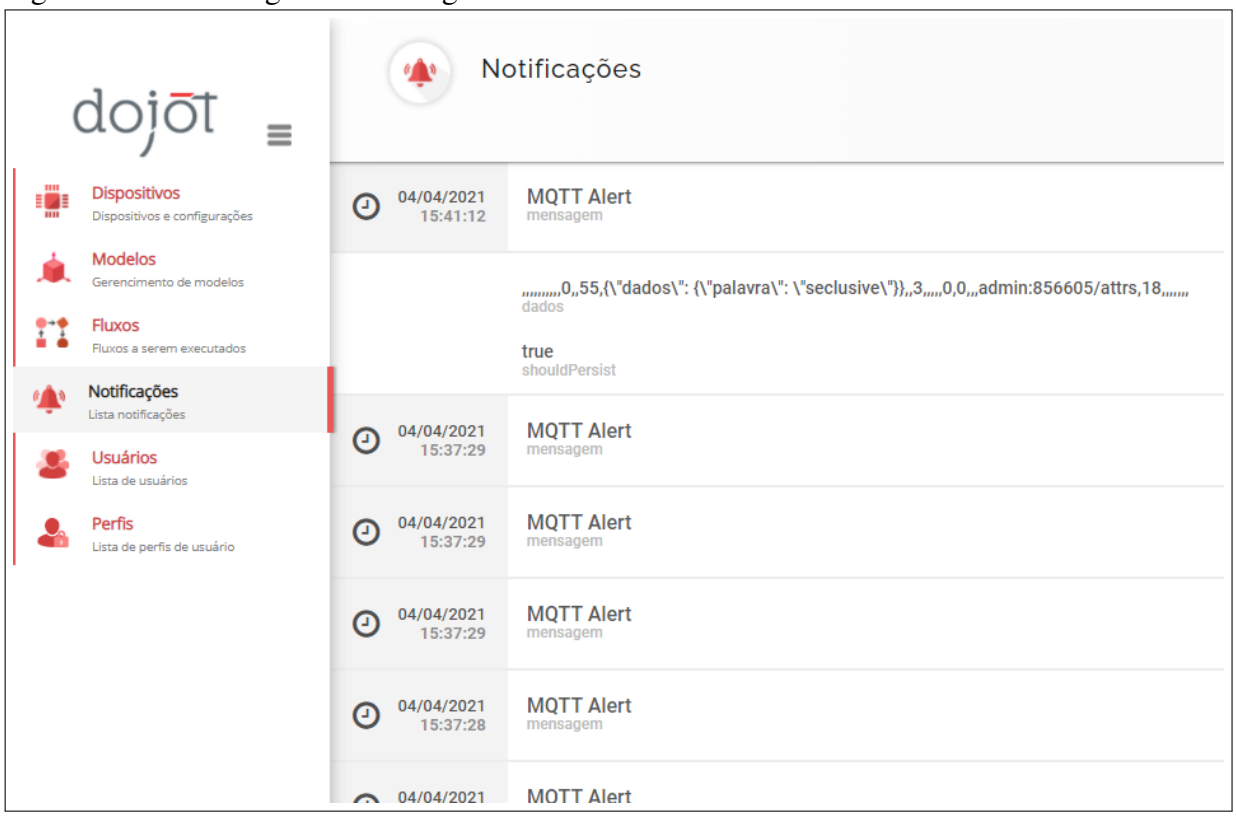
Fonte: Elaborado pelo autor.

Na visualização do funcionamento do agente SDI, é possível observar os alertas gerados na Página Web, além dos dados enviados pelos dispositivos como apresentam as Figuras 10 e 11.

Figura 10 – Visualização dos dispositivos na Página Web.

Fonte: Elaborado pelo autor.

Figura 11 – Alertas gerados na Página Web.



Fonte: Elaborado pelo autor.

6.4 Discussão dos Resultados

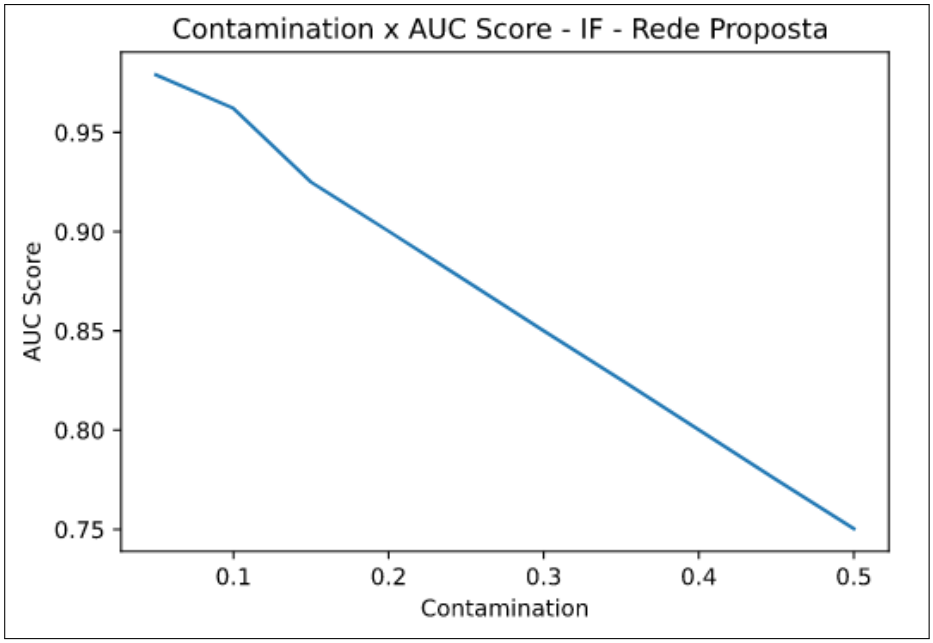
Apesar de ter atingido uma taxa de acurácia baixa em comparação aos trabalhos relacionados que utilizaram diferentes *features* e diferentes algoritmos, os modelos aplicados obtiveram um maior *recall* em dados normais, o que torna-se interessante na defesa em redes IoT devido ao problema que diversos SDI sofrem: alto número de falsos positivos. Além disso a acurácia não se diferenciou muito das duas abordagens (*offline* e *online*), o que indica que a utilização do modelo *online* pode ser mais escalável e utilizável em aplicações em tempo real, diferentemente dos trabalhos relacionados que utilizaram uma abordagem *offline*.

Em geral a acurácia dos modelos aplicados mantiveram a média entre 75% e 82% e obtiveram *recall* de dados normais acima de 80% e ataques acima de 60% utilizando a abordagem *online*.

No experimento da rede proposta foi utilizada uma taxa de anomalia fixa, um dos motivos de diminuição da performance do modelo, já que em aplicações em tempo real ainda não é possível definir a quantidade de ataques que estão sendo realizados em razão da quantidade

total de dados na comunicação. Esse fenômeno pode ser visto na Figura 12.

Figura 12 – Relação *contamination* e AUC ROC



Fonte: Elaborado pelo autor.

Por fim, a execução dos dois modelos propostos nas duas redes distintas apresentaram comportamentos semelhantes em razão dos parâmetros definidos. A utilização do algoritmo *IFASD* torna-se escalável e utilizável na rede proposta, tendo em vista a comunicação em tempo real dos dispositivos e a implantação e atuação dos agentes coletor, de ML e SDI.

7 CONCLUSÕES E TRABALHOS FUTUROS

Com o aumento dos dispositivos IoT e em paralelo o surgimento de diversas falhas de seguranças atreladas, acompanhados da não capacidade de defesa por parte dos dispositivos devido os baixos recursos que possuem, torna-se necessário a construção de uma solução de segurança que detecte não somente ataques conhecidos, mas também ataques anômalos, conhecidos como de *zero-day*.

Este trabalho então, se propôs a apresentar uma solução modular baseada na análise de dados de redes IoT operando com o protocolo MQTT, por meio de agentes detectores de intrusão atrelados à técnicas de *Machine Learning* para *stream* de dados. Por meio dos resultados foi possível analisar a performance da solução proposta, que apesar de ter atingido uma acurácia geral baixa, detectou anomalias com uma taxa média de 60% de acerto. Além de resolver, por meio da técnica de aprendizado *online*, o problema de re-treino e mudança de domínio presente nas técnicas tradicionais de *Machine Learning*.

Como trabalhos futuros, para construir uma solução ainda mais precisa, podem ser utilizados outros algoritmos baseados no aprendizado *online* em conjunto, obtendo os resultados de vários algoritmos e gerando um resultado mais preciso a partir deles. Além disso, uma melhor extração dos dados para diferenciação no algoritmo pode ser feita, técnicas para extração de contexto a partir das *features* podem ser úteis, tendo em vista o funcionamento da detecção de anomalias em dados muito diferentes dos normais.

No âmbito da aplicação em redes IoT, avaliações de escalabilidade dos algoritmos e agentes utilizados podem trazer informações utilizadas para melhorar o desempenho da solução, como a avaliação do tempo de treinamento e detecção dos modelos, quantidade de recursos gastos no uso dos agentes e entre outros. Ademais, devido à propriedade de heterogeneidade dessas redes, a solução necessita de novos testes em redes com protocolos diferentes e novos dispositivos.

Além disso, na utilização do algoritmo IFASD é necessária a escolha pré-definida da taxa de anomalia da rede. Em ambientes reais não é possível ainda detectar a taxa de ataques contidos, dessa forma faz-se necessária uma pesquisa para a mudança da taxa de anomalia de maneira automática, para evitar a redução de performance dos algoritmos com os diferentes números de ataques.

REFERÊNCIAS

AHMAD, S.; LAVIN, A.; PURDY, S.; AGHA, Z. Unsupervised real-time anomaly detection for streaming data. **Neurocomputing**, Elsevier, v. 262, p. 134–147, 2017.

Ahmed, S.; Lee, Y.; Hyun, S.; Koo, I. Unsupervised machine learning-based detection of covert data integrity assault in smart grid networks utilizing isolation forest. **IEEE Transactions on Information Forensics and Security**, v. 14, n. 10, p. 2765–2777, 2019.

AITUDE. **Supervised vs Unsupervised vs Reinforcement**. 2020. Disponível em: https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/#:~:text=Unsupervised%20Learning%20discovers%20underlying%20patterns,patterns%20and%20predict%20the%20output. Acesso em: 27 mar. 2021.

Akamai Technologies. **Overview: The MQTT Protocol**. 2020. Disponível em: https://developer.akamai.com/iot-edge-connect/mqtt. Acesso em: 17 fev. 2021.

Al-Sarawi, S.; Anbar, M.; Alieyan, K.; Alzubaidi, M. Internet of things (iot) communication protocols: Review. In: **2017 8th International Conference on Information Technology (ICIT)**. [*S. l.*: *s. n.*], 2017. p. 685–690.

ALAIZ-MORETON, H.; AVELEIRA-MATA, J.; ONDICOL-GARCIA, J.; MUÑOZ-CASTAÑEDA, A. L.; GARCÍA, I.; BENAVIDES, C. Multiclass classification procedure for detecting attacks on mqtt-iot protocol. **Complexity**, Hindawi, v. 2019, 2019.

ASHTON, K. *et al.* That 'internet of things' thing. **RFID journal**, v. 22, n. 7, p. 97–114, 2009.

BACE, R. G.; MELL, P. **Intrusion detection systems**. [*S. l.*]: US Department of Commerce, Technology Administration, National Institute of . . . , 2001.

Beyond Trust. **IoT Bots Cause Massive Internet Outage October 21st, 2016**. 2016. Disponível em: https://www.beyondtrust.com/blog/entry/iot-bots-cause-october-21st-2016-massive-internet-outage. Acesso em: 27 jan. 2021.

Bhat, S.; Singh, S. One-class support vector machine for data streams. In: **2020 IEEE REGION 10 CONFERENCE (TENCON)**. [*S. l.*: *s. n.*], 2020. p. 1130–1135.

BLUM, A. On-line algorithms in machine learning. In: **Online algorithms**. [*S. l.*]: Springer, 1998. p. 306–325.

CIKLABAKKAL, E.; DONMEZ, A.; ERDEMIR, M.; SUREN, E.; YILMAZ, M. K.; ANGIN, P. Artemis: an intrusion detection system for mqtt attacks in internet of things. In: IEEE. **2019 38th Symposium on Reliable Distributed Systems (SRDS)**. [*S. l.*], 2019. p. 369–3692.

Cisco Systems, Inc. **IEEE 802.11ax: The Sixth Generation of Wi-Fi White Paper**. 2020. Disponível em: https://www.cisco.com/c/en/us/products/collateral/wireless/white-paper-c11-740788.html. Acesso em: 16 fev. 2021.

DIETTERICH, T. G. Machine learning. In: ____. **Encyclopedia of Computer Science**. GBR: John Wiley and Sons Ltd., 2003. p. 1056–1059. ISBN 0470864125.

DING, Z.; FEI, M. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. **IFAC Proceedings Volumes**, Elsevier, v. 46, n. 20, p. 12–17, 2013.

DING, Z.; FEI, M. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. **IFAC Proceedings Volumes**, v. 46, n. 20, p. 12–17, 2013. ISSN 1474-6670. 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013. Disponível em: https://www.sciencedirect.com/science/article/pii/S1474667016314999.

ELSHOUSH, H. T.; OSMAN, I. M. Alert correlation in collaborative intelligent intrusion detection systems—a survey. **Applied Soft Computing**, Elsevier, v. 11, n. 7, p. 4349–4365, 2011.

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**, IEEE, v. 7, n. 8, p. 6882–6897, 2020.

Gartner. **Gartner Says 5.8 Billion Enterprise and Automotive IoT Endpoints Will Be in Use in 2020**. 2019. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io. Acesso em: 27 jan. 2021.

Gendreau, A. A.; Moorman, M. Survey of intrusion detection systems towards an end to end secure internet of things. In: **2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)**. [*S. l.*: *s. n.*], 2016. p. 84–90.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. **Future generation computer systems**, Elsevier, v. 29, n. 7, p. 1645–1660, 2013.

HASAN, M.; ISLAM, M. M.; ZARIF, M. I. I.; HASHEM, M. Attack and anomaly detection in iot sensors in iot sites using machine learning approaches. **Internet of Things**, Elsevier, v. 7, p. 100059, 2019.

HINDY, H.; BAYNE, E.; BURES, M.; ATKINSON, R.; TACHTATZIS, C.; BELLEKENS, X. Machine learning based iot intrusion detection system: An mqtt case study. **arXiv preprint arXiv:2006.15340**, 2020.

HiveMQ. **Introducing the MQTT Protocol - MQTT Essentials: Part 1**. 2015. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/. Acesso em: 17 fev. 2021.

HiveMQ. **Publish & Subscribe - MQTT Essentials: Part 2**. 2015. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/. Acesso em: 17 fev. 2021.

HiveMQ. **MQTT Topics & Best Practices - MQTT Essentials: Part 5**. 2019. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/. Acesso em: 17 fev. 2021.

HOANG, X. D.; HU, J.; BERTOK, P. A program-based anomaly intrusion detection scheme using multiple detection engines and fuzzy inference. **Journal of Network and Computer Applications**, Elsevier, v. 32, n. 6, p. 1219–1228, 2009.

HOENS, T. R.; POLIKAR, R.; CHAWLA, N. V. Learning from streaming data with concept drift and imbalance: an overview. **Progress in Artificial Intelligence**, Springer, v. 1, n. 1, p. 89–101, 2012.

HOI, S. C. H.; SAHOO, D.; LU, J.; ZHAO, P. Online learning: A comprehensive survey. **CoRR**, abs/1802.02871, 2018. Disponível em: http://arxiv.org/abs/1802.02871.

IBM. **What is machine learning?** 2020. Disponível em: https://www.ibm.com/cloud/learn/machine-learning. Acesso em: 27 mar. 2021.

INSTITUTO DE ENGENHEIROS ELETRICISTAS e ELETRôNICOS (IEEE). **Towards a Definition of the Internet of Things (IoT)**. 2015. Disponível em: https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf. Acesso em: 15 fev. 2021.

KASINATHAN, P.; COSTAMAGNA, G.; KHALEEL, H.; PASTRONE, C.; SPIRITO, M. A. An ids framework for internet of things empowered by 6lowpan. In: **Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security**. [*S. l.*: *s. n.*], 2013. p. 1337–1340.

Kasinathan, P.; Pastrone, C.; Spirito, M. A.; Vinkovits, M. Denial-of-service detection in 6lowpan based internet of things. In: **2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)**. [*S. l.*: *s. n.*], 2013. p. 600–607.

Li, H.; Ota, K.; Dong, M. Learning iot in edge: Deep learning for the internet of things with edge computing. **IEEE Network**, v. 32, n. 1, p. 96–101, 2018.

LIAO, H.-J.; LIN, C.-H. R.; LIN, Y.-C.; TUNG, K.-Y. Intrusion detection system: A comprehensive review. **Journal of Network and Computer Applications**, Elsevier, v. 36, n. 1, p. 16–24, 2013.

Liu, F. T.; Ting, K. M.; Zhou, Z. Isolation forest. In: **2008 Eighth IEEE International Conference on Data Mining**. [*S. l.*: *s. n.*], 2008. p. 413–422.

MADAKAM, S.; LAKE, V.; LAKE, V.; LAKE, V. *et al.* Internet of things (iot): A literature review. **Journal of Computer and Communications**, Scientific Research Publishing, v. 3, n. 05, p. 164, 2015.

Moyer, S. Iot sensors and actuators. **IEEE Internet of Things Magazine**, v. 2, n. 3, p. 10–10, 2019.

MQTT Community Wiki. **Differences between 3.1.0 and 3.1.1**. 2020. Disponível em: https://github.com/mqtt/mqtt.org/wiki/Differences-between-3.1.0-and-3.1.1. Acesso em: 17 fev. 2021.

NARKHEDE, S. **Understanding AUC - ROC Curve**. 2018. Disponível em: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5. Acesso em: 28 mar. 2021.

OASIS. **MQTT Version 3.1.1 - OASIS Standard**. 2014. Disponível em: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Acesso em: 17 fev. 2021.

OASIS. **OASIS Message Queuing Telemetry Transport (MQTT) TC** . 2016. Disponível em: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt. Acesso em: 17 fev. 2021.

O'Callaghan, L.; Mishra, N.; Meyerson, A.; Guha, S.; Motwani, R. Streaming-data algorithms for high-quality clustering. In: **Proceedings 18th International Conference on Data Engineering**. [*S. l.*: *s. n.*], 2002. p. 685–694.

PAHL, M.-O.; AUBET, F.-X. All eyes on you: Distributed multi-dimensional iot microservice anomaly detection. In: IEEE. **2018 14th International Conference on Network and Service Management (CNSM)**. [*S. l.*], 2018. p. 72–80.

Palo Alto Networks. **2020 Unit 42 IoT Threat Report**. 2020. Disponível em: https://start.paloaltonetworks.com/unit-42-iot-threat-report. Acesso em: 27 jan. 2021.

PATEL, A.; TAGHAVI, M.; BAKHTIYARI, K.; JÚNIOR, J. C. An intrusion detection and prevention system in cloud computing: A systematic review. **Journal of network and computer applications**, Elsevier, v. 36, n. 1, p. 25–41, 2013.

SCARFONE, K.; MELL, P. Guide to intrusion detection and prevention systems (idps). **NIST special publication**, v. 800, n. 2007, p. 94, 2007.

SHALAGINOV, A.; SEMENIUTA, O.; ALAZAB, M. Meml: resource-aware mqtt-based machine learning for network attacks detection on iot edge devices. In: **Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion**. [*S. l.*: *s. n.*], 2019. p. 123–128.

TAN, S. C.; TING, K.; LIU, F. Fast anomaly detection for streaming data. In: **IJCAI**. [*S. l.*: *s. n.*], 2011.

TAN, S. C.; TING, K. M.; LIU, T. F. Fast anomaly detection for streaming data. In: **Twenty-Second International Joint Conference on Artificial Intelligence**. [*S. l.*: *s. n.*], 2011.

VACCARI, I.; CHIOLA, G.; AIELLO, M.; MONGELLI, M.; CAMBIASO, E. Mqttset, a new dataset for machine learning techniques on mqtt. **Sensors**, v. 20, n. 22, 2020. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/20/22/6578.

YILMAZ, S. F.; KOZAT, S. S. Pysad: A streaming anomaly detection framework in python. **arXiv preprint arXiv:2009.02572**, 2020.

Yu, W.; Liang, F.; He, X.; Hatcher, W. G.; Lu, C.; Lin, J.; Yang, X. A survey on the edge computing for the internet of things. **IEEE Access**, v. 6, p. 6900–6919, 2018.

ZD Net. **CCTV cameras worldwide used in DDoS attacks**. 2015. Disponível em: https://www.zdnet.com/article/cctv-cameras-worldwide-used-in-ddos-attacks/. Acesso em: 02 fev. 2021.

ŽLIOBAITĖ, I.; BIFET, A.; PFAHRINGER, B.; HOLMES, G. Active learning with drifting streaming data. **IEEE transactions on neural networks and learning systems**, IEEE, v. 25, n. 1, p. 27–39, 2013.

APÊNDICE A – QUADRO DE FEATURES CONTIDAS NOS DATASETS DOS EXPERIMENTOS

Quadro 5 – *Features* contidas nos *datasets*.

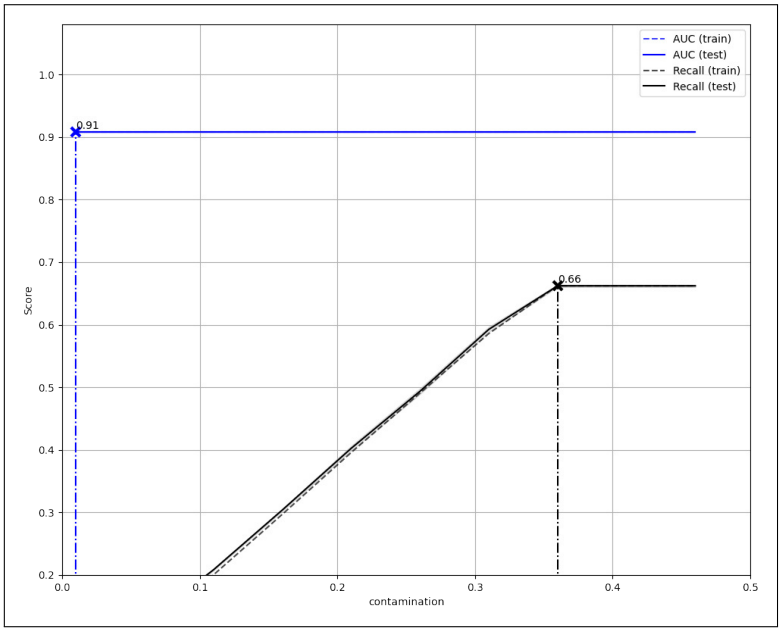
Feature	**Tipo**	**Descrição**
tcp.flags	numérica	*Flags* do pacote TCP.
tcp.time_delta	numérica	Tempo entre um pacote e outro.
tcp.len	numérica	Tamanho do pacote TCP.
mqtt.conack.flags	categórica	*Flags* de resposta de conexão.
mqtt.conack.flags.reserved	numérica	*Flag* de conteúdo reservado na resposta.
mqtt.conack.flags.sp	numérica	*Flag* para definir se há sessão presente.
mqtt.conack.val	numérica	Valor da resposta de conexão.
mqtt.conflag.cleansess	numérica	*Flag* define a escolha de limpar a sessão ou manter.
mqtt.conflag.passwd	numérica	*Flag* define se tem senha.
mqtt.conflag.qos	numérica	Define o QoS da conexão.
mqtt.conflag.reserved	numérica	*Flag* de conteúdo reservado nas mensagens.
mqtt.conflag.retain	numérica	*Flag* define se tem *retain*.
mqtt.conflag.uname	numérica	*Flag* define se tem nome de usuário.
mqtt.conflag.willflag	numérica	*Flag* define se tem *Will Message*.
mqtt.conflags	categórica	*Flags* de conexão.
mqtt.dupflag	numérica	*Flag* de mensagem duplicada.
mqtt.hdrflags	categórica	*Flags* contidas no cabeçalho da mensagem.
mqtt.kalive	numérica	Número em segundos para manter conexão ativa, caso não haja interação.
mqtt.msg	categórica	Conteúdo da mensagem.
mqtt.len	numérica	Tamanho da mensagem.
mqtt.msgid	numérica	Identificador da mensagem.
mqtt.msgtype	numérica	Tipo de mensagem.
mqtt.protoname	categórica	Nome do protocolo utilizado.
mqtt.proto_len	numérica	Tamanho do *Protoname*.
mqtt.qos	numérica	QoS da comunicação.
mqtt.retain	numérica	*Flag* para solicitação de retenção de mensagem.
mqtt.sub.qos	numérica	Nível de QoS requisitado na inscrição.
mqtt.suback.qos	numérica	Nível de QoS obtido na inscrição.
mqtt.ver	numérica	Versão do Protocolo MQTT utilizada.
mqtt.willmsg	categórica	Mensagem enviada aos clientes na desconexão do dispositivo.
mqtt.willmsg_len	numérica	Tamanho da *Will Message*.
mqtt.willtopic	categórica	Tópico utilizado para envio da *Will Message*.
mqtt.willtopic_len	numérica	Tamanho do *Will Topic*.
target	categórica	Define o tipo da instância, normal ou tipo de ataque.

Fonte: Elaborado pelo autor.

APÊNDICE B – SELEÇÃO DE PARÂMETROS ISOLATION FOREST EXPERIMENTO MQTTSET

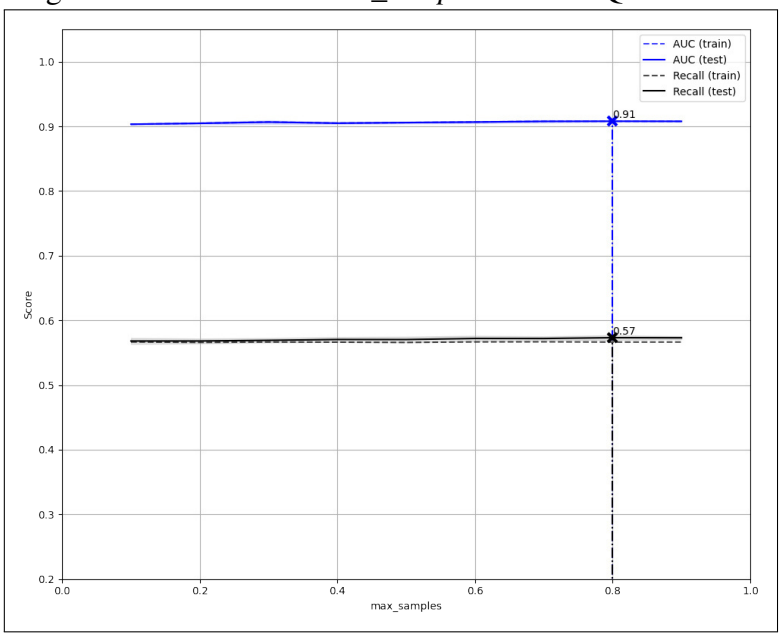
As Figuras 13 e 14 apresentam os resultados obtidos a partir de uma série de execuções e validações para definir os melhores parâmetros de contaminação e *samples* respectivamente.

Figura 13 – Parâmetro *contamination* Rede MQTTset.



Fonte: Elaborado pelo autor.

Figura 14 – Parâmetro *max_samples* Rede MQTTset..



Fonte: Elaborado pelo autor.