INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores

Integração de sistema de deteção de intrusões numa *gateway* de comunicações para a ferrovia

FILIPE FIDALGO TORRES DA COSTA
(Licenciado)

Trabalho de Projeto de natureza científica para obtenção do grau de Mestre em Engenharia de Eletrónica e Telecomunicações

Orientadores:
Professor Doutor Nuno Cruz
Professor Doutor José Simão

Júri:
Presidente:
Professor Doutor Nuno Cota
Vogais:
Professor Doutor Nuno Datia
Professor Doutor José Simão

Janeiro 2022

Agradecimentos

Antes de mais quero demonstrar o meu profundo agradecimento aos meus dois orientadores, Professor Doutor Nuno Cruz e ao Professor Doutor José Simão, que desde o início até ao final do desenvolvimento deste trabalho, deram todo o apoio, disponibilidade, motivação e conhecimento de forma a ultrapassar os desafios encontrados.

Agradeço de uma forma especial ao meu grande amigo Gonçalo Lopes, um "crânio" em assuntos relativos à Inteligência Artificial, sem ele o meu percurso teria sido bastante mais demorado e trabalhoso. Obrigado pelo teu tempo disponibilizado, conhecimento e paciência.

À minha namorada e melhor amiga por ter paciência para mim todos os dias, por me ter auxiliado durante todo o decorrer e na revisão do documento.

Aos meus melhores amigos por me terem distraído de vez em quando do trabalho de forma a clarificar as ideias e os objetivos.

Um agradecimento muito importante aos meus pais e familiares por proporcionarem todas as condições e por todo o apoio incondicional ao longo dos anos.

Por fim, aos meus professores que se cruzaram comigo ao longo do meu percurso académico e que partilharam o seu conhecimento tornando possível a realização desta tese.

Resumo

Atualmente os sistemas ferroviários estão a atravessar uma transformação na forma como as comunicações móveis são utilizadas nas diferentes aplicações relativas à ferrovia. Deste modo, estão a ser usados novos protocolos de comunicações, tais como o 5G, 4G e *Low Power Wide Area Network* (LPWAN) para dar suporte às exigências das aplicações atuais e futuras. Contudo, estes novos protocolos trazem consigo não só melhores capacidades de débito binário e maior robustez na comunicação, como também novas vulnerabilidades nunca antes consideradas nos sistemas ferroviários. Estando o sistema mais vulnerável a novas ameaças, é essencial uma rápida deteção das intrusões e posteriormente a sua notificação.

Esta tese propõe como solução a utilização de um *Intrusion Detection System* (IDS) por cada *gateway* instalada numa rede LoRaWAN. A análise do comportamento dos dispositivos na rede e a deteção das intrusões é realizada com recurso a algoritmos de aprendizagem automática. Para tal, recorreu-se ao IDS Suricata (*open-source*) sobre o qual se desenvolveram programas, através das linguagens de programação Lua e Python, para aceder a uma base de dados com o histórico dos dispositivos e com os modelos de classificação previamente calculados. Como algoritmo de aprendizagem automática, optou-se pelo *K-Nearest Neighbors* (KNN) devido à sua simplicidade, eficiência e tendo em conta o tipo de varáveis em questão. Como base de dados, utilizou-se o CrateDB, baseado em SQL, relacional, escalável, adequado para aprendizagem automática e para dados baseados no tempo. Foram utilizados dados de tráfego de sensores localizados na cidade de Lisboa para a criação, aprendizagem e validação dos modelos. Os resultados mostram que foi possível recorrer ao KNN para classificar o comportamento dos dispositivos, baseando-se no histórico das amostras, com uma precisão e exatidão superior a 90%.

Palavras-chave: Aprendizagem Automática, FRMCS, IDS, IoT, KNN, LoRaWAN

Abstract

Nowadays, railway systems are undergoing a mobile communication transformation on how to use for different type of applications. Thus, new communications protocols, such as 5G, 4G and *Low Power Wide Area Network* (LPWAN), are being used to support the requirements of current and future applications.
However, these new protocols bring with them not only better throughput and more robustness in communication, but also new vulnerabilities never before analyzed in the railway systems.
Being the system more vulnerable to new threats, rapid detection of intrusions and subsequent notification is essential.

This thesis proposes as a solution the use of an Intrusion Detection System (IDS) for each gateway installed in a LoRaWAN network. The analysis of the behavior of devices on the network and the detection of intrusions is performed using machine-learning algorithms. To this end, we used the Suricata IDS (open-source) on which programs were developed, through the Lua and Python programming languages, to access a database with the history of the devices and execute the classification model. As a machine-learning algorithm, K-Nearest Neighbors (KNN) was chosen due to its simplicity, efficiency and taking into account the type of variables in question. As a database, CrateDB was used because it is SQL-based, relational, scalable, suitable for machine learning and time-based data. Traffic data from sensors located in the city of Lisbon were used to create, learn, and validate the models. The results show that it was possible to use KNN to classify the behavior of devices, based on the sample history, with a precision and accuracy of over 90%.

Keywords: Behavioral analysis, FRMCS, IDS, IoT, KNN, Supervised learning, UIC

Glossário

3G/4G/5G	3rd/4th/5th Generation
3GPP	3rd Generation Partnership Project
6LoWPAN	IPv6 Low power Wireless Personal Area Networks
ABP	Activation By Personalization
AIDS	Anomaly Intrusion Detection System
AMQP	Advanced Message Queuing Protocol
ANN	Artificial Neural Network
AS	Application Server
BLOB	Binary Large Object
CCTV	Closed Circuit Television
CIDS	Collaborative Intrusion Detection System
CoAP	Constrained Application Protocol
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
DNS	Domain Name System
DoS	Denial of Service
DT	Decision Tree
ED	End Device
EPLRS	Enhanced Position Location Reporting System
ETSI	European Telecommunications Standards Institute
FRMCS	Future Railway Mobile Communication System
GSM	Global System for Mobile Communications
GSM-R	Global System for Mobile Communications - Railway
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IP	Internet Protocol
IPS	Intrusion Prevention System
IoT	Internet of Things
JS	Join Server
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbours

KLD	Kullback Leibler Divergence
LPWAN	Low Power Wide Area Network
LSNR	LoRa Signal Noise Ratio
LSTM	Long Short Term Memory
LTE	Long Term Evolution
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
NB-IoT	Narrowband IoT
NS	Network Server
OSI	Open System Interconnection
OISF	Open Information Security Foundation
OTAA	Over The Air Activation
RAM	Random Access Memory
RF	Radio Frequency
RNN	Recurrent Neural Network
RSSI	Received signal strength indication
RTSP	Real Time Streaming Protocol
SIDS	Signature Intrusion Detection System
SIP	Session Initiation Protocol
SSH	Secure Shell
SVM	Support Vector Machine
SMB	Samba
TETRA	Terrestrial Trunked Radio
TLS	Transport Layer Security
UDP	User Datagram Protocol
UIC	International Union of Railway

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

O presente capítulo apresenta uma contextualização do projeto a desenvolver, as motivações que levaram ao desenvolvimento do mesmo, os seus objetivos e, por fim, a organização geral deste documento.

1.1. Enquadramento e Motivação

Os sistemas de comunicações móveis ferroviários estão em mudança, com o objetivo de acompanharem as evoluções tecnológicas, principalmente na área de *Internet of Things* (IoT). Desta forma, criaram-se várias iniciativas no sentido de desenvolver novas normas de comunicação, nomeadamente integrando serviços de IoT, como o caso do *Future Railway Mobile Communication System* (FRMCS). No entanto, com a utilização massiva das redes públicas para as comunicações ferroviárias, surgem consequentemente novas ameaças, nomeadamente ciberataques.

É essencial, para além da implementação de mecanismos de segurança no perímetro da rede, a deteção dos ataques o mais rápido possível, distinguindo o tráfego considerado normal de uma intrusão e alertando a infraestrutura quando necessário.

Dos vários protocolos de comunicações existentes em IoT para LPWAN, como o caso de NB-IoT, LTE-M, SigFox e Zigbee, optou-se por usar neste projeto o protocolo LoRaWAN. Este tem como principais vantagens o seu longo alcance (mais do que 15 km em determinados cenários), baixo consumo nos dispositivos (podendo atingir os 10 anos de bateria), alta capacidade de rede (até 1 milhão de dispositivos) e a utilização de espetro que não necessita de licenciamento. Estas vantagens fazem com que seja um protocolo bastante robusto, amplamente utilizado e em constante crescimento [1].

Tendo em consideração que o LoRaWAN é um dos protocolos mais utilizados para a ligação dos dispositivos IoT com a rede para longas distâncias [2], faz com que seja mais atrativo a exploração de vulnerabilidades, levando posteriormente a ataques indesejados ao sistema.

1.2. Objetivos

O objetivo principal do trabalho assenta na definição e implementação de uma solução de IDS para utilização em contexto ferroviário com recurso à aprendizagem automática. No entanto, os conceitos abordados permitem que seja também implementado, com as devidas alterações, noutros sistemas que utilizem dispositivos numa rede IP.

Neste contexto, pretende-se que o IDS analise o tráfego na rede de forma passiva, avalie o comportamento normal dos dispositivos e, caso seja identificada uma intrusão, esta seja notificada o mais celeremente possível, não adicionando qualquer latência às comunicações existentes.

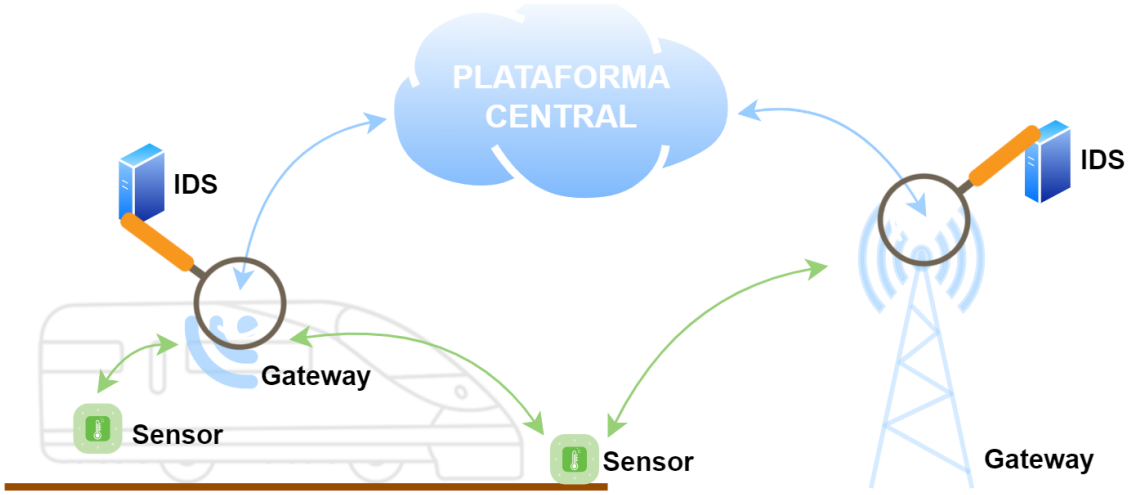

Figura 1 - Arquitetura simplificada

A Figura 1 representa de forma simplificada o cenário onde este projeto incide, o qual é composto por sensores que podem estar instalados num comboio ou numa infraestrutura, associados a uma ou mais *gateways*. Estas são responsáveis por realizar a ponte entre os sensores e a central. Agregado a cada *gateway* encontra-se um IDS.

O sistema tem de ser capaz de definir um modelo para cada dispositivo, a partir de algoritmos de classificação. Não tendo acesso ao *payload* das mensagens, visto que se encontram encriptadas, é necessário recorrer a outros mecanismos para perceber o comportamento da rede.

Para testar a validade da solução proposta, recorreu-se a um conjunto de dados recolhidos através de uma *gateway* LoRaWAN instalada numa das Torres das Amoreiras, propriedade do ISEL e ligada à rede *The Things Network* (TTN). A *gateway* recebe mensalmente uma média de 500 000 mensagens de milhares de dispositivos.

Parte deste trabalho foi objeto de comunicação no INForum 2021, na sessão de Segurança de Sistemas de Computadores e Comunicações, com o título: “Intrusion Detection in LoRaWAN”, e os autores Filipe Costa, Nuno Cruz e José Simão.

1.3. Organização do documento

Este documento encontra-se organizado em seis capítulos. No capítulo "Introdução", é realizada uma introdução ao projeto a ser desenvolvido e implementado, apresenta-se o enquadramento e a motivação para a sua realização e os principais objetivos a serem alcançados.

No capítulo "Conceitos e estado de arte" é feita uma descrição dos principais conceitos abordados ao longo deste documento de forma a facilitar a contextualização do leitor e o estado de arte, nomeadamente sobre FRMCS, IDS, LoRaWAN e modelos de classificação.

No capítulo "Arquitetura" é abordada a arquitetura proposta e tida em conta para o trabalho, seguida dos casos de uso e dos parâmetros das comunicações que se recorreram para caracterizar o comportamento normal da rede e uma análise aos dados disponíveis de uma *gateway* LoRaWAN.

O quarto capítulo, "Implementação", aborda a implementação do IDS com aprendizagem automática, justificando a escolha em relação ao IDS, à base de dados e ao algoritmo de classificação. É também apresentada a máquina de estados, os fluxos de dados existentes e a estrutura de dados utilizada.

No capítulo "Resultados", são apresentados os resultados obtidos a partir dos dados existentes de um dos dispositivos ligados à *gateway* e é realizada uma análise dos mesmos.

Por fim, as conclusões são abordadas no sexto e último capítulo, "Conclusões", comparando as amostras do dispositivo com a classificação dada pelo sistema a cada amostra.

O código-fonte encontra-se disponibilizado online[1] na plataforma GitHub.

[1] https://github.com/phil155/LoRaWAN_IDS

2. Conceitos e estado de arte

Neste capítulo apresenta-se uma descrição dos principais conceitos relacionados com o trabalho a desenvolver, o contexto e o enquadramento tecnológico em que está inserido.

2.1. FRMCS

Em 1994, a *International Union of Railways* (UIC) selecionou a tecnologia *Global System for Mobile Communications* (GSM), mais conhecida como 2G, como suporte para a primeira transformação digital nos sistemas de comunicações nas ferrovias. Rapidamente a organização ETSI/3GPP constatou que os sistemas ferroviários têm necessidades mais críticas e que era necessária uma nova especificação [3]. Desta forma surgiu o *GSM-Railways* (GSM-R) como resposta, garantindo um desempenho sem perdas nas comunicações para velocidades até 500 km/h [4].

A primeira implementação operacional do GSM-R foi realizada em 1999 e adotada a nível global em 2004. Tal como esperado, esta especificação tem sido um sucesso não só na Europa, onde mais de 100 000 km de linhas ferroviárias utilizam a tecnologia, como globalmente, permitindo uma constante evolução da tecnologia nos vários continentes. Adicionalmente, a sua vasta utilização permite uma correção e diminuição das vulnerabilidades do sistema [3].

Contudo, em 2012, a UIC percebeu que era necessário passar para uma nova especificação de forma a acompanhar as evoluções tecnológicas presentes nas telecomunicações, como a utilização de novas gerações de comunicações (4G, 5G). Por conseguinte, começou a realizar estudos para o sucessor do GSM-R, o FRMCS. Este novo sistema pensado pela UIC tem como principal propósito fornecer padrões de forma a permitir flexibilidade na sua implementação, mas ainda assim garantir um sistema robusto, de baixo custo, adequado para cada finalidade e com a capacidade de interoperatividade entre redes [5].

Atualmente, o FRMCS tem presente vários documentos que se encontram em constante atualização, nomeadamente as especificações dos requisitos de utilizador esperadas, diagramas da arquitetura, casos de uso, avaliação de possíveis variações do processo de migração para o FRMCS, entre outros.

2.2. Intrusion Detection System

Um IDS é um dispositivo ou *software* que está em constante monitorização da rede ou sistema, analisando assim todos os pacotes que lhe chegam, à procura de qualquer intrusão ou violação das politicas de segurança associadas [6]. O IDS é passivo nas comunicações do sistema, ou seja, não adiciona qualquer latência com a sua análise dos pacotes. Contudo, não tem a capacidade de intervir diretamente no sistema de forma a prevenir ataques.

O dispositivo capaz de realizar a função de prevenção é o *Intrusion Prevention System* (IPS). Este é inserido nas comunicações entre uma determinada origem e destino. Ao estar fisicamente presente no percurso das comunicações, vai, por um lado, adicionar mais latência a estas, mas, em contrapartida, vai ser capaz de terminá-las. Este necessita de um maior cuidado na sua análise do tráfego porque podem vir a existir muitos falsos positivos e comunicações legitimas poderão vir a ser interpretadas como intrusões.

Existem duas técnicas principais utilizadas para a deteção de intrusões:

- *Signature IDS* (SIDS)
- *Anomaly IDS* (AIDS)

A primeira é a forma mais simples e é realizada baseando-se em assinaturas conhecidas dos ataques. Caso um pacote tenha uma parte do seu conteúdo da mesma forma que a assinatura, é gerado um alerta [7]. De forma a ser percetível o conceito de assinatura num IDS, a Figura 2 apresenta um exemplo de uma assinatura simples:

```
alert icmp $HOME_NET any -> $HOME_NET any (msg:"PING ICMP"; itype:8;   \
    classtype:not-suspicious; sid:1; rev:1;)
```

Figura 2 - Estutura de uma assinatura

Na assinatura apresentada, um pacote ICMP que for enviado com o endereço de origem e destino presente na lista `$HOME-NET`, com qualquer porto de origem e destino, e com o *icmp-type* = 8 (*Echo*) (ou seja, quando se realiza um *ping* para um dispositivo), o IDS vai gerar um alerta para o sistema com a mensagem: “PING ICMP”. No Capítulo 4.4 irá ser realizada uma análise mais pormenorizada das assinaturas.

A utilização de SIDS faz com que seja bastante difícil a deteção de ataques denominados como “zero-day”, ou seja ataques ainda não são conhecidos pela base de dados de assinaturas [7].

A segunda técnica (AIDS) é baseada não só em assinaturas como na análise comportamental dos dispositivos no sistema, ou seja, guarda numa base de dados o comportamento ao longo do tempo e caso este se altere é gerado um alerta. Desta maneira é possível detetar novos ataques, mesmo que estes não estejam ainda identificados com uma assinatura. Consequentemente, podem vir a ser gerados muitos falsos positivos se o IDS não estiver devidamente dimensionado [7].

2.3. LoRaWAN

LoRaWAN é um protocolo concebido para redes de longa distância. Permite que dispositivos de baixa potência comuniquem com aplicações na Internet [8], especifica o formato dos pacotes e define a forma como a rede tem de processar os pacotes. São identificadas três classes de dispositivos, cada uma apropriada para o tipo de dados a enviar.

A Figura 3 ilustra as camadas presentes do protocolo:

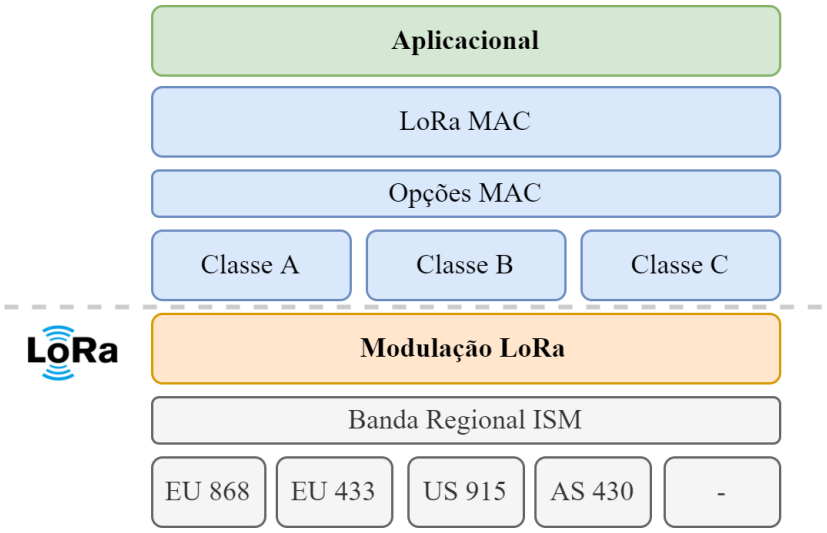

Figura 3 - Camadas do LoRaWAN

A arquitetura LoRaWAN tem na sua camada física o protocolo LoRa, o qual opera em bandas de frequência não licenciadas, o que faz da sua utilização, em termos monetários, algo mais em conta em comparação com protocolos que tiram proveito de bandas licenciadas. No caso da União Europeia é possível utilizar as frequências 868 e 433 MHz, sendo a primeira a mais utilizada.

2.3.1. Arquitetura genérica

A rede de LoRaWAN utiliza uma topologia em estrela, apresentada na Figura 4, onde os dispositivos IoT, também conhecidos como equipamentos finais, *End-Devices* (ED), estão conectados via LoRa com uma ou mais *gateways* LoRa. Os dispositivos podem enviar dados para qualquer uma das *gateways* da mesma rede e esta vai tratar de encaminhar a informação para o servidor correto.

As *gateways* LoRa são transparentes para os EDs, que são responsáveis por realizar o *packet forwarding* (processo de passagem de LoRa para IP), e estão interligadas com o servidor de rede, *Network Server* (NS).

Por sua vez, o NS está ligado ao servidor aplicacional, *Application Server* (AS), encaminhando as mensagens dos EDs para a aplicação correta. No interior do NS encontra-se, por fim, um *Join Server* (JS) e este é responsável pela autenticação dos EDs na rede.

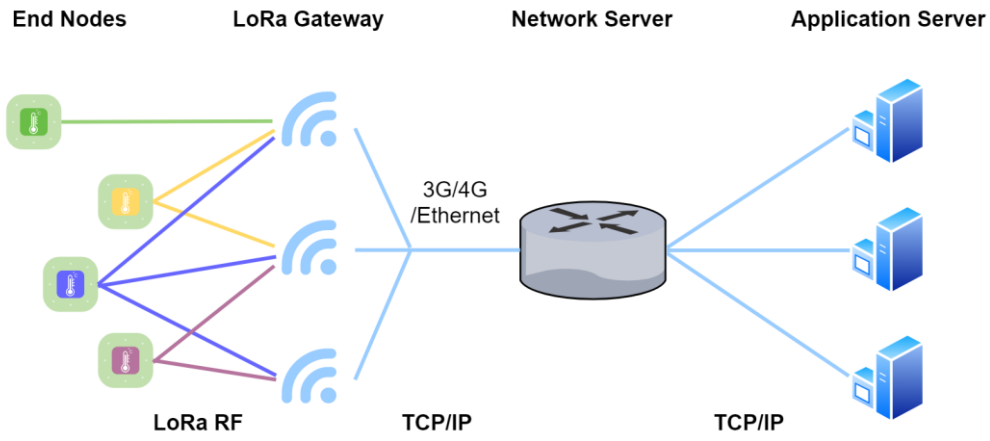

Figura 4 - Arquitetura LoRaWAN

2.3.2. Campos das mensagens

As mensagens transmitidas no domínio LoRaWAN podem ser de dois tipos, ou de transmissão (no sentido NS para *gateway*) ou de receção (no sentido *gateway* para NS).

No caso de uma mensagem enviada da *gateway* para o NS, esta tem como designação "PUSH-DATA". Para além de outros parâmetros, o seu *payload* vai em JSON e tem o seguinte formato:

```
{
    "rxpk":[ {"ED1"}, {"ED2"}, ... ]
}
```

Figura 5 - Objeto JSON raiz

Conforme se verifica pela Figura 5, o objeto JSON raiz é composto por um *array* com o nome: "rxpk" e este pode conter um ou mais objetos JSON. Cada objeto contêm um pacote RF recebido do lado do LoRa.

```
{"rxpk":[
       {
              "time":"2013-03-31T16:21:17.528002Z",
              "tmst":3512348611,
              "chan":2,
              "rfch":0,
              "freq":866.349812,
              "stat":1,
              "modu":"LORA",
              "datr":"SF7BW125",
              "codr":"4/6",
              "rssi":-35,
              "lsnr":5.1,
              "size":32,
              "data":"40F17DBE4900020001954378762B11FF0D"
       }
]}
```

Figura 6 - Exemplo de um array JSON da Semtech

Dos vários campos que são visíveis pela Figura 6, são de salientar:

- "**time**": Representa a data e hora que a *gateway* recebeu o pacote;
- *Received Signal Strength Indicator* (**RSSI**): Corresponde ao nível de sinal recebido em dBm, tem 1dB de precisão [9] e recomenda-se que o valor mínimo seja -120 dBm [10];
- *LoRa Signal Noise Ratio* (**LSNR**): Simboliza a relação sinal-ruído em dB, com 0,1 dB de precisão e tipicamente os valores variam entre -20 dB e 10 dB [10];
- "**datr**": Representa o *Spreading Factor* (SF), este é um valor inteiro e varia entre 7 a 12, e o *Bandwidth* (BW), apenas pode ser 125, 250 ou 500 kHz;
- "**freq**": Indica a frequência pela qual a informação foi enviada e o "**chan**" o número do canal associado;
- "**data**": Contêm os dados recebidos pelo sensor e pode ser codificado em formato hexadecimal ou em base64 [9].

O conteúdo do campo "**data**", de uma mensagem "**rxpk**", encontra-se definido no protocolo LoRa com o formato apresentado na Figura 7:

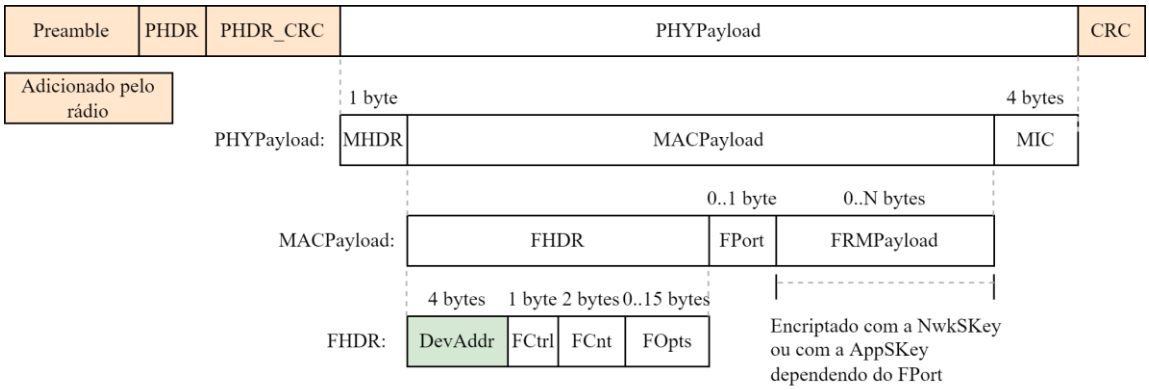

Figura 7 - Formato dos dados LoRa

Os campos selecionados com a cor de laranja são adicionados pelo módulo de rádio LoRa de forma a garantir uma comunicação entre o emissor e o recetor e não são abordados neste âmbito. O campo "PHYPayload" é colocado no interior de um pacote IP e este é passível de ser analisado numa rede IP.

Tendo em conta que se pretende implementar um IDS sem que este tenha qualquer conhecimento das chaves de sessões da rede em que vai estar inserido, é necessário encontrar identificadores que permitam distinguir os vários dispositivos. Logo, o campo "FRMPayload" não é possível de se aceder ao conteúdo porque se encontra encriptado.

2.3.3. Segurança

A Figura 8 representa os principais elementos da especificação de segurança do protocolo LoRaWAN.

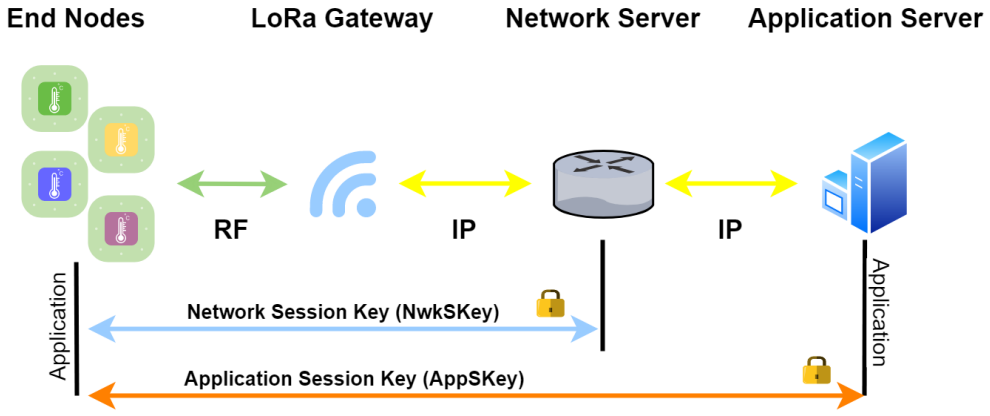

Figura 8 - Segurança LoRaWAN

Existem duas camadas de segurança. A primeira diz respeito à camada de rede, que começa no ED e termina no NS, e a segunda camada que corresponde à camada aplicacional que se inicia no ED e termina no AS.

Um dispositivo é identificado unicamente pelo DevEUI, este com 64-bit, criado pelo fabricante e imutável. Ao contrário do DevEUI, que identifica o dispositivo globalmente, o DevAddr de 32-bit, identifica o dispositivo numa mesma rede. Este não é único e vários dispositivos podem ter o mesmo endereço [11].

Os dispositivos para poderem enviar dados na rede têm de se autenticar previamente comunicando com o JS. Atualmente são permitidas duas técnicas de autenticação:

- *Over The Air Activation* (**OTAA**):
    - A ativação do dispositivo é realizada no ar, ou seja, quando o dispositivo se liga e inicia a comunicação é realizado, em primeiro lugar, um *join-procedure*. Este modo é a forma mais segura de ativação e entrada na rede visto que são sempre geradas novas sessões de segurança;
    - A rede inicia uma nova sessão de segurança com uma nova *NetworkSessionKey* (NwSKey) e um novo *Device Address* (DevAddr).

- *Activation By Personalization* (**ABP**):
    - Neste caso, não é necessário realizar o *join-procedure* com a rede, visto que foi previamente definido um DevAddr e um NwSKey, em ambos os pontos da comunicação, e mantêm-se durante a sua existência;
    - Este modo simplifica o processo de autenticação, mas reduz significativamente a segurança, visto que não são renovadas as sessões e consequentemente as chaves.

Em ambos é necessário definir previamente uma *Application Key* (AppKey). No caso mais simples de autenticação na rede, o ABP, o valor do DevAddr mantém-se inalterado durante a existência do dispositivo, até que o administrador do dispositivo o altere. Desta forma, é possível identificar o dispositivo na rede apenas com o DevAddr e foi esta a primeira forma utilizada.

2.4. Inteligência Artificial

O conceito de inteligência artificial (*Artificial Intelligence* - AI) foi formalmente apresentado numa conferência em Hanover, Alemanha no ano de 1956 [12], no entanto, o trabalho anteriormente realizado por Alan Turing, foi essencial para se perceber de forma abstrata os princípios por detrás da programação [13].

Categorizou-se a inteligência artificial de duas formas, Forte AI e Fraca AI [14]:

- Forte AI: Esta é programada de forma a ter capacidade de replicar habilidades cognitivas presentes no ser humano. Quando colocada à prova numa tarefa não conhecida, esta tem a capacidade de encontrar uma solução de acordo com determinados algoritmos. São estas que na teoria, conseguem passar no teste de *Turing* e no da sala chinesa;
- Fraca AI: É um sistema desenvolvido e treinado para executar uma determinada tarefa previamente definida. Robôs industriais, sistemas de reconhecimento facial, assistentes virtuais pessoais como o caso da "Siri" da Apple, são alguns exemplos.

O sistema desenvolvido para este trabalho está inserido na última categoria apresentada anteriormente, o código e o algoritmo escolhido foram definidos apenas para executar uma determinada ação.

Para além das duas grandes categorias de AI, estas podem ainda ser diferenciadas em 4 tipos. As do primeiro tipo são conhecidas também como máquinas reativas, não utilizam qualquer memória para uma determinada tarefa. As do segundo tipo já recorrem à memória de forma a guardar experiências passadas para informar decisões futuras. O trabalho encontra-se neste caso em particular. Os restantes tipos dizem respeito à compreensão das emoções e à capacidade da máquina ter consciência [15].

A aprendizagem automática é um estado em que a máquina atua independente da programação e recorre a algoritmos para tomar uma decisão. Existem três tipos de algoritmos:

- Aprendizagem supervisionada: A amostra de dados está classificada previamente e a máquina aprende a distinção entre os dados, conseguindo classificar dados desconhecidos;
- Aprendizagem não supervisionada: A amostra de dados não se encontra previamente classificada e a máquina é responsável por procurar padrões, semelhanças e diferenças entre eles;

- Aprendizagem reforçada: A amostra de dados também não se encontra classificada, mas após um conjunto de ações o sistema começa a responder. O sistema é baseado na recompensa e adequa as suas ações de forma a melhorar a recompensa [16].

Para que seja possível perceber qual o melhor algoritmo a utilizar em determinadas tarefas ou situações é necessário recorrer a medidas de desempenho, desta forma existem essencialmente quatro, a exatidão (*accuracy*), precisão (*precision*), sensibilidade (*recall*) e a pontuação F1 [17]. Para se compreender as medidas é essencial entender quatro parâmetros:

- Verdadeiro-Positivos (*True-Positives* – TP): Significa que a classificação indicada na amostra de dados é positiva e o sistema classificou como positiva;
- Verdadeiro-Negativos (*True-Negatives* – TN): Significa o oposto do TP, ou seja, que a classificação indicada na amostra de dados é negativa e o sistema classificou como negativa;
- Falsos-Positivos (*False-Positives* – FP): Significa que a classificação indicada na amostra de dados é negativa e o sistema classificou como positiva;
- Falsos-Negativos (*False-Negatives* – FN): Significa que a classificação indicada na amostra de dados é positiva e o sistema classificou como negativa.

Desta forma, a exatidão (*accuracy*) é o rácio das classificações corretamente observadas pelo sistema em relação ao total das observações.

$$accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{1}$$

A precisão (*precision*) é o rácio das classificações positivas corretamente observadas pelo sistema em relação ao total das observações positivas (verdadeiras e falsas).

$$precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{2}$$

A sensibilidade (*recall*) é o rácio das classificações positivas corretamente observadas pelo sistema em relação a todas as amostras positivas. Ou seja, permite saber das amostras que realmente são positivas, quantas é que foram classificadas pelo sistema.

$$recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3}$$

A pontuação F1 (F1 *score*) é uma média ponderada da precisão com a sensibilidade, tendo em conta ambos os parâmetros positivos e negativos.

$$\mathrm{F1}\ score = \frac{2 \times (recall \times precision)}{recall + precision} \tag{4}$$

2.5. Estado de arte

Sendo o FRMCS uma iniciativa relativamente recente, as especificações da sua arquitetura ainda se encontram em desenvolvimento, nomeadamente os tipos de serviços e aplicações que vão ser necessários para que a infraestrutura funcione e seja capaz de substituir o sistema até agora implementado.

Contudo, esta iniciativa define que é pretendido uma utilização massiva de dispositivos IoT a fim de medir e atuar ao longo da infraestrutura. Consequentemente, esta utilização faz com que grande parte das vulnerabilidades existentes no IoT sejam transpostas para os sistemas ferroviários.

O autor *Shafiullah,* em 2007, analisou a importância das comunicações sem-fios na monitorização e manutenção das linhas ferroviárias, evidenciando de uma forma inicial as necessidades que foram crescendo ao longo do tempo com o objetivo de garantir o correto funcionamento da infraestrutura ferroviária. Comparou os vários protocolos de comunicação existentes para suportar a monitorização, nomeadamente o GSM-R, o *Terrestrial Trunked Radio* (TETRA) usado nos *walkie talkies*, o *Enhanced Position Location Reporting System* (EPLRS), satélite, entre outros. Concluiu, de uma forma generalizada, que a saúde de um comboio pode ser analisada de quatro formas: pelas suas vibrações, aceleração, temperatura e humidade. A análise destas variáveis possibilita a prevenção de possíveis acidentes futuros [18].

Um estudo realizado em 2014 [19], aborda os vários tipos de IDS utilizados em redes de computadores e as diferentes abordagens existentes para os algoritmos de inteligência artificial. A metodologia sugerida para a utilização de um IDS com inteligência artificial passa primeiro pela realização de um pré-processamento dos dados recebidos e de seguida pela aprendizagem do modelo. Posteriormente este mesmo modelo é utilizado para testar e avaliar se os dados recebidos são característicos de uma intrusão. No estudo são abordados os principais algoritmos, tais como, *Decision Tree* (DT), KNN, *Support Vector Machine* (SVM), *Artificial Neural Network* (ANN) e *Recurrent Neural Network* (RNN). Observou-se que a maioria dos IDS existentes tiram partido de inteligência artificial com algoritmos de *deep learning* para identificar possíveis intrusões e prever o funcionamento da rede. Adicionalmente, concluiu-se que quanto maior o treino do modelo, maior a sua eficiência para detetar intrusões.

Trabalhos anteriores analisados no artigo [20], revelam a importância da introdução de um IDS para a deteção de intrusões nos dispositivos IoT no protocolo *IPv6 Low power Wireless Personal Area Networks* (6LoWPAN). A solução para detetar ataques como *Denial of Service* (DoS), *Distributed* DoS (DDoS), ataques de repetição, pode passar pela introdução de um IDS na rede. Verificou-se que, nestes tipos de ambientes, os atacantes podem tentar obter acesso a dispositivos nos quais não tem autorização. Os IDS tradicionais utilizados em redes não estão dimensionados para a complexidade do IoT. Os últimos desenvolvimentos indicam que é necessário incorporar

mecanismos de inteligência artificial, de forma a ter perceção do comportamento do sistema, detetar novas formas de ataque e recorrer a IDS colaborativos.

No artigo "Intrusion Detection System for the Internet of Things Based on Blockchain and Multi-Agent Systems" os autores investigaram a possibilidade de introduzir inteligência artificial com tecnologia *blockchain* de forma a criar um IDS colaborativo. Recorreram a redes neuronais, *Deep Neural Networks* (DNN), para analisar o comportamento da rede e de determinados protocolos, nomeadamente ICMP, UDP e TCP. O sistema consegue detetar intrusões em 0,18s, em média, o que é praticamente em tempo real, com pelo menos uma exatidão de 94% [21].

O autor Kasinathan propôs a utilização de um IDS *open-source*, Suricata [22] para detetar ataques de DoS em redes com o protocolo 6LoWPAN. Para simular o ataque recorreram a uma ferramenta de penetração (Metasploit [23]) e na deteção do ataque utilizaram assinaturas estáticas, ou seja, caso os pacotes tenham uma cadência superior à admitida, o IDS deteta como intrusão. Todavia, o trabalho foca-se apenas na especificação de assinaturas estáticas para a deteção e não recorre a aprendizagem automática [24].

Uma aproximação semelhante foi usada no artigo "Network Intrusion Detection System for Jamming Attack in LoRaWAN Join Procedure", mas para o protocolo LoRaWAN, que demonstrou ser suscetível a ataques de repetição, *jamming*, *wormhole* e *flipping*. O artigo aborda com maior detalhe os ataques de *jamming*, com o intuito de negar aos dispositivos IoT o acesso ao serviço. Neste caso, recorrem a algoritmos matemáticos conhecidos como o *Kullback Leibler Divergence* (KLD) e o *Hamming Distance*. Estes estão a correr num Raspberry Pi 2 e foram programados 3 dispositivos LoRa para realizarem um *Join Request* a cada 30 segundos. Na proximidade de um dos dispositivos, encontra-se um *jammer* que se liga durante uma hora com uma potência constante e impede os dispositivos de transmitir a informação para a *gateway*. O ensaio teve uma duração total de 2:30h e concluiu-se que os modelos tiveram uma taxa de deteção na ordem dos 90% com 5% de falsos positivos [25].

O artigo [26] pretende relacionar a previsão do comportamento de dispositivos LoRaWAN com a utilização de algoritmos de inteligência artificial. Desta forma aplicou-se um algoritmo de classificação, o KNN, para criar grupos de dispositivos que partilham o mesmo comportamento. Estes grupos são, posteriormente, utilizados pelo algoritmo DT ou pelo *Long Short Term Memory* (LSTM) para prever o comportamento. O DT é um algoritmo mais simples em comparação com o LSTM, que recorre a redes neuronais. Os parâmetros usados para a criação do modelo de aprendizagem automática foram o "Devaddr", "FCnt", SF, RSSI, LSNR, comprimento do pacote e o tempo entre pacotes. A amostra de dados contém 372 milhões de pacotes LoRaWAN recolhidos ao longo do ano de 2019 em Itália. Conseguiu-se prever o tempo entre pacotes com um erro de 3,5% para 77% dos casos.

Por fim, os autores num artigo realizado em 2021 revelam a importância que uma *gateway* tem na deteção de anomalias numa rede com dispositivos IoT [27]. Atualmente, esta tem apenas como

funcionalidade o reencaminhamento e tradução dos pacotes de uma rede IoT para uma rede de pacotes IP. Não obstante, devido à sua posição estratégica, a *gateway* devia de monitorizar as comunicações e o comportamento dos dispositivos ou, pelo menos, disponibilizar meios de conexão para interligar IDS. São definidos 18 requisitos de segurança, dos quais 3 deles, relacionados com a deteção de anomalias, não são abrangidos pela maioria das *gateways* IoT, o que evidencia a falta de segurança associada a estes dispositivos.

O estado de arte permitiu salientar a importância da existência de um mecanismo de segurança capaz de analisar os pacotes existentes numa rede IoT e classificá-los em termos de intrusão. Esta rede tem requisitos bastante específicos e diferentes comparativamente com uma rede de computadores. Tal característica traduz-se numa maior dificuldade de implementação e, consequentemente, numa reduzida existência de sistemas capazes.

3. Arquitetura

Neste capítulo apresentam-se os diferentes casos de uso abordados pelo FRMCS, os casos de estudo abordados por este projeto, a arquitetura geral , as suas características e por fim uma análise exaustiva de amostras de dispositivos reais de uma *gateway* LoRaWAN.

3.1. Casos de Uso na Ferrovia

Para que seja possível a correta implementação de um IDS na ferrovia, é necessário analisar e compreender os principais sistemas de comunicações e os casos de uso associados. Desta forma, e tendo em conta o documento do FRMCS relativo aos casos de uso [28], optou-se pela seguinte estrutura de serviços:

- **Telemetria**:
    - Este serviço está associado maioritariamente à comunicação dos sensores de IoT com os servidores de processamento de dados. É possível utilizar qualquer protocolo amplamente usado em IoT, nomeadamente o LoRaWAN, AMQP, CoAP, entre outros.
    - De acordo com os documentos publicados até agora pela UIC [3], não são indicadas sugestões de protocolos a utilizar.
- **Voz**:
    - Encontram-se associadas comunicações entre o condutor, central, passageiros e funcionários. Sendo como sugestão de protocolo o *Session Initiation Protocol* (SIP), indicado pelo documento "ETSI TS 103 389" [29].
- **Vigilância**:
    - A este serviço, normalmente associado a *Closed Circuit Television* (CCTV), encontram-se as comunicações das câmaras de vigilância com a central. Um dos protocolos mais utilizados em CCTV, ainda que fora do contexto ferroviário, é o *Real Time Streaming Protocol* (RTSP).

3.2. Telemetria

Optou-se por escolher como caso de uso do IDS, a deteção de intrusões relacionadas com Telemetria/IoT. O protocolo de comunicação escolhido foi o LoRaWAN visto que é aquele que se encontra atualmente em maior crescimento de utilização [2]. A Figura 9 representa a arquitetura geral em que o trabalho se encontra inserido.

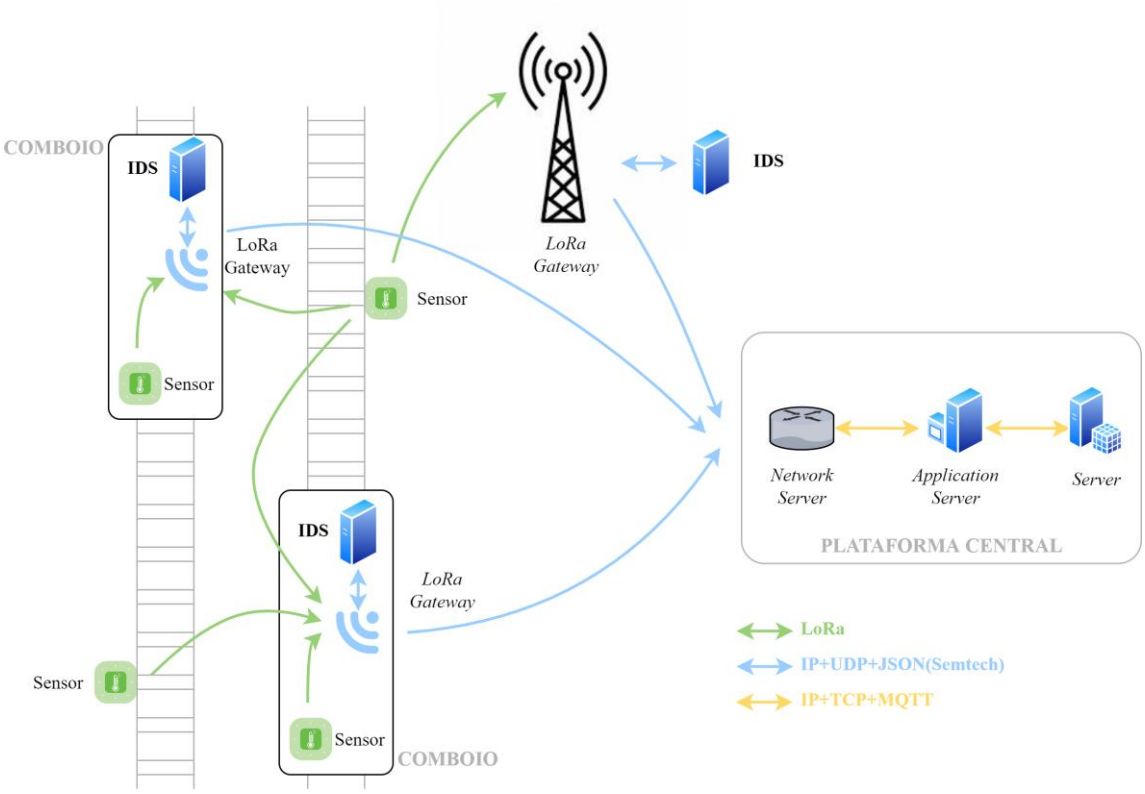

Figura 9 - Arquitetura Geral

Na Figura acima estão representados dois comboios e uma plataforma central onde se encontram os servidores associados ao funcionamento do LoRaWAN. Dentro de cada comboio encontra-se uma ou mais *gateways*, dependendo da dimensão do comboio, um ou mais sensores LoRa e um IDS que analisa o tráfego IP entre a *gateway* e o NS. Na infraestrutura da linha, podem existir um ou mais sensores e uma ou mais *gateways*. As setas a verde representam a comunicação entre o sensor e a *gateway* com recurso ao protocolo LoRa. As setas a azul representam a comunicação lógica entre a *gateway* e o NS, com recurso ao UDP sobre IP, em que o seu conteúdo é enviado em formato JSON. Por fim, as setas a amarelo representam a comunicação dos servidores na central que recorrem ao IP com MQTT que, por sua vez, utiliza TCP. Diferenciou-se as comunicações dos sensores da seguinte forma:

- Comboio → Central:

  Os sensores a bordo comunicam os seus dados via LoRa com a *gateway* presente no comboio. A *gateway* insere depois os dados num pacote IP e envia para o NS.

- Infraestrutura → Central

  Os sensores presentes na infraestrutura da linha podem enviar os dados via LoRa para um comboio que se encontra a passar perto dele ou para uma outra *gateway* que se encontra a fazer cobertura LoRa na zona.

É necessário existir um IDS por cada *gateway* LoRa de forma a garantir que o sistema tem a capacidade de perceber o comportamento dos dispositivos, dependendo da localização, uma vez que o comboio tem a possibilidade de se mover. Adicionalmente, esta decisão permitirá ao sistema desativar determinadas comunicações entre a *gateway* e o NS, caso os modelos criados tenham uma boa precisão.

Tendo em conta que o comboio tem a particularidade de se poder mover e receber mensagens de outros dispositivos para além da distância prevista pelo LoRaWAN, a informação relativa à localização de cada *gateway* é importante porque é desta forma que é possível garantir uma coerência entre o comportamento do dispositivo e a posição deste. No âmbito deste trabalho, analisou-se apenas as mensagens LoRaWAN com origem na *gateway* e término no NS. A Figura 10 ilustra a mensagem e o JSON *payload* tem como campos os que se encontram representados na Figura 5 e Figura 6.

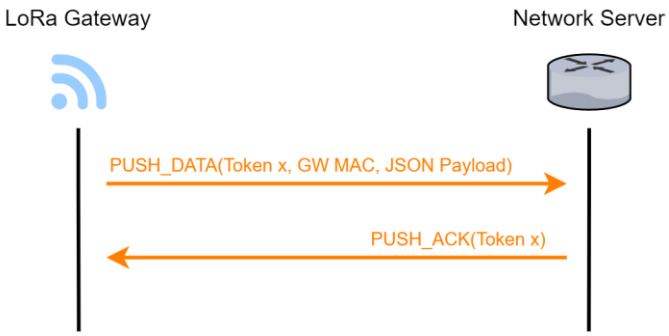

Figura 10 - Mensagem "PUSH_DATA"

A Figura 11 detalha a arquitetura do IDS e o processamento dos pacotes recebidos dos sensores.

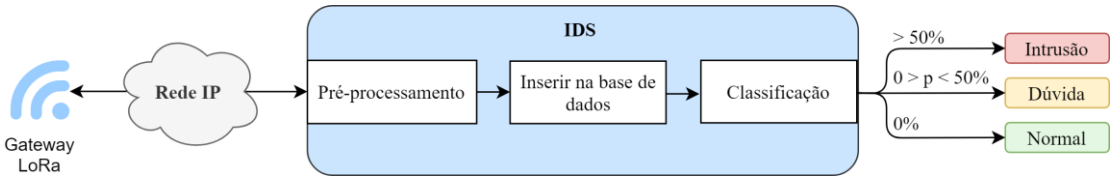

Figura 11 - Arquitetura do IDS

Quando um pacote é enviado pela rede IP, este é analisado pelo IDS e é verificado se este tem como origem a *gateway*. Caso tenha origem na *gateway*, é realizado um pré-processamento de forma a retirar os parâmetros do JSON *payload*, tais como o RSSI, LSNR, SF. Estes são guardados numa base de dados associados ao respetivo "DevAddr". Desta forma o IDS, recorrendo à base de dados, é capaz de executar algoritmos de classificação, identificar o comportamento normal do dispositivo e criar um modelo para posteriormente identificar intrusões.

São possíveis três resultados do IDS: caso a probabilidade seja 0%, o pacote é identificado como comportamento normal; caso esta seja superior a 50%, o pacote é identificado como intrusão; no caso em que a probabilidade está entre 0 e 50%, o pacote é identificado como dúvida.
Neste último caso os pacotes são enviados para uma tabela diferente da base de dados de forma a serem analisados e classificados manualmente por um utilizador com recurso a uma interface gráfica.
Para que seja possível validar o correto funcionamento do IDS é essencial a definição dos vetores de ataque que são abrangidos. Dos vários existentes em LoRaWAN, optou-se por abordar os seguintes:

1. Alteração da localização do dispositivo;
2. Alteração do *payload* das mensagens;
3. Ataques por repetição;
4. Ataques com *jamming*.

O primeiro vetor de ataque tem como principal característica o facto de se realizar ao nível físico. Tendo em conta que os dispositivos em LoRaWAN se encontram maioritariamente em localizações possíveis de serem alcançadas por ação humana, a sua alteração pode provocar falsas medidas e levar a notificações erradas. Num cenário em que os sensores se encontram no exterior a monitorizar os taludes da ferrovia, é possível que um atacante se aproxime dos dispositivos, altere a localização, de forma a notificar o sistema que ocorreu um deslizamento e notificando o sistema que a linha ferroviária provavelmente se encontra inacessível. Isto pode conduzir a um redireccionamento ou paragem do comboio para um local desejado pelo atacante.
O segundo ataque também ocorre essencialmente com recurso ao *hardware* do sensor. Neste caso o atacante tenta aceder ao código que se encontra na memória e altera os parâmetros enviados para a central, notificando eventos que não correspondem à realidade. Por exemplo, caso seja um sensor a reportar a temperatura de um comboio, o atacante pode alterar os dados inseridos no *payload* e desta forma enganar o sistema.
O terceiro vetor de ataque é bastante comum nos protocolos sem fios, sejam eles de curto ou longo alcance. Em LoRaWAN, conforme foi mencionado nos capítulos anteriores, não é possível desencriptar as comunicações sem ter acesso à AppSKey e também não é possível alterar o seu conteúdo sem que o sistema seja notificado devido ao MIC, garantindo assim confidencialidade e integridade. Contudo um atacante pode encontrar-se à escuta de mensagens LoRa e repeti-las de forma a enganar o sistema e atingir o valor máximo do contador de forma a provocar um *reset*. Uma das redes mais conhecidas de LoRaWAN, a *The Things Network* (TTN), fornece uma ferramenta de segurança no NS, designada de *frame counter*, que tem como objetivo prevenir ataques por repetição.

Por último, o ataque de *jamming* tem como principal objetivo negar o serviço de um determinado dispositivo [30], impedindo que as mensagens cheguem com sucesso a uma *gateway* e, por sua vez, a um NS.

3.3. Análise dos dados disponíveis

Com o objetivo de desenvolver um modelo de classificação, associado a cada dispositivo, mais realista, é necessário recorrer a amostras de dados de dispositivos reais. Para tal, utilizou-se um conjunto de dados de uma *gateway* LoRa, que pertence ao ISEL, instalada numa das Torres das Amoreiras, em Lisboa, e ligada à TTN. Esta recebe mensalmente uma média de 500 000 mensagens de milhares de dispositivos. Realizou-se um pré-processamento do conjunto de dados com recurso à biblioteca *ProfileReport* do *Pandas_profiling* [31] de forma a gerar um relatório sucinto dos dados.

A amostra analisada foi relativa ao mês de Junho de 2020, mês este de plena pandemia Covid-19 em Portugal, mas já com algum crescimento de atividade após o primeiro confinamento. Do relatório foi possível concluir que existiram 509 042 mensagens enviadas de 2876 dispositivos. Dos vários parâmetros das mensagens, selecionaram-se os mais relevantes.

A Figura 12 e a Figura 13 ilustram a evolução do LSNR e o RSSI da amostra total:

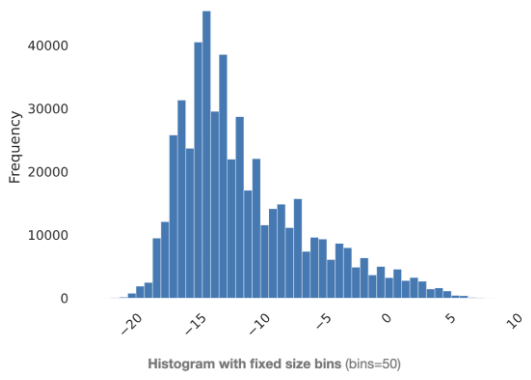

Figura 12 - LSNR da amostra total

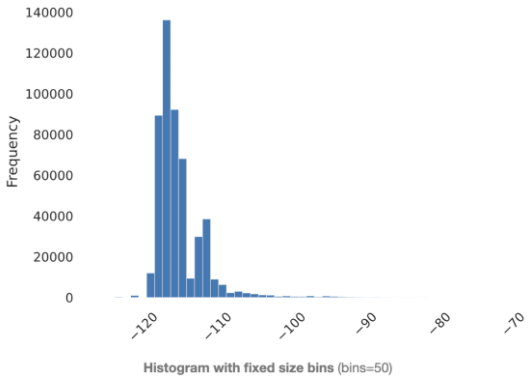

Figura 13 - RSSI da amostra total

Verifica-se que em relação ao LSNR, este varia entre -20 a 6 dB conforme esperado, com maior predominância nos -15 dB. O RSSI varia entre -120 dBm a -90 dBm, destacando-se nos -118 dBm.

Para efeitos de validação do sistema proposto, optou-se por utilizar apenas um dispositivo com o endereço 0000BF53. Este tem a seguinte evolução de LSNR e RSSI:

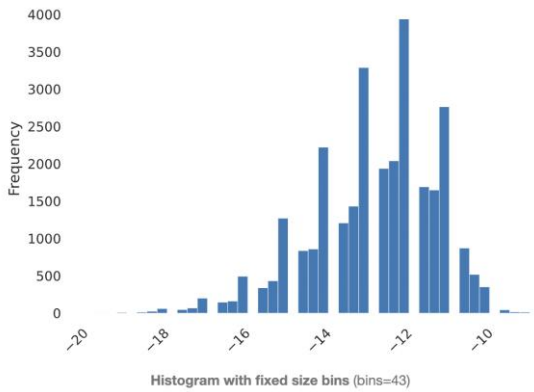

Figura 14 - LSNR do dispositivo: 0000BF53

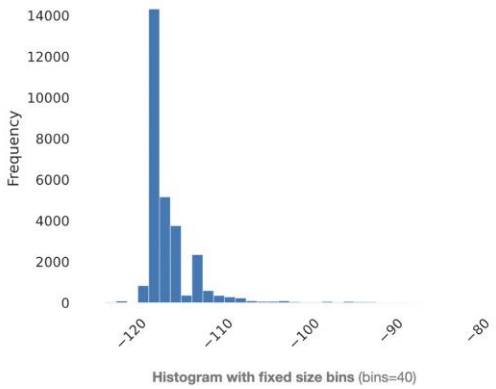

Figura 15 - RSSI do dispositivo: 0000BF53

Pela Figura 14 e Figura 15 verifica-se que o LSNR varia entre -18 dB e -10 dB e o RSSI entre os -120 dBm e os -105 dBm. Estes dois parâmetros, em conjunto com a periodicidade das amostras, permite concluir que o dispositivo se encontra numa posição geográfica fixa durante o mês do estudo.

Relativamente ao dispositivo 0000BF53, encontram-se ilustrados o SF e a BW na Figura 16:

Common Values

Value	Count	Frequency (%)
SF12BW125	28841	99.7%
SF11BW125	49	0.2%
SF10BW125	25	0.1%
SF9BW125	8	< 0.1%
SF8BW125	1	< 0.1%

Figura 16 - SF e BW do dispositivo: 0000BF53

Observa-se que os valores de SF = 12 e BW = 125 são os mais predominantes e muitos outros valores não foram utilizados, o que pode estar relacionado com o facto deste dispositivo ter o *bit* ADR ativo no *Frame Control*. O dispositivo ao indicar que pretende fazer ADR, o NS fica apto para alterar o SF, a BW ou a potência de transmissão do dispositivo, de forma a otimizar o débito binário, o tempo de propagação e o consumo de energia.

Por fim, a biblioteca utilizada permite também averiguar as correlações existentes entre os vários parâmetros, nomeadamente, a correlação de *Pearson*.

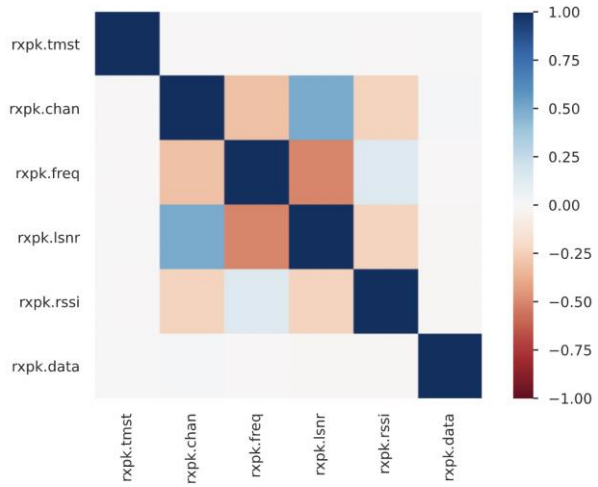

Figura 17 - Correlação de *Pearson*

A correlação de *Pearson* é uma medida estatística amplamente utilizada que relaciona, de forma linear, duas variáveis contínuas [32]. Esta foi utilizada para perceber que variáveis fazem mais sentido colocar no modelo de classificação.

Observa-se pela Figura 17 que os valores de correlação variam entre -1 a 1, sendo que o valor "1" significa que existe uma relação perfeita positiva entre duas variáveis, ou seja, quando uma variável aumenta, a outra também tem o mesmo comportamento, e o valor "-1" significa que existe uma relação inversamente proporcional entre duas variáveis, ou seja quando uma variável aumenta, a outra diminui.

A variável "lsnr" tem uma relação de 0,6 com a variável "chan", ou seja, quando a relação sinal-ruído aumenta, o número do canal também aumenta. Tal não acontece com a variável "freq", que diminui com o aumento da variável "lsnr". Verifica-se também uma relação, embora menor, entre a relação sinal-ruído e o nível de sinal recebido, "rssi".

A variável "tmst", pelo contrário, devido ao facto de ter um valor diferente para cada pacote, de ter correlação máxima com ela própria e não ter com mais nenhuma outra variável, não faz sentido ser utilizada como variável no algoritmo. Não obstante, esta pode ser utilizada para criar outras variáveis com maior relevância, como por exemplo, a periodicidade dos pacotes.

4. Implementação

Neste capítulo é apresentada a implementação de um protótipo para validar a arquitetura apresentada anteriormente, tendo em conta os casos de uso abordados no capítulo anterior. É explicado de forma detalhada a estrutura de rede usada no desenvolvimento do protótipo, o IDS escolhido e o código elaborado para detetar a intrusão em particular.

4.1. Escolha do IDS

A escolha do IDS para a deteção de ataques nas comunicações da ferrovia tem como requisito a utilização de *software open-source*. Assim sendo, dos vários IDS presentes no mercado [33], selecionaram-se os seguintes:

1. Snort;
2. Suricata;
3. Bro (Zeek);
4. OpenWIPS-ng;
5. Sguil;
6. OpenDLP.

Para uma análise das vantagens e desvantagens de cada um, optou-se pelos dois primeiros IDS mais conhecidos, visto que, à partida têm uma maior comunidade de utilizadores e consequentemente um maior suporte.

4.1.1. Análise comparativa dos IDS

- **Snort**

Originalmente desenvolvido em 1998 por Martin Roesch, foi adquirido pela Cisco em 2013 mantendo a sua característica de *open-source*. Para além de realizar funções de IDS faz também funções de *Intrusion Protection System* (IPS), embora não utilizadas neste projeto.

Devido ao facto de ter uma grande comunidade de utilizadores, faz com que este seja o IDS preferido e mais conhecido. O Snort tem como base apenas uma aplicação com interface de consola para configuração e análise de comportamento, não tendo nenhuma interface gráfica oficial. Sendo *open-source*, várias interfaces gráficas foram criadas tais como a BASE [34] e o Sguil [35].

A nova versão 3.0, lançada em Janeiro de 2021 [36], permitiu a introdução de *plugins*, nomeadamente, a possibilidade do utilizador desenvolver as suas próprias regras, recorrendo à linguagem de programação Lua. A capacidade de processamento *multithread* na análise do pacote, também acrescentada nesta nova versão, melhorou significativamente o seu desempenho [37].

No entanto, este não tem perceção da camada aplicacional do modelo *Open System Interconnection* (OSI), uma vez que apenas analisa os pacotes e procura correspondência com as regras definidas.

- **Suricata**

Começou a ser desenvolvido em 2009 e em 2010 foi lançada a primeira versão pela *Open Information Security Foundation* (OISF) [22].

Para além da funcionalidade de IDS também realiza funções de IPS e é bastante conhecido pela sua capacidade de *multithreading* e de *hardware acceleration*. Adicionalmente, este apresenta a possibilidade de recorrer à placa gráfica da máquina para aumentar o desempenho [38].

Tem o mesmo princípio que o Snort, permitindo reutilizar muitas das regras, à exceção daquelas desenvolvidas especificamente para este IDS [39]. Para além disso, tem também uma comunidade muito elevada de utilizadores, logo o suporte e a adição de novas regras é elevada. Este IDS permite deteção automática na camada aplicacional, identificando HTTP, SSH, DNS, SMB, TLS, entre outros.

Possibilita guardar automaticamente ficheiros encontrados nas comunicações, permitindo ao administrador da rede aceder de forma rápida aos ficheiros e analisar a presença de vírus ou *malware* noutras aplicações.

Este IDS admite a utilização de Lua *scripting* (LuaJIT), tornando exequível a criação de regras mais complexas para a deteção de intrusões e integração com bases de dados. Ao contrário do Snort, este já tinha a funcionalidade desde a sua versão inicial.

	Snort	**Suricata**
Comunidade	Elevada	Médio
Multi-threading	✓(v3.0)	✓
Hardware acceleration	-	✓
Lua *scripting*	✓(v3.0)	✓
Camada aplicacional	-	✓
Interface gráfica	× (oficiais)	× (oficiais)

Tabela 1 - Comparação do Snort com Suricata

A Tabela 1 apresenta um breve resumo dos dois IDS em estudo. Para este projeto optou-se por utilizar Suricata porque, embora a sua comunidade de utilizadores seja inferior ao do Snort, ele tem suporte há mais tempo para funcionalidades necessárias ao projeto, o que à partida se traduz em maior estabilidade.

Com a possibilidade de utilização da linguagem de programação Lua, é possível transformar o Suricata, o qual deteta intrusões com base em assinaturas (SIDS), num sistema que faz a deteção com base no comportamento da rede (AIDS).

4.2. Escolha da base de dados

Para que o Suricata tenha capacidades de análise comportamental dos dispositivos presentes no sistema da ferrovia, é necessário guardar, de forma persistente, determinados parâmetros das comunicações referidos na Secção 3.2.

Tendo em conta que o IDS estará a ler e a escrever na base de dados cada vez que lhe chegam pacotes de uma *gateway* LoRa, é essencial que esta base de dados esteja adequada para receber modelos de aprendizagem automática e que as velocidades de escrita/leitura sejam relativamente elevadas de forma a que o desempenho do IDS não seja prejudicado.

Analisaram-se várias bases de dados apropriadas para guardar dados de dispositivos de IoT [40] e optou-se pela utilização da base de dados CrateDB [41]. Esta tem como principais características as seguintes:

- Baseada em SQL;
- Adequada para aprendizagem automática;
- Permite agregação de dados;
- Permite escalabilidade;
- Possibilidade de guardar dados em binário num formato definido pela aplicação, também conhecidos como *Binary Large Objects* (BLOBs), útil para guardar modelos.

4.3. Escolha do algoritmo

Conforme é mencionado na descrição da Figura 11, recorreu-se a algoritmos de aprendizagem automática de forma a concretizar o objetivo de classificar, de forma precisa, o estado de cada pacote recebido pelo IDS.

Existem dois tipos de algoritmos de classificação [42]:

- Modelos Lineares:
    - *Logistic Regression;*
    - *Support Vector Machines;*
- Modelos Não-Lineares:
    - K-*Nearest Neighbours* (**KNN**);
    - *Kernel* SVM;
    - *Naive Bayes*;
    - *Decision Tree Classification*;
    - *Random Forest Classification.*

Numa primeira fase, implementou-se o algoritmo *Logistic Regression*. Contudo, para o tipo de dados do trabalho, o algoritmo não mostrou ser muito preciso na classificação das amostras. Desta forma optou-se por utilizar o algoritmo K-*Nearest Neighbours* devido ao tipo de dados analisados, aos valores conhecidos e não discrepantes, ao resultado esperado, (que apenas apresenta duas possibilidades - comportamento normal ou anormal) e à facilidade de implementação [43].
O KNN classifica novos dados baseando-se em semelhanças medidas com dados anteriormente guardados. O K representa o número de vizinhos mais próximos usados para classificar os novos dados. Tipicamente este valor é igual a 5 [44] mas depende do tipo de dados das amostras, podendo ser necessário realizar retificações de forma a aumentar a precisão do sistema. Para este trabalho optou-se pela utilização do valor igual a 5, tendo em conta que é aquele que oferece melhor precisão e exatidão para os dados atuais.

4.4. Lua scripting no Suricata

A linguagem Lua pode ser utilizada no Suricata de duas formas possíveis [45]:

- *Lua Detection*;
- *Lua Output.*

O *Lua Detection* permite ao utilizador executar *scripts* em Lua cada vez que chega um novo pacote para ser analisado pelo Suricata, sendo equivalente à utilização de assinaturas estáticas, ainda que com maior liberdade. É necessário colocar o *script* na diretoria onde se encontram as regras ativas. A primeira linha deste *script* tem de ter: `lua:[!]<scriptfilename>` e o ficheiro tem de conter duas funções, a `init()`, que regista os *buffers* que são necessários inspecionar, e a `match()`, executada pacote a pacote, podendo retornar "1" se ocorrer uma correspondência ou "0" se não ocorrer. Como *buffers* são possíveis os seguintes:

- *packet* (corresponde ao pacote IP incluindo os cabeçalhos)*;*
- *payload* (corresponde ao conteúdo transportado pelo pacote)*;*
- *dns;*
- *ssh;*
- *http;*
- entre outros.

O *Lua Output* possibilita ao utilizador obter um maior detalhe no resultado de uma determinada assinatura, ou seja, é executado quando ocorre uma correspondência por parte de uma assinatura. É necessário definir quatro funções, a `init()` com o mesmo propósito que a anterior, a

`setup()` que é executada quando o Suricata inicia e serve para instanciar variáveis Lua, como por exemplo o *filesystem*. A função `deinit()` para limpar *buffers* e variáveis criadas no *script* e, por fim, a função `log()` que permite ao utilizador criar um *log* personalizado. O *Lua Output* não se encontra ativo por defeito, sendo necessário ativar previamente no ficheiro de configuração.

4.5. Máquina de Estados

O sistema é composto por 4 estados possíveis, que se encontram ilustrados na Figura 18:

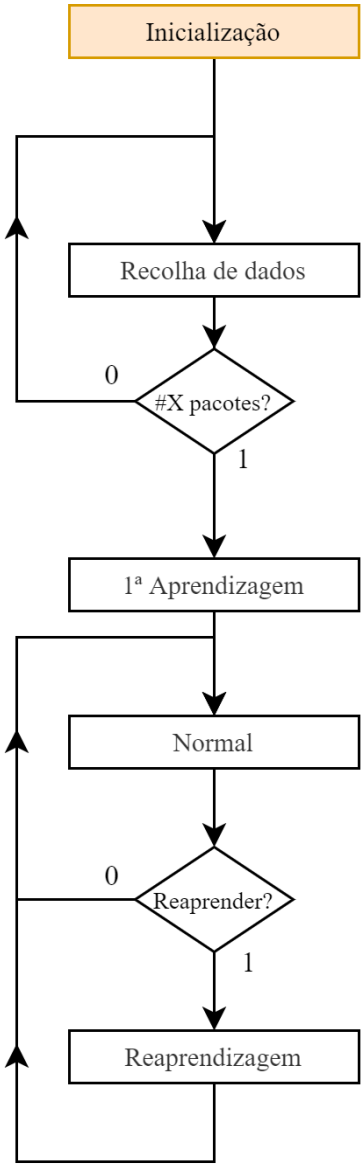

Figura 18 - Máquina de Estados

O sistema começa pelo primeiro estado, o de "Inicialização", o qual é responsável por garantir que a base de dados se encontra em correto funcionamento e pela criação das tabelas necessárias caso ainda não estejam criadas. Neste processo os dados históricos dos sensores não são eliminados.

No passo seguinte o sistema executa o estado de "Recolha de dados", sendo neste estado que o IDS cria um histórico dos dados de cada dispositivo com recurso à base de dados.

O IDS só passa para o estado seguinte se já tiver recolhido "X" pacotes de um determinado dispositivo.

A primeira aprendizagem do modelo, associado a um dispositivo, só é realizada quando já existe um histórico com um determinado número de entradas. Após a criação do modelo, este é guardado na base de dados para posteriormente ser utilizado. Para esta aprendizagem são alterados um número específico de pacotes de forma a representarem intrusões/anomalias de forma programática para que o algoritmo crie as *labels* adequadas e saiba distinguir os comportamentos.

No estado "Normal" o sistema já se encontra apto para analisar os pacotes e identificar se é um comportamento esperado, uma intrusão ou um caso de dúvida para um determinado dispositivo.

Por fim, existe a possibilidade de, ao fim de um determinado número de pacotes recebidos, ou por ação manual, o sistema realizar uma reaprendizagem com os novos dados e atualizar o modelo associado ao dispositivo.

4.6. Configurações do IDS

Para este trabalho optou-se por desenvolver a integração com a aprendizagem automática no ponto de interceção de saída (*Lua Output*). Desta forma, é necessário criar uma assinatura estática e que esta faça correspondência quando chega o pacote certo.

Segundo a Figura 9, pretende-se que o IDS funcione sobre uma rede IP e analise os pacotes UDP provenientes de uma *gateway* LoRa para um *Network Server*.

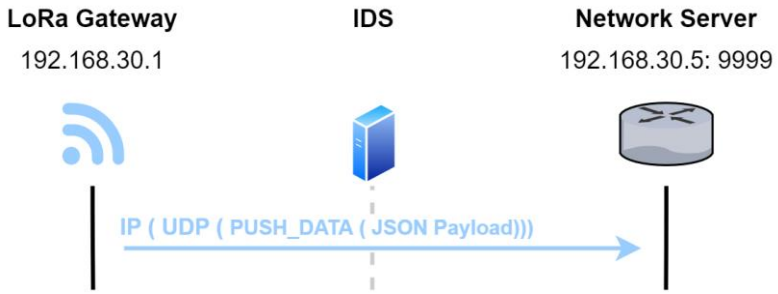

Figura 19 - Modelo Simplificado da interceção das mensagens LoRaWAN

Tendo em conta que o NS tem como endereço IPv4 o 192.168.30.5 e que se encontra à escuta no porto 9999 UDP, criou-se uma regra estática de forma a fazer correspondência no IDS, conforme é ilustrado na Figura 20.

```
alert udp  $HOME_NET any -> 192.168.30.5 9999 (msg:"GW LoRa";      \
    classtype:not-suspicious; sid:2; rev:1;)
```

Figura 20 - Assinatura para alertar dos pacotes da *gateway*

Sempre que for enviado um pacote IP com esta assinatura, é executado o *script* em Lua, permitindo assim instanciar a arquitetura proposta na Figura 11. O parâmetro: `sid:2` permite identificar a assinatura no sistema.

É no ficheiro: `…/suricata/lua-output/local.lua` que se encontra o *script* Lua que, por sua vez, contém uma função `log()` que é executada cada vez que existe uma correspondência com uma assinatura estática.

De uma forma resumida, o ficheiro tem a seguinte configuração:

```
function init (args)
    [...]
    os.execute(dir.."1_init.py")                    #ESTADO: Inicialização

    local needs = {}
    needs["type"] = "packet"
    needs["filter"] = "alerts"
    return needs
end

function setup (args)
    [...]
end

function log (args)
    sid, rev, gid = SCRuleIds()
    p = SCPacketPayload()

    if sid == 2 then
        --print("Match SID 2")

        [...]                                       #Pré-processamento

        local arg = " "..devaddr.." "..tmst_actual.." "..latitude.."      \
"..longitude.." "..sf.." "..bw.." "..lsnr.." "..rssi.." "..lenpayload.."  \
"..data.." "..tmst_last.." "..tmst_dif.." "..count


        if count < num_packets then                 #ESTADO: Recolha de dados
            os.execute(dir.."sqlclient.py"..arg)

        elseif count == num_packets then            #ESTADO: 1ª Aprendizagem
            os.execute(dir.."3_train_model.py"..arg)

        else                                        #ESTADO: Normal
            os.execute(dir.."sqlclient.py"..arg)
            os.execute(dir.."4_normal.py"..arg)
        end
    end
end

function deinit (args)

end
```

Figura 21 - Resumo do *script* Lua

Observando a Figura 21, verifica-se as quatro funções obrigatórias do Suricata, conforme foi mencionado anteriormente. É na função `init()` que se encontra a execução do estado "Inicialização", realizado a partir da linguagem Python, e é também instanciado os *buffers* necessários para o *script*. A função `log()`é chamada sempre que existe uma correspondência com uma assinatura e a primeira condição tem como propósito restringir apenas para a assinatura com `sid=2`. A variável `arg`, resultado do pré-processamento, consiste em retirar parâmetros do pacote IP em questão e prepará-lo para as funções seguintes. É possível constatar os parâmetros que são utilizados na aprendizagem, nomeadamente o endereço do dispositivo (devaddr), o *timestamp* atual (tmst_atual), latitude, longitude da *gateway*, o fator de espalhamento (sf), a largura de banda (bw), a relação sinal-ruído (lsnr), a potência de sinal recebido na *gateway* (rssi),

o comprimento do *payload* hexadecimal (lenpayload) e a diferença entre o *timestamp* atual com o anterior (tmst_dif). Por fim, encontram-se as condições para os outros estados e funções Python associadas.

O sistema só passa para o estado "Recolha de dados" quando o número de pacotes recebidos, associados ao dispositivo, ultrapassa um determinado valor. No entanto, é possível configurar o sistema, não por número de pacotes, mas por horas ou dias de funcionamento.

4.7. Estrutura de dados

A CrateDB é a base de dados utilizada para armazenar todos os dados que são importantes para o correto funcionamento do sistema, estando definidas quatro tabelas. A tabela "**sensores**", representada na Figura 22, tem como função guardar todo os parâmetros necessários de cada sensor de forma a criar um histórico. A tabela "**dúvidas**" tem a função de guardar as mensagens que o sistema identificou como caso de dúvida. A tabela "**models**", representada na Figura 23, tem como responsabilidade o armazenamento dos ponteiros para os modelos criados pelo algoritmo. Por fim a tabela "**modeltable**" tem a característica de ser BLOB de forma a guardar o modelo em si.

Figura 22 - Tabela "sensores"

A tabela "**sensores**" tem três variáveis: a "devaddr", que tem como valor o endereço do dispositivo; a variável "tmst", que tem como valor o parâmetro LoRa "time" (presente na Figura 6), convertido para *timestamp*; e, por fim, a variável "message", que corresponde a um objeto JSON. Existiu a necessidade de se utilizar uma variável "tmst" para que seja possível distinguir um pacote de outro.

A estrutura de dados "message" tem os vários parâmetros que foram considerados pertinentes para criar um comportamento normal para cada dispositivo. O parâmetro "flag" é o que identifica a mensagem como sendo uma intrusão, um comportamento normal ou uma dúvida.

A tabela "**dúvidas**" é semelhante à tabela anterior em termos de parâmetros, à exceção da "flag" que neste caso representa a probabilidade da mensagem ser intrusão para, posteriormente, um utilizador avaliar manualmente.

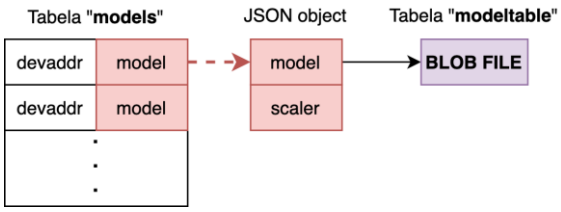

Figura 23 - Tabela "models" e "modeltable"

A tabela "**models**" tem presente duas variáveis, a "devaddr", que tem como propósito associar o modelo ao dispositivo. A variável "model" é um objeto JSON que tem um ponteiro para um ficheiro BLOB (com o modelo em si) e o "scaler" com o *scaler* do modelo.

4.8. Fluxo do algoritmo de classificação

O primeiro fluxograma, representado na Figura 24 , diz respeito à função: `3_train_model.py`, executada no estado "1ª Aprendizagem", numa primeira fase, para todos os dados existentes na tabela "**sensores**", para um determinado dispositivo.

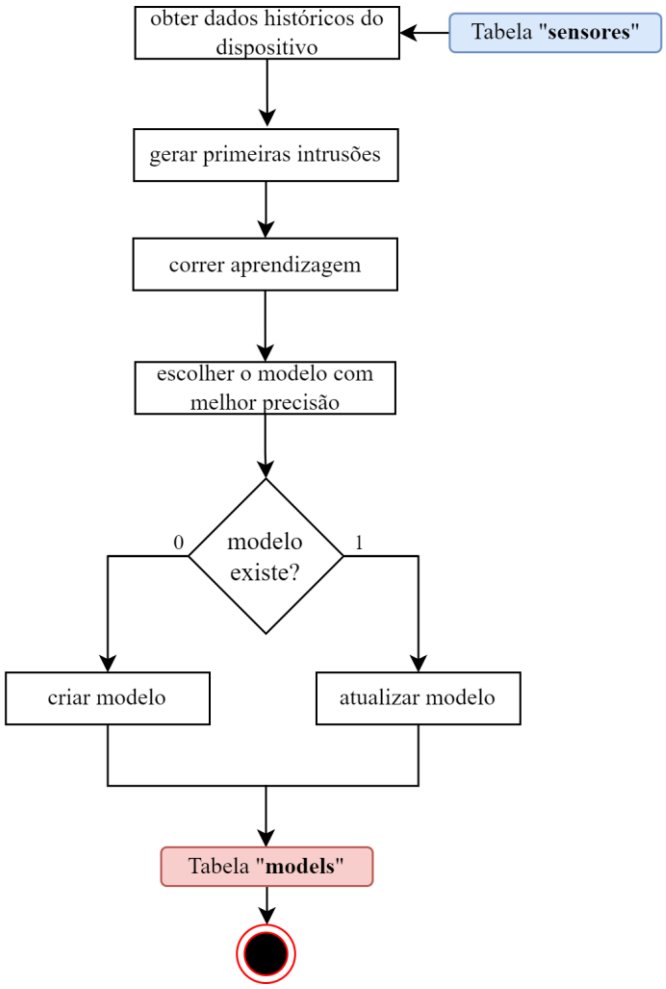

Figura 24 - Fluxograma função "3_train_model.py"

Para que seja possível realizar uma primeira aprendizagem sobre os dados existentes é necessário, em primeiro lugar, gerar um determinado número de intrusões de forma a criar *labels* para que o algoritmo aprenda a distinguir um comportamento normal de uma anomalia. É a função: `2_auto_intrusions.py` que é responsável por criar as primeiras intrusões com base nos valores históricos do dispositivo. Com os dados previamente preparados, recorreu-se ao *multithreading* para executar mais eficazmente a função `fit()` e escolher o modelo com maior precisão. Com o novo modelo calculado, o sistema atualiza ou cria uma nova entrada na base de dados para o *scaler* e o modelo associados ao dispositivo.

```
[...]
threads = [None] * NUM_THREADS
results = [None] * NUM_THREADS
for i in range(len(threads)):
    threads[i] = Thread(target=test_model, args=(X, Y, results, i))
    threads[i].start()

for i in range(len(threads)):
    threads[i].join()

average = [ ]
for i in range(len(results)):
    average.append(results[i][0])

average_max_index = average.index(max(average))
[...]
```

Figura 25 - Resumo da função "3_train_model.py"

A Figura 25, tem como objetivo mostrar a utilização de *multithreading* de forma a encontrar o valor mais elevado de precisão do modelo no menor espaço de tempo.

A função de aprendizagem, `test_model()`, está representada na Figura 26.

```
def test_model(X, Y, result, index):

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)


    X_t = X_test                                        #Original

    X_train = np.array(X_train)
    Y_train = np.array(Y_train)
    Y_test = np.array(Y_test)

    scaler = StandardScaler()
    scaler.fit(X_train)                                 #Aprendizagem do scaler

    X_train = scaler.transform(X_train)                 #Preparação dos dados
    X_test = scaler.transform(X_test)


    classifier = KNeighborsClassifier(n_neighbors=5)
    classifier.fit(X_train, Y_train)                    #Aprendizagem do modelo

    y_pred = classifier.predict(X_test)
    y_pred_prob = classifier.predict_proba(X_test)
    average_precision = average_precision_score(Y_test, y_pred)

    result[index] = [average_precision, classifier, scaler, X_train,    \
    X_test, Y_train, Y_test, y_pred, y_pred_prob,  X_t]
```

Figura 26 - Função aprendizagem

Esta função recebe como parâmetro o *dataset* com o qual o modelo vai ser ensinado, separa 30% dos dados para teste e o restante para treino, baralha e realiza a aprendizagem com o algoritmo KNN com 5 vizinhos. Com o resultado da função `fit()` e com os dados para teste, é possível calcular a precisão de forma a escolher qual o melhor modelo.

O segundo fluxograma representa a função: `4_normal.py`, executada no estado “Normal”, onde o sistema se vai encontrar na maior parte do tempo.

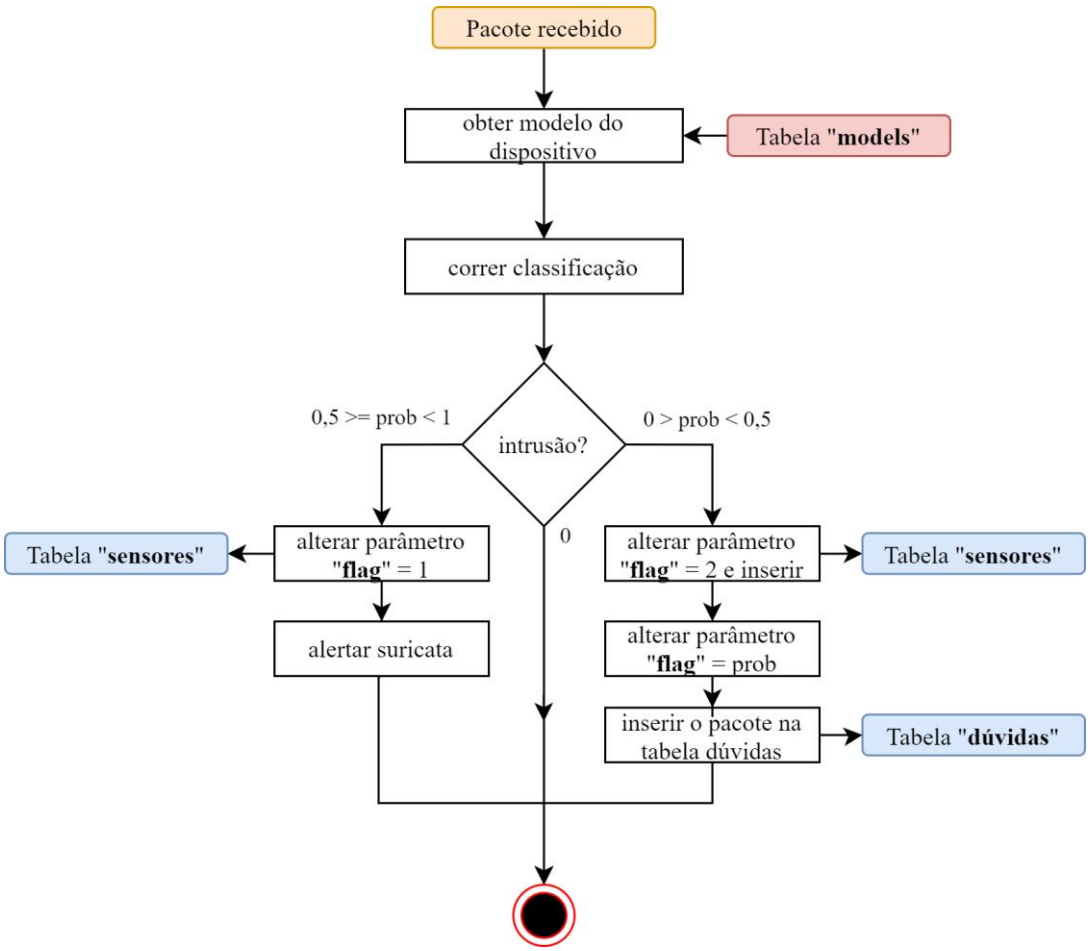

Figura 27 - Fluxograma função "4_normal.py"

Esta função, ao contrário da anterior, não é executada sobre um conjunto de amostras, mas sim pacote a pacote recebido pelo IDS. No estado “Normal”, após o pré-processamento, é necessário ir à tabela “**models**” da base de dados buscar o modelo associado ao dispositivo e, numa primeira fase, transformar o pacote em questão com recurso ao *scaler*. Posteriormente a função `predict()` é executada e obtém-se o resultado da classificação. Podem ocorrer 3 caminhos possíveis. Se a probabilidade for igual a zero, significa que não foi detetada intrusão, é alterado o parâmetro `flag=0` e a função termina. Caso o valor seja superior a 0,5, este é classificado como intrusão, é alterado o parâmetro `flag=1` na tabela “**sensores**” e alertado o Suricata do acontecimento. Por fim, caso o valor esteja compreendido entre 0 e 0,5, não inclusive, o sistema classifica o pacote como caso de dúvida. Neste caso específico é necessário alterar o parâmetro `flag=2` na tabela “**sensores**” e copiar o pacote para a tabela “**dúvidas**” com o parâmetro `flag` igual à probabilidade classificada. Na tabela “**sensores**” o valor do parâmetro `flag=2` tem de ser

alterado de forma a que, em caso de reaprendizagem do modelo, este não entre para o processo visto ser um caso de dúvidas que necessita de ser revisto. Optou-se por alterar o valor do parâmetro `flag` para o valor da probabilidade no caso de dúvida para que o utilizador, quando analisar a tabela, saiba o grau de incerteza da classificação.

4.9. Interface Gráfica

Conforme foi mencionado na Secção 3.2, o utilizador tem a possibilidade de visualizar de forma gráfica os pacotes que foram classificados pelo sistema como sendo caso de dúvida. A Figura 28 ilustra a janela principal do programa.

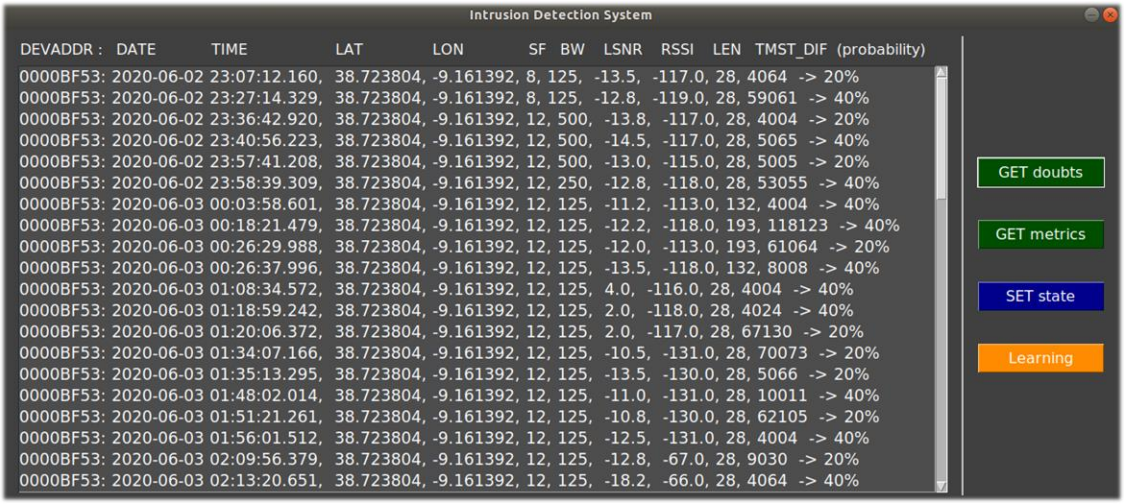

Figura 28 - Janela principal da interface gráfica

Após a inicialização do programa, o utilizador tem à sua disposição quatro botões posicionados verticalmente à direita da janela e à esquerda uma lista dos pacotes classificados como dúvida juntamente com a legenda na parte superior da janela. É o botão “GET doubts” que tem a função de realizar o contacto com a base de dados, recolher os casos de dúvida e apresentar em formato de lista. O botão “GET metrics”, quando previamente selecionado um pacote da lista, tem a função de abrir uma janela informativa, representada na Figura 29, do lado direito da janela principal. O botão a azul “SET state” permite ao utilizador classificar o pacote selecionado como sendo uma intrusão ou comportamento normal. Por fim, o botão a laranja “Learning” tem como objetivo permitir ao utilizador indicar ao sistema para realizar uma nova aprendizagem tendo como base os dispositivos alterados.

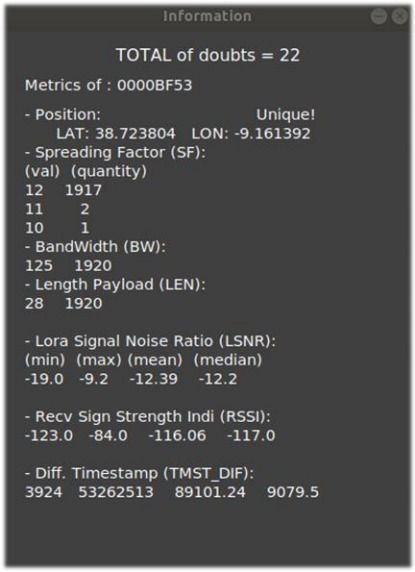

Figura 29 - Janela informativa

A janela informativa, executada pelo botão "GET metrics" tem como função mostrar ao utilizador um conjunto de métricas associadas a um determinado dispositivo, de forma a auxiliar no processo de decisão. Esta indica quantos casos de dúvida existem para serem avaliados, se a posição geográfica da *gateway* é única ou não, a quantidade e valor associados aos parâmetros "Spreading Factor" e "Bandwidth" e o comprimento do *payload*. Por último, informa os valores mínimos, máximos, a média e a mediana associada ao parâmetro "LSNR", ao "RSSI" e à periodicidade entre pacotes. Estas informações são recolhidas da tabela "**sensores**" e calculadas no momento da execução.

Com o complemento da informação obtida da janela informativa, o utilizador pode classificar o pacote como sendo uma intrusão ou como sendo um comportamento normal clicando no botão "SET state" que abre a seguinte janela:

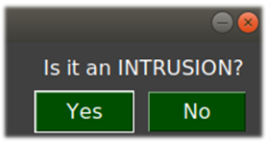

Figura 30 - *Popup* de confirmação

Caso o utilizador selecione a opção "Yes", o programa altera o parâmetro `flag=1` na tabela "**sensores**" e apaga o pacote atual da tabela "**dúvidas**". Caso selecione a opção "No", ocorre o mesmo procedimento que o anterior, contudo o valor é alterado para `flag=0`. Por fim, caso o utilizador se engane e não queira alterar o estado do pacote, basta apenas fechar a janela.

5. Resultados

Os resultados tiveram como base uma parte da amostra de dados analisada na Secção 3.3. A arquitetura do sistema foi implementada e testada numa máquina virtual com o sistema operativo *Ubuntu* LTS de 64bit e com a versão do Suricata 6.0.2.

A máquina virtual tem as seguintes características:

- Processador Intel Core i7-6700HQ @ 2.60GHz de 4 núcleos;
- 12 GB de RAM;
- 20 GB de armazenamento interno.

Optou-se por dividir os resultados com base em 4 experiências. As amostras de dados, relativas a um mês, foram previamente analisadas na Secção 3.3. Uma primeira experiência consistiu em utilizar os dois primeiros dias para treino e o primeiro dia como teste, desta forma valida-se a aprendizagem do sistema. Uma segunda experiência consistiu em utilizar os dois primeiros dias para treino e o terceiro dia como teste, com isto valida-se a resposta do sistema a dados desconhecidos. Uma terceira experiência consistiu em utilizar os dois primeiros dias para treino e o terceiro dia, com intrusões, para teste, deste modo valida-se a resposta a intrusões. Por fim a quarta experiência tem o mesmo propósito que a anterior, mas com mais amostras de dados para treino.

Da primeira experiência, os dados referentes ao primeiro dia das amostras, dia 01/06/20, estão representados na Figura 31 e é possível verificar o comportamento normal do dispositivo com o endereço “0000BF53”. A Figura 32 corresponde à classificação proveniente do sistema para as amostras do primeiro dia.

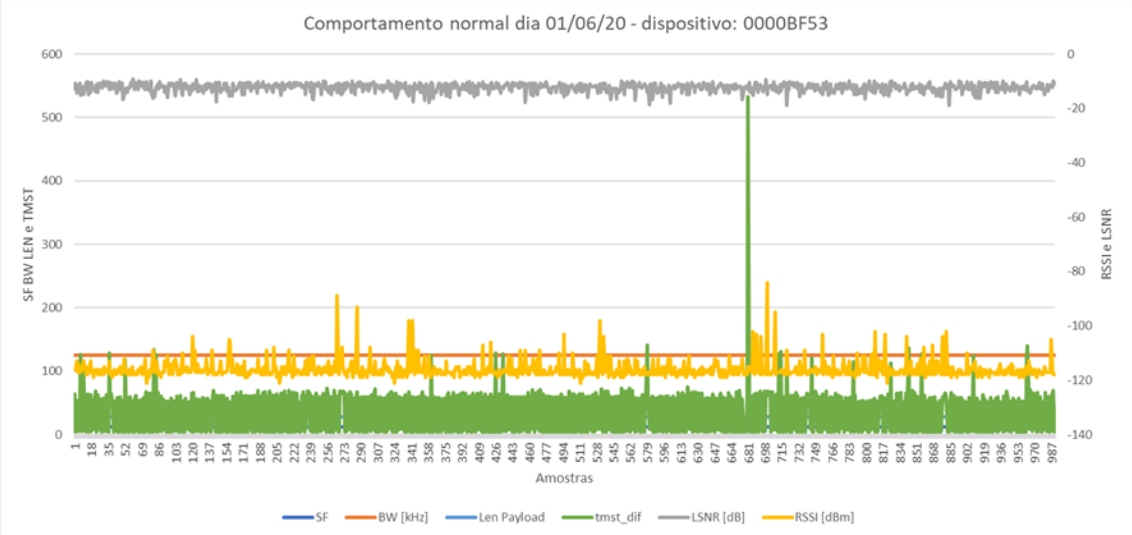

Figura 31 - Amostras do dispositivo 0000BF53 no dia 01/06/20

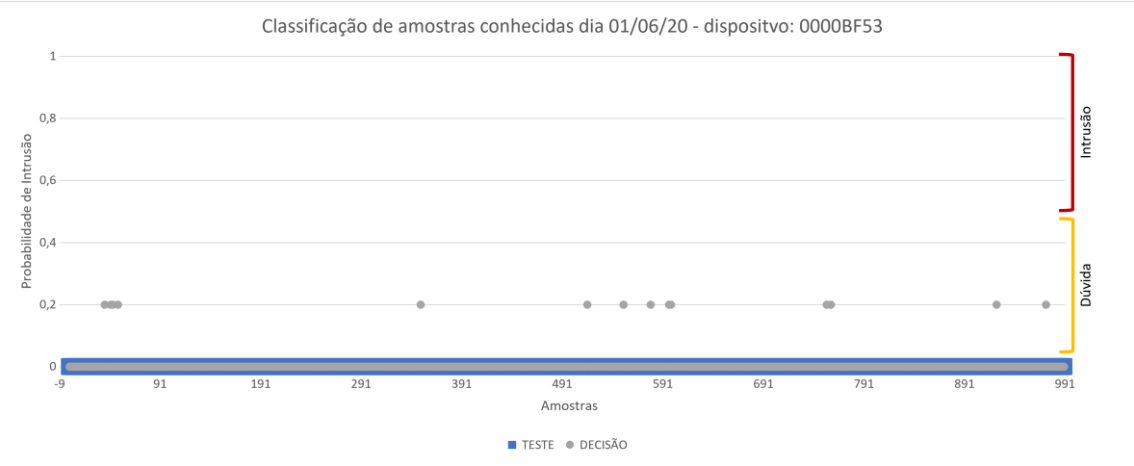

Figura 32 - Classificação das amostras do primeiro dia

O IDS foi previamente treinado com 70% dos dados do dia 01/06/20 e do dia 02/06/20, sendo que os restantes 30% serviram para testar o modelo. Numa primeira fase pretendeu-se validar de que forma o sistema classifica um conjunto de amostras já conhecidas. Do primeiro dia existiam 991 amostras e do segundo dia existiam 975 amostras, totalizando 1966 amostras.

A Figura 31 representa o dia 01/06/20 em que todas as amostras são consideradas comportamento normal, no entanto ao analisar a Figura 32 o sistema não classificou todas as amostras como esperado. Das 991 amostras, 14 foram interpretadas como caso de dúvida, representando apenas 1,4%. Idealmente todas as amostras deviam de ter sido classificadas como normais visto que o sistema já tinha conhecimento das mesmas.

Observa-se pela Figura 31 que todos os parâmetros representados têm um comportamento praticamente estável ao longo do dia, o LSNR encontra-se em torno de -15 dB, o RSSI em torno de -115 dBm, a BW com um valor de 125 KHz e o "len" com 28. A diferença de tempo entre pacotes (tmst_dif) tem um comportamento periódico ao longo do dia à exceção de uma determinada altura do dia que o dispositivo altera a sua periodicidade (amostra 681).

O resultado desta classificação pode estar relacionado com a função: `3_train_model.py`, uma vez que esta é responsável por alterar os parâmetros de 45 amostras de forma a ensinar o sistema do que é uma anomalia. Definiu-se como casos positivos as amostras classificadas como intrusão. Nesta experiência não existiu nenhum caso positivo, logo não fez sentido calcular medidas de desempenho, visto que estas encontram-se relacionadas com o número de casos positivos.

Uma segunda experiência consistiu em manter o mesmo modelo já ensinado com os dois primeiros dias e pedir ao sistema para classificar o dia 03/06/20, amostras desconhecidas e sem intrusões. O seu comportamento encontra-se na Figura 33 e a classificação na Figura 34.

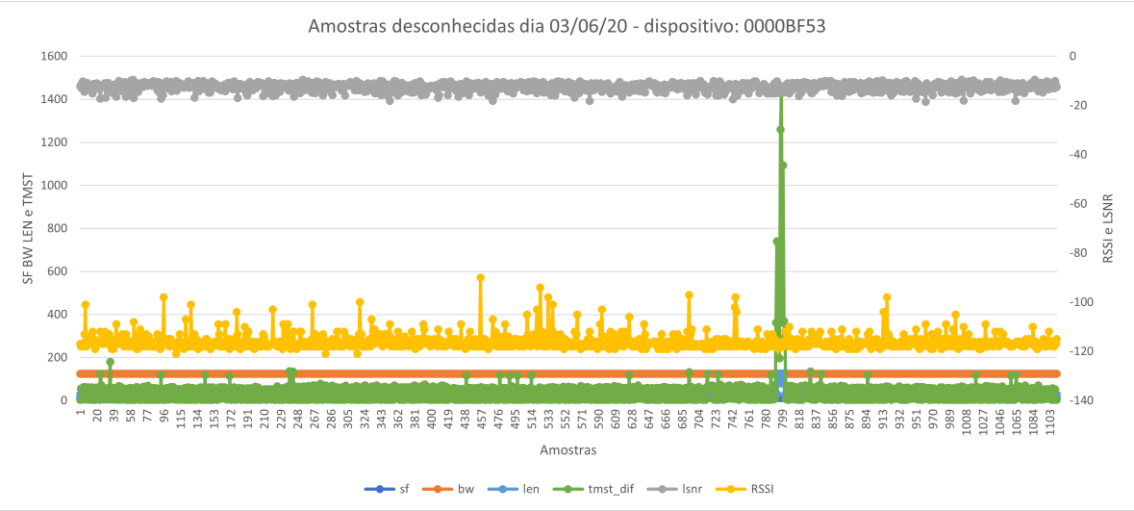

Figura 33 - Amostras do dispositivo 0000BF53 no dia 03/06/20

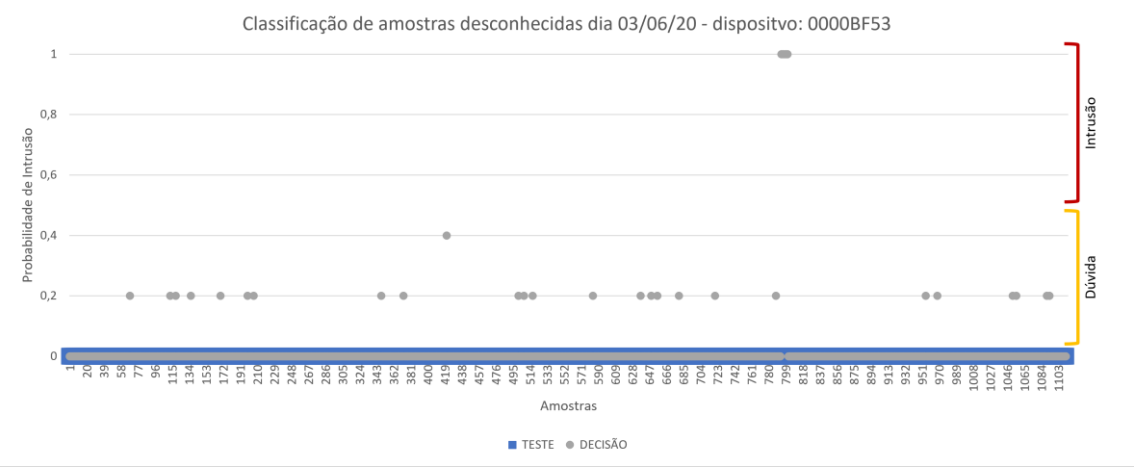

Figura 34 - Classificação das amostras do terceiro dia

As amostras apresentadas na Figura 33 do terceiro dia foram bastante idênticas ao primeiro dia do dispositivo. A classificação errada das amostras como intrusões, falsos positivos, pode estar relacionada com o facto do parâmetro "tmst_dif" ter valores mais discrepantes que o primeiro dia. Idealmente esperava-se que o sistema classificasse todas as amostras com uma probabilidade de intrusão = 0, no entanto, conforme a Figura 34, de um total de 1112 amostras, 30 foram detetadas como caso de dúvida e 4 como intrusão.

Tal como a experiência anterior, não foi realizada uma classificação com amostras definidas como intrusão, logo não faz sentido calcular medidas de desempenho.

Uma terceira experiência consistiu em manter o mesmo modelo já ensinado com os dois primeiros dias e utilizar na mesma o terceiro dia do dispositivo para classificar, mas desta vez com intrusões fictícias, verdadeiros-positivos. As intrusões foram geradas com o mesmo algoritmo utilizado na primeira aprendizagem do modelo visto que, até ao momento, não foram encontradas intrusões reais. Do conjunto de 1112 amostras, gerou-se 34% de intrusões, ou seja, 384 amostras foram

definidas como intrusões e as restantes não foram alteradas. A Figura 35 evidencia as novas amostras do terceiro dia e a Figura 36 corresponde à sua classificação.

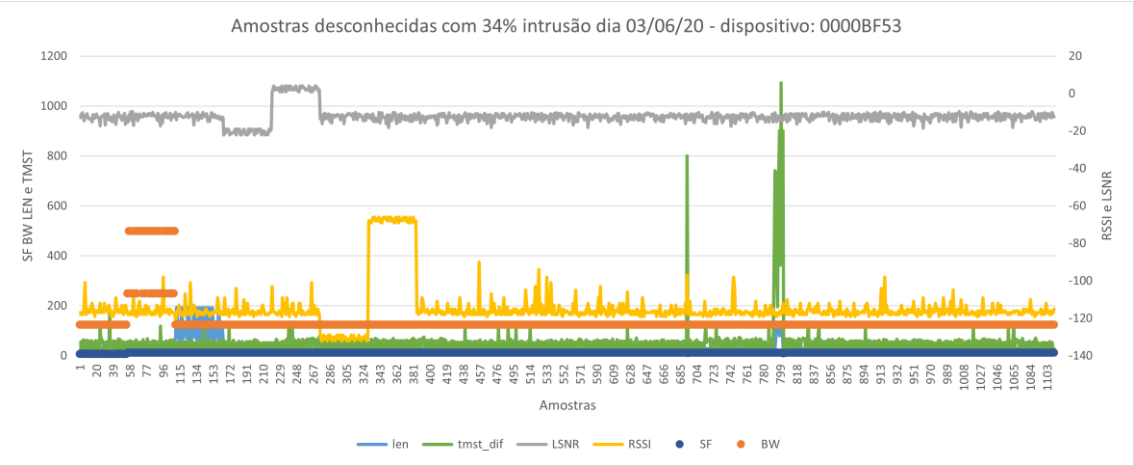

Figura 35 - Amostras do dispositivo 0000BF53 no dia 03/06/20 com 34% de intrusões

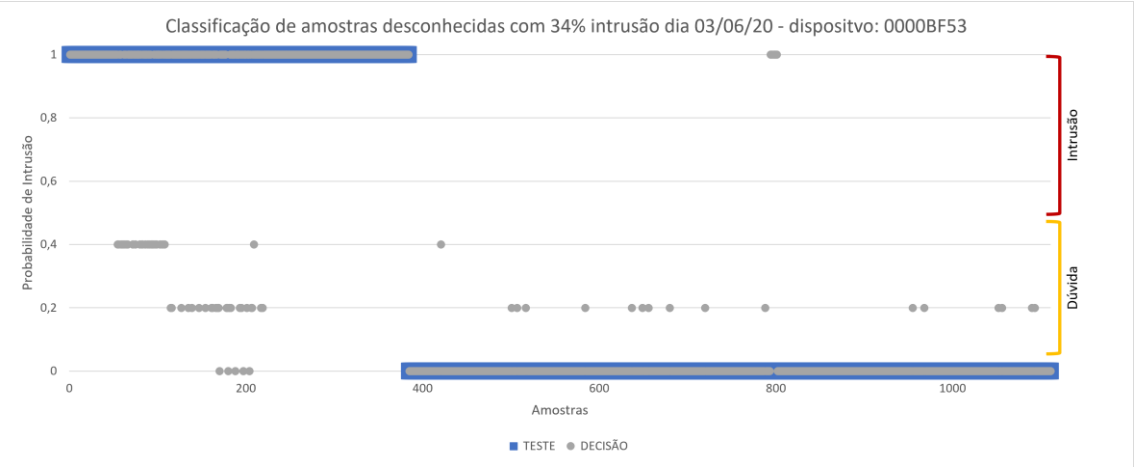

Figura 36 - Classificação das amostras do terceiro dia com 34% de intrusões

É possível identificar na Figura 35 que as primeiras 384 amostras foram definidas como intrusões, visto que têm um comportamento bastante díspar das restantes amostras. Na geração das intrusões fictícias fez-se questão de alterar o seu padrão de forma individual para cada parâmetro, com o objetivo de validar o funcionamento do classificador em relação a cada parâmetro.

A Figura 36 evidencia também a importância da criação da terceira classificação, os casos de dúvida, visto que é desta forma que o sistema garante que possíveis intrusões sejam mal classificadas e que posteriormente aprenda com elas, como é o caso das amostras 59 à 219.

Da terceira experiência a matriz de confusão foi a seguinte:

Real \ Pred.	Intrusões	Normal
Intrusões	**330**	**8**
Normal	**5**	**702**

Tabela 2 - Matriz de confusão da 3ª Experiência

- 330 amostras como intrusões corretamente, verdadeiros-positivos;
- 8 amostras como intrusão erradamente, falsos-positivos;
- 702 amostras como normais corretamente, verdadeiros-negativos;
- 5 amostras como normais erradamente, falsos-negativos;
- **71** amostras como **caso de dúvida**.

A Figura 36 permite observar que a classificação das amostras próximas da 795 (correspondendo a falsos-positivos), coincide com a variação abrupta do parâmetro "tmst_dif" e do "len". É de salientar que o algoritmo que foi utilizado para gerar 34% das intrusões é o mesmo que é usado pelo sistema numa primeira fase de aprendizagem. Logo, o sistema aprendeu com a variação imposta no algoritmo de geração de intrusões.

A precisão do sistema é calculada da seguinte forma:

$$\text{precisão} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{330}{330 + 8} \cong 0{,}976 \tag{5}$$

A sensibilidade (*recall*) do sistema calculou-se de acordo com a fórmula:

$$\text{sensibilidade} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{330}{330 + (54 + 5)} \cong 0{,}848 \tag{6}$$

A exatidão do sistema, contando com os casos de dúvida é a seguinte:

$$\text{exatidão} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = \frac{330 + (702 + 17)}{330 + (702 + 17) + 8 + (54 + 5)} \cong 0{,}940 \tag{7}$$

Por fim, o F1 *score* é o seguinte:

$$\text{F1 } score = \frac{2 \times (recall \times precision)}{recall + precision} = \frac{2 \times (0{,}848 \times 0{,}976)}{0{,}848 + 0{,}976} \cong 0{,}908 \tag{8}$$

	Precisão	**Sensibilidade**	**Exatidão**	**F1 *score***
3ª Experiência	0,976	0,848	0,940	0,908
3ª Experiência[2] (s/ dúvidas)	0,976	0,985	0,987	0,980

Tabela 3 - Medidas de desempenho para a 3ª Experiência

A Tabela 3 ilustra dois tipos de cálculos para a mesma experiência, o primeiro considerando os casos de dúvida como falsos-negativos/verdadeiros-negativos e o segundo ignorando os casos de dúvida. Para o primeiro caso, tendo em consideração que os casos de dúvida são falsos-negativos, onde as amostras são consideradas intrusões, ou verdadeiros-negativos, onde as amostras são consideradas não intrusões, obteve-se 54 amostras falsas-negativas (parte mais à esquerda da Figura 36) e 24 amostras verdadeiras-negativas (parte mais à direita da Figura 36). No segundo caso os cálculos são mais simplificados visto que os casos de dúvida não são contabilizados.
Verifica-se que a precisão do sistema não é alterada porque esta tem apenas em conta os casos positivos. O facto de se ter implementado os casos de dúvida no sistema fez com que possíveis intrusões não fossem automaticamente interpretadas como falsos-negativos, levando a um aumento da sensibilidade. Contudo, possibilitou que possíveis comportamentos normais, fossem igualmente interpretados como casos de dúvida. Embora as medidas de desempenho, não utilizando casos de dúvida, apresentem valores bastante elevados, é necessário ter em conta o facto de que possíveis intrusões sejam contabilizadas como comportamento normal, induzindo o sistema em erro ao longo do seu tempo de funcionamento.

Uma quarta experiência foi realizada com o intuito de perceber o que foi mencionado no artigo [19]. Os autores indicaram que quanto maior a amostra de dados com que o sistema aprende, maior a precisão associada. Desta forma recorreu-se a 6 dias para aprendizagem, desde o dia 01/06/20 ao dia 06/06/20 e o sétimo dia foi utilizado para validar o classificador. Realizou-se o mesmo procedimento que o da experiência anterior, ou seja, gerou-se 34% de intrusão do total de 1025 amostras. A Figura 37 aborda as amostras que foram enviadas para que o sistema as classificasse e a Figura 38 mostra o resultado da classificação.

[2] Experiência não considerando os casos de dúvida.

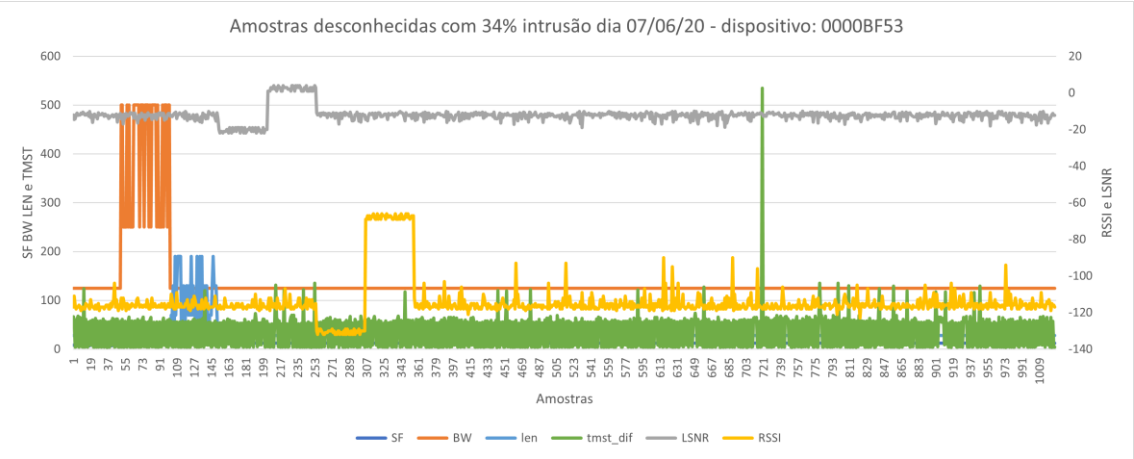

Figura 37 - Amostras do dispositivo 0000BF53 no dia 07/06/20 com 34% de intrusões

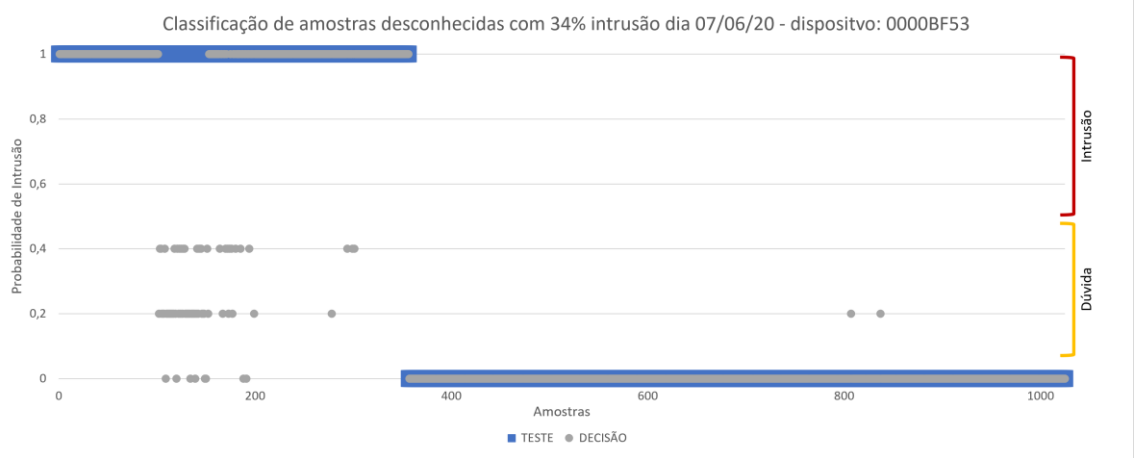

Figura 38 - Classificação das amostras do sétimo dia com 34% de intrusões

Pela Figura 37 observa-se que no sétimo dia, embora ainda existam alguns valores superiores do parâmetro "tmst_dif", este já não é considerado uma intrusão pelo sistema. Relativamente à classificação errada de não intrusão, nas amostras 102 à 200, o motivo já não foi devido ao parâmetro LSNR mas sim à variação do parâmetro SF.

Da quarta experiência a matriz de confusão foi a seguinte:

Real \ Pred.	Intrusões	Normal
Intrusões	**287**	**0**
Normal	**8**	**667**

Tabela 4 - Matriz de confusão da 4ª Experiência

- 287 amostras como intrusões corretamente, verdadeiros-positivos;
- 0 amostras como intrusão erradamente, falsos-positivos;
- 667 amostras como normais corretamente, verdadeiros-negativos;
- 8 amostras como normais erradamente, falsos-negativos;
- **63** amostras como **caso de dúvida**.

As medidas de desempenho foram calculadas tendo em conta as mesmas fórmulas que as anteriores e obteve-se o seguinte:

	Precisão	**Sensibilidade**	**Exatidão**	**F1 *score***
4ª Experiência	1,00	0,806	0,933	0,893
4ª Experiência[3] (s/ dúvidas)	1,00	0,973	0,992	0,986

Tabela 5 - Medidas de desempenho para a 4ª Experiência

Comparando a Tabela 3 com a Tabela 5 verifica-se que houve um aumento da precisão para 1,00, conforme era esperado. Isto significa que o sistema não classificou nenhuma amostra de comportamento normal como sendo uma intrusão. A sensibilidade diminuiu devido ao facto de terem existido mais casos de dúvida como sendo falsos-negativos.

Verificou-se que a implementação do sistema de dúvidas permitiu um maior controlo das amostras que entram para a aprendizagem/reaprendizagem do modelo, diminuindo assim a probabilidade de o sistema começar a aprender com amostras classificadas erradamente ao longo do tempo. Tal provocou um aumento nas medidas de desempenho, que rondaram sempre em torno dos 0,98, e garantiu que possíveis intrusões não fossem classificadas como normais. No entanto, este maior controlo obriga a existência de um utilizador que periodicamente analise o funcionamento e histórico dos dados para classificar manualmente as amostras.

[3] Experiência não considerando os casos de dúvida.

6. Conclusões

Os sistemas ferroviários estão a passar por um processo de modernização dos protocolos de telecomunicações para melhor suportar cenários de IoT, como a sensorização e telemetria.. Desta forma foi desenvolvido um novo sistema, o FRMCS. Este tem como propósito definir um conjunto de requisitos, tanto ao nível protocolar como ao nível de telemetria, IoT, possibilitando às redes ferroviárias recorrerem a protocolos já em uso pela população (4G, 5G, LPWAN) para suportar as comunicações. O problema inerente ao recorrer a redes de uso generalizado implica que a segurança aplicada às redes tenha de ser mais exigente, tendo em conta que os sistemas utilizados na ferrovia podem ser considerados críticos para as pessoas.

O presente documento abordou a possibilidade de adicionar ao sistema FRMCS, um mecanismo de segurança amplamente utilizado em redes de computadores, o IDS/IPS. O mecanismo tem a função de analisar o comportamento de uma rede e reportar para a central caso ocorra alguma anomalia. O projeto abordou casos de uso relacionados com IoT, nomeadamente com o protocolo LoRaWAN, visto que os dispositivos de IoT tendem a ser mais imprevisíveis e existem atualmente poucos sistemas capazes de avaliar o estado da rede.

Recorreu-se a um conjunto de dados reais, de uma *gateway* LoRaWAN, da cidade de Lisboa em Portugal. Esta recebeu mensalmente uma média de 500 000 mensagens de milhares de dispositivos e serviu para realizar uma análise aos vários parâmetros da rede e perceber quais é que faziam mais sentido de serem utilizados e qual a relação entre eles.

Tendo em conta a variação dos dados e a impossibilidade de obter o conteúdo da mensagem enviada pelos dispositivos, verificou-se que seria benéfico recorrer a algoritmos de aprendizagem automática, em particular de classificação. Este método facilitaria a criação de um modelo, baseando-se no histórico dos dispositivos, e permitiria classificar novos dados mais facilmente. Aferiu-se que devido ao movimento normal das carruagens seria necessário um IDS por cada carruagem e um modelo por cada dispositivo.

Analisaram-se vários IDS *open-source* presentes no mercado e constatou-se que todos recorrem a assinaturas estáticas (SIDS) e não com base em anomalias (AIDS) para detetar intrusões. Deste modo foi necessário recorrer a classificadores como o KNN e a uma base de dados de forma a guardar o histórico e os modelos dos dispositivos. Como linguagens de programação utilizou-se *Lua* e *Python*, a primeira para interligar o IDS Suricata com o classificador e a segunda para tirar partido dos algoritmos de classificação.

Foram realizadas várias experiências com o intuito de validar a precisão e exatidão do modelo. Observou-se que, numa primeira fase, as 45 intrusões fictícias criadas no sistema para efeitos de aprendizagem inicial permitiram uma precisão elevada na distinção de um comportamento normal de uma anomalia na rede. Averiguou-se também que quanto maior o número de amostras

presentes no processo de aprendizagem, maior a precisão do sistema. A implementação de uma terceira posição de classificação, os casos de dúvida, permitiu aumentar a precisão do sistema, possibilitando ao utilizador classificar manualmente amostras que foram interpretadas com uma probabilidade baixa de intrusão.

Pode-se concluir que o objetivo principal do projeto foi alcançado com sucesso embora não tenha sido testado com intrusões reais devido à inexistência de amostras de uma rede LoRaWAN. No entanto, as alterações efetuadas nas intrusões fictícias tiveram em conta os casos de uso do sistema. A precisão do sistema manteve-se superior a 90% e observou-se que quanto mais amostras o sistema tiver para aprender, melhor a sua precisão.

6.1. Trabalho Futuro

Propõem-se algumas hipóteses como trabalho futuro, nomeadamente: utilizar outros algoritmos de aprendizagem automática, como por exemplo, redes neuronais e *deep learning* de forma a obter uma maior precisão e exatidão no resultado da classificação; realizar mais experiências, com mais amostras e diferentes tipos de dispositivos para se perceber o comportamento do sistema na classificação dos mesmos; avaliar a hipótese de recorrer a mais parâmetros LoRaWAN, tal como o canal ou o campo de dados da camada LoRa com o objetivo de criar uma melhor identidade do dispositivo. Atualmente o sistema invoca o classificador por cada pacote que faz correspondência na assinatura, no entanto uma melhoria ao nível do desempenho seria reduzir o número de chamadas ao classificador, guardando em memória um determinado número de amostras e em seguida enviá-las para o classificador. Outra melhoria significativa seria ao nível da aprendizagem do modelo, uma vez que neste momento é utilizado o histórico todo conhecido a cada nova aprendizagem. Caso se diminua o histórico, aumenta-se o desempenho do sistema, diminuindo o tempo de processamento e mantendo a mesma precisão.

Por fim, tendo em conta que o sistema determina a existência de um IDS por cada comboio, seria interessante recorrer à tecnologia *blockchain* para criar uma rede de IDS distribuídos, garantindo a descentralização da informação e aumentando a robustez contra possíveis ataques à rede.

Bibliografia

[1] «what-is-lorawan.pdf». Acedido: 10 de Maio de 2022. [Em linha]. Disponível em: https://lora-alliance.org/wp-content/uploads/2020/11/what-is-lorawan.pdf

[2] «Why LoRa Is the Best Option for Smart City and Smart Building Applications», *Intellias*, 24 de Setembro de 2019. https://www.intellias.com/why-lora-is-the-best-option-for-smart-city-and-smart-building-applications/ (acedido 29 de Março de 2021).

[3] U.-I. union of railways, «FRMCS», *UIC - International union of railways*, 4 de Março de 2021. https://uic.org/rail-system/frmcs/ (acedido 4 de Março de 2021).

[4] «GSM-R: the railway's mobile communication system», *Network Rail*. https://www.networkrail.co.uk/running-the-railway/gsm-r-communicating-on-the-railway/ (acedido 10 de Maio de 2022).

[5] «FRMCS: next-generation train radio begins to take shape», *International Railway Journal*. https://www.railjournal.com/in_depth/frmcs-next-generation-train-radio-begins-to-take-shape/ (acedido 4 de Março de 2021).

[6] «What is an Intrusion Detection System?», *Palo Alto Networks*. https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids (acedido 10 de Maio de 2022).

[7] A. Khraisat, I. Gondal, P. Vamplew, e J. Kamruzzaman, «Survey of intrusion detection systems: techniques, datasets and challenges», *Cybersecurity*, vol. 2, n. 1, p. 20, Jul. 2019, doi: 10.1186/s42400-019-0038-7.

[8] «LoRaWAN™ Architecture - Developer Help». https://microchipdeveloper.com/lora:lorawan-architecture (acedido 10 de Dezembro de 2020).

[9] «Lora-net/packet_forwarder», *GitHub*. https://github.com/Lora-net/packet_forwarder (acedido 23 de Março de 2021).

[10] «LoRa — LoRa documentation». https://lora.readthedocs.io/en/latest/ (acedido 12 de Outubro de 2021).

[11] «ABP vs OTAA». https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/ (acedido 22 de Março de 2021).

[12] «History of artificial intelligence», *Wikipedia*. 17 de Outubro de 2021. Acedido: 18 de Outubro de 2021. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=History_of_artificial_intelligence&oldid=1050304770

[13] «artificial intelligence - Alan Turing and the beginning of AI», *Encyclopedia Britannica*. https://www.britannica.com/technology/artificial-intelligence (acedido 18 de Outubro de 2021).

[14] «What is Artificial Intelligence (AI)? - AI Definition and How it Works», *SearchEnterpriseAI*. https://searchenterpriseai.techtarget.com/definition/AI-Artificial-Intelligence (acedido 21 de Outubro de 2021).

[15] «What is Artificial Intelligence? How Does AI Work? | Built In». https://builtin.com/artificial-intelligence (acedido 25 de Outubro de 2021).

[16] «Reinforcement learning», *Wikipedia*. 26 de Setembro de 2021. Acedido: 22 de Outubro de 2021. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Reinforcement_learning&oldid=1046562875

[17] «Accuracy, Precision, Recall & F1 Score: Interpretation of Performance Measures», *Exsilio Blog*, 9 de Setembro de 2016. https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/ (acedido 22 de Outubro de 2021).

[18] G. M. Shafiullah, A. Gyasi-Agyei, e P. Wolfs, «Survey of Wireless Communications Applications in the Railway Industry», em *The 2nd International Conference on Wireless Broadband and Ultra Wideband Communications (AusWireless 2007)*, Ago. 2007, pp. 65–65. doi: 10.1109/AUSWIRELESS.2007.74.

[19] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, e F. Ahmad, «Network intrusion detection system: A systematic study of machine learning and deep learning approaches», *Transactions on Emerging Telecommunications Technologies*, vol. 32, n. 1, p. e4150, 2021, doi: 10.1002/ett.4150.
[20] M. F. Elrawy, A. I. Awad, e H. F. A. Hamed, «Intrusion detection systems for IoT-based smart environments: a survey», *J Cloud Comp*, vol. 7, n. 1, p. 21, Dez. 2018, doi: 10.1186/s13677-018-0123-6.
[21] C. Liang *et al.*, «Intrusion Detection System for the Internet of Things Based on Blockchain and Multi-Agent Systems», *Electronics*, vol. 9, n. 7, p. 1120, Jul. 2020, doi: 10.3390/electronics9071120.
[22] «Suricata (software)», *Wikipedia*. 13 de Fevereiro de 2021. Acedido: 24 de Março de 2021. [Em linha]. Disponível em: https://en.wikipedia.org/w/index.php?title=Suricata_(software)&oldid=1006558896
[23] «Metasploit | Penetration Testing Software, Pen Testing Security | Metasploit». https://www.metasploit.com/ (acedido 30 de Março de 2021).
[24] P. Kasinathan, C. Pastrone, M. A. Spirito, e M. Vinkovits, «Denial-of-Service detection in 6LoWPAN based Internet of Things», em *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Out. 2013, pp. 600–607. doi: 10.1109/WiMOB.2013.6673419.
[25] S. M. Danish, A. Nasir, H. K. Qureshi, A. B. Ashfaq, S. Mumtaz, e J. Rodriguez, «Network Intrusion Detection System for Jamming Attack in LoRaWAN Join Procedure», em *2018 IEEE International Conference on Communications (ICC)*, Mai. 2018, pp. 1–6. doi: 10.1109/ICC.2018.8422721.
[26] F. Cuomo, D. Garlisi, A. Martino, e A. Martino, «Predicting LoRaWAN Behavior: How Machine Learning Can Help», *Computers*, vol. 9, n. 3, Art. n. 3, Set. 2020, doi: 10.3390/computers9030060.
[27] F. A. A. Lins e M. Vieira, «Security Requirements and Solutions for IoT Gateways: A Comprehensive Study», *IEEE Internet of Things Journal*, vol. 8, n. 11, pp. 8667–8679, Jun. 2021, doi: 10.1109/JIOT.2020.3041049.
[28] «frmcs_use_cases-mg_7900-v2.0.0.pdf». Acedido: 2 de Dezembro de 2021. [Em linha]. Disponível em: https://uic.org/IMG/pdf/frmcs_use_cases-mg_7900-v2.0.0.pdf
[29] «ts_103389v030101p.pdf». Acedido: 23 de Março de 2021. [Em linha]. Disponível em: https://www.etsi.org/deliver/etsi_ts/103300_103399/103389/03.01.01_60/ts_103389v030101p.pdf
[30] K. Grover, A. Lim, e Q. Yang, «Jamming and anti-jamming techniques in wireless networks: a survey», *IJAHUC*, vol. 17, n. 4, p. 197, 2014, doi: 10.1504/IJAHUC.2014.066419.
[31] «An overview and comparison of free Python libraries for data mining and big data analysis | IEEE Conference Publication | IEEE Xplore». https://ieeexplore.ieee.org/document/8757088/ (acedido 21 de Setembro de 2021).
[32] «What Is the Correlation Coefficient?», *Investopedia*. https://www.investopedia.com/terms/c/correlationcoefficient.asp (acedido 14 de Outubro de 2021).
[33] «Open Source IDS Tools: Comparing Suricata, Snort, Bro (Zeek), Linux». https://cybersecurity.att.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview (acedido 26 de Dezembro de 2020).
[34] «BASE», *SourceForge*. https://sourceforge.net/projects/secureideas/ (acedido 6 de Janeiro de 2021).
[35] «Sguil - Open Source Network Security Monitoring». http://bammv.github.io/sguil/news.html (acedido 6 de Janeiro de 2021).
[36] «Snort - Network Intrusion Detection & Prevention System». https://www.snort.org/eol (acedido 24 de Março de 2021).
[37] «Snort - Network Intrusion Detection & Prevention System». https://www.snort.org/snort3 (acedido 7 de Janeiro de 2021).

[38] «All features», *Suricata*, 25 de Setembro de 2012. https://suricata-ids.org/features/all-features/ (acedido 24 de Março de 2021).
[39] «6.39. Differences From Snort — Suricata 7.0.0-dev documentation». https://suricata.readthedocs.io/en/latest/rules/differences-from-snort.html (acedido 17 de Janeiro de 2022).
[40] «The best IoT Databases for the Edge - an overview and compact guide», *ObjectBox*, 10 de Outubro de 2019. https://objectbox.io/the-best-iot-databases-for-the-edge-an-overview-and-compact-guide/ (acedido 25 de Março de 2021).
[41] «CrateDB: distributed open-source SQL database for IoT & time-series data», *CrateDB*. https://crate.io/ (acedido 21 de Setembro de 2021).
[42] «Classification Algorithm in Machine Learning - Javatpoint», *www.javatpoint.com*. https://www.javatpoint.com/classification-algorithm-in-machine-learning (acedido 3 de Novembro de 2021).
[43] A. Kumar, «KNN Algorithm: What?When?Why?How?», *Medium*, 27 de Maio de 2020. https://towardsdatascience.com/knn-algorithm-what-when-why-how-41405c16c36f (acedido 4 de Novembro de 2021).
[44] R. Kulkarni, «Summary of KNN algorithm when used for classification», *Analytics Vidhya*, 23 de Maio de 2020. https://medium.com/analytics-vidhya/summary-of-knn-algorithm-when-used-for-classification-4934a1040983 (acedido 4 de Novembro de 2021).
[45] «16.1. Lua usage in Suricata — Suricata 5.0.4 documentation». https://suricata.readthedocs.io/en/suricata-5.0.4/lua/lua-usage.html (acedido 26 de Março de 2021).