Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

INTELIGÊNCIA ARTIFICIAL APLICADA À DETEÇÃO DE INCIDENTES DE SEGURANÇA EM REDES IOT

TIÉZER COSTA DE MELO

Leiria, Março de 2023

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

INTELIGÊNCIA ARTIFICIAL APLICADA À DETEÇÃO DE INCIDENTES DE SEGURANÇA EM REDES IOT

TIÉZER COSTA DE MELO
Número: 2200175

Dissertação realizada sob orientação do Professor Doutor Carlos Manuel da Silva Rabadão (carlos.rabadao@ipleiria.pt), do Professor Doutor Leonel Filipe Simões Santos (leonel.santos@ipleiria.pt) e do Professor Doutor Rogério Luís de Carvalho Costa (rogerio.l.costa@ipleiria.pt).

Leiria, Março de 2023

AGRADECIMENTOS

Primeiramente, quero gostaria de agradecer ao meu Deus, que me sustentou em todos os momentos de dificuldade e sempre se manteve fiel.

Gostaria de agradecer também aos meus pais, Cláudia e Gilberto, que são os melhores pais do mundo, que não mediram esforços para me proporcionar a realização de um sonho e sempre que necessário, traziam palavras de conforto. Também, a minha avó, que todos os dias demonstra o que é o amor em forma de pessoa. Agradecer também a Kelen, que mesmo com a distância, sempre se manteve presente, trazendo palavras de incentivo e algumas verdades, quando necessário.

Agradeço aos meus orientadores, Carlos, Leonel e Rogério, pela extrema paciência e solicitude em meio a inúmeros questionamentos e curiosidades e por todo os ensinamentos ao longo deste trabalho.

Aos meus colegas, Rafaela e António, e ao meu professor Professor Doutor Miguel Frade, agradeço pela oportunidade de ter participado de inúmeros momentos inusitados em meio às aulas e trabalhos académicos. Aos meus colegas Rafael Ascensão e João Cardo pelo coleguismo nas atividades profissionais.

Por fim, mas não menos importante, agradeço também aos meus amigos e colegas por, de alguma forma, terem participado desta pequena e desafiadora fase da minha vida e terem contribuído para a conclusão dela.

RESUMO

Internet of Thing ou IoT são dispositivos de limitado poder computacional, interconectados através da internet ou outra rede de comunicação, que partilham informação entre si e atuam de forma autónoma com uma mínima intervenção humana. Devido a algumas destas características, eles têm sido utilizados em diversas áreas da sociedade. Porém, apesar dos diversos benefícios trazidos por este tipo de dispositivos, estes apresentam alguns problemas de segurança. Tais problemas surgem devido à sua menor capacidade computacional, que impede a aplicação de técnicas de proteção mais complexas, e à grande diversidade ou heterogeneidade de tecnologias utilizadas (*hardware*, protocolos etc.). Como alternativa, técnicas de *machine learning* (ML) tem sido aplicadas como forma de melhorar a capacidade de deteção de ataques e tráfego anómalo. Neste trabalho foram criados dois *datasets* com intuito de representar os serviços de uma *smart greenhouse* e um conjunto de apartamentos que utilizam sistemas inteligentes de controlo. Os *datasets*, que representam o tráfego de dados destas duas redes IoT, são compostos pelos protocolos CoAP e MQTT. Foi realizada a revisão e a análise das ferramentas de simulação e geração de tráfego IoT, onde, através de comparação das características, foram selecionadas as ferrametas Contiki e Netsim. As simulações foram executadas através destas duas ferramentas e, além do tráfego normal, foram simulados 6 diferentes ataques, cuja maior parte destes estava relacionado com o protocolo RPL. Aos dados destes *datasets*, foram aplicados modelos de aprendizagem de máquina com o intuito de identificar os ataques utilizados, onde foi obtido um alto índice de acerto no que se refere à classificação do tráfego malicioso.

ABSTRACT

Internet of Thing or IoT are devices with limited computational power, interconnected via the internet or another communication network, which share information with each other and act autonomously with minimal human intervention. Due to some of these characteristics, they have been used in different areas of society. However, despite the many benefits brought by this type of devices, they have some security problems. Such problems arise due to its lower computational capacity, which prevents the application of more complex protection techniques, and to the great diversity or heterogeneity of technologies used (hardware, protocols, etc.). As an alternative, machine learning (ML) techniques have been applied as a way to improve the ability to detect attacks and anomalous traffic. In this work, two datasets were created in order to represent the services of a smart greenhouse and a set of apartments that use intelligent control systems. The datasets, which represent the data traffic of these two IoT networks, are composed of the CoAP and MQTT protocols. A review and analysis of the IoT simulation and traffic generation tools was carried out, where, by comparing the characteristics, the Contiki and Netsim tools were selected. The simulations were performed using these two tools and, in addition to normal traffic, 6 different attacks were simulated, most of which were related to the RPL protocol. Machine learning models were applied to the data from these datasets in order to identify the attacks used, where a high success rate was obtained regarding the classification of malicious traffic.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

6LowPAN	IPv6 over Low-Power Wireless Personal Area Networks.
AIDS	Anomaly-based Intrusion detection systems.
AMQP	Advanced Message Queuing Protocol.
API	Application Programming Interface.
APT	advanced persistent threat.
AWS	Amazon Web Services.
BLE	Bluetooth Low Energy.
BSD	Berkeley Source Distribution.
CIDS	Colaborative Intrusion detection systems.
CO2	Dióxido de Carbono.
CoAP	Constrained Application Protocol.
CPU	Central Processing Unit.
CSV	Comma-separated values.
DDoS	Distributed Denial of Service.
DL	Deep Learning.
DNS	Domain Name System.
DoS	Denial of Service.
DTC	Decision Tree Classifier.
FN	false negative.
FP	false positive.

GA	Genetic Algorithm.
GPRS	General Packet Radio Service.
GPS	Global Positioning System.
GSM	Global System for Mobile communication.
GUI	graphical user interface.
HIDS	Host-based intrusion detection system.
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol Secure.
IA	Inteligência Artificial.
IBM	International Business Machines Corporation.
IDGF	IoT Dataset Generation Framework.
IDS	Intrusion detection systems.
IETF	The Internet Engineering Task Force.
IIoT	Industrial internet of things.
IOT	Internet of Things.
IP	Internet Protocol.
IPFIX	Internet Protocol Flow Information Export.
IPv4	Internet Protocol versão 4.
IPv6	Internet Protocol versão 6.
JSON	JavaScript Object Notation.
KNN	K-neighbors Classifier.
LAN	Local Area Networks.
LPWAN	Low-power, wide-area network.
LP-WAN	Low-power, wide-area network.

LRC	Logistic Regression Classifier.
LTE	Long Term Evolution.
MITM	Man-in-the-middle attack.
ML	Machine Learning.
MQTT	Message Queuing Telemetry Transport.
MQTT-SN	MQTT for Sensor Networks.
NAT	Network address translation.
NFC	Near field communication.
NIDS	Network-based intrusion detection system.
NoSQL	Non SQL ou not only SQL.
OSI	Open Systems Interconnection.
PCAP	Packet Capture.
pH	Potential of hydrogen.
PKI	Public key infrastructure.
QUIC	Quick UDP Internet Connections.
RFC	Random Forest Classifier.
RFID	Radio-frequency identification.
RNA	Redes Neuronais Artificiais.
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks.
RSSF	Redes de Sensores sem-fio.
SIDS	Signature-based Intrusion detection systems.
SO	Sistema Operativo.

TCP	Transmission Control Protocol.
TIC	Tecnologia da Informação e Comunicação.
TLS	Transport Layer Security.
TN	true negative.
TP	true positive.
UART	universal asynchronous receiver / transmitter.
UDP	User Datagram Protocol.
UWB	Ultra-wideband.
WAN	Wide Area Netwoks.
WSN	wireless sensor networks.
XML	extensible markup language.
XMPP	Extensible Messaging and Presence Protocol.
YAF	Yet Another Flowmeter.

1

INTRODUÇÃO

A Internet das Coisas (*Internet of Things* ou IoT) desempenha um papel vital na sociedade atual, isso devido ao fato de poder ser empregada nas mais diversas áreas da sociedade, tais destacam-se: indústria; agricultura; casas, carros e cidades inteligentes; *wearables*, entre outras (Mohanta et al., 2020; Sain et al., 2017). Globalmente, o número deste tipo de dispositivos já ultrapassa a casa dos bilhões e esse número aumenta a cada ano (Balaji et al., 2019; Al-Garadi et al., 2020). Ao longo dos últimos anos e devido a sua diversificada aplicabilidade, o tema IoT tem sido um tema de grande interesse para a indústria, sendo também abordado num grande número de pesquisas académicas.

A IoT é composta por dispositivos e sensores inteligentes interligados a partir de uma rede de comunicação, com e sem-fio, local ou via internet, e que geram grandes volumes de dados em tempo real. Estes dispositivos ou infraestruturas têm como característica atuar de forma automatizada e, muitas vezes, coordenada, com a mínima intervenção humana. Eles comunicam-se entre si, partilhando informação, e colaboram para a execução de ações e monitorização diversas, como realizar o monitoramento da rede de uma organização para detetar a realização de ataques dentro desta rede. Eles ainda podem realizar alguma ação inteligente com base em alguma regra anteriormente definida e contribuir para a tomada de decisão (Hameed e Alomary, 2019).

Estes dispositivos possibilitam a integração de uma variedade de elementos de forma com que estes partilhem informação e atuem de forma conjunta para um bem comum. Através da integração destes dispositivos no cotidiano das pessoas e na sociedade, foi possível coletar e analisar, com maior facilidade e de forma ativa, os dados referentes a processos e serviços, o que permitiu uma melhor tomada de decisão, assim como trazer inovações e novas funcionalidades.

Devido a algumas características dos dispositivos IoT, existem algumas limitações quanto a aplicabilidade de controlos de segurança mais complexos, a exemplo dos algoritmos de criptografia mais comumente utilizados (Hameed e Alomary, 2019). Meneghello et al. (2019) apresenta que os principais fatores que contribuem para esta limitação são as características destes dispositivos, a saber, o reduzido poder

computacional e capacidade energética e a heterogeneidade de tecnologias, tais como configuração de *hardware*, arquiteturas e protocolos de comunicação. Hassan et al. (2019) acrescenta ainda a escala ou número de dispositivos como uma destas limitações. O amplo espectro de configurações eleva a possibilidade da existência de vulnerabilidades e má-configurações, que possibilitam que atacantes ou *botnets* utilizem destas falhas e executem ações indevidas ou criminosas

Apesar das suas características benéficas para a sociedade, este tipo de dispositivo está também exposto a ameaças e a ataques dos mais diversos tipos e são potenciais vetores de entrada para ameaças caso não sejam aplicados os devidos controlos de segurança. Os ataques não estão limitados apenas à confidencialidade e à integridade da informação, mas também à disponibilidade dos dispositivos e da comunicação da infraestrutura. Há alguns anos, um grande número de dispositivos IoT foram comprometidos por alguns *malwares*, a exemplo do Mirai, Hajime e Reaper (Kolias et al., 2017). Após o seu comprometimento, estes dispositivos passaram a fazer parte de uma rede zumbi, ou *botnet*, e passaram também a ser controlados remotamente para a realização de ataques de negação de serviço em larga escala a grandes organizações.

Como destacado, os ambientes e redes IoT possuem algumas limitações computacionais e de segurança. Devido a isso, pesquisadores e profissionais tentam identificar alternativas para melhorar a performance dos controlos de proteção e criar soluções mais eficientes (Ibitoye et al., 2019). Neste sentido, técnicas relacionadas com inteligência artificial, principalmente as técnicas de *machine learning* (ML) e *deep learning* (DL), têm sido desenvolvidas e aplicadas na IoT. Ambas as técnicas apresentam considerável eficiência na melhora da capacidade de deteção de anomalias e ataques.

É de destacar que tais conjuntos de dados terão um impacto direto na capacidade de identificação de padrões do algoritmo de aprendizagem (Buczak e Guven, 2015). Por isso, a composição do *dataset* é fundamental para treinar um modelo de detecção, assim como para avaliar a eficiência da capacidade de deteção deste modelo (Al-Hadhrami e Hussain, 2020). Porém, Al-Hadhrami e Hussain (2020) apresentam ainda que muitos dos *datasets* existentes são desatualizados ou são compostos por dados que podem não representar um cenário do mundo real, além de não representarem uma diversidade muito grande de ataques. Booij et al. (2021) afirma que muitos dos *datasets* existentes não possuem muita heterogeneidade nos dados que o compõem e poucos são designados para deteção de anomalias em redes IoT. Ele afirma ainda que a heterogeneidade dos dados do *dataset* podem ter um grande impacto no desempenho da técnica de deteção.

1.1 OBJETIVOS E CONTRIBUIÇÕES

Com o rápido crescimento dos ataques de IoT em grande escala e o impacto negativo causado por eles, é importante desenvolver sistemas, técnicas e tecnologias que possam ser utilizadas como medida de segurança e utilizadas na identificação e mitigação de ataques, vulnerabilidades e ameaças. Estas tecnologias, a exemplo de técnicas de deteção baseadas em aprendizado de máquina, devem estar de acordo com as limitações características deste tipo de dispositivos, mas ainda apresentar eficiência e boa capacidade de deteção. Também, de forma a suprir uma carência da atualidade, há a necessidade de desenvolver *datasets* que sejam compostos com grande quantidade e variedade de dados, com diferentes dispositivos, ataques e protocolos.

Com base na informações supracitadas, o atual trabalho tem como objetivo explorar técnicas de ML que possam ser aplicadas à deteção de incidentes de segurança em redes IoT. Primeiramente, pretende-se desenvolver um *dataset* que possa ser utilizado para o treinamento de sistemas de deteção que sejam baseados em técnicas de aprendizagem, assim como por qualquer outra técnica ou metodologia aplicável. Tal *dataset* terá como característica principal a heterogeneidade dos dados que o compõe, de forma a preencher uma lacuna existente nesta área. Ainda, serão utilizados alguns algoritmos de *machine learning* para demonstrar a utilidade dos *datasets* criados e criar uma base de resultados para serem utilizados como comparação para trabalhos futuros. Espera-se que tanto os métodos quanto o *dataset* desenvolvidos sirvam como contributo para as áreas académica e empresarial. Por fim, realizar a revisão e a avaliação das ferramentas que são utilizadas na geração das simulações e do tráfego de dados IoT.

1.2 ESTRUTURA DO TRABALHO

Este documento está estruturado da seguinte forma: no Capítulo 2, é realizada uma apresentação dos conceitos importantes para o atual trabalho, assim como a uma análise do que existe na literatura sobre os conceitos correlatos. No Capítulo 3, são apresentadas algumas das ferramentas de simulação e geração de tráfego e de fluxo de dados existente e qual as ferramentas selecionadas para compor o atual trabalho. O ambiente que servirá como base para as simulações e as características da simulação, tais temporização, número de sensores, ataques selecionados, são apresentadas no Capítulo 4. O resumo do desenvolvimento do trabalho e características dos

datasets criado são apresentados no Capítulo 5. A criação e a aplicação dos modelos de *machine learning* para identificação de tráfego malicioso sobre os *datasets* são realizadas no Capítulo 6. Por fim, no Capítulo 7, são apresentados os pontos de fechamento do trabalho, tal como sugestões para trabalhos futuros, contributos e conclusão.

2

BACKGROUND

Neste capítulo serão apresentados os conceitos fundamentais para a copreensão do corrente trabalho.

2.1 INTERNET OF THINGS

Nesta secção, serão apresentados alguns dos conceitos relacionados com a Internet das coisas, tais como sua definição, arquiteturas, protocolos, ataques mais comuns, entre outros.

2.1.1 *Caracterização de Internet of Things*

O conceito de IoT está relacionado com um conjunto de tecnologias que coletivamente compreendem uma rede composta por dispositivos conetados a outros dispositivos e serviços e que atuam de forma autônoma, ou quase, em busca de atender a uma ou mais necessidades específicas. Devido ao cada vez maior número de dispositivos, espera-se que as infraestruturas IoT suportem o dimensionamento ou escalonamento desta grande quantidade de dispositivos. Portanto, uma das características imprescindíveis de uma infraestrutura IoT é a capacidade de ela ser escalável e poder crescer em número de dispositivos, de acordo com as necessidades do projeto.

Na *Internet of Things* cada dispositivo possui um identificador, que pode ser um endereço IP ou outro tipo de identificador, para que ele possa ser identificado unicamente na rede. Estes dispositivos possuem acesso à internet, as vezes dependendo de um intermediário chamado *gateway*, e partilham informações entre si, com os serviços localizados na nuvem e podem realizar interações com pessoas. Os elementos que compõem a infraestrutura IoT permitem novas abordagens e facilitam aspectos do cotidiano ao criar soluções inteligentes, como melhorar a tomada de decisão com base nos dados coletados. Porém, as utilidades surgem também com problemas de segurança e privacidade dos dados.

Ambientes IoT apresentam como características a capacidade de obter dados do mundo real, a conectividade, a partilha de informações, a escalabilidade, o baixo consumo energético e a integração entre diversas tecnologias (Alam et al., 2020; Bansal e Kumar, 2020; Sobin, 2020).

2.1.2 *Componentes*

Alguns dos componentes que compõem uma infraestrutura IoT serão apresentados abaixo (Bansal e Kumar, 2020). Esta composição diferencia-se apenas no que diz respeito a finalidade da aplicação, arquitetura e tecnologias estabelecidas.

- dispositivos: servem como base para sensores e atuadores. Ainda, pode realizar o processamento de parte dos dados coletados pelos sensores sem o intermédio de um sistema externo especializado;
- sensores: coletam informação do mundo físico (transforma dados analógicos em dados digitais);
- atuadores: controlam e acionam ações em dispositivos do mundo físico (transforma sinais digitais em ações físicas);
- rede de comunicação: permite a troca de informação entre os diferentes componentes;
- *gateways* de rede: agrega e converte os dados de alguns tipos de dispositivos para o formato digital de forma que estes dados possam ser encaminhados para a nuvem;
- *data analytics systems*: desempenha o papel de processamento e análise dos dados adquiridos. Lida com grandes volumes de dados e requer grande banda de rede. São aplicadas técnicas de *machine learning* e visualização de dados. Apresenta resultados referentes aos dados coletados e contribui para a tomada de decisão;
- *storage systems*: armazenamento dos dados na nuvem. Os dados do ambiente IoT poderá ser combinado com outras fontes de dados para *insights* mais completos.

Dentre os componentes apresentados acima, *gateways* e sistemas de *data analytics* nem sempre estão presentes nas infraestruturas IoT. No início da sua implementação, os sensores limitavam-se a apenas coletar e armazenar os dados e estes apenas eram utilizados e analisados em momento posterior. Com o avanço tecnológico, os

dispositivos, além de monitorizar e coletar dados do ambiente, passaram também a analisar os dados coletados e, assim, apresentar as informações em tempo real, permitindo uma tomada de decisão mais rápida e eficiente.

Alguns dispositivos utilizam de protocolos que não são possíveis de se conectarem diretamente a nuvem, ou internet, e dependem de dispositivos chamados *gateways* para este fim. As *gateways* realizam a *interface* entre os dispositivos finais e os sistemas que se encontram na nuvem. Sensores que não precisam de *gateways* podem podem partilhar diretamente os dados com outros dispositivos e serviços da nuvem. As *gateways* recebem dados brutos de uma variedade de dispositivos e os encaminha para as camadas *fog* ou *cloud*, onde serão armazenados ou processados e posteriormente poderão ser analisados por sistemas especializados como ML ou *Big Data*. As *gateways* podem, inclusive, realizar o pré-processamento dos dados recebidos dos dispositivos, conceito chamado de *edge computing*, que implica que o processamento e o controle dos dados sejam realizados mais proximamente aos dispositivos fim e não mais na nuvem, diminuindo a latência do processamento em tempo real da infraestrutura.

2.1.3 *Arquiteturas*

A arquitetura de uma infraestrutura define a disposição dos componentes na rede, as possíveis tecnologias e respetivas configurações e formato de dados. A arquitetura pode variar bastante a depender do tipo de ambiente, requisitos, segurança e fins de implementação.

Inicialmente, é necessário elucidar alguns conceitos importantes no que diz respeito ao IoT. Em relação à arquitetura, esta poderá ser composta por três diferentes formas de processar e analisar os dados coletados. As três formas em questão são *cloud computing*, *fog computing* e *edge computing*, que são descritas a seguir:

- *cloud computing*: é o padrão de armazenamento e processamento de dados na IoT. Os dados são armazenados em serviços ou *data centers* na nuvem (de terceiros ou da própria entidade) e que podem ser acedidos pela *internet*. O grande volume de dados é armazenado para depois ser analisado e utilizado para fins estratégicos a partir de *data analytics*;
- *fog computing*: o processamento dos dados é realizado de forma descentralizada e distribuída na rede local onde os dispositivos estão alocados e não na nuvem como acontece na *cloud computing*. Apresenta maior segurança e privacidade

e uma latência menor que a *cloud computing*. Pode haver comunicação direta entre camadas *fog* diferentes;

- *edge computing*: os dados coletados são processados pelos próprios dispositivos de captura das características do ambiente ou nos *gateways* da rede. Isso decorre do avanço tecnológico dos dispositivos, que trouxe maior capacidade de processamento. Essa característica traz maior segurança e melhora o tempo de resposta. Também, implica numa maior capacidade de alterar o comportamento e operação dos dispositivos em tempo real, além de diminuir a quantidade de informação trafegada na rede, diminuir o risco de fuga de dados e diminuir a latência para obtenção da informação crítica que tem que ser em tempo real.

Em relação à arquitetura de sistemas IoT, esta é organizada de acordo com os níveis ou quantidade de camadas que a compõe. Foram identificadas na literatura diferentes representações de arquiteturas, descritas com variadas quantidades de camadas, como sendo de 3 camadas (Li, 2012; Liang e Kim, 2021; Tewari e Gupta, 2020), 4 camadas (Mohanta et al., 2020), 5 camadas (Granjal et al., 2015; Said e Masud, 2013) e 6 camadas.

A representação mais básica da arquitetura IoT é a de 3 camadas. As diferentes camadas da representação de 3 camadas são as seguintes:

- *Perception layer*: também chamada de camada de reconhecimento, é a camada física da arquitetura, composta por dispositivos inteligentes (sensores, atuadores, controladores, etc.), que tem a responsabilidade de detetar e coletar informações sobre o ambiente físico onde estão localizados (umidade, temperatura etc.) e transformá-los em dados digitais. Podem usar tecnologias de curto alcance ou outras tecnologias com e sem-fio ou via satélite;
- *Network layer*: também chamada de *transmission layer*, possibilita a transferência de dados e a comunicação ativa entre redes, sistemas e dispositivos. Pode ocorrer de maneira direta através dos protocolos TCP e UDP ou pelo intermédio de *gateways*, que agem como *link* entre a LAN e a WAN;
- *Application layer*: camada onde os dados são processados e analisados para possibilitar *business intelligence* e *data analytics* ou entregar algum outro tipo de serviço específico ao utilizador.

Na arquitetura de 4 camadas foi adicionada a camada *Support Layer*, que fica localizada entre as camadas *Application Layer* e *Network Layer*. Esta nova camada,

que é baseada em *fog computing*, tem como responsabilidade adicionar fatores de segurança complementares à arquitetura de 3 camadas e serviços baseados em APIs.

A arquitetura de 5 camadas adiciona duas nova camadas: "*Processing Layer*"(também chamada de *Middleware layer*) e "*Business Layer*". A primeira foi adicionada entre as camadas *Application Layer*"e "*Network Layer*", substituindo a camada *Support Layer* da arquitetura de 4 camadas, e a segunda no topo da arquitetura, acima da camada *Application Layer*". As demais camadas, a saber *perception layer*, *network layer* e *application layer*, mantém as mesmas responsabilidades das arquiteturas anteriores. Abaixo, são apresentadas as responsabilidades destas duas camadas:

- *Middleware Layer:* também chamado de *Processing layer*, tem como responsabilidade o armazenamento, a rotulagem, a filtragem, o processamento e a análise dos dados, o que possibilita a tomada de ações e decisão com maior eficiência;
- *Business Layer:* apresenta os dados obtidos para o consumidor final (*flowcharts*, gráficos, análise de resultados etc.). Os dados são validados para serem uteis e aplicados no planejamento estratégico e de negócio e definição de metas.

A arquitetura de 6 camadas adicionou uma nova camada em relação à arquitetura de 5 camadas. Esta nova camada, *Security Layer*, é específica para implementar aspetos relacionados com a segurança da infraestrutura e respetivas comunicações.

A Figura 1 apresenta graficamente as arquiteturas IoT de 3 a 6 camadas.

2.1.4 *Protocolos*

Os protocolos são utilizados para a manutenção da comunicação e a transmissão de dados entre dispositivos, *gateways* e serviços na *nuvem*. Existem diferentes protocolos e eles podem ser utilizados a partir de radiofrequência ou meio cabeado. É de destacar que estes protocolos podem estar localizados em diferentes camadas, a depender da arquitetura adotada.

As redes de sensores sem-fio (RSSF), ou *Wireless Sensor Network* (WSN) em língua inglesa, são as redes mais utilizadas em ambientes IoT e têm sido aplicadas para contribuir com precisão na análise dos dados da produção na agricultura inteligente e *smart cities*. Uma das vantagens das RSSF é a facilidade com que se pode criar uma rede composta por diversos dispositivos, facilitando a comunicação entre eles, o *gateway* e qualquer outro dispositivo ou serviço na nuvem.

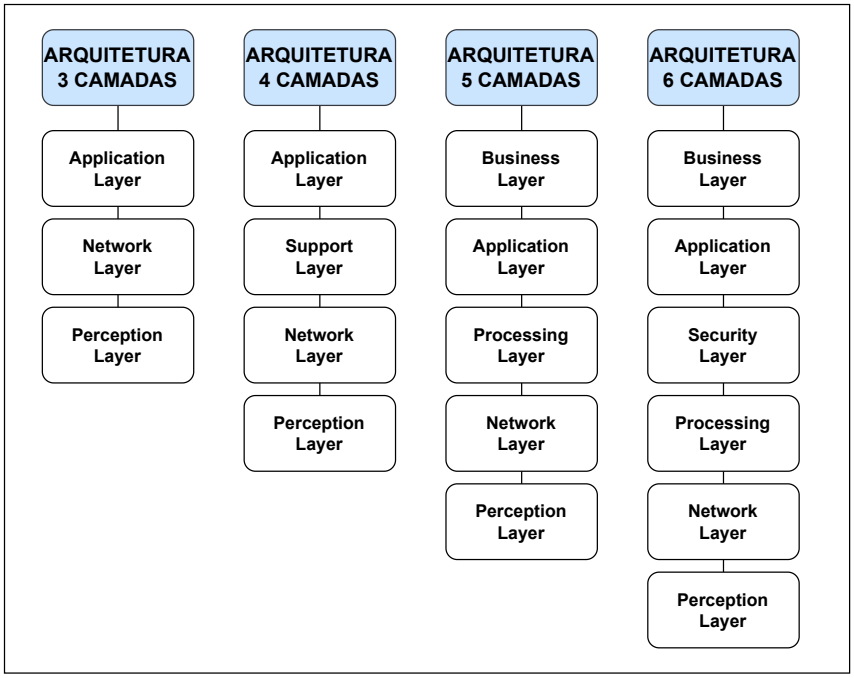


Figura 1: Representação das arquiteturas IoT baseadas em 3, 4, 5 e 6 camadas.

Na *Perception layer*, são encontrados os protocolos que possibilitam a comunicação entre os dispositivos e o ambiente no qual eles estão alocados. Abaixo, são apresentado alguns dos protocolos desta camada:

- Bluetooth Low Energy (BLE);
- Long-term evolution (LTE);
- Near field communication (NFC);
- Power Line Communication (PLC);
- Radio frequency identification (RFID);
- Wi-Fi/802.11;
- Z-Wave;
- Zigbee.

IOT Technologies and protocols: Microsoft Azure (2022) divide os procolos da *Network layer* em 3 camadas distintas (*Data link layer*, *Transport layer* e *Network layer*), o que facilita a compreensão da função de determinado protocolo dentro da pilha de protocolos. A maior parte das literaturas não realiza tal separação, e os protocolos são apresentados como pertencentes apenas à *Network layer*. Abaixo, alguns dos protocolos são apresentados:

- IEEE 802.15.4;
- Low-Power Wide Area Network (LPWAN);
- *Transmission Control Protocol* (TCP);
- *User Datagram Protocol* (UDP);
- *Internet Protocol* (IP);
- *Routing Protocol for Low-Power and Lossy Network* (RPL);
- *IPv6 over Low-Power Wireless Personal Area Networks* (6LoWPAN);

Por fim, são apresentados os protocolos da *Application Layer*, que fornecem uma interface para troca de mensagens entre os utilizadores e os componentes IoT:

- *Advanced Message Queuing Protocol* (AMQP);
- *Constrained Application Protocol* (CoAP);
- *Data Distribution Service* (DDS);
- *Message Queue Telemetry Transport* (MQTT);
- *Extensible Messaging and Presence Protocol* (XMPP).

2.1.5 *Segurança, ameaças e vulnerabilidades*

Devido a algumas das características dos dispositivos IoT, estes apresentam algumas limitações quanto a aplicabilidade de controlos de segurança mais complexos, a exemplo dos algoritmos de criptografia mais comumente utilizados (Hameed e Alomary, 2019). Meneghello et al. (2019) apresenta que os principais fatores que contribuem para esta limitação são as características destes dispositivos, a saber, o reduzido poder computacional, a reduzida capacidade energética e a heterogeneidade de tecnologias, tais como configuração de *hardware*, arquiteturas e protocolos de comunicação. Hassan et al. (2019) acrescenta ainda a escala ou número de dispositivos como uma destas limitações.

Ainda, podem ser acrescentadas a falta de controlos de segurança extras na rede, como IDS e IPS, os problemas físicos nos dispositivos críticos da infraestrutura, o ciclo de vida das baterias e a falta de substituição destas e a falta de aplicação de controlos que monitorem acessos indevidos e garantam a segurança do ambiente físico no qual estão localizados os componentes da infraestrutura IoT.

Por fim, podem ser citados outros fatores como protocolos e *firmware* proprietários (dificuldade de acesso ao código fonte), a existência de *bugs* e vulnerabilidades no desenvolvimento do *firmware* e a falta de atualização e *patches* de segurança para correção de vulnerabilidades.

Esta grande quantidade de diferentes combinações de configurações e características aumenta a possibilidade de ocorrências de vulnerabilidades e má-configurações (não aplicação de criptografia, tráfego de dados em *clear text* e não modificação da configuração e senhas *default*), que possibilitam que atacantes ou *bots* utilizem destas falhas e executem ações indevidas ou criminosas. Como os dispositivos são produzidos em massa, uma vulnerabilidade encontrada poderá também estar presente em um grande número de dispositivos de mesma marca e modelo.

Por decorrência do notável número de vulnerabilidades e ataques, foi possível observar que controlos de segurança são imprescindíveis para o futuro do IoT. Devido a isso, existem pesquisas que buscam encontrar alternativas que mitiguem essas vulnerabilidades e aumentem a segurança do ambiente IoT, devendo abranger todos os seus componentes (dispositivos, sistemas na nuvem e meios de comunicação). Quanto mais crítica é a informação, maiores devem ser os controlos aplicados para garantir a segurança da informação.

2.1.6 *Ataques*

Como foi apresentado, ambientes IoT são frequentemente alvos de ataques dos mais diversos tipos. Estes ataques comprometem os princípios de segurança da informação e não estão limitados apenas à confidencialidade e à integridade da informação, mas também à disponibilidade dos dispositivos e dados da infraestrutura. Há alguns anos, um grande número de dispositivos IoT foram comprometidos por alguns *malwares*, a exemplo do Mirai, Hajime e Reaper (Kolias et al., 2017). Após o seu comprometimento, estes dispositivos passaram a fazer parte de uma rede zumbi, ou *botnet*, e passaram também a ser controlados remotamente para a realização de ataques de negação de serviço em larga escala a grandes organizações.

Diversos trabalhos abordam esse tema, apresentando as características e camadas afetadas pelos respetivos ataques, assim como mecanismos de controlo ou mitigação (Hassan et al., 2019; Hassija et al., 2019; Liang e Kim, 2021; Mohanta et al., 2020; Tahsien et al., 2020; Xiao et al., 2018).

Alguns exemplos de ataques encontrados em ambientes IoT são apresentados na lista abaixo.

- *Fake Node Attack* - ocorre quando um invasor adiciona um nó a um sistema projetado para inserir dados falsos. Esse ataque visa impedir que o nó transmita informações reais, consuma energia de nós autênticos e, potencialmente, destrua a rede.
- *Replay Attack* - também conhecido como "ataque de reprodução", é quando um invasor escuta uma conversa entre o remetente e o destinatário e rouba informações do remetente. Em seguida, enviam essas informações à vítima na tentativa de provar sua autenticidade ou como "prova" de sua identidade. Depois de assumirem a identidade do remetente real, eles podem motivar o destinatário a realizar qualquer número de ações.
- *Denial-of-Service* (DoS): O ataque de negação de serviço tem o objetivo de tornar os serviços de um sistema ou dispositivo indisponíveis ou fazer com que o funcionamento destes sejam interrompidos por um período de tempo. O ataque consiste em sobrecarregar a capacidade dos dispositivos através de inúmeras requisições, fazendo com que novas requisições não possam ser atendidas uma vez que o serviço encontra-se sobrecarregado e indisponível;
- *Distributed Denial-of-Service* (DDoS): Tem o mesmo funcionamento do ataque DoS, porem é realizado de forma distribuída.;
- *Man-in-the-middle* (MiTM): os ataques MiTM ocorrem quando um terceiro intercepta e, em seguida, altera as comunicações entre um receptor e um remetente, alterando as mensagens para atender às suas próprias necessidades. Isso significa uma grande violação de segurança, pois permite que o invasor manipule informações em tempo real.
- *Sniffing attack*: referem-se ao roubo ou interceptação de dados capturando o tráfego de rede usando um farejador de pacotes.
- *Code Injection Attacks*: Este é um ataque realizado através da injeção de código ou *scripts* do lado do cliente em um site confiável. Isso permite que o invasor insira código malicioso que poderá ser executado pelos demais utilizadores;
- *Database Injection Attacks*: Este ataque é realizado a partir da injeção de comandos SQL em *inputs* que não realizam o tratamento dos dados de entrada e possibilitam que o atacante possa realizar operações diretamente na base de dados do site ou aplicação.

- *Buffer overflow*: Este ataque consiste em sobrescrever fragmentos de memória de um processo com mais dados do que o tamanho suportado pelo *buffer* e, dessa forma, aceder áreas de memória subsequentes e fora dos limites do bloco alocado pelo processo, podendo corromper dados, travar programas e causar a execução de código malicioso;
- *Blackhole*: Neste ataque, o dispositivo malicioso descarta todos os pacotes que chegam a ele, cujos pacotes deveriam ser encaminhados para outros dispositivos. Pode ser considerado um negação de serviço uma vez que zero pacotes que chegam ao dispositivo são retransmitidos para os dispositivos vizinhos;
- *DIO Supression*: neste ataque, o dispositivo malicioso transmite a todos os nos próximos repetidas mensagens DIO, fazendo com que os demais dispositivos suprimam o envio de suas próprias mensagens DIO. Como consequência, os dispositivos e rotas podem ser afetadas;
- *Sinkhole*: neste ataque, o dispositivo malicioso atrai o tráfego dos dispositivos próximos para si ao encaminhar mensagens para estes dispositivos com informações que o referenciem como rota para outro conjunto de dispositivos (rotas falsas). O objetivo deste ataque é justamente atrair todo os dados para si e então decidir o que fazer com os dados que chegam, podendo alterá-los ou eliminá-los;
- *Hello Flood*: o dispositivo atacante encaminha mensagens DIO HELLO para o máximo de dispositivos vizinhos, o que faz com que novas mensagens DIO sejam encaminhadas na rede a cada novo envio, aumentando o tráfego na rede;
- *Version Number*: neste ataque, um dispositivo malicioso altera o seu próprio *rank*, fazendo com que os demais dispositivos tenham de recalcular os seus próprios *ranks* e, posteriormente, propagar esta atualização para a rede.

2.1.7 *Smart Farming*

Conforme previsão de pesquisas de entidades governamentais e comunidade académica, há uma projeção de que haverá um considerável aumento populacional a nível mundial nas próximas décadas. Este aumento terá como consequência uma maior necessidade de produção de alimentos como consequência do aumento na demanda. Como forma de vencer os não tão futuros desafios na produção alimentícia, faz-se

necessário encontrar alternativas que proporcionem uma produção mais sustentável e eficiente de alimentos do que é encontrado na agricultura convencional, que é a metodologia que não utiliza de muita tecnologia nos processos de produção.

A agricultura tradicional sofreu uma série de transformações ao longo dos últimos anos e passou a contar com uma maior utilização e integração de tecnologias, como o IoT. A essa era tecnológica da agricultura dá-se o nome de *smart farming*, *precision farming* ou *smart agriculture*. Devido a suas características, a IoT impactou grandemente a agricultura e assumiu um papel fundamental neste processo de inovação, e contribuiu para solucionar alguns dos problemas encontrados na agricultura tradicional.

Ao usar dispositivos inteligentes, é possível automatizar vários processos do ciclo de produção na agricultura. Estes dispositivos podem ser utilizados para monitorizar e controlar diferentes parâmetros do ambiente onde se encontram fisicamente localizados. Os dados coletados são então armazenados ou processados pelos próprios dispositivos ou serviços na nuvem. A partir deste ponto, tem-se acesso a dados de diferentes atividades e a possibilidade de se tomar decisões mais assertivas e em tempo real.

Graças à utilização de novas tecnologias, são muitos os benefícios trazidos pelas *smart faming*, dentre os quais destacam-se: aprimoramento da monitorização e controlo dos processos relacionados à produção, aumento da produtividade para a mesma área de produção, melhoramento na gestão do tempo da produção, possibilidade de resposta mais rápida e eficiente para os casos de risco ou calamidade, redução das prejuízos e custos, menor gasto com mão de obra humana e maiores lucros.

A utilização da IoT contribuiu também para a utilização de outras tecnologias, que facilitam a visualização e compreensão dos dados coletados pelos sensores. Como exemplo destas outras tecnologias, podem ser citadas inteligência artificial e suas subcategorias, *data analytics*, *Big Data*, entre outros. Inclusive, graças ao conceito de *edge computing*, os próprios sensores podem identificar e analisar padrões nos dados e tomar decisão de forma autônoma, sem depender de serviços localizados na nuvem. Porém, na maior parte das vezes, os dados são armazenados na *cloud*, e sistemas de *data analytics* são utilizados para possibilitar uma melhor compreensão destes dados, o que possibilita que os utilizadores possam gerir os ativos de forma mais adequada e organizada.

Dentre as possíveis aplicações de dispositivos IoT nas *smart farmings*, podem ser citadas:

- controlo sobre a saúde e níveis nutricionais e alimentares dos animais;
- rastreamento do deslocamento dos animais;
- monitorização de velocidade do vento, previsão meteorológica, incidência de luz solar, umidade, temperatura, concentração de CO2 do ambiente;
- controlo e monitorização de salinidade, pH, nutrientes e umidade do solo;
- rastreamento do crescimento das mudas e sementes;
- monitorização e controlo do microclima controlado (temperatura, luzes, nível de CO2, irrigação, umidade, saúde do plantio) de espaços de cultivos internos, como *greenhouses* e hidropônicos [1];
- aplicação automatizada de fertilizantes e pesticidas;
- Controlo e monitorização de aplicação de fertilizantes;
- monitorização e controlo de irrigação;
- controlo e monitorização das instalações de acondicionamento dos produtos;
- Serviços de localização geográfica como GPS ou rastreamento por satélite;
- Veículos agrícolas autônomos equipados com sensores inteligentes e precisos, GPS e machine learning para navegação autônoma;
- monitorização e controlo sobre gases nocivos, ventilação e qualidade do ar em ambientes fechados de concentração de animais;
- Monitorização dos veículos de logística e da cadeia de distribuição da produção para comerciantes e cliente final;
- veículos inteligentes e autônomos;
- imagens de satélites;
- *Drones* (terrestres e aéreos).

Dentre os exemplos citados acima, imagens de satélites, que são utilizados para monitorizar áreas de grandes proporções, *drones*, que possibilitam a captura e coleta de imagens aéreas e terrestre durante a atividade, e veículos autónomos, que desempenham de forma automatizada atividades que antes eram exercidas por trabalhadores, vêm ganhando destaque. Vale ressaltar que, além de desempenhar atividades de forma autônoma, os agentes autónomos podem ser também integrados com inteligência artificial, o que permite que eles desempenhem ações sem intervenção humana, como reconhecimento de doenças ou infestações de pragas (fungos, insetos,

1 Cultivo hidropônico: são vegetais que usam nutrientes ao invés de solo como meio de plantio.

larvas e outros contaminantes microbiológicos) e aplicação de pesticidas de forma automatizada.

Existe uma variedade de tecnologias IoT que podem ser utilizadas para a comunicação e a troca de mensagens dos dispositivos e sistemas da rede na *smart agriculture*. O que deve ser considerado é o ambiente no qual os dispositivos serão estabelecidos e a área de atuação destes dispositivos. Como exemplo, para áreas de menor dimensão, podem ser utilizadas tecnologias cabeadas, tecnologias que exijam maior poder computacional e energético ou tecnologias que possuam menor distância de atuação.

Para casos de grandes plantações ou fazenda de animais, o que implica em grandes áreas e grande distância entre os componentes da rede, o requisito passa a ser a distância e a alimentação solar ou baterias. Isto implica na necessidade de tecnologias que exijam menor poder computacional e tecnologias que consigam atuar em grandes distâncias.

2.1.8 *Smart Cities*

As cidades possuem numerosos requisitos e uma grande variedade de serviços. Algumas cidades do mundo passaram a gerenciar os seus recursos através da implantação de diferentes tecnologias e, a partir dos dados coletados, poder prestar serviços melhores e até mais especializados para a sociedade. O conceito de *smart city* ou cidade inteligente se desenvolveu a partir da implementação de tecnologias avançadas que facilitam a interconectividade e interoperabilidade entre diferentes dispositivos inteligentes e serviços. Tais cidades utilizam tecnologias como Internet das Coisas (IoT), *cloud computing*, Big Data, inteligência artificial, entre outros.

Estas tecnologias tentam solucionar alguns dos problemas urbanos das cidades e impulsionar a inovação e a transformação em todos os níveis da sociedade. Como exemplo, a tecnologia contribui para a melhora da capacidade de desenvolvimento económico e sustentável, para a qualidade da infraestrutura, da segurança pública e dos serviços públicos e para propiciar uma melhor qualidade de vida para as pessoas. Elas contribuem também para o gerenciamento e monitorização inteligente e em tempo real das operações da cidade, o que reduz o tempo necessário para a execução de ações mais eficientes e assertivas.

As pessoas, através de *wearables* ou dispositivos móveis, contribuem também para a geração de dados sobre atividades do cotidiano e suas localizações, cujos dados poderão ser partilhados e utilizados por sistemas das *smart cities*.

Como requisito básico para o bom funcionamento dos serviços e processos nas *smart cities*, podem ser citados as necessidades da qualidade, da baixa latência e da largura de banda dos meios de comunicação. Além disso, devido a quantidade de dados gerados pelos sensores, é necessário que sejam utilizados dispositivos com maior capacidade computacional. Tal necessidade é consequência de realizar a análise destes dados no menor tempo e possibilitar a melhoria nos serviços, por exemplo.

Nas *smart cities*, as principais áreas de aplicação incluem:

- eficiência energética e condição estrutural de casas, edifícios e outras estruturas;
- gestão inteligente do abastecimento de água, de energia elétrica e de gás natural;
- gestão inteligente da coleta de lixo e outros resíduos;
- transporte público;
- monitorização de casos de desastres naturais: abalos sísmicos, tsunamis, inundações, incêndios, tempestades, etc.;
- acionamento de alarmes em casos de risco de desastres naturais ou ação militar inimiga;
- gestão da iluminação da via pública;
- segurança pública: sensores acionados quando ocorre disparo de armas de fogo, vídeo-monitorização e reconhecimento facial;
- tráfego de veículos: semáforos inteligentes, ruas e estradas com meios de comunicação inteligente em caso de eventos inesperados, acidentes, engarrafamentos ou obras;

2.2 INTELIGENCIA ARTIFICIAL

Com o aumento do número dos casos de ataques cibernéticos, cujas técnicas são cada vez mais sofisticadas e mais difíceis de serem detetadas, torna-se cada vez mais necessário a utilização de controlos de segurança com capacidades de deteção mais aprimoradas, que viabilizem a segurança da infraestrutura e da rede de comunicação.

A partir dessa necessidade, estão a ser desenvolvidas algumas alternativas que contribuam para a melhora da segurança de ambientes e sistemas, tais como o *blockchain* e algumas das técnicas da inteligência artificial.

Inteligência artificial pode ser definida como a capacidade de uma máquina aprender e realizar, de forma automatizada, tarefas e comportamentos humanos a partir da realização de treinamentos especializados. O treinamento consiste na realização de contínuos processos e cálculos com base num conjunto de dados. Tal conjunto de dados é composto por registos que representam o domínio de interesse e possibilitam que a máquina consiga aprender a identificar padrões nestes dados. O treinamento é realizado repetidas vezes e, a cada ronda, é realizado o ajuste de alguns parâmetros utilizados para o cálculo dos dados, de forma a aumentar a precisão do modelo e encontrar mais facilmente o melhores resultados.

A partir do treinamento, é então estabelecido o modelo que será responsável por identificar os padrões nos dados de validação. A partir disso, ela poderá, com base nos padrões identificados no histórico de treinamentos anteriores, identificar e classificar também padrões em novos conjuntos de dados, assim como contribuir para previsões futuras.

A Inteligência Artificial é utilizada em diversas áreas, sendo abordada de forma abrangente pela comunidade académica e científica. Dentro da cibersegurança, ela tem apresentado considerável sucesso em alguns tipos de aplicações, sendo uma destas a deteção de anomalias e ataques em redes e infraestruturas de comunicação, inclusive em ambientes IoT.

No que se refere a deteção de anomalias, as técnicas de maior destaque são *machine learning* (ML) e *deep learning* (DL), que são subconjuntos da inteligência artificial (IA). Conforme identificado na literatura, estas técnicas têm sido bastante utilizadas para detetar alguns tipos de ataques, como DoS/DDoS, *spam*, *phishing*, *malwares*, *Zero-day attacks* e detecção de APT (*Advanced Persistent Threat*). Algumas outras literaturas apresentam a utilização destas técnicas também em ambientes IoT.

A *machine learning*, que significa aprendizado de máquina em português, tem como objetivo fazer com que as máquinas aprendam a tomar decisões de forma análoga aos seres humanos. O aprendizado é construído a partir do treinamento com um conjunto de dados e, então, o modelo será capaz de identificar os padrões existentes nestes dados e em conjuntos de dados futuros. Este aprendizado é estabelecido de forma quase autónoma, mas ainda necessita de supervisão e controlo humano em algumas operações e ajustes de parâmetros. Existem seis diferentes métodos de aprendizado de máquina, em que cada um apresenta uma característica e aplicação

específica. Devido a isso, cada um destes modelos requer padrões específicos de dados para apresentarem um resultado eficiente.

A *deep learning* é uma subcategoria da ML que realiza o reconhecimento de padrões com base no comportamento e estrutura do cérebro humano, cujo funcionamento tem como base os neurônios que são representados pelas redes neuronais. Estas redes neuronais são compostas por múltiplas camadas ocultas (*hidden layers*) e cada camada possui um número específico de neurônios. Com base nos coeficientes e hiperparâmetros, os valores calculados em cada neurônio são repassados para os neurônios da camada posterior e, então, combinados em cada neurônio desta camada (cada camada usa a saída da camada anterior como dados de entrada). A cada ronda, coeficientes e hiperparâmetros são ponderados para se chegar a melhores resultados. No campo da deteção de ameaças e reconhecimento de padrões desconhecidos, a técnica de DL apresenta um melhor desempenho em relação à ML.

Os modelos podem ser classificados de acordo com sua técnica de aprendizado em modelos baseados em *supervised learning*, *unsupervised learning*, *semi-supervised learning* e *reinforcement learning* (Verbraeken et al., 2020). Estas classificações são apresentadas abaixo:

- *Supervised learning*: a aprendizagem supervisionada utiliza dados de treinamento rotulados e classifica os dados com base em classes conhecidas ou conjunto predefinido de atributos-alvo (*target values*);
- *unsupervised learning*: a aprendizagem não supervisionada utiliza dados não rotulados como entrada do conjunto de treinamento, o que significa que as variáveis de entrada são fornecidas sem variáveis de saída correspondentes. Eles agrupam os dados em categorias com base na semelhança entre características destes dados; identifica primeiramente as classes dos dados a partir do padrão identificado e então os categoriza. Esta categoria de algoritmos não necessita de grande intervenção humana;
- *semi-supervised learning*: a aprendizagem semi-supervisionada é o método que combina as aprendizagens supervisionada e não supervisionada, que reduz os esforços de rótulo ou usa os dados rotulados parciais, proporcionando alta precisão dos algoritmos em dados grandes;
- *reinforcement learning*: O aprendizado desta técnica baseia-se na abordagem de tentativa e erro para identificar padrões nos dados fornecidos. Neste modelo a classificação é baseada em recompensas para classificações corretas ou para as classificações erradas de acordo com o valor apresentado pela predição.

Na lista abaixo, são apresentados alguns dos algoritmos mais comumente utilizados na literatura, assim como a sua descrição (Al-Garadi et al., 2020; Verbraeken et al., 2020).

- *Decision Tree*: o algoritmo cria um modelo baseado em árvore. Os nós da árvore aplicam a condição sobre os atributos para decidir qual o caminho será seguido de forma a separar os valores em classes e as folhas são o agrupamento dos registos de acordo com as respetivas classes;
- *Random Forest*: usa várias árvores de decisão e calcula a média da previsão feita pelas árvores individuais para definir a precisão geral do modelo;
- *Linear Regression*: é um modelo que utiliza uma função linear para determinar uma linha, reta de regressão, para representar a relação entre as variáveis;
- *Logistic Regression*: é um modelo muito parecido com a regressão linear, mas usado para problemas de classificação. Este modelo estima, através de uma função sigmoide, a probabilidade de um registo pertencer a uma classe, usando o atributo-alvo para isso;
- *Support Vector Machine* (SVM): utiliza uma reta, que também pode ser chamado hiperplano de separação, para separar os dados em diferentes classes. O algoritmo pode gerar mais um hiperplano de separação, mas leva em consideração a que melhor separa as classes. A reta criada tenta maximizar a distância entre um ponto de uma classe que está mais próximo a um ponto de uma outra classe;
- *K-nearest neighbors*: é um algoritmo que que realiza a classificação do dados com base na distância de um valor em relação aos K vizinhos mais próximos. Conforme a maior parte dos vizinhos pertencerem a uma classe especifica, o elemento em questão será também classificado com a classe equivalente a maioria destes vizinhos;
- *K-means*: o algoritmo é baseado no conceito de *clustering*, que é uma técnica utilizada para agrupar os dados em *clusters* que contenham características semelhantes;

Ainda que tenham apresentado boa eficiência no aprendizado a partir dos dados, os algoritmos de ML são diretamente dependentes da qualidade dos dados utilizados para o treinamento do modelo. Sendo assim, os dados de treinamento fornecidos devem ser selecionados com cuidado e estarem de acordo com o domínio de interesse.

Os modelos de aprendizagem dependem de grandes volumes de dados e de um *dataset* de qualidade para serem capazes de cobrir a maior parte dos casos do domínio

em questão e apresentar uma alta taxa de eficiência. Porém, deve-se tomar cuidado com o excesso de dados para treinamento, evitando, desta forma, o *overfitting*. O *overfitting* ocorre quando um modelo possui mais parâmetros ou os conjunto de dados possui um volume maior que o necessário, o que impactará em uma complexidade de análise maior e o modelo ficará viciado nos padrões encontrados. Este comportamento pode gerar um falso-positivo de que o modelo é eficiente sem o realmente ser. Neste sentido, existem algumas técnicas na literatura que são utilizadas para evitar o *overfitting* dos métodos de aprendizagem e construção do conjunto de dados.

Outro ponto de destaque está relacionado com os hiperparâmetros, que influenciam grandemente os resultados e desempenho do modelo de aprendizagem. O ajuste de hiperparâmetros, que pode ser realizado de forma manual ou automatizada, normalmente envolve treinar um modelo com diferentes configurações e conjuntos de dados e identificar os melhores resultados. Os temas otimização e configuração de hiperparâmetros também são muito abordados na literatura, pois procura encontrar formas mais eficientes para reduzir o tempo e custo computacional de identificação dos parâmetros que contribuam para a criação de modelos com melhores desempenhos. Neste sentido, se os resultados não forem bons o suficiente, novas etapas de treinamento são executadas com o ajuste dos hiperparâmetros. Atualmente, existem também algumas técnicas que exploram vulnerabilidades no algoritmo de treinamento (exemplo o *adversary machine learning*) e no processo de treinamento (*poisoning data attack*).

2.3 INTRUSION DETECTION SYSTEM

Os ativos tecnológicos, tal como a informação, estão frequentemente exposta a ameaças. Uma falha de segurança pode levar à perda ou ao roubo de informações sensíveis e impactar negativamente a imagem da entidade ou da pessoa. Portanto, garantir a proteção de tecnologias e ativos de informação deve ser sempre considerado como prioridade. Para este fim, existe um grande conjunto de controlos e ferramentas de segurança, com diversas características e funcionalidades, que são utilizadas para prevenir o comprometimento de qualquer um dos princípios de segurança da informação, como proteger dados confidenciais de acessos não autorizados e *data breaches*, mitigar o impacto causado por ataques cibernéticos ou restabelecer algum serviço em caso indisponibilidade deste e garantir a continuidade do negócio.

Como visto anteriormente, os componentes dos ambientes IoT são altamente interconectados e, graças as suas características já destacadas, são privados de aplicação de alguns mecanismos convencionais de segurança e passam a poder contar com alguma vulnerabilidade, que pode ser exploradas por pessoas mal intencionadas. Desta forma, deve haver uma monitorização adequada para impedir que ataques tenham êxito e, caso algum esteja a ocorrer, este seja brevemente detetado e impelido. Portanto, um sistema de deteção tem fundamental importância para a segurança da rede, inclusive no que está relacionado às redes de comunicação de ambientes IoT.

Um *Intrusion Detection System* (IDS) é um sistema ou *hardware* cuja finalidade é detetar atividades anômalas ou maliciosas no tráfego de rede e, assim, contribuir para a manutenção da segurança do ambiente. Desta forma, ele tem como responsabilidade identificar ataques, intrusões, atividades suspeitas e outras anomalias na rede em diferentes camadas da arquitetura IoT. Estes sistemas, ou equipamentos, analisam o tráfego de rede em busca de alguma anomalia ou comportamento diferente ao normal e disparam alertas para que sistemas especializados de defesa sejam realizem alguma ação de controlo ou combate ao fato ocorrido.

Os sistemas IDS podem ser classificados de acordo com o seu posicionamento na infraestrutura e de acordo com a forma como ele opera na deteção das ameaças. De acordo com a sua localização na infraestrutura, os sistemas IDS são classificados como HIDS (*Host based Intrusion Detection System*) e NIDS (*Network based Intrusion Detection System*).

Já em relação com o método utilizado para a deteção dos ataques e anomalias, os IDS são classificados, prioritariamente, em sistemas baseados em assinatura (SIDS) e sistemas baseados em anomalias ou comportamentos (AIDS). Os AIDS são frequentemente abordados em pesquisas da comunidade científica que incluem técnicas de ML/DL para aprimorar ainda mais a capacidade de deteção do sistema e, assim, diminuir a necessidade de intervenção humana neste sentido, pois os sistemas aprendem a partir das características dos dados fornecidos. Estas duas abordagens são apresentadas a seguir:

- **baseados em assinatura:** utiliza a técnica de correspondência de padrão que relaciona as assinaturas de ataques localizadas de uma base de dados que contém a assinatura de ataques conhecidos para analisar o padrão do ataque. Usam a técnica de correspondência de padrões para identificar ataques com assinaturas correspondentes aos da base de dados. Como vantagem, são muito eficazes na deteção de ataques conhecidos e produz um número

baixo de falso-positivo se as descrições do ataque e as regras de correção forem precisas. Em contrapartida, apenas os ataques conhecidos podem ser detetados, portanto, ataques cujas assinaturas não estejam na base de dados dificilmente serão detetados. Desta forma, *zero-day attacks* ou ataques com ligeiras alterações em seu comportamento podem não ser detetados até que a assinatura correspondente seja adicionada à base de dados;

- **baseados em comportamento:** também chamado de sistema baseado em anomalia, utilizam métodos heurísticos de deteção e tem como base o fato de que o comportamento das anomalias é diferente do comportamento normal da rede. Inicialmente, é definido um modelo base que sirva como representação do padrão legítimo e normal da rede. A partir disso, o sistema passa a monitorizar o tráfego em busca de detetar algum desvio no comportamento padrão da rede, o que pode indicar um ataque. Como vantagem, podem detetar ataques desconhecidos, mas podem apresentar altos índices de falsos-positivos ao interpretar um desvio normal como anomalia ou ataque;

De acordo com algumas literaturas, os IDS ainda podem assumir uma terceira e uma quarta classificação. Ele pode ser classificado como IDS híbrido quando utilizar das duas abordagens anteriormente apresentadas e realizar a deteção simultaneamente com base na assinatura e no comportamento. A quarta classificação está relacionada com IDS colaborativo (CIDS), que corresponde aos sistemas baseados no partilhamento de informações através de bases de dados coletivas e que é realizada por múltiplos colaboradores.

2.4 DATASETS

Um conjunto de dados, também chamado de *dataset*, é uma coleção de dados que podem ser utilizados por uma máquina ou humano para fins analíticos e de previsão. Os *datasets* têm um papel importante no treinamento dos algoritmos de deteção baseados em algumas das técnicas aprendizagem da IA, pois contribuem para que estes encontrem padrões no conjunto de dados e possam fazer o mesmo na detecção em tempo real. Como exemplo, uma máquina poderá utilizar algum algoritmo de aprendizagem para realizar o desenvolvimento de um modelo de previsão e, assim, encontrar padrões em novos conjuntos de dados.

A construção do *dataset* de treinamento poderá consumir considerável parte do tempo dependido na construção do modelo, porém, a qualidade do conjunto de dados utilizado para treinamento influenciará o sucesso deste modelo. Portanto,

é essencial compreender o domínio para assim definir o conjunto de dados a ser coletado e o modelo adequado a ser utilizado no treinamento. O *dataset* inicial será composto por dados em estado bruto e poderá conter dados em diferentes formatos. Como exemplo, o conjunto de dados pode conter registos que representem o histórico de um ou mais tipos de ataques, juntamente a dados referentes ao tráfego de rede normal.

Posteriormente, para alguns algoritmos de aprendizagem, é necessário que haja uma preparação dos dados brutos para que estes sejam utilizáveis pelo algoritmo de treinamento. Estes dados deverão ser pré-processados e rotulados antes de serem executados no treinamento do modelo, de forma melhorar uma maior qualidade dos dados do *dataset* de treinamento e garantir um bom desempenho do modelo.

O conjunto de dados deve atender a padrões de qualidade e balanceamento dos dados e deverá conter um volume suficiente para o sucesso do treinamento e consequente modelo a ser desenvolvido. Um *dataset* de maior qualidade possibilitará que o algoritmo de aprendizagem possa realizar a identificação e a análise mais eficiente das tendências e padrões ocultos no conjunto de dados apresentado. Com base nisso, dois pontos são de crucial importância para a qualidade do *dataset* e poderão contribuir diretamente para o sucesso no treinamento dos algoritmos de aprendizagem e para a eficiência do modelo criado.

O primeiro ponto tem relação com os dados, que devem estar distribuídos de forma equilibrada e uniforme, pois, caso contrário, poderá haver uma interferência nos resultados devido a tendências existentes nos dados. O segundo ponto está relacionado com o volume de dados que compõem o *dataset*. É importante destacar que é importante que este volume seja suficientemente grande para conter a maior parte dos casos possíveis do domínio em questão. Desta forma, o modelo terá uma probabilidade (capacidade) maior de identificar, no melhor dos casos, o padrão referente a totalidade dos casos esperados, inclusive os mais raros.

Como ponto de cuidado, deve-se evitar ao máximo que o volume de dados seja demasiado grande, pois isto pode causar o *overtraining* ou *overfitting* do modelo. Tal característica não trará benefícios para treinamento, mas tornará a análise dos dados mais complexa e apenas tornará este processo mais lento.

O conjunto de dados é normalmente dividido em dois subconjuntos, a saber, subconjunto de treinamento e subconjunto de teste. Estes dois subconjuntos de dados são apresentados a seguir:

- **subconjunto de treinamento:** é utilizado para construir o modelo preditivo. É através deste *dataset* que o modelo irá ser treinado e irá aprender para

identificar os padrões existentes no dados fornecidos. Em relação ao volume de dados, é o maior dos dois subconjuntos;

- **subconjunto de teste:** é utilizado para verificar a taxa de eficiência e a qualidade da predição do modelo. Ele é utilizado ao final da fase de treinamento do modelo, em que se verifica o desempenho do modelo treinado com os dados de treino, utilizando-o para fazer previsões com base nos dados de teste e comparando os resultados previstos com os esperados.

Existem alguns *datasets* que são frequentemente utilizados em pesquisas académicas e indústria para avaliação e treinamento de alguns modelos de aprendizagem. Em sua maioria, estes *datasets* são públicos e podem ser utilizados livremente pelos interessados. A lista abaixo apresenta alguns dos *dataset* encontrados e uma breve descrição de cada um.

- Bot-IoT (Koroniotis et al., 2019);
- CICIDS2017 (Sharafaldin et al., 2018);
- Edge-IIoTset (Ferrag, Friha et al., 2022);
- IOT23 dataset (Parmisano et al., 2020);
- MQTT-IoT-IDS2020 (Hindy et al., 2021);
- MQTTset (Vaccari et al., 2020);
- N-BaIoT Dataset (Meidan et al., 2018);
- NSL-KDD (Tavallaee et al., 2009);
- TON-IoT (Moustafa et al., 2020).

Como foi destacado em algumas literaturas, os pontos negativos de destaque dos *datasets* existentes são:

- qualidade dos dados;
- não especifidade para ambientes IoT;
- reduzido número de dispositivos;
- reduzida variedade de protocolos;
- reduzida variedade de ataques abordados.

Devido a isso, faz-se necessário o desenvolvimento de um *dataset* que contemple uma variedade de ataques e protocolos e que consiga representar um ambiente IoT

real. Desta forma, será possível a utilização de algoritmos de *machine learning* para a deteção dos ataques presentes nestes *datasets*.

2.5 TRABALHOS RELACIONADOS

Xin et al. (2018) e Dargan et al. (2019) apresentam as principais características, aplicações e diferenças entre as técnicas de ML e DL. Técnicas deste tipo buscam contribuir com a deteção de ataques ou anomalias, como é o caso das vulnerabilidade *zero-day*, a partir da identificação de comportamentos anômalos no tráfego através de técnicas de aprendizagem (Tahsien et al., 2020).

Técnicas de ML têm sido adotadas em alguns trabalhos académicos nos últimos anos. Ela surge como uma solução promissora para a segurança dos dispositivos IoT, apresentando uma abordagem diferente na identificação e defesa de ataques se comparada a outros métodos tradicionais (Hassija et al., 2019). Ela propicia a inclusão de fatores de inteligência para a defesa e a segurança do sistema ou infraestrutura, assim como na melhoria da capacidade na identificação de ameaças e ataques (Dwibedi et al., 2020). Ibitoye et al. (2019) e Roopak et al. (2019) apresentam que as técnicas de DL, que é uma subcategoria da de ML, têm apresentado um bom desempenho na deteção de ataques e tráfego anômalo em dispositivos IoT. Como característica, estes dois tipos de técnica requerem grandes quantidades de dados para ter um bom desempenho na identificação de ataques, o que implica em tempos maiores de treinamento (Roopak et al., 2019).

Um ponto de destaque é que a eficiência da capacidade de detecção está diretamente relacionada com o conjunto de dados (*dataset*) utilizado para o treinamento aplicado aos métodos de identificação (Buczak e Guven, 2015). Por isso, a composição do *dataset* é fundamental para treinar um modelo de detecção, assim como para avaliar a eficiência da capacidade de deteção deste modelo (Al-Hadhrami e Hussain, 2020). Porém, Al-Hadhrami e Hussain (2020) apresentam ainda que muitos dos *datasets* existentes são desatualizados ou são compostos por dados que podem não representar um cenário do mundo real, além de não representarem uma diversidade muito grande de ataques. Booij et al. (2021) afirma que muitos dos *datasets* existentes não possuem muita heterogeneidade nos dados que o compõem e poucos são designados para deteção de anomalias em redes IoT. Ele afirma ainda que a heterogeneidade dos dados do *dataset* podem ter um grande impacto no desempenho da técnica de deteção.

Mohanta et al. (2020) identificaram alguns problemas de segurança existentes em infraestruturas IoT. Ainda, identificaram algumas das tecnologias já adotadas na tentativa de resolver algum dos problemas identificados, a exemplo da técnicas de inteligência artificial e *blockchain*. Várias ameaças à segurança e problemas relacionadas às diferentes camadas de um sistema IoT foram apresentadas em (Hassija et al., 2019). Ainda, foram apresentadas algumas possíveis soluções para estes problemas, incluindo *blockchain*, *fog* e *edge computing* e *machine learning*.

Xiao et al. (2018) investigou modelos de deteção de ataques em sistemas IoT, cujos modelos eram baseados em técnicas de ML. Ainda, apresentou alguns controlos de proteção promissores que poderiam ser adoptados em dispositivos IoT. Tahsien et al. (2020) realizou uma revisão da literatura sobre segurança de IoT baseada em ML. A revisão aborda temas a arquitetura de sistemas IoT, os diferentes tipos de ataques e vetores de ataques, vários modelos e soluções de segurança baseadas em ML. Buczak e Guven (2015) realizou um revisão da literatura referente à utilização de métodos de identificação baseados em ML e DL.

Alguns outros trabalhos adotaram técnicas baseadas em *deep learning*. Ibitoye et al. (2019) desenvolveu dois métodos baseados em técnicas de DL para a deteção de intrusão no contexto de IoT. Para o treinamento destes métodos, foi utilizado o *dataset* BoT-IoT (Koroniotis et al., 2019). Roopak et al. (2019) propõe alguns modelos de DL, apresentando também os desafios relacionados com a implementação deste tipo de técnica nos sistemas IoT. Os modelos propostos foram avaliados usando o *dataset* CICIDS2017 (*IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB* 2021) para a deteção de ataques DDoS e estes foram também comparados com alguns modelos de ML. Foi realizado um estudo comparativo de sete métodos, baseados em DL, aplicados à detecção de intrusão em (Ferrag, Maglaras et al., 2020). Esses métodos foram comparados a partir de dois *datasets* (CSE-CICIDS2018 (*IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB* 2021) e Bot-IoT), sendo considerados na comparação três indicadores de desempenho: taxa de falso-positivos, precisão e taxa de detecção.

Dwibedi et al. (2020) apresenta as características de um conjunto de *datasets* e explora mais detalhadamente três destes *datasets*, fazendo uma comparação das características destes. Compara também o desempenho de modelos de ML com cada um dos três *datasets* utilizados. Koroniotis et al. (2019) desenvolveu um novo *dataset* chamado Bot-IoT. Esta *dataset* é composto por tráfego de rede real relacionado à sistemas IoT e outros dispositivos de rede, juntamente com o tráfego referente a ataques comumente realizados por *botnets*. Além disso, o *dataset* desenvolvido e

alguns outros *datasets* foram avaliados a partir de alguns métodos de aprendizagem baseados em ML e DL.

Foram identificados alguns *datasets* utilizados para o treinamento dos sistemas de deteção de ataques. Estes *datasets* são compostos por dados que representam tráfego, real ou simulado, de dispositivos IoT e de ataques realizados ao ambiente. Alguns destes *datasets* são compostos com tráfego gerado por múltiplos dispositivos IoT, com diferentes protocolos de comunicação. Ring et al. (2019) fez uma descrição e análise de 34 *datasets* existentes e identifica 15 propriedades diferentes, agrupadas em 5 categorias, para avaliar a adequação de conjuntos de dados individuais para cenários de avaliação específicos. Os *datasets* mais utilizados na literatura revisada foram BoT-IoT (Dwibedi et al., 2020; Ferrag, Maglaras et al., 2020; Ibitoye et al., 2019; Koroniotis et al., 2019), CSE-CICIDS2018 (Dwibedi et al., 2020; Ferrag, Maglaras et al., 2020) e UNSW-NB15 (Dwibedi et al., 2020).

De forma a emular ou simular tanto os dispositivos IoT quanto o tráfego de rede destes dispositivos, é possível utilizar algumas das ferramentas apresentadas nos trabalhos pesquisados. Al-Hadhrami e Hussain (2020) utilizou a ferramenta Cooja para emular os dispositivos IoT na realização dos testes. Esta ferramenta realizou o monitoramento e a recolha dos dados e os encaminhou a unidade de agregação de dados. Hassan et al. (2019) adoptou a ferramenta ns-2 para simular os dispositivos. Outras ferramentas utilizadas para este fim são ns-3 (da mesma fabricante da ns-2), Opnet, Omnet++, Qualnet and NetSim. A ferramenta Ixia foi utilizada para gerar o tráfego de rede no dataset UNSW-NB15 e a Ostinato para gerar o tráfego do dataset Bot-IoT (Dwibedi et al., 2020). Esta ferramenta foi também utilizada para geração do tráfego massivo em (Koroniotis et al., 2019). Outra ferramenta utilizada para geração do tráfego é a Node-red (Koroniotis et al., 2019).

Outras ferramentas podem ser utilizadas para a realização da captura do tráfego de rede gerado pelas ferramentas anteriormente mencionadas ou por dispositivos reais. O tráfego de rede do *dataset* CSE-CIC-IDS2018 foi capturado através da ferramenta CICFlowMeter-V3 (Dwibedi et al., 2020). A ferramenta Sensniff, que pode ser equipada com a biblioteca Libpcap, é outra ferramenta utilizada para monitorizar a rede em sistemas IoT (Al-Hadhrami e Hussain, 2020). A ferramenta tshark tool foi utilizada em Koroniotis et al. (2019) para coletar os pacotes reais e de ataques, que também utilizou a ferramenta Argus para a extração de recursos e análise forense dos dados.

Mohanta et al. (2020) identificaram alguns problemas de segurança existentes em infraestruturas IoT. Ainda, identificaram algumas das tecnologias já adotadas

na tentativa de resolver algum dos problemas identificados, a exemplo da técnicas de inteligência artificial e *blockchain*. Várias ameaças à segurança e problemas relacionadas às diferentes camadas de um sistema IoT foram apresentadas em (Hassija et al., 2019). Ainda, foram apresentadas algumas possíveis soluções para estes problemas, incluindo *blockchain*, *fog* e *edge computing* e *machine learning*.

O protocolo RPL, que é comumente utilizado em ambientes IoT, tem o seu funcionamento apresentado em (Simha et al., 2020), (Accettura et al., 2011) e (Lamaazi e Benamar, 2020). O funcionamento dos protocolos CoAP e MQTT são apresentados em (Arvind e Narayanan, 2019) e (Soni e Makwana, 2017) respetivamente.

3

SOLUÇÕES DE GERAÇÃO DE DADOS

Ferramentas são utilizadas desde o início da história humana como forma de facilitar a execução de tarefas e processos do cotidiano, com o objetivo de reduzir o tempo e o esforço necessário para completá-los. Na área da tecnologia, elas podem ser utilizadas para facilitar o desenvolvimento de pesquisas e produtos, já que podem simular o comportamento real, ou quase, de *hardwares*, sistemas e protocolos de comunicação. Desta forma, a complexidade e o investimento despendidos no projeto e no desenvolvimento de novas tecnologias podem ser reduzidos, pois permite uma avaliação antecipada destas tecnologias antes de elas serem lançadas.

Levando isso em consideração, foi necessário identificar as ferramentas que poderiam ser utilizadas para cumprir as etapas do atual trabalho, tal como gerar o tráfego de dados, capturar o tráfego e gerar o fluxo de dados. Neste capítulo serão apresentadas as ferramentas de geração ou simulação de tráfego de dados e as ferramentas de geração de fluxo de dados analisadas, assim como as suas características e, no final do capítulo, a seleção das ferramentas que serão utilizadas no atual trabalho.

3.1 SOLUÇÕES DE GERAÇÃO DE DADOS

Como visto anteriormente, os algoritmos de aprendizagem são dependentes de sucessivos treinamentos, cujo treinamento é realizado através de *datasets* específicos ao domínio em questão. A criação destes *datasets* pode ser realizada a partir de algumas ferramentas ou simuladores específicos. Elas tem o objetivo de emular os dispositivos, inclusive dispositivos IoT, e simular o tráfego de rede com base na utilização de alguns protocolos.

3.2 CARACTERÍSTICAS DE INTERESSE

É importante que estas ferramentas, mesmo que gerem dados fictícios, consigam representar o comportamento de dispositivos e comunicações encontrados em um

ambiente real nas simulações, assim como a capacidade de emular as características específicas de produtos de diferentes marcas. Além disso, busca-se por identificar nas ferramentas o suporte a diferentes tecnologias de IoT para ser possível criar uma maior heterogeneidade nos dados gerados. Por fim, busca-se pela possibilidade de incorporar novas funcionalidades a partir de implementação de código, sem que isso gera impacto substancial no tempo despendido nas simulações.

Com base nestes pontos iniciais, foi desenvolvida a lista a seguir, que contém algumas características identificada como importantes para a seleção da ferramenta que será utilizada neste trabalho:

- suportar diferentes arquiteturas e protocolos (heterogeneidade da infraestrutura IoT);
- simular o comportamento real de ambientes IoT;
- geração de fluxo de dados baseados em tecnologias IoT;
- facilidade de implementação de novas funcionalidades e características;
- realização de ataques de forma nativa;
- geração do fluxo de dados em menor tempo que o real definido como duração da simulação;
- exportação dos dados para formatos CSV ou PCAP.

Os pontos constantes na lista acima referida são considerados os mais relevantes para o atual trabalho e devem constar nas funcionalidades das ferramentas estudadas e serão considerados para a avaliação da ferramenta a ser escolhida para a realização das simulações e geração dos dados que farão parte do *dataset*.

Em relação as ferramentas, foram testadas as apresentadas na lista abaixo e estas serão mais detalhadamente descritas nas secções a seguir:

- Contiki OS;
- Network Simulator 3;
- IoT *dataset* Generator Framework;
- IoT-Flock;
- MQTT Generator;
- COAP Protocol Simulator;
- Scapy;
- Netsim.

3.3 CONTIKI/COOJA

Contiki OS[1] e Contiki-NG[2] são emuladores de dispositivos sem fio de baixa potência. A ferramenta está sob licença 3-Clause BSD License e foi desenvolvida em Java. A ferramenta é combinada com o simulador de sensores chamado Cooja, que é quem realiza a simulação composta pelos nós definidos na ferramenta Contiki.

3.3.1 *Características e Funcionalidades*

A ferramenta apresenta uma interface gráfica que facilita a configuração dos dispositivos a serem utilizados na simulação. Ela permite que seja implementado o *firmware* de alguns modelos de dispositivos IoT reais, o que possibilita a emulação do *hardware* destes dispositivos na simulação. Desta forma, o código compilado utilizado na simulação pode ser também utilizado em dispositivos reais. Além disso, podem ser incorporados à simulação um ou mais dispositivos reais, que são externos ao simulador.

Entre outros protocolos, o sistema Contiki apresenta suporte nativo para os protocolos CoAP e MQTT e possibilita ainda que novas tecnologias (protocolos, *firmwares* etc.) sejam adicionadas a partir de *scripts* Python e C. É de destacar que os *scripts* que implementam o comportamento de um serviço MQTT não está disponível na mais recente versão do sistema Contiki OS e foi necessário resgatá-lo em um *commit* anterior. Uma característica importante desta ferramenta é a de que ao final de cada simulação, são automaticamente gerados *logs* dos eventos e atividades da simulação executada, sem que seja necessário que o utilizador realize qualquer operação.

O tráfego de dados é gerado automaticamente na ferramenta Cooja e pode ser exportado no formato PCAP, desta forma, não necessitando de uma ferramenta externa para esse fim. Os dados gerados podem ser monitorizados em tempo de execução através da interface gráfica da ferramenta.

As ferramentas Contiki OS e Cooja foram utilizadas em muitas simulações que envolviam a realização de ataques no ambiente. Apesar deste ponto, os dispositivos que realizavam o papel de atacantes tiveram de ser implementadas pelos autores destes trabalhos. Isso decorre do fato de a ferramenta não implementar a característica de ataque na sua forma nativa e tal comportamento deverá ser implementado

1 https://github.com/contiki-os/contiki
2 https://github.com/contiki-ng/contiki-ng

através de *scripts* em uma das linguagens suportadas pela ferramenta, de acordo com a necessidade da simulação.

Como apresentada anteriormente, a ferramenta Cooja dispõe de inúmeras características importantes, motivo pelo qual ela fora utilizada em uma grande quantidade de trabalhos acadêmicos. Apesar disso, novas funcionalidades podem ser requeridas nas simulações e estas devem ser implementadas através de *scripts* de forma a adicionar tais funcionalidades.

3.3.2 *Avaliação das Características*

Apesar de não fornecer a funcionalidade de executar ataques de forma nativa, tal característica pode ser implementada pelo desenvolvedor através de *scripts*. Alguns trabalhos apontam a lentidão na realização das simulações como um ponto fraco desta ferramenta.

Em relação aos pontos positivos, destacam-se a grande quantidade de funcionalidades trazidas nativamente pela ferramenta e a confiabilidade estabelecida ao longo do tempo. Além disso, a ferramenta é baseada em interface gráfica e possibilita a configuração de alguns parâmetros durante a realização das simulações. Por fim, a ferramenta já possui a implementação dos três protocolos de interesse deste trabalho (protocolos RPL, CoAP e MQTT).

3.4 NETWORK SIMULATOR 3

Network Simulator 3 (ns-3)[3] é uma ferramenta de simulação de redes IoT com e sem-fio, sucessora da ferramenta ns-2[4].

3.4.1 *Características e Funcionalidades*

A ferramenta ns-3 não apresenta suporte nativo aos protocolos da camada de aplicação desejados, portanto, não há suporte nativo para a realização de simulações que relacionem os protocolos CoAP e MQTT, além de não apresentar suporte para o protocolo RPL. Através da ferramenta, é possível que sejam integrados dispositivos reais à rede simulada. Apesar de não haver suporte nativo a interface gráfica, pode

3 https://www.nsnam.org/
4 https://www.isi.edu/nsnam/ns/

ser adicionado o NetAnim[5] para visualização dos dados gerados anteriormente em uma simulação.

A simulação e respetivo fluxo de dados são inteiramente configurados a nível do código criado pelo utilizador. O tráfego de dados gerado durante a simulação pode ser armazenado como ficheiro pcap, o que facilita a sua análise através de ferramentas específicas.

Apesar de não disponibilizar ataques de forma padrão, existem alguns ataques já desenvolvidos e apresentados na literatura. Ainda, podem ser criados outros a partir do desenvolvimento das funcionalidades referentes ao ataque desejado.

A ferramenta possibilita a adição de novas funcionalidades e características, o que deverá ser feito a partir de *scripts* nas linguagens C++ e Python.

3.4.2 *Avaliação das Características*

Apesar de ser considerada como uma ótima ferramenta de simulação na literatura, a ferramenta está limitada a não suportar os protocolos desejados (RPL, COAP e MQTT). A ferramenta também não apresenta de forma nativa bibliotecas que representem ataques a redes IoT. Desta forma, ela não está apta para fazer parte deste trabalho sem a necessidade de implementação das funcionalidades desejada ou sem o intermédio de outra ferramenta que supra tais limitações.

3.5 IOT *DATASET* GENERATOR FRAMEWORK

IoT *Dataset* Generation Framework (IDGF)[6] é um gerador de tráfego MQTT para ambientes IoT desenvolvido em Java, .

3.5.1 *Características e Funcionalidades*

Esta ferramenta apresenta uma interface gráfica que possibilita a fácil criação de uma rede IoT e a adição de dispositivos a esta rede. A ferramenta já traz, de forma nativa, dois exemplos de redes IoT, cuja primeira representa uma *smart home* e

5 https://www.nsnam.org/wiki/NetAnim
6 https://git.tk.informatik.tu-darmstadt.de/SPIN/IoTDatasetGenerationFramework

a segunda refere-se a um ambiente de tratamento de água, os quais podem ser utilizados como exemplos para gerar o tráfego MQTT.

A ferramenta permite que o tráfego gerado na simulação seja exportado para um ficheiro de texto, definido na tela de configuração da simulação. É de destacar que não foi possível a captura das mensagens com ferramentas como como Wireshark[7] ou tcpdump[8]. Tal característica pode estar relacionado com o fato de as trocas de mensagens ocorrerem apenas na camada de aplicação, em que as mensagens são apresentadas no terminal de linha de comandos, não havendo participação de camadas inferiores.

A ferramenta já implementa os ataques DoS e DDoS e apresenta duas anomalias, em que a primeira representa um dispositivo defeituoso e não operacional e a segunda representa o comportamento de anomalia nos valores informados pelo dispositivo. Tal comportamento possibilita que sejam gerados dados que representem anomalias nos dados trafegados na rede, mas que não necessariamente estejam relacionados com algum tipo de ataque e que apenas representem algum problema físico ou de sistema.

Segundo os autores, é possível que sejam adicionados novos protocolos, dispositivos e ataques através da devida implementação destas funcionalidades. Um ponto de destaque é o de que não é gerado um tráfego de rede real, porém, é gerada uma representação de mensagens customizadas que representa uma comunicação entre os dispositivos da rede e que, posteriormente, será utilizada pelo algoritmo Python de análise para definir o índice de acerto na identificação dos ataques e fluxo real de dados.

3.5.2 *Avaliação das Características*

Os pontos positivos da ferramenta são fácil configuração da simulação, o que é realizado a partir da interface gráfica, e a existência de ataques na versão padrão. Em relação aos pontos negativos, está não apresenta suporte para o protocolo CoAP e não foi possível a captura da comunicação dos componentes da simulação.

7 https://www.wireshark.org/
8 https://www.tcpdump.org/

3.6 IOT-FLOCK

IoT-Flock[9] é uma ferramente de geração de tráfego de dados de ambientes IoT desenvolvida em C++.

3.6.1 *Características e Funcionalidades*

Ela utiliza de uma interface gráfica que possibilita a configuração de diversos parâmetros dos dispositivos a serem utilizados na simulação e estas configurações serão guardadas em um ficheiro XML. A ferramenta apresenta suporte para dois dos principais protocolos da camada de aplicação da IoT, que são os protocolos MQTT e COAP. A ferramenta permite a comunicação com dispositivos externos, a exemplo de executar ataques a algum *broker* MQTT público.

A execução da simulação se dá através da linha de comandos e com a utilização do ficheiro XML anteriormente citado. A ferramenta não realiza automaticamente a captura do fluxo de dados gerado, portanto, é necessário que esta operação seja realizada por uma ferramenta externa como o Wireshark.

A ferramenta já possibilita de forma nativa que sejam gerados dados referentes a alguns tipos de ataques de rede IoT, sendo possível determinar o número de mensagens por segundo, a mensagem a ser encaminhada, o tipo de ataque e os endereços e portos de origem e destino.

A adição de novas funcionalidades poderá ser feita a partir da adição de código na linguagem C++, mas haverá a necessidade de recompilação do novo código-fonte.

3.6.2 *Avaliação das Características*

A ferramenta apresenta suporte aos dois protocolos pretendidos, assim como possui interface gráfica para configuração da simulação e já disponibiliza ataques em sua versão padrão. Em relação aos pontos negativos, apenas é possível configurar os componentes da simulação e servidores com endereços IPv4. A adição de novas funcionalidades demonstrou-se por ser um pouco complexa, devido a código não estar desenvolvido de uma forma organizada.

9 https://github.com/ThingzDefense/IoT-Flock

3.7 MQTT GENERATOR

MQTT Generator[10] (MQTT Gen.) é uma implementação (*script*) da biblioteca Python chamada paho-mqtt[11].

3.7.1 *Características e Funcionalidades*

Ela permite a geração de dados MQTT provenientes da troca de mensagens de dispositivos e *broker*. O *broker* depende do pacote mosquitto[12] para funcionar e este deve ser instalado no sistema onde é executado o *script* da ferramenta MQTT Generator. A ferramenta funciona em linha de comando (não há interface gráfica) e é necessária a utilização de ferramentas de *sniffing* para a captura dos dados gerados por ela.

Os dispositivos e as demais configurações da infraestrutura, que é baseada no protocolo MQTT, são definidas em um único ficheiro JSON e estas configurações são então importadas, em tempo de execução, pelo *script* Python para que a simulação seja executada.

O *script* apenas tenta representar a comunicação entre dispositivos e *broker* MQTT. Não foram implementadas no *script* funcionalidades referentes a ataques e ela também não realiza a comunicação com dispositivos externos, portanto, tais características devem ser desenvolvidas e implementadas pelo utilizador.

Como se trata de um *script* e não de uma ferramenta em si, é possível que novas funcionalidades sejam facilmente adicionadas por meio de alteração do *script* Python, o que implica na necessidade de conhecimento em lógica de programação e conhecimento em Python.

3.7.2 *Avaliação das Características*

O ponto positivo da ferramenta é a facilidade com que ela pode ser configurada e executada, assim como a possibilidade de se criar novas funcionalidades. Porém, em quase nada ela consegue atender as necessidades do atual trabalho, tal como os protocolos suportados e ataques previamente existentes.

10 https://gist.github.com/marianoguerra/be216a581ef7bc23673f501fdea0e15a
11 https://www.eclipse.org/paho/
12 https://mosquitto.org/

3.8 COAP PROTOCOL SIMULATOR

Coap-protocol-simulator[13] (COAP PS) é uma ferramenta desenvolvida em Java que possibilita a simulação de pacotes CoAP.

3.8.1 *Características e Funcionalidades*

As mensagens podem ser salvas em ficheiro através de comandos específicos do próprio terminal de comandos. A ferramenta não realiza a leitura de dados diretamente de dispositivos externos.

A ferramenta realiza as operações através da leitura de valores presentes em ficheiros de extensão CSV. Um agente com a função de cliente irá realizar a leitura dos valores do CSV e irá encaminhá-los para o agente com função de servidor CoAP. Por sua vez, o agente servidor, ao receber uma mensagem do cliente, irá responder esta mensagem com valores lidos também de um ficheiro CSV.

Não há implementação de ataques, uma vez que a ferramenta apenas visa simular fluxo de dados CoAP. A ferramenta possibilita que novas funcionalidades sejam incorporadas à simulação, porém estas deverão ser implementadas a nível de código Java.

3.8.2 *Avaliação das Características*

O ponto positivo da ferramenta é a facilidade com que ela pode ser configurada e executada, assim como a possibilidade de se criar novas funcionalidades. Porém, em quase nada ela consegue atender as necessidades do atual trabalho, tal como os protocolos suportados e ataques previamente existentes.

3.9 SCAPY

Scapy[14] é um manipulador de pacotes de rede desenvolvido em Python que possibilita que o utilizador manipule um grande número de protocolos.

13 https://github.com/CiscoIOx/coap-protocol-simulator
14 https://scapy.net/

3.9.1 *Características e Funcionalidades*

Além de manipular pacotes de redes a partir de diversos protocolos, a ferramenta pode atuar com um *packet sniffer* e realizar a captura dos pacotes que estão a ser trafegados na rede. A ferramenta funciona como um *framework* com suporte a um considerável número de protocolos de rede, incluindo os protocolos MQTT e CoAP. A ferramenta não possui interface gráfica e a sua utilização é feita através de linha de comando.

O tráfego de redes precisa ser implementado a nível de *script* pelo utilizador, assim como o comportamento da simulação, a exemplo dos campos dos cabeçalhos e dados dos protocolos a serem utilizados. Requer conhecimento de Python e também da pilha de camadas e protocolos da arquiteturas de redes TCP/IP e OSI de forma a representar fidedignamente um ambiente real.

A ferramenta não apresenta de forma nativa a implementação de ataques, portanto, qualquer que seja o ataque pretendido, este deve ser integralmente desenvolvido pelo utilizador através de *script*. São impostas as mesmas complexidades apresentadas para o caso da geração dos dados.

É possível a implementação de novas funcionalidades por parte do utilizador através da implementação de *scripts* Python.

3.9.2 *Avaliação das Características*

Um ponto de destaque da ferramenta é a prévia implementação de um grande números de protocolos. Porém, a ferramenta apresenta um considerável número de pontos negativos, o que faz com que ela não atenda os requisitos desejados. Para criar uma simulação, seria necessário desenvolver todo o ambiente do zero(sensores, protocolos, lógica de comunicação entre dispositivos, ataques, etc.), o que torna inviável a utilização desta ferramenta. Além disso, não é apresentado suporte a interface gráfica.

3.10 NETSIM

NetSim[15] é uma ferramenta comercial, com suporte apenas para o sistema operativo Windows, que possibilita a simulação de sistemas IoT através de uma interface gráfica.

3.10.1 *Características e Funcionalidades*

A ambiente a ser simulado na ferramenta é totalmente configurável a partir da interface gráfica, sendo necessário apenas posicionar os componentes no ambiente, estabelecer as interligações entre os componentes, configurar as aplicações necessárias e executar a simulação. Ao término desta, será gerado um resumo da simulação executada, com gráficos, ficheiro PCAP e fluxo de dados no formato CSV. Através do módulo "*NetSim Network Emulator*", é possível adicionar ao ambiente simulado fluxos de dados provenientes de redes externas à ferramenta. Dentro os protocolos possíveis, a ferramenta apresenta suporte ao CoAP e RPL.

Os dados são gerados a partir da execução da simulação e de acordo com as aplicações e protocolos definidos na configuração do ambiente. O fluxo de dados de cada componente pode ser exportado para ficheiros PCAP.

A ferramenta não implementa, de forma padrão, a funcionalidade de ataques durante a execução da simulação. Tal funcionalidade deverá ser adicionada previamente pelos interessados a partir do desenvolvimento de código ou a partir do *download* e incorporação de alguns ataques previamente disponíveis no repositório Github[16] da empresa que mantém a ferramenta. Em relação a versão 13.1 da ferramenta, existem um conjunto considerável de ataques possíveis, dentre os quais, alguns são específicos para ambientes IoT.

A ferramenta apresenta a característica de permitir com que novas funcionalidade sejam adicionadas a partir do desenvolvimento e adição destas funcionalidades. Exemplos destas características é a opção de adicionar novos ataques através dos modelos que estão disponíveis no repositório do Github. Além disso, podem ser desenvolvidas novas funcionalidades de interesse do autor, bastando implementar o código e realizar a recompilação do código original com as alterações acrescidas.

15 https://www.tetcos.com/
16 https://github.com/NetSim-TETCOS?tab=repositories

3.10.2 *Avaliação das Características*

Dentre as características, os pontos de maior destaque são o resumo da simulação e a facilidade com a qual a simulação é configurada, onde tudo é feito via interface gráfica. O módulo "*NetSim Network Emulator*"possibilita a adição de fluxo de dados externos à ferramenta, como por exemplo, fluxo de dados de protocolos não suportados pela ferramenta. Os pontos negativos são a não implementação do protocolo MQTT e a necessidade de realizar a configuração de geração do ficheiro PCAP em cada um dos componentes do ambiente (não há forma de centralizar a captura de dados dos sensores sem fio). Esta última característica implica com que haja a necessidade de realizar a integração de todos os ficheiros pcap em um só, de forma a facilitar a gestão de todos os fluxos de dados em um só ficheiro.

3.11 RESUMO COMPARATIVO DAS FERRAMENTAS

Após a realização de testes em cada uma das ferramentas e do devido estabelecimento das características, foi possível resumir em uma tabela, os pontos identificados. A Tabela 1 apresenta uma comparação das características das ferramentas de geração de dados IoT.

3.12 DEFINIÇÃO DA FERRAMENTA

Após o levantamento das características das ferramentas apresentadas, faz-se necessário definir a ou as ferramentas que serão utilizadas no processo de simulação e geração do tráfego de dados. Nenhuma das ferramentas atende plenamente todas as

Tabela 1: Comparação das características das ferramentas de geração de dados IoT

Ferramenta	**Ling.**	**Protocolos**	**GUI**	**Scripts**	**Ataques**	**D. Ext.**
Cooja	Java	CoAP/MQTT	sim	C/Python	não	não
NS-3	C++	-	sim	C++/Python	não	sim
COAP PS	Java	CoAP	não	java	não	não
MQTT Gen	Python	MQTT	não	Python	não	não
IDGF	java	MQTT	sim	java	sim	não
IoT-Flock	C++	CoAP/MQTT	sim	C++	sim	não
Scapy	Python	CoAP/MQTT	não	Python	não	sim
NetSim	C++	CoAP	sim	-	sim	sim

funcionalidades compreendidas como fundamentais para o desenvolvimento deste trabalho. Desta forma, deverá ser considerada aquela cujas características mais se aproximam das funcionalidades desejadas. Além disso, deverá ser considerada a complexidade afeta ao desenvolvimento de novas características e funcionalidades e, assim, permitir com que os objetivos do atual trabalho sejam atendidos plenamente.

De acordo com as especificações e características das ferramentas analisadas e apesar destas apresentarem características bem próximas, algumas destas ferramentas não atendem às necessidades necessárias para o desenvolvimento do atual trabalho. A maior parte das ferramentas apenas têm suporte nativo a um dos dois protocolos desejados ou não apresenta por padrão ataques que possam ser utilizados sem a necessidade de implementação adicional. Desta forma, as ferramentas Network Simulator 3, IoT *dataset* Generator Framework, MQTT Generator, COAP Protocol Simulator e Scapy foram imediatamente descartadas.

A ferramenta IoF-Flock apresenta um grande conjunto de características e requisitos identificadas como necessários para o desenvolvimento do atual trabalho, como a implementação de ataques na versão padrão da ferramenta e o suporte aos protocolos MQTT e CoAP. Porém, ela apresenta uma limitação relacionada a apenas possibilitar o endereçamento IPv4 aos dispositivos que compõem a simulação. Outro ponto de destaque é o de que não foi possível gerar o tráfego malicioso com essa ferramenta. Esta tentativa foi realizada em duas diferentes versões do sistema operativo Ubuntu (18.04 e 20.04), mas em ambas o tráfego malicioso não foi gerado. Devido a essas limitações, esta ferramenta foi desconsiderada, visto que não representa a tendência dos dispositivos IoT que é o endereçamento IPv6 e por não ter sido possível a geração do tráfego malicioso.

Dentre o conjunto de ferramentas apresentadas, acabaram por serem selecionadas não somente uma, mas duas ferramenta, a saber Contiki e Netsim. A primeira foi selecionada devido a já ter sido utilizada e num grande número de trabalhos, além de já possuir uma grande quantidade de protocolos previamente configurados e possibilitar que o código gerado possa ser exportado para dispositivos reais. A segunda possui a limitação de não implementar o protocolo MQTT, porém, possui a funcionalidade de incorporar fluxo de dados de ambientes externos dentro da própria ferramenta através do módulo *NetSim Network Emulator*, o que poderá suprir a limitação mencionada. Ainda, existem alguns ataques que podem ser incorporados à simulação e que podem ser encontrados no Github. Um ponto em comum entre ambas as ferramentas é que novas funcionalidades podem ser facilmente adicionadas após o desenvolvimento destas funcionalidades.

3.13 SOLUÇÕES DE GERAÇÃO DE FLUXO DE DADOS

Um dos objetivos inicialmente avaliados para o atual trabalho era gerar fluxos de dados, possivelmente de acordo com o padrão IPFIX, a partir dos ficheiros PCAP gerados nas simulações e, com isso, analisar se é possível identificar ataques através das estatísticas obtidas ao utilizar *machine learning*. Este tipo de análise é uma alternativa ao da análise realizada diretamente nos pacotes dos ficheiros PCAP.

Inicialmente, fluxo de dados são basicamente estatísticas relacionadas ao tráfego (uni ou bilateral) gerado pelos componentes da rede, que normalmente é gerado com base nos ficheiros PCAP. Alguns exemplos de estatísticas são:

- Tempo total da comunicação;
- Portos de origem e destino;
- IP de origem e destino;
- Total de *bytes* e *frames* trafegados.

3.14 ANÁLISE DAS FERRAMENTAS DE GERAÇÃO DE FLUXO

Quando se fala em flux de dados em redes de computadores, existem alguns protocolos como IPFIX (Trammell et al., 2013) e NetFlow (Claise, 2004) que são alguns dos padrões adotadas para apresentar *metadados* do tráfego de dados. Algumas ferramentas estabelecem o seu padrão com base nos principais padrões, em sua maioria, podendo-se estabelecer uma relação entre os campos do *output* destas ferramentas com, por exemplo, os campos do padrão IPFIX.

Dentre as ferramentas utilizadas para o fim de gerar fluxo de dados, podem ser citadas:

- Tranalyzer[17];
- Tshark[18];
- YAF[19];
- Zeek[20].

17 https://tranalyzer.com/
18 https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html
19 https://tools.netsa.cert.org/yaf/index.html
20 https://zeek.org/

Estas ferramentas são utilizadas para exportar as estatísticas dos tráfegos de dados a partir dos pacotes contidos em um ficheiro PCAP. Como já dito anteriormente, estes pacotes representam as comunicações realizadas entre os diferentes componentes da rede durante as simulações.

A ferramenta YAF é geralmente utilizada juntamente a outras duas ferramentas, super_mediator[21] e SILK[22]. Este conjunto de ferramentas realiza a geração do fluxo de dados e a geração dos rótulos deste fluxo, levando em consideração os parâmetros definidos nos ficheiros de configuração.

A ferramenta Zeek realiza as mesmas operações que a ferramenta YAF, mas sem a necessidade de integração com outras ferramentas. Um ponto de destaque é o de que o *output* desta ferramenta é diferente do da ferramenta YAF, mas os valores do *output* ainda podem ser relacionados com os valores do padrão IPFIX.

A ferramenta Tranalyzer é um gerador de fluxo e analisador de pacotes de rede. A ferramenta, que é *open-source* e implementada em C com base na biblioteca libpcap[23], fornece funcionalidades para gerar estatísticas de pacotes de redes *Tranalyzer - About* s.d.

A ferramenta tshark, que é um analisador de protocolos de rede, é versão de linha de comandos da ferramenta Wireshark. Ela já está incluída em parte dos sistemas operativos *open-source*, tal como o Ubuntu.

As primeiras ferramentas a serem testadas foram YAF, Zeek e tranalyzer. Estas ferramentas foram testadas através de imagens Docker, cujas imagens podem ser encontradas na diretoria *docker-files*[24] do repositório do Github. Ao executar os *containers* destas ferramentas, foram apresentados erros que foram identificados como sendo relacionados aos protocolos RPL e IEEE 802.15.4 (LR-WPAN). Foram também realizados testes em ambientes físicos, não virtualizados, como forma de validação destes erros. Porém, foram apresentados os mesmos erros encontrados também nos ambientes conteinerizados. Desta forma, não foi possível gerar o fluxo de dados com estas três ferramentas (YAF, Zeek e Tranalyzer), em virtude de as comunicações serem realizados com base no protocolo RPL e as ferramentas não possuírem suporte para o protocolo em questão.

A última das ferramentas, tshark, foi testada somente em ambientes não conteinerizados. através desta ferramenta, foi possível obter alguns resultados dos fluxos

21 https://tools.netsa.cert.org/super_mediator/index.html
22 https://tools.netsa.cert.org/silk/index.html
23 https://www.tcpdump.org/
24 https://github.com/tiezermelo/contiki/tree/master/utils/docker-files

de dados dos protocolos TCP, UDP e ICMPv6. Os comandos utilizados nos testes são apresentados no repositório do Github.

Porem, não foi possível obter resultados referentes ao protocolo RPL, cujo está relacionado com a maior parte dos ataques deste trabalho.

Desta forma, apenas foi possível gerar o fluxo de dados do tráfego baseados em TCP e UDP através da ferramenta tshark. Outro ponto de destaque é a limitação desta ferramenta em relação à geração de estatísticas relacionadas ao protocolo ICMPv6, que também está relacionado com o protocolo RPL que é o protocolo relacionado a maior parte dos ataques selecionados para compor este trabalho.

Uma alternativa para esta limitação é a implementação de um algoritmo que realize o processo de vincular os dados dos ficheiros PCAP e realizar a geração das estatísticas destes dados. Esta representação estatísticas deve ser compatível com o *output* padrão dos modelos IPFIX e relacionados ou, minimamente, com alguma das ferramentas citadas.

Devido às limitações apresentadas, a opção de análise por fluxo de dados foi desconsiderada, em que esta foi substituída pela análise dos atributos associados a pacotes de dados. Esta análise consiste basicamente na utilização de ficheiros PCAP ou através de algum outro tipo de representação, como o CSV.

4

DEFINIÇÃO E CARACTERIZAÇÃO DO AMBIENTE E SIMULAÇÃO

O número de subgrupos das *smart cities* tem crescido ao longo dos últimos anos, especialmente no caso de edifícios que utilizam de controlos inteligentes. Assim como já acontece com o *smart city*, o conceito *smart farming* vem crescendo ao longo dos anos e já faz parte de muitas aplicações em diversos países do globo, já que são uma alternativa para uma produção sustentável em ambientes controlados áreas com poucos metros quadrados.

Neste capítulo será realizada a definição das características dos ambientes ou serviços a serem simulados, assim como a apresentação dos ataques selecionados para comporem os *datasets*.

4.1 CARACTERÍSTICAS E DEFINIÇÕES GERAIS

Os ambiente desenvolvidos neste trabalho tem como objetivo retratar uma *smart city* no contexto de um conjunto de apartamentos com sistema de controlo inteligente e uma *smart greenhouse* no âmbito das *smart farmings*. A definição da escolha pelas *smart greenhouses* ocorre devido ao crescente número deste tipo de ambiente a nível mundial. Os ambientes proposto têm o objetivo de representar um ambiente real dentro dos dois contextos apresentados.

No que se refere ao ambiente do conjunto de apartamentos, cujo ambiente é presentados na Figura 2, estão presentes 70 sensores ao longo dos 2500 metros quadrados. A lista de serviços disponíveis é apresentada a seguir:

- Controlo de acesso físico;
- Sistema de identificação facial;
- Sistema de videovigilância;
- Sensores de presença;
- Iluminação inteligente;

- Sensores de incêndio;
- Sistema anti-chamas.

Em relação às *smart greenhouses*, o ambiente considerado é representado na Figura 3. Este ambiente será composto por um total de 92 dispositivos dispostos ao longo do ambiente, que realizam a coleta de dados das seguintes características:

- Água no Solo;
- CO2 (Dióxido de Carbono);
- Luminosidade;
- Nutrientes do Solo;
- Temperatura;
- Umidade.

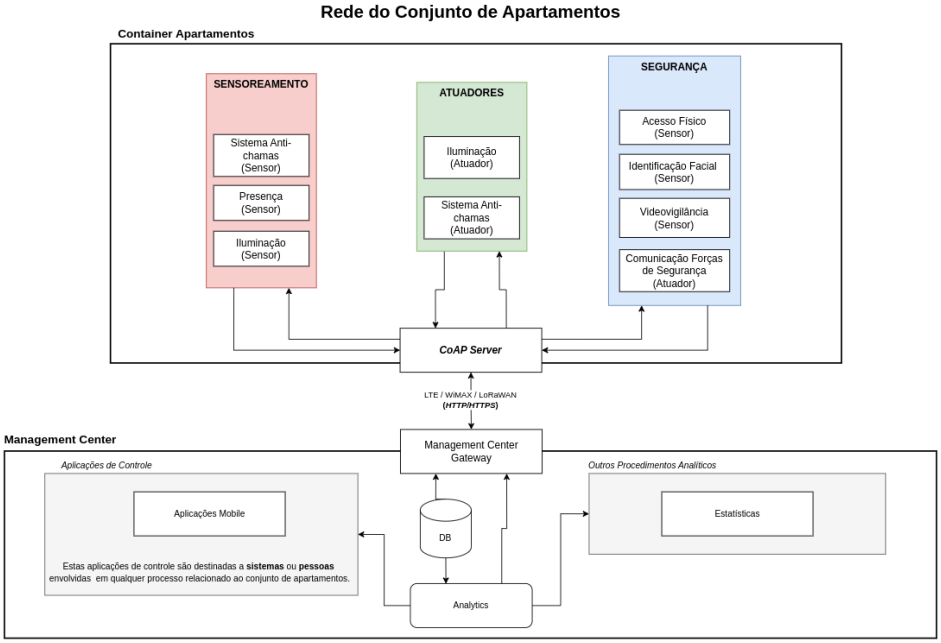


Figura 2: Representação das tecnologias e disposição dos sensores no ambiente do conjunto de apartamentos.

4.2 DEFINIÇÃO DO ÂMBITO DAS SIMULAÇÕES

O primeiro ponto a ser considerado é identificar os principais serviços no âmbito dos dois contextos definidos e, assim, selecionar os que serão considerados nas simulações.

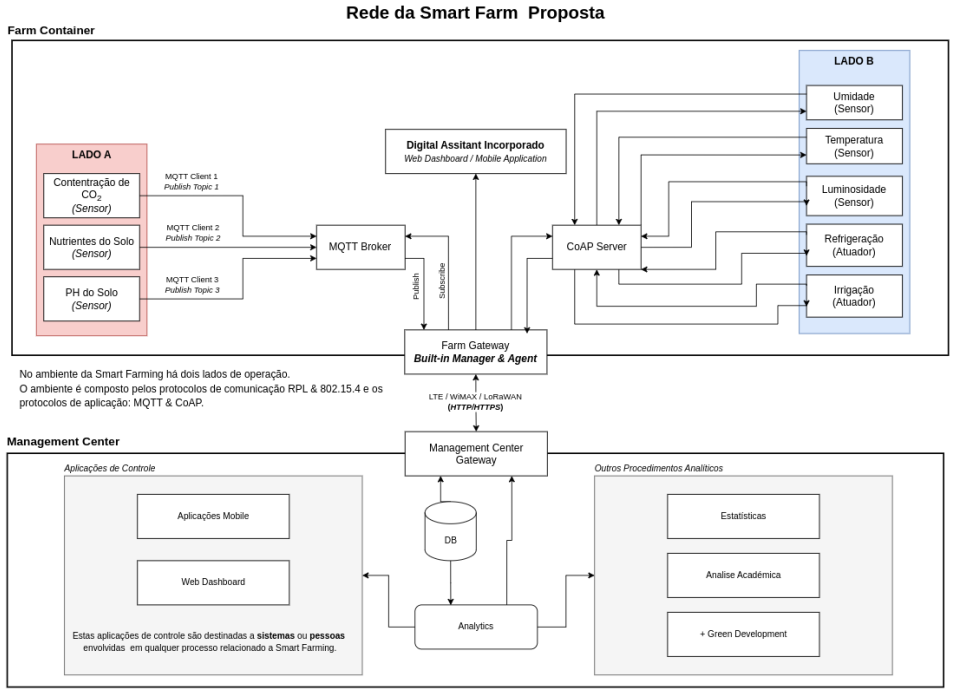


Figura 3: Representação das tecnologias e disposição dos sensores no ambiente da *smart greenhouse*.

No contexto do conjunto de apartamentos com sistemas inteligentes de controlo, foi feita a restrição para 2 serviços descritos na lista a seguir. No ambiente selecionado há, no total, 18 dispositivos, dentre eles sensores, servidores, roteadores e *gateways*.

- Controlo de acesso físico;
- Sistema de identificação facial.

Com o objetivo de determinar os serviços mais comumente utilizados em ambientes de *smart greenhouses*, foi realizada uma pesquisa na literatura que considera este tipo de ambiente, dando-se prioridade aos que estejam dentro do contexto do atual trabalho. De acordo com o que foi identificado, observou-se que os sensores mais utilizados envolvem irrigação, ventilação, controle de temperatura, concentração de gases, umidade e iluminação.

Com base nisso, foi proposto o ambiente composto por um total de 2 serviços, onde 42 dispositivos (dentre eles sensores, servidores e outros dispositivos de rede) realizam a coleta dos dados referentes a estes serviços. Dentro deste contexto, foram selecionados para fazerem parte da simulação os serviço abaixo apresentados:

- Concentração de dióxido de carbono;
- Temperatura.

Destaca-se que tais características foram selecionadas devido a também terem sido utilizadas em outros trabalhos identificados na literatura e também serem importantes, e muitas vezes fundamentais, para o bom desenvolvimento da produção em *smart greenhouses*.

Os sensores da simulação devem realizar a captura representativa dos dados referentes às quatro características ambientais citadas. Além disso, alguns destes serviços podem ser implementados no Contiki, pois existem sensores que emulam a leitura destas características do ambiente. Também, leva-se em consideração que a ferramenta Netsim não considera os valores da camada de aplicação especificamente, gerando valores aleatórios que representam esta camada.

Os serviços selecionados serão simulados em diferentes redes, isto é, cada um destes serviços será executado em uma rede específica e separada da rede de outros serviços. Optou-se por realizar-se a segmentação dos serviços em diferentes redes para que cada serviço seja executado em momento distinto e de maneira separada em cada ferramenta. A justificativa para isso decorre do fato de seguir boas práticas de segurança em redes de comunicação através do isolamento dos serviços.

É de salientar que os valores capturados nos sensores não serão considerados para fim de avaliação do pacote como ataque ou não. Tal opção ocorre devido a estes valores não serem relevantes para a análise do ataque em si. O processo de desenvolvimento do ambiente simulado será dividido em 6 estágios, conforme apresentado abaixo:

1. Realização das alterações necessárias nas ferramentas;
2. Desenvolvimento da infraestrutura;
3. Configuração da simulação;
4. Geração do tráfego normal;
5. Geração do tráfego malicioso;
6. Integração dos ficheiros PCAP, quando necessário.

4.3 CARACTERÍSTICAS TECNOLÓGICAS DAS SIMULAÇÕES

As simulações devem gerar um *dataset* que apresente heterogeneidade dos protocolos e dos ataques simulados. Um ponto considerado como importante para este trabalho é a necessidade de serem utilizados endereços IPv6 pelos sensores da simulação, de

forma a representar fidedignamente o que é encontrado na configuração da maior parte das redes IoT a nível mundial.

Em relação aos protocolos, deu-se preferência pela utilização de protocolos que sejam mais comumente utilizados em ambientes IoT. Desta forma, para a camada de aplicação, foram selecionados os protocolos CoAP e MQTT. Um ponto importante a ser considerado é o que não foram encontrados *datasets* que eram composto mutuamente por estes dois protocolos. Outro protocolo selecionado foi o RPL, por ser considerado importante para os ataques que serão empregados nas simulações. Além deste motivo, este mesmo protocolo é frequentemente utilizado para redes de sensores sem-fio de baixa potência, que é justamento o que se busca representar no atual trabalho.

4.4 CARACTERIZAÇÃO DOS AMBIENTES PROPOSTOS

Esta secção apresenta a arquitetura dos ambientes selecionados no contexto de *smart greenhouses* e *smart city*. Como visto na secção 4.2, os cenário propostos consideram quatro diferentes serviços a serem simulados pelas ferramentas. O tráfego gerado seria posteriormente transformado em fluxo de dados para servir como treinamento e ser também analisado pelo sistema de deteção. Porém, devido à limitação apresentada na subsecção 3.14, a análise por fluxo de dados foi alterada para a análise de atributos sobre os pacotes de dados dos ficheiros PCAP das simulações.

Para fazer parte da simulação, foram selecionados os protocolos da camada de aplicação mais comumente abordados na literatura no contexto da *Internet of Things*, que são os protocolos COAP e MQTT. Ainda, com base nos protocolos da camada de aplicação, naturalmente foram selecionados os protocolos TCP e UDP da camada de transporte. Além destes, foi definido o RPL como protocolo de rede em ambas as ferramentas e que servirá como base para a maior parte dos ataques escolhidos.

Os serviços selecionados serão divididos e representados nas duas ferramentas selecionadas. As características de cadas um dos ambientes será apresentada nas subsecções 4.4.1 e 4.4.2.

4.4.1 *Composição da rede na ferramenta Netsim*

Na ferramenta Netsim, será representado serviço referente à coleta dos dados referentes ao controlo de acesso físico e ao sistema de identificação facial. A rede será composta por sensores, *6LoWPAN Gateway*, *router* e *server* (representado por um componente do tipo *node*). Os sensores da rede sem-fio utilizam apenas endereços IPv6, enquanto que *routers* e *nodes* utilizam apenas endereços IPv4. A exceção é o componente *6LoWPAN Gateway* que utiliza tanto endereços IPv4 quanto IPv6.

Foram selecionadas a quantidade de 9 sensores para referenciar o sistema de identificação facial e a quantidade de 3 sensores para representar a captura do controlo de acesso físico. Considera-se que o número elevado de sensores não é um contributo para o desenvolvimento do atual trabalho e uma grande quantidade de sensores apenas geraria uma complexidade adicional desnecessária.

A captura dos dados não é automaticamente realizada no âmbito da simulação toda, mas deve ser configurada em cada um dos dispositivos em que se pretende capturar o tráfego. É importante salientar que o dispositivo do tipo *6LowPAN Gateway* não possui o recurso para realizar a captura dos pacotes. Para cada um dos sensores da rede sem fio, deverá ser feita a configuração para captura do tráfego, assim como para *routers* e nós (com e sem-fio), como pode ser visto na Figura 4. Os dispositivos do tipo *router* criam um ficheiro PCAP para cada uma das *interfaces* de rede.

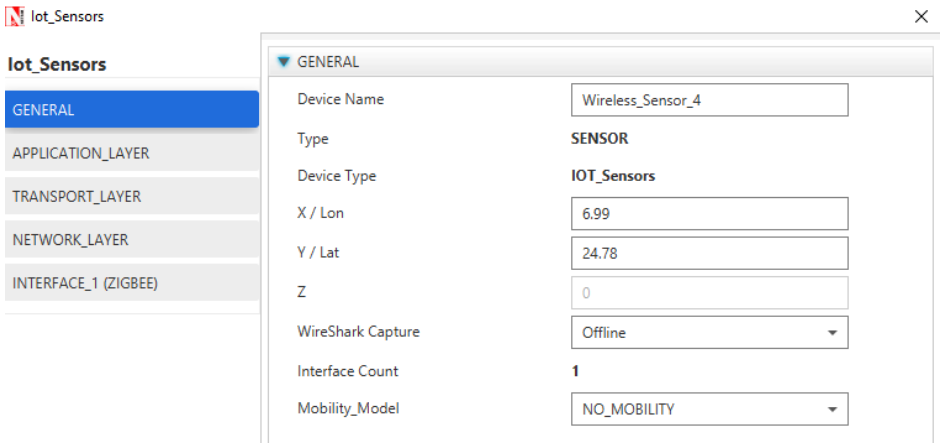


Figura 4: Configuração no componente para geração do ficheiro PCAP na ferramenta Netsim.

4.4.2 *Composição da rede na ferramenta Contiki*

O cenário simulado no Contiki será composto por 10 motes (sensores) executando os protocolos MQTT v3.1.1 (clientes apenas, sendo usado o *broker* da biblioteca mosquitto[1] a nível do *host*) e CoAP (clientes e *server*). Para a simulação, serão utilizados o mote do tipo Zolertia Z1 e Wismote, cujos ficheiros utilizados e as alterações necessárias serão apresentados na secção 5.1.1. Este ambiente representará os serviços de captura dos valores referentes à concentração de dióxido de carbono e à temperatura.

Os clientes CoAP, que representam a simulação de sensoreamento da temperatura, realizarão operações/requisições do tipo *POST* para o servidor CoAP com a informação do serviço virtualmente coletada ainda dentro da rede do Contiki.

Os clientes MQTT, que simulam o sensoreamento da concentração de dióxido de carbono, farão o envio das informações coletadas para o *broker* mosquitto fora da rede do Contiki, sendo utilizado a rede virtual criada pelo RPL *border router* através da ferramenta TUNSLIP[2].

A comunicação dos clientes, localizados na rede da ferramenta Contiki, com os *servers*, localizados e executados no *host*, ocorre a partir de um *border router* que utiliza a ferramenta TUNSLIP, que possibilita a comunicação entre o *border router gateway*, e consequentemente os sensores emulados pela ferramenta Cooja, e a rede externa ao Contiki (rede do *host*), como acontece com o *broker* mosquitto utilizado nas simulações referentes ao protocolo MQTT.

4.5 CARACTERÍSTICAS DA SIMULAÇÃO

A simulação será realizada nas duas ferramentas selecionas, Netsim e Contiki, de forma a contemplar uma maior diversidade de características aos *datasets*. Com base nisso, será criado um *dataset* para cada uma das ferramentas, uma vez que os protocolos gerados em cada uma das ferramentas podem conter *flags* atribuídas de forma diferente. O diagrama da topologia de rede e distribuição dos componentes da ferramenta Netsim são apresentadas respectivamente nas Figuras 5 e 6. Já a

1 https://mosquitto.org/

2 Tunslip é uma ferramenta que possibilita a conexão do *host* e outro elemento de rede, normalmente um roteador de borda, a partir de interface de rede virtual criada com o protocolo SLIP através de uma conexão serial.

topologia de rede e respetiva distribuição dos componentes da ferramenta Contiki é apresentada na Figura 7.

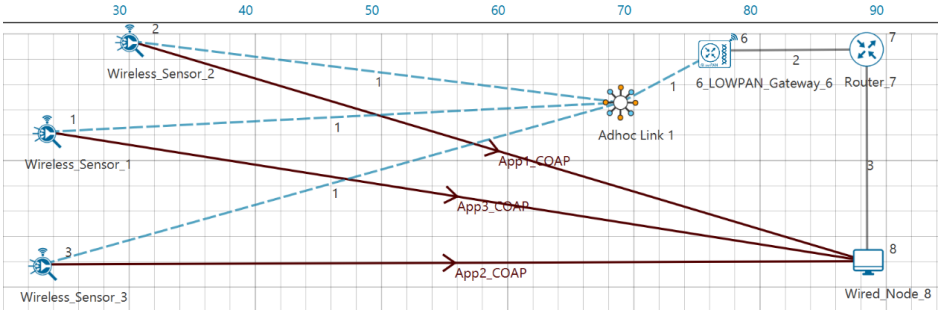


Figura 5: Topologias de rede das simulações com 3 sensores realizadas no Netsim.

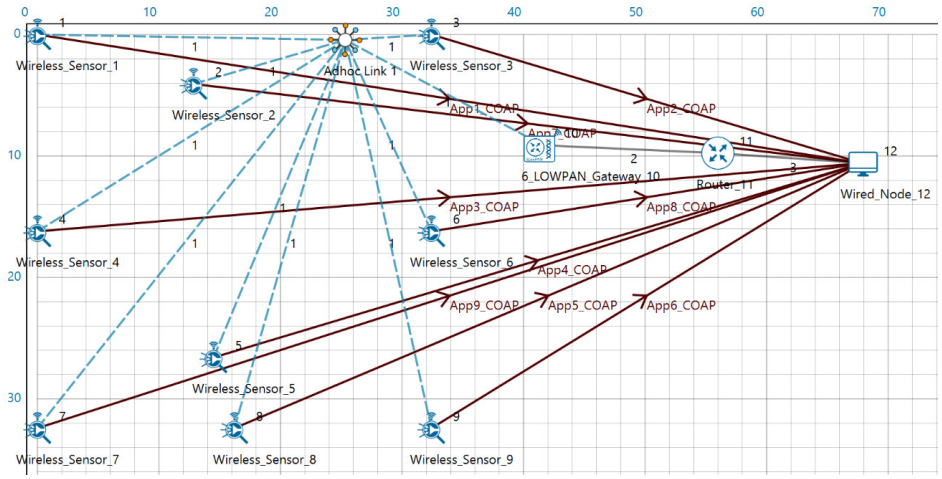


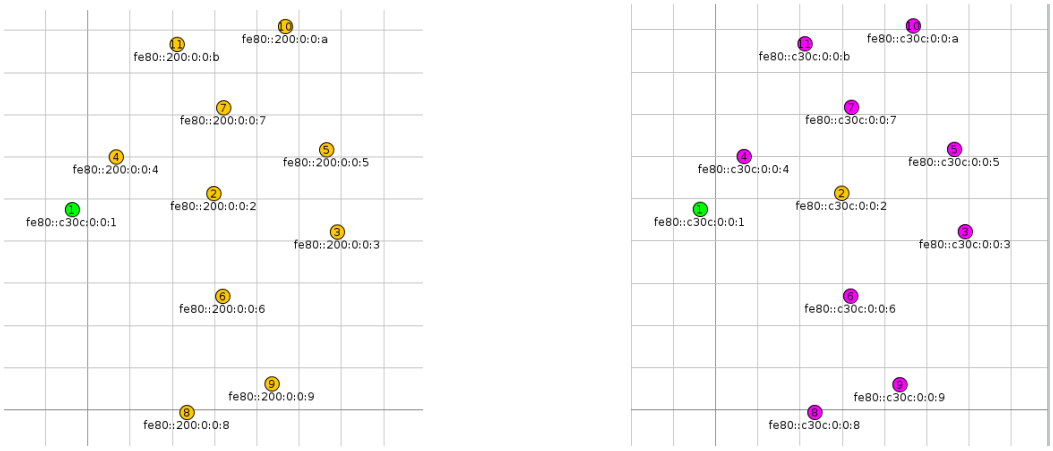
Figura 6: Topologias de rede das simulações com 9 sensores realizadas no Netsim.

Após estas configurações, poderão ser gerados os tráfegos normal e malicioso dos ambientes em cada uma das ferramentas.

4.6 TEMPORIZAÇÃO DAS SIMULAÇÕES

Em relação ao tempo de execução para cada simulação no Netsim, foram definidos diferentes períodos de tempo, variando de 300 a 1 000 segundos. Considera-se que este tempo é o suficiente para gerar um tráfego normal com dados o suficiente. Como apresentado na secção anterior, ao término de cada simulação, são gerados ficheiros PCAP para cada dispositivo e, caso seja de interesse, estes ficheiros poderão ser integrados em um único ficheiro PCAP através da ferramenta Wireshark[3] ou através do comando mergepcap[4]. No caso deste trabalho, foi gerado um pequeno *script* que

3 https://www.wireshark.org/
4 https://www.wireshark.org/docs/man-pages/mergecap.html



(a) Simulação do protocolo MQTT.

(b) Simulação do protocolo CoAP.

Figura 7: Topologias de Rede das simulações realizadas no Contiki.

realiza a identificação de todos as diretorias que contêm os dados referentes a cada execução de simulação e, após isso, integra (*merge*) os diversos ficheiros PCAP de uma simulação em um único ficheiro PCAP. Na Listagem 1 é realizada a inclusão do código que realiza a integração dos diferentes ficheiros PCAP em um único ficheiro.

Listagem 1: Código fonte Python utilizado para integrar os diferentes ficheiros PCAP em um único ficheiro.

```
import os

pcap_path = ""
print("\n\n--- Getting file names ---")
dirs = os.listdir(pcap_path)
for dir in dirs:
    if(dir.split('-')[0] != 'coap'):
        continue

    print("\n\n--- Directory:  " + dir)
    list = os.listdir(pcap_path+dir)
    str = ''
    for el in list:
        if el.endswith(".pcap"):
            str = str + ' ' + dir + '/'+ el

    print(str)
    os.system("mergecap -w merged/" + dir + ".pcap " + str)
```

No que se refere às simulações a serem executadas na ferramenta Contiki, não foi estabelecido um período de tempo para a duração da simulações tanto para o tráfego normal como para o tráfego malicioso. Como não há uma configuração específica para definir o tempo máximo da simulação, o término da mesma deverá ser realizado manualmente enquanto na execução da simulação na ferramenta Cooja. Deve ser destacado que deverá ser habilitado o recurso de captura dos dados no formato

PCAP, pois a ferramenta não realiza este procedimento de forma automática. As simulações serão executadas em diferentes momentos e repetidas por mais de uma vez. Cada simulação de ataque será exportada para um ficheiro PCAP exclusivo.

4.7 ATAQUES SELECIONADOS

Buscou-se fazer com que o tráfego de dados dos ataques conseguisse representar com o máximo de veracidade as características de um ataque real. Para gerar uma maior diversidade para os *datasets*, foram utilizados ataques que estão relacionados com mais de uma camada da arquitetura IoT. Para isso, serão utilizados protocolos das camadas de transporte e de redes.

A ferramenta Netsim possui uma lista de ataques que podem ser implementados nas simulações da ferramenta[5]. O código fonte de tais ataques pode ser encontrado no repositório do Github[6] mantido pela própria empresa. Dentre as possibilidades para a versão 13.1 da ferramenta, foram selecionados os seguintes ataques para compor o *dataset*:

- SinkHole attack in IoT RPL[7];
- DoS Attack in Iot[8];
- DIO Supression Attack[9].

As Figuras 8, 9 e 10 apresentam os ambientes das simulações da ferramenta Netsim dos ataques RPL *DIO Supression*, DoS e *Sinkhole* respetivamente.

O ataque RPL DIS Flooding[10] foi também considerado e algumas simulações foram executadas. Porém, quando habilitada a opção para gerar os ficheiros PCAP, era apresentado o erro da Figura 11 e os dados do sensor ID 1, que representava o sensor malicioso, não eram gerados. Desta forma, este ataque foi desconsiderado para o atual trabalho.

Como apresentado anteriormente, a ferramenta Contiki não possui ataques implementados de forma padrão no código original da ferramenta. Desta forma, foi necessária a implementação de funcionalidades que simulem o comportamento dos ataques pretendidos. Alguns ataques relacionados com o roteamento com RPL foram

5 https://tetcos.com/file-exchange.html
6 https://github.com/NetSim-TETCOS?tab=repositories
7 https://github.com/NetSim-TETCOS/SinkHole_attack_in_IoT_RPL_v13.1
8 https://github.com/NetSim-TETCOS/DOS_Attack_IoT_v13.1
9 https://github.com/NetSim-TETCOS/DIO_Suppression_v13.1
10 https://github.com/NetSim-TETCOS/RPL_DIS_FLOODING_v13.1

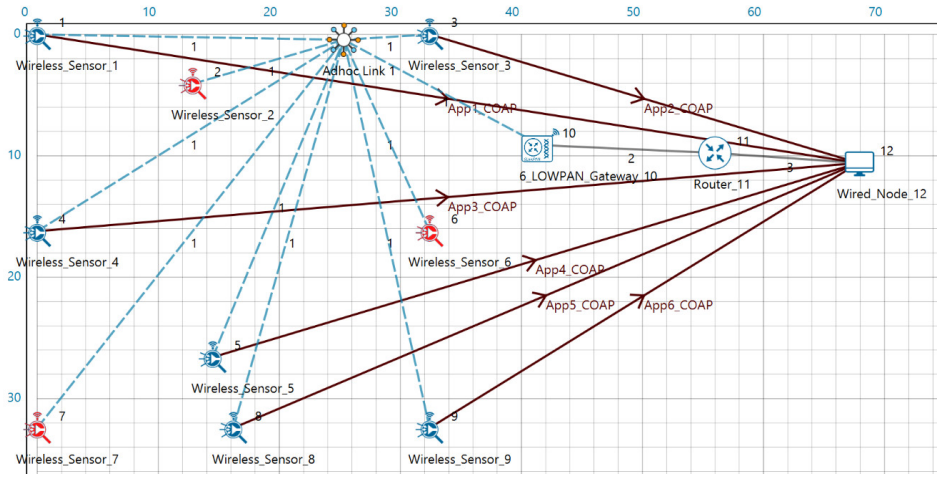


Figura 8: Ambiente da simulação do ataque DoS na ferramenta Netsim.

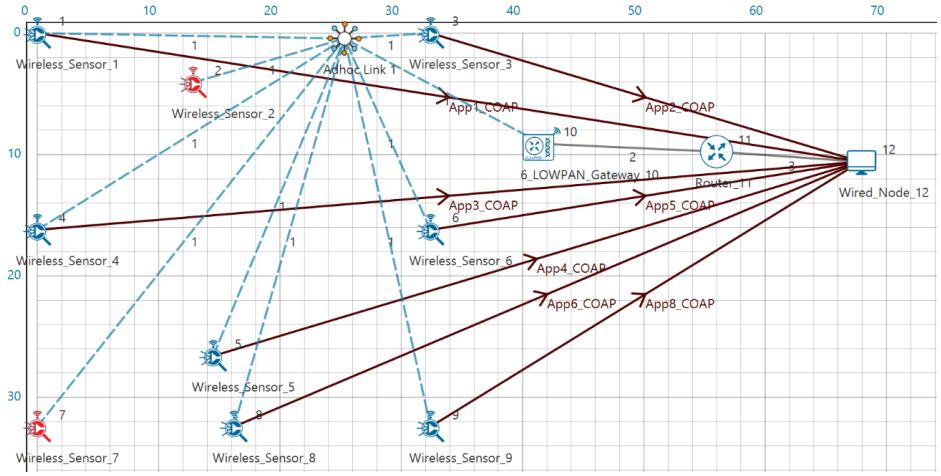


Figura 9: Ambiente da simulação do ataque *Sinkhole* na ferramenta Netsim.

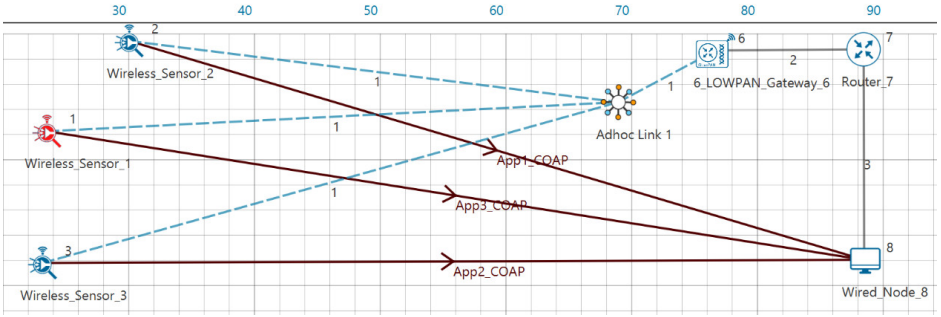


Figura 10: Ambiente da simulação do ataque DIO Supression na ferramenta Netsim.

implementados na ferramenta RPL-attacks[11]. Uma característica importante desta ferramenta é a de que após a execução da simulação, é gerado automaticamente

11 https://github.com/dhondta/rpl-attacks

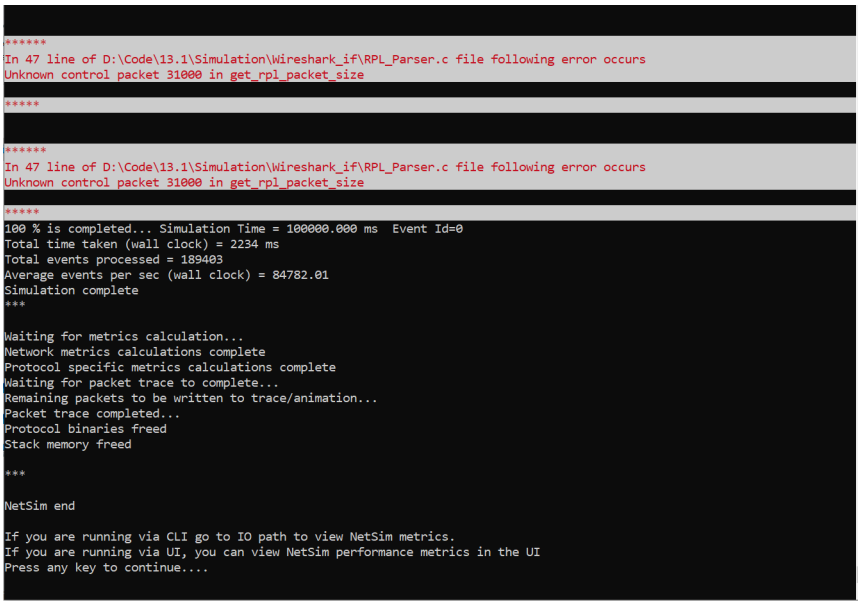


Figura 11: Erro apresentado na simulação do ataque *RPL DIS Flooding* quando habilitada a opção de gerar os ficheiros PCAP.

um relatório para cada um dos ataques selecionados com o resumo da simulação. A ferramenta implementa os seguintes ataques RPL:

- *Blackhole attack*;
- *Versioning attack*;
- *Flooding attack*.

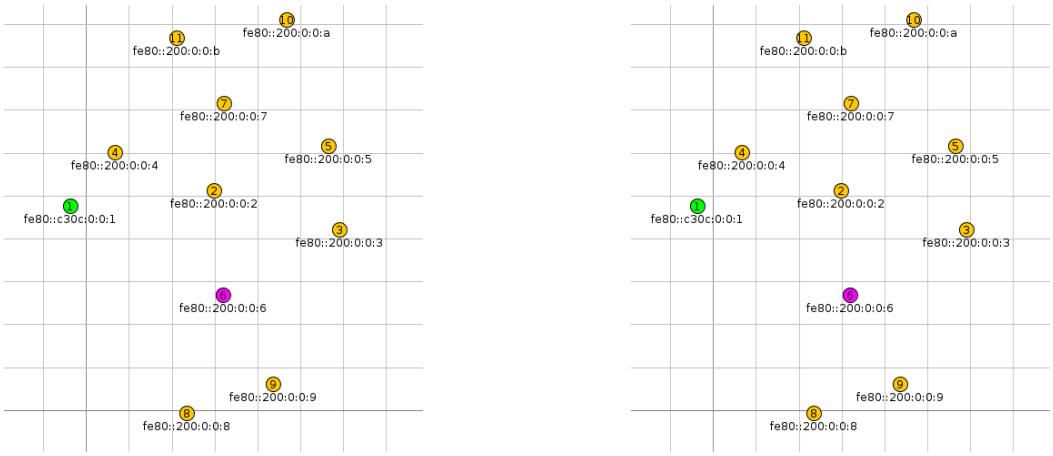
Tais ataques foram adaptados à ferramenta Contiki e o código utilizado pode ser encontrado no repositório do Github[12]. A configuração da ferramenta, sensores e ataques foi automatizada através de um *script*, que será descrito na secção 5.2. As Figuras 12 e 13 apresentam as topologias de rede dos ataques nas simulações executadas na ferramenta Contiki quando utilizados os protocolos CoAP e MQTT.

4.8 LIMITAÇÕES DAS FERRAMENTAS

As duas ferramentas de geração de dados escolhidas apresentam algumas limitações em suas funcionalidades, que serão apresentadas a seguir.

No que se refere a ferramenta Netsim, apesar de esta apresentar a funcionalidade de possibilitar que sistemas externos fossem adicionados a simulação a partir de um modulo chamado *NetSim Network Emulator*, o comportamento de tal módulo não foi

12 https://github.com/tiezermelo/contiki/

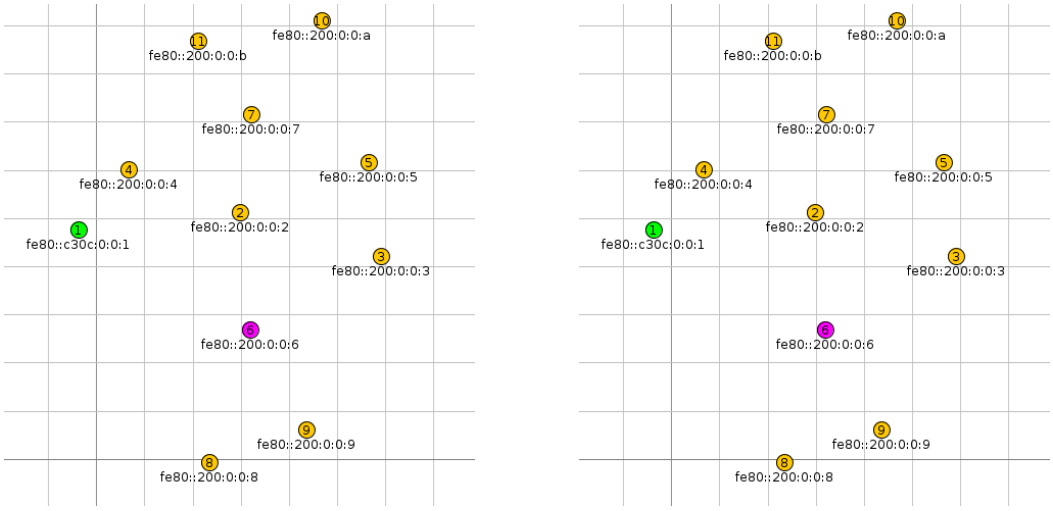


(a) Simulação com 1 sensor malicioso.

(b) Simulação com 2 sensores malicioso.

Figura 12: Topologias de rede das simulações dos ataques realizados com o protocolo MQTT no Contiki.

o esperado. Foram realizados experimentos com dois ou mais nós, que representariam a comunicação simples entre clientes e servidor. A ferramenta conseguiu capturar adequadamente o tráfego de dados gerados apenas quando a simulação continha exclusivamente dois nós (um único cliente e o *server*). Para os demais casos, cujo número máximo de nós testados foi o de cinco no total (*server* e quatro clientes), a ferramenta não foi capaz de gerir o fluxo de dados e capturar todos os dados trafegados, perdendo a maior parte dos pacotes provenientes dos clientes. Desta forma, optou-se por apenas utilizar a funcionalidade de simulação desta ferramenta, não incluindo sistemas externos a partir do módulo *NetSim Network Emulator*. A



(a) Simulação com 1 sensor malicioso.

(b) Simulação com 2 sensores maliciosos.

Figura 13: Topologias de rede das simulações dos ataques realizados com o protocolo CoAP no Contiki.

ferramenta Netsim ainda representa o protocolo COAP apenas como UDP, o que pode representar que a ferramenta não implementou o protocolo em si, mas apenas representa o comportamento de sistemas que utilizam este protocolo.

Já no que consta a ferramenta Contiki, a implementação do protocolo MQTT é referente a versão 3.1.1 e não há suporte para MQTT v5. Outro ponto negativo, mas de menor impacto, é o fato de não haver uma forma trivial de definir o tempo máximo para execução da simulação, cuja interrupção deverá ser realizada manualmente.

4.9 VERSÕES DOS SOFTWARES UTILIZADOS

Ainda, é apresentado na Tabela 2 a listagem dos *softwares* e respectivas versões utilizadas para realizar as simulações.

Tabela 2: Lista dos *softwares* utilizados na simulação e respectivas versões.

Software	**Versão**
Ubuntu OS	20.04.4 LTS
Contiki OS	commit/32b5b17f674232867c22916bb2e2534c8e9a92ff
Virtualbox	7.0.6
Netsim	13.1
Tshark	3.6.11
mergecap	3.6.11
Pandas	1.4.4
Numpy	1.22.4
matplotlib	3.7.1
Seaborn	0.12.2
Scikit-learn	1.2.2
split	9.0

5

SIMULAÇÕES E DATASETS

Neste capítulo serão apresentados os passos a serem seguidos para o desenvolvimento dos requisitos necessários para a fase de desenvolvimento deste trabalho. Em relação a estes requisitos, eles podem ser separados em diferentes etapas, cujas são apresentadas na lista abaixo:

1. Configurar as ferramentas;
2. Gerar o tráfego a partir das simulações;
3. Capturar os dados gerados nas simulações no formato PCAP;
4. Transformar PCAP em CSV;
5. Pré-processar os dados;
6. Gerar os *datasets*.

Estas etapas serão melhor apresentadas ao longo das próximas secções.

5.1 ALTERAÇÕES REALIZADAS NAS FERRAMENTAS DE SIMULAÇÃO

O tráfego de dados gerado teve como objetivo representar um cenário do mundo real dentro do contexto de *smart cities* e *smart greenhouses*. Para isso, foram utilizadas as ferramentas já apresentadas, nomeadamente Netsim e Contiki.

Para cumprir com os requisitos necessárias para a correta execução das ferramentas selecionadas, assim como o propósito do atual trabalho, foi necessária a realização de algumas mudanças a nível de código e a configuração de parâmetros na execução das simulações nestas ferramentas. Estas alterações serão descritas nas secções a seguir.

5.1.1 *Alterações necessárias na ferramenta Contiki*

Devido a algumas limitações que representam a capacidade de memória disponível nos motes do tipo Zolertia Z1, foi necessário alterar alguns valores para que seja possível compilar o código para este tipo de dispositivos. O código do exemplo *er-rest-example*[1] foi usado como base para a construção do código utilizado para os sensores clientes, *server* e maliciosos das simulações que utilizavam o protocolo CoAP.

Ainda, foram realizadas alterações nos ficheiros da diretoria core/net/rpl, uma vez que eram necessárias alterações a nível do protocolo RPL enquanto na simulações de cada um dos ataques. Os ataques selecionados realizavam alterações diferente a diferentes ficheiros desta diretoria, portanto, foi necessária a alteração destes ficheiros na execução de cada tipo de ataque.

Outro alteração necessária no código fonte original do Contiki-OS é a inclusão dos ficheiros referentes à implementação do protocolo MQTT para os motes do tipo Wismote. Este código não mais está incluso na última versão do Contiki no Github e foi necessário resgatar o código de uma versão anterior[2] e adicioná-lo a versão mais recente do sistema. Foi utilizado como base o código desenvolvido para o mote do tipo Zolertia Z1, em que foram feitas algumas modificações para se adequar ao novo tipo de mote.

Por fim, foram realizadas alterações nos valores a serem representados nos sensores. Para o caso do CoAP, que realiza a coleta da temperatura ambiente, foi atribuído um valor base de 22.4, sendo atribuído um valor aleatório na casa centesimal que varia de 0 a 9. Desta forma, os valores sensoreados variam de 22.40 a 22.49. No caso do sensores MQTT, foram também alterados os valores sensoreados, onde estes variavam de 0 a 99 também nas casas decimais o valor de base é 479. Portanto, para estes sensores, os valores variavam de 479.00 a 479.99.

5.1.2 *Alterações necessárias na ferramenta Netsim*

A simulação do ataque DIO *Supression* tem como dependência a ferramenta Matlab para ser executada corretamente. Caso o Matlab[3] não esteja instalado no sistema e,

1 https://github.com/tiezermelo/contiki/tree/master/examples/er-rest-example
2 https://github.com/contiki-os/contiki/tree/f373825c025295435d0de7293fc0dd089774f6a2/examples/z1/mqtt-demo
3 https://www.mathworks.com/products/matlab.html

caso esteja, se o caminho para o executável desta ferramenta não esteja devidamente configurado na variável de sistema *PATH* do Windows, a ferramenta não executará a simulação.

Devido a isso, foi necessária a eliminação, a nível de código, de todas as funções que realizavam as chamadas para as funções relacionadas ao Matlab. Ao todo, foram editados 3 ficheiros, onde foram realizadas a eliminação das chamadas as funções mencionadas. Na simulação dos ataques DoS, foi feita a configuração de 3 sensores maliciosos a nível de código. Para isso, foi necessária a alteração de 2 ficheiros. Na simulação dos ataques *Sinkhole*, foi feita a configuração de 2 sensores maliciosos a nível de código. Para isso, foram também alterados 2 ficheiros.

Ao fim de cada alteração, foi necessária a recompilação do projeto para que as alterações fossem aplicadas ao Netsim e as simulações pudessem ser realizadas.

5.2 PREPARAÇÃO DO AMBIENTE DA FERRAMENTA CONTIKI

Foram desenvolvidos dois *scripts* que realizam a configuração do sistema Linux Ubuntu utilizado para prepará-lo para a execução da ferramenta Contiki. Os dois *scripts* criados foram:

- preparing.sh;
- config_system.sh.

Os *scripts* podem ser encontrados no repositório do Github, mais especificamente na diretoria *utils*[4]. O primeiro *script*, de nome preparing.sh, automatiza o processo de configuração do sistema operativo. Neste processo, é realizado o *download* de dependências e do compilador MSP430-GCC, assim como a definição de algumas variáveis de ambiente e definição da versão Java 8 como *default* do sistema.

O segundo *script*, de nome config_system.sh, automatiza a configuração da ferramenta Contiki. Dentre os processos executados por ele, podem ser citados:

- Clonar o repositório mspsim[5];
- Adicionar ao *Path* do sistema a diretoria do compilador MSP430-gcc;
- Criar os ficheiros respetivos a cada ataque;
- Copiar e substituir os ficheiros RPL usados nos ataques;

4 https://github.com/tiezermelo/contiki/tree/master/utils
5 https://github.com/contiki-ng/mspsim

- Criar git *branches* para cada ataque;
- Compilar os binários que representam os sensores normais e maliciosos e o dispositivo *border router* RPL.

Como pode ser visto na lista acima, os ataques foram configurados em diferentes git *branches*. Esta opção se deu pelo fato de se considerar mais adequado a separação de cada ataque em seu respetivo *branch* para que não houvesse nenhum tipo de iteração entre diferentes configurações da ferramenta. Assim, tem-se um maior isolamento entre os ambientes de cada um dos diferentes tipos de tráfego. A lista dos *branches* criados é apresentada abaixo:

- black-hole;
- hello-flood;
- version-number.

Por fim, o *branch master* contém o código para executar as simulações relacionadas ao tráfego normal. A sequência de comandos abaixo exemplifica como alterar o *branch* da diretoria Contiki no caso de se querer executar uma simulação do ataque *blackhole*. Para isso, podem ser seguido o comandos:

```
$ cd <CONTIKI_PATH>
$ git checkout black-hole
$ git branch
```

Cooja Simulator - Contiki (2016) apresenta os passos necessários para executar uma simulação na ferramenta Cooja através do Contiki OS. Deve ser destacado que somente as simulações respetivas a cada *branch* poderão ser executadas com sucesso. Com base nisso, caso o utilizador tente executar a simulação do ataque *blackhole* em qualquer *branch* que não seja o *black-hole*, será apresentado um erro no momento do *build* da simulação, conforme apresentado na Figura 14

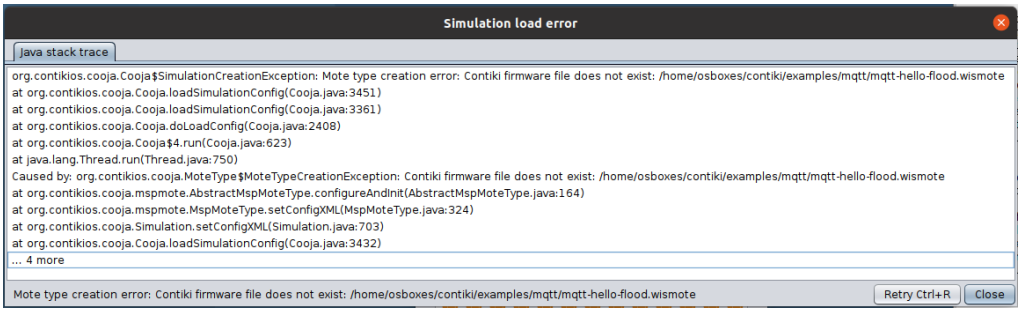


Figura 14: Erro apresentado ao tentar inicializar uma simulação em um *branch* diferente ao do ataque selecionado.

5.2.1 *Configuração da simulação na ferramenta Contiki*

A primeira configuração necessária é adicionar os sensores a simulação, portanto o utilizador deve selecionar os diferentes tipos de sensores disponíveis na ferramenta. Para isso, foi definido um modelo de disposição dos sensores no ambiente simulado e este modelo foi utilizado para ambos os cenários, CoAP e MQTT. Além disso, foram definidos cenários cujo número de sensores maliciosos era 1 ou 2 sensores. Para o caso das simulações do protocolo CoAP, o número de sensores era de um a menos se comparado ao número de sensores da simulação do MQTT. Isso ocorre porque um dos dispositivos da simulação era o servidor CoAP, diferentemente ao que ocorre nas simulações do MQTT, cujo *broker* não está dentro da rede simulada no Contiki (o *broker* está posicionado no sistema *host* e não na rede do Contiki). Desta forma, as simulações de ambos os protocolos possuem esta disposição:

- COAP: 1 RPL *Border Router* + 1 servidor CoAP + 9 sensores
- MQTT: 1 RPL *Border Router* + 10 sensores (*broker* está localizado no sistema *host*, fora do ambiente Contiki)

Uma segunda configuração necessária é habilitar a captura do tráfego e habilitar a opção de que este seja salvo no formato PCAP. Para isso, foi realizado o seguinte processo para habilitar esta funcionalidade: clicar em Tools > Radio messages..., e na tela que será aberta, selecionar Analyzer > 6LoWPAN Analyzer with PCAP. Após esta configuração, o tráfego gerado nesta simulação será salvo no formato PCAP na diretoria tools/cooja/build/ com o nome radio-<timestamp>.pcap

A terceira configuração e o único requisito para o correto funcionamento das simulações é a existência de um roteador de borda RPL e a execução do comando sudo make connect-router-cooja para conectar este router com a simulação realizada na ferramenta Cooja. Este requisito se dá pelo fato de que se não houver um roteador, não será atribuído IP ao dispositivos. E o comando é quem realiza esta atribuicao a partir do roteador. Para isso, abrir um terminal, ir a examples/ipv6/rpl-border-router e inserir o comando mencionado.

Após estas configurações, basta inicializar a simulação a partir do botão *Start* da janela *Simualtion control*. Para finalizar uma simulação, basta carregar no botão *Pause* na mesma janela.

A configuração das simulações realizados neste trabalho foram exportadas para ficheiros do tipo CSC (formato dos ficheiros relacionados a configuração das simula-

ções do Contiki) e podem ser encontrada na diretoria *simulations*[6] do Github. Esta simulações podem ser carregadas para uma nova execução do Contiki. Para isso, basta clicar em File > Open simulation > Browse, selecionar a diretoria simulations e escolher a simulação desejada. Porem, para cada tipo de simulação será necessário que o utilizador use o *branch* correto, conforme foi mencionado anteriormente.

5.2.2 *Configuração da simulação na ferramenta Netsim*

O protocolo da camada de aplicação a ser utilizado na simulação é definido no menu "Application - Traffic Generator". No caso deste trabalho, o protocolo definido no campo application_type da aplicação foi o CoAP. Algumas outras características podem ser selecionadas na tela de configuração da aplicação. Para representar a comunicação entre cada sensor e *server*, foi necessária a criação de uma aplicação, sendo definidos os identificadores (IDs) do sensor e do *server*. Ainda podem ser definidos o tempo de início e fim de cada aplicação, assim como se o tempo de início deve ser aleatório, o intervalo para encaminhamento da mensagem (*COAP REQUEST INTERARRIVAL TIME* - *constant* ou *exponential*), o tamanho da mensagem (*PAGE PROPERTY* - - *constant* ou *exponential*) e se haverá confirmação do recebimento da mensagem (*ACK REQUIRED* - *true* ou *false*). Apesar de poder ser configurado confirmação de recebimento de mensagem (ACK REQUIRED), esta funcionalidade foi limitada e não se notou diferença no comportamento da simulação, assim como não houve a adição de novos pacotes com este propósito nos ficheiros PCAP.

No que se refere à tela de execução da simulação, foram utilizados diferentes valores para o campo "*Simulation time*", cujos valores variaram de 300 a 1 000. No que se refere a opção "*Packet Animation*", foi selecionada a opção "*Don't Play. Record Animation (Simulation will run fast)*". Esta última opção está relacionada com uma animação para visualizar a troca de pacotes entre os dispositivos da rede que compõem a simulação. A opção selecionada faz com que a animação não seja executado ao termino da simulação, mas seja apenas armazenada para ser visualizada futuramente, caso seja de interesse do utilizados.

6 https://github.com/tiezermelo/contiki/tree/master/simulations

5.3 REALIZAÇÃO DAS SIMULAÇÕES

O tráfego de dados foi aplicado ao longo de um período de diversos dias. A duração de cada simulação não seguiu um período de tempo determinado, mas tinha como única limitação o limite de memória disponível nas máquinas virtuais utilizadas para executar a ferramenta Contiki. Neste sentido foi determinado um limite de até 130 000 pacotes para as simulações realizadas no Contiki. No caso das simulações realizadas na ferramenta Netsim, não foi encontrada nenhuma limitação no que se refere a execução das simulações.

Para as simulações geradas a partir da ferramenta Contiki, foram utilizadas duas diferentes quantidades de sensores maliciosos nas simulações, a saber 1 e 2 sensores maliciosos. Desta forma, temos simulações com 1 ou 2 sensores maliciosos para cada um dos 3 ataques selecionados. Esta diferente composição contribui para que haja uma maior diversidade nos dados que compõem o *dataset*. O mesmo padrão não foi adotado nas simulações realizadas na ferramenta Netsim, portanto não há diferentes quantidades de sensores maliciosos para um mesmo tipo de ataque.

5.4 CAPTURA DO TRÁFEGO E GERAÇÃO DOS FICHEIROS PCAP

Os dados do tráfego gerado nas simulações foram capturados pelas próprias ferramentas, não havendo a necessidade de utilização de ferramenta externa para este fim. No caso da ferramenta Contiki, a cada nova simulação, os ficheiros PCAP foram armazenados no caminho ${CONTIKI_PATH}/tools/cooja/build/radio-<timestamp>. No caso da ferramenta Netsim, que também realiza a captura do tráfego automaticamente, os ficheiros PCAP eram armazenados no caminho C:\Users\%username%\AppData\Local\Temp\NetSim\13.1.

Foi seguido um padrão de nomenclatura para cada uma das capturas, de forma a facilitar a aplicação dos algoritmos de processamento dos ficheiros PCAP. A metodologia adotada para a nomenclatura dos ficheiros foi a seguinte: <protocolo>-<tipo_ataque>-<número_sensores_maliciosos>malic-<número_sensores_simulação>sensors-<id_captura>.pcap e <protocolo>-normal-<número_sensores_simulação>sensors-<id_captura>.pcap. A disposição das informações que compõem o nome do ficheiro segue a ordem apresentada abaixo:

1. Protocolo;
2. Normal ou tipo do ataque;

3. Número de sensores maliciosos e o termo "malic";
4. Número de sensores da simulação e o termo "sensors";
5. Identificador da captura (número inteiro e sequencial)

Com base neste padrão de nome adotado, podem ser apresentados os seguintes modelos utilizados:

1. coap-normal-9sensors-1.pcap: apresenta uma captura realizada com o protocolo CoAP, composta por tráfego não malicioso, com a quantidade total de 9 sensores e sendo a primeira captura a ter sido realizada;
2. mqtt-black_hole-2malic-10sensors-3.pcap: apresenta uma captura realizada com o protocolo MQTT, composta por tráfego anômalo com 2 sensores maliciosos que realizam o ataque *blackhole*, com a quantidade total de 10 sensores e cuja captura foi a terceira a ser realizada.

Esta representação de nome de ficheiro facilita a identificação dos protocolos utilizadas na simulação e possibilitam que o utilizador utilize apenas aquelas de interesse, caso este não queira utilizar a totalidade dos ficheiros. Além disso, este padrão de nomenclatura possibilitou a execução do pré-processamento dos dados através de um *script* criado pelo autor, cujo processo será descrito na secção 5.6.

Ao total, foram gerados 231 ficheiros PCAP, sendo que 179 destes ficheiros PCAP foram gerados através da ferramenta Contiki e os restantes 42 ficheiros PCAP foram gerados a partir da ferramenta Netsim. O nome de cada um destes ficheiros auxilia na identificação do tipo de tráfego que compõe cada um dos ficheiros PCAP.

5.5 DEFINIÇÃO DOS FILTROS DE PROTOCOLOS

Nesta fase, serão selecionados os filtros da ferramenta Wireshark, executados através da ferramenta Tshark, para extrair alguns dos valores dos pacotes contidos nos ficheiros PCAP. Cada um destes valores corresponde a um campo do pacote ao qual o filtro está relacionado. Por exemplo, o filtro de nome "ipv6.src"corresponde ao campo "*Source Address*"do protocolo IPv6. Desta forma, se este filtro for utilizado, o ficheiro CSV conterá uma coluna referente a este valor.

Portanto, será através destes filtros que os valores respetivos a cada um dos filtros serão exportados do ficheiro PCAP para o ficheiro CSV. É importante ressaltar que estes valores serão chamados de atributos quando na fase de aprendizado de

máquina. Um atributo descreve uma característica individual do *dataset*, sendo comumente representada através de colunas.

Optou-se por selecionar todos os filtros dos seguintes protocolos, de forma a não perder nenhuma informação que poderia ser importante para o algoritmo de aprendizagem.

- frame;
- wpan;
- 6lowpan;
- icmp;
- icmpv6;
- ipv4;
- ipv6;
- tcp;
- udp;
- coap;
- mqtt.

O significado do campo de cada um dos filtros selecionados, assim como a lista completa de todos os filtros possíveis, pode ser encontrada em (*Wireshark - Display Filter Reference* 2023).

Apesar de a quantidade total de filtros ser consideravelmente grande para alguns dos protocolos, será possível reduzir esta quantidade através de técnicas de seleção de atributos, que será discutida em momento oportuno.

Durante a execução do *script*, foram encontrados erros relacionados a alguns dos filtros da lista. Devido a isso, tais filtros foram eliminados do *script* final e, assim, foi possível gerar o ficheiro CSV corretamente. A lista contendo os filtros utilizados e os filtros descartados pode ser encontrado no repositório do Github[7].

5.6 PRÉ-PROCESSAMENTO DE DADOS

O conceito de pré-processamento está relacionado com o fato de gerar dados de maior qualidade, eliminando dados desnecessários e mantendo apenas os que

7 https://github.com/orgs/CIIC-C-T-Polytechnic-of-Leiria/repositories

contenham informações importantes. Estas etapas trazem inúmeros benefícios na fase de processamento dos dados. Assim, foi desenvolvido um *script* Python que realizou os seguintes processos de pré-processamento nos dados dos ficheiros CSV:

- Identificação de todos os ficheiros PCAP;
- Conversão dos ficheiros PCAP para CSV;
- Remoção de colunas compostas integralmente por valores vazio e constantes;
- Adição das colunas alvo (*IS_MALICOUS* e *ATTACK_TYPE*);
- Integração dos ficheiros CSV e geração do *dataset* final.

O primeiro passo realizado pelo *script* é identificar o nome de todos os ficheiros PCAP existentes em uma diretoria informada pelo utilizador na execução do *script*. Esta lista de nomes será utilizada ao longo de muitos dos processos que compõem o *script*. O *script* requer também a especificação da diretoria para onde os dados gerados pelos filtros serão exportados como ficheiros CSV.

5.6.1 *Conversão de ficheiros PCAP em CSV*

Ao *script* que contém todos os filtros utilizados, foi dado o nome *filters.sh*. O *script bash* executa o comando Tshark e aplica os filtros destacados a todos os ficheiros PCAP de uma diretoria determinada pelo utilizador. Foi atribuído ao nome do ficheiro um número que será utilizado na fase de integração dos ficheiros. Tal opção foi tomada para evitar que os ficheiros de um mesmo tipo de tráfego fossem integrados sequencialmente.

Este número estará entre o intervalo de 1 até a quantidade total de ficheiros PCAP. Por exemplo, para o caso dos ficheiros gerados pelo Contiki, os valores variam de 1 a 179. Em relação ao nome do ficheiro, podemos citar o exemplo do ficheiro mqtt-normal-10sensors-9.csv, ao qual foi atribuído o valor 177, onde o nome do ficheiro passou a ser 177-mqtt-normal-10sensors-9.csv.

Ao final desta etapa, os ficheiros CSV foram criados na diretoria especificada no inicio da execução do *script*. Estes ficheiros possuem os atributos correspondentes aos filtros da ferramenta Tshark, sendo na totalidade 1693 atributos.

5.6.2 *Separação dos dados*

Esta fase consiste em dividir os ficheiros CSV em ficheiros menores com a mesma extensão, sendo realizada somente para os ficheiros provenientes das simulações executadas na ferramenta Netsim. Esta necessidade decorre do fato de a quantidade de registos serem grandes em parte dos ficheiros, juntamente a grande quantidade de atributos do tipo *float64*. Essa combinação causava o estouro de memória na hora de realizar alguns dos processos do *script* de pré-processamento ao lidar com alguns dos ficheiros CSV.

Uma primeira tentativa de solucionar este problema foi identificar os atributos reportados como *float64* e fazer a conversão destes valores para *float32* e, posteriormente, *float16*, de forma a reduzir o consumo de memória, mas ainda assim, ocorreu o estouro de memória.

A alternativa definitiva foi a separação dos ficheiros maiores em ficheiros menores, isto é, dividir os ficheiros a cada 200 000 registos, o que resolveu o problema do estouro da memória. Para este fim, foi utilizado o comando split[8] do Linux, sendo utilizado a opção do comando de adicionar um prefixo ao nome dos ficheiros criados. Tal opção foi utilizada para manter o padrão de nomenclatura adotado e anteriormente apresentado.

Através deste comando, foi feita a divisão de um ficheiro a cada 200 000 registos e a cada novo conjunto de valor de mesma quantidade (200 000), um novo ficheiro era criado. Para clarificar esse processo, vamos utilizar o ficheiro coap-dio_supression-1malic-3sensors-3.csv, que possui o total de 627 631 registos. Na lista abaixo, são apresentados os ficheiros que foram criados a partir da execução do comando split sobre o ficheiro coap-dio_supression-1malic-3sensors-3.csv e a respetiva quantidade de registos de cada um dos ficheiros gerados:

- coap-dio_supression-1malic-3sensors-3-aa.csv - 200 000 registos;
- coap-dio_supression-1malic-3sensors-3-ab.csv - 200 000 registos;
- coap-dio_supression-1malic-3sensors-3-ac.csv - 200 000 registos;
- coap-dio_supression-1malic-3sensors-3-ad.csv - 27 631 registos.

Como pode ser visto, ao executar o comando split no ficheiro coap-dio_supression-1malic-3sensors-3.csv, foram criados 4 novos ficheiros, devido a quantidade de registos que o ficheiro continha originalmente. Observa-se também que a base do nome do

8 https://man7.org/linux/man-pages/man1/split.1.html

ficheiro originário foi mantida nos novos ficheiros, sendo apenas acrescido um sufixo ao nome (aa, ab, ac, etc.).

O exemplo acima apresenta a nomenclatura dos ficheiros apos a execução do comando split. Este mesmo processo foi repetido para todos os ficheiros e a divisão somente ocorria quando um ficheiro possuía uma quantidade maior que 200 000 registos.

5.6.3 *Limpeza dos dados*

Na etapa de limpeza dos dados, devemos executar ações para preparar os dados que serão processados pelos algoritmos de aprendizagem. Basicamente, o conjunto inicial de dados passará por diversas transformações até que ele atinja um padrão de maior qualidade. Esta etapas são fundamentais para o bom desempenho do modelo, uma vez que contribuem para as previsões e determinação de padrões nos dados, assim como para o desempenho do algoritmo em si já que uma menor quantidade de atributos implica na redução do tempo dispendido na fase de treinamento do modelo.

Nesta etapa, foi realizada a eliminação dos atributos que continham, para todos os seus registos, valores nulos ou vazios (identificados no *dataset* com o valor "nan") e valores contantes (um único e mesmo valor para todos os registos de um atributo). Este procedimento, também chamado de redução de dimensionalidade, contribui para a redução do número total de atributos, gerando um conjunto de dados de maior qualidade. É importante destacar que não foram executadas etapas de transformação de valores dos dados dos ficheiros CSV. Ao final desta etapa, os ficheiros CSV contém apenas os atributos que não são nulos ou contantes.

5.6.4 *Adição dos atributos-alvo*

Neste passo, são adicionadas os atributos que servirão como alvo dos algoritmos de aprendizagem. Foram adicionadas as seguintes colunas a cada um dos ficheiros CSV:

- *IS_MALICIOUS*: será usada na classificação binária e identifica se um registo é correspondente a tráfego normal (valor 0) ou a tráfego malicioso (valor 1);

- *ATTACK_TYPE*: é utilizada na classificação multi-classe e é utilizado na identificação do tipo de ataque ao qual o registo corresponde. Caso corresponda a tráfego normal, será inserido o valor 0. Caso o registo seja referente a um ataque, será inserido o valor correspondente ao tipo de ataque (valores variam de 1 a 7).

Os valores de cada uma das classes supracitadas sao apresentadas na Tabela 3.

Tabela 3: Valor respetivo a cada classe nos atributos *IS_MALICIOUS* e *ATTACK_TYPE*.

TIPO DE TRÁFEGO	*IS_MALICIOUS*	*ATTACK_TYPE*
Normal	0	0
Blackhole	1	1
Hello flood	1	2
Version number	1	3
DIO supression	1	5
Sinkhole	1	6
DoS	1	7

Para inserir os respetivos valores, foi utilizada a bibliotecas Pandas em conjunto com a Numpy[9]. Foi utilizado a atributo "ipv6.src"como forma de identificar o IP malicioso no caso em que o nome do ficheiros fosse diferente a normal. O *script* levava também em consideração a quantidade de IP maliciosos que era informado no nome do ficheiro e utilizava de a lista de IP maliciosos de nome *malicious_ip_list* e o dicionário de nome e código de ataque de nome *attacks_list*. Era então realizada uma iteração nesta lista e o registo correspondente a um dos IP da lista era configurado com o valor referente ao ataque do ficheiro corrente. É importante destacar que esta lista e este dicionário devem ser informados pelo utilizador no momento da execução do *script*.

Por exemplo, considerando um ficheiro CSV de nome "coap-black_hole-2malic-9sensors-2.csv", a lista a conter os valores "::1:2:3:4"e "::4:5:6:7"e o dicionário a conter o valor "black-hole: 1". O *script* irá atribuir o valor referente ao tipo de ataque identificado a partir do nome do ficheiro e identificará o valor respectivo ao ataque encontrado no dicionario, sendo considerado o valor 1 no caso do exemplo apresentado.

Portanto, o *script* atribuirá o valor 1 para todos os registos cujo valor do atributo "ipv6.src"seja igual a um dos valores contidos na lista de IP e atribuirá o valor 0 para os demais registos. Para o caso de o nome do ficheiro conter o termo normal, a

9 https://numpy.org/

lista de IP não era utilizada e todos os registos dos atributos *IS_MALICIOUS* e *ATTACK_TYPE* foram então marcados com o valor 0.

5.6.5 *Eliminação seletiva de atributos*

A penúltima etapa da fase de pré-processamento dos dados consiste na análise exploratória dos atributos que compõem o *dataset*. Nesta etapa foi realizada uma revisão dos atributos conservados após a execução do *script* de pré-processamento, de forma a selecionar somente os atributos considerados mais importantes para gerar a versão final de cada um dos *datasets*.

Como critério de eliminação, foi levado em consideração o atributo em relação ao protocolo correspondente. Como exemplo, podemos citar o atributo "ipv6.geoip.dst_city"que corresponde a descrição "*Destination GeoIP City*"no *Wireshark Display Filter Reference*. Este atributo foi considerado como não sendo importante para fazer parte do *dataset*, portanto foi descartado neste processo.

Fez-se então uma análise exploratória destes atributos, de forma a identificar aqueles que, em um aspecto mais observacional, foram considerados os maior importantes para cada um dos *datasets*.

5.6.6 *Geração dos datasets finais*

A última e não menos importante etapa do pré-processamento dos dados consiste na criação dos dois *datasets* que serão separados de acordo com a ferramenta a qual gerou tráfego que compõe ambos os *dataset*. Portanto, todos os ficheiros CSV gerados serão anexados em um dos dois ficheiros a seguir, a depender da ferramenta utilizada para gerar os dados. Nesta etapa, foi seguida a numeração atribuída a cada ficheiro, sendo seguida a ordem crescente dos números do nome de cada conjunto de ficheiros, que fora atribuído no momento da criação dos ficheiros CSV. A exemplificar, levando em consideração os ficheiros 1-coap-dos-3malic-9sensors-7.csv, 2-coap-sink_hole-2malic-9sensors-4.csv e 3-coap-sink_hole-2malic-9sensors-5.csv, será seguida a ordem crescente com base nos números dos nomes destes ficheiros, portanto (1 -> 2 -> 3 e assim por diante).

No final desta fase, foram gerados os *datasets* finais referentes as simulações de cada uma das ferramentas, sendo estes apresentados abaixo:

- smart_greenhouse: *dataset* gerado a partir dos dados provenientes da ferramenta Contiki;
- smart_city: *dataset* gerado a partir dos dados provenientes da ferramenta Netsim.

Ao termino das simulações, foram gerados os ficheiros CSV com base nos ficheiros PCAP (cujo número de ficheiros CSV é diretamente correspondente ao número de ficheiros PCAP). Os ficheiros passaram pela etapa de pré-processamento dos dados, onde foram eliminados os atributos vazios e constantes.

Cada um dos dois *datasets* será detalhadamente descritos em capítulo especifico. Por fim, os ficheiros CSV originais que foram utilizados para gerar os dois *datasets* serão também publicados de forma a contribuir para trabalhos futuros. Desta forma, um utilizador poderá utilizar somente os ficheiros de interesse.

5.7 DESCRIÇÃO DOS *DATASETS* GERADOS

Nesta secção, serão descritos os dois *datasets* criados.

Neste capítulo será realizada a descrição dos *datasets smart_greenhouse*, apresentado na subsecção 5.7.1, e *datasets smart_city*, apresentado na subsecção 5.7.2, que foram gerados a partir dos dados provenientes das simulações executadas nas ferramentas Contiki e Netsim respetivamente.

5.7.1 *Descrição do dataset smart_greenhouse*

O *dataset smart_greenhouse* foi gerado a partir dos dados provenientes da ferramenta Contiki e é composto por dados correspondentes a tráfego malicioso e normal. A Tabela 4 apresenta a quantidade de ficheiros CSV criados no total, assim como a quantidade de pacotes gerados em cada tipo de simulação.

A manutenção dos ficheiros CSV separados por tipo de tráfego e por protocolo facilita a realização futura de uma análise especifica ao domínio pretendido. Por exemplo, pode-se realizar a análise apenas do tráfego corresponde ao ataque black-hole com a utilização do protocolo MQTT, sem ter de fazer qualquer tipo de filtragem dos dados no *dataset* final. Para isso, basta que os ficheiros correspondentes sejam utilizados, isto é, usar todos ou parte dos ficheiros cujo nome contenha o termo "mqtt-black_hole".

Tabela 4: Número de ficheiros para cada tipo de simulação realizada na ferramenta Contiki

SIMULAÇÃO	NÚMERO DE FICHEIROS	TOTAL DE PACOTES
coap-normal	10	991 079
coap-black_hole-1malic	7	369 613
coap-black_hole-2malic	7	651 005
coap-hello_flood-1malic	12	800 611
coap-hello_flood-2malic	7	779 565
coap-version_number-1malic	12	1 120 589
coap-version_number-2malic	22	731 384
mqtt-normal	19	2 206 075
mqtt-black_hole-1malic	16	1 829 312
mqtt-black_hole-2malic	12	1 362 208
mqtt-hello_flood-1malic	18	1 933 396
mqtt-hello_flood-2malic	12	1 170 006
mqtt-version_number-1malic	13	1 212 173
mqtt-version_number-2malic	12	1 162 291
Total	179	16 319 307

É importante destacar que não foi possível utilizar a função do *script* de pré-processamento que faz a integração dos *datasets* parciais em um único *dataset* . Este problema se deu devido a limitação de memória do dispositivo utilizado. Portanto, para solucionar este problema, foi utilizado o código apresentado na Listagem 2 para gerar o *dataset smart_greenhouse* preveniente das simulações executadas no Contiki.

Listagem 2: Código fonte Python utilizado para gerar o *dataset smart_greenhouse*.

```
# SMART GREENHOUSE FINAL DATASET GENERATION
csv_path = "csv-contiki/"
csv_path_final = "_final-datasets/"
from natsort import os_sorted
print("\n\n--- Creating Contiki final dataset ---")
files = os_sorted(os.listdir(csv_path))
os.system("awk '(NR == 1) || (FNR > 1)' " + csv_path + files[0] + "* > " +
↪  csv_path_final + "contiki_final_dataset.csv")
files.pop(0)
counter = 1
full = len(files)
for name in files:
    print("Filename: {}.csv ({}/{})".format(name,counter,full) )
    os.system("awk 'FNR > 1' " + csv_path + name + "* >> " + csv_path_final +
    ↪  "contiki_final_dataset.csv")
    counter+=1
```

É importante destacar que as simulações de um mesmo protocolo utilizam sempre a mesma faixa de endereços IP. A exemplificar, os endereços IP das simulações relacionados com o protocolo CoAP estão no intervalo de "fe80::c30c:0:0:1"a

"fe80::c30c:0:0:b". O mesmo ocorre com as simulações referentes ao protocolo MQTT, cujo intervalo de endereços IP está entre "::200:0:0:1"a "::200:0:0:b"e entre "fe80::200:0:0:1"e "fe80::200:0:0:b". O intervalo de endereços IP foi mantido para todos os ataques de um mesmo protocolo devido a não haver alterações a nível de ambiente e infraestrutura nas simulações. A Tabela 5 apresenta a lista de endereços IP maliciosos de cada um dos tipos de simulações.

Tabela 5: Endereços IP maliciosos para cada tipo de ataque nas simulações realizadas na ferramenta Contiki

SIMULAÇÃO	ENDEREÇOS IP MALICIOSOS
coap-black_hole-1malic	fe80::c30c:0:0:6
coap-black_hole-2malic	fe80::c30c:0:0:6, fe80::c30c:0:0:7
coap-hello_flood-1malic	fe80::c30c:0:0:6
coap-hello_flood-2malic	fe80::c30c:0:0:6, fe80::c30c:0:0:7
coap-version_number-1malic	fe80::c30c:0:0:6
coap-version_number-2malic	fe80::c30c:0:0:6, fe80::c30c:0:0:7
mqtt-black_hole-1malic	fe80::200:0:0:6, ::200:0:0:6
mqtt-black_hole-2malic	fe80::200:0:0:6, ::200:0:0:6, fe80::200:0:0:7, ::200:0:0:7
mqtt-hello_flood-1malic	fe80::200:0:0:6, ::200:0:0:6
mqtt-hello_flood-2malic	fe80::200:0:0:6, ::200:0:0:6, fe80::200:0:0:7, ::200:0:0:7
mqtt-version_number-1malic	fe80::200:0:0:6, ::200:0:0:6
mqtt-version_number-2malic	fe80::200:0:0:6, ::200:0:0:6, fe80::200:0:0:7, ::200:0:0:7

Como pode ser identificado, no que se refere às simulações relacionadas com o protocolo MQTT, os endereços IP são representados em duas faixas diferentes. Isso ocorre devido a haver tráfego com a rede externa a da simulação, especificamente com a rede do *host*. Desta forma, a ferramenta abrange as duas faixas de IP apresentadas, a saber "fe80::200:0:0:X"e "::200:0:0:x".

Inicialmente, antes do pré-processamento dos dados, os ficheiros CSV possuem uma quantidade total de 1693 atributos, que é correspondente a quantidade de filtros da ferramenta tshark usados na transformação dos ficheiros PCAP para CSV. Porém, a maior parte dos atributos são completamente vazios ou compostas por valores constantes (um único valor em todos os registos de determinado atributo) e esta quantidade de atributos foi consideravelmente reduzida com a utilização do *script* de pré-processamento. Neste processo, foram eliminados o total de 1457 atributos, restando apenas a quantidade de 236 atributos. Ainda, foi realizada uma análise, de forma a identificar os atributos possíveis de serem eliminados. Neste sentido, foi realizada a eliminação de 198 atributos, com o objetivo de eliminar atributos considerados não importantes para o *dataset* final.

Apos este processo de eliminação de atributos com base em uma análise, chegamos a versão final do *dataset*. O *dataset* final contém dados dos 3 ataques selecionados mais o tráfego normal e possui a quantidade de 16 319 307 registos e 38 atributos, destes, sendo 2 os atributos-alvo (um para classificação binaria e outro para análise multi-classe). A lista de atributos do *dataset* é apresentada na Figura 15 e o resumo do *dataset* é apresentado na Tabela 6:

```
Data columns (total 38 columns):
 #   Column                  Dtype
---  ------                  -----
 0   6lowpan.dst             object
 1   6lowpan.next            object
 2   6lowpan.src             object
 3   coap.code               float64
 4   coap.payload_length     float64
 5   coap.type               float64
 6   frame.len               int64
 7   frame.protocols         object
 8   icmpv6.code             float64
 9   icmpv6.rpl.dao.dodagid  object
 10  icmpv6.rpl.dio.dtsn     float64
 11  icmpv6.rpl.dio.rank     float64
 12  icmpv6.rpl.dio.version  float64
 13  ipv6.dst                object
 14  ipv6.nxt                float64
 15  ipv6.plen               float64
 16  ipv6.src                object
 17  mqtt.conflag.willflag   float64
 18  mqtt.conflags           object
 19  mqtt.len                float64
 20  mqtt.msg                object
 21  mqtt.qos                float64
 22  mqtt.topic              object
 23  tcp.dstport             float64
 24  tcp.flags               object
 25  tcp.len                 float64
 26  tcp.nxtseq              float64
 27  tcp.srcport             float64
 28  udp.dstport             float64
 29  udp.length              float64
 30  udp.srcport             float64
 31  wpan.dst64              object
 32  wpan.dst_addr_mode      object
 33  wpan.src64              object
 34  wpan.src_addr_mode      object
 35  wpan.version            float64
 36  IS_MALICIOUS            int64
 37  ATTACK_TYPE             int64
dtypes: float64(20), int64(3), object(15)
```

Figura 15: Lista de atributos e respetivos tipos de dados do *dataset smart_greenhouse*.

Tabela 6: Número de registos normais e maliciosos do *dataset smart_greenhouse*.

DATASET	MALICIOSOS	NORMAIS	TOTAL
smart_greenhouse	2 624 135	13 695 172	16 319 307

5.7.2 *Descrição do dataset smart_city*

Para gerar os ficheiros CSV produzidos a partir da ferramenta Netsim, foi executado o mesmo *script* de pré-processamento executado também nos ficheiros prodzidos pela ferramenta Contiki. A maior parte dos atributos são também vazios ou constantes, da mesma forma como ocorreu com os atributos dos ficheiros prevenientes das simulações realizadas com a ferramenta Contiki. Desta forma, estes atributos foram reduzidos a quantidade de 92 com o *script*, sendo originalmente o total de 1693 atributos. Foi ainda realizada a análise exploratória dos *datasets* e foram eliminados 60 atributos que foram considerados como irrelevantes para o *dataset*. Nesta fase, foram gerados 90 ficheiros CSV no total, que eram referentes aos 42 ficheiros CSV originários. A Tabela 7 apresenta a quantidade de ficheiros criados e a quantidade total de registos para cada tipo de tráfego.

Tabela 7: Número de ficheiros para cada tipo de simulação realizada na ferramenta Netsim.

SIMULAÇÃO	NÚMERO DE FICHEIROS	TOTAL DE PACOTES
coap-normal-3sensors	5	1 153 882
coap-normal-9sensors	5	1 391 174
coap-dos-3malic	12	3 061 910
coap-dio_supression-1malic	10	4 851 311
coap-sink_hole-2malic	10	3 823 500
Total	42	14 281 777

A Tabela 8 apresenta a lista de endereços IP maliciosos de cada um dos tipos de simulações que representavam os ataques selecionados:

Tabela 8: Endereços IP maliciosos para cada tipo de ataque nas simulações realizadas na ferramenta Netsim.

SIMULAÇÃO	ENDEREÇOS IP MALICIOSOS
coap-dos-3malic	fdec:3017:e256:9bb8:1fe7:9d62:8210:1225, fdec:3017:e256:9bb8:1fe7:3515:f5c6:468d, fdec:3017:e256:9bb8:1fe7:3587:c2fe:3d24
coap-dio_supression-1malic	fdec:3017:e256:9bb8:1fe7:e907:5783:96a3
coap-sink_hole-2malic	fdec:3017:e256:9bb8:1fe7:9d62:8210:1225, fdec:3017:e256:9bb8:1fe7:3587:c2fe:3d24

Da mesma forma como ocorreu com o *dataset* gerado com a ferramenta Contiki, não foi possível utilizar a função do *script* de pré-processamento para fazer a integração dos ficheiros CSV em um único *dataset*, também devido a limitação de memória do dispositivo utilizado. A Listagem 3 apresenta o código utilizado para gerar o *dataset smart_city* preveniente das simulações executadas no Netsim.

Listagem 3: Código fonte Python utilizado para gerar o *dataset smart_city*.

```
# SMART CITY FINAL DATASET GENERATION
csv_path = "csv-netsim-final/"
csv_path_final = "_final-datasets/"
from natsort import os_sorted
print("\n\n--- Creating Netsim final dataset ---")
files = os_sorted(os.listdir(csv_path))
os.system("awk '(NR == 1) || (FNR > 1)' " + csv_path + files[0] + "* > " +
↪  csv_path_final + "netsim_final_dataset.csv")
files.pop(0)
counter = 1
full = len(files)
for name in files:
    print("Filename: {}.csv ({}/{})".format(name,counter,full) )
    os.system("awk 'FNR > 1' " + csv_path + name + "* >> " + csv_path_final +
    ↪  "netsim_final_dataset.csv")
    counter+=1
```

Este comando adiciona sequencialmente todos os ficheiros CSV em um único ficheiro CSV, que neste exemplo foi dado o nome *smart_city*.csv. O *dataset* final possui a quantidade de 14 281 777 registos e 32 atributos, sendo 2 os atributos-alvo (um para classificação binaria e outro para análise multi-classe). Ele é composto somente por tráfego que representa um ambiente que utiliza o protocolo CoAP, que é representado por tráfego UDP, uma vez que o Netsim não implementa o protocolo da camada de aplicação. A lista de atributos do *dataset* é apresentada na Figura 16 e o resumo do *dataset* é apresentado na Tabela 9:

Tabela 9: Número de registos normais e maliciosos do *dataset smart_city*

DATASET	MALICIOSOS	NORMAIS	TOTAL
smart_city	4 656 207	9 625 570	14 281 777

```
Data columns (total 32 columns):
 #   Column                          Dtype
---  ------                          -----
 0   frame.len                       int64
 1   frame.protocols                 object
 2   icmpv6.code                     float64
 3   icmpv6.rpl.dio.rank             float64
 4   icmpv6.rpl.dio.dagid            object
 5   icmpv6.rpl.opt.type             float64
 6   icmpv6.rpl.dao.sequence         float64
 7   icmpv6.rpl.opt.length           float64
 8   icmpv6.rpl.opt.target.prefix    object
 9   icmpv6.type                     float64
 10  ip.dst                          object
 11  ip.flags                        object
 12  ip.len                          float64
 13  ip.proto                        float64
 14  ip.src                          object
 15  ip.version                      int64
 16  ipv6.dst                        object
 17  ipv6.nxt                        float64
 18  ipv6.plen                       float64
 19  ipv6.src                        object
 20  ipv6.version                    float64
 21  tcp.dstport                     float64
 22  tcp.flags                       object
 23  tcp.len                         float64
 24  tcp.nxtseq                      float64
 25  tcp.port                        float64
 26  tcp.srcport                     float64
 27  udp.dstport                     float64
 28  udp.length                      float64
 29  udp.srcport                     float64
 30  IS_MALICIOUS                    int64
 31  ATTACK_TYPE                     int64
dtypes: float64(19), int64(4), object(9)
```

Figura 16: Lista de atributos e respetivos tipos de dados do *dataset smart_city*.

6

APLICAÇÃO DE ALGORITMOS DE MACHINE LEARNING

O conceito *machine learning* ganhou muita notoriedade ao longo das últimas décadas, principalmente pelo avanço tecnológico que propiciou a aplicação de técnicas mais complexas e que dependem de maior poder computacional. Neste capítulo, serão descritos e desenvolvidos os algoritmos de aprendizado de máquina selecionados para compor os modelos de classificação. O primeiro ponto a ser considerado neste aspeto é identificar qual será forma na qual os algoritmos serão desenvolvidos e, além da possibilidade de desenvolver o algoritmo "*from scratch*", existem algumas bibliotecas que podem ser utilizadas para este fim. Dentre as opções, podem ser citados Pytorch[1], Tensorflow[2] e scikit-learn[3]. Para desenvolvimento dos algoritmos deste trabalho, optou-se pela biblioteca scikit-learn, devido a esta exigir uma menor curva de aprendizagem em relação às outras opções, conforme avaliado pelo autor.

A partir desta biblioteca, foram criado dois *scripts* para realizar as diferentes etapas a serem realizadas. Os dois *scripts* mencionados são:

- create_datasets: realiza a criação dos *datasets* com base dos *datasets* originários;
- ml_application: realiza a limpeza e tratamento dos dados, aplica os modelos de ML selecionados e apresenta as métricas referentes a estes modelos.

A utilização do primeiro *script* decorre do fato de ter sido necessário selecionar apenas uma parte dos dados dos *datasets smart_greenhouse*.csv e *smart_city*.csv, uma vez que o computador utilizado não possuía memória o suficiente para lidar com a totalidade de dados destes *datasets*. A partir disso, optou-se por criar dois *datasets* para cada um dos *datasets* originais, onde cada um destes será utilizado em um tipo de classificação. Como mencionado, o *dataset smart_greenhouse* foi dividido em dois outros *datasets* menores, um para ser utilizado na classificação binária e outro na classificação multi-classe. Da mesma forma, o *dataset smart_city* foi também dividido em dois *datasets* menores, que também serão utilizados nas

1 https://pytorch.org/
2 https://www.tensorflow.org/
3 https://scikit-learn.org/

classificações acima citadas. Portanto, foram criados no total quatro novos *datasets* menores que possuem parte dos registos dos *datasets* originais.

Aos novos *datasets*, foi dado o nome respetivo ao *dataset* originário e o tipo de classificação ao qual ele será aplicado, conforme apresentado na lista a seguir:

- *smart_greenhouse-binary*;
- *smart_greenhouse-multiclass*;
- *smart_city-binary*;
- *smart_city-multiclass*.

Para gerar estes quatro *datasets*, os *datasets* originais foram divididos em *chunks* (divisão dos dados em porções menores) e, a cada iteração, eram selecionados destes *chunks* os registos referentes a cada uma das classes correspondentes aos atributos-alvo. A quantidade total de registos destes *datasets* é apresentada a seguir.

A quantidade de registos que compõem os *datasets smart_greenhouse-binary* e *smart_greenhouse-multiclass* é apresentadas na Tabela 10:

Tabela 10: Números de registos dos dois novos *datasets* criados a partir do *dataset smart_greenhouse* com base no tipo de classificação.

TIPO DE TRÁFEGO	NORMAL	MALICIOSO	TOTAL
smart_greenhouse-binary	450 000	450 000	900 000
smart_greenhouse-multiclass	150 000	450 000	600 000

Para o *dataset* especifico para a classificação multi-classe, os 450 000 registos correspondentes são referentes a cada uma das 3 classes de tráfego malicioso. Desta forma, são 150 000 registos referentes a cada um dos 3 ataques (*blackhole*, *hello flood* e *version number*), onde 75 000 registos de cada ataque são respetivos a cada um dos dois protocolos da camada de aplicação utilizados (CoAP e MQTT).

A Tabela 11 apresenta a quantidade de registos de cada tipo de tráfego para os ficheiros criados com base no *dataset smart_greenhouse* com base no tipo de classificação.

Referente a cada tipo de classificação, a quantidade de registos dos *datasets* criados com base no *dataset smart_city* são apresentados na Tabela 12.

Na Tabela 13, são apresentados os números de ataques correspondentes aos *datasets smart_city-multiclass* e *smart_city-multiclass*.

Como pode ser visto, foi selecionada uma amostra com uma quantidade de registos que mantivesse o balanceamento do *dataset*, mantendo uma equivalência entre os

Tabela 11: Números de registos por tipo de tráfego nos *datasets* baseados no *smart_greenhouse* com base no tipo de classificação.

TIPO DE TRÁFEGO	BINÁRIA	MULTI-CLASSE
coap-normal	225 000	75 000
mqtt-normal	225 000	75 000
coap-black_hole	75 000	75 000
mqtt-black_hole	75 000	75 000
coap-hello_flood	75 000	75 000
mqtt-hello_flood	75 000	75 000
coap-version_number	75 000	75 000
mqtt-version_number	75 000	75 000
Total	900 000	600 000

Tabela 12: Números de registos do dois novos *datasets* criados a partir do *dataset* *smart_greenhouse* com base no tipo de classificação.

TIPO DE TRÁFEGO	NORMAL	MALICIOSO	TOTAL
smart_city-binary	317 700	317 700	635 400
smart_city-multiclass	150 000	317 700	467 700

Tabela 13: Números de registos por tipo de tráfego nos *datasets* baseados no *smart_city* com base no tipo de classificação.

TIPO DE TRÁFEGO	BINÁRIA	MULTI-CLASSE
Normal	317 700	150 000
DIO Supression	150 000	150 000
Sinkhole	17 700	17 700
DoS	150 000	150 000
Total	635 400	467 700

registos de cada uma das classes. A única excepção para esta quantidade está relacionada ao ataque *Sinkhole*, cuja quantidade total de registos foi de apenas 17 700.

Para desenvolver os modelos de aprendizagem, apresentado no ficheiro ml_application, foram seguidas as etapas abaixo apresentadas.

- Eliminar os dados selecionados;
- Separar os dados em atributos de predição e atributos-alvo;
- Separar os dados de treino e teste;
- Realizar o tratamentos dos dados;
- Reduzir o domínio dos dados;

- Identificar os melhores hiperparâmetros;
- Aplicar os algoritmos de classificação;
- Avaliar os modelos criados.

Estas diferentes etapas foram executadas sequencialmente no *script* e elas serão apresentadas nas secções a seguir.

6.1 SELEÇÃO DOS ALGORITMOS

A lista de algoritmos é considerável e podem ser classificados de acordo com a técnica utilizada para identificar o padrão nos dados, como a utilização de árvores ou a distância entre os valores. Além disso, os algoritmos ainda podem ser classificados em supervisionados e não supervisionados, como visto na secção 2.2.

Após uma análise dos algoritmos comumente utilizados na literatura, foram selecionados quatro algoritmos pertencentes ao grupo dos supervisionados, conforme apresentado na lista a seguir:

- *Decision Tree Classifier* (DTC)[4];
- K-nearest Neighbors (KNN)[5];
- Logistic Regression Classifier (LRC)[6];
- Random Forest Classifier (RFC)[7].

Tais algoritmos foram selecionados devido a poderem ser utilizados tanto na classificação binária quanto na classificação multi-classe, que serão explicadas na secção 6.2. Optou-se por não utilizar nenhum algoritmo do grupo dos não supervisionados, pois os dados foram rotulados desde o momento da da criação dos ficheiros CSV.

6.2 TÉCNICAS DE PREDIÇÃO

Nesta etapa, o objetivo é definir as técnicas que serão utilizadas para classificar os dados dos *datasets*. Esta classificação tem como base analisar as características de cada um dos registos do *dataset* e determinar o tipo de classe (atributo-alvo) ao qual um registo corresponde.

4 https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
5 https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
6 http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
7 http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Neste sentido, são utilizadas as seguintes técnicas de predição para classificação dos dados:

- Classificação binária;
- Classificação multi-classe.

A primeira das etapas de classificação é relativa a classificação binária, onde existem apenas duas possíveis classes de dados. Neste passo, os algoritmos selecionados serão utilizados para identificar se o tráfego corresponde a um tráfego anômalo ou a um tráfego normal.

Na classificação multi-classe, busca-se identificar a qual das diferentes classes o registo pertence. Diferentemente ao que acontece na classificação binária, na classificação multi-classe o número de classes deve ser de 3 ou mais. No contexto deste trabalho, será avaliado se o registo pertence a cada uma das 4 classes de cada um dos *datasets*. Para isso, serão utilizados os mesmos algoritmos utilizados também na classificação binária.

Cada uma destas duas técnicas será aplicada aos respetivos *datasets*. Assim, a classificação binária será aplicada aos *datasets smart_greenhouse-binary* e *smart_city-binary* e a classificação multi-classe será aplicada aos *datasets smart_greenhouse-multiclass* e *smart_city-multiclass*, porém realizando a alteração dos parâmetros quando necessário.

6.3 ELIMINAÇÃO DE DADOS

Nesta fase, foram eliminados manualmente os atributos de predição que identificavam um registo de forma única e os que foram identificados como não importantes para a análise. Optou-se por esta remoção para tornar os dados mais genéricos. Ainda, era esperado que não fosse possível determinar a classe de um registo através de um único atributo apenas, como o endereço IP.

Foram identificados os atributos que identificavam unicamente os registos e estes foram eliminados das etapas posteriores. Com base nestas informações, foram utilizados apenas os seguintes atributos os dois *datasets*:

- *smart_greenhouse-binary* e *smart_greenhouse-multiclass*: frame.len, 6lowpan.next, coap.code, coap.type, icmpv6.code, icmpv6.rpl.dio.dtsn, icmpv6.rpl.dio.rank, icmpv6.rpl.dio.version, ipv6.nxt, mqtt.conflags, mqtt.msg, mqtt.qos, mqtt.topic, tcp.flags, tcp.len, udp.length, *IS_MALICIOUS*, *ATTACK_TYPE*;

- *smart_city-binary* e *smart_city-multiclass*: frame.len, icmpv6.code, icmpv6.rpl.dio.rank, icmpv6.rpl.opt.type, icmpv6.rpl.dao.sequence, icmpv6.rpl.opt.length, icmpv6.type, ip.flags, ip.len, ip.proto, ipv6.nxt, ipv6.plen, ipv6.version, tcp.flags, tcp.len, udp.length, *IS_MALICIOUS*, *ATTACK_TYPE*.

6.4 SEPARAÇÃO DOS DADOS

Nesta etapa, os dados foram divididos de acordo com a sua função no *dataset*. Na primeira etapa, os dados foram divididos em atributos de predição e atributos-alvo. Estes foram divididos em duas diferentes variáveis, uma vez que o atributo-alvo "*IS_MALICIOUS*"é utilizado na classificação binária e o atributo-alvo "*ATTACK_TYPE*"é utilizado na classificação multi-classe.

Para o atual trabalho, foram utilizados os seguintes atributos como *targets* do *dataset*:

- *IS_MALICIOUS*: usado na classificação binária;
- *ATTACK_TYPE*: usado na classificação multi-classes.

Na segunda etapa, foi realizada a separação da amostra em duas diferentes partes, nomeadamente treino e teste, sendo definida a quantidade de 75% dos dados para treinamento e 25% para teste. A distribuição apresentada na Tabela 14 percentagem de dados:

Tabela 14: Distribuição dos dados de treino e teste em cada *dataset*.

DATASET	TREINO	TESTE
smart_greenhouse-binary	675 000	150 000
smart_greenhouse-multiclass	450 000	150 000
smart_city-binary	476 550	158 850
smart_city-multiclass	350 775	116 925

6.5 TRATAMENTO DOS DADOS

Nesta etapa, realiza-se o tratamento dos dados que compõem o *dataset*. Com base nisso, busca-se evitar problema relacionados com a qualidade dos dados, tais como dados ruidosos, inconsistentes, redundantes, incompletos e vazios.

Inicialmente, foi identificado que alguns valores eram infinitos, representados pelo valor np.inf, e isso poderia causar problemas nas etapas futuras. Desta forma,

tais valores foram substituídos pelo valor np.nan, que será tratado em etapas de tratamento dos dados.

Seguindo este padrão, realizaram-se processos como normalização e codificação dos dados para que estes tivessem uma melhor qualidade e pudessem ser corretamente processados pelos modelos criados.

6.6 NORMALIZAÇÃO DOS DADOS

Outra característica relevante a ser considerada é a de que alguns algoritmos de aprendizagem são sensíveis quanto a diferença na escala dos valores que compõem o *dataset*. Desta forma, para estes tipos de algoritmos, a normalização dos dados é um requisito, já que eles podem apresentar dificuldade em convergir para o resultado esperado se os dados não forem normalizados ou padronizados. Além disso, algoritmos que utilizam cálculo de distância, como, por exemplos, os algoritmos KNN, K-means e SVM (*Support vector machines*), e os algoritmos de redes neuronais tendem a produzir melhores resultados quando os valores estão normalizados. Diferentemente a estes, os algoritmos baseados em árvores não são afetados com a falta de normalização dos dados, pois a diferente nas escalas não afeta a forma como as árvores serão construídas.

Em relação ao que foi apresentado e considerando os 4 algoritmos selecionados, a Tabela 15 apresentada as características relacionadas quanto ao algoritmo ser sensível a variação na escala dos dados e a consequente necessidade ou não de normalização dos dados.

Tabela 15: Sensibilidade dos algoritmos quanto a variância na escala dos dados.

ALGORITMOS	SENSÍVEL A ESCALA
Decision Tree	Não
Random Forest	Não
K-nearest Neighbors	Sim
Logistic Regression	Sim

Como pode ser visto, os algoritmos *Decision Tree* e *Random Forest* não são sensíveis a variância dos dados. Em contrapartida, os algoritmos *K-nearest Neighbors* e *Logistic Regression* o são. Desta forma, optou-se por por realizar a normalização dos dados para uma escala comum. Para este fim, as técnicas mais comumente utilizadas são a padronização e a normalização de dados, cujo processo de normalização

pode ser facilmente realizado com a biblioteca scikit-learn através das funções StandardScaler[8] e MinMaxScaler[9].

A primeira das duas funções, nomeadamente *StandarScaler*, atribui uma distribuição padronizada aos valores, onde a média com o valor 0 (zero) e o desvio padrão em 1. Como não há limites no que diz respeito aos valores dos atributos, podem ser encontrados valores negativos nos registos após a aplicação da função. Esta padronização é também conhecida como normalização *z-score*, cuja equação é apresentada em 1.

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

A segunda função, nomeadamente *MinMaxScaler*, redimensiona os dos valores dos registos para o intervalo entre 0 e 1. A equação geral da normalização min-max é apresentada em 2.

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{2}$$

Para o atual trabalho, foi selecionada a técnica de normalização, sendo então utilizada a função *MinMaxScaler* do scikit-learn.

6.7 CODIFICAÇÃO DE VARIÁVEIS CATEGÓRICAS

Os atributos podem ser divididos em categórico e qualitativo a depender do tipo de dados que compõe o atributo. Estas duas categorias podem ainda ser divididas em duas outras subcategorias cada.

Caso um atributo pertença ao grupo de valores categóricos e caso este atributo seja relevante para o *dataset*, ele deverá passar por uma transformação na qual ele passe da categoria categórico para numérico. Esse necessidade ocorre devido ao fato de os algoritmos de aprendizagem, em sua grande maioria, requererem que os valores dos registos sejam numéricos para serem corretamente processados. Esta conversão de valores pode ser facilmente realizada com a biblioteca scikit-learn através das funções *OrdinalEncoder*[10] e *OneHotEncoder*[11].

A função *OrdinalEncoder* atribui um valor numérico inteiro exclusivo com base nos textos em ordem alfabética. Para esta função, os valores serão representados

8 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
9 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
10 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html
11 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

em uma mesma coluna, mas, mesmo que não haja uma ordem ou classificação entre os valores, o algoritmo poderá levar em consideração os valores e atribuir uma relação de ordem ou classificação entre eles. Um ponto de destaque é o de que, caso haja mais de dois domínios no atributo e não haja uma ordem entre estes valores, recomenda-se que seja utilizada a função OneHotEncoder.

A função *OneHotEncoder* criar uma nova coluna para cada valor único da lista de valores categóricos. Dessa forma, será criado o número de colunas igual ao número de valores únicos da coluna em questão. Este tipo de comportamento gera um problema chamado *Dummy variabe Trap* (Armadilha das variáveis fictícias), que consiste em um valor de uma variável ser facilmente previsto pelas outras variáveis criadas, já que estas são altamente correlacionadas entre si.

Neste processo de codificação dos valores dos atributos categóricos, foram seguidos os seguintes critérios:

- *OrdinalEncoder*: aplicado aos atributos com mais de 5 valores únicos;
- *OneHotEncoder*: aplicado aos atributos que possuem até 5 valores únicos.

6.8 REDUÇÃO DO DOMÍNIO DE DADOS

Uma das ações mais importantes dentro do *machine learning* é o processo de seleção de atributos (também chamado de redução do domínio dos dados ou *feature engineering* em língua inglesa). Esta etapa tem objetivo de identificar e manter apenas os atributos mais relevantes e remover os atributos menos importantes.

O processo de seleção de atributos está relacionado com grande parte do tempo despendido no aprendizado de máquina. Os atributos têm relação direta com a capacidade de um modelo determinar ou predizer a classe de um determinado registo e a capacidade que este tem de identificar os padrões e correlação nos dados analisados.

A seleção de atributos foi realizada através de métodos automatizados com a biblioteca scikit-learn. Foram então utilizadas a função SelectKBest[12] com duas diferentes funções de pontuação, a saber f_classif e chi2.

Em um segundo momento, foi também usado o algoritmo de seleção recursiva de atributos[13] (RFE ou *Recursive Feature Elimination* em língua inglesa), de forma a trazer os atributos selecionados com base em modelos de aprendizado de máquina.

12 https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
13 https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html

Juntamente ao RFE, foram utilizados os classificadores *DecisionTreeClassifier*, *LogisticRegressionClassifier* e *RandomForestClassifier*.

Cada um destes 5 seletores de atributos identificou os 10 melhores atributos para cada um dos 4 *datasets*. Portanto, estes 5 conjuntos de atributos selecionados serão utilizados na aplicação dos algoritmos de aprendizagem em cada um dos *datasets*.

Na Tabela 16 são apresentados o número de identificação de cada um dos 5 conjuntos de atributos e o respetivo algoritmo utilizado no processo de seleção de atributos.

Tabela 16: Número identificador do conjunto de atributos e o algoritmo utilizado na seleção destes atributos.

NÚMERO DO CONJUNTO	ALGORITMO
Conjunto 1	SelectKBest e chi2
Conjunto 2	SelectKBest e f_classif
Conjunto 3	RFE e DTC
Conjunto 4	RFE e RFC
Conjunto 5	RFE e LRC

6.9 IDENTIFICAÇÃO DOS MELHORES HIPERPARÂMETROS

Nesta etapa, foi realiza a identificação dos hiperparâmetros que geravam os melhores resultados para cada um dos 4 algoritmos selecionados. Para este processo foi então utilizada a função *GridSearchCV*[14]. Esta função é uma técnica de validação cruzada que busca identificar a melhor combinação de parâmetros de uma função dentro de um conjunto de possíveis combinações. A lista de parâmetros utilizados com a função *GridSearchCV* é apresentado no repositório do Github.

Ao final desta etapa, foi possível identificar o conjunto de parâmetros que encontravam os melhores resultados, levando em consideração a métrica *accuracy*, para cada um dos 4 *datasets*.

Os algoritmos selecionados e seus respetivos parâmetros são apresentados na secção 6.10.

14 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

6.10 CARACTERIZAÇÃO DOS ALGORITMOS DE CLASSIFICAÇÃO

No que se refere a classificação binária quando aplicada ao *dataset smart_greenhouse-binary*, foram utilizados os seguintes hiperparâmetros:

- DecisionTreeClassifier(ccp_alpha=0.01, critérion='entropy', max_depth=5, max_features='auto');
- RandomForestClassifier(ccp_alpha=0.01,critérion='gini',max_depth=2, max_features='auto',n_estimators=250);
- LogisticRegression(C=0.001,max_iter=50,penalty='l2',solver='newton-cg');
- KNeighborsClassifier(n_neighbors=10,weights='uniform').

Também em relação a classificação binária, foram utilizados os seguintes hiperparâmetros quando aplicados ao *dataset smart_city-binary*:

- DecisionTreeClassifier(ccp_alpha=0.01, critérion='entropy', max_depth=5, max_features='auto');
- RandomForestClassifier(critérion='gini', max_depth=2,max_features='auto', n_estimators=250);
- LogisticRegression(C= 0.001,penalty= 'l2',max_iter=50,solver= 'newton-cg');
- KNeighborsClassifier(n_neighbors=13,weights= 'uniform').

Já, no que se refere a classificação multi-classe, foram utilizados os seguintes hiperparâmetros quando aplicados ao *dataset smart_greenhouse-multiclass*.

- DecisionTreeClassifier(ccp_alpha=0.001, max_depth=8, max_features='auto');
- RandomForestClassifier(ccp_alpha=0.001,critérion='entropy',max_depth=10, max_features='log2',n_estimators=500);
- LogisticRegression(max_iter=250,multi_class='ovr', solver='liblinear');
- KNeighborsClassifier(n_neighbors=13,weights='distance').

Por fim, foram utilizados os seguintes hiperparâmetros quando os algoritmos forama aplicados ao *dataset smart_city-multiclass*

- DecisionTreeClassifier(ccp_alpha=0.001, critérion='entropy',max_depth=5, max_features='auto');

- RandomForestClassifier(ccp_alpha=0.001,critérion='entropy', max_depth=2,max_features='auto',n_estimators=250);
- LogisticRegression(C=0.01,max_iter=50,multi_class='ovr',penalty='l2', solver='newton-cg');
- KNeighborsClassifier(n_neighbors=15,weights='uniform').

Nota-se que a maior parte dos hiperparâmetros não foram coincidentes em todas as ocorrências. Através destes algoritmos, foi executada uma única ronda de testes, sendo considerados os resultados obtidos nesta ronda. Ao final de cada iteração, foram apresentados as métricas de avaliação de cada um dos modelos quando aplicadas ao respetivo *dataset* com cada um dos 5 conjuntos de atributos selecionados anteriormente de forma automatizada.

6.11 MÉTRICAS DE AVALIAÇÃO

Um ponto muito relevante no aprendizado de máquina é o desempenho que o modelo atingiu ao classificar os dados com base nos valores apresentados como entrada. Ele pode ser mensurado através de alguns métricas de avaliação muito conhecidos e utilizados na literatura. Em outras palavras, elas servem para quantificar a qualidade, ou desempenho, dos resultados obtidos com estes modelos. A bibliografia traz diferentes métricas, que buscam identificar os resultados de cada uma das possibilidades.

No atual trabalho, os modelos serão avaliados de acordo com as seguintes métricas:

- *Confusion Matrix*;
- *Accuracy*;
- *Recall*;
- *Precision*;
- *F1-Score*.

Estas métricas são baseadas na proporção de registos previstos como verdadeiros ou falsos pelos modelos. Desta forma, os registos podem ser classificados de acordo com as quatro possibilidade apresentadas na lista a seguir:

- Verdadeiro positivo (*true positive* — TP): quando o modelo prevê corretamente a classe procurada;

- Falso positivo (*false positive* — FP): quando o modelo identifica de forma errada a classe procurada. É também chamado de erro tipo I;
- Falso verdadeiro (*true negative* — TN): quando a classe que não está a ser buscada foi prevista corretamente;
- Falso negativo (*false negative* — FN): quando a classe que não esta a ser buscada foi prevista incorretamente. É também chamado de erro tipo II.

Com base nestes valores, pode então ser criada a matriz de confusão, que é uma forma visual de representar o desempenho de um modelo. A matriz de confusão apresenta a quantidade de registos correta e incorretamente classificados, separando-os em FP, FN, TP e TN na tabela.

A métrica chamada *accuracy* apresenta a quantidade de registos que foram corretamente classificados, independentemente da classe. A equação desta métrica é apresentada em 3.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

Precision é a métrica de avaliação que tem o objetivo de mensurar a proporção de previsões que estavam corretas, isto é, dentre todo o conjunto de registos classificados como positivos, quantos são realmente positivos. Na equação 4 é apresentada a definição da métrica *precision* .

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

A métrica *recall* tem o objetivo de mensurar a capacidade do modelo de detetar com sucesso a quantidade real de registos de uma classe específica. A equação 5 apresenta a definição da métrica *recall*.

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

Por fim, a métrica *F1 Score* leva em consideração a media ponderada entre as métricas *recall* e *precision*. A equação da função *F1 Score* é apresentada em 6.

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{6}$$

Cada uma destas métricas busca identificar diferentes características dos resultados obtidos. Alguns destes pontos, se forem levados em consideração isoladamente, podem transmitir a falsa sensação de que o modelo criado apresenta um bom desempenho. Como exemplo, podemos citar o caso em que a *acurracy* esteja com

índices elevados, próximo aos 100%, mas, quando levado em consideração a avaliação da métrica *recall* ou *precision*, os valores de uma das classes é baixo, demonstrando a ineficiência do modelo para determinar o padrão para esta classe em específico. Mais uma vez, estas métricas de avaliação podem ser facilmente aplicadas com a biblioteca scikit-learn através da função *classification_report*[15]. Um exemplo de um *output* dessa função é apresentado na Figura 17.

Classification Report :

	precision	recall	f1-score	support
0	0.93	0.64	0.76	37616
1	0.69	0.94	0.79	37447
2	1.00	0.87	0.93	37255
3	0.81	0.87	0.84	37682
accuracy			0.83	150000
macro avg	0.86	0.83	0.83	150000
weighted avg	0.86	0.83	0.83	150000

Figura 17: Exemplo do *output* da função *classification_report*.

Para o critério de seleção da métrica a ser utilizada na avaliação dos nossos modelos de classificação, o fator mais importante é classificar o tráfego malicioso corretamente. Portanto, é preferível que tenhamos uma pequena, preferencialmente nenhuma, quantidade de registos maliciosos que sejam erroneamente considerados como tráfego normal se comparado a uma incorreta classificação de casos correspondentes a tráfego normal que foram erroneamente classificados como tráfego malicioso.

Com base nisso, a métrica selecionada para avaliar os modelos utilizados foi o *recall*, mais especificamente a métrica *recall weighted avg.*, que pode ser aplicada devido aos *datasets* serem maioritariamente balanceados. Essa opção se deu devido a querer minimizar a quantidade de valores falso negativos, uma vez que isso traria riscos para o utilizador.

Após a aplicação dos modelos sobre os dados dos *datasets* e a avaliação destes modelos sob a óptica das métricas acima descritas, foi possível avaliar o resultados do desempenho dos modelos criados. A apresentação dos resultados obtidos será dividido de acordo com a técnica de predição utilizada em cada etapa. Desta forma, os resultados serão apresentados nas subsecções 6.12 e 6.13, que são baseadas nas técnicas de predição binária e multi-classe respetivamente.

15 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

6.12 CLASSIFICAÇÃO BINÁRIA

A primeira etapa de classificação dos dados foi a realização da classificação binária, que consiste em identificar a classe do registo com base em apenas duas possíveis classes, tráfego normal, correspondente ao valor 0 no atributo-alvo, e tráfego anômalo, correspondentes ao valor 1. Aos *dataset* foram aplicados os 4 algoritmos em combinação com os 5 conjuntos de atributos criados, 2 desenvolvidos a partir do *SelectKBest* e 3 pelo RFE com diferentes algoritmos (DTC,RFC e LRC).

As matrizes de confusão para um mesmo conjunto de atributos são apresentadas em uma mesma figura. Desta forma, uma figura contempla 4 matrizes de confusão que é referente a um mesmo conjunto de atributos com cada um dos 4 algoritmos selecionados. Esta representação contribui para a identificação do desempenho dos algoritmos a usar um mesmo conjunto de atributos. Portanto, no total, foram criadas 20 matrizes de confusão, 5 para cada *dataset*. Destaca-se que foram apresentas somente as matrizes de confusão mais relevantes. O relatório contendo as matrizes de confusão e as demais métricas de avaliação pode ser encontrado no repositório do Github.

Os resultados obtidos na aplicação da classificação binária nos *datasets smart_city-binary* e *smart_greenhouse-binary* são apresentados respetivamente nas subsecções 6.12.1 e 6.12.2.

6.12.1 *smart_city-binary*

O *dataset smart_city-binary* contem parte dos dados do *dataset smart_city* que foram utilizados na classificação binária. A Tabela 17 apresenta a quantidades de registo correspondente a cada uma das classes pertencentes ao *dataset smart_city-binary* no que se refere à classificação binária.

Os 5 conjuntos de atributos selecionados de maneira automatizada para o *dataset smart_city-binary* são apresentados na Tabela 18:

Tabela 17: Número de registos de cada classe no conjunto de teste do *dataset smart_city-binary* quando usada a classificação binária.

CLASSE	TIPO DE TRÁFEGO	NÚMERO DE REGISTOS
0	Normal	79 324
1	Malicioso	79 526

A Tabela 19 apresenta os valores referentes às métricas *accuracy*, *precision*, *recall* e *F1 Score* dos algoritmos ao serem aplicados ao *dataset smart_city-binary*.

Tabela 18: Conjunto de atributos selecionados pelos algoritmos de seleção de atributos a serem usados no *dataset smart_city-multiclass*.

NÚMERO DO CONJUNTO	ATRIBUTOS
Conjunto 1	['frame.len', 'icmpv6.code', 'icmpv6.rpl.dio.dtsn', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.dio.version', 'ipv6.nxt', '6lowpan.next_0x00', '6lowpan.next_0x3a', '6lowpan.next_missing', 'tcp.flags_0x0018']
Conjunto 2	['frame.len', 'coap.code', 'icmpv6.code', 'icmpv6.rpl.dio.dtsn', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.dio.version', 'ipv6.nxt', 'tcp.len', '6lowpan.next_0x3a', '6lowpan.next_missing']
Conjunto 3	['frame.len', 'coap.code', 'icmpv6.code', 'icmpv6.rpl.dio.dtsn', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.dio.version', 'udp.length', 'mqtt.msg', '6lowpan.next_0x3a', '6lowpan.next_missing']
Conjunto 4	['icmpv6.rpl.dio.version', 'udp.length', '6lowpan.next_missing', 'mqtt.topic_sensors/evt/status/fmt/json', 'tcp.flags_0x0002', 'tcp.flags_0x0010', 'tcp.flags_0x0011', 'tcp.flags_0x0014', 'tcp.flags_0x0018', 'tcp.flags_missing']
Conjunto 5	['coap.code', 'coap.type', 'icmpv6.rpl.dio.dtsn', 'icmpv6.rpl.dio.rank', 'ipv6.nxt', 'tcp.len', 'udp.length', '6lowpan.next_0x00', '6lowpan.next_missing', 'tcp.flags_missing']

Tabela 19: Resultados das métricas em relação ao *dataset smart_city-binary*

CONJUNTO	ALGORITMOS	ACCURACY	PRECISION	RECALL	F1 SCORE
1	DTC	0.97	0.97	0.97	0.97
	RFC	0.97	0.97	0.97	0.97
	LRC	0.97	0.97	0.97	0.97
	KNN	0.97	0.97	0.97	0.97
2	DTC	0.97	0.97	0.97	0.97
	RFC	0.97	0.97	0.97	0.97
	LRC	0.97	0.97	0.97	0.97
	KNN	0.94	0.95	0.94	0.94
3	DTC	0.97	0.97	0.97	0.97
	RFC	0.97	0.97	0.97	0.97
	LRC	0.97	0.97	0.97	0.97
	KNN	0.94	0.95	0.94	0.94
4	DTC	0.97	0.97	0.97	0.97
	RFC	0.97	0.97	0.97	0.97
	LRC	0.97	0.97	0.97	0.97
	KNN	0.97	0.97	0.97	0.97
5	DTC	0.97	0.97	0.97	0.97
	RFC	0.97	0.97	0.97	0.97
	LRC	0.97	0.97	0.97	0.97
	KNN	0.97	0.97	0.97	0.97

Os valores de *recall* para cada classe se manteve em 94% para tráfego normal e 100% para malicioso, média de 97% de *recall*, *precision* e *F1 Score* com os 5 conjuntos de dados quando utilizados em combinação com os algoritmos DTC, RFC e LRC. A Figura 18 apresenta a matriz de confusão referentes ao *dataset smart_city-binary* quando usado com o conjunto de atributos número 1.

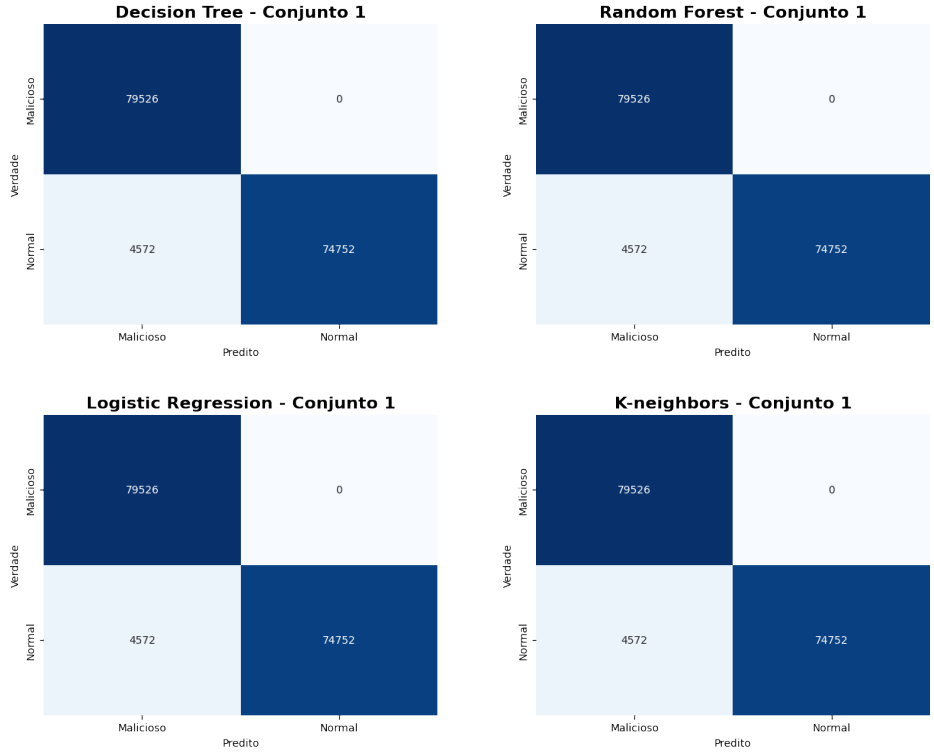


Figura 18: Matriz de confusão dos algoritmos aplicados ao *dataset smart_city-binary* com o conjunto de atributos número 1.

O algoritmo KNN apresentou os mesmos valores que os demais quando utilizado em combinação com os conjuntos 1, 4 e 5, mas apresentou valores mais baixos de *recall* quando utilizado com os conjuntos 2 e 3, onde aumentou o índice de identificação do tráfego normal, mas reduziu em 11% o índice de identificação do tráfego malicioso (99% normal e 89% para malicioso). Essa informação pode ser vista na Figura 19, através da matriz de confusão referente a este algoritmo quando ele utilizou o conjunto de atributos número 2.

A Tabela 20 apresenta os melhores resultados obtido em cada um dos conjuntos de dados ao aplicar as métricas *accuracy* e as médias ponderadas das métricas *recall*, *precision* e *F1-score*.

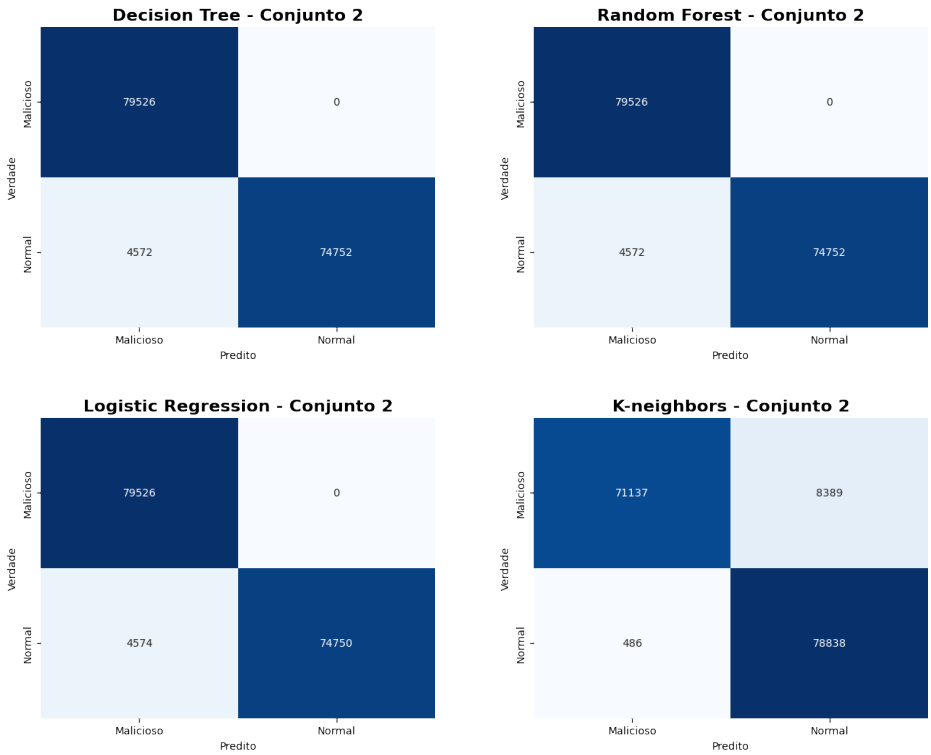


Figura 19: Matriz de confusão dos algoritmos aplicados ao *dataset smart_city-binary* com o conjunto de atributos número 2.

Tabela 20: Resultados do desempenho dos algoritmos aplicados ao *dataset smart_city-binary* de acordo com as métricas de avaliação selecionadas.

CONJUNTO	ALGORITMO	ACCURACY	PRECISION	RECALL	F1 SCORE
Conjunto 1	Todos	0.97	0.97	0.97	0.97
Conjunto 2	DTC/RFC/LRC	0.97	0.97	0.97	0.97
Conjunto 3	DTC/RFC/LRC	0.97	0.97	0.97	0.97
Conjunto 4	Todos	0.97	0.97	0.97	0.97
Conjunto 5	Todos	0.97	0.97	0.97	0.97

6.12.2 *smart_greenhouse-binary*

Em relação ao *dataset smart_greenhouse-binary*, os valores correspondentes ao tipo de tráfego na classificação binária é apresentado na Tabela 21

Tabela 21: Número de registos de cada classe no conjunto de teste do *dataset smart_greenhouse-binary* quando usada a classificação binária.

CLASSE	TIPO DE TRÁFEGO	NÚMERO DE REGISTOS
0	Normal	112 519
1	Malicioso	112 481

Tabela 22: Conjunto de atributos selecionados pelos algoritmos de seleção de atributos a serem usados no *dataset smart_city-binary*.

NÚMERO DO CONJUNTO	ATRIBUTOS
Conjunto 1	['frame.len', 'icmpv6.code', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.opt.type', 'icmpv6.rpl.opt.length', 'ipv6.nxt', 'tcp.len', 'udp.length', 'tcp.flags_0x0002', 'tcp.flags_missing']
Conjunto 2	['ip.proto', 'ipv6.nxt', 'ipv6.plen', 'ipv6.version', 'tcp.len', 'udp.length', 'ip.flags_0x00', 'ip.flags_missing', 'tcp.flags_0x0002', 'tcp.flags_missing']
Conjunto 3	['frame.len', 'icmpv6.code', 'icmpv6.rpl.opt.type', 'icmpv6.rpl.dao.sequence', 'icmpv6.type', 'tcp.len', 'udp.length', 'ip.flags_missing', 'tcp.flags_0x0002', 'tcp.flags_missing']
Conjunto 4	['icmpv6.code', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.opt.type', 'icmpv6.rpl.opt.length', 'icmpv6.type', 'ipv6.nxt', 'ipv6.version', 'udp.length', 'ip.flags_0x00', 'tcp.flags_0x0002']
Conjunto 5	['frame.len', 'icmpv6.code', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.opt.type', 'icmpv6.rpl.opt.length', 'icmpv6.type', 'tcp.len', 'udp.length', 'tcp.flags_0x0002', 'tcp.flags_missing']

Os 5 conjuntos de atributos selecionados de maneira automatizada para o *dataset smart_greenhouse-binary* são apresentados na Tabela 22:

A Tabela 23 apresenta os valores das métricas *accuracy*, *precision*, *recall* e *F1 Score* referentes aos 4 algoritmos ao serem aplicados ao *dataset smart_smartgreenhouse-binary*.

Os conjuntos 1, 2 e 4 apresentaram bons resultados na identificação dos valores maliciosos quando usados com os algoritmos DTC e LRC, conjuntos 1 e 2, e DTC e RFC com o conjunto de atributos número 4. A métrica *recall* de identificação dos registos maliciosos variou de 84% a 100%, com a média ponderada da mesma métrica a variar entre 76% a 81%. Porém, através das mesmas combinações de conjuntos e algoritmos, for apresentados valores baixos de identificação do tráfego normal, com valores a variar entre 57% a 76%.

As matrizes de confusão destes 3 conjuntos de dados podem ser visualizadas no repositório do Github. Na Figura 20 , é apresentado o exemplo do resultado obtido com a utilização do conjunto de atributos número 1.

Os algoritmos DTC e LRC em combinação com conjunto de atributos número 3, apresentaram boas taxas de *recall*, 82% e 92%, mas maus resultados quando na classificação do tráfego normal, 78% e 65%. Ainda, estes algoritmos apresentaram a média de 80% e 78% de *recall* e os mesmos valores para *F1 Score*. A Figura 21 apresenta os valores da matriz de confusão referentes aos resultados obtidos com a utilização do conjunto de atributos número 3.

Tabela 23: Resultados das métricas em relação ao *dataset smart_smartgreenhouse-binary*

CONJUNTO	ALGORITMOS	ACCURACY	PRECISION	RECALL	F1 SCORE
1	DTC	0.79	0.85	0.79	0.77
	RFC	0.77	0.79	0.77	0.77
	LRC	0.79	0.81	0.79	0.78
	KNN	0.76	0.79	0.76	0.75
2	DTC	0.81	0.82	0.81	0.81
	RFC	0.77	0.79	0.77	0.77
	LRC	0.79	0.81	0.79	0.78
	KNN	0.82	0.82	0.82	0.82
3	DTC	0.80	0.80	0.80	0.80
	RFC	0.77	0.79	0.77	0.77
	LRC	0.78	0.80	0.78	0.78
	KNN	0.82	0.82	0.82	0.82
4	DTC	0.77	0.77	0.77	0.76
	RFC	0.77	0.77	0.77	0.76
	LRC	0.74	0.75	0.74	0.74
	KNN	0.73	0.73	0.73	0.73
5	DTC	0.81	0.86	0.81	0.80
	RFC	0.82	0.85	0.82	0.82
	LRC	0.79	0.81	0.79	0.78
	KNN	0.84	0.87	0.84	0.83

Os algoritmos DTC e KNN combinados com o conjunto de atributos número 5, apresentaram boas taxas de *recall* para malicioso, 100% e 99% para ambos, mas maus resultados quanto a classificação do tráfego normal, 62% e 69%, apresentando a média ponderada de *recall* de 81% e 84% e 80% e 83% para *F1 Score*.

A Figura 22 apresenta os valores da matriz de confusão referentes aos resultados obtidos com a utilização do conjunto de atributos número 5.

Como pode ser visto, todos os 4 algoritmos, independentemente dos atributos utilizados, não apresentaram bom desempenho na identificação do tráfego normal.

A Tabela 24 apresenta os melhores resultados obtido em cada um dos conjuntos de dados ao aplicar as métricas *accuracy* e as médias ponderadas das métricas *recall*, *precision* e *F1-score*.

No conjunto 2, optou-se pelo LRF frente ao DCT, mesmo que a média do *recall* fosse maior no segundo, devido a maior índice de identificação do tráfego normal, 76% a 64%. O primeiro teve índices melhores ao identificar o tráfego malicioso, com 93% a 81%.

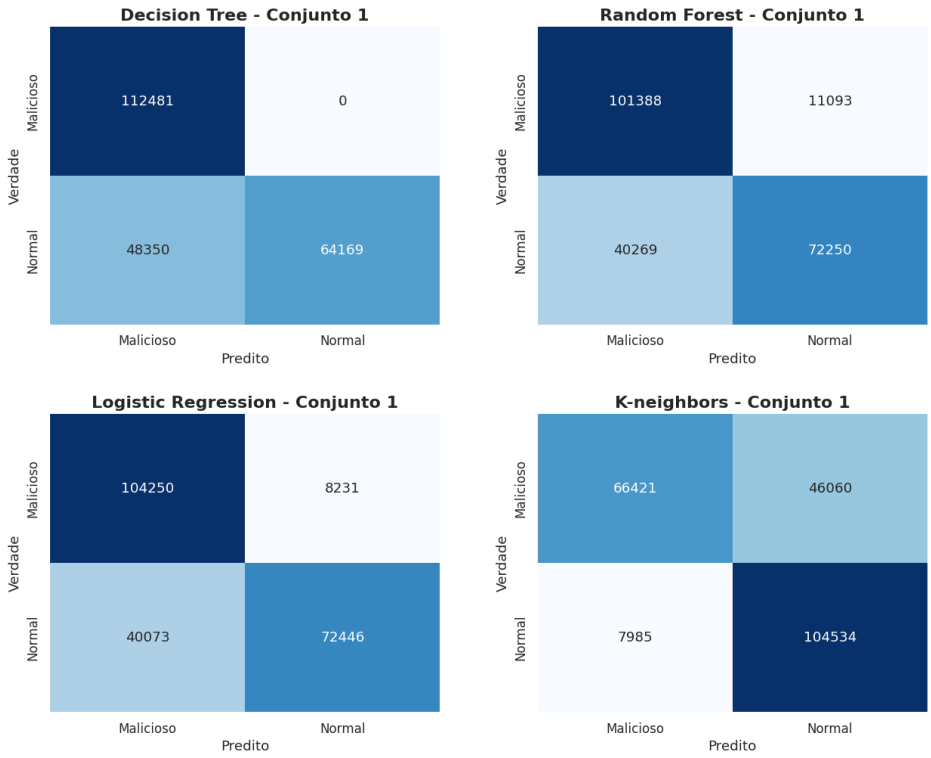


Figura 20: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-binary* com o conjunto de atributos número 1.

No conjunto 3, optou-se pelo LRC frente ao DCT, pois ele teve 10% a mais na identificação de malicioso, 92% a 82%, e uma menor no normal, 65% a 78%, ainda que a média do primeiro fosse 2% menor que o segundo 78% a 80%.

6.13 CLASSIFICAÇÃO MULTI-CLASSE

Para a classificação multi-classe, foram seguidas as mesmas etapas que as realizadas na classificação binária no que se à utilização dos algoritmos. Porem, houve a

Tabela 24: Resultados do desempenho dos algoritmos aplicados ao *dataset smart_greenhouse-binary* de acordo com as métricas de avaliação selecionadas.

CONJUNTO	ALGORITMO	ACCURACY	PRECISION	RECALL	F1 SCORE
Conjunto 1	DTC	0.79	0.85	0.79	0.77
Conjunto 2	LRC	0.79	0.81	0.79	0.78
Conjunto 3	LRC	0.78	0.80	0.78	0.78
Conjunto 4	DTC/RFC	0.77	0.77	0.77	0.76
Conjunto 5	KNN	0.84	0.87	0.84	0.83

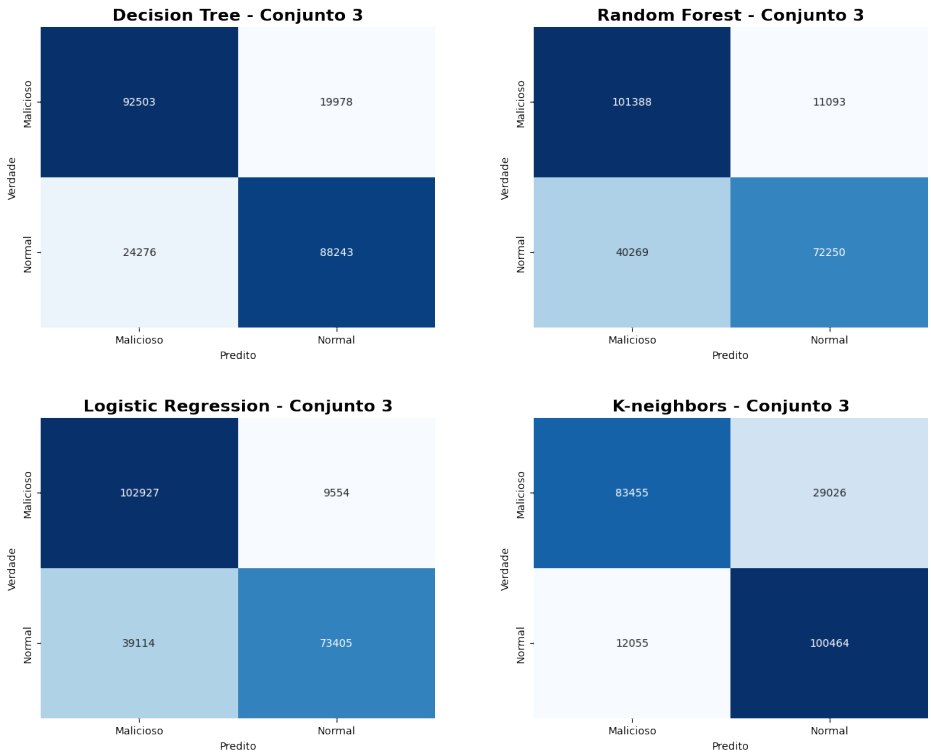


Figura 21: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-binary* com o conjunto de atributos número 3.

mudança no que diz respeito aos *datasets* utilizados e conjuntos de atributos. A diferença da classificação multi-classe em relação a classificação binária está na quantidade de classes apresentadas no atributo-alvo. No caso da classificação multi-classe, o número de classes para os dois *datasets* foi de 4 por *dataset*.

Serão apresentadas somente parte das figuras referentes as matrizes confusão dos dois *datasets*. As imagens que não forem apresentadas poderão ser encontradas no repositório do Github para posterior consulta.

Os resultados obtidos na aplicação da classificação binária nos *datasets smart_city-binary* e *smart_greenhouse-binary* e são apresentados respetivamente nas subsecções 6.13.1 e 6.13.2.

6.13.1 *smart_city-multiclass*

A Tabela 25 apresenta a quantidades de registo correspondente a cada uma das classes pertencentes ao *dataset smart_city-multiclass* no que se refere à classificação binária.

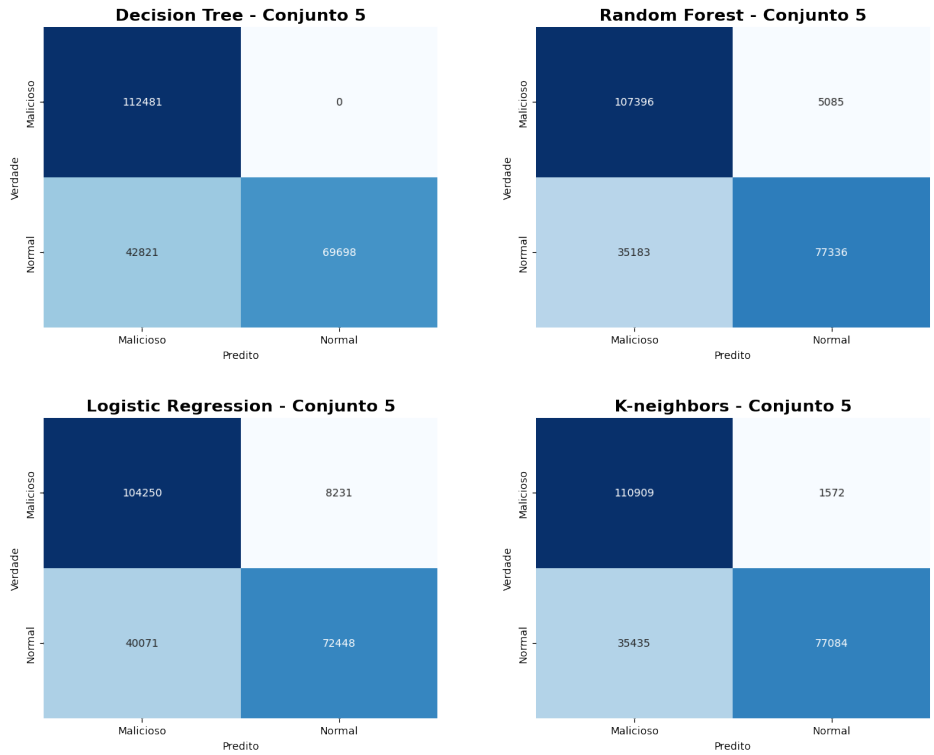


Figura 22: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-binary* com o conjunto de atributos número 5.

Tabela 25: Número de registos de cada classe no conjunto de teste do *dataset smart_city-multiclass* quando usada a classificação multi-classe.

CLASSE	TIPO DE TRÁFEGO	TOTAL
0	Normal	37 572
5	DIO Supression	37 359
6	Sink Hole	4 479
7	DoS	37 515

Os 5 conjuntos de atributos selecionados de maneira automatizada para o *dataset smart_city-multiclass* são apresentados na Tabela 26:

A Tabela 27 apresenta os valores das métricas *accuracy*, *precision*, *recall* e *F1 Score* referentes aos 4 algoritmos ao serem aplicados ao *dataset smart_city-multiclass*.

Os conjuntos 1, 4 e 5 apresentaram valores muito semelhantes nos valores da média de *recall*, com valores a variar entre 91% a 95%. No caso dos conjuntos 4 e 5, o algoritmo LRC não identificou o ataque *Sinkhole* e, no restante dos casos, os algoritmos apresentaram um valor de 15% na identificação do tráfego deste mesmo ataque. Para o restante das classes, todos os algoritmos apresentaram valores muito

Tabela 26: Conjunto de atributos selecionados pelos algoritmos de seleção de atributos a serem usados no *dataset smart_city-multiclass*.

NÚMERO DO CONJUNTO	ATRIBUTOS
Conjunto 1	['icmpv6.code', 'icmpv6.type', 'ip.len', 'ip.proto', 'ipv6.nxt', 'ipv6.version', 'udp.length', 'ip.flags_0x00', 'ip.flags_missing', 'tcp.flags_0x0002']
Conjunto 2	['frame.len', 'icmpv6.rpl.opt.length', 'ip.len', 'ip.proto', 'ipv6.nxt', 'ipv6.plen', 'ipv6.version', 'udp.length', 'ip.flags_0x00', 'ip.flags_missing']
Conjunto 3	['frame.len', 'icmpv6.rpl.opt.length', 'ip.len', 'ip.proto', 'ipv6.nxt', 'ipv6.plen', 'tcp.len', 'udp.length', 'ip.flags_0x00', 'ip.flags_missing']
Conjunto 4	['ip.proto', 'ipv6.nxt', 'ipv6.plen', 'ipv6.version', 'tcp.len', 'udp.length', 'ip.flags_0x00', 'ip.flags_missing', 'tcp.flags_0x0002', 'tcp.flags_missing']
Conjunto 5	['frame.len', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.opt.type', 'icmpv6.rpl.opt.length', 'ip.len', 'tcp.len', 'udp.length', 'ip.flags_missing', 'tcp.flags_0x0002', 'tcp.flags_missing']

Tabela 27: Resultados das métricas em relação ao *dataset smart_city-multiclass*

CONJUNTO	ALGORITMOS	ACCURACY	PRECISION	RECALL	F1 SCORE
1	DTC	0.95	0.95	0.95	0.94
	RFC	0.92	0.93	0.92	0.91
	LRC	0.94	0.91	0.94	0.92
	KNN	0.95	0.96	0.95	0.94
2	DTC	0.94	0.91	0.94	0.92
	RFC	0.90	0.89	0.90	0.89
	LRC	0.90	0.89	0.90	0.89
	KNN	0.94	0.91	0.94	0.92
3	DTC	0.94	0.91	0.94	0.92
	RFC	0.94	0.91	0.94	0.92
	LRC	0.94	0.91	0.94	0.92
	KNN	0.94	0.93	0.94	0.93
4	DTC	0.95	0.96	0.95	0.94
	RFC	0.95	0.94	0.95	0.94
	LRC	0.94	0.91	0.94	0.92
	KNN	0.95	0.96	0.95	0.94
5	DTC	0.95	0.96	0.95	0.94
	RFC	0.92	0.93	0.92	0.91
	LRC	0.94	0.91	0.94	0.92
	KNN	0.95	0.96	0.95	0.94

altos de identificação destas classes. A Figura 23 apresenta, por exemplo, a matriz de confusão dos algoritmos quando aplicados ao conjunto de atributos número 1.

Figura 23: Matriz de confusão dos algoritmos aplicados ao *dataset smart_city-multiclass* com o conjunto de atributos número 1.

No caso do conjunto de atributos número 2, nenhum dos algoritmos foi capaz de identificar os registos correspondentes ao ataque *Sinkhole*, conforme apresentado na Figura 24.

Por fim, o algoritmo KNN foi o único a conseguir identificar os registos correspondentes ao ataque *Sinkhole* quando utilizado o conjunto de atributos número 5, com uma taxa de acerto de 31%, sendo esta a mais alta entre todos os conjuntos de dados. Para este mesmo algoritmo, no que se refere às demais classes, foram apresentados os valores de *recall* de 95%, 100% e 94%, para as classes normal, *DIO Supression* e DoS respetivamente, sendo apresentado o valor médio de *recall* de 94%. Para os demais algoritmos, ainda que estes tenham apresentado valores muito altos de média de *recall*, eles não foram capazes de identificar o tráfego correspondente ao ataque *Sinkhole*. A Figura 25 apresenta a matriz de confusão dos algoritmos quando estes utilizaram o conjunto de atributos número 5:

A Tabela 28 apresenta os melhores resultados obtido em cada um dos conjuntos de dados ao aplicar as métricas *accuracy* e as médias ponderadas das métricas *recall*, *precision* e *F1-score*. Conjunto 5, KNN ao invés do DTC, pois o primeiro tem 1% a mais na capacidade de identificar corretamente o tráfego do tipo normal.

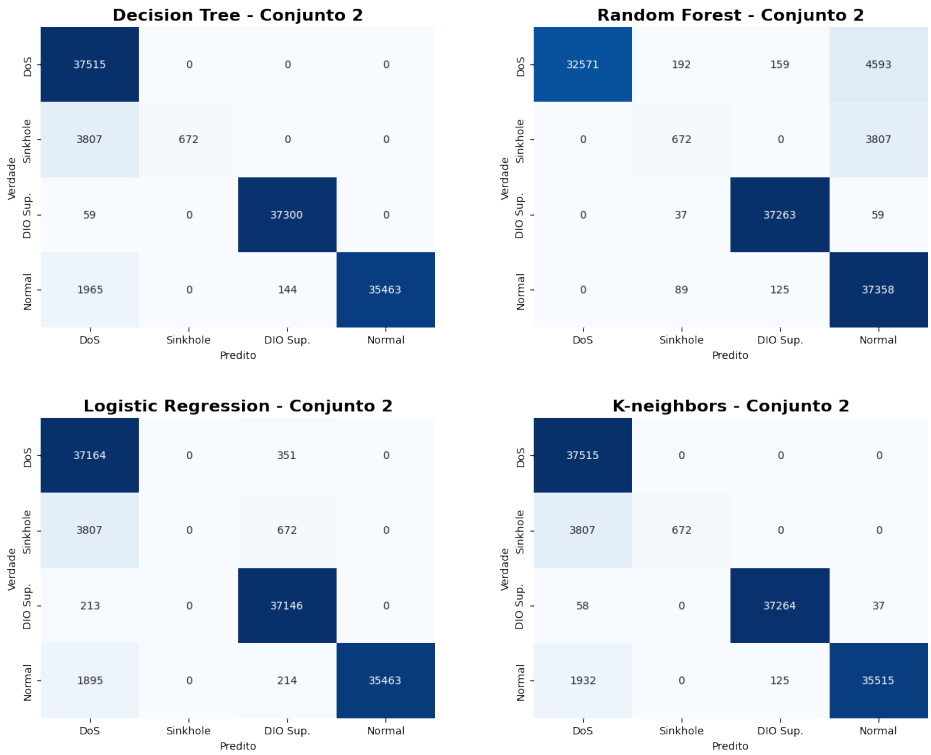


Figura 24: Matriz de confusão dos algoritmos aplicados ao *dataset smart_city-multiclass* com o conjunto de atributos número 2.

6.13.2 *smart_greenhouse-multiclass*

Em relação ao *dataset smart_greenhouse-multiclass*, os valores correspondentes ao tipo de tráfego na classificação binária é apresentado na Tabela 29.

Os 5 conjuntos de atributos selecionados de maneira automatizada para o *dataset smart_city-multiclass* são apresentados na Tabela 30:

A Tabela 31 apresenta os valores referentes às métricas *accuracy*, *precision*, *recall* e *F1 Score* dos algoritmos ao serem aplicados ao *dataset smart_greenhouse-multiclass*.

Tabela 28: Resultados do desempenho dos algoritmos aplicados ao *dataset smart_city-multiclass* de acordo com as métricas de avaliação selecionadas.

CONJUNTO	ALGORITMO	ACCURACY	PRECISION	RECALL	F1 SCORE
Conjunto 1	KNN	0.95	0.96	0.95	0.94
Conjunto 2	Nenhum	0	0	0	0
Conjunto 3	KNN	0.94	0.93	0.94	0.93
Conjunto 4	DCT/KNN	0.95	0.96	0.95	0.94
Conjunto 5	RFC/KNN	0.95	0.96	0.95	0.94

Tabela 29: Número de registos de cada classe no conjunto de teste do *dataset smart_grenhouse-multiclass* quando usada a classificação multi-classe.

CLASSE	TIPO DE TRÁFEGO	TOTAL
0	Normal	37 616
1	Black Hole	37 447
2	Hello Flood	37 255
3	Version Number	37 682

Tabela 30: Conjunto de atributos selecionados pelos algoritmos de seleção de atributos a serem usados no *dataset smart_greenhouse-multiclass*.

NÚMERO DO CONJUNTO	ATRIBUTOS
Conjunto 1	['icmpv6.code', 'icmpv6.rpl.dio.rank', 'tcp.dstport', 'frame.protocols', '6lowpan.next_0x00', '6lowpan.next_0x3a', '6lowpan.next_missing', 'tcp.flags_0x0018', 'wpan.dst_addr_mode_0x0002', 'wpan.dst_addr_mode_0x0003']
Conjunto 2	['coap.code', 'icmpv6.code', 'icmpv6.rpl.dio.dtsn', 'icmpv6.rpl.dio.version', 'ipv6.nxt', 'frame.protocols', '6lowpan.next_0x3a', '6lowpan.next_missing', 'wpan.dst_addr_mode_0x0002', 'wpan.dst_addr_mode_0x0003']
Conjunto 3	['frame.len', 'coap.type', 'icmpv6.code', 'icmpv6.rpl.dio.dtsn', 'icmpv6.rpl.dio.rank', 'icmpv6.rpl.dio.version', 'ipv6.nxt', 'udp.length', '6lowpan.next_0x3a', '6lowpan.next_missing']
Conjunto 4	['icmpv6.rpl.dio.rank', 'udp.length', '6lowpan.next_missing', 'mqtt.topic_missing', 'mqtt.topic_sensors/cmd/+/fmt/json', 'mqtt.topic_sensors/evt/status/fmt/json', 'tcp.flags_0x0002', 'tcp.flags_0x0010', 'tcp.flags_0x0011', 'tcp.flags_0x0014']
Conjunto 5	['coap.code', 'coap.type', 'icmpv6.code', 'icmpv6.rpl.dio.rank', 'ipv6.nxt', 'tcp.len', 'udp.length', '6lowpan.next_0x00', '6lowpan.next_0x3a', 'tcp.flags_missing']

Figura 25: Matriz de confusão dos algoritmos aplicados ao *dataset smart_city-multiclass* com o conjunto de atributos número 5.

O conjunto de atributos número 1, combinados aos algoritmos DTC e RFC apresentaram bons resultados médios de *recall*, 84%. Ainda, ambos os algoritmos apresentaram bons resultados na identificação das classes relacionadas ao tráfego maliciosos, cujos valores a variar entre 86% 95%. Para quase todos os algoritmos, os resultados na identificação no tráfego normal ficou abaixo dos 70%, com o algoritmo KNN a apresentar o valor de a 77% para o tráfego normal, mas a apresentar o valor de 36% na identificação do ataque *version number*. A Figura 26 apresenta os valores da matriz de confusão referentes aos resultados obtidos com a utilização do conjunto de atributos número 1. *dataset smart_greenhouse-multiclass* quando usado o conjunto de atributos número 1.

Os algoritmos DTC e RFC, quando utilizados com o conjunto de atributos 2, apresentaram bons resultados médios de *recall*, 82% , o primeiro a apresentar melhores resultados na identificação no tráfego normal e o segundo na do ataque *version number*, com resultado equivalentes na identificação dos ataques *blackhole* e *hello flood*. Em todos os algoritmos foi apresentado um índice de identificação do tráfego normal abaixo dos 70%. O algoritmo KNN, ainda que a identificação do tráfego normal tenha ficado em 65%, a percentagem de identificação da classe referente ao ataque *blackhole* ficou a 96%, sendo apresentado para os demais ataques

Tabela 31: Resultados das métricas em relação ao *dataset smart_greenhouse-multiclass*

CONJUNTO	ALGORITMOS	ACCURACY	PRECISION	RECALL	F1 SCORE
1	DTC	0.84	0.87	0.84	0.84
	RFC	0.84	0.87	0.84	0.84
	LRC	0.80	0.81	0.80	0.80
	KNN	0.74	0.77	0.74	0.72
2	DTC	0.82	0.84	0.82	0.82
	RFC	0.82	0.84	0.82	0.82
	LRC	0.74	0.75	0.74	0.74
	KNN	0.81	0.83	0.81	0.81
3	DTC	0.83	0.86	0.83	0.83
	RFC	0.84	0.87	0.84	0.84
	LRC	0.78	0.80	0.78	0.78
	KNN	0.85	0.87	0.85	0.85
4	DTC	0.72	0.83	0.72	0.73
	RFC	0.74	0.82	0.74	0.74
	LRC	0.70	0.75	0.70	0.70
	KNN	0.72	0.82	0.72	0.72
5	DTC	0.82	0.85	0.82	0.83
	RFC	0.83	0.86	0.83	0.83
	LRC	0.79	0.81	0.79	0.80
	KNN	0.83	0.86	0.83	0.83

a percentagem foi de 86 e 77%. O algoritmo LRC apresentou maus resultados na identificação do ataque *version number*, cujo valor apresentado foi 55%. A Figura 27 apresenta os valores da matriz de confusão referentes aos resultados obtidos com a utilização do conjunto de atributos número 2.

Com o conjunto de atributos 3, foi obtido o valor de 85% de média ponderada de *recall* com o algoritmo KNN, sendo este o melhor resultado para o atual *dataset*. Para todos os algoritmos, a percentagem na identificação do tráfego normal ficou abaixo dos 70%, mas, em contrapartida, há bons índices de identificação do tráfego das classes referentes aos ataques, cujas percentagens variariam entre 78% a 97%. A Figura 28 apresenta a matriz de confusão referentes ao *dataset smart_greenhouse-multiclass* quando usado o conjunto de atributos 3.

O conjunto de atributos 4 apresentaram baixos índices médios de *recall*, onde os algoritmos DTC, RFC e LRC tiveram uma taxa de acerto de 46% para a classificação do ataque *blackhole*, assim como para tráfego normal, cujo índice variou de 62% a 67%. O algoritmo KNN apresentou resultados na mesma faixa dos demais para o tráfego normal, 68%, resultados melhores na identificação do ataque *blackhole*, 93%, mas resultados piores na identificação do ataque *version number*, 40%. A Figura 29

Figura 26: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-multiclass* com o conjunto de atributos número 1.

apresenta a matriz de confusão referentes ao *dataset smart_greenhouse-multiclass* quando usado com o conjunto de atributos número 4.

Por fim, com o conjunto de atributos número 5, os algoritmos RFC e KNN apresentaram resultados muito parecidos, onde foi apresentado para ambos a média ponderada de *recall* de 83%, mas o segundo algoritmo apresentou resultados melhores nas métricas *precision* e, consequentemente, *F1 Score*. Deve ser destacado que para ambos os algoritmos os valores de *recall* na identificação do tráfego normal ficou na casa dos 60% . O resultado da métrica *recall* na identificação do tráfego malicioso foi bom, variando entre 78 e 91%. A Figura 30 apresenta a matriz de confusão referentes ao *dataset smart_greenhouse-multiclass* quando usado com o conjunto de atributos número 5.

Como pode ser visto, de forma equivalente ao que foi encontrado na classificação binária, os resultados dos 4 algoritmos na classificação do tráfego normal não foram bons, independentemente dos atributos utilizados.

A Tabela 32 apresenta os melhores resultados obtido em cada um dos conjuntos de dados ao aplicar as métricas *accuracy* e as médias ponderadas das métricas *recall*, *precision* e *F1-score*.

Figura 27: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-multiclass* com o conjunto de atributos número 2.

No conjunto de atributos número 2, o RFC possui uma maior quantidade de identificação de ataques que o DTC, 78% a 77%, que possui mais em normal, 67% a 66%, por isso o primeiro foi selecionado.

No conjunto 4, foi selecionado o modelo RFC frente ao DTC, pois o primeiro possui 5% a mais de deteção do tráfego normal em troca de 1% a menos de capacidade de identificar o ataque *version number*, maior média de *recall*.

Tabela 32: Resultados do desempenho dos algoritmos aplicados ao *dataset smart_smartgreenhouse-multiclass* de acordo com as métricas de avaliação selecionadas.

CONJUNTO	ALGORITMO	ACCURACY	PRECISION	RECALL	F1 SCORE
Conjunto 1	DTC/RFC	0.84	0.87	0.84	0.84
Conjunto 2	RFC	0.82	0.84	0.82	0.82
Conjunto 3	KNN	0.85	0.87	0.85	0.85
Conjunto 4	RFC	0.74	0.82	0.74	0.74
Conjunto 5	RFC/KNN	0.83	0.86	0.83	0.83

Figura 28: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-multiclass* com o conjunto de atributos número 3.

6.14 RESULTADOS

O ponto que mais surpreendeu nos resultados obtidos foi o desempenho dos algoritmos em identificar o tráfego normal nos dois *datasets* baseados no *dataset smart_greenhouse*. Ainda que apresentassem um ótimo desempenho na identificação dos ataques, seja na classificação binária, seja classificação multi-classe, os resultados obtidos na classificação do tráfego normal foram consideravelmente baixos.

Para o *dataset smart_greenhouse-binary*, todas as combinações de conjuntos de atributos e algoritmos não apresentaram resultados importantes na identificação do tráfego normal. Com base no que foi apresentado, o melhor conjunto de atributos é o 5 para malicioso e o 3 para o tráfego normal. O algoritmo DTC esteve sempre presente em todos os conjuntos com os melhores resultados, ainda que não foi o com melhor resultado quando considerada a média ponderada de *recall*. O melhor valor de média do *recall* foi apresentado no conjunto 5 com o KNN, com média de 84% de *recall*, ainda que o índice de identificação do tráfego normal estivesse abaixo do normal. Com o conjunto 5, DTC teria uma maior eficiência na identificação dos maliciosos, mas apresentaria uma maior taxa de alertas referentes a falsos-

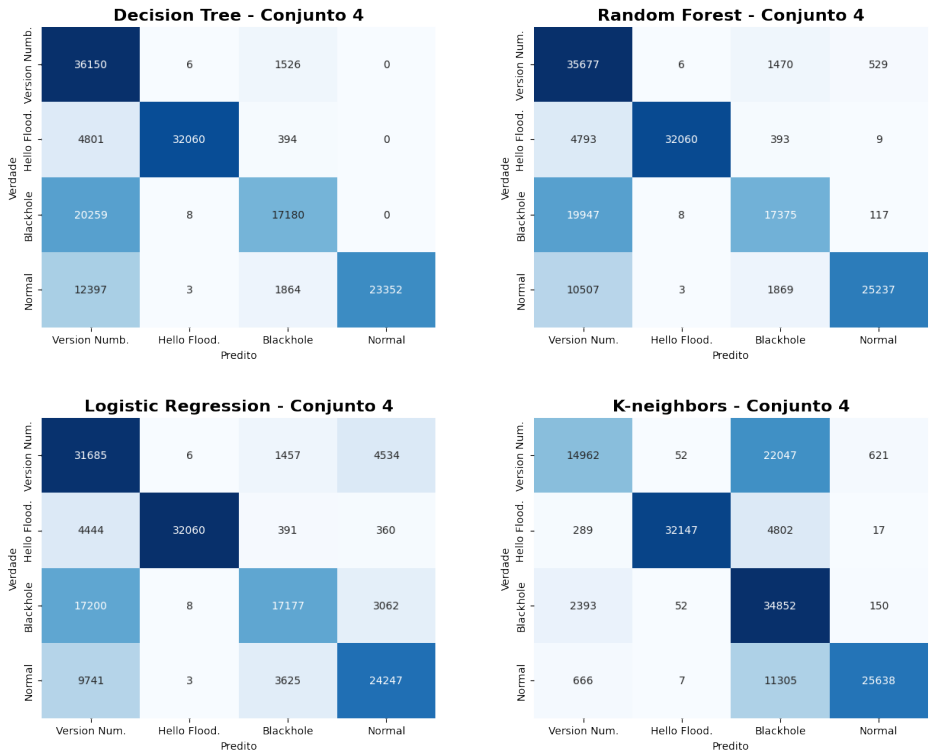


Figura 29: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-multiclass* com o conjunto de atributos número 4.

positivos se comparado ao KNN, que teria menos falso-positivos, mas uma leve menor eficiência na identificação dos maliciosos.

Já, em relação ao *dataset smart_greenhouse-multiclass*, apesar de uma boa capacidade de identificar os registos maliciosos, ainda há a possibilidade de melhoria dos modelos em relação a identificação dos registos normais, pois foram apresentados valores consideravelmente maus na identificação do tráfego normal e um número considerável de falsos-negativos. Para todos os algoritmos, a percentagem na identificação do tráfego normal ficou abaixo dos 70%, mas, em contrapartida, há bons índices de identificação do tráfego das classes referentes aos ataques, cujas percentagens variam entre 78% a 97%. O algoritmo que atenderia da melhor forma para identificação dos ataques deste *dataset* é o KNN com o conjunto de atributos 3, onde foi obtido o valor de 85% de média ponderada de *recall*, sendo este o melhor resultado para o atual *dataset*.

No que se refere aos *datasets smart_city*, os algoritmos tiveram uma desempenho muito alto quando na identificação dos ataques DoS e *DIO Supression*, mas tiveram dificuldade de identificar o ataque do tipo *Sinkhole*. Isso pode ser consequência da pequena quantidade de registos pertencentes a esta classe se comparado ao número

Figura 30: Matriz de confusão dos algoritmos aplicados ao *dataset smart_greenhouse-multiclass* com o conjunto de atributos número 5.

de registos das demais classes. Em ambos os *datasets*, a identificação das classes referentes ao tráfego normal foram muito bons.

Com o *dataset smart_city-binary*, os algoritmos DTC, RFC e LRC apresentaram ótimos resultados quando avaliados em relação a média ponderada da métrica *recall* em combinação com todos os conjuntos de atributos. Esse resultado pode indicar que os dados referentes a ataques podem ser facilmente preditos, o que pode ser um sinal de *overfitting* do modelo ou que há um ou mais atributos que relacionam diretamente um registo a um ataques, pois os algoritmos conseguiram identificar todos os registos maliciosos. O KNN apresentou um resultado abaixo dos demais quando usado em combinação com os conjuntos de atributos números 2 e 3. Qualquer que seja o conjunto de dados e os algoritmos escolhidos, salvo excepção do KNN para os conjuntos 2 e 3, haverá um bom desempenho na classificação dos dados.

Em relação ao *dataset smart_city-multiclass*, os algoritmos apresentaram um resultado muito semelhante no que se refere ao desempenho encontrado na classificação binária. Dentre os resultados obtidos, o conjunto de dados que apresentou o melhor resultado foi obtido com o algoritmo KNN em combinação com conjunto de atributos número 3, uma vez que conseguiu identificar a maior quantidade de

ataques do tipo *Sinkhole*, 31% da amostra, acima dos 15% e 0% apresentado com no resultados dos outros conjuntos de atributos. Com isso, este algoritmo atingiu a marca de 94% na métrica *recall weighted avg.* Devido a isso, este algoritmo poderia ser utilizado juntamente ao conjunto de atributos 3 para a classificação multi-classe dos dados do *dataset smart_city-multiclass*. O conjunto de atributos número 2 não identificou nenhuma ataque *Sinkhole*, o que demonstrou que ele não é bom. Os demais conjuntos de dados apresentaram o valor de 95% na métrica *recall weighted avg.* quando utilizados em combinação com os algoritmos DTC e RFC, mas conseguiam identificar apenas 15% do tráfego referente ao ataque *Sinkhole*. Nota-se que alguns algoritmos não conseguiram identificar o ataque do tipo *Sinkhole*, o que pode ser uma consequência da pequena quantidade de registos pertencentes a esta classe se comparado ao número de registos das demais classes.

7

CONCLUSÕES

Machine Learning e *Internet of Things* são termos muito frequentes no cotidiano das pessoas hoje em dia. Esta realidade não é diferente ao que acontece na agricultura, que passou a utilizar de tecnologia para o aprimoramento dos seus processos de produção.

Em relação a isso, este trabalho buscou criar dois *datasets* compostos por dados IoT, um voltado a representar dados relacionados a *smart cities* e o outro a representar dados relacionados com *smart greenhouses* e, no final do processo de criação destes *datasets*, analisar se era possível identificar os ataques apresentados nestes *datasets*.

Foi realizado o levantamento das ferramentas comumente utilizadas na literatura para gerar tráfego de dados. Após uma análise experimental, foram identificadas as características e limitações destas ferramentas na realização do processo de geração de dados. As ferramentas Netsim e Contiki foram então selecionadas e usadas para criar os ambientes e realizar a simulação dos dados destes ambientes.

Na fase de avaliação da ferramentas, não houve dificuldade para executar as simulações, uma vez que foram utilizados os ficheiros já contidos de modo padrão nas ferramentas. Porém, houve um grande acréscimo de complexidade quando na criação ou adaptação de alguns ficheiros da ferramenta Contiki. Os ficheiros em questão são referentes aos sensores CoAP e MQTT e referentes aos ficheiros que implementavam os ataques. Primeiramente, houve dificuldade de encontrar código referentes a ataques para a ferramenta Contiki. Quando encontrado, foi dispendido um grande esforço para identificar um exemplo de código que funcionasse corretamente com os tipos de *motes* usados nas simulações MQTT. E não somente, foi necessária a alteração do código padrão, uma vez que estes realizavam operações dispensáveis, como a dependência de carregar no botão do sensor para habilitar o envio das mensagens nos sensores CoAP.

Como forma de tentar facilitar o processo de configuração desta ferramenta, foram criados pelo autor os *scripts preparing.sh* e *config_system.sh*. Estes *scripts* são de grande valia para futuras utilizações, uma vez que automatizam a configuração do

sistema e da ferramenta Contiki, reduzindo drasticamente o tempo necessário e a complexidade relacionados ao processo de configuração da ferramenta Contiki.

Quanto a ferramenta Netsim, as sua maior limitação está relacionada ao módulo *NetSim Network Emulator*, o que poderia trazer maior diversidade para as simulações executadas nesta ferramenta. Apesar dos testes, não foi possível utilizar mais que 2 dispositivos externos sem que houvesse perda de informação. Outra limitação encontra-se no fato de a ferramenta não apresentar suporte ao protocolo MQTT. A última limitação está relacionada com o fato de a ferramenta apenas representar os dados do protocolo CoAP ao invés de implementá-lo realmente. Desta forma, os dados gerados são representados como dado UDP na captura.

Com o objetivo de realizar a análise de fluxo de dados, foi também realizado o levantamento das ferramentas usadas para gerar fluxos de dados com base em ficheiros PCAP. Foram então identificadas 4 ferramentas e com elas foram executados alguns testes com ficheiros PCAP gerados nas simulações das ferramentas Contiki e Netsim, mas estas foram então descartadas devido a não conseguirem processar os ficheiros PCAP, devido a falta de suporte aos protocolos RPL e IEEE 802.15.4 (LR-WPAN). Graças a esta limitação, não foi possível realizar a análise comparativa dos resultados obtidos no atual trabalho com as estatísticas obtidas na geração dos fluxos de dados. Desta forma, o foco da análise foi integralmente dedicado à análise de atributos de pacotes de dados.

Na etapa de pré-processamento de dados, foi criado um *script* que eliminou a maior parte dos atributos não relevantes e, ao final da execução deste *script*, foram eliminadas ainda outros atributos não relevantes, sendo levado em consideração o valor do atributo em relação ao protocolo ao qual ele está relacionado. Foram então criados os *datasets smart_city* e *smart_greenhouse*, compostos por 14 281 777 registos e 32 atributos e 16 319 307 registos e 38 atributos respetivamente.

Como forma de validar a capacidade destes algoritmos, a eles foram aplicados algoritmos ML. Para isso, foram selecionados 4 algoritmos de aprendizado de máquina e foram usados 5 algoritmos de seleção de atributos para identificar de forma automatizada os 10 melhores atributos em cada *dataset*.

Através da métrica *recall weighted avg.*, pode ser realizada a avaliação dos modelos criados ao serem aplicados aos dados dos 4 *datasets* que contém parte dos *datasets* originais. No geral, foram obtidos bons resultados quanto à identificação do tráfego malicioso, que era o objetivo principal do modelos.

Como demonstrado no exemplo dos algoritmos aplicados aos *datasets* baseados no *dataset smart_greenhouse*, houve uma baixa capacidade do algoritmos a identificar

o tráfego normal, apesar de terem sido encontrados ótimos resultados no que se refere ao tráfego malicioso. Essa realidade está diretamente relacionada com os atributos que foram utilizados pelos modelos de ML.

Em relação aos *dataset* baseados no *dataset smart_city*, os resultados, na maior parte dos casos, foi muito bom. Porém, foi apresentado baixo desempenho na identificação do ataque do tipo *Sinkhole*, sendo que alguns algoritmos nem sequer conseguiram identificar este ataque. Isso pode ser consequência da pequena quantidade de registos pertencentes a esta classe se comparado ao número de registos das demais classes.

Foi diretamente experienciado que a quantidade de dados para cada classe é de extrema importância para o bom desempenho dos algoritmos de aprendizado de máquina. Outro ponto de extrema importância que foi identificado é a necessidade da correta seleção dos atributos, pois pode ocorrer uma diferença considerável do desempenho do modelo ao usar diferentes combinações de atributos.

Foi possível demostrar que os *datasets* originais, mesmo que não tenham sido utilizados integralmente, podem ser aplicados a algoritmos de aprendizagem e, assim, contribuir para o desenvolvimento de ferramentas que busquem identificar ataques como os que foram apresentados no atual trabalho.

Em suma, pode-se afirmar que os objetivos inicialmente propostos foram atingidos no atual trabalho.

7.1 CONTRIBUTOS

No atual trabalho foram criados alguns contributos que, aos olhos do autor, são de grande valor para a academia. Estes contributos são apresentados na lista abaixo:

- *Script* de configuração do sistema e das simulações do Contiki para simular um ambiente composto por tráfego normal e por tráfego malicioso de diferentes tipos;
- *Dataset smart_city* que contém 14 281 777 milhões de registos referente a tráfego normal e malicioso obtido em ambiente IoT simulado e rotulado segundo o tipo de tráfego e ataque;
- *Dataset smart_greenhouse* que contém 16 319 307 milhões de registos referente a tráfego normal e malicioso obtido em ambiente IoT simulado e rotulado segundo o tipo de tráfego e ataque;

- *Script* de tratamento dos dados brutos contemplando seleção de atributos, limpeza e transformação de dados;
- Aplicação de 4 algoritmos de *machine learning* sobre os conjuntos de dados *smart_city* e *smart_greenhouse* para realização de classificação binária e multi-classes, avaliação dos resultados e disponibilização do código-fonte;
- Levantamento das ferramentas de geração de dados, sendo identificadas as suas características e limitações.

7.2 TRABALHOS FUTUROS

Como recomendação para trabalhos futuros, são apontados os casos a seguir:

- Gerar fluxos de dados dos ficheiros PCAP criados nas simulações;
- Realizar análises baseadas em fluxos de tráfego e comparar com os resultados obtidos no atual trabalho;
- Avaliar eventual melhoria no desempenho dos modelos utilizados nesse trabalho através de *tuning* dos hiperparâmetros e da engenharia de atributos;
- Desenvolver e aplicar modelos baseados em redes neuronais e algoritmos de *machine learning* não supervisionados e comparar os resultados com estes algoritmos com resultados obtidos no do atual trabalho.

A primeira etapa seria realizar a geração dos fluxos de dados, uma vez que não foi possível gerá-los no atual trabalho, devido às limitações já mencionadas. Posteriormente, realizar a análise comparativa dos resultados obtidos no atual trabalho com as estatísticas obtidas na geração dos fluxos de tráfego.

Os modelos criados neste trabalho foram utilizados para mostrar a utilidade dos *datasets* criado e criar uma *baseline* de comparação para os modelos futuros a utilizar um ou ambos os *datasets*. Os modelos criados no atual trabalho não apresentaram um bom desempenho na identificação do tráfego normal. Desta forma, pode ser realizado o processo de *tuning* dos modelos através da alteração dos seus hiperparâmetros. Além disso, pode ser também realizada uma nova etapa de engenharia de atributos, de forma a identificar novos conjuntos de atributos que impactem positivamente o desempenho do modelo.

Por fim, desenvolver modelos baseados em redes neuronais e algoritmos de *machine learning* não supervisionados. Através desses algoritmos poderá ser realizada uma análise do impacto do rótulos na qualidade dos dados.

BIBLIOGRAFIA

Accettura, Nicola et al. (2011). «Performance analysis of the RPL routing protocol». Em: *2011 IEEE International Conference on Mechatronics*. IEEE, pp. 767–772.

Alam, Shadab et al. (2020). «Internet of things (IoT) enabling technologies, requirements, and security challenges». Em: *Advances in Data and Information Sciences: Proceedings of ICDIS 2019*. Springer, pp. 119–126.

Arvind, S e V Anantha Narayanan (2019). «An overview of security in CoAP: attack and analysis». Em: *2019 5th international conference on advanced computing & communication systems (ICACCS)*. IEEE, pp. 655–660.

Balaji, Subramanian, Karan Nathani e Rathnasamy Santhakumar (2019). «IoT technology, applications and challenges: a contemporary survey». Em: *Wireless personal communications* 108, pp. 363–388.

Bansal, Sharu e Dilip Kumar (2020). «IoT ecosystem: A survey on devices, gateways, operating systems, middleware and communication». Em: *International Journal of Wireless Information Networks* 27, pp. 340–364.

Booij, Tim M et al. (2021). «ToN_IoT: The Role of Heterogeneity and the Need for Standardization of Features and Attack Types in IoT Network Intrusion Datasets». Em: *IEEE Internet of Things Journal*.

Buczak, Anna L e Erhan Guven (2015). «A survey of data mining and machine learning methods for cyber security intrusion detection». Em: *IEEE Communications surveys & tutorials* 18.2, pp. 1153–1176.

Claise, Benoît (out. de 2004). *Cisco Systems NetFlow Services Export Version 9*. RFC 3954. DOI: 10.17487/RFC3954. URL: https://www.rfc-editor.org/info/rfc3954.

Cooja Simulator - Contiki (2016). https://anrg.usc.edu/contiki/index.php/Cooja_Simulator. (Accessed on 02/12/2023).

Dargan, Shaveta et al. (2019). «A survey of deep learning and its applications: a new paradigm to machine learning». Em: *Archives of Computational Methods in Engineering*, pp. 1–22.

Dwibedi, Smirti, Medha Pujari e Weiqing Sun (2020). «A Comparative Study on Contemporary Intrusion Detection Datasets for Machine Learning Research». Em: *2020 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, pp. 1–6.

Ferrag, Mohamed Amine, Othmane Friha et al. (2022). «Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning». Em: *IEEE Access* 10, pp. 40281–40306.

Ferrag, Mohamed Amine, Leandros Maglaras et al. (2020). «Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study». Em: *Journal of Information Security and Applications* 50, p. 102419.

Al-Garadi, Mohammed Ali et al. (2020). «A survey of machine and deep learning methods for internet of things (IoT) security». Em: *IEEE Communications Surveys & Tutorials* 22.3, pp. 1646–1685.

Granjal, Jorge, Edmundo Monteiro e Jorge Sá Silva (2015). «Security for the internet of things: a survey of existing protocols and open research issues». Em: *IEEE Communications Surveys & Tutorials* 17.3, pp. 1294–1312.

Al-Hadhrami, Yahya e Farookh Khadeer Hussain (2020). «Real time dataset generation framework for intrusion detection systems in IoT». Em: *Future Generation Computer Systems* 108, pp. 414–423.

Hameed, Ali e Alauddin Alomary (2019). «Security issues in IoT: A survey». Em: *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*. IEEE, pp. 1–5.

Hassan, Wan Haslina et al. (2019). «Current research on Internet of Things (IoT) security: A survey». Em: *Computer networks* 148, pp. 283–294.

Hassija, Vikas et al. (2019). «A survey on IoT security: application areas, security threats, and solution architectures». Em: *IEEE Access* 7, pp. 82721–82743.

Hindy, Hanan et al. (2021). «Machine learning based IoT intrusion detection system: An MQTT case study (MQTT-IoT-IDS2020 dataset)». Em: *Selected Papers from the 12th International Networking Conference: INC 2020*. Springer, pp. 73–84.

Ibitoye, Olakunle, Omair Shafiq e Ashraf Matrawy (2019). «Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks». Em: *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, pp. 1–6.

IDS 2017 | Datasets | Research | Canadian Institute for Cybersecurity | UNB (2021). [Online; accessed 4. Jul. 2021]. URL: https://www.unb.ca/cic/datasets/ids-2017.html.

IDS 2018 | Datasets | Research | Canadian Institute for Cybersecurity | UNB (2021). [Online; accessed 4. Jul. 2021]. URL: https://www.unb.ca/cic/datasets/ids-2018.html.

IOT Technologies and protocols: Microsoft Azure (2022). [Online; accessed 21. Jan. 2022]. URL: https://azure.microsoft.com/en-us/overview/internet-of-things-iot/iot-technology-protocols/.

Kolias, Constantinos et al. (2017). «DDoS in the IoT: Mirai and other botnets». Em: *Computer* 50.7, pp. 80–84.

Koroniotis, Nickolaos et al. (2019). «Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset». Em: *Future Generation Computer Systems* 100, pp. 779–796.

Lamaazi, Hanane e Nabil Benamar (2020). «A comprehensive survey on enhancements and limitations of the RPL protocol: A focus on the objective function». Em: *Ad Hoc Networks* 96, p. 102001.

Li, Lan (2012). «Study on security architecture in the Internet of Things». Em: *Proceedings of 2012 international conference on measurement, information and control*. Vol. 1. IEEE, pp. 374–377.

Liang, Xingwei e Yoohwan Kim (2021). «A Survey on Security Attacks and Solutions in the IoT Network». Em: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, pp. 0853–0859.

Meidan, Yair et al. (2018). «N-baiot—network-based detection of iot botnet attacks using deep autoencoders». Em: *IEEE Pervasive Computing* 17.3, pp. 12–22.

Meneghello, Francesca et al. (2019). «IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices». Em: *IEEE Internet of Things Journal* 6.5, pp. 8182–8201.

Mohanta, Bhabendu Kumar et al. (2020). «Survey on IoT security: Challenges and solution using machine learning, artificial intelligence and blockchain technology». Em: *Internet of Things* 11, p. 100227.

Moustafa, Nour et al. (2020). «Federated TON_IoT Windows datasets for evaluating AI-based security applications». Em: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, pp. 848–855.

Parmisano, A, Sebastian Garcia e Maria Jose Erquiaga (2020). «A labeled dataset with malicious and benign iot network traffic». Em: *Stratosphere Laboratory: Praha, Czech Republic*.

Ring, Markus et al. (2019). «A survey of network-based intrusion detection data sets». Em: *Computers & Security* 86, pp. 147–167.

Roopak, Monika, Gui Yun Tian e Jonathon Chambers (2019). «Deep learning models for cyber security in IoT networks». Em: *2019 IEEE 9th annual computing and communication workshop and conference (CCWC)*. IEEE, pp. 0452–0457.

Said, Omar e Mehedi Masud (2013). «Towards internet of things: Survey and future vision». Em: *International Journal of Computer Networks* 5.1, pp. 1–17.

Sain, Mangal, Young Jin Kang e Hoon Jae Lee (2017). «Survey on security in Internet of Things: State of the art and challenges». Em: *2017 19th International*

conference on advanced communication technology (ICACT). IEEE, pp. 699–704.

Sharafaldin, Iman, Arash Habibi Lashkari e Ali A Ghorbani (2018). «Toward generating a new intrusion detection dataset and intrusion traffic characterization.» Em: *ICISSp* 1, pp. 108–116.

Simha, SN Vikram et al. (2020). «A review of RPL protocol using contiki operating system». Em: *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*. IEEE, pp. 259–264.

Sobin, CC (2020). «A survey on architecture, protocols and challenges in IoT». Em: *Wireless Personal Communications* 112.3, pp. 1383–1429.

Soni, Dipa e Ashwin Makwana (2017). «A survey on mqtt: a protocol of internet of things (iot)». Em: *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*. Vol. 20, pp. 173–177.

Tahsien, Syeda Manjia, Hadis Karimipour e Petros Spachos (2020). «Machine learning based solutions for security of Internet of Things (IoT): A survey». Em: *Journal of Network and Computer Applications* 161, p. 102630.

Tavallaee, Mahbod et al. (2009). «A detailed analysis of the KDD CUP 99 data set». Em: *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee, pp. 1–6.

Tewari, Aakanksha e Brij B Gupta (2020). «Security, privacy and trust of different layers in Internet-of-Things (IoTs) framework». Em: *Future generation computer systems* 108, pp. 909–920.

Trammell, Brian, Arno Wagner e Benoît Claise (set. de 2013). *Flow Aggregation for the IP Flow Information Export (IPFIX) Protocol*. RFC 7015. DOI: 10.17487/RFC7015. URL: https://www.rfc-editor.org/info/rfc7015.

Tranalyzer - About (s.d.). https://tranalyzer.com/. (Accessed on 03/19/2023).

Vaccari, Ivan et al. (2020). «MQTTset, a new dataset for machine learning techniques on MQTT». Em: *Sensors* 20.22, p. 6578.

Verbraeken, Joost et al. (2020). «A survey on distributed machine learning». Em: *Acm computing surveys (csur)* 53.2, pp. 1–33.

Wireshark - Display Filter Reference (2023). https://www.wireshark.org/docs/dfref/. (Accessed on 01/25/2023).

Xiao, Liang et al. (2018). «IoT security techniques based on machine learning: How do IoT devices use AI to enhance security?» Em: *IEEE Signal Processing Magazine* 35.5, pp. 41–49.

Xin, Yang et al. (2018). «Machine learning and deep learning methods for cybersecurity». Em: *Ieee access* 6, pp. 35365–35381.

DECLARAÇÃO

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Inteligência artificial aplicada à deteção de incidentes de segurança em redes IoT"*, é original e foi realizado por Tiézer Costa de Melo (2200175) sob orientação de Professor Doutor Carlos Manuel da Silva Rabadão (carlos.rabadao@ipleiria.pt), de Professor Doutor Leonel Filipe Simões Santos (leonel.santos@ipleiria.pt) e de Professor Doutor Rogério Luís de Carvalho Costa (rogerio.l.costa@ipleiria.pt).

Leiria, Março de 2023

Tiézer Costa de Melo