SAPIENZA
UNIVERSITÀ DI ROMA

Challenges and Opportunities in LoRaWAN Security: Exploring Protocol Vulnerabilities, Privacy Threats and the Role of Edge Computing

Department of Information Engineering, Electronics and Telecommunications
PhD in Information and Communication Technology (XXXVI cycle)

Pietro Spadaccino
ID number 1706250

Advisor
Prof.ssa Francesca Cuomo

Academic Year 2022/2023

Thesis defended on 17 January 2024
in front of a Board of Examiners composed by:

Alessandra Budillon (chairman)

Franco Mazzenga

Piero Tognolatti

Challenges and Opportunities in LoRaWAN Security: Exploring Protocol Vulnerabilities, Privacy Threats and the Role of Edge Computing
PhD thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: pietro.spadaccino@uniroma1.it

Vorrei dedicare questo lavoro di tre anni della mia vita ai miei genitori e a mia sorella per essermi stati sempre accanto, alla prof. Cuomo per avermi dato sempre le giuste direzioni, a tutto il Netlab, al Nesya e pure all'Ispamm, ai mozzi zozzi Zizzo e Francesco, a Federica che mi ha sopportato, ad Alexey che tutto è tranne che uno slabak, a T. perché nonostante tutto mi è impossibile non volerle un mondo di bene.

Abstract

The Internet of Things (IoT) paradigm is nowadays adopted by many applications, involving several communication technologies. In this context, Low-Power Wide-Area Network (LPWAN) protocols are quickly developing, achieving high range communication while maintaining a low energy consumption. Such protocols had to be designed from scratch to cope with the scarce computational resources offered by an IoT device. LoRaWAN is one of the most rapidly expanding LPWAN protocols, thanks to its low cost of devices and operation. During the years, LoRaWAN underwent extensive security assessments, discovering and mitigating new vulnerabilities, and the newest standard version 1.1 is considered to be secure. However, in this thesis we discover new vulnerabilities that could be exploited by a malicious third party to achieve different goals. One vulnerability deals with a Denial-of-Service of downlink messages, impacting application operation as well as network control. The other one deals with the de-anonymization of a device, discovering the correlation between DevAddr and DevEUI, causing information leakages and privacy concerns. Moreover, we consider the role of Edge computing in LoRaWAN and how one can leverage it in order to enhance the security and to optimize network operation in a LoRaWAN environment.

Contents

Chapter 1

Introduction

The Internet of Things (IoT) paradigm is nowadays adopted by many applications and it involves several communication technologies. IoT networks are more and more used to face various aspects of the Information Technology world, such as security, smart industries, smart homes, utilities support, and many others. When deploying smart devices, the most common issues that arise are related to power consumption and signal coverage, for example in a smart city environment we do not want to deploy many access points, an operation that requires a large amount of resources in terms of cost and maintenance. For that reason, the so called Low-Power Wide-Area Network (LPWAN) protocols are quickly developing. Such networks possess features such as long-range signal coverage, low power consumption and low cost devices and maintenance. To reach these goals, it is necessary to use lightweight protocols to exchange data in order to reduce hardware and software complexity and power consumption.

LPWAN protocols have been designed in order to be executed by IoT devices with tight constraints on memory and computation capabilities. For this reason, existing security frameworks such as SSL / TLS are generally not leveraged by such protocols, since they may be too demanding for IoT devices, and security mechanisms and cryptographic schemes have been re-engineered from scratch. This had led to numerous security concerns and vulnerabilities at the time that the first LPWAN protocols were spreading. Nowadays, such communication protocols have undergone extensive security assessments by both the academic and the industry world, and they are generally considered secure.

Nevertheless, research can be still carried out on the topic, in the hope that new vulnerabilities and security threats can be found and disclosed. In the context of LPWAN, one of the most rapidly expanding technologies is LoRaWAN, its main characteristics including low-energy consumption, high-rage communication and low-data rate. For reasons that will be highlighted later in the thesis we decided to focus our research efforts on LoRaWAN technology. After an extensive review of the state of the art for LPWAN and LoRaWAN security, we will show the steps and methods we used to lead our investigation in the right direction, managing to find new security issues affecting the LoRaWAN standard. We will discuss how they emerge and how an operator could mitigate such issues, with an emphasis on the ever-growing paradigm of Edge computing.

The main contributions of this thesis are the following:

- Identification of new challenges that an Intrusion Detection System (IDS) faces when deployed at the Edge of the network;
- Extraction of key performance metrics for LoRaWAN traffic profiling;
- Discovery of a new LoRaWAN vulnerability enabling an attacker to perform a Denial-Of-Service (DoS) attack against a target device;
- Identification of the threats that the newly identified DoS attack poses to the customer and the network operator;
- Development of a new algorithm performing a de-anonymization attack finding the hidden correlation between a DevAddr and a DevEUI of a LoRaWAN device;
- Identification of privacy threats that a user and a network operator are exposed to under a de-anonymization attack;
- Exploitation of the newly identified LoRaWAN vulnerabilities using real world devices and datasets and development of techniques to detect and to mitigate such vulnerabilities;
- Address the relation between LoRaWAN and Edge computing and why it is difficult to integrate the two parts;
- Leverage Edge computing to enhance the security of a LoRaWAN network by introducing an IDS capable of alerting whenever it detects that a de-anonymization attack is possible;
- Leverage Edge computing to perform a distributed radio resources allocation without the centralized control of the Network Server.

1.1 Structure of the Thesis

We start in Chapter 2 by analyzing security methods and architectures specifically designed for the IoT [82]. IoT environments present unique challenges from a security perspective, for reasons such as energy requirements, limited bandwidth or communication protocols which may not work on top of the classical TCP/IP stack. In order to detect and/or mitigate security vulnerabilities in such environments, several IDSs were proposed in the literature. Such IDSs may also leverage an architecture based on Edge computing, in order to improve performance metrics such as response time, energy optimization and cloud security. However, we show that deploying IDSs at the Edge presents some unique challenges and complexities.

In Chapter 3, we introduce LoRaWAN, which will be the main focus of this thesis, as well as the main reasons that led us to the choice of investigating it. Specifically, we take an in-depth look at the current security status of the LoRaWAN protocol, reviewing the state of the art of the disclosed vulnerabilities and security concerns presented in the literature. We show that two main open security issues exist, which

can be grouped into two categories: one deals with the security of the protocol itself and the other related to device profiling and subsequent leakage of information.

In Chapter 4 we analyze the traffic behavior and characteristics of LoRaWAN devices in the wild [81]. To make this analysis general and robust we leverage the use of public datasets. Specifically, we consider the LoED dataset [11], containing millions of packets of devices belonging to different owners and serving different applications. Such dataset was built by passively capturing packets in the air in a large geographic area. The idea is that the dataset will serve us as a general sample of the population of LoRaWAN devices. By doing such investigation, we uncovered two particular behaviors of LoRaWAN devices, reported in Sections 4.3.6 and 4.3.7, which could be exploited by a malicious attacker to achieve different goals.

The first vulnerability is shown in Chapter 5, where an attacker is able to perform a DoS attack against one or more target devices, by making them unable to receive any downlink communications [54], [55]. We discuss in depth such vulnerability, reproduce it using commercial software (targeting our own devices) and show how a network operator can detect and mitigate such an attack.

The second vulnerability is shown in Chapter 6, where we show which information can an attacker learn about by passively listening to traffic generated by devices [83], [84]. In particular, we show how a malicious third party can discover the hidden mapping between DevEUI and DevAddr of a LoRaWAN device, performing a de-anonymization attack.

Finally, in Chapter 7 we introduce the ELEGANT EU research project, introducing a framework to unify IoT, Edge and Cloud applications. By leveraging ELEGANT, we can re-engineer our vulnerability introduced in Chapter 6 as an IDS operating at the Edge of the network [88]. Moreover we develop an edge-enabled distributed algorithm which allocates radio resources in a LoRaWAN network [33]. This will highlight the possibilities that could arise by exploiting Edge computing in a LoRaWAN environment, paving the way for future work based on this thesis.

The overall key findings and results are reported in Chapter 8.

Chapter 2

Security architectures for the IoT

In this chapter we explore security methods and architectures for the Internet of Things (IoT). In order to do so, we focus on Intrusion Detection Systems (IDSs) for the IoT. In this way, we explore the most widely spread class of attacks in the IoT, as well as detection and mitigation techniques. We will then move our analysis on edge computing, where we will observe how it could enhance the possibilities of a correct attack detection. Moreover, we will also identify new challenges that arise when deploying an IDS in an edge environment.

2.1 Intrusion Detection System Taxonomy

IDSs are key components of current cybersecurity methods, where different techniques and architectures are applied to detect intrusions. The goal of an IDS is to prevent unauthorized access to an information system, as such access could potentially jeopardize the confidentiality, integrity, or availability of information. An IDS accomplishes its task by scrutinizing network traffic and/or resource utilization, subsequently issuing an alert when it identifies malicious activity.

IDSs fall into two primary categories, each rooted in a distinct strategy for intrusion detection. The first category entails cross-referencing monitored events with a database containing known intrusion techniques, and it is known as *signature-based* detection. The second category, referred to as *anomaly-based* detection, revolves around learning and monitoring the typical behavior of the system, subsequently flagging any anomalous events that deviate from this norm.

Signature-based IDS

Signature-based IDSs belong to a class of systems that rely on a database containing signatures of documented attacks. These signatures are extracted from ongoing activities, and matching techniques and/or protocol compliance checks are employed to compare these signatures with those stored in the database. If a match is identified, an alarm is triggered. These IDSs can function in both online mode, where they directly monitor hosts and issue real-time alerts, and offline mode, where they

analyze logs of system activities. In the literature, this class of IDS is also referred to as misuse detection, specification-based detection, or knowledge-based detection [48].

One significant advantage of employing such an IDS is its high precision. When well-defined signatures of known attacks are available, the IDS can consistently generate alerts when these attacks occur during system operation.

However, a drawback of the signature-based approach lies in the process of extracting traffic signatures, which can be a laborious and time-consuming task. The complexity of this task depends on the specific traffic features considered and the number of signatures required. In many cases, these signatures are manually crafted by experts possessing in-depth knowledge of the exploits the system is intended to detect.

Furthermore, it's worth noting that signature-based IDSs have limitations when it comes to dealing with zero-day attacks. Zero-day attacks refer to those that have no corresponding signature in the IDS's database. The increasing prevalence of zero-day attacks, as reported by Symantec [87], diminishes the overall effectiveness of signature-based IDSs. Consequently, to address this challenge, a new class of IDSs known as anomaly-based IDSs was developed. These IDSs operate by modeling the typical behavior of a computer system and subsequently alerting administrators to any substantial deviations from this established baseline.

Anomaly-based IDS

Anomaly-based IDSs were developed to address the limitations of signature-based IDSs. Typically, these IDSs include a training phase in which they construct a model of the normal behavior exhibited by the system. Once deployed, they continuously monitor computer hosts and compare their behavior to the established baseline model. If a substantial deviation between the hosts' behavior and the model is detected, the IDS may trigger an alert. This approach potentially equips an anomaly-based IDS with the capability to detect zero-day attacks since it doesn't rely on matching current behavior to predefined attack signatures in a database.

Another advantage of anomaly-based IDSs is their capacity to make it challenging for attackers to understand the normal behavior of a target host without interacting with it. Communication with a target host is likely to trigger alerts from the IDS [49], [53]. Additionally, anomaly-based IDSs can serve as valuable system analysis tools. Anomalies reported by the IDS indicate deviations from baseline conditions, which can signal not only intrusion but also the presence of bugs in a device's logic.

However, it's important to note a significant limitation of anomaly-based IDSs: they tend to generate a higher rate of false positives compared to signature-based IDSs. During operation, the behavior of the monitored system can change slightly or significantly without any intrusion occurring. If an anomaly-based IDS is not aware of these potential variations, it may generate false alerts.

Statistic-based IDS

During the learning phase, an IDS that relies on statistical techniques constructs a probability distribution model of the computing system during its normal, or

nominal, behavior. This model is created by collecting measurements from various parameters and events occurring within the computing system. Once the IDS is deployed, it assesses the probability associated with all the monitored events within the system and triggers alerts when events with low probability occur.

The simplest approach to building this statistical model is known as the Univariate strategy, which involves considering each measurement independently, without taking into account correlations or relationships between them. An advancement from this is the Multivariate strategy, which focuses on identifying correlations and connections between two or more measurements. Hybrid approaches are also feasible. For instance, Ye et al. proposed a hybrid system that combines elements of both the univariate and multivariate approaches. In this approach, profiles are built for each measurement independently, and then multivariate correlations are identified to reduce the false positive rate [98].

When dealing with a large number of measurements, employing multivariate statistical techniques directly on raw data can introduce a significant level of noise. To address this issue, several systems, as described in [57], [19], [13], and [74], have adopted Principal Component Analysis (PCA). PCA is a statistical technique used to reduce the dimensionality of input vectors before applying standard multivariate statistical methods. By reducing the dimensions, PCA helps in simplifying the analysis and mitigating the noise inherent in high-dimensional data.

Another family of statistic-based techniques for anomaly detection leverages time series data analysis, as discussed in [12] and [32]. These techniques have also found applications in network anomaly detection. When calculating the probability of an event occurring, the time factor is taken into consideration. An alert is raised if an event is deemed unlikely to have occurred at a specific time. This time-aware approach enhances the accuracy of anomaly detection by considering temporal patterns and deviations from expected behavior.

Zhao et al. in their work [100] employed techniques for mining frequent patterns in network traffic data. They also incorporated time-decay factors to distinguish between newer and older patterns. This approach enables IDSs to continuously update their system baseline, enabling them to adapt to the highly dynamic behavior of users. By differentiating between recent and historical patterns, these IDSs can better identify emerging threats.

When developing a statistic-based anomaly IDS, particularly one that works with time series data, it's crucial to consider data seasonality. Seasonality refers to the presence of periodic variations in data that occur over months, days, or even hours. These variations can be influenced by human factors like holidays and work hours or other external factors such as weather, depending on the specific application. To address the challenge of detecting outliers in time series data affected by seasonality, Reddy et al. proposed an algorithm [73] that employs a double pass of Gaussian Mixture Models (GMMs). During the learning phase, the time series is divided into seasonal time bins, GMMs are trained, and outliers are removed from the data. To enhance performance, a second set of GMMs is constructed using the cleaned data. These secondary GMMs are then utilized for the final anomaly detection, improving the accuracy of anomaly detection in seasonality-affected time series data.

Machine learning-based IDS

Machine Learning (ML) has found extensive applications in the field of cybersecurity [3]. Various specialized branches of ML have been harnessed to develop anomaly-based IDSs, leveraging machine learning techniques such as data mining [26], deep learning [96] [2], deep reinforcement learning [63], and more recently, adversarial learning [72]. ML-based IDSs utilize machine learning models to automatically learn and characterize the normal behavior of computing systems.

In the design of an ML-based system, the initial step involves identifying the relevant features within the data to be analyzed, and IDS development is no exception to this process [1]. Preliminary efforts typically focus on assessing the quality of traffic features using publicly available datasets and applying baseline ML algorithms. Researchers like Khraisat [48], Bajaj [8], and Elhag [29] have evaluated the significance of dataset features through methods such as Information Gain (IG), correlation attribute evaluation, and genetic-fuzzy rule mining techniques. Based on this evaluation, they eliminate features that contribute low IG or carry redundant information. Subsequently, various ML algorithms such as C4.5 Decision Trees, Naïve Bayes, NB-Tree, Multi-Layer Perceptron, Support Vector Machines (SVM), and k-means Clustering are applied for anomaly detection. Other techniques used for IDS feature selections include Principal Component Analysis (PCA) [77], [22] and Genetic Algorithms (GA) [93].

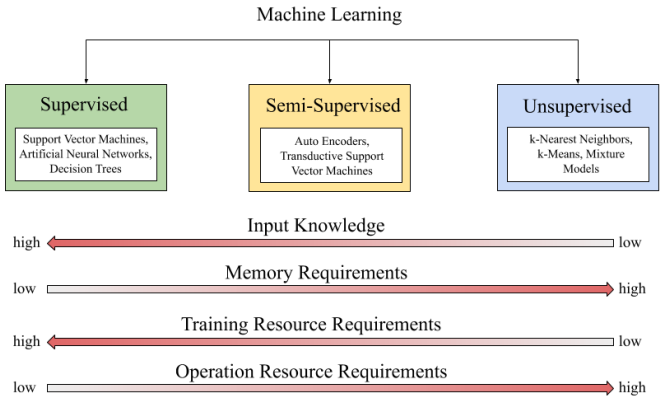


Figure 2.1. General taxonomy of machine learning techniques applied to IDS and their main requirements

Machine learning models can be categorized into two main types based on their learning approach: Supervised Learning and Unsupervised Learning. These approaches differ in terms of the knowledge and requirements needed for model training and operation, as illustrated in Figure 2.1.

- Supervised Learning: This approach requires a significant amount of input knowledge. In supervised learning, the model is trained using labeled data, meaning that the input data is paired with corresponding ground-truth labels. The model learns to make predictions or classifications based on this labeled training data. Supervised learning models tend to have high input knowledge

requirements because they rely on labeled data to build an efficient model. Examples of supervised learning algorithms include decision trees, random forests, and gradient boosting algorithms. These models generally have lower memory requirements as they only need to store the trained model parameters.

- Unsupervised Learning: Unsupervised learning models require less prior knowledge and do not rely on ground-truth labels. Instead, they aim to discover patterns, structures, or relationships within the data without explicit guidance. Unsupervised learning can be further divided into subcategories like clustering and dimensionality reduction. For example, instance-based models like k-means clustering or k-nearest neighbors (kNN) do not have a traditional training phase, but they may require higher memory because they need to store the entire dataset for prediction. On the other hand, models like Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) have a training phase that can be computationally intensive and may require specialized hardware, such as GPUs. However, once trained, they often have lower memory requirements and can make predictions efficiently through matrix multiplication operations.

In summary, supervised learning demands the most input knowledge, while unsupervised learning requires less but may have higher memory requirements depending on the specific algorithm. The computational complexity varies as well, with instance-based classifiers having high prediction complexity and models like ANNs and SVMs being more efficient for making predictions once trained. The choice of which approach and model to use depends on the specific problem and the availability of labeled data, computational resources, and memory.

Specification-based IDS

Specification-based IDSs can be classified as expert systems. These IDSs rely on a knowledge source that defines the expected or legitimate traffic signatures and behavior of a system. In this approach, any event or activity that deviates from this established profile is identified as an anomaly and potentially indicative of an intrusion. The knowledge source used in specification-based IDSs is often manually crafted and curated. It typically includes rules and specifications detailing the normal traffic patterns and behaviors of the systems being monitored. Furthermore, this knowledge source can encompass Finite State Machines (FSMs) that are applied to various Internet protocols like IP, TCP, HTTP, and others. These FSMs are used to ensure the compliance of the monitored host with the specified protocols. By analyzing the sequence of actions and interactions between a host and the network, specification-based IDSs can effectively identify any deviations or violations from the expected protocol behavior. This expert system approach allows for a precise definition of what constitutes normal behavior and enables the detection of any deviations from these predefined norms as potential security threats.

Ensuring protocol compliance via an FSM could be a hard task, since we have to model our state machine on top of the targeted protocol, which can be complex. If otherwise the communication protocol is implemented by writing code, one should

always refer to existing implementations, e.g. open protocol stacks found in the Linux kernel.

Specification-based IDSs have certain limitations, one of which is their vulnerability to Denial-Of-Service (DoS) attacks. DoS attacks can be challenging for specification-based IDSs to detect because they often exploit messages and payloads that conform to the protocol standards, thus not violating the IDS's predefined specifications. In such cases, there may be no apparent violation of the IDS's specifications, and consequently, no alerts are raised. Anomaly-based IDSs, which focus on deviations from normal behavior, can be more effective in identifying DoS attacks, as they can detect unusual patterns of traffic or resource consumption that indicate an attack.

Furthermore, specification-based IDSs may face scalability issues in complex systems involving multiple distinct IoT protocols. In such systems, a separate specification-based IDS may be required for each IoT communication protocol that needs to be monitored. This can lead to increased complexity in the setup and operation of the IDS, as administrators must manage multiple IDS instances and configurations. Anomaly-based IDSs, which can adapt to changes and variations in network traffic more easily, may be better suited for large and diverse IoT environments where multiple protocols are in use.

2.2 Intrusion Detection Systems for IoT

In the context of IoT security, IDSs can be categorized into two main types: IoT-specific IDSs and IoT-agnostic IDSs, each with its own advantages and considerations.

- IoT-Specific IDSs:
  - Target: IoT-specific IDSs are designed to focus on devices that use specific communication technologies, such as 6LoWPAN, Bluetooth Low Energy (BLE), LoRaWAN, etc.
  - Deployment: These IDSs are typically deployed on the same network as the IoT devices they are meant to protect.
  - Detection Approach: They often make predictions based on the messages sent by IoT devices, leveraging control information specific to the chosen technology. This may include checking protocol compliance and monitoring communication patterns.
  - Advantages: IoT-specific IDSs excel at detecting low-level attacks that originate at the device level. They are finely tuned to the nuances of the chosen communication technology, making them effective at identifying technology-specific vulnerabilities.
- IoT-Agnostic IDSs:
  - Target: IoT-agnostic IDSs are not tied to any particular IoT technology; they can monitor traffic generated by devices using various communication technologies.

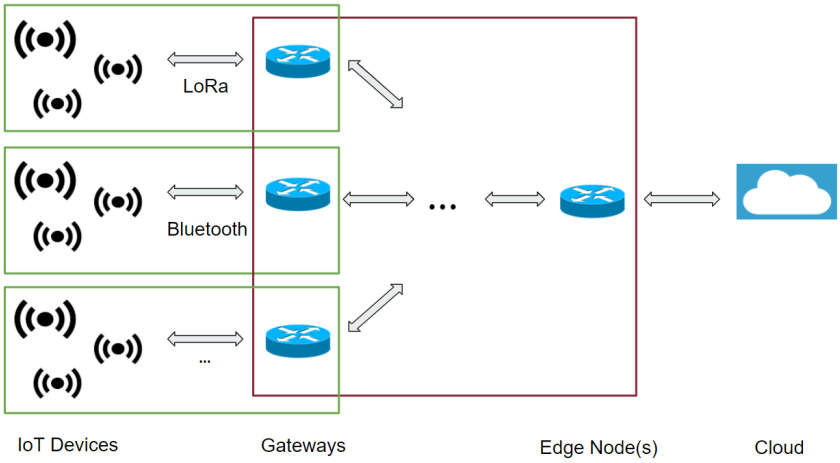


Figure 2.2. Network architecture of an edge-enabled IoT system. Traditional IoT IDSs are deployed at the device or gateway level (green boxes in figure). These systems protect the network against attacks generated by some malicious IoT or non-IoT devices in the specific network. However, the network edge offers new attack surfaces to be exploited by malicious parties. IDSs could be deployed at the network edge (red box in figure). In this case new challenges arise and have to be solved, as a consequence new IDSs specifically designed for the edge should be implemented.

  - Deployment: These IDSs are suitable for deployment in edge environments where heterogeneous devices use different communication technologies.
  - Detection Approach: IoT-agnostic IDSs rely on information available regardless of the specific technology in use, such as TCP/IP traffic analysis and anomaly detection techniques.
  - Advantages: The primary advantage of IoT-agnostic IDSs is scalability and versatility. A single IoT-agnostic IDS can handle multiple IoT devices using different communication technologies, reducing the complexity of deploying and managing multiple IDS instances.

In summary, IoT-specific IDSs offer precision and effectiveness in detecting technology-specific threats but may require multiple deployments for environments with diverse IoT communication technologies. In contrast, IoT-agnostic IDSs provide a more versatile and scalable solution that can accommodate a range of IoT devices and technologies but may not offer the same level of fine-tuned detection for low-level attacks specific to a single technology. The choice between IoT-specific and IoT-agnostic IDSs depends on the specific IoT deployment, its technology diversity, and the level of granularity required for threat detection.

An IoT-specific IDS commonly operates on the network section highlighted in green in Fig. 2.2, while an IoT-agnostic IDS on the one highlighted in red.

The taxonomy of IDSs targeting IoT, as described in Figure 2.3, includes several key characteristics that distinguish these systems in the context of the IoT ecosystem:

- **Device Location and Implementation**. IoT IDSs can be implemented at various levels within the IoT architecture. This includes deployment at cloud

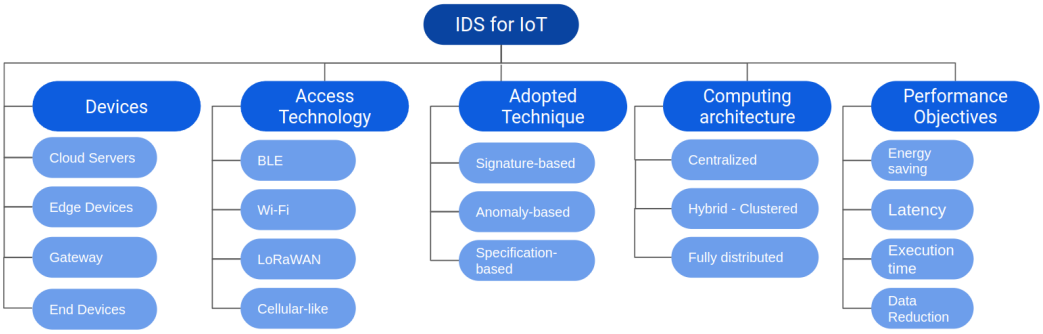


Figure 2.3. Classification of IDS targeting IoT devices.

servers, network gateways, or even at the very edge on end devices themselves. The choice of where the IDS operates affects its scope and capabilities.

- **Access Technologies**. IoT encompasses a wide range of communication technologies, including short-range (e.g., Bluetooth), medium-range (e.g., Wi-Fi), and long-range (e.g., LoRaWAN) technologies. IoT IDSs may be tailored to specific access technologies based on the network's requirements and vulnerabilities associated with each technology.

- **Detection Techniques**. the choice of detection techniques is a critical aspect of IoT IDS design. These techniques can include signature-based detection, anomaly detection, behavior analysis, and machine learning approaches. The selected technique impacts the IDS's ability to identify and respond to threats effectively.

- **Computing Architecture**. the computing architecture of an IoT IDS can vary, ranging from resource-intensive cloud-based solutions to lightweight and resource-efficient edge-based solutions. The architecture determines the processing capabilities and resource requirements of the IDS.

- **Performance Objectives**. different IoT IDSs may have varying performance objectives. Some IDSs prioritize minimizing energy consumption on IoT devices to extend their battery life, while others focus on optimizing latency or reducing the quantity of data exchanged. The choice of performance objectives is influenced by the specific IoT application and its requirements.

IoT-specific IDSs have been developed to address vulnerabilities and security threats associated with various communication technologies in the IoT landscape. Here are some notable examples of IoT-specific IDSs for different communication technologies:

- **Wi-Fi**. In the context of Wi-Fi networks, researchers in [94] have developed neural network-based detection approaches. These techniques were tested on the Aegean Wi-Fi Intrusion Dataset (AIWD) [50] and achieved high accuracies in detecting different attack types, including injection attacks, impersonation attacks, and flooding attacks.

- **LoRaWAN**. For LoRaWAN, a specific vulnerability that can lead to permanent disconnection of end devices from the network was addressed in [21]. The authors proposed a detection technique based on the Hamming distance and Kullback Leibler Divergence (KLD). They tested the system on a testbed and achieved high detection rates with low false positives. Another LoRaWAN-specific IDS, presented in [88], focuses on detecting re-identification attacks that link the DevAddress and the DevEUI of an end device. This IDS employs pattern matching and demonstrates scalability as the number of end devices increases.

- **ZigBee**. In the context of ZigBee networks, the Hybrid Advanced Network Intrusion Detection and Prevention System (HANIDPS) was introduced in [47]. HANIDPS combines specification-based and machine learning-based techniques, targeting the ZigBee protocol. It leverages Q-learning to adapt to the environment and protect devices against new and unseen attacks.

- **Bluetooth**. BLE networks were addressed in [51], where researchers developed a machine learning-based IDS. This IDS is designed to detect known DoS attacks in BLE networks. To overcome the lack of a suitable dataset for these attacks, the authors created a data collection system based on ESP32.

Detecting attacks on the physical network layer (PHY) in IoT environments, such as jamming attacks, presents a unique set of challenges. These attacks can disrupt communication by interfering with the physical signals, making them difficult to detect using traditional methods that rely on monitoring higher-layer protocols. For example, authors in [16] propose an attack on the BLE physical layer which selectively jams the signal on specific channels whenever a device tries to connect.

Rather than concentrating on individual protocols, other approaches have aimed to develop IDSs capable of operating within a unified IoT environment. In the work presented by Hosseinpour et al. [45], a framework for an IDS based on an Artificial Immune System (AIS) is proposed. This IDS is designed to be distributed across various components of the IoT ecosystem, including IoT devices, edge nodes, and cloud nodes. Specifically, on IoT devices, the framework deploys lightweight detectors. At the edge layer, alerts generated by IoT devices undergo analysis and processing using smart data concepts. Finally, the cloud layer is responsible for data aggregation and the training of the detectors. In this architecture, the training of heavyweight detectors' models occurs in the cloud, while IoT devices only perform lightweight model applications. This design minimizes the computational load on IoT devices, ensuring their efficient operation within the unified IoT environment.

Like other expert systems, IoT-specific IDSs typically attain remarkable precision and maintain a low rate of false alarms. These systems are often tailored for identifying particular, clearly defined attacks. However, they are not equipped to detect zero-day vulnerabilities or atypical patterns of network resource usage by connected devices. In contrast, IoT-agnostic IDSs operate independently of the communication technology employed by IoT devices. These IDSs can be deployed on IoT gateways, disregarding physical (PHY) or Media Access Control (MAC) layer details, or in alternative subnetworks where they make use of TCP/IP traffic characteristics.

Moreover, in the IoT there are specific and emerging security concerns that represent a challenge to the management and operation of networks. These challenges encompass areas such as routing, topology control, and network maintenance. Regarding routing, new protocols have been developed to cater to devices with limited resources. An example of such a protocol is the Routing Protocol for Low Power Lossy Network (RPL) [20]. RPL establishes a communication framework based on the concept of a Destination Oriented Directed Acyclic Graph (DODAG), which is constructed by IoT devices following the protocol. This DODAG structure facilitates various types of communications, including point-to-point, point-to-multipoint, and multipoint-to-point. In RPL networks, certain known attacks aim to manipulate routing processes, potentially leading to suboptimal routes and reducing the lifespan of devices. These attacks can also disrupt the overall routing of the network, for instance, by causing black holes in data transmission. In response to the deployment of the RPL protocol, numerous IDSs have been proposed: Mayzaud et al. introduced an IDS in their work [59] that specifically targets attacks related to the version number of RPL. Their solution employs a distributed monitoring architecture in which multiple detection algorithms are deployed. Furthermore, this IDS has the capability to not only detect but also pinpoint the malicious node responsible for carrying out the attack. In another study [18], the authors proposed an IDS that utilizes neural networks to combat attacks involving version numbers, worst parent selection, and hello flood attacks in RPL networks. What sets this IDS apart is its consideration of not only routing-layer features but also link-layer features. This inclusion of link-layer features proves particularly beneficial in reducing false positives, especially when dealing with version number attacks.

2.3 The Edge-Enabled Approach

Edge computing has emerged as a promising approach to enhance the characteristics and reliability of traditional IoT applications, as noted in [76] and [44]. This paradigm allows IoT applications to delegate computational, storage, or management tasks to edge nodes, leading to various improvements in quality.

One of the key benefits of edge computing is the reduction of latency, which is critical for real-time applications. Additionally, it enables real-time network management and more efficient data handling. In this context, security applications, such as IDSs, can be moved to the edge nodes, as indicated by the red box in Figure 2.2. This transition offers several advantages for IDSs:

- **Increased Computational Resources**. Edge nodes typically have more computational power than IoT devices. This allows IDSs to employ more complex algorithms and carry out resource-intensive tasks effectively.
- **Enhanced Storage Capabilities**. Edge nodes can provide additional storage capacity, enabling IDSs to store system logs for later analysis or perform memory-intensive operations.
- **Lower Latency**. Edge nodes can offer lower latency compared to cloud-based solutions, which is crucial for real-time IoT applications where rapid threat detection and response are essential.

Eskandari et al. [30] introduced the Passban IDS, a system designed to enhance the security of IoT devices directly connected to it. This IDS primarily focuses on defending against TCP/IP-related attacks and does not address attacks specific to IoT technologies, such as Port Scanning, HTTP and SSH brute force, and SYN flood attacks. A notable feature of the Passban IDS is its efficiency. It does not require extensive computational resources and can be deployed on cost-effective edge devices and IoT gateways, such as Raspberry Pis or similar hardware. Despite its focus on a relatively limited set of attacks, the system exhibits a remarkable performance with a very low false positive rate and high accuracy. One particularly positive aspect of the Passban IDS is its comprehensive implementation. It encompasses the entire spectrum of IDS functionality, from the detection algorithms to the alerting system, complete with a user-friendly web interface. This holistic approach simplifies the deployment and management of the IDS, making it a valuable tool for enhancing the security of IoT devices in a user-friendly manner.

Terenzi et al. [88] have devised an IDS architecture tailored specifically for LoRaWAN devices, as illustrated in Section 7.3. The primary objective of this IDS is to notify network operators if an attacker manages to determine a device's DevEUI solely by monitoring application traffic and extracting the DevAddr from the transmitted packets. The proposed technique relies on the assumption that devices send application packets periodically. It capitalizes on the temporal correlation between application and join request messages to establish an exact match between a DevAddr and DevEUI. This approach allows the IDS to identify potentially malicious activities related to the discovery of device identifiers. One key advantage of this IDS architecture is its ability to operate at the edge of the LoRaWAN network, specifically on LoRaWAN gateways. These gateways serve as critical network entry points, making them an ideal location for monitoring and detecting such attacks. Additionally, it's worth noting that this IDS does not necessitate packet decryption. Instead, it makes predictions based solely on packet arrival timings, making it an efficient and privacy-preserving solution for safeguarding LoRaWAN networks against certain types of attacks.

Sandhu et al. [78] have analyzed the identification of potentially malicious edge devices, which play a crucial role in storing and processing data generated by numerous IoT devices. If an attacker gains control over such an edge node, they could potentially manipulate or intercept the data transmitted by the connected IoT devices. To address this security concern, the authors have proposed a comprehensive framework comprising several components. First, the framework utilizes a two-stage Markov Model to analyze and process the data. In the first stage, the model categorizes the specific fog (edge) node that is under scrutiny. In the second stage, it predicts whether or not a Virtual Honeypot Device (VHD) should be attached to the edge node for which an alert has been triggered. Then, the IDS is employed within the framework to monitor the behavior of edge devices. When the IDS detects unusual or potentially malicious activity, it raises an alert. When an alert is generated by the IDS, the framework decides whether to attach the VHD to the specific edge node associated with the alert. The VHD acts as a decoy or bait, emulating a vulnerable device that may attract malicious activity. It stores logs of all connected edge nodes, allowing security experts to later investigate and analyze potential threats.

Pandeeswari et al. [65] introduced a system to improve the detection accuracy of an IDS by using a combination of fuzzy c-means clustering and ANNs at the edge. They claim that their approach outperforms traditional ANN techniques, especially in detecting attacks with a low frequency.

Hafeez et al. [41] introduced a novel system designed for anomaly detection at network edge gateways. This innovative system employs feature representations of network traffic that are technology-agnostic, focusing solely on TCP/IP attributes that can be observed at the network's edge. One significant advantage of this approach is its compatibility with diverse IoT communication technologies, allowing multiple systems, each utilizing different communication protocols, to be seamlessly integrated into the same IDS. Regarding their dataset, the researchers gathered IoT data from a real-world testbed. Furthermore, they conducted an extensive analysis of the distribution patterns exhibited by the various considered features. Remarkably, their observations revealed that a significant portion of these features follows a heavy-tailed Gaussian distribution. The ultimate step in their anomaly detection process involves the utilization of fuzzy clustering techniques. Employing this method on their custom dataset, the authors achieved remarkable results, demonstrating both high accuracy and a notably low false positive rate.

Schneible et al. [79] introduced a framework aimed at facilitating distributed anomaly detection on edge nodes within a network. This innovative system involves the deployment of auto-encoder models across multiple edge nodes strategically positioned in various network regions. The core of the anomaly detection process relies on the conventional auto-encoder approach. What sets this system apart is its adaptivity. As the edge nodes operate, they continuously update their models based on newly observed data, enabling them to identify emerging trends within network traffic. Subsequently, an edge node transmits its updated model to a central authority, which aggregates these updates and disseminates them to other edge agents. One noteworthy benefit of this approach is a significant reduction in bandwidth overhead. Instead of transmitting all observed data, the network traffic generated primarily consists of the auto-encoder models. In this context, auto-encoders serve a dual purpose by not only detecting anomalies but also automatically extracting and compressing features from observed data. This feature compression minimizes the volume of data exchanged between the edge nodes and the central authority.

While edge nodes possess superior computing capabilities compared to IoT devices, their capacity to perform resource-intensive tasks, such as heavyweight machine learning model training, is limited. Recognizing this challenge, prior research has explored alternative systems that do not demand such intensive operations. Sudqi et al. [85] proposed a host IDS designed to run on energy-constrained devices. Sedjelmaci et al. [80], on the other hand, put forth a more sophisticated system that strikes a balance between energy consumption and detection accuracy. Their system comprises two components: a signature-based IDS, known for its energy efficiency but prone to generating a higher number of false positives, and an anomaly-based IDS, which consumes more power but conducts a more precise analysis. During regular operation, only the signature-based IDS remains active. When an alert is triggered, it is then forwarded to the anomaly-based IDS for confirmation or dismissal. Furthermore, this system is formulated as a security game model, where the anomaly-based IDSs base their predictions on the Nash Equilibrium, a key

concept in game theory. However, it's worth noting that one drawback of this system is its dependency on a continuously operational cloud infrastructure for optimal functionality.

Anomaly detection techniques have a broader utility beyond just identifying network intrusions. They can serve as valuable tools for detecting various irregularities, including bugs in devices' firmware and deviations from the normal state of a system. In the realm of Industrial Internet of Things (IIoT), there has been notable research and development aimed at leveraging these techniques to detect such anomalies effectively.

Utomo et al. [92] have developed a system tailored for performing anomaly detection on sensor readings within power grids. The alerts generated by this system serve a dual purpose, not only indicating potential illegal intrusions but also contributing to grid safety by preventing failures and blackouts. To execute the anomaly detection effectively, the system harnesses the power of ANNs, specifically utilizing Long-Short Term Memory (LSTM) cells. LSTM neural networks are a subset of Recurrent Neural Networks (RNNs), a category of ANN architecture renowned for its prowess in processing sequential data. In the context of power grid sensor readings, which often exhibit high non-linearity, the LSTM-based approach proves to be particularly valuable in identifying anomalies and ensuring the stability and reliability of the grid. This parallels the use of RNNs, such as LSTMs, in Natural Language Processing (NLP) to process sequences of words.

Niedermaier et al. [64] made an observation that a single IDS deployed at the network perimeter may struggle to effectively monitor, capture, and analyze all network events. To address this limitation, they proposed a distributed IDS architecture built around multiple IIoT agent-edge devices, in conjunction with a central unit that consolidates and manages the logs generated by these agents. At the heart of this IDS lies an anomaly detection approach rooted in one-class classification techniques. The authors operate under the assumption that they possess knowledge of the system's normal behavior, a pattern that can be learned by the IIoT agents. What sets this IDS apart is its suitability for deployment on low-power micro-controllers, as it doesn't necessitate resource-intensive computations. Moreover, the authors have taken an exceptional step by developing a proof-of-concept implementation of their system, a practice not always seen in similar research works.

Hafeez et al. [42] introduced a lightweight technique named IOT GUARD, designed for distinguishing between malicious and benign IoT traffic. Their approach primarily adopts an unsupervised methodology, but it incorporates a semi-supervised aspect by requiring a small subset of labels to be manually verified, which technically classifies it as semi-supervised. The foundation of IOT GUARD relies on Fuzzy C-Mean (FCM) clustering. To enhance its performance, the authors employed an aggregation strategy that combines features of devices belonging to the same host and offering the same service. Unlike time-based aggregation, which aggregates features over specific time intervals (e.g., the number of connections in the last T seconds between devices A and B), their connection-based aggregation strategy operates over the n most recent device connections. This approach offers an advantage in scenarios where attackers introduce time delays between successive connection attempts, as it accommodates such variations. It's important to note that the evaluation of IOT

GUARD was conducted using a proprietary dataset, not accessible to the public. While the achieved accuracy was favorable, the research did not provide a practical comparison with other existing solutions or baseline algorithms.

2.4 Device Classification at the Edge

Recent efforts have been focused on the identification and classification of IoT devices by analyzing their network traffic fingerprints. By examining network packets, it becomes possible to develop classifiers that can categorize devices according to their specific device class (such as motion sensors, security cameras, smart bulbs, and plugs) or to learn and establish unique device signatures.

Creating these signatures is a fundamental step in constructing an IDS because they serve as a reference point for comparing known signatures against those extracted during system operation. Consequently, if unauthorized devices attempt to connect to the network or if existing devices suddenly alter their behavioral signatures, an IDS can swiftly generate alerts and trigger security responses. The capability to detect intrusions exclusively based on network traffic is a crucial requirement for IDSs designed to be deployed at the network edge.

In this context, a device's signature or fingerprint represents the unique characteristics of the network traffic it generates. Multiple combinations of features can be employed to construct these signatures. To discern which features are more suitable than others for this purpose, Desai et al. [24] devised a feature-ranking system tailored for IoT device classification. The assessment of each feature's utility is grounded in statistical methods. To extract these features from traffic flows, they adopted a time window approach, using intervals of 15 minutes. Within these windows, they identified a sub-portion referred to as the activity period. This activity period spans from the reception of the first packet to the reception of the last packet for each device and can exhibit varying lengths depending on the device class. In their experimentation, they trained classifiers using two different feature sets: all available features and only the top-k ranked ones. Their findings revealed that classifiers trained with just the top-k features (where $k = 5$) exhibited only a relative drop in accuracy of approximately 6%. This indicates that a substantial reduction in computational tasks can be achieved without significantly impacting the accuracy of the classification process.

Thangavelu et al. [89] introduced a distributed device fingerprinting technique named DEFT, designed for recognizing IoT device fingerprints. In this system, IoT gateways are responsible for extracting features from the traffic sessions of connected devices. These extracted features are then transmitted to central edge nodes, which collect and utilize them to train machine learning models and classifiers. Subsequently, these classifiers are sent back to the gateways, which perform the final identification of the devices. One remarkable feature of DEFT is its ability to autonomously recognize new devices without prior knowledge of their traffic signatures. When a new device joins the network or an existing device alters its usual traffic pattern (e.g., due to a firmware update), the classifier on the gateways flags the traffic as having a low probability of belonging to a known device class. In such cases, the gateway transmits the captured features to the edge node. If another

device of the same unknown class (i.e., with the same traffic signature) connects to a different gateway, that gateway also sends its captured features to the edge node. At this point, the edge node can cluster and identify the new device category. However, if there isn't a second device of the same class connecting to another gateway, this strategy may not work. The entire system can be managed as a Software Defined Network (SDN) function, enhancing its flexibility and adaptability. Furthermore, the classification process is conducted on a flow-wise basis rather than a packet-wise basis, which helps optimize resource utilization, making it a cost-effective solution in terms of computational resources. Indeed, the DEFT technique, which extracts fingerprints of IoT devices based on their network traffic patterns, can serve as a fundamental building block for an anomaly-based IDS. An IDS constructed using these device fingerprints can be effectively deployed at the edge of the network, since the final application of the trained models is done at the gateway level.

Bai et al. [6] have introduced an innovative device classification technique specialized in identifying new and previously unseen devices. This approach distinguishes itself from the majority of other methods, which typically require prior training data for each specific device to be recognized. This novelty is particularly valuable for IDSs with the primary objective of tracking and alerting when new or unauthorized devices connect to the network. The classification process in this technique relies on information streams generated by devices and employs an LSTM-CNN model that effectively accounts for time dependencies within network traffic. Traffic data is divided into fixed time windows, each of length T seconds. The specific value of T used in their experiments is not explicitly mentioned in the paper, but it appears to be a fixed, non-adaptive parameter. Features are then extracted to differentiate between incoming and outgoing packets, as well as between user-level packets (e.g., TCP, UDP, MQTT, HTTP) and control packets (e.g., ICMP, ARP, DNS). Various statistics of the packets are computed. The processed data is finally fed into an LSTM network, which learns an encoding of the data. This LSTM network is then connected to a CNN (Convolutional Neural Network) for final classification. The achieved results from this approach demonstrate a commendable level of accuracy, although there is room for further improvement.

2.5 Challenges for Edge-enabled Architectures

The architecture of an edge-enabled IoT application, as illustrated in Figure 2.2, introduces new attack surfaces that can be exploited by malicious actors. While traditional IoT-oriented IDSs are typically placed at the gateway or device level with a focus on protecting against malicious IoT devices, it's important to recognize that attacks can be directed specifically at the edge network. This scenario raises the possibility of an edge node becoming compromised, either through remote attacks or physical tampering. In situations where edge nodes are deployed in public areas, the physical accessibility of these nodes makes them susceptible to tampering by attackers. When an attacker gains control of an edge node, they have the potential to manipulate all the traffic passing through it. This manipulation can take various forms, including generating deceptive packet streams within the edge, masquerading as a legitimate IoT gateway or device, or selectively forwarding packets of interest

while discarding others.

Already existing IDSs could be used and deployed in an edge environment, however it is possible that detection algorithms have to be re-engineered to work in a distributed way while not having the full picture that an IDS on the cloud level has. In this context, some new challenges arise and hinder the reliability of the intrusion detection system. They include:

- **Traffic Encryption.** When an IDS is deployed on IoT gateways or more external edge nodes, it encounters a scenario where the observed network traffic is often encrypted. This assumption holds true when IoT devices and the cloud communicate via secure protocols. Additionally, the same encryption scenario can arise when the IDS is deployed on IoT gateways, and the IoT devices possess a TCP/IP stack, enabling direct communication with the cloud. In such cases, the role of the gateway is primarily routing, and packet-level encryption is still in place. Packet encryption presents a significant challenge for IDS operations because it renders the actual content of the packets unreadable to the IDS. Instead, the IDS can only perform operations based on non-encrypted fields, such as TCP/IP headers, timestamps, source and destination IP addresses, and other metadata.

- **High Resource Variability.** IDSs can leverage several techniques to carry out the detection, which can be highly variable in terms of required computational resources. However, also edge nodes show high computational resource variability, which could range from a commodity PC with specialized hardware to a Raspberry Pi. The problem that may arise is that the requested resources for the IDS to work are too high for the edge node which is running the system, which could add communication latency and could block the whole system execution. On the other hand, an edge node that offers a lot of resources costs more, and if the resources are not exploited by the IDS the extra cost is wasted. Edge IDSs should be adaptive to the available resources, using a variety of algorithms requiring different capabilities and selecting them based on the current execution platform.

- **Distributed IDS architecture on Edge/IoT.** Due to the resource variability between the IoT and the edge, the execution of IDS for the network edge should be somewhat distributed. A single IDS could be composed of many subsystems which cooperate for the correct working of the system or to improve the detection performance. The cooperation of different subsystems, however, brings distributed systems challenges into the intrusion detection system, increasing its complexity.

- **Aggregated traffic.** If the protocol stack of IoT devices and the protocol stack of the edge differ, it could make an observer on the edge, including an IDS, unaware of the source end device of a packet. This is caused by IoT gateways which receive packets from end devices using their specific IoT communication technology and craft new packets using the protocols of the network edge, such as TCP/IP. This issue and its aftermath will be illustrated in more detail in Section 2.5.1.

In order to develop communication schemes which are resilient to malicious edge nodes theory of distributed systems could be leveraged, treating edge nodes as potentially byzantine nodes [27] and treating each packet that goes through the edge as a byzantine consensus problem. However, theorems [27] state that, in a non-authenticated and partially synchronous communication scheme, it must hold $N > 3f$, where N is the number of parties and f is the maximum number of tolerated byzantine nodes, in order for a byzantine consensus to be successful. This, however, would require a transmission of the same packet from multiple edge nodes. Moreover, if the packet was originated from an IoT device, it would require the same device to send the same packet to multiple edge nodes, representing a waste of energy and network resources, which could be unacceptable in a power-constrained IoT environment.

Leveraging IPsec (Internet Protocol Security) [34] is a valuable solution for the aforementioned problems, providing essential security features such as authenticity, integrity, and encryption (in ESP mode) to both packet headers and data, helping to mitigate the risks posed by malicious edge nodes. However, some challenges and considerations still persist:

- **IPsec does not protect against traffic rerouting or selective-forwarding.** Attackers could decide which packets to forward and which ones to discard (selective-forwarding). They could also route the packets with additional delay, which can impact the real time characteristic of the IoT application.
- **IPsec fails to guarantee security specification in a physical tampering scenario.** If a device gets tampered, attackers have the possibility to access the private keys of an edge node, compromising the whole IPsec secure communication scheme for the device.
- **IPsec increases fractional overhead.** In a usual IoT application, packets sent from IoT devices are a few bytes long, meaning a low ratio of payload data over header data. The use of an additional control header increases even more the payload data, making the communication even more inefficient in terms of fractional overhead.

2.5.1 Aggregated Traffic

Another problem is that the edge may not have the possibility to differentiate the traffic flows coming from the IoT devices, in other words it could only observe the aggregated traffic generated by all the devices combined as if it was generated by a single device. This issue verifies whether IoT devices and the edge have different protocol stacks and the gateway has to translate the protocols used by the IoT to the ones used by the edge/cloud. The observation of the aggregated traffic will cause both signature-based and anomaly-based IDS to carry out unreliable predictions.

Let us consider the scenario depicted in Fig. 2.4. We have IoT devices connected to the gateways with some IoT specific communication technology (BLE, LoRa, etc.) and the gateways connected to the edge and the cloud via TCP/IP. When the IoT devices send data to the cloud, they send a packet to their gateway using their IoT communication technology. The gateway then crafts a new TCP/IP packet and

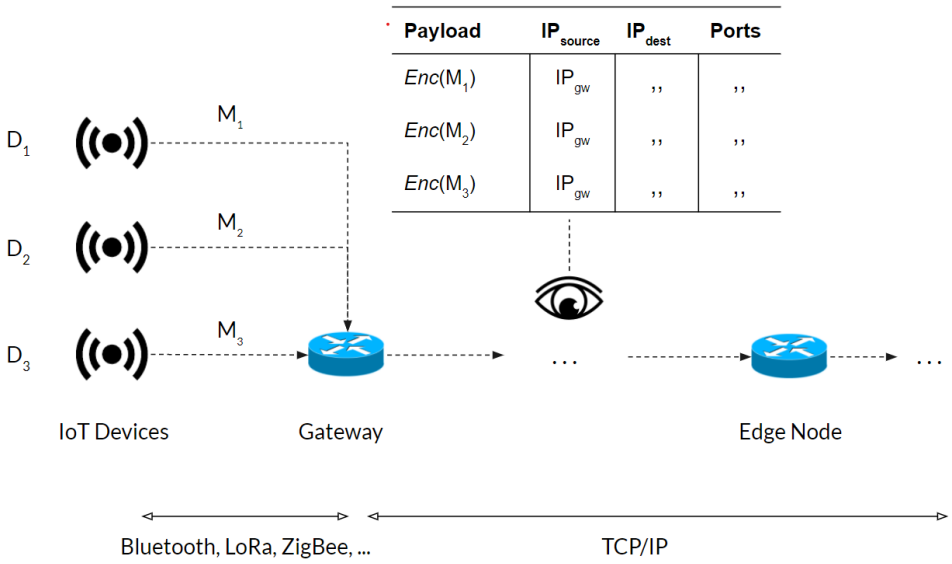
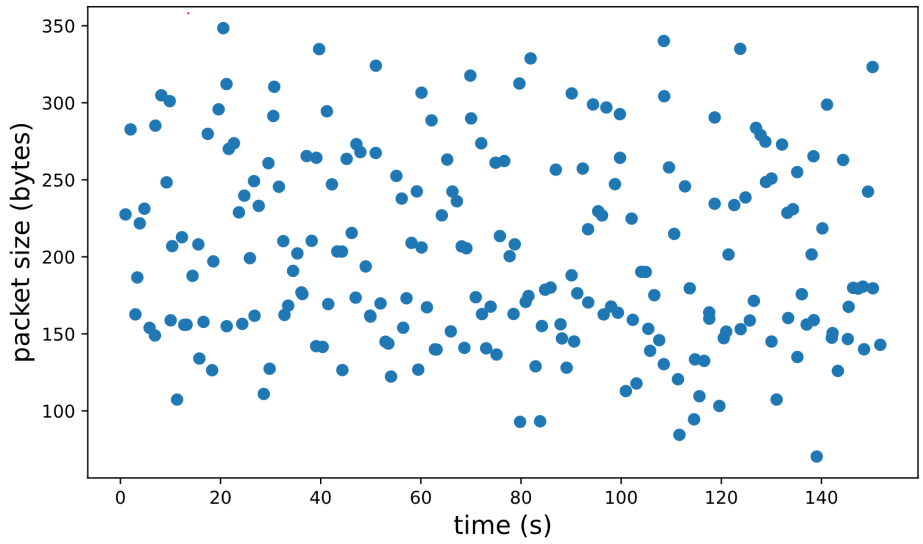


Figure 2.4. An example of why the edge node may be able to observe only the cumulative traffic, thus being unable to identify the end device which generated the observed packet. Some IoT devices send their data to the cloud. They first communicate to their gateway using their specific IoT communication technology. The gateway then crafts packets which will be sent to the edge and forwarded to the cloud, assuming using TCP/IP as protocols. The packets crafted by the gateway will have the same source IP, possibly the same destination IP (the same application server) and could use the same TCP ports. This causes any observer after the gateway, including an edge node, to be unaware of the devices behind the gateway. The edge node is only able to observe the cumulative traffic, without being able to identify the source device of an observed packet.

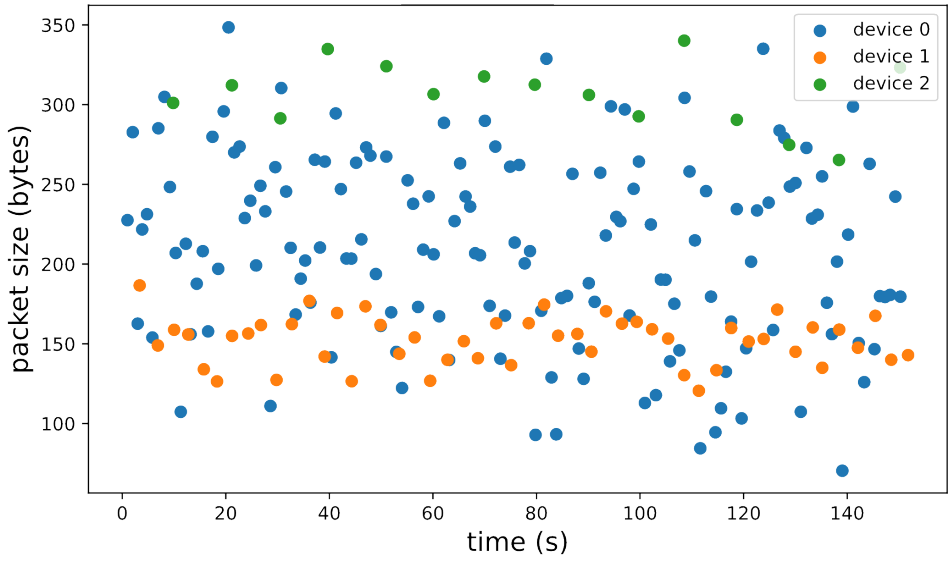
forwards it to the edge and to the cloud. This newly created packet by the gateway will have as source IP address the one of the gateway, regardless of which IoT end device produced it. Moreover, these packets could have the same IP destination address (same application server) and could use the same TCP ports for every IoT end device. This causes any observer beyond the gateways, including the edge nodes, to be unable to tell the source device of an observed packet. Being unable to separate the TCP flows, the edge node would regard the observed traffic as it was generated by a single device, since it has no means of knowing which devices are connected beyond the gateways.

The aggregated traffic poses problems for existing IDSs, both signature-based and anomaly-based:

- Signature-based IDSs cannot isolate packets coming from or going to the same device. This causes the inability to extract patterns from the observed traffic stream, thus making an IDS unable to recognize an attack signature. Methods could be developed to adapt existing signature-based IDSs to solve this issue, for example by mining patterns from the cumulative traffic. However, since the observed traffic is the sum of the various traffic streams generated by each device, there could be cases where a signature could be mistakenly marked as malicious. For instance, let's consider a pattern which is malicious only if



(a) *Unlabeled packets*



(b) *Labeled packets*

Figure 2.5. Consider three IoT devices. Each device produces packets with its own mean length, its own mean time between them and own variances. In the plots, each dot represents a network packet. The edge is not able to tell which IoT devices are connected and therefore it cannot assign a packet to its most likely IoT producer device. So what the edge observes is an unlabeled flow of packets, in Fig. 2.5a, not knowing the source/destination device of a packet. Applying anomaly detection strategies on the cumulative traffic yields poor performance, since too much variance is experienced by the algorithms. In Fig. 2.5b is depicted the same traffic but with packets labeled with its producer device. Applying anomaly detection on the labeled flow, should help algorithms to improve detection accuracy.

IDS Approach	Effects of Cumulative Traffic Observation	Result
Signature-based	Unable to reliably extract signatures from cumulative traffic	Increase of false positives.
Anomaly-based	Anomaly detection algorithm experiences too much variance	Increase of false negatives.

Table 2.1. Summary of the expected issues that causes the observation of the cumulative traffic on the edge by an IDS. In the case of signature-based IDSs, the system is not able to extract precisely patterns and signatures from the traffic, ultimately increasing the ratio of false positives. In the case of anomaly-based systems, their algorithms would experience too much variance during the learning phase. This will cause an inexact anomaly report with a high ratio of false negatives.

generated by a single device (e.g. a particular exchange of messages between it and the server). If two or more devices generate non-malicious messages, it could be that when mining attack patterns, the sum of these flows generates a signature match. This increases the ratio of false positives.

- Anomaly-based IDSs will have to deal with the high variance of the aggregated traffic, since it is presumable for the cumulative traffic to have a higher variance than the traffic flows generated by each single IoT device. To carry out anomaly detection, an IDS has to learn the state of a system in a normal condition i.e. without an anomaly taking place. Then an anomaly is reported when the observed state deviates substantially from the expectation. If the normal state is learned via the cumulative traffic, too much variance could be experienced by the anomaly detection algorithm. The higher variance poses the risk that malicious anomalies are marked as non-malicious oscillations of the expectation, since these oscillations are acceptable given the variance of the normal system state. This increases the ratio of false negatives.

Table 2.1 summarizes the effects of the cumulative traffic on existing anomaly-based and signature-based IDSs.

An example of anomaly detection on the aggregated traffic is illustrated in Fig. 2.5. An anomaly detection algorithm deployed at the edge, should learn the normal system behavior from the cumulative network traffic instead of device-wise traffic. However the cumulative traffic presents more variance than the traffic split in a device-wise manner, which could drastically impact the performance of the anomaly detection strategy. One first step to improve anomaly detection at the edge, could be to split the cumulative traffic into flows, one for each IoT device. Once this split is done, existing algorithms could be used to learn the normal behavior of the system, not from the cumulative traffic but from the flows of each device. However, this task could not be carried out by an edge node alone, since it doesn't have the knowledge of which IoT devices are connected beyond the gateways.

Chapter 3

LoRaWAN technology

After taking into consideration several different IoT communication protocols, we identified LoRaWAN to be the focus of our research. LoRaWAN (LoRa Wide Area Network) is a low-power, wide area networking protocol built on top of the LoRa radio modulation technique. It wirelessly connects IoT devices to a central server and manages communication between end-node devices and network gateways. The use of LoRaWAN in industrial spaces and smart cities is growing thanks to its low cost, long-range and bi-directional communication, with very low power consumption.

In this Chapter we will delve into LoRa and LoRaWAN technologies. First we will analyze the LoRa physical modulation. Then we will move to the principal aspects of the LoRaWAN protocol, with a focus on the security of the protocol itself.

3.1 Why LoRaWAN

The reasons why we decided to focus on LoRaWAN technology are the following:

- LoRaWAN is one of the most rapidly expanding IoT technologies nowadays. This is due for reasons such as its reduced cost, since it operates on unlicensed ISM bands, and its high energy efficiency, being suitable to be applied to common IoT applications such as smart sensing.

- LoRaWAN protocol and its specification are open. This means that one can precisely know the actual bytes sent by a device. Moreover, one can program its own LoRaWAN device in order to perform research and test solutions on the protocol itself.

- The openness of the standard enables in-depth information security research, which would be harder on a black-box protocol. Moreover, LoRaWAN has recently transitioned from the legacy 1.0 version [60] to the newer 1.1 version [61]. The new version brings several security fixes and enhancements. Therefore, we wanted to explore the novelties introduced in the protocol to perform security research and to (try to) find any undisclosed vulnerabilities in the standard.

- LoRaWAN has the potential to be blended with the concept of edge computing. Indeed, the architecture of LoRaWAN already foresees hardware placed at the

edge of the network, the so called gateways. However, nowadays the gateways don't do any useful operation for the execution of the protocol, being simple packet forwarders between the devices and the core network. One can envision adaptations of the protocol for the gateways to perform management operations for the network, assisting new applications leveraging edge computing, as we will see in the Chapter 7, Section 7.4.

3.2 LoRa

LoRa (LongRange) is the physical layer developed by Semtech [69], enabling the LoRaWAN end-device to achieve long-range communication through radio frequency modulation. LoRa modulation is based on chirp spread spectrum (CSS) [66]. It offers immunity to multi-path and tolerate small frequency offset caused by Doppler effect, making LoRa ideal for communication in urban environments. A chirp is characterized by a frequency excursion all over the bandwidth. The base chirp, or up-chirp, is characterized by linear increasing frequency, starting from the value f_{min} and ending with a value f_{max}. Instead the chirp that starts with frequency f_{max} and ends with f_{min} is called down-chirp. For any digital bit sequence the LoRa device will produce a time shifted chirp to the respect of the base-chirp. In the CSS modulation the number of different symbols/chirps depends on the Spreading Factor (SF), $N_{Symb} = 2^{SF}$. The value of the SF is also equal to the number of encoded bits per symbol. For example a modulated signal with SF7 can produce 128 distinguishable symbols each of one encoding a different sequence of 7 bits. Symbols are organized in frames (also named packets in the following). The frame structure is reported in Section 3.3.1. Packets with different spreading factors are orthogonal, appearing as a noise to each other. Therefore, two packets that arrive at the same time on the same receive channel at different spreading factors will not collide and, both will be demodulated by the gateway. However, two packets with the same spreading factor arriving at the same time on the same channel result in a collision. According to LoRa specification the SF may assume 6 different values, from 7 to 12. The bit rate R_b depends on the SF. Given a bandwidth BW and a coding rate CR, the R_b is given by the following expression:

$$R_b = SF \cdot \frac{BW}{2^{SF}} \frac{4}{4+CR} \text{bit/s} \tag{3.1}$$

The higher the SF the lower the bit rate. If the same message is transmitted with two consecutive values spreading factors, e.g., 7 and 8, the time needed to transmit the message roughly doubles. The LoRa modulation characteristic vary for each region. In Europe, there are channels with bandwidth of 125 kHz and 250 kHz. LoRa transmits in the ISM band that in Europe is set in the frequency range 863–870 MHZ.

In our experiments, we tested the range of LoRa devices. By taking a walk in the center of the urban area of Rome, we broadcasted packets and observe which gateways will receive that specific packet, as can be seen in figure 3.1. In our tests the maximum range achieved was roughly 27km.

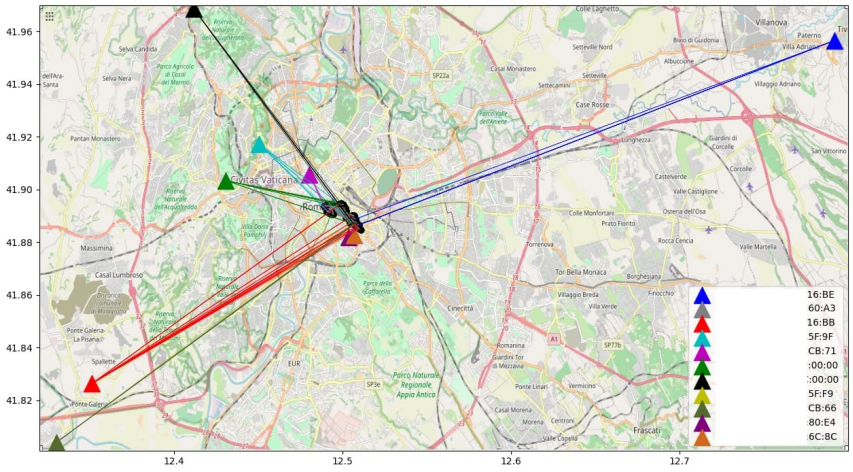


Figure 3.1. We sent packets in different urban location. In the figure are reported the position from which we sent a packet (black dots in the middle of the figure) and the approximate positions of the gateways receiving that packet (colored triangles). The packets which travelled the longer distance are reported with the blue color, covering a distance of more than 27km. All the tests were performed with SF12.

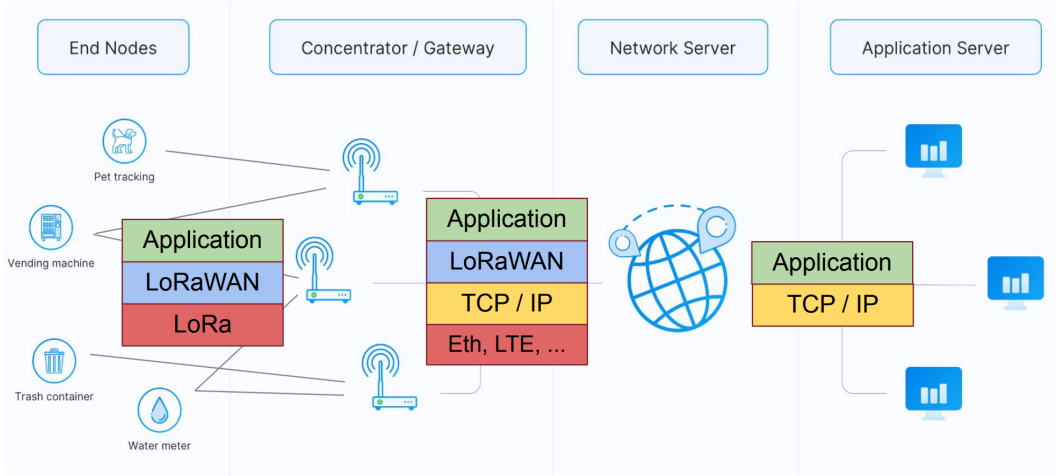
3.3 LoRaWAN

LoRaWAN [61] is an open networking protocol that delivers secure bidirectional communication standardized and maintained by the LoRa Alliance.

The LoRaWAN architecture is deployed in a star-of-stars topology and it is composed by four principals components: End Devices (EDs), Gateways (GWs), a Network Server (NS), and application servers, as schematized in Figure 3.2. Gateways relay messages between end-devices and a central network server that forwards the message to or from the application sever. The gateways are connected to the NS via standard IP connections and act as a transparent bridge, simply converting radio frequency packets to IP packets and vice versa. The wireless communication takes advantage of the long range characteristics of the LoRa physical layer, allowing a single-hop link between the end-device and one or many gateways.

The LoRaWAN network server manages the entire network, dynamically controls the network parameters to adapt the system to every changing conditions. The NS carries on the Over The Air Activation (OTAA), started by the end-devices. This is responsible for the authenticity of the end-devices connected to the network, assigning to each of them a temporary unique identifier, the so called Device Address (DevAddr), and generating a 128-bit AES key in order to establish encrypted communication with the EDs. The NS schedules the acknowledgment and forwards it through the optimal gateway, which is evaluated on the Received Signal Strength Indicator RSSI value of the uplink packet. The NS dynamically manages the transmission parameters of the end-devices with the Adaptive Data Rate (ADR) algorithm. The transmission power and SF of the transmitted packet change in

Figure 3.2. LoRaWAN architecture with all of its principal components and the protocol that they use to communicate.



order to be adapted to the network load and maximize the throughput. Moreover, the NS is also responsible to perform packet de-duplication: a LoRa packet sent by an end-device could be received by multiple gateways and, therefore, the NS will receive multiple copies of the same packet. The NS should consider and/or forward to the application server only one copy of the packet.

LoRaWAN Gateway receives LoRa packets from any devices in the radio range and forwards these data packets to the NS, connected through IP backbone. Each ED can be served by multiple gateways in the radio frequency range. There is no fixed association between an end-device and the gateway. In this way, the packet error rate is reduced, since there is a higher probability that at least one gateway receives the packet, and significantly reduces the battery consumption for end-devices that are mobile. LoRaWAN gateway operates only in the physical layer forwarding the packets to the network server through an IP based Wi-Fi, Ethernet or Cellular backhaul. They only check the data integrity of each incoming LoRa packet. If the CRC status is incorrect the packet will be dropped. If correct the gateway will forward it to the NS, together with some metadata, such as the RSSI of the packet at the receiver. For the downlink packet, the gateway executes transmission requests coming from the network server, without any interpretation of the payload. It is up to the NS to select the appropriate gateway with which to transmit the downlink packet.

End devices are sensors or actuators that wirelessly communicate to a LoRaWAN through a gateway, using LoRa radio frequency modulation. They are autonomous battery operator devices. The LoRaWAN standard defines three classes of end-devices, Classes A-B-C. All such devices must support all operational modes of the lowest class. In other words, all the devices must be able to operate as Class A devices, Class B devices must support both Class A and Class B modes, and Class C devices must support all three modes of operation. The distinction between three Class devices have to do with how the device communicates with the network. In Class A

devices, in order to establish bidirectional communication, each uplink transmission is followed by two short receiving windows during which the end-device listens for possible downlink traffic. It is the end-device to trigger the downlink communication. The two receiving windows start at 1 s and 2 s, respectively, after the end of the uplink transmission. Class B devices open other receiving windows at scheduled times, in order to increase the downlink capability. The Gateway transmits beacons to synchronize downlink communication between network server and Class B end-devices. Class C devices are all the time available to receive downlink communication, except when they transmit. Class A devices consume less power, since most of the time they are asleep. Instead, Class C are the ones with highest power consumption, since they are always active.

Before starting to exchange information in a LoRaWAN, each end-device has to be personalized and activated. The activation of an end-device can be completed in two ways: via OTAA or via Activation By Personalization (ABP).

To join the network through OTAA, an end-device needs to perform a join procedure before starting to exchange information with the network server. A very important point is that an end-device needs to perform a new join procedure each time it looses the session context information. In order to perform the join procedure, the end-device has to be personalized with the following information:

- **DevEUI**. Global end-device ID in IEEE EUI64 address space that uniquely identifies the ED. DevEUI is the recommended unique device identifier by Network Servers, whatever activation procedure is used, to identify a device roaming across networks;
- **JoinEUI**. Global application ID in IEEE EUI64 address space that uniquely identifies the Join Server that is able to assist in the processing of the Join procedure and the session keys derivation;
- **NwkKey** and **AppKey**. Root keys AES-128 specific to the end-device that are assigned to the ED during fabrication.

The OTAA procedure can be resumed in five steps and graphically represented as in the scheme in Figure 3.3.

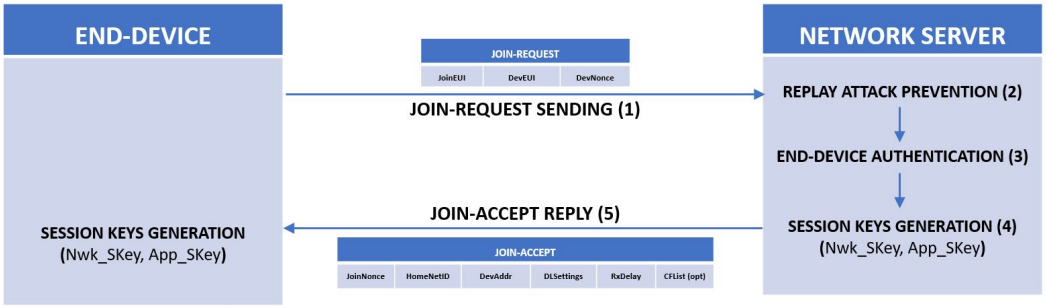


Figure 3.3. Over-The-Air Activation (OTAA) procedure flow graph.

The join procedure is always initiated by the end-device by sending a join request

message (in Figure 3.4) that is a message that contains the JoinEUI and DevEUI of the end-device followed by a nonce of 2 octets (DevNonce).

Figure 3.4. Join request message structure.

When the network server receives a join request, it performs the replay attack prevention process based on the validity of the DevNonce that needs to be such that a pair (JoinEUI, DevNonce) is a unique pair. If the join request is a valid one, the server performs the authentication of the end-device and if the ED passes the authentication, the NS generates the keys Nwk_SKey and an App_SKey.

Once all the security checks are completed, the end-device is permitted to join the network, so the network server will respond with join-accept message, in Figure 3.5.

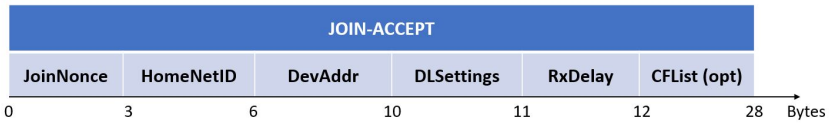


Figure 3.5. Join response message structure.

The JoinNonce is a device specific unique counter value provided by the Join Server and used by the end-device to derive the session keys FNwkSIntKey, SNwkSIntKey, NwkSEncKey, and AppSKey. JoinNonce is incremented with every Join-accept message.
When receiving a join-response the end-device verifies that is a valid message by looking at the JoinNonce and if it is a valid one it stores the DevAddr in itself. The DevAddr is a 32-bit ephemeral device address, that will identify the end-device during a session. When an end-device close or lost its session connection, its DevAddr becomes free and it will be reassigned another end-device, and if an end-device re-join the network a new DevAddr is assigned. Each end-device uses its DevAddr to derive the keys it has to use to send messages through the network.

ABP directly ties an end-device to a specific network by-passing the join request–join accept procedure. In this case, the end-device already contains all what it needs to derive the keys to starting send messages. In other words, the end-device is equipped with the required information for participating in a specific LoRa network as soon as it is started.

3.3.1 Packet structure

In LoRa, there are two types of packets: downlink and uplink packets. Uplink are sent by end-devices to the NS relayed by one or many gateways. Uplink packets use the LoRa radio packet explicit mode in which the LoRa physical header (PHDR)

plus a header CRC (PHDR_CRC) are included. The integrity of the payload is protected by a CRC.

Each downlink message is sent by the Network Server to only one end-device and is relayed by a single gateway. Downlink messages use the radio packet explicit mode in which the LoRa physical header (PHDR) and a header CRC (PHDR_CRC) are included.

Both uplink and downlink messages carry a physical payload that contains all the information needed to forward the message through the network. The physical payload is composed by a single-octet MAC header (MHDR), followed by a MAC payload, and ending with a 4-octet message integrity code (MIC). The structure of a LoRaWAN frame is shown in Figure 3.6.

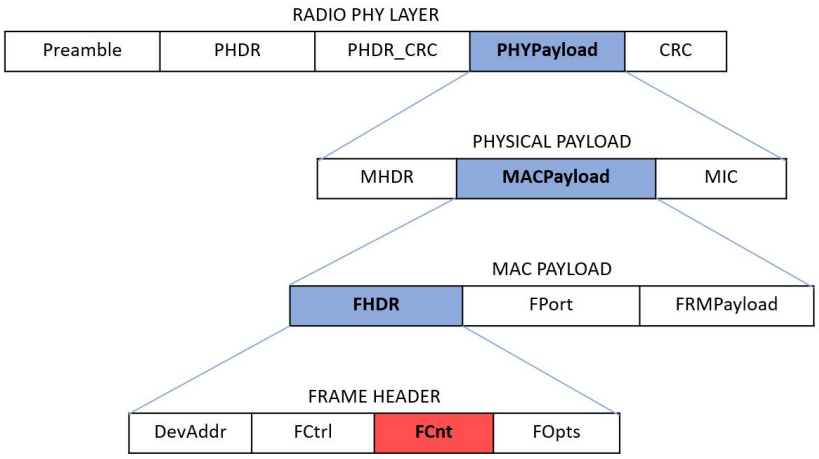


Figure 3.6. Structure of LoRaWAN packet represented in a hierarchical schema. The CRC field in the radio physical layer is present only in uplink packets.

Inside the MAC payload there is the frame header (FHDR) that is the header of the packet at data link layer. The frame header is composed by device address (DevAddr) of 4 bytes, a frame control octet (FCtrl) of 1 byte, a 2-bytes frame counter (FCnt), and up to 15 bytes of frame options (FOpts) used to transport MAC commands. The FCnt is a counter that allows the devices to keep track of the number of data frames sent uplink to the Network Server (FCntUp), and sent downlink from the Network Server to the device (FCntDown). Whenever an OTAA end-device successfully process a join procedure their frame counters is reset to 0 while the ABP devices have their FCnt initialized to 0 at fabrication ant it must never be reset to 0.

3.4 Applications

By using sensor devices with LoRa chipsets having a LoRaWAN connection, we can accommodate a vast range of IoT applications. Thanks to its flexibility, low cost, and robust modulation, the LoRa technology can be used in many different cases, such as smart agriculture, smart cities, Industrial IoT (IIoT), smart environment,

smart homes and buildings, smart utilities and metering, and smart supply chain and logistics [68].

One big challenge is to use LoRaWAN devices to build the next generation cities. In order to build a smart city there are at least four fields where LoRaWAN sensors may be employed: water and energy management, smart homes, indoor/outdoor asset tracking, and smart lighting [71].

To improve the water management in a city LoRaWAN sensors may be deployed in the existing water infrastructure to rapidly detect anomalies (e.g., water leaks) and reduce the waste of water. Since installation, the city of Lyon, France has identified and repaired 1200 water leaks, achieved an 8% increase in water network efficiency and saves an average of 1 million cubic meters of water annually [71].

On the other side, using LoRaWAN in a smart home allows to connect all the devices of the home, indoor and outdoor ones, using one network. We can think to a house with a garden, in this case a Wi-Fi network does not guarantee to connect all the devices of the home. By using LoRaWAN-enabled devices, we have a higher probability of accomplishing this, ensuring a smart home ecosystem with the highest degree of interoperability. To achieve the asset tracking goal the idea is to equip vehicles, products and trucks with LoRaWAN sensors in order to automatically locate, track, and monitor physical assets. The LoRa Edge [67] platform delivers one of the IoT industry's lowest power and most affordable geolocation capabilities than traditional GPS technologies. The Yabby Edge features advanced Cloud-based location calculations, significantly reducing power consumption and extending battery life for up to 12 years. Location data can be easily forwarded to any customer platform or system for simple integration and device settings can be configured to fit any tracking application.

Moreover, LoRaWAN is already used in some cities to monitoring the resource consumption, such as light, gas, and water. OrionM2M is one of the first Kazakstani developers and manufacturers of wireless data transmission systems to integrate Semtech's LoRaWAN devices and the LoRaWAN standard into its smart metering solutions to monitor water, gas, and electricity usage data, as well as smart lighting solutions to deliver operations management and service reliability, with up to 30% reduction in technical losses.

OrionM2M also exploits LoRaWAN to monitor COVID-19 storage and transportation in the last two years [70]. The monitoring system is based on six main points: (i) real-time monitoring of refrigeration chamber temperature parameters, (ii) automatic notification when temperature changes above the set threshold, (iii) logging of events both at the device level with a deep archive and in the system itself, (iv) analytics throughout the network for the required period with the formation and unloading of reports, (v) accounting for operating time and downtime of refrigeration equipment, and (vi) locations map of refrigeration chambers. The temperature and the humidity control is performed 24 h per 7 days and the detected values are stored hourly in a cloud archive. In case of anomalies a notification is sent to the distributor. There are some advantages in using LoRaWAN technologies in this case, the most relevant are the high reliability and availability of server applications backup, short terms of implementation and return of investment, scalability of the system and the ease to add and connect new devices to the network.

3.5 Security

3.5.1 Protocol Security

LoRaWAN uses standard, well-known algorithms and end-to-end security of the communication. Furthermore, as devices are supposed to be deployed for long periods, sometimes for many years, security must think about the future too. The fundamental properties supported by LoRaWAN security are mutual authentication, integrity protection, confidentiality.

Mutual authentication is made quite easy, as the Network Server must know a priori the hardcoded information of the device (root keys and DevEUI for a OTAA device, session context for a ABP device) as this information will be part of the message since the very beginning. Even in the first join message sent by an OTAA device, even if the whole packet is sent in plaintext, the MIC appended at the end of the message is derived cryptographically using one of the root keys, already known by the server, which can use it to ensure the authenticity and integrity of the join message at once. The following join-accept packet is instead totally encrypted with the same shared root key, ensuring the device that this response has been issued by the genuine Network Server. The allocation of EUI-64 identifiers, used for JoinEUI and DevEUI identifiers, requires that these unique identifiers have to be issued by a central authority, in this case the IEEE Registration Authority. LoRaWAN networks instead are identified by a 24-bit globally unique identifier assigned by the LoRa Alliance.

To ensure the integrity of the communication, a cryptographic Message Integrity Code (MIC) is calculated on both the MHDR and the payload of the message. The MIC for a normal uplink/downlink message is calculated using its specific key and AES-CMAC, which is a block cypher-based message authentication code algorithm. The output of AES-CMAC is then truncated to the first 4 bytes and appended at the end of the message. When received by the server, it will first check the message integrity and, if it passes the check, it decrypts the whole packet (at this moment the payload is still encrypted with the AppSKey, unknown to the Network Server) and transfers the message to the application server. For Join Request messages, as the appropriate keys have not been generated yet, the MIC is calculated using the root AppKey instead.

LoRaWAN implements end-to-end encryption for application payloads exchanged between devices and application servers [43]. This choice comes from the idea that in most cases, LoRaWAN network providers are not owned by the same organization that owns the application server in the same way mobile network providers are just a means of transport for packets and do not need to know the real content of the message. To support this, many upper layers like TLS have been developed to ensure confidentiality over an untrusted means. Such an approach is not well suited in the IoT world, where additional security layers add non-negligible power consumption and complexity. The security mechanisms implemented rely on the standardized AES cryptographic algorithms. In particular, LoRaWAN security uses the AES cryptographic primitive combined with several modes of operation: cypher-based message authentication code (CMAC) for integrity protection and counter-mode encryption (CTR) for effectively encrypting the message. In other words, all

LoRaWAN traffic is protected using different session keys, one for the payload and others for the whole content of the package after encrypting the payload, depending on the message type. Each payload is encrypted by AES-CTR and carries and has a message integrity code computed with AES-CMAC to avoid packet tampering. In LoRaWAN 1.0, there were only two different session keys: AppSKey and NwSKey. They were used to encrypt respectively the payload and the whole packet. With LoRaWAN 1.1 instead the mechanism gets more complicated as many different keys are defined each for a specific type of communication. The key derivation scheme is reported in Figures 3.7 and 3.8. The keys NwkSEncKey, SNwkSIntKey, FNwkSIntKey, AppSKey are AES-128 keys, strong enough to theoretically resist common attacks on an encrypted payload, with the additional possibility to be refreshed in a pseudorandom-fashion whenever an ED wants to, with a simple join or re-join request. The possibility to refresh the session keys is another layer of enhanced security.

To be protected against replay attacks, every single message is equipped with an incremental counter, different for different types of messages (a counter for the uplinks, a counter for the downlinks and a counter for the join). The purpose of these counters is to keep track of the messages received to drop possible replays from malicious actors or peculiar cases of re-transmission with the same parameters. Being incremental, the receiver, once checked the MIC and decrypted the message, looks for the counter and expects it to be at least equal to the last counter encountered plus one, but it accepts every message with a counter strictly greater than the last one received. The incremental counter is a new entry in LoRaWAN as, before LoRaWAN 1.1, these counters were random values that the standard expects to be remembered by both the ED and the Network Server, but this logic was doomed to fail as in practice none can really keep track of billions of different nonces, eventually enabling the replay attack once that specific nonce has been forgotten by the receiving party.

Unfortunately, LoRaWAN security features have always been a problem since the very first release of the standard. During the years many security issues have arisen. The flaws in LoRaWAN security mostly regard the communication between end devices and the network server [97], [5], as the ED can be considered the weakest link of the chain, causing, in the best case, a disconnection from the network indefinitely. The innate lack of security of the ABP devices, due to the static nature of its session keys, can be exploited in various ways too [10]. All these problems have resulted in the release of a breaking change update of the standard, LoRaWAN 1.1, that introduced substantial improvements to the security, filling the holes left in the 1.0.X version. Nevertheless, LoRaWAN 1.1 is still far from perfect, as not every problem has been addressed [28] and new flaws have come out in recent years [17]. Moreover, it foresees the possibility for the Network Server to downgrade and use a 1.0.X version, bringing back vulnerabilities from the previous version [28].

3.5.2 Privacy Leakage

As opposed to security, privacy aspects in LPWANs have received little attention. As shown in Figure 3.9, since the long communication range of LPWAN technologies allows messages to be received several kilometers away, an eavesdropper can easily record sensitive information, even if located many hundred meters from endpoints.

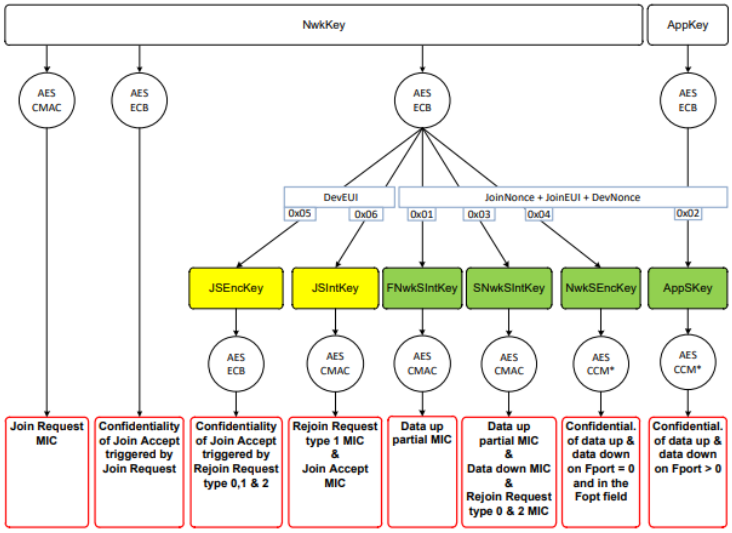


Figure 3.7. The key-derivation scheme used by a LoRaWAN 1.1 ED and Network Server to calculate the different session keys after a successful join procedure, complete with the goal of each key.

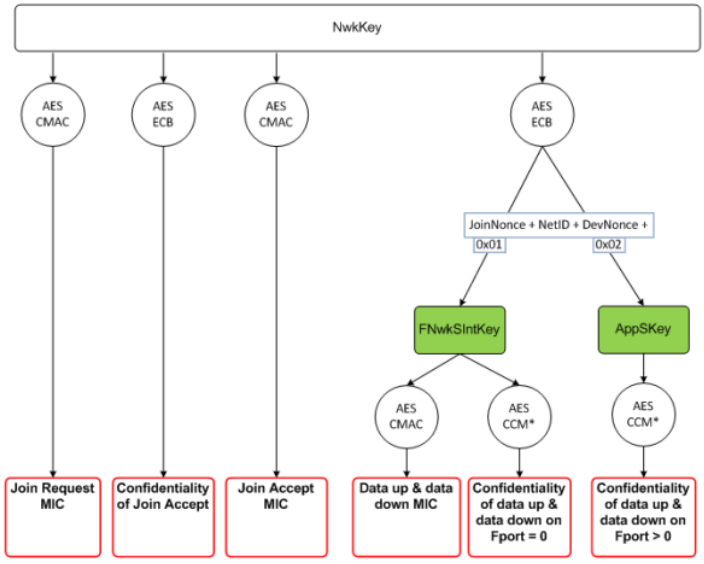


Figure 3.8. The key-derivation scheme used by a LoRaWAN 1.0 ED and Network Server to calculate the different session keys after a successful join procedure.

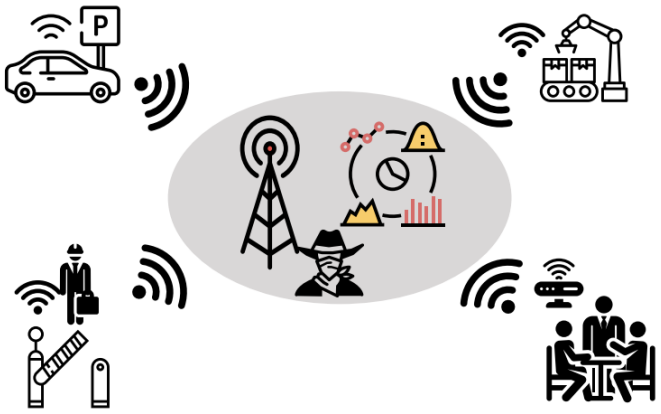


Figure 3.9. When end devices transmit messages to the server, the passive adversary can easily collect sensitive information.

By leveraging features of the raw wireless signal, attackers can interfere with the activities of devices operating in the network. In detail, we assume that the passive adversary, equipped with one or more receivers collecting end device transmissions, is within the communication range of at least one gateway and that it has a *priori* knowledge about the application associated with an ED. With these assumptions, the intruder presents two relevant privacy-related threats.

1. **Fingerprinting**. Device Fingerprinting denotes the set of techniques used to identify a device using the information extrapolated from the packets it uses to communicate over the network [40]. Fingerprinting is commonly used in general-purpose devices to track user behavior and application usage. Interesting implementations include browser fingerprinting for web analytics, fraud detection, and accountability [46]. Despite the benefits for the user experience, Fingerprinting poses also security and privacy risks. For example, LoRa devices can be uniquely recognized using physical layer fingerprinting [75]. It leverages on differences in the analog RF signals sent by wireless devices, caused by imperfections introduced in the analog hardware components during the manufacturing process [14], demonstrating that an adversary can identify a transmitter independently of the modulation scheme or cryptographic mechanisms used.

2. **Information leakage**. Since LPWAN is technology-constrained, attackers can easily reconstruct a view of the network they monitor. While other wireless devices transmit messages not necessarily related to real situations, LPWAN sensors are simpler and often dedicated to no more than one function, such as transmitting data when a given event occurs [52]. Moreover, the LPWAN event space is small and binary, making the purpose of the individual sensors even clearer. Just consider the case of a parking system application that notifies clients if a given parking lot is occupied or not [9]. Endpoints send only two binary messages, the lot is empty or the lot is not empty, and only when there is a change of state (a vehicle left the lot). In absence of obfuscation

techniques, a potential attacker recognizes the presence of real messages by simply eavesdropping on the LPWAN channel. In other scenarios, a deviation from standard transmission behavior presupposes the occurrence of an event. For example, in the case of an application that monitors the availability of a meeting room, the LoRa sensor transmits less when there is a series of long meetings that keep the room busy. The eavesdropper, collecting the messages in a given time frame, can use traffic analysis techniques to extrapolate the transmission behavior and gain information about the current situation.

3.6 Open Issues in LoRaWAN Security

After analyzing the security of the security of the LoRaWAN communication technology from an holistic perspective, we concluded that security issues still persist:

- **Protocol Security.** We thoroughly analyzed the security of LoRaWAN from an information security point of view, and we discovered a flaw in the protocol which enables an attacker to perform a Denial-Of-Service (DoS) attack towards victim devices. Our findings are reported in Chapter 5.
- **Privacy leakages.** After analyzing the behavior of hundreds of LoRaWAN devices in the wild, as reported in Chapter 4, we discovered that the devices exhibit a periodic and predictable behavior when sending uplink packets. This presents a privacy threat, and can be exploited by an attacker to perform the correlation between the two LoRaWAN identifiers for a device, the DevAddr and the DevEUI, as reported in Chapter 6.

Chapter 4

Device profiling

In this Chapter, we conduct an in-depth examination of traffic behavior of LoRaWAN devices. Our analysis is based on a dataset obtained from existing literature, which contains LoRaWAN packets from devices serving several real applications. The goal of this investigation is to introduce key performance metrics tailored for evaluating LoRaWAN as a network operator. In the current literature there are limited existing approaches that offer a mean to process actual LoRaWAN datasets for the extraction of performance indicators. Much of the prior research relies on simulations, whereas we have created software tools to enable the application of these metrics to any existing dataset.

The analysis of LoRaWAN traffic behavior, in the perspective of the key performance metrics we have defined, will open us to different security concerns and vulnerabilities that will be presented in the next Chapters of this thesis.

4.1 LoRaWAN Traffic Characterization in the wild

The analysis of LoRaWAN behavior has been explored in various research papers, each adopting different methodologies. Some studies make use of powerful simulation tools to examine network performance across a range of parameter configurations, as demonstrated in the work by Magrin et al. [56]. Others opt for a data-driven approach, where real-world datasets are analyzed, often incorporating classification techniques to characterize network behavior, as seen in the research conducted by Garlisi et al. [36].

It's important to note that our approach involves passive packet capture at the Gateway (GW) level. Passive packet capture means that we do not actively interfere with or manipulate the data being transmitted. Instead, we are using monitoring systems at the GW, allowing us to observe and record the packets as they naturally traverse the network. This approach is non-intrusive and reflects real-world network behavior accurately.

Capturing packets at the GW level is essential for gaining insights into network performance and behavior. GWs serve as critical hubs in LoRaWAN networks, receiving and relaying messages between end-devices and the Network Server (NS). By capturing packets at this point, we can analyze the entire communication process, from packet initiation by end-devices to their reception at the GW and subsequent

forwarding to the NS.

Passive packet capture at the GW level provides a comprehensive view of network traffic, enabling us to evaluate factors such as packet loss, signal quality, and GW utilization. It allows us to assess network performance and identify any anomalies or issues without introducing artificial disruptions. This approach is particularly valuable when working with real-world LoRaWAN deployments, as it ensures that our analysis accurately represents the network's operational conditions and challenges. By passively capturing packets at the GW level, we maintain the integrity of the data and gain valuable insights into LoRaWAN network behavior.

4.2 LoED Dataset

In this research we leverage the LoRaWAN at the Edge Dataset (LoED) [11], a publicly available dataset that serves as our case study. The LoED dataset offers a unique advantage due to its extensive size, containing 1,263,001 packets, and its comprehensiveness, including traffic generated by a diverse range of IoT applications.

One notable aspect of the LoED dataset is that it was collected passively at the GW level, meaning that the GWs were configured to listen for LoRa frames in the air without any active interference or influence on the data. This passive collection strategy ensures that all packets transmitted within the network were captured, regardless of their source or application origin.

Furthermore, the dataset originates from a deployment in London involving nine GWs strategically positioned across different urban areas. Among these, five GWs were situated outdoors, typically on the rooftops of buildings, providing clear line-of-sight communication with devices. In contrast, the remaining four GWs were placed indoors, with varying degrees of line-of-sight limitations. Notably, one indoor GW was located on the ground floor and had no direct line of sight to any devices.

This approach to data collection in a real-world setting, where packets are observed passively at the GW level, offers valuable insights into authentic LoRaWAN network behavior, reflecting the actual conditions and challenges faced in operational environments. The collected packets use the explicit header mode, which allows the extraction of metadata information, such as:

- time: Time at which the packet was received by the GW;
- physical payload: Raw payload contained in the received packet;
- gateway: Identifier of the GW that has received the packet;
- crc status: Physical layer CRC;
- frequency: Transmission frequency;
- Spreading Factor: Transmission SF of the packet;
- bandwidth: Bandwidth used by the received packet;
- code-rate: LoRa coding-rate of the packet;
- RSSI: Sampled RSSI value of packet reception;

Table 4.1. GW numbers to GW IDs.

GW Number	GW ID
00000f0c210281c4	GW1
00000f0c22433141	GW2
00000f0c210721f2	GW3
00000f0c224331c4	GW4
00800000a0001914	GW5
00800000a0001793	GW6
00800000a0001794	GW7
7276ff002e062804	GW8
0000024b0b031c97	GW9

- SNR: Sampled SNR value of packet reception;
- device-address: Device-address of the device that has sent the packet;
- mtype: mtype bit fields of the packet;
- fcnt: Frame counter value of the packet;
- fport: Port of the packet.

The GW numbers consist of a hexadecimal string with 16 characters. For simplicity, we've assigned each GW an ID following the format GWX, where X is a number ranging from 1 to 9. The mapping between these IDs and the GW numbers is provided in Table 4.1. During the acquisition period, a total of 11,263,001 packets were collected from devices, involving 145,023 distinct device addresses.

4.3 LoRaWAN Behavior Analysis

In this section, we delve into various facets of the LoRaWAN protocol. To accomplish this, we have organized our investigation into six distinct segments, each concentrating on a particular aspect of the network:

1. Gateway radio indicators;
2. Packets loss;
3. Device–Gateway interaction;
4. End-Device addressing and activation;
5. Duty-Cycle enforcement;
6. Performance of de-duplication procedure;
7. Periodic behavior analysis of packet transmission.

Table 4.2. Table reporting all mtype fields foreseen by LoRaWAN with their related message type.

mtype	Message Type
000	Join request
001	Join accept
110	Rejoin request
010	Unconfirmed data-up
100	Confirmed data-up
011	Unconfirmed data-down
101	Confirmed data-down
111	Proprietary

Now, let's begin by examining the dataset in terms of packet quantity and packet types. LoRaWAN encompasses various "types" of packets, such as packets necessitating an acknowledgment, join request messages, and more. The specific type of a packet is encoded using three bits in the mtype field of the MAC Header. This field is transmitted in plain text with the packet and is included in the dataset. Table 4.2 provides a comprehensive list of all possible mtype values along with their descriptions.

To initially characterize the traffic in the dataset, we tallied the total number of packets for each mtype.

As outlined in Chapter 3, the LoRaWAN protocol involves receiving duplicate copies of the same packet. The dataset includes these duplicate copies, so we performed two separate counts: one that includes duplicates and another that removes them.

Table 4.3 presents the total number of packets in the dataset and the count after eliminating duplicates. There are 11,263,001 collected packets in the dataset, and after removing duplicates, there are 8,266,868 distinct packets.

The "Unknown" packets refer to those captured by the LoRa antennas on the GWs but do not conform to the LoRaWAN protocol. Many of them have empty PHYPayloads and may be beacon packets for proprietary systems using LoRa. We treat all these packets as distinct since we cannot confirm if duplicate receptions occur.

In the subsequent sections, we will introduce the features we intend to analyze. Then, we will outline our approach for conducting these specific analyses and present the results on the LoED dataset.

4.3.1 Gateway Radio Indicators

The primary objective of our initial analysis is to scrutinize the behavior of two key radio parameters: RSSI and Spreading Factor (SF). Through the examination of these parameters, we aim to gain insights into the network performance as observed at the GW level.

RSSI, measured in dBm, denotes the received signal power. This metric serves as an indicator of signal quality received by the GW. In our dataset (LoED), RSSI

Table 4.3. Number of packets contained in LoED dataset grouped by message type. In the centre column it is reported the total number of packets including the duplicates, while on the right it is reported the number without duplicates.

Message Type	Number of Packets	Number of Distinct Packets
Join request	246,272	170,476
Rejoin request	27,576	26,561
Join-Accept	20,971	20,660
Conf. data-up	664,474	441,742
Unconf. data-up	7,441,505	4,776,723
Conf. data-down	63,216	32,537
Unconf. data-down	24,140	23,989
Proprietary	30,206	29,539
Unknown	2,744,641	2,744,641
Total	**11,263,001**	**8,266,868**

represents the signal strength received at the GW for uplink messages.

SF, on the other hand, regulates the chirp rate of an end-device in a LoRaWAN, directly influencing data transmission speed. SF can be dynamically managed by the NS using the Adaptive Data Rate (ADR) protocol in conjunction with the End Device (ED), or it can be set manually by the ED. Lower SF values correspond to faster chirps and, consequently, higher data transmission rates. As SF increases, the chirp sweep rate decreases, and data transmission rate is reduced accordingly. However, higher SF values enhance the likelihood of successful packet reception by a GW.

Table 4.4 provides a breakdown of SF values and their resulting bit rates.

Table 4.4. Spreading factor values with relative bit rate. [66]. In this table we use the 125kHz bandwidth channel.

Spreading Factor	Bit Rate [bps]
SF12	150
SF11	260
SF10	580
SF9	1360
SF8	2740
SF7	4840

To analyze the behavior of RSSI and SF values at the GW level in LoED, we segment the packets based on the receiving GW. For each subset, we record the RSSI and SF values for individual packets.

Figures 4.1 and 4.2 present histograms depicting the distribution of RSSI and SF values per GW, respectively. These values are measured at the GW level.

From Figure 4.2, it's apparent that a significant portion of LoRa packets exhibit low SF values, typically around 7 or 8. This choice of SF is often made to achieve maximum data rates, minimize network resource consumption, and optimize energy

efficiency.

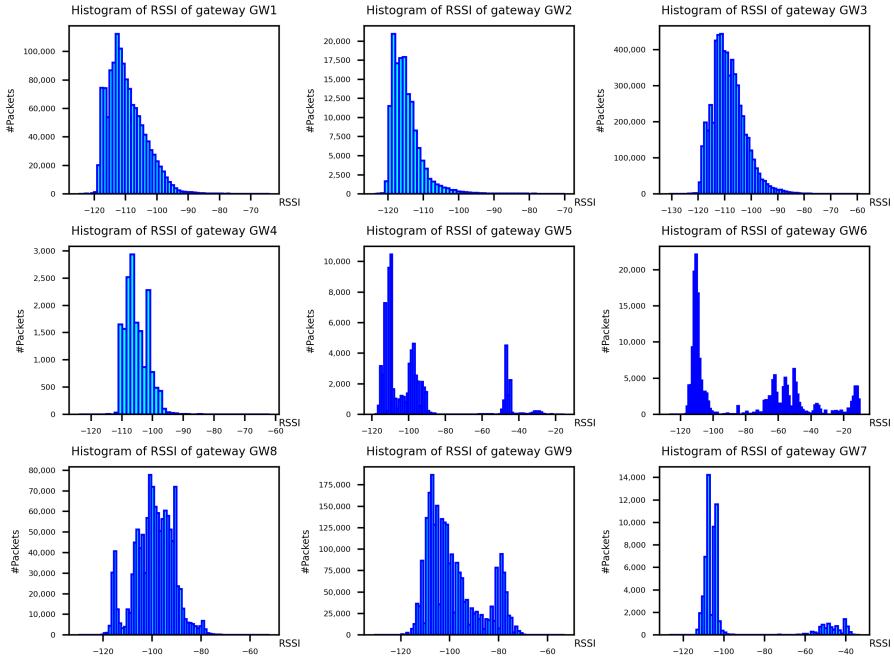


Figure 4.1. Histogram of RSSI per gateway.

The observations made regarding GWs GW8 and GW9, where a substantial portion of packets employ SF12, are consistent with their geographical placement. These GWs are situated atop tall buildings with unobstructed surroundings, enabling them to receive packets transmitted by remote EDs that use higher SF values to ensure their packets are received.

In relation to the RSSI behavior depicted in Figure 4.1, it is evident that each GW exhibits distinct characteristics. Specifically, gateways GW1, GW2, and GW3, corresponding to the first three plots in Figure 4.1, display well-defined RSSI distributions with high packet density. These GWs maintain an average RSSI value of approximately -110 dBm and exhibit an exponentially decreasing tail of higher RSSI values. This behavior can be attributed to their deployment on elevated locations with minimal obstructions, resulting in the reception of a significant number of LoRa packets and a clearly defined RSSI distribution.

Conversely, gateways GW8 and GW9, while sharing some similarities, display slightly less well-defined RSSI distributions. These GWs also benefit from elevated placements and are likely to receive a substantial volume of LoRa packets.

gateways GW4, GW5, and GW6, situated indoors within a university building, differ notably in their RSSI distributions. These GWs receive fewer packets, resulting in distinct peaks in their RSSI distribution. One peak is centered around -110 dBm and likely originates from packets generated by devices located outside or at a distance from the GWs. The second peak, around -50 dBm, is likely produced by devices within the same building and in close proximity to the GWs. This dual-peak behavior can be attributed to the indoor deployment environment, which affects signal propagation and results in varying RSSI values.

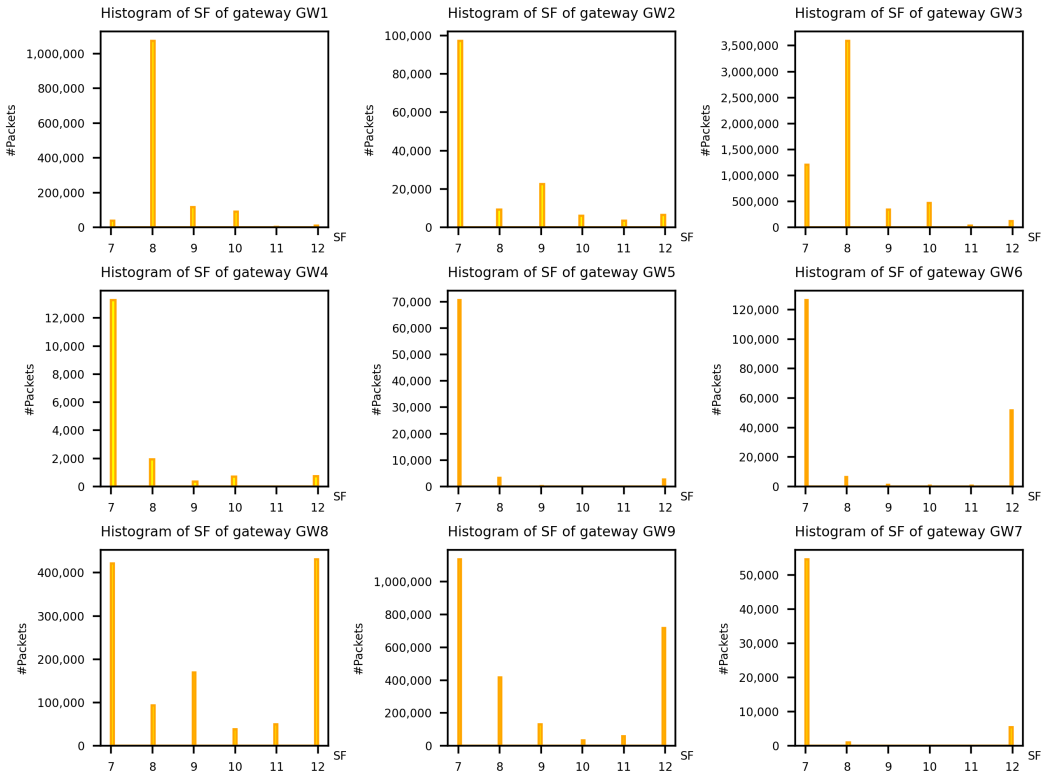


Figure 4.2. Histogram of spreading factor per gateway.

4.3.2 Packet Loss

In this section, our attention turns to the issue of packet loss within a LoRaWAN network. Packet loss in such networks can stem from a range of factors. Notably, previous research endeavors have aimed to establish models for packet collisions occurring at the LoRa physical layer, as detailed in references [91] and [31].

The predominant portion of LoRaWAN traffic is categorized as "Unconfirmed," implying that it doesn't necessitate an acknowledgment, as outlined in Table 4.3. Unconfirmed traffic is advantageous as it places fewer demands on LoRaWAN resources, making it particularly suitable for devices with limited energy supplies. However, it's important to note that in the case of unconfirmed uplinks, the NS remains unaware of any lost packets because acknowledgments and/or retransmissions are not mandated.

Our objective is to determine the distribution of lost packets during transmission. To do this, we segment the dataset by device address DevAddr. For each individual device, we take into account the FCnt (Frame Counter) located in the frame header. The FCnt is a 16-bit counter that increments by one for each uplink packet transmitted by an ED. Importantly, FCnt is reported in plain text in the packet header, which means we can analyze the proportion of lost packets even without access to decryption keys.

If any packet is lost, the observed FCnt values will be continuous. However, in the presence of lost packets, there will be gaps in the FCnt values. Specifically, if

we observe a packet with FCnt $= n$, followed by another packet from the same ED with FCnt $= m$, the number of lost packets in between is equal to $m - n - 1$. This approach is illustrated in the following Figure 4.3.

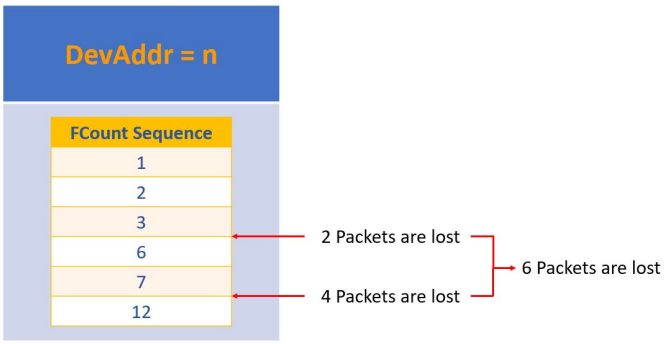


Figure 4.3. Scheme of the adopted strategy to compute the number of lost packets for a given DevAddr.

To compute the distribution of lost packets, we first isolate the packets for each device in the network. For each device, we calculate the ratio of lost packets using the previously described method. Then, by computing the probability density function (PDF) of these values, we obtain the result shown in Figure 4.4. In Figure 4.5, the same distribution is presented as a Cumulative Distribution Function (CDF). To enhance the reliability of the analysis, we only considered packets from devices that transmit regularly to the network.

Figure 4.4 illustrates that most devices lose only a small fraction of their packets, while others experience packet loss rates of up to 50%.

To conduct this analysis, we took into account a total of $N_C = 3,374,952$ uplink packets that were correctly received, and we calculated that $N_L = 1,333,398$ packets were lost. The cumulative fraction of lost packets in the entire LoRaWAN network can be expressed as follows:

$$\text{lost} = \frac{N_L}{N_C + N_L} = 0.283$$

4.3.3 Device-Gateway Interaction

The objective of this analysis is to examine the number of devices managed by each GW. Given that we do not have access to the unique device identifier (DevEUI), we count the number of temporary devices, which can be distinguished by the DevAddr field encoded as clear-text in the packet header.

Our approach involved parsing the dataset and grouping the packets by GW. We then tallied the number of distinct device addresses observed for each GW. The results of this analysis are visualized in Figure 4.6.

The histogram presented in Figure 4.6 illustrates the number of distinct device addresses observed by each GW. Notably, the two GWs that have encountered the highest number of distinct DevAddrs, GW6 and GW5, were in operation for extended durations of 552 and 573 days, respectively. However, these GWs are

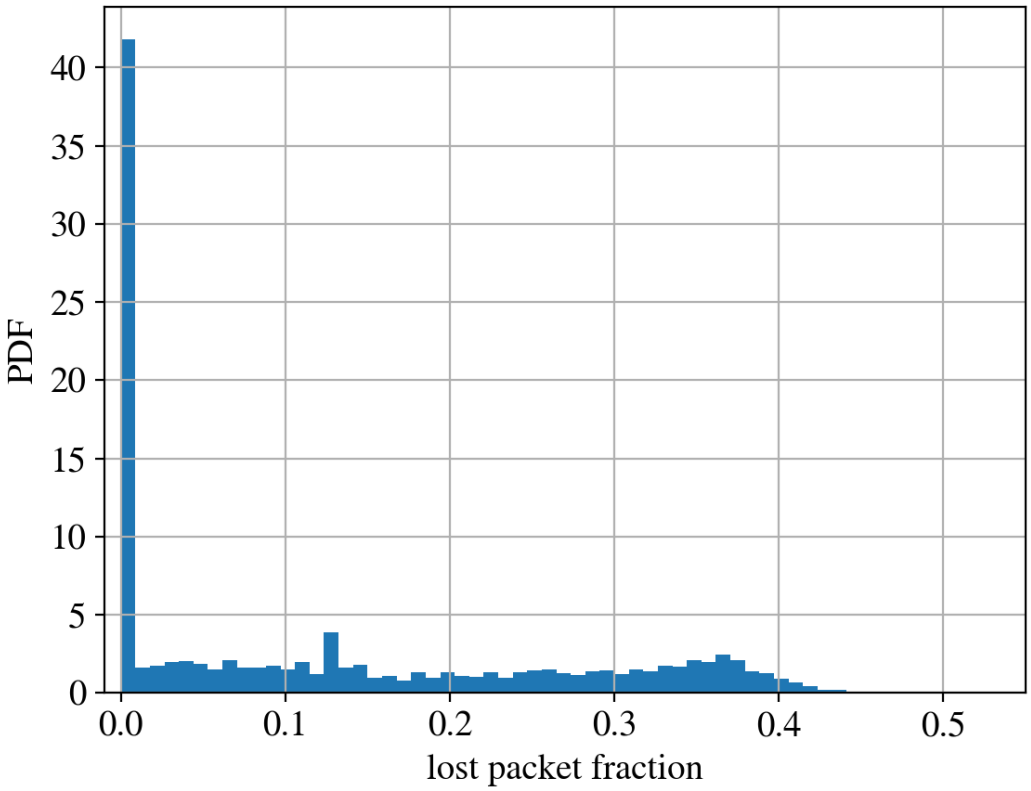


Figure 4.4. PDF distribution of lost packets. To compute it, we isolated the packets of each device in the network. Then, we computed the fraction of lost packets for each device. The PDF in figure is the distribution of those ratios.

situated in an indoor environment, which is suboptimal for receiving LoRa packets. Consequently, they managed only 186,592 and 76,706 packets, respectively.

In contrast, GW3 interacted with packets from only 2,705 distinct DevAddrs, yet it handled the highest packet count of 5,757,575 within a relatively short operational window of 56 days.

4.3.4 End-Device Addressing and Activation

In this section, we investigate the behavior of the Over The Air Activation (OTAA) by looking at the device addresses and the real identifiers of the devices.

The OTAA activation method foresees an exchange of join request/join accept messages between an ED and the NS. During this procedure, several network parameters could be set by the NS, including the DevAddr, a public ephemeral identifier for the ED, which will be sent as plaintext in every Confirmed/Unconfirmed Data Up packet.

During its lifetime, an OTAA device could assume several distinct values of a DevAddr. However, it has one and only DevEUI, which is a unique identifier for a LoRaWAN device. The DevEUI is assigned by the manufacturer in a similar way to a MAC address, containing information about the producer and the unique identifier for the ED.

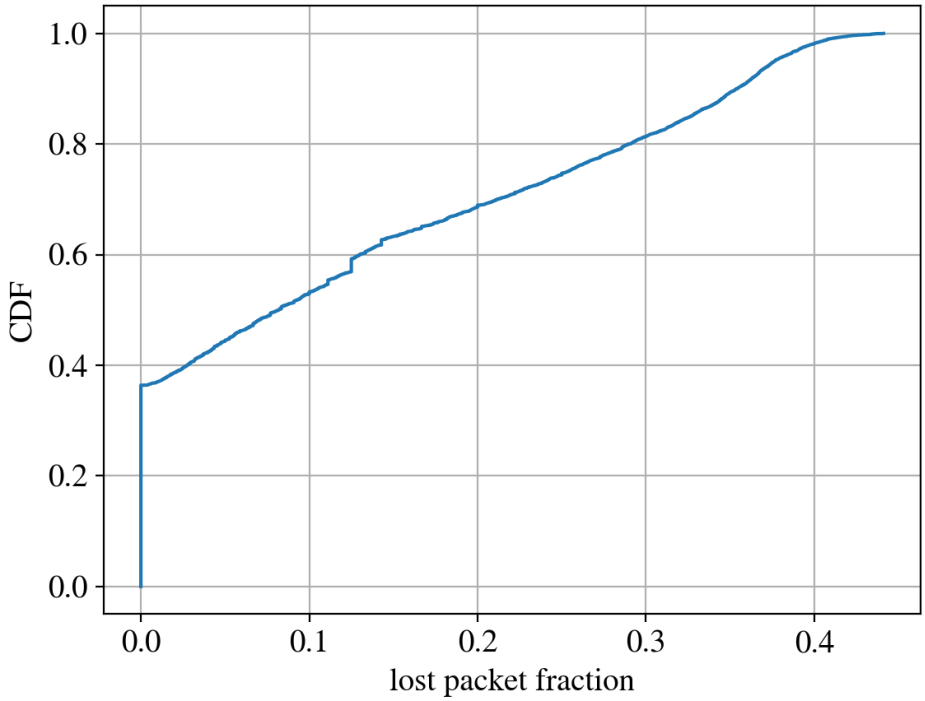


Figure 4.5. Plot of CDF of devices in function of percentage of packet loss.

The DevEUI is not used in Confirmed/Unconfirmed traffic for several reasons, including privacy, as noted in [84], [83] and previously in this chapter, and, in general, it is not reported in public datasets. However, the DevEUI is sent as plaintext in a join request message. This opens up a possibility for us to fetch the DevEUI of the devices that performed an OTAA procedure in the LoED dataset.

As stated earlier, the entries of the LoED dataset do not have the DevEUI of the end-device that has sent the packet, but we can retrieve the DevEUI from the payload of the join request messages. In order to retrieve the DevEUIs, we filter all the join request messages that we identify by looking at the mtype field, which in the case of join request is equal to 000. Once we have identified a join request, we consider its payload, and we extract a substring of 8 bytes, from the 8th to the 15th, as shown in Figure 4.7.

By following this strategy, we counted the number N_{DevEUI} of distinct DevEUI in the dataset $N_{DevEUI} = 1481$. N_{DevEUI} represents the number of distinct devices which are present in the dataset and have gone through an OTAA procedure at least once. The number of distinct DevAddr is instead $N_{DevAddr} = 49,876$. We can compute the ratio between $N_{DevAddr}$ and N_{Dev}:

$$\frac{N_{DevAddr}}{N_{Dev}} \simeq 33.68 \tag{4.1}$$

This ratio represents, on average, the number of DevAddr assumed by a single ED.

We also have analyzed the number of join request, rejoin request, and join accept

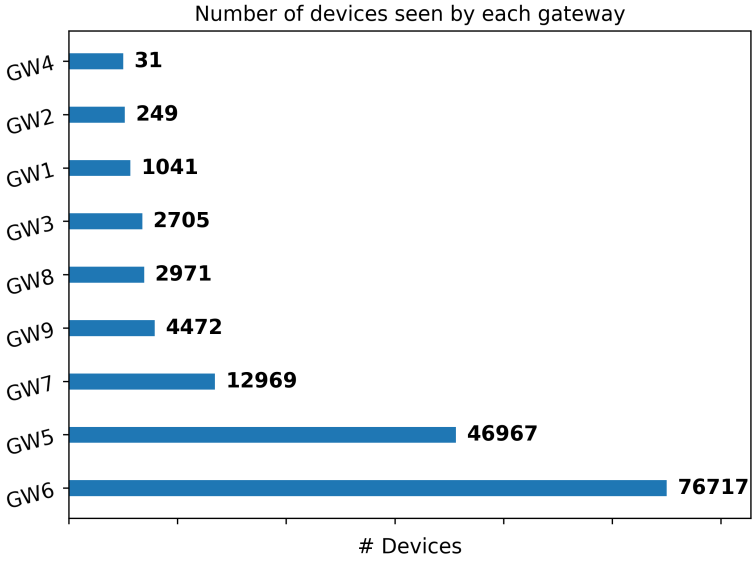


Figure 4.6. Histogram representing the number of devices managed by each GW in the network.

Size (bytes)	8	8	2
Join-request	JoinEUI	DevEUI	DevNonce

Figure 4.7. Structure of the PHYPayload of a join request message.

messages. We could easily filter them by the mtype field, since these message types have mtype, respectively, set to 000, 001, and 110. Without considering the duplicate messages we have the following:

- Number of join request: $N_{JoinReq} = 170,476$;
- Rejoin request: $N_{RejoinReq} = 26,561$;
- Join accept: $N_{JoinAcc} = 20,660$.

We calculated the average number of completed join procedures per device by considering one completed join procedure per join accept. Additionally, according to the LoRaWAN standard, a rejoin request may not always result in a join accept; it could be a standard downlink message, possibly containing MAC commands to configure network parameters. To account for this, we included one completed join procedure per rejoin request. To compute the average number of completed join procedures per device, we applied the following formula:

$$\frac{N_{JoinAcc} + N_{RejoinReq}}{N_{Dev}} \simeq 32 \tag{4.2}$$

On average, we found that each device completed approximately 32 join procedures during the dataset's packet collection period. Additionally, we examined the

ratio of join accept messages to the total number of observed DevAddr:

$$\frac{N_{JoinAcc}}{N_{DevAddr}} = 0.41 \tag{4.3}$$

The ratio of 0.41 indicates that within the network, there are more DevAddr values than the number of completed join procedures. This surplus of DevAddr values could be attributed to the presence of ABP (Activation By Personalization) devices, which don't necessitate a join procedure. It may also include devices that had already performed a join procedure prior to the commencement of the packet collection period.

4.3.5 Duty-Cycle Enforcement

The LoRaWAN standard enforces regulations on the duty cycle that all devices must adhere to. In the context of LoRaWAN, the duty cycle is defined as the ratio of time during which a packet is actively transmitted over the airwaves to the time of radio silence. An example of duty cycle behavior is illustrated in Figure 4.8.

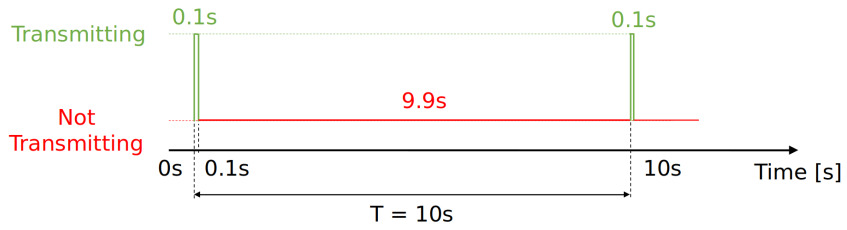


Figure 4.8. Schema representing behaviour of device respecting a duty-cycle of 1%.

The specific limitations on duty cycle depend on various factors, including the region of operation, the channel being used, and the type of packets being sent. In the EU868 band, for instance, the maximum allowable duty cycle is 1%, meaning a device can transmit packets over the air for 1% of the time and must remain silent for the remaining 99%.

The duration of radio operation depends on the Time on Air (ToA) of a packet, which, in turn, is influenced by LoRa radio parameters. For example, if a device sends a join request with a payload of 23 bytes, uses SF10, operates on a 125 kHz channel, and has a final ToA of approximately 500 ms, it must wait for 99 times 500 ms, which is equal to 49,500 ms or 49.5 seconds, before sending another message.

To investigate whether the duty cycle constraints are adhered to by the end-devices in the real LoRaWAN network represented by the LoED dataset, we first examined the duty cycle regulations applicable in the United Kingdom, where the LoRaWAN used to collect the LoED dataset is based. In the UK, the duty cycle constraint must be less than 1%. The strategy employed to compute the duty cycle for each device observed in the network can be summarized in three main steps:

1. Find the period of time of connection of the device to the network, computed as $T_{Dev} = T^f_{Dev} - T^i_{Dev}$ where T^i_{Dev} is the first time in which the device is seen in the network and T^f_{Dev} is the last time in which the device is seen in the network;

2. Compute the amount of time T_{Dev}^{act} in which the device was active, we find this amount time by sum all the ToA of the packets sent by the device;

3. Compute the duty-cycle ratio as:

$$\text{DutyCycle} = \frac{T_{Dev}^{act}}{T_{Dev}}. \tag{4.4}$$

   This percentage represents the ratio of the time the device was actively transmitting over the total time it was connected to the network. This analysis allows us to assess whether devices in the LoRaWAN network are complying with the duty cycle regulations, which is crucial for efficient and fair spectrum usage.

The three steps described are carried out individually for each device address observed in the network. To calculate the ToA of a packet, the following formula is applied:

$$\text{ToA} = \left(m_{ph} + \frac{8 \times L_p}{4 \times \text{SF}} \times (4 + RDD)\right) \times \frac{2^{SF}}{BW} \tag{4.5}$$

where, we have:

- Preamble $m_{ph} = 1$;
- $RDD = 1$;
- SF is the Spreading Factor of the packet;
- Length of the payload L_p of the packet;
- Bandwidth used to transmit the packet expressed in Hz.

This strategy was applied to calculate the duty cycle compliance of devices in the LoRaWAN network. The results are depicted in the plots in Figure 4.9. In the bottom-right part of the figure, you can see the CDF of all the devices plotted against the percentage of duty cycle.

The blue vertical line represents the 1% duty cycle constraint, which is the maximum allowable duty cycle for devices in the United Kingdom according to LoRaWAN specifications. As shown in the plot, this constraint is respected by all the devices in the network.

In the zoomed-in section of the plot, there is a green vertical line representing the common LoRa duty cycle constraint of 0.1%. The CDF value of 0.91 is marked on the plot, indicating that 91% of the devices maintain a duty cycle lower than 0.1%.

These results demonstrate that the vast majority of devices in the LoRaWAN network adhere to the duty cycle regulations, ensuring fair use of the radio spectrum and compliance with regional regulations.

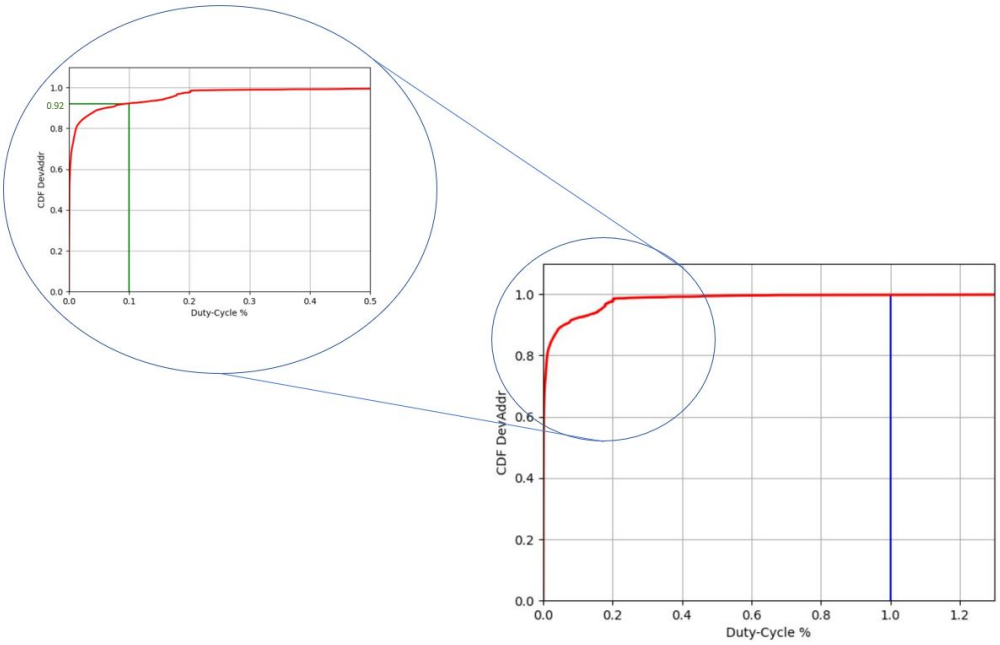


Figure 4.9. Plot of CDF of end-devices in the function of the percentage of the duty-cycle with zoom on duty-cycle percentage range [0, 0.5]. In the complete graph, the blue line represents the duty-cycle constraint of 1% while in the zoom the green line represents the duty-cycle of 0.1% and the correspondent CDF of 0.91.

4.3.6 Performance of De-Duplication Procedure

In a LoRaWAN network, when the NS sends a downlink packet to an end-device, it needs to select the GW that will forward the packet to the end-device. This selection is straightforward when the NS receives a single uplink packet because it can simply choose the GW from which it received the uplink packet. However, in a multi-GW network, a single uplink packet from a device could be received by multiple GWs. All these duplicate packets are then forwarded to the NS. In this situation, the NS must perform packet de-duplication, meaning that only one packet among the duplicates is retained and forwarded to the Application Server. Additionally, the GW that received the de-duplicated packet is the one selected to transmit a downlink packet to the device if necessary.

It's important to note that LoRaWAN specifications do not show the exact procedure that the NS should follow for de-duplication and GW selection. One common criterion for selecting a GW is to choose the one that forwarded the uplink packet with the highest SNR [25] or the highest RSSI.

To collect all the uplink packets, the NS opens a time window known as the de-duplication window, which has a duration denoted as T_D. LoRaWAN specifications do not prescribe fixed values for T_D, but some popular open-source LoRaWAN NS [15], [38], [39] implementations, such as ChirpStack and The Things Network (TTN), use a value of $T_D = 200$ms, as we discovered in Chapter 5. To investigate the behavior of duplicate packets, we can analyze them as if we were a NS performing a de-duplication procedure. This involves carefully selecting the value of T_D, the

de-duplication window duration.

In Figure 4.10, two cases of de-duplication are illustrated. In both cases, there are three copies of packet p, but they differ in the duration of the de-duplication window T_D. In the upper scheme, T_D is set such that all the duplicates are considered for the de-duplication procedure. In this case, all the duplicates are deleted, and only one response is sent. However, the drawback in this scenario is that T_D is too long. After receiving the first copy of p from GW 3, the NS waits for a long time before performing the de-duplication, even though there are no more duplicates to arrive.

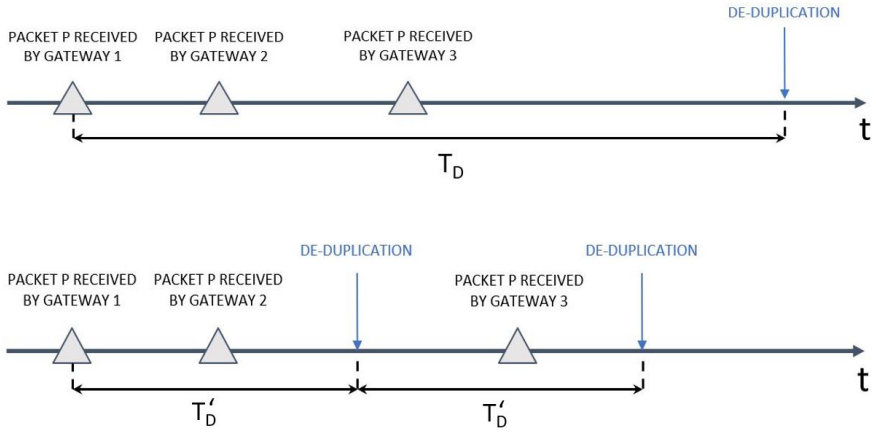


Figure 4.10. Scheme of the de-duplication behaviour in function of the time. In the upper scheme only one de-duplication window T_D is represented, while in the scheme below two de-duplication windows of time T'_D are represented. At the end of each de-duplication window, the NS performs the de-duplication procedure and commits the uplink into its database.

In the second case illustrated in Figure 4.10, there are two de-duplication windows with durations T'_D. In this scenario, two duplicates are considered in the first de-duplication window, and the last duplicate arrives in the second de-duplication window. Since the duplicates are divided into two de-duplication windows, the NS will delete one of the duplicates in the first de-duplication window and consider the third duplicate as a wrong packet because it has an already used frame counter.

The challenge here is to find an optimal trade-off for T_D, the de-duplication window duration, which will be used by the NS, which can be found as follows:

1. Identify the duplicate packets and analyze the time T_F when the first duplicate appears and the time T_L when the last duplicate appears for each group of duplicate frames.

2. Use these values to compute the optimal de-duplication window duration for each duplicate packet p as follows:

$$T_D^p = T_L^p - T_F^p \tag{4.6}$$

After calculating the optimal de-duplication window duration for each group of duplicates, we generated a plot illustrating the distribution of the optimal de-duplication window times relative to the quantity of duplicate groups. As seen in

Figure 4.11, our analysis results suggest that we can deem a de-duplication time as suitable when it corresponds to the first elbow point on the distribution curve. Beyond this point, the curve levels out. In our case, this optimal value is 200 ms.

By setting $T_D = 200$ ms, we achieve a balanced trade-off between eliminating duplicates and maintaining network performance. This choice ensures that we can identify and eliminate approximately 96.8% of duplicate packets.

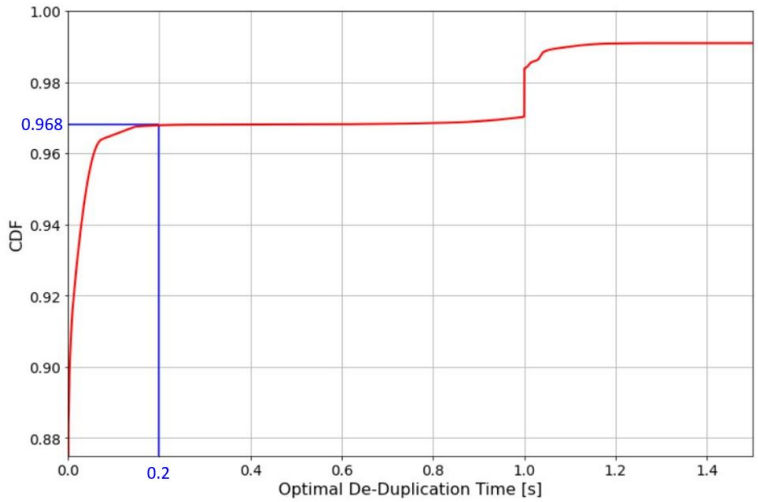


Figure 4.11. Plot of the distribution of the groups of duplicates over the optimal de-duplication time expressed in second.

4.3.7 Periodic Behavior of Devices

We conducted an analysis of the temporal patterns exhibited by the devices in their packet transmission behavior. In other words, we wanted to find if the devices follow some sort of rule or temporal behavior in the time they transmit packets. By finding such behavior, one could, in some way, predict when a device will send the next packet. This could lead to further investigation for network optimization algorithms or open the possibility for security concerns, as further explained.

In order to analyze the temporal behavior of the devices, we have computed the difference of time of arrivals between consecutive packets, transmitted by the same device, from now on referred as inter-arrival time. Note that by consecutive packets we mean the packets that have consecutive value of the counter FCnt, as shown in Section 4.3.2. When analyzing the inter-arrival times, one must take care of the lost packets, otherwise the computation will be wrong. This is true especially in a LoRaWAN environment, where the majority of the packets are not confirmed by acknowledgments, and in the LoED dataset, where the GW that have collected the packets may not be in the vicinity of the devices, observing a high fraction of lost packets.

By analyzing the inter-arrival times, we found out that the majority of the devices follows a simple periodic behavior. In other words, generally a device transmits a packet, does nothing for a fixed period of time and then repeats the loop. In Figure

4.12 we report the inter-arrival times computed on the whole dataset expressed as a probability density function. Figure 4.12a covers a span of inter-arrival times up to roughly one hour, while Figure 4.12b is a zoom on inter-arrival times up to 1000 seconds. What we can observe is that the majority of inter-arrival times follows some fixed values. Moreover, such values are not random, but are round whole numbers which appear to be programmed by human. For example, some of the most frequent values are 30, 60, 300, 600 and 900 seconds, equivalent to some minutes. However, analyzing the inter-arrival times all together gives more weight to smaller inter-arrival times, since the smaller the time, the higher the packet frequency and so the higher the number of packets. In order to remove this bias, for each device we computed its own average inter-arrival time. Results are shown in Figure 4.12c and 4.12d, where the latter is a zoom on inter-arrival times up to 1000 seconds. Here we see that the inter-arrival time of 600 seconds (10 minutes) is the most frequent one, and the 30 and 60 second times are scaled down. Moreover, we can observe a little peak at 3600 seconds (1 hour), which was barely visible in Figure 4.12a.

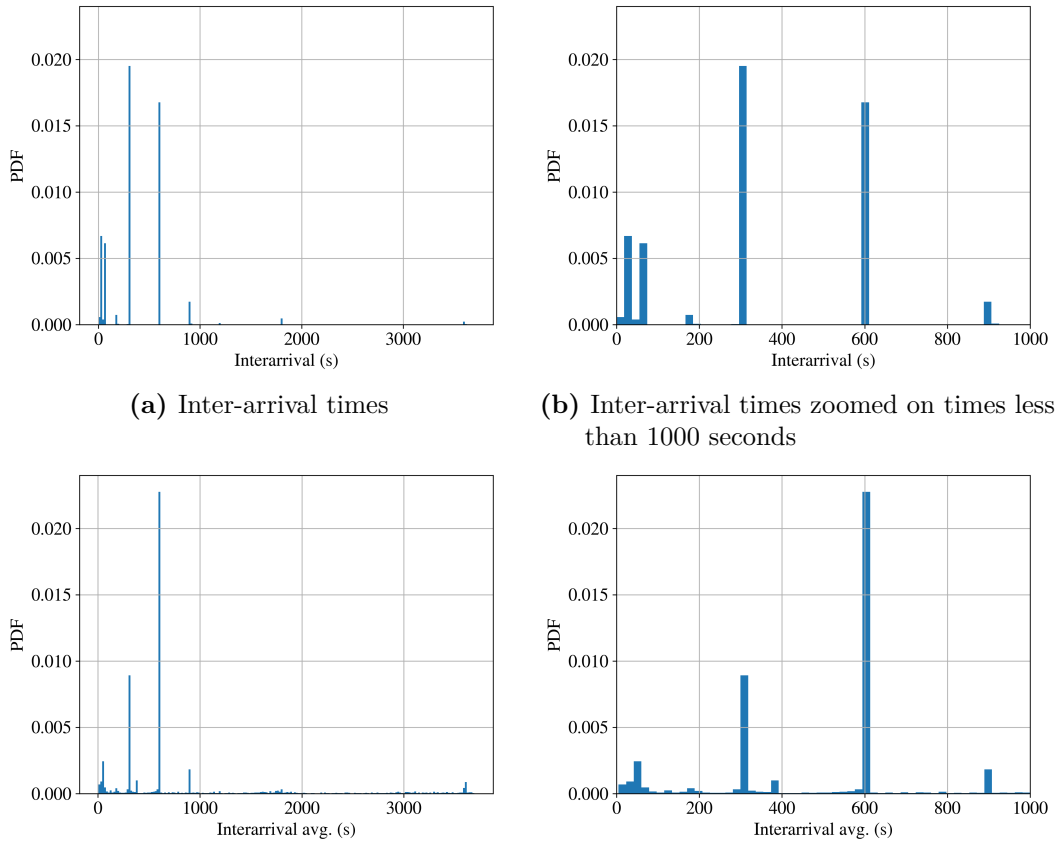
Given the simple periodic rules that we have found, one can predict, with a high degree of accuracy, when a device is bound to transmit a new packet. A nice application of this findings is to predict the exact times in which lost packets were transmitted. As an example, we consider a device in the dataset, having as DevAddr the value `00aa9ca4`. In Figure 4.13 we report the packets transmitted by such device: on the x-axis there is the value of FCnt, while on the y-axis there is the absolute time of transmission, expressed as a UNIX timestamp. We can observe in Figure 4.13a evident holes in the graph, this is because some FCnt are missing, meaning that those packets were indeed lost. In Figure 4.13b the same packets are plotted, but this time we also plotted the predicted time of arrivals of lost packets, in red. We can observe that the prediction indeed matches the observed packets of the device.

4.4 Overview and Findings

In this section, we provide a concise overview of the tests conducted using the proposed methodology. The results obtained from these tests align with LoRaWAN specifications, affirming that the network's observed performance adheres to the expected behavior.

In the initial analysis (Section 4.3.1), we examined the RSSI and SF of all the network packets. These parameters consistently fell within the anticipated ranges. The RSSI values were greater than -120 dBm, and SF values ranged from 7 to 12. As discussed earlier, the variability in RSSI aligns with the ADR mechanism, where devices adjust power to achieve a specific RSSI level before changing the SF. This adaptation ensures a stable transmission quality.

Moving on to Section 4.3.2, we utilized the FCount field in the packet header to quantify the number of lost packets per device. The results indicated that all tested devices had a packet loss rate below 45%, with an average packet loss rate of 28.3%. This measurement is critical for addressing potential issues, particularly in applications where a 28.3% packet loss rate might have significant consequences. This highlights the trade-off between using a low-power network and ensuring reliable data transmission.



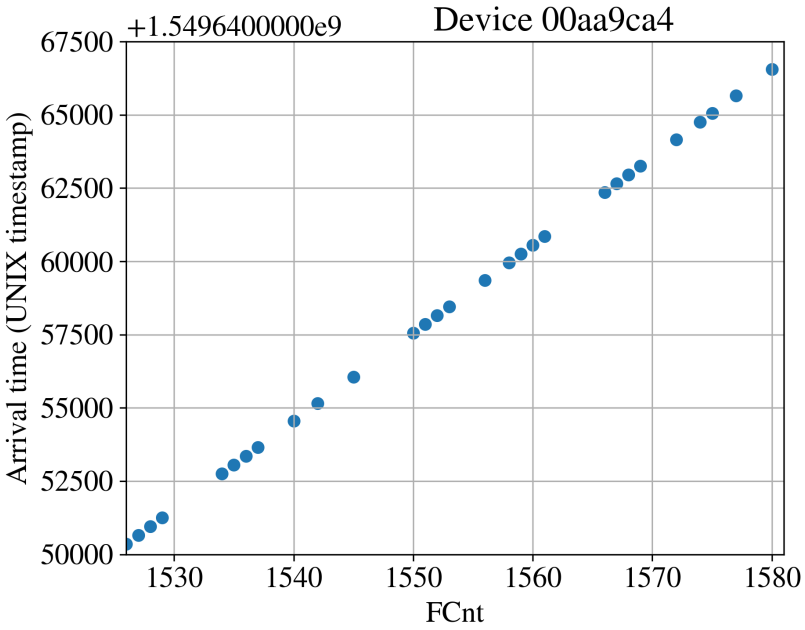
(a) Inter-arrival times

(b) Inter-arrival times zoomed on times less than 1000 seconds

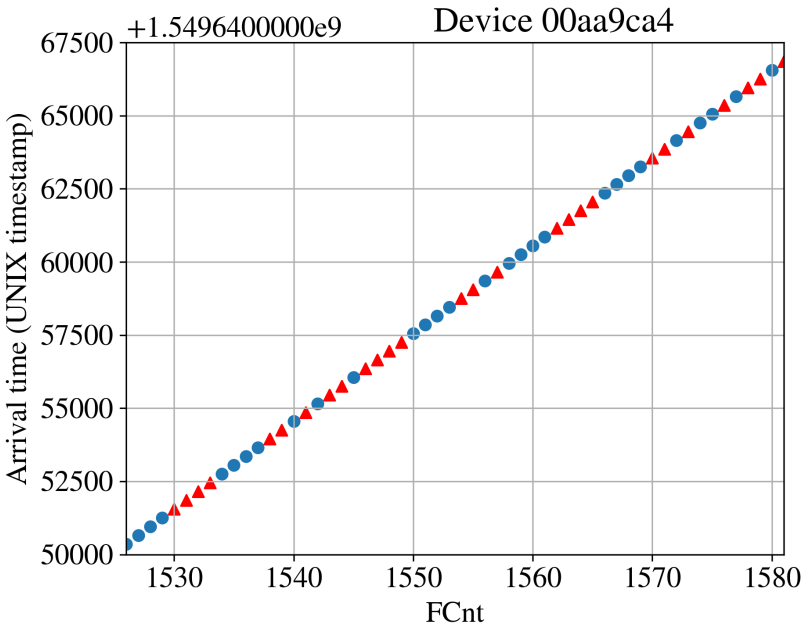
(c) Inter-arrival times averaged for each device

(d) Inter-arrival times averaged for each device zoomed on times less than 1000 seconds

Figure 4.12. Inter-arrival times of packets transmitted by devices, as a probability density function. Figure 4.12a shows the total distribution considering all the devices contained in the dataset. Figure 4.12c shows the distribution of the average inter-arrival time for each device. In this way, smaller inter-arrival times, which imply more frequent packets, get scaled down. Figures 4.12b and 4.12d reports respectively the same data but only for inter-arrival times less than 1000 seconds.



(a) Observed packets. The missing values of FCnt indicate the presence of lost packets.



(b) Observed packets alongside the prediction of the timings of the lost packets, in red.

Figure 4.13. Packets transmitted by a sample device `00aa9ca4`. On the x-axis the FCnt value is reported and on the y-axis the absolute time of arrival of the packet.

Looking at the behavior of RSSI, it's evident that all the GWs make efforts to optimize transmission parameters for effective packet forwarding. Additionally, we can examine the number of devices captured by the GWs, a factor closely tied to GW placement and operational duration, as discussed in Section 4.3.3. Notably, GW5 and GW6, operating for 552 and 573 days respectively, managed the highest number of devices. Conversely, GW4, operational for 15 days, and GW7, for 17 days, show significant variations in the number of devices managed. GW7 handled 12,969 distinct devices, whereas GW4 managed only 31, likely due to its unfavorable placement with limited line-of-sight.

Another critical aspect of LoRaWAN networks is the join procedure, which allows end-devices to connect to the network. In Section 4.3.4, we initially determined the ratio between the number of DevAddr and the number of distinct physical devices, computed by counting unique DevEUIs. This ratio is approximately 33.68, indicating that, on average, each physical end-device has roughly 33.68 different DevAddrs associated with it. To validate this, we also calculated the ratio between the number of completed join procedures and the number of physical devices, resulting in an average of 32 completed join procedures per device.

Additionally, our observation of join messages revealed that the ratio between the number of join-accept packets and the number of physical devices is approximately 0.41. This implies that only 41% of the devices in the network either join for the first time or use the OTAA procedure. The remaining 59% were already part of the network before we began collecting packets in the dataset or joined the network using Activation by Personalization (ABP).

Moreover, we moved on analyzing the duty-cycle. As discussed in Section 4.3.5, we measured the duty-cycle for each device. Remarkably, we found that all the devices observed in the network adhered to the protocol's constraint of a 1% duty-cycle.

Then, we explored the concept of de-duplication window time, which represents the duration during which a NS waits for duplicate packets. Given the absence of clear specifications regarding this time, we sought to determine an optimal de-duplication window duration for our specific dataset. This was achieved by collecting groups of duplicate packets and calculating a suitable de-duplication time for each group, as outlined in Section 4.3.6. Our analysis led to the identification of a global optimal de-duplication window duration of 200ms, which should be suitable for our dataset. However, it's important to note that the choice of this duration may vary when applying the same methodology to a different dataset, as it depends on the specific traffic patterns within the network. Notably, the value of $T_D = 200$ms that we found, is the actual default value used in the most widely known NS implementations [38], [15], [39].

Finally, in Section 4.3.7 we have shown that the majority of the devices exhibits a periodic behavior. In other words, the transmission of the packets, generally is performed at definite and predictable times, varying device by device.

The findings presented in this section will serve as a basis for two security vulnerabilities that we identified in LoRaWAN. The first one is presented in Chapter 5, where an attacker can exploit the lengthy duration of the de-duplication window to architect a wormhole attack denying the service of one (or more) victim EDs. This closely relates with the de-duplication phase and the value T_D of the de-duplication

window. The second one is presented in Chapter 6, where an attacker could be able to identify the hidden DevEUI of a device by observing the transmitted packets containing the DevAddr. This is based on the easy prediction of the time of arrival of the packets, which enables an attacker to find the next DevAddr assigned to a device after a Join procedure.

Chapter 5

LoRaWAN protocol vulnerabilities

In this Chapter we show a first LoRaWAN vulnerability which allows an attacker to perform a Denial-of-Service attack towards a target device, making it unable to receive downlink communications coming from the NS. This security concern exploits the vulnerable procedures of de-duplication, which has been analyzed previously in Section 4.3.6, and downlink GW selection, which will be covered in this Chapter. Our attack is based on manipulating radio indicators used by the NS to control the GW selection, allowing an attacker to control downlink path selection and potentially hijack it for malicious purposes.

It's important to note that the LoRaWAN standard does not specify the downlink GW selection process, leaving it to specific implementations. We have analyzed several widely known NS implementations, such as ChirpStack [15], The Things Network (TTN) [38] and Gotthard [39], and we have discovered that they use a procedure based on RSSI and SNR for downlink GW selection, which is vulnerable to our attack. Moreover, we note that this vulnerability is applicable not only on the legacy version of LoRaWAN 1.0, but also to the newest LoRaWAN 1.1.

5.1 A new DoS exploit

We now introduce our novel attack for LoRaWAN networks. To understand its workings, we first introduce in Section 5.1.1 the GW selection procedure that a LoRaWAN server applies to scheduled downlink messages. Then in Section 5.1.2 we illustrate the setting and the scenario that must be set up by an attacker in order to carry out the attack. At the end of the section we illustrate the procedure of the attack itself.

5.1.1 Downlink Gateway Selection Procedure

In the context of LoRaWAN, specific terminology is used to distinguish between different types of packets and processes:

- **Uplink Packet**. This term refers to a LoRaWAN packet sent by an ED and received by one or more GWs.

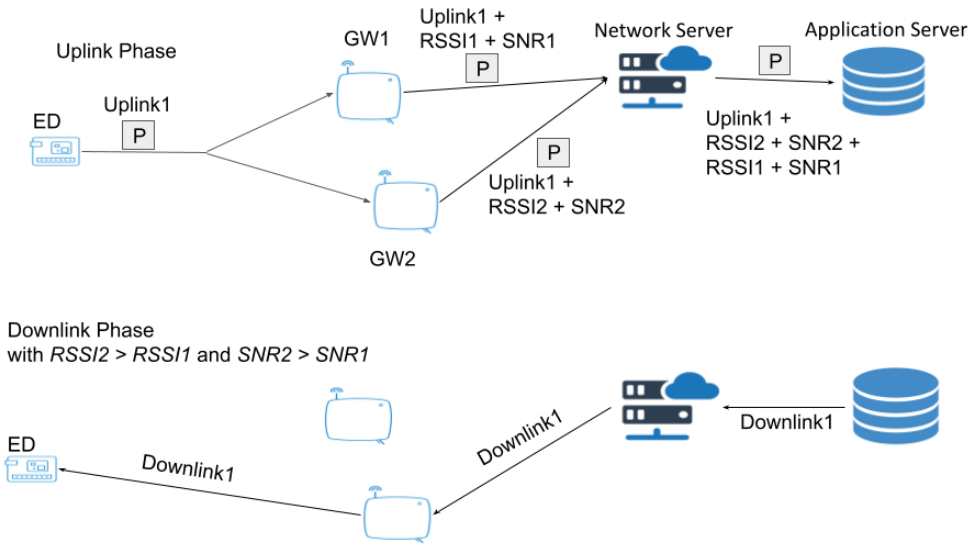


Figure 5.1. LoRaWAN Architecture with an uplink, de-duplication phase and the following downlink selection

- **Downlink Packet**. On the other hand, a downlink packet is one sent by the NS towards an ED.

In LoRaWAN networks, it's common for multiple GWs to receive the same uplink packet. To address this, the NS performs a de-duplication procedure, which is particularly valuable in scenarios with a large number of deployed GWs, as already pointed out in Section 4.3.6.

The LoRaWAN standard does not specify how de-duplication should be handled, leaving this aspect to the developers. However, well-known open-source implementations like ChirpStack [15], TTN [38], and Gotthard [39] often utilize a default de-duplication window of $T_D = 200ms$. Within this window, which starts from the reception time of the first packet, the NS processes packets received from all GWs, as shown in Figure 5.1.

At the end of this de-duplication window, if a downlink packet needs to be scheduled, the NS has to select a specific GW to transmit it to the ED. Indeed, in LoRaWAN there is no handshake mechanisms between a GW and an ED: the ED transmits its uplinks in broadcasts which may be received by every GW and every GW can transmit downlinks to every ED. In the case of downlink, in order to not overload the network, only one GW is selected to transmit the downlink. To optimize communication quality, the NS selects the downlink GW the one that received the last uplink with the best radio parameters, such as the one with the highest RSSI and SNR. This ensures that the chosen GW provides the most favorable conditions for communication. Figure 5.3 offers a qualitative overview of this procedure.

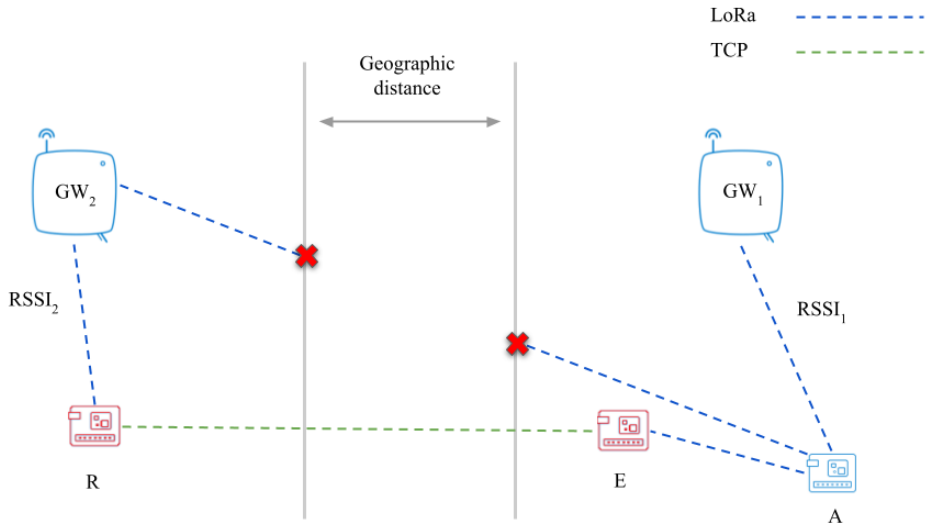


Figure 5.2. Scheme of the attack. Device E, R are two malicious devices. Device A is the victim device and GW_1, GW_2 are two legitimate GWs connected to the same NS. The gateway GW_2 is out of range of A. Device R is placed in such a way to have an excellent radio connection with GW_2, for example could be placed in the neighbors of GW_2, so to obtain $RSSI_2 > RSSI_1$.

5.1.2 Attack Setting

The described attack is based on the nearly instantaneous replay of an uplink message, such as a Join Request, in close proximity to a geographically distant GW. This attack can manipulate the NS's de-duplication phase, causing it to select the remote GW for sending the downlink response, such as a Join Accept. As a result, the device, referred to as the victim device (A), will be unable to receive expected downlink messages from the NS, including critical ones like the Join Accept after a Join Request.

In Figure 5.2, the physical architecture of our Time Window Replay (TWR) attack is depicted. Here, A represents the victim device, while E and R are two malicious devices under the attacker's control. Additionally, GW_1 and GW_2 are two legitimate GWs connected to the same NS. A is assumed to be entirely unaware of this potential attack.

Let's define the following components and conditions:

- GW_1: Gateway capable of receiving packets from A.
- GW_2: Gateway sufficiently distant from A, preventing A from receiving packets sent by GW_2 and vice versa.
- E: Device in close proximity to A, capable of eavesdropping on the packets sent by A.
- R: Device in close proximity to GW_2, such that the RSSI of the packets sent by R to GW_2 ($RSSI_2$) is greater than the RSSI of packets sent by A and received

by GW_1 ($RSSI_1$).

Now, let's introduce some notation:

- p: A LoRaWAN uplink packet;
- $RSSI_1(p)$: The RSSI of packet p as observed by gateway GW_1;
- $RSSI_2(p)$: The RSSI of packet p as observed by gateway GW_2;
- $rs_1(p)$ and $re_1(p)$: The time instants when the LoRa reception starts and ends at gateway GW_1 for packet p;
- $rs_2(p)$ and $re_2(p)$: The time instants when the LoRa reception starts and ends at gateway GW_2 for packet p;
- $ToA(p)$: The ToA of packet p, defined in Equation 4.5.

Additionally:

- E and R can communicate using the backhaul network.
- T_R: The link delay between E and R using the backhaul network.
- T_{NS}: The communication delay between the NS and the GWs, assumed to be constant and independent of the GW (i.e., all GWs experience the same connection delay).
- T_H: The time required by the hardware for internal functions unrelated to communication.

5.1.3 TWR Attack

The whole attack flow is reported in Fig. 5.4. Instead, the flow of the packets without the attack taking place is reported in Fig. 5.3.

The attack procedure starts when A sends an uplink packet p_1, which could be either a confirmed/unconfirmed uplink packet or a Join Request. Device E eavesdrops the packet and immediately forwards it to R using the backhaul network. Device R receives the eavesdropped data after the transmission time T_R. Finally, R replays the received packet on gateway GW_2. We name the replayed packet as p_2.

We now analyze the times in which these receptions take place. A packet p is received by the NS when the LoRa packet has been completely captured at time $re(p)$ by a GW and the GW forwards it to the NS, after a link delay of T_{NS}.

We express as T_A the attack time, which is the time difference between p_1 and p_2 being received by the NS:

$$T_A = re_2(p_2) + T_{NS} - (re_1(p_1) + T_{NS}) \tag{5.1}$$

If the following condition is verified the attack is successful:

$$T_A < T_D \tag{5.2}$$

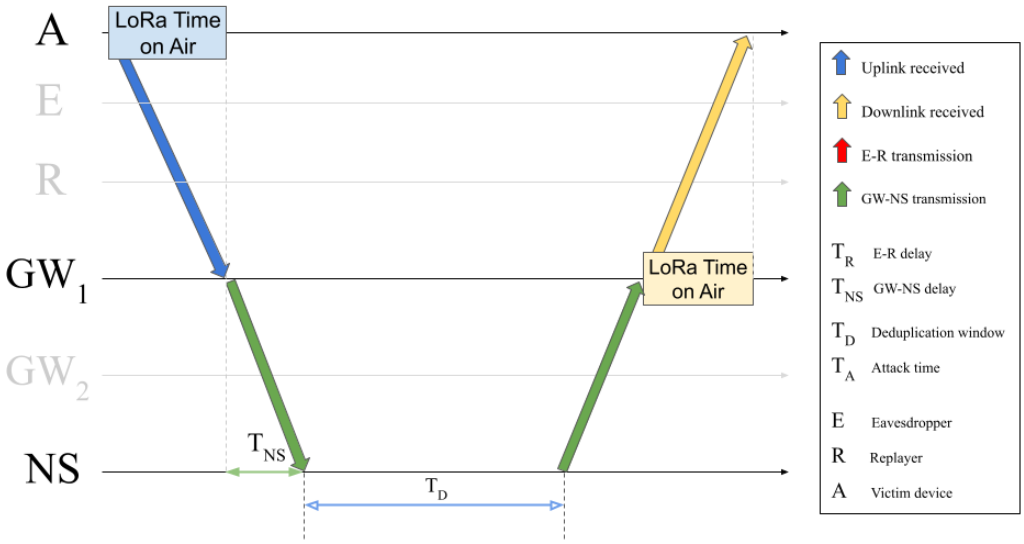


Figure 5.3. Scheme of the standard behaviour in the uplink/downlink procedure: the device A sends an uplink and it is received by the gateway GW_1 that on its turn forwards it to the NS. The NS after waiting for a de-duplication time period T_D, i.e. the period in which duplicate copies may be received, sends back to GW_1 the message for the downlink and this is finally received by A.

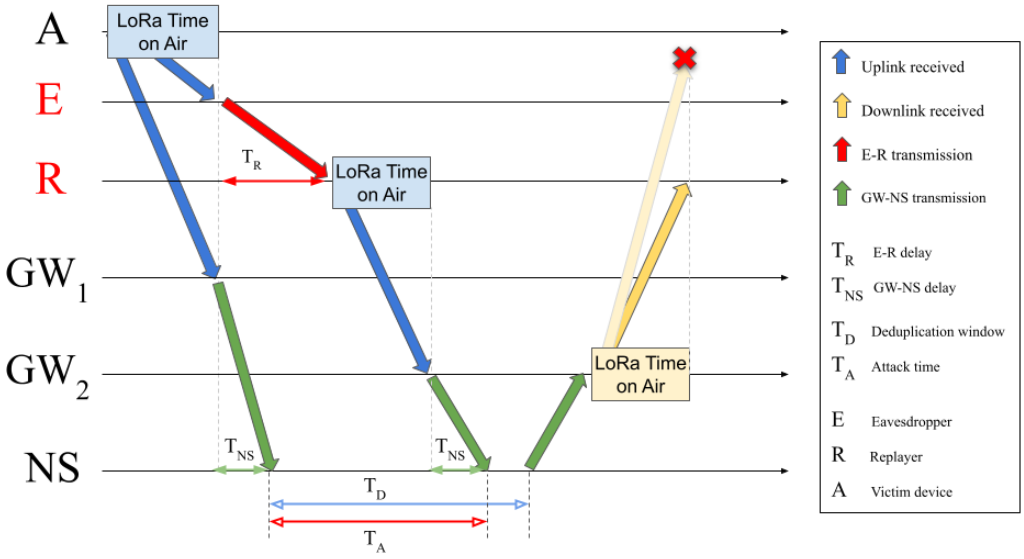


Figure 5.4. Scheme of the uplink/downlink procedure with TWR attack. The victim device A sends an uplink which is received by the gateway GW_1 and the malicious device E. The eavesdropped packet is sent to the malicious device R using the backhaul network. Finally, R replays the received LoRa message to GW_2. By construction, the uplink received at GW_2 has a greater RSSI than the one received at GW_1, therefore the NS will schedule the downlink on GW_2. However, GW_2 is out of reach for device A, therefore the downlink sent by GW_2 won't be received by A.

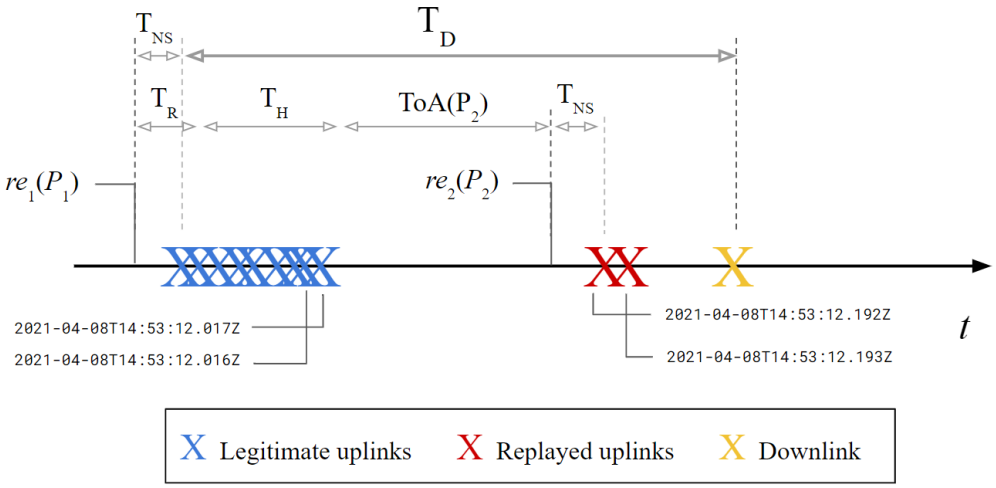


Figure 5.5. Different time instants on the timeline as experienced by the NS. The blue crosses are legitimate duplicate uplinks arriving at the NS from different GWs, the red ones represent the replayed uplinks and the yellow one is the single scheduled downlink after the de-duplication window T_D.

In this case, the two packets p_1, p_2 would fall inside the de-duplication window of the NS. The NS will consider p_2 as a duplicate of packet p_1, and it will consider it in the de-duplication procedure. By construction, we have $\text{RSSI}_2(p_2) > \text{RSSI}_1(p_1)$. This makes the NS select GW_2 for the downlink reply.

An equivalent expression of the attack time T_A is the following:

$$T_A = T_R + T_H + \text{ToA}(p_2) \tag{5.3}$$

where T_R is the time required for E to send on the backhaul the eavesdropped packet p_1, $\text{ToA}(p_2)$ is the ToA required by R to replay on LoRa p_2, and T_H is the hardware processing time required by devices E, R, which is not related to data transmission/reception. The times are plotted in a timeline in Fig. 5.5.

If this attack is successful, it makes A unable to receive a downlink from the NS. An example of the exploitation of this vulnerability is if this scheme is applied during a join or re-join procedure. In this case, A will not be able to receive the Join Accept message.

5.2 Effects

In this section, we describe in detail the effects that this attack would cause.

LoRaWAN foresees downlink traffic for different reasons, as identified in [25], including ACKs of confirmed uplink traffic, join accept messages, and messages including MAC commands [61].

If the NS is configured to send downlinks using the GW with the best transmission parameters, the attack described in the previous section will make NS always schedule the downlink to GW_2, which is not in the visibility of the victim device A. Since A will not receive the response from the NS, it will reschedule the transmission of the

packet, according to the LoRaWAN parameter NbTrans [58]. These retransmissions should obey the duty-cycle limitations imposed by LoRa regional standard or the NS itself. This means that the Join procedure may last a lot of time or even be aborted at a given moment.

The effects of blocking downlink communications are multiple and span through a wide range of possibilities, which in the long run may cause serious problems for the whole LoRaWAN network. Starting from the simplest and most immediate ones: blocking the downlink of a (re)join request blocks the attempts of an ED performing a join procedure, making it send requests over and over in the hope of eventually joining the network. It also makes the ED waste DevNonce values that will never be refreshed, shortening the lifespan of the device. Finally, this mechanism makes the victim waste energy resources.

An important role in the LoRaWAN protocol is played by various MAC commands, used by the ED or the NS to exchange configurations used to communicate, like changing spreading factor, frequencies, or data rate, and these commands must be acknowledged by the receiver. If this attack scheme is carried out:

- MAC commands requests sent by the NS will not be received by the ED.
- MAC commands requests sent by the ED will be received by the NS but the ED will not receive the response from the NS.

This makes some MAC commands be continuously retransmitted.

A key functionality that may be affected by this attack is the Adaptive Data Rate (ADR) [37]: if a device does not receive an answer after sending ADR_ACK_LIMIT and ADR_ACK_DELAY, the ED will increase the Spreading Factor (SF) and transmission power, in the hope of reaching the GW and obtaining an acknowledgment for its communications. If any downlink is not received by the ED, it will continue to increase the SF. A side effect is that, by selecting a SF that may not be the optimal one (e.g., SF9 instead of 7), the ToA is increased, and the overall performance of the network deteriorates. The scaling up of SF increases the probability of packet collisions in a crowded IoT scenario, causing more network resources to be wasted. Moreover, the extra energy required for transmitting at a high SF shortens the lifespan of the devices.

Second, it may cause problems with the duty-cycle restrictions of LoRaWAN. In LoRaWAN, a device is bound to a small duty cycle, usually around 1%, which is around 14 minutes of cumulative ToA every day. Even stricter regulations are posed on the re-transmission of ED uplinks requiring acknowledgment or answer, as described in the Retransmission back-off chapter in LoRaWAN 1.1 standard [61].

Moreover, TWR can influence the quality of the communication, making the ED and the NS believe that they are closer or further from each other than they really are. On the one hand, an attacker can act as a bridge between the ED and the NS, receiving the hijacked communication with higher values of RSSI and SNR, leading ADR to settle on a low-power and low-SF communication. On the other hand, since TWR can cut the communication between the ED and the NS, the ED will believe it is not well covered by any GW, causing it to increase the SF.

These controlled changes in the SF can be exploited in two opposite ways:

1. Let's assume that A is not well covered by the LoRa network, and its uplinks cannot be received if it uses SF7. By replaying on GW_2, we fool the NS into thinking that A has a good value of RSSI, indicating good coverage. This will likely make the NS send a downlink containing a LinkADRReq command, instructing A to set its SF to 7. We then replay the downlink to A. Now A will send its following messages with SF7, and they will not be received by the network. Due to Adaptive Data Rate (ADR), A will begin raising the SF after

$$\text{ADR\_ACK\_LIMIT} + 2 \cdot \text{ADR\_ACK\_DELAY}$$

   uplinks, which have all been sent with SF7 and not received by the NS.

2. By not receiving any downlink, the ADR algorithm running on the victim device will continue to raise the SF. This increase in SF causes higher ToA and energy consumption, while putting an excessive load on the network.

Finally, this attack can be applied to exploit the well-known issue regarding the Join Accept replay vulnerability in LoRaWAN 1.0 [28], [86], [62], allowing exploitation in a simpler way than jamming the signal or assuming enough packets are lost in the transmissions. Using TWR as a subroutine, it is enough to hijack the first Join Request message, saving the received Join Accept. When the ED retries to send a Join Request, we apply TWR again, so that the ED will not receive the acknowledgment from the NS, and then replay the first received Join Accept. This allows us to misconfigure an ED, which calculates wrong session keys and generates invalid uplinks that will be ignored by the NS.

5.3 Experimental setup

To test the attack, we used three Libelium Waspmote PRO v1.5 devices equipped with Microchip RN2483A LoRaWAN extension boards. These devices acted as follows: one as the victim (A), one as the malicious replayer (R), and one as the eavesdropper (E), which listened for outgoing packets from device A. Additionally, we employed a Raspberry Pi 3B as the remote TCP server used to connect E and R.

Both the GWs and the replayer were equipped with custom code specifically designed to intercept on the default join-request and join-accept frequencies (868.1MHz, 868.3MHz, and 868.5MHz) and time windows (Rx1 after 5s and Rx2 after 6s). The internet connection used to bridge between the eavesdropper and the replayer was built on top of a domestic connection with a bit rate of about 30 Mbps. It's important to note that these devices were connected to the backhaul with Wi-Fi, introducing another set of delays and unreliability.

We conducted the attack testing relying on a LoRaWAN network provider in Italy in a black-box fashion. The victim device was a personal device normally configured to operate using the Over-the-Air Activation (OTAA) join procedure. It's worth noting that we had no access to the code deployed on the NS or the GWs.

In Fig. 5.6, we present the positions of the devices during the testing phase. On the right side of the figure, you can see a red dot representing the replayer board R and a green dot representing the downlink gateway GW_2 that we aimed to have

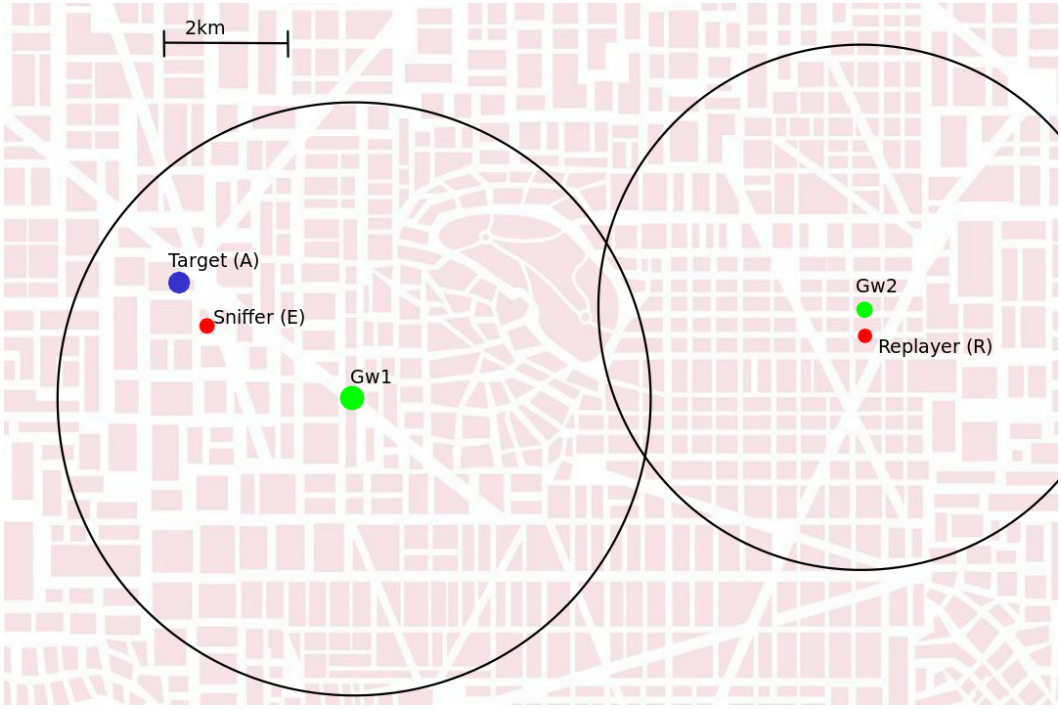


Figure 5.6. Geographical positions of the nodes during the test phase

chosen. On the left side, there is the victim device A that we intended to cut off from the network. The black rings around the GWs represent the approximate range of the GW in a S7 transmission. Gateway GW_2 is positioned on top of a building, and we placed R inside a nearby building.

5.4 Experimental Results

In this section, we present the results of our test implementation. We considered the test successful if we were able to hijack the join accept to our chosen gateway GW_2 instead of GW_1, which was within range of device A. We analyzed these results with respect to the RSSI, although the same principles apply to SNR.

As shown in Table 5.1, the SF plays a crucial role in the success rate of the attack. Specifically, an increase in SF results in a decreased success rate. Higher SF values lead to longer ToA for device R to replay packet p_2. This makes it less likely for the replayed packet to fall within the de-duplication window T_D. However, it's worth noting that this timing limitation could potentially be overcome by using specialized hardware, as described in Section 5.6.2. Moreover, we note that if a packet is sent using a specific SF, the attacker cannot replay it using a lower SF value: the SF value is used in the computation of the cryptographic MIC and changing it would invalidate the MIC itself.

Our off-the-shelf hardware introduced an average processing time T_H of approximately 47ms, which was unrelated to the attack. Additionally, the connection delay between R, E, and the Raspberry Pi was unstable due to the Wi-Fi connection. The partial unreliability of the setup was chosen specifically in order to simulate the

Table 5.1. Attack implementation results

Parameters	**SF7**	**SF8**	**SF9**
Total Attempts	144	42	54
Successful Attempts	128	19	5
Success Rate	95.8%	45.2%	9.3%
ToA (ms)	62	113	206
Avg. T_R (ms)	34	44	13
Avg. T_H (ms)	53	46	43
Avg. T_A (ms)	149	203	262
Std. dev. T_A (ms)	55	70	113

worst conditions that an attacker may face.

With SF7, where we had the smallest ToA, we achieved a success rate of approximately 95% in our tests. In these cases, the NS selected GW_2 instead of GW_1 as the gateway for downlink transmission. Failures in these tests were primarily attributed to network congestion, which occasionally caused a significant increase in transmission time, as indicated by the variation in T_R. With SF8, the average attack time T_A approached the critical 200ms mark, and we observed an almost 50% success rate. This suggested that the LoRaWAN provider had a de-duplication time T_D set to 200ms. For SF9 and higher values, the ToA became too long to fit within the de-duplication window, meaning that T_A exceeded T_D. To make this scheme work with higher SF values, one might consider using specialized hardware, as discussed in Section 5.6.2.

5.5 Countermeasures

After identifying and testing this type of attack, it's essential to consider potential countermeasures to mitigate or prevent such attacks in LoRaWAN networks

5.5.1 Attack Detection

To ascertain the occurrence of a TWR attack, one viable approach is to conduct an analysis of the timestamps associated with uplink receptions at the GWs. Due to the inherent delays introduced by our off-the-shelf hardware, including T_R, T_H, and ToA delays, a noticeable time gap emerges between legitimate packets and maliciously replayed ones. This temporal discrepancy is evident in Fig. 5.5, facilitating the identification of a distinct cluster of "delayed packets" through the use of various anomaly detection methods.

Even when an attacker employs specialized hardware, note that the attack time T_A could not be lower than T_R, as shown in Equation 5.3. This temporal factor should not be underestimated, particularly considering that the eavesdropped packet, intercepted by E, must be transmitted to R, which may be situated remotely.

5.5.2 Attack Mitigation

To address the potential consequences of the attack, we've explored several solutions and considered their drawbacks:

1. **Resize de-duplication window T_D.** with a smaller window, a malicious third party would have less time to replay the sniffed message, making the attack more difficult to succeed. However, a small value of T_D can have impacts on the performance of the network: packets could arrive to the NS from a sub-optimal GW and the same duplicate packet from an optimal GW. If the packet arrives from the sub-optimal first and from the optimal one after a time greater than T_D, the optimal GW is discarded from the selection. Numerical comparison on the choice of T_D was already covered in the previous chapter, in Section 4.3.6.

2. **Send Downlink on a Subset of Gateways.** While this straightforward solution is easy to implement, it has significant downsides. It could lead to network overload and an increased probability of packet collisions.

3. **Static Downlink Gateway Assignment.** Statically assigning downlink GWs for each device might be effective but has its limitations. It can be cumbersome to configure, lacks scalability, and is only suitable for static, non-mobile devices.

4. **Schedule Response Downlink on the First Received Uplink Gateway.** This approach schedules the response downlink on the GW that first received an uplink. It assumes a constant link delay between GWs and the NS, which should work but doesn't optimize the chances of ED reception.

5. **Past Downlink Paths Analysis.** Utilizing an anomaly detector, this method seeks clusters of late packets or unusual GW activity to detect ongoing attacks. It can adapt the de-duplication window T_D based on device history, allowing the NS to ignore unusual GWs that received uplinks.

6. **Randomized Gateway Selection.** This approach always includes a random component in the choice of the downlink GW. Instead of deterministically selecting the GW with the best radio parameters, it chooses one with a probability proportional to signal quality. While it doesn't always optimize ED reception probability, it ensures that the ED can eventually receive a downlink. For instance, the ChirpStack NS [15] already incorporates a degree of randomness in its GW selection policy. It randomly selects a GW from those that received the packet with an SNR value above a threshold. If no GW meets the SNR constraint, it always selects the GW with the best SNR. However, the default SNR threshold value is relatively high and achievable only if the device is in close proximity to a GW, such as in the case of the replayer device R.

5.6 Further discussion

5.6.1 Scalability

The attack scenario we've discussed can potentially target more than one LoRaWAN ED. Here are some considerations:

1. **Eavesdropping on Multiple Devices.** Device E, configured as an eavesdropper, has the capability to listen for any LoRa packet in the air, not limited to packets sent by a single device A. This means it can capture packets from multiple EDs and send them to R for potential replay attacks.

2. **Multiple Replay Device.** Extending the attack architecture is possible by employing more than one replay device, denoted as $R_1, R_2, ...$ These additional replay devices can be strategically positioned near different GWs. The advantage of this approach is that if the downlink GW selection follows a probability distribution based on signal quality, the attacker's chances of success increase. This is because many GWs could receive the replayed packet with high signal quality.

3. **Off-the-Shelf Hardware.** Importantly, we've demonstrated that this attack can work with off-the-shelf hardware, making it relatively simple to implement and cost-effective to scale.

In summary, the attack scenario has the potential to impact multiple EDs and can be further refined and expanded, making it a significant security concern for LoRaWAN networks.

5.6.2 Specialized Hardware

The success rate of the TWR attack tests was notably influenced by the choice of SF. Specifically, a higher SF led to a longer ToA, resulting in an extended attack time T_A, as indicated by Eq. 5.3. Higher SF values were found to cause the TWR attack to fail when using our off-the-shelf hardware.

However, a potential way to overcome this limitation is to employ specialized hardware, like a Software Defined Radio (SDR). With SDR technology, it becomes possible to replay a LoRa packet in real-time, rather than performing store-and-forward. In this enhanced scenario, LoRa chirps would be immediately transmitted from the eavesdropping device E to the replay device R via the backhaul, and then instantly replayed by R using LoRa. R would need to buffer the incoming chirps to ensure a consistent emission rate.

In this scenario, the new attack time (T_A^*) would be calculated as follows:

$$T_A^* = T_R + T_H + T_B \tag{5.4}$$

Here, T_B represents the buffering time, which can be determined by addressing the underlying Delay Equalization Problem [7]. It's worth noting that T_B should be significantly smaller than the ToA of a LoRa packet, ensuring that $T_A^* < T_D$ virtually always, where T_D is the de-duplication window.

In essence, utilizing specialized hardware like SDRs could potentially make the TWR attack more effective, even with higher SF values, by minimizing the attack time and increasing the likelihood of packets falling within the de-duplication window.

5.7 Findings

In this chapter, we introduced the TWR attack, which takes advantage of a weakness in the LoRaWAN de-duplication process to disrupt the downlink communication between the NS and an ED. It's important to note that the de-duplication procedure is not precisely defined in the LoRaWAN standard, leaving it implementation-dependent across different NSs. Some widely-used open-source implementations schedule downlinks on GWs that have received uplinks with the best radio parameters (e.g., SNR, RSSI). This characteristic opens up the possibility of executing a replay attack on a distant GW, redirecting the downlink path to prevent the victim device from receiving it.

We implemented and tested this attack on a LoRaWAN network provider, using our own boards as target devices. Furthermore, we explored the potential consequences of this attack and discussed various mitigation strategies. This attack has been demonstrated as straightforward to execute and can be scaled to target multiple LoRaWAN devices simultaneously.

We believe that the LoRaWAN standard should be extended to include guidelines for the de-duplication procedure to address this vulnerability effectively. Moreover, since this procedure can be leveraged to easily exploit the well-known vulnerability of Join Accept replay in LoRaWAN 1.0, our attack represents another strong reason to migrate from LoRaWAN 1.0 to 1.1.

Chapter 6

LoRaWAN privacy leaks

In this Chapter we show a second LoRaWAN vulnerability which concerns privacy leakages. The results of the traffic analysis presented in Chapter 4, specifically in Section 4.3.7, show that many devices follow some sort of periodic behavior in the transmission of their uplinks. This periodicity can be exploited by a malicious attacker by predicting, with a high degree of accuracy, when the next uplink is broadcasted. This prediction allows the attacker to re-identify an ED after the change of its security context. More specifically, the attacker could be able to link the DevAddr to the hidden DevEUI of an ED, performing a de-anonymization attack. This information leakage brings numerous privacy concerns, as already highlighted in Section 3.5.2.

Throughout this Chapter we introduce our de-anonymization algorithm. In the next Chapter we will show how our algorithm can be re-engineered as an IDS detecting if such an attack is possible given the current network situation.

6.1 Privacy Leaks

As discussed earlier in Section 3.5, LoRaWAN underwent extensive security research and vulnerability assessment. One of the most security-critical aspects of the LoRaWAN protocol, is the Join and activation procedure. Both activation methods offer a high level of security through symmetric encryption during message exchanges between an ED and a NS. However, potential attackers can compromise the ED's identity, leading to two distinct security threats: ED identification and privacy leaks.

In LoRaWAN, the inbound and outbound traffic to and from the GW can be monitored in order to acquire ED information including user activities of the people that are related to the ED. One of these aspects is the identification, by simply eavesdropping on the traffic, of the DeVEUI that is a globally unique identifier assigned by the manufacturer, or the owner, of the ED. Since the LoRaWAN protocol is designed to use a temporary address, named DevAddr, during the communications (for security and privacy issues) the possibility to derive this parameter is quite critical. Included in every data packet, the tracking of the DevAddr can be used from an eavesdropper to associate traffic activity to entities allowing him to derive information by leveraging the temporal feature of the traffic.

The DevEUI can be leveraged to infer information on the ED (manufacturer, type,

or the associated application and sensor) that is only exposed during the association process. By linking a DevAddr with a DevEUI, an adversary could combine the information brought by each element and thus increase its knowledge on the ED, its activities and the associated applications. For example, the forwarded consumption data can be analyzed using load monitoring techniques to infer activities of the consumers, which is considered to be sensitive data that must be protected for preserving the consumers' privacy [90], [99].

The same concern was investigated in [4], where a method for associating the DevAddr with a DevEUI is proposed. This technique relies on analyzing the time sequence of the Join request emitted by an ED and the first message sent by the same ED after the Join request. This approach, known as temporal linking, differs from our proposal, which presents a pattern-based methodology for characterizing an ED based on the entire sequence of transmitted packets and their patterns. Furthermore, it's worth noting that the temporal linking between the Join and the first uplink message is not always consistent. There may be instances where the NS instructs the ED to immediately transmit a ReJoin request using the MAC command ForceRejoinReq. Since this is a MAC command, it remains concealed from the application layer and is instead managed by the LoRaWAN stack. Consequently, the ReJoin message and the first application uplink may occur at different temporal intervals.

In this Chapter we propose our algorithm named DEVIL (DEVice Identification and privacy Leakage) which performs a de-anonymization attack linking the DevAddr to a DevEUI of an ED. We conduct a comparative analysis between DEVIL and the algorithm introduced in [4]. Our results demonstrate that, in the best-case scenario, our approach improves accuracy by 58%. Furthermore, as outlined in the introduction, DEVIL can be harnessed in a constructive manner as well. It can function as an auxiliary module that assists the NS by alerting the network operator whether it detects that a de-anonymization attack is possible, offering potential mitigation strategies and network optimization recommendations based on externally derived ED profiling.

6.2 Datasets Available

In this section, we present an analysis of various datasets derived from real-world IoT application scenarios, particularly emphasizing the periodic nature of traffic generated by typical IoT application services like metering applications. Our study encompasses four distinct datasets:

1. Real-world LoRaWAN Datasets by UNIDATA S.p.A.: These datasets were collected by UNIDATA S.p.A., an IoT operator overseeing various application services in Italy. There are two real datasets:

   - An energy meter application dataset, involving 130 EDs.
   - A water meter application dataset, comprising 300 EDs.

   Both of these services operate in Italy, and our analysis covers 18 months of data from January 2019 to April 2020. The energy metering EDs rely

on tick-counters connected to energy distribution meters to monitor energy consumption. Conversely, the water metering EDs are equipped with embedded LoRaWAN transceivers.

2. LoRaWAN at the Edge Dataset [11]: This dataset is open and publicly available, providing valuable insights into LoRaWAN traffic patterns at the edge of the network. The dataset was already introduced in Section 4.2.

3. Synthetic Dataset: This dataset is synthetic in nature and has been created for research purposes, allowing us to compare and contrast its behavior with the real-world datasets.

In the following sections we will introduce the characteristics of the aforementioned datasets useful for our analysis.

6.2.1 Energy and Water metering applications

In the energy meter application, all data transmission is done through a single GWs, and each ED typically forwards measurements approximately once per hour, on average. The entire dataset comprises a total of 1,131,685 packets. Figure 6.1 provides a visual representation of this data, with the blue line on the left y-axis illustrating the number of received packets per week for the entire network. Additionally, the red line on the right y-axis represents the number of Join request packets received per week.

On the other hand, for the water meter dataset, each ED generates approximately 18 packets per week. The coverage area for these 300 water meters is served by two GWs. Figure 6.2 displays the number of received packets for the first 130 EDs over the specified period. These EDs are sorted by SNR. It's noteworthy that the complete dataset comprises 929,587 packets. To ensure the accuracy of our analysis, we've focused on EDs with a high average SNR, as this selection helps exclude devices with an error rate exceeding 5%. This is crucial for interpreting the timing of events correctly, as devices with lower SNR may introduce timing inaccuracies in the data.

The analysis of the transmission patterns of the network devices, particularly in the context of water meter EDs, has revealed an interesting structure in the sent packets. Figure 6.3 provides a detailed view of a device's transmissions, where the x-axis represents the frame counter of each packet, and the y-axis represents the inter-arrival times expressed in hours. Throughout this analysis, we'll refer to the time elapsed between the arrival of two consecutive packets from the same device as the inter-arrival time.

From Figure 6.3, two key observations can be made:

1. Regular Pattern: There is a regular part in the transmission pattern, characterized by a cycle of 9 packets. Within this regular pattern, the first seven packets exhibit inter-arrival times of 12 hours each. However, the last two packets have smaller temporal distances, but their combined inter-arrival times still sum up to 12 hours (as seen in the figure, where they are 4 and 8 hours, respectively). Additionally, the inter-arrival times of the eighth and ninth packets remain consistent over time for the same device. The figure also highlights the absence

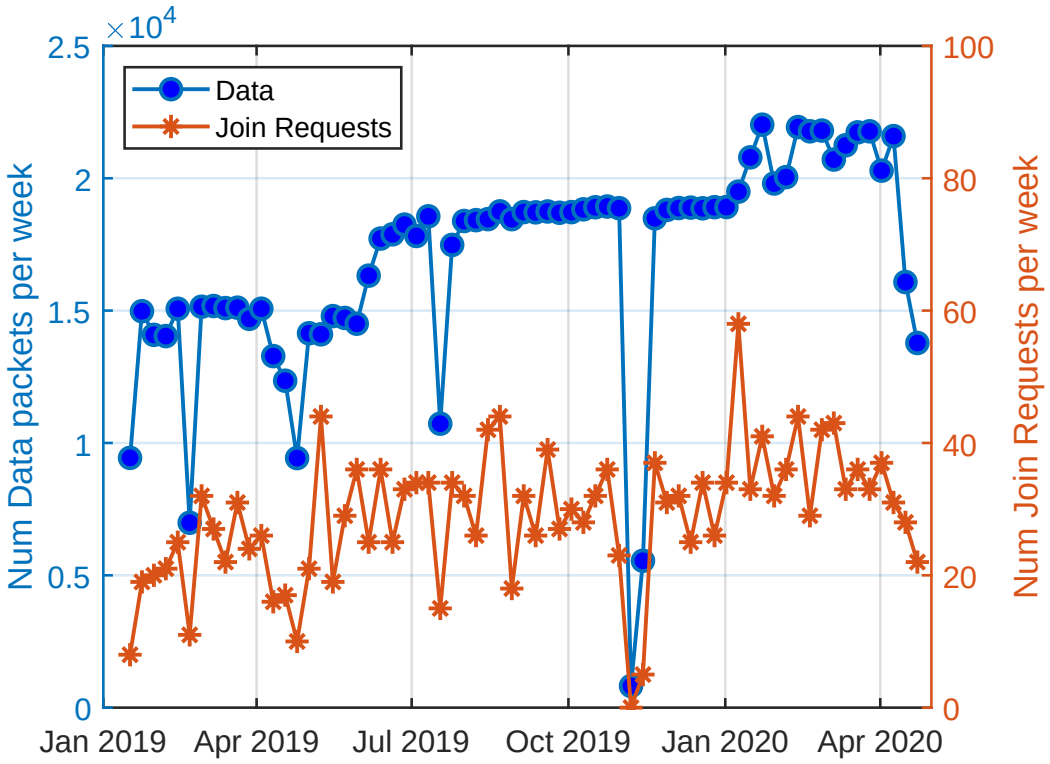


Figure 6.1. Number of received packets per week (blue line and left y axis), and number of Join request packets per week (red line and right y axis), in a period of 18 months for the energy metering applications service.

of a packet that has not been received by the NS (evident at the 256th frame counter value).

2. Non-Regular Part: The transmission pattern also contains a non-regular component. Further investigation has revealed occasional irregularities in the patterns, which follow a distinct characteristic. Occasionally, one of the packets in the regular pattern can be replaced by two or more packets. Figure 6.3 illustrates one such situation occurring at the 275th and 276th frame counter values.

Based on this analysis, it can be deduced that the total number of distinct inter-arrival times is limited. In Figure 6.3, 57 inter-arrival times (corresponding to 56 packets, spanning more than 3 weeks of operation) are shown, and only 5 different inter-arrival time values are observed.

To provide a more detailed depiction of this inter-arrival behavior, we specifically focused on the first 130 EDs, sorted by SNR. We isolated the packets sent by each device and constructed a sequence, referred to as the inter-arrival sequence, in which element i represents the time elapsed between packet i and packet $i+1$, accounting for any lost packets.

Using this inter-arrival sequence, we generated Figure 6.4, which illustrates the distribution of the number of distinct values of inter-arrival times per ED. We

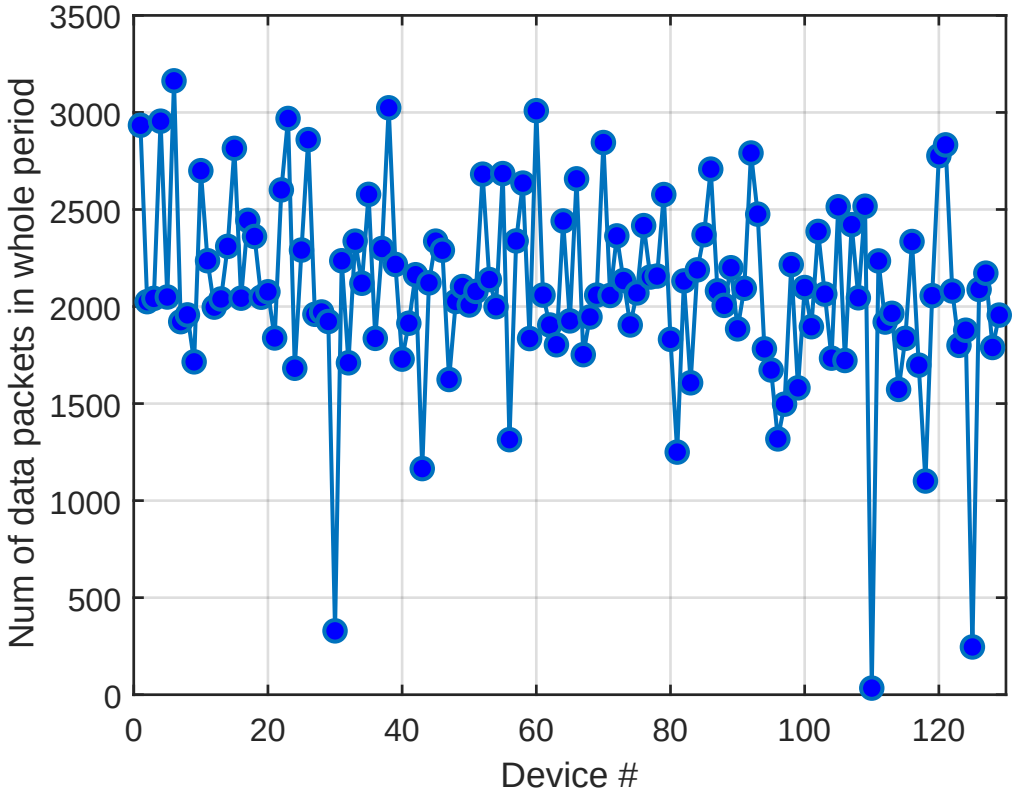


Figure 6.2. Number of received packets in a period of 18 months for the water meter application service.

considered the top 95% of the total occurrences for this analysis. From the figure, we can observe the following:

- The maximum number of distinct values of inter-arrival times per ED is 10;
- Over 85% of occurrences fall within the range of 3 to 7 distinct inter-arrival times;
- The most prominent bar corresponds to 5 different values of inter-arrival times per ED, which occurs in approximately 25% of the EDs. This finding supports our previous analysis.

In addition, Figure 6.5 provides a distribution of the inter-arrival time values. According to the CDF, approximately 27% of inter-arrival times are uniformly distributed within the range of 10 minutes to 12 hours (720 minutes). The remaining 73% of inter-arrival times consistently have a value of 12 hours. This result underscores the analysis of the pattern depicted in Figure 6.3. Similar observations can be made for the energy meter dataset.

6.2.2 LoED Dataset

The third dataset under consideration is the LoED open dataset [11], already introduced in Section 4.2. This dataset comprises packets generated by smart city

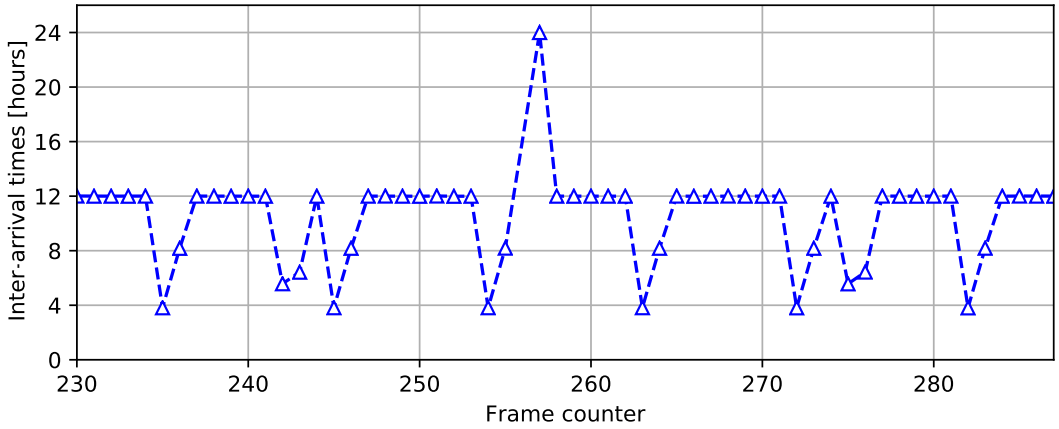


Figure 6.3. Time evolution of the inter-arrival times for the LoRaWAN water metering devices.

applications and research deployments over a period of 4 months. The data was collected by 9 GWs located in central London, resulting in a total of 11,263,001 packets originating from devices with 145,023 different DevAddr. LoED is an open dataset that offers valuable insights into the real-world operation of LoRaWAN in urban settings.

For our analysis, we filtered the total number of devices based on their SNR values, focusing on devices with a high average SNR to exclude those with error rates exceeding 5%. This filtering process resulted in a subgroup of 500 EDs. Within this subgroup, we identified two distinct types of behavior:

- Continuous Transmissions: Approximately 60% of the EDs in this subgroup exhibit continuous transmissions, characterized by inter-arrival times in the range of 1-10 minutes.
- Periodic Behavior: The remaining 40% of EDs in this subgroup display periodic behavior.

Figure 6.6 illustrates the CDF of the inter-arrival times for the EDs with periodic behavior over the entire observation period. Notably, 40% of the inter-arrival times are equal to 60 minutes, and other inter-arrival times are multiples of 60 minutes. This distribution confirms the periodic nature of packet transmissions for this subset of EDs.

We note that the major result that we found out when we analyzed this dataset in Chapter 4, is that the majority of the devices exhibit some sort of periodicity in uplink messages. The vulnerability that we will show in the rest of the Chapter exploits this periodicity.

6.2.3 Synthetic Dataset

Based on the previous analysis, we have observed that typical traffic generated by LoRaWAN EDs exhibits some sort of a periodic behavior. To further validate the proposed approach and assess its performance under different conditions, we

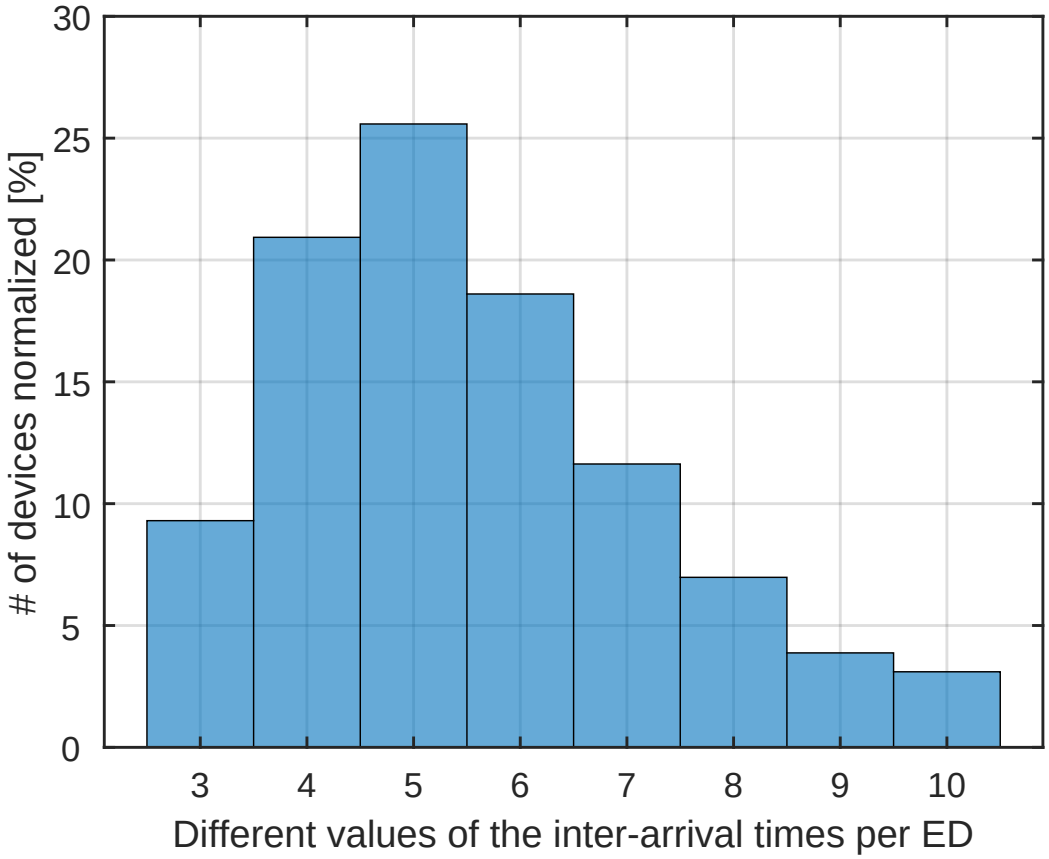


Figure 6.4. Distribution of different values of the inter-arrival times per ED, in a period of 18 months (we consider 95% of the total occurrences). We remark that the average number of packets for each ED in the reference period is 2000.

have constructed a synthetic dataset. The purpose of this synthetic dataset is to create a controlled environment that allows us to manipulate parameters and evaluate DEVIL's performance. The synthetic dataset is generated by specifying two parameters:

- The number of devices, denoted as N.
- The maximum length of a periodic pattern, denoted as S.

Each synthetic dataset is comprised of approximately 10^6 packets. For each of the N devices, we generate the inter-arrival time between each packet i and $i+1$ as following: first, we draw a time $\hat{t}_i$ from a specific inter-arrival time distribution, which will be highlighted in the next paragraph. Then, we introduce a random jitter, uniformly distributed between 0 and $\alpha \cdot \hat{t}_i$, where α is a scalar parameter.

$$t_i = \hat{t}_i + \text{uniform}\left(0, \alpha \cdot \hat{t}_i\right) \tag{6.1}$$

The scalar α is a noise parameter and regulates how much the generated inter-arrival times follow the underlying distribution extracted from the LoED dataset.

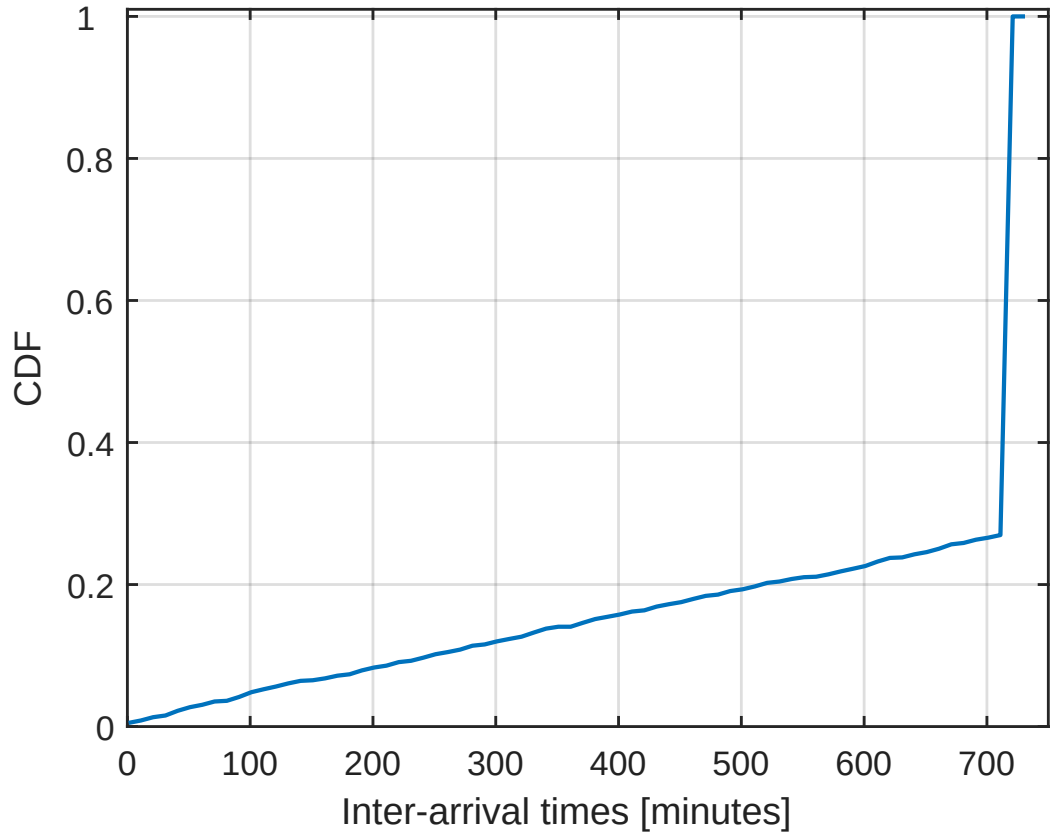


Figure 6.5. CDF of the inter-arrival times in a period of 18 months for the water metering application.

When we had to choose which distribution to use for the inter-arrival times, we thought the following. We have the LoED dataset, as already described in a previous Chapter in Section 4.2 which can serve us as a random sample of the population of LoRaWAN devices. Therefore, we can extract from the dataset the distribution of inter-arrival times for each device and model our synthetic traffic dataset on top of it.

6.3 Attacker Model and Features Extraction

Table 6.1 provides a summary of the relevant LoRaWAN fields utilized in DEVIL. These fields have been extracted and pre-processed to serve as key features in our analysis. The targeted EDs should be LoRaWAN v1.1 compliant. In this LoRaWAN v1.1 compliant scenario, EDs may occasionally transmit Join or ReJoin requests to the NS. The NS, in turn, responds with a Join accept message that contains a new DevAddr for the ED. In LoRaWAN communications:

- The Join request contains the DevEUI in plaintext;

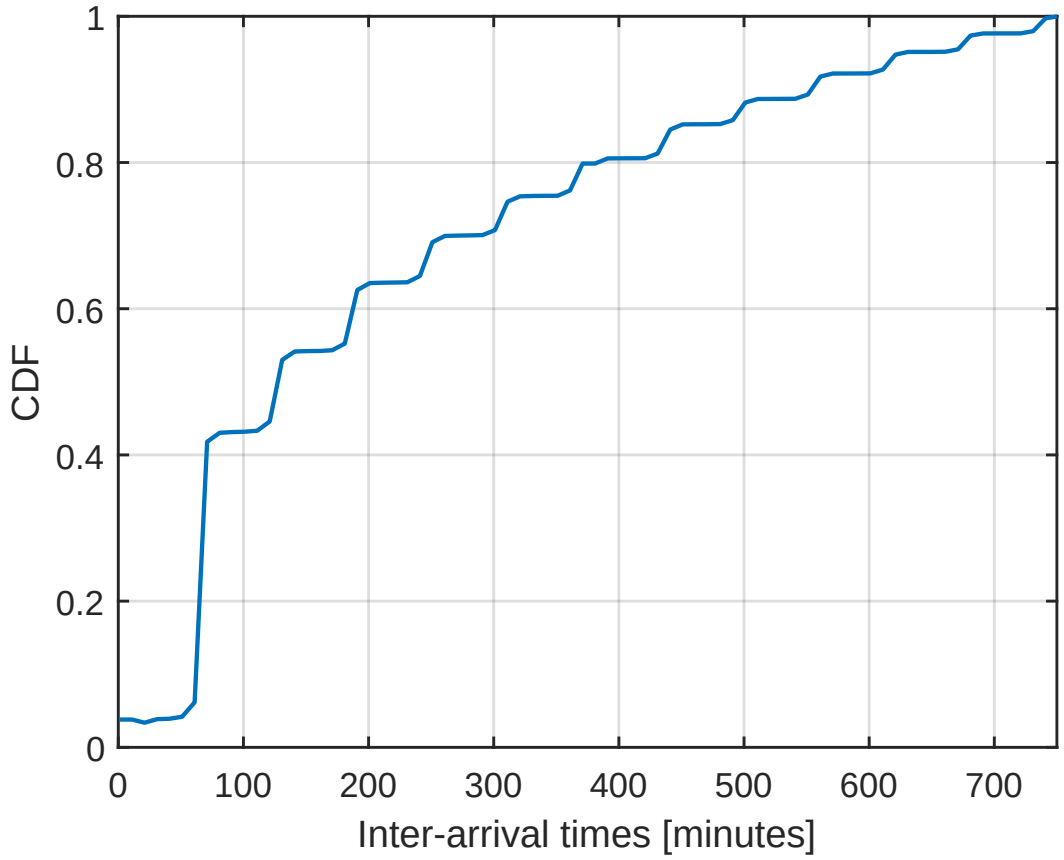


Figure 6.6. CDF of the inter-arrival times in the whole period of 4 months for 40% EDs in the LoED dataset.

- The Join accept message contains the DevAddr encrypted;
- All the uplinks (messages sent from EDs to the NS) and downlinks (messages sent from the NS to EDs) contain the DevAddr in plaintext.

The attacker's objective is to associate a LoRaWAN DevAddr to the real DevEUI of an ED by passively monitoring network traffic. To clarify the goal, let's define the following terms:

- e represents a specific DevEUI;
- a represents a DevAddr;
- $A(e) = a_1, a_2, \ldots$ denotes the set of DevAddr addresses assigned to the DevEUI e within a given time span.

The attacker's aim is to determine which DevAddr addresses belong to a particular ED's DevEUI, essentially inferring $A(e)$ for a target DevEUI e.

To begin this analysis, the available data described in the previous Section was examined. The packets sent by a specific ED were isolated, and an inter-arrival sequence was constructed. In this sequence, each element i corresponds to the time

Table 6.1. Key fields present in the datasets. Note that the field DEV_EUI is not used by DEVIL algorithm. Its purpose is to use it for cross checking and accuracy computation of the predictions of DEVIL. All the other fields are sent in cleartext in the LoRaWAN packet header and can be observed by passively listening to the traffic.

Parameter	**Description**
DEV_ADDR	Unique identifier of the ED in the network
DEV_EUI	Unique identifier of the physical ED (None if the ED is unknown)
FCnt	Frame counter: counter (increased by 1 for each packet sent from an ED)
TMST	Time of arrival of the packet
TYPE	Type of packet, specifies if the packet is data or Join request

elapsed between two consecutive packets, namely packets i and $i+1$. To ensure that two packets are considered consecutive, it is necessary that $\mathrm{FCnt}_{i+1} = 1 + \mathrm{FCnt}_i$, where FCnt_i denotes the value of the FCnt field of packet i. This condition ensures that packet $i+1$ immediately follows packet i and that they are sent by the same device. The inter-arrival sequence is therefore designed to exclude inter-arrival times related to packets that are not sent sequentially, which could occur in cases involving lost packets. Furthermore, it's essential to emphasize that the attacker's capabilities are limited to passive monitoring of the LoRaWAN traffic. The attacker does not have access to encrypted information at the NS level, meaning that they are only able to eavesdrop on the transmitted data without decrypting it.

The results of the traffic analysis brought up in Chapter 4, specifically in Section 4.3.7, show that the majority of the devices follow some sort of temporal patterns in their packet transmissions. A typical example of such a temporal pattern is sending one packet every hour during the day and no packets during the night. For instance, in the energy metering dataset, a majority of EDs send 24 packets, one packet every hour, followed by a 25th packet after approximately 30 seconds. This pattern then repeats. Other EDs follow different patterns, such as sending one packet roughly every 24 hours. It's worth noting that there are also EDs external to the application and LoRaWAN operator that do not conform to these specific temporal behaviors. These temporal patterns must be discovered for each individual device since they are not known beforehand.

In the context of this work, we treat the application as a black box, providing no application-specific information to the algorithm. An attacker attempting de-anonymization in this scenario must contend with cumulative traffic, which includes non-targeted EDs. These non-targeted EDs transmit packets that are also intercepted by the attacker but do not follow any specific temporal behavior. These non-targeted EDs contribute to the noise in the data, and the attacker must have the capability to handle this noise, as they have no prior knowledge of whether a packet belongs to a targeted ED.

As part of our analysis, we focus on the time elapsed between a packet and the next one from the same ED, whether in the uplink or downlink. Let t_i be the

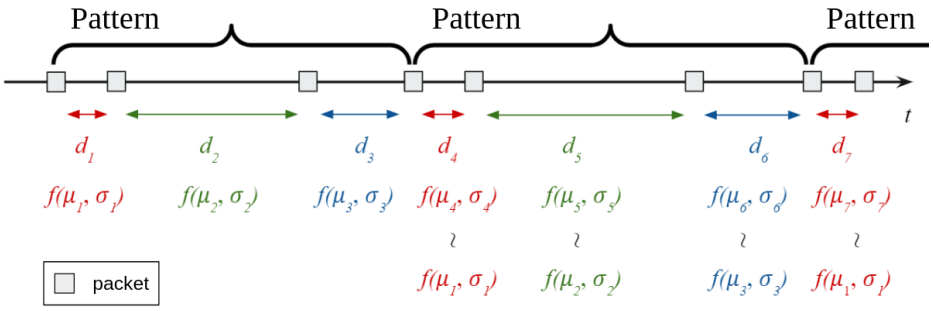


Figure 6.7. Example of modeling of the periodic sequence for the packet inter-arrival times. The line is the temporal axis, and the gray squares are packets belonging to a single ED. The time d_i that occurs between two consecutive packets p_i, p_{i+1} is distributed following a probability distribution $f(\mu_i, \sigma_i)$. The chain has period $\tau = 3$, with $\mu_i = \mu_{i+\tau}$ and $\sigma_i = \sigma_{i+\tau}$. The inter-arrival times highlighted in red d_1, d_4, d_7 follow the same distribution $f(\mu_1, \sigma_1)$, the ones in green d_2, d_5 follow another distribution $f(\mu_2, \sigma_2)$ and the ones in blue d_3, d_6 follow $f(\mu_3, \sigma_3)$.

timestamp of a packet p with frame counter (FCnt) $p_{\text{FCnt}} = i$, relative to a given DevAddr. Both timestamp and FCnt are packet fields used in our approach and outlined in Table 6.1. The timestamp represents the time added by the GWs at the exact moment when the packet was received.

Since in LoRaWAN v1.1, the FCnt is a counter of sent frames, the time $d_i = t_{i+1} - t_i$ signifies the time between two consecutive packets, commonly referred to as the inter-arrival time. Furthermore, the FCnt field is sent as unencrypted data in both the uplink and downlink, making it observable by third parties. Our assumption is that the values d_i exhibit some periodic behavior. Specifically, we model each d_i as a random variable with distribution $d_i \sim f(\mu_i, \sigma_i)$. Let the integer value τ represent the period when the inter-arrival time has the same distribution, meaning that $d_i, d_{\tau+i}, d_{2\tau+i}, \cdots \sim f(\mu_i, \sigma_i)$—in other words, the inter-arrival time distribution repeats after every τ inter-arrivals. Figure 6.7 provides an illustrative example of this model, highlighting the periodic d_i values with a period of $\tau = 3$.

Given that we do not possess prior knowledge of the value of τ, let $\hat{\tau}$ represent the predicted or estimated value of the true τ. We will now describe a subroutine of DEVIL designed to extract temporal patterns from the inter-arrival sequence while estimating the sequence's period, denoted as $\hat{\tau}$.

Firstly, this procedure takes into account the potential presence of lost packets, which are packets transmitted by a device but not received by the NS. To identify these lost packets, we observe which FCnt values are absent in the data. Let's define $L(a)$ as the set of FCnt values that are not observed for a specific DevAddr a due to packet losses:

$$L(a) = \{i < z < j \text{ s.t. } \exists\, p_{\text{FCnt}} = i, \exists\, p_{\text{FCnt}} = j, \nexists\, p_{\text{FCnt}} = z\} \tag{6.2}$$

The first step of the subroutine yielding the period of the temporal sequence is to calculate inter-arrival times from packets:

$$d_i = \begin{cases} t_{i+1} - t_i & \text{if } i, i+1 \notin L(a) \\ 0 & \text{otherwise} \end{cases} \tag{6.3}$$

The value d_i represents the time between the reception of two packets, namely p_1 and p_2, with FCnt values of $p_{1,\text{FCnt}} = i$ and $p_{2,\text{FCnt}} = i + 1$, provided that both p_1 and p_2 have been received. However, if either of these packets, p_1 or p_2, is not observed, the inter-arrival time d_i is set to zero. It's important to note that a value of zero serves as a placeholder for subsequent steps rather than indicating a real inter-arrival time.

In this subroutine, a fixed period $\hat{\tau}$ is assumed, and the subroutine generates an estimation metric. This metric is low if $\hat{\tau}$ is likely to be close to the true value of τ and high otherwise. The estimation process involves extracting sequences as follows:

$$s_j = [d_j\,, d_{\hat{\tau}+j}\,, d_{2\hat{\tau}+j}\,, d_{3\hat{\tau}+j}, ...],\ \forall j\{\in 1, \ldots, \hat{\tau} - 1\} \tag{6.4}$$

Then it computes the standard deviation σ_{s_j} for each of the sequences s_j. In the computation, the components $d_i = 0$ are not considered and are discarded: in this way we are ignoring the inter-arrival times of non-consecutive packets in case of a packet loss. The values of the standard deviations are then summed up:

$$\xi_{\hat{\tau}} = \sum_{j\in\{1,\ldots,\hat{\tau}-1\}} \sigma_{s_j} \tag{6.5}$$

The quantity $\xi_{\hat{\tau}}$ represents an estimation error for the prediction $\hat{\tau}$. A lower value of $\xi_{\hat{\tau}}$ suggests a higher likelihood that $\hat{\tau}$ is equal to the true value τ. In essence, the estimation error serves as a measure of how closely the predicted period aligns with the actual period. While more accurate aggregation and error functions could be employed, such as ones that consider the standard deviation of a time d_i, for our purposes the sum of the standard deviations, as expressed in Eq. 6.4, has proven to be effective.

In Figure 6.8, the orange line illustrates an example of the $\xi_{\hat{\tau}}$ values for different values of $\hat{\tau}$. This visual representation helps to demonstrate how the estimation error varies with different period predictions, providing insights into the likelihood of $\hat{\tau}$ being equal to the true τ.

The algorithm, which will be detailed in the upcoming section, will base its decision on the estimation error given by Eq. (6.5). This error, being an aggregation of standard deviations, relies on the presence of some form of periodicity in the inter-arrival times of the packets. As hinted in Fig. 6.8, non-periodic arrivals will result in a high estimation error, while in periodic scenarios, this error will tend to be close to zero. In essence, DEVIL leverages the inherent periodicity in the inter-arrival sequences of the devices for its matching operation. One potential countermeasure against this attack could involve introducing a random component, such as random jitter, to the scheduling of uplink packets, as described in Section 6.6. This randomization would disrupt the predictability of packet arrival times and make it more challenging for attackers to exploit the periodicity in the inter-arrival times.

6.4 DEVIL Procedure

In this section, we will describe the DEVIL algorithm, which is designed to derive the associated LoRaWAN DevEUI given a DevAddr. The procedure is divided into two distinct steps:

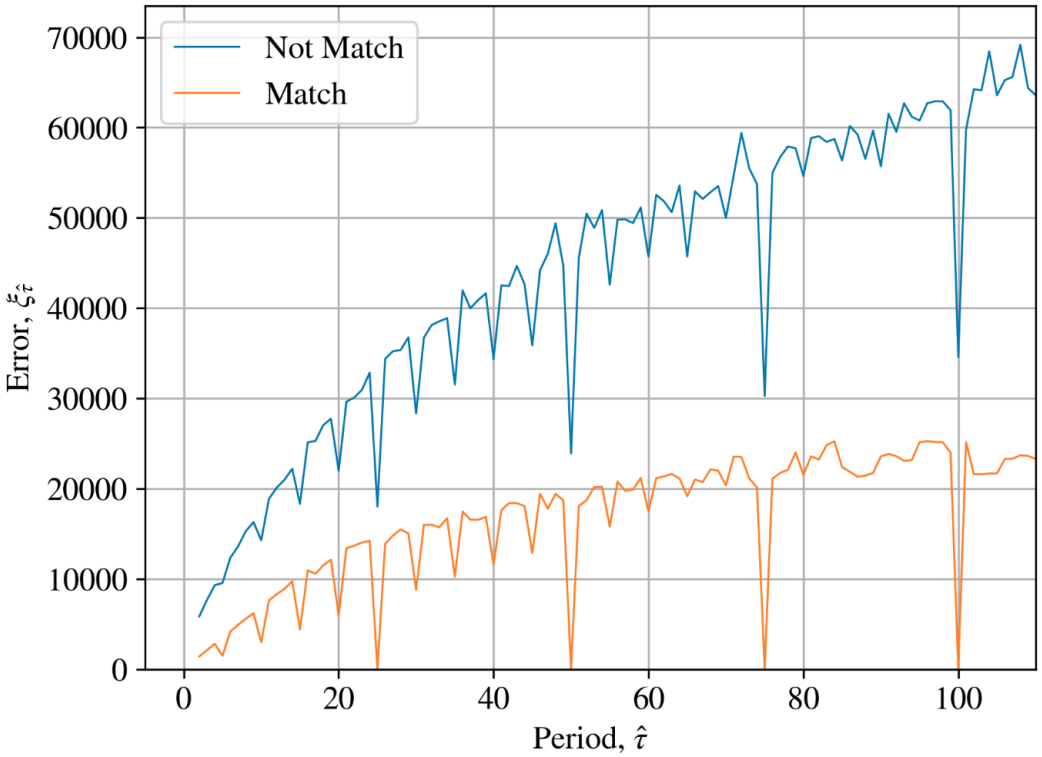


Figure 6.8. Estimation error $\xi_{\hat{\tau}}$ obtained on a sequence of packet inter-arrival times with different values of period $\hat{\tau}$. We observe the error spiking to almost zero when the predicted period $\hat{\tau}$ is equal to the true period τ. Moreover, we have multiple spikes in correspondence to the integer multiples of the period, since if τ is the period of the sequence also $k\tau$ is a valid period of the sequence, for any integer k. When associating a_1, a_2 DevAddr, their packets are concatenated and the sequence of inter-arrival times is extracted. The plot shows the error $\xi_{\hat{\tau}}$ as a function of $\hat{\tau}$ in case a_1, a_2 are a correct matching, i.e. they belong to the same device, and in case they are an incorrect matching, represented by the orange and blue lines, respectively.

- **Step 1.** Produces a mapping denoted as M, where $M(a_1) \to a_2$ if a device that was originally assigned the address a_1 is subsequently assigned the address a_2;
- **Step 2.** Utilizes the information from Step 1 to infer the final set of addresses assigned to a specific DevEUI e, denoted as $A_p(e)$. Here, $A_p(e)$ represents the algorithm's output, while $A_t(e)$ represents the ground truth or true set of addresses assigned to DevEUI e.

It's important to note that the two tasks performed in these two steps are distinct: in the first step, the goal is to identify that two addresses belong to the same device. In the second step, the aim is to precisely determine the DevEUI of that device.

To introduce some notation:

- Let a represent a LoRaWAN DevAddr;
- $S(a)$ denotes the timestamp of the first packet received from DevAddr a;

- $E(a)$ represents the timestamp of the last packet received from DevAddr a.

Two DevAddrs, a_1 and a_2, are considered *consecutive* if the following condition holds:

$$S(a_2) > E(a_1) \ \wedge \ S(a_2) < [E(a_1) + T_w] \tag{6.6}$$

where the time window T_w a parameter of DEVIL. With the previous formula we imply that, for a_1, a_2 to be consecutive it is necessary that the first packet of a_2 should occur inside a window which starts at the time of the last packet of a_1 and has duration of T_w seconds. We denote by $C(a)$ the set of consecutive addresses of a. In other words, $C(a)$ is the set of Device Addresses which have sent their first packet within a time window T_w after the last packet of a is observed.

n the first step of the DEVIL algorithm, the procedure iterates over all DevAddrs a_1. For each a_1, it iterates over all its consecutive addresses $a_2 \in C(a_1)$. The algorithm concatenates the packets from a_1 and a_2, treating them as if they were generated by a single device. The inter-arrival times d_i of this packet sequence are extracted using Eq. 6.3. The subroutine described in Section 6.3 is called, returning the values $[\sigma_{\hat{\tau}_1}, \sigma_{\hat{\tau}_2}, \dots]$ for all periods $\hat{\tau}_i$ of interest. Let n be an integer, and let $\langle \hat{\xi} n \rangle$ represent the n smallest values obtained from $\xi \hat{\tau}_i$. The address a_2^* that generated the lowest values in $\langle \hat{\xi}_n \rangle$ is identified. The final mapping is then updated by setting $M(a_1) \rightarrow a_2^*$. Figure 6.8 illustrates the values of $\xi_{\hat{\tau}}$ in cases where $\langle a_1, a_2 \rangle$ represents either a correct or incorrect association. It's worth noting that, due to the periodic traffic patterns of the devices, a DevAddr a_1 is always associated with a DevAddr $a_2 \in C(a_1)$. If this is not the case, a threshold on $\langle \hat{\xi}_n \rangle$ could be applied. In such a scenario, the algorithm would recognize that a_1 has ceased transmitting, indicating that there is no $a_2 \in C(a_1)$ associated with the same device. Additionally, other radio-space features, such as the RSSI experienced at the GWs, could be leveraged to further refine predictions. Physical layer information is always available at the GWs level and can provide valuable insights for refining associations.

The output of the first step of DEVIL is the mapping $M(a_1) \rightarrow a_2$, where a_2 is the DevAddr that a device gets assigned after a_1.

The pseudocode of the first step is reported in Alg. 1.

For the second step of the algorithm, we denote as $J(a_1, a_2) = \{e_1, e_2, \dots\}$ the set of DevEUI values that performed a Join or ReJoin request in a time window spanning from $E(a_1)$ to $S(a_2)$ i.e. from the time of the last received packet of a_1 to the first received packet of a_2. The second step of DEVIL iterates on all DevAddr a_i. It gets a_{i+1} following the update rule:

$$a_{i+1} \leftarrow M(a_i) \tag{6.7}$$

and constructs the set $A_{p,i+1} = A_{p,i} \cap J(a_i, a_{i+1})$. The algorithm iterates following the update rule in Eq. 6.7. If for some value of $i = i'$ we have that $a_{i'} \notin M$ or $|A_{p,i'+1}| = 0$, the algorithm stops updating and for all DevEUI $e \in A_{p,i'}$ it updates the final solution:

$$A_p(e) = A_p(e) \cup \{a_i, a_{i+1}, \dots, a_{i'}\}, \ \forall e \in A_{p,i'} \tag{6.8}$$

In other words, the second step of DEVIL checks for the DevEUI of the EDs performing Join requests in a time frame $[E(a_1), S(a_2)]$, for any DevAddr addresses

Algorithm 1 First step of DEVIL algorithm. It produces the mapping $M(a_1) \to a_2$, where a_1 is a DevAddr assigned to a device and a_2 is the next DevAddr used by the same device after a Join or ReJoin procedure.

for $a_1 \in$ all DevAddr **do**
  $a^* \leftarrow$ NaN
  $\langle \hat{\xi}_{n,\min} \rangle \leftarrow \langle \infty \rangle$
  for $a_2 \in C(a_1)$ **do**
    concatenate packets having DevAddr a_1, a_2
    extract d_i sequence
    compute $\langle \hat{\xi}_n \rangle$
    if $|\{i \text{ s.t. } \langle \hat{\xi}_n \rangle_i < \langle \hat{\xi}_{n,\min} \rangle_i\}|$ **then**
      $\hat{\xi}_{n,\min} \leftarrow \hat{\xi}_n$
      $a^* \leftarrow a_2$
    end if
  end for
  $M(a_1) \leftarrow a^*$
end for
return M

a_1, a_2 such that $M(a_1) \to a_2$. The pseudocode for the second step of the algorithm is reported in Alg. 2.

Algorithm 2 Second step of DEVIL. The output is A_p where $A_p(e) = \{a_1, a_2, \dots\}$ is the predicted set of DevAddr values which were assigned to the device having DevEUI e.

for $a \in$ all DevAddr **do**
  $\bar{E} \leftarrow$ set of all DevEUI
  $\bar{A} \leftarrow \{a\}$
  while $a \in M$ and $|\bar{E} \cap J(a, M(a))| > 0$ **do**
    $\bar{E} \leftarrow \bar{E} \cap J(a,\, M(a))$
    $a \leftarrow M(a)$
    $\bar{A} \leftarrow \bar{A} \cup a$
  end while
  for $e \in \bar{E}$ **do**
    $A_p(e) \leftarrow A_p(e) \cup \bar{A}$
  end for
end for
return A_p

6.4.1 Computational Complexity

We now analyze the computational complexity of both the first and second steps of the DEVIL algorithm. In the first step, the outer loop iterates over any observed Device Address a_1. The inner loop iterates over the set of all consecutive Device Addresses $a_2 \in C(a_1)$. The number of iterations in the inner loop depends on the

size of the consecutive device addresses, which is influenced by the time window T_w and the available data. In general, the number of iterations of the inner for-loop is equal to the average size of the consecutive addresses $|C|$, defined as follows:

$$|C| = \frac{1}{|A|} \sum_{a} |C(a)| \tag{6.9}$$

where A is the set of all Device Addresses in the considered dataset. On our dataset, with $N = 150$ devices and a time window $T_w = 6$ hours, the average value of the consecutive addresses was $|C| \approx 90$. Then, for every pair $\langle a_1, a_2 \rangle$ the algorithm computes cost values for each possible length of the inter-arrival sequence, from 1 to S, where S is the maximum length of a periodic pattern in the inter-arrival sequence. Therefore, the final computational cost of the first step is $\mathrm{O}(|A| \cdot |C| \cdot S)$.

In the second step, the algorithm goes through all device addresses a and examines the Join Request messages observed within a time frame from the last packet of the DevAddr a to the first packet of the DevAddr $M(a)$. Iteratively, the algorithm follows the change of DevAddr by updating $a \leftarrow M(a)$ until the M chain is finished or there are no Join Request messages between a and $M(a)$. This procedure is executed for each address a, making the final computational cost of the second step $\mathrm{O}(|A|^2)$. The reason for the squared term is that an address is analyzed multiple times: once in the outer loop as a and possibly several times while following the change of DevAddr M. We also tried a variation of the algorithm in which we analyzed a DevAddr only once, bringing the computational cost down to $\mathrm{O}(|A|)$. However, we observed a reduction in terms of accuracy, since errors made in the first step for the computation of the mapping M had more influence on the final result.

6.5 Numerical results

In this section, we report the results of applying DEVIL to data generated by a real LoRaWAN application. Table 6.2 provides the numerical values of the parameters used in our experiments. The time T_w represents the length of the time window in which two DevAddrs can possibly be consecutive. The integer n is the threshold used in the first step of the algorithm. The ratio β_{lost} is the fraction of packets that were lost during the communication session. Note that β_{lost} is not negligible, as it accounts for the impact of missed packets on the temporal analysis of the traffic. Neglecting missed packets could introduce phase-shifts in the analysis, potentially leading to inaccurate results. If this is not done, phase-shifts would have certainly been verified in the temporal analysis of the traffic, which would have invalidated the results. Moreover, the application does not always respect the periodicity assumption and causes noise to the system.

Table 6.2. Parameters of the algorithm and characterization of the data.

Variable	**Value**
T_w	3700 s
n	5
β_{lost}	5%

We evaluated the accuracy of the first and second steps of DEVIL separately. While the final accuracy of DEVIL is primarily determined in the second step, where it infers the DevAddr assigned to a DevEUI, the first step of DEVIL serves as a valuable tool to determine whether two DevAddrs were assigned to the same ED. In general, it's reasonable to expect that the accuracy of determining whether two consecutive DevAddrs were assigned to the same ED is higher than accurately inferring the exact DevEUI of the ED.

The evaluation of the accuracy of DEVIL's first step, which involves constructing the mapping M, was done by computing all pairs $\langle a_1, a_2 \rangle$ such that $M(a_1) \to a_2$. The accuracy was measured by calculating the ratio of correct predictions to the total number of predictions.

In the evaluation of the accuracy of the second step, it was taken into account that a DevAddr a could be associated with one or more DevEUIs, meaning that a could belong to both $A_p(e_1)$ and $A_p(e_2)$ simultaneously. This situation occurs when the algorithm is uncertain about which DevEUI to associate with the address a. To measure accuracy in this context, an indicator variable i_a was introduced, with a value of 1 indicating a correct prediction and 0 indicating an incorrect one. Additionally, the number of times a appeared in the solution was counted as $n_a = |e \text{ s.t. } a \in A_p(e)|$. The accuracy was then expressed as the quantity $\frac{i_a}{n_a}$ and averaged across all DevAddrs a.

In Figure 6.9 the accuracy of the first and second step of DEVIL is reported as a function of the T_w parameter. From this analysis we extract the best value of the T_w parameter ($3700s$), reported in Table 6.2.

Table 6.3 provides an overview of the accuracy achieved by the two steps of the DEVIL algorithm, evaluated separately. The first step demonstrates an accuracy rate of over 95% in constructing the mapping M. This result underscores the algorithm's capability to process data from a real LoRaWAN application and reliably identify when a change in DevAddr occurs for an ED. The second step of the algorithm achieves an accuracy rate of 93% in successfully identifying all the LoRaWAN addresses assigned to a specific DevEUI. This step effectively reconstructs the hidden mapping between a DevAddr and its corresponding DevEUI.

Table 6.3. Accuracy of the two separate steps of DEVIL algorithm using the water metering dataset.

Algorithm	**Description**	**Accuracy**
First step	Construct mapping $M(a_1) \to a_2$ from a DevAddr a_1 to the next DevAddr a_2 utilized by the same device	0.959
Second step	Produce $A_p(e) \to \{a_1, a_2, \dots\}$ reconstructing the set of addresses which were assigned to DevEUI e	0.936
Final Accuracy		**0.936**

In Table 6.4, we have presented the accuracy results of DEVIL in comparison to the algorithm introduced in [4]. This comparison is conducted during the second step of DEVIL, as it is this specific stage that yields the final de-anonymization

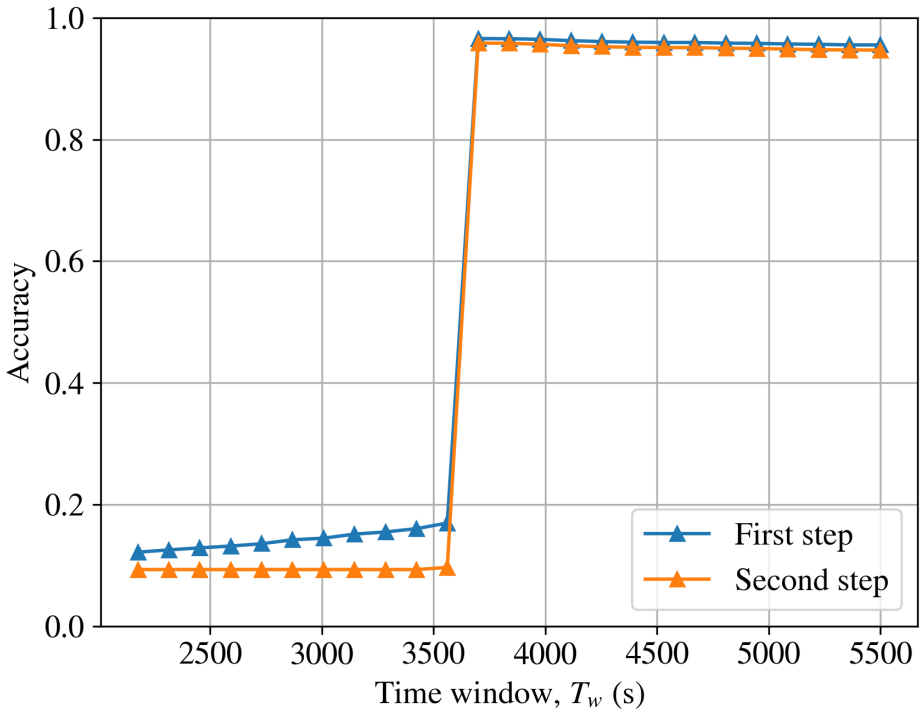


Figure 6.9. Accuracy of the first and second step when varying the parameter T_w. For low values of T_w, the algorithm does not find the correct matching of a DevAddr and the accuracy of the whole algorithm is low. This happens since, given a DevAddr a, the short window causes the set of consecutive addresses $C(a)$ to not contain the good match of a. For $T_w \geq$ 3600s we observe high values of accuracy. This is due to the majority of devices sending an uplink packet roughly every hour. Moreover, when T_w is greater than 3600s we observe a slight decrease in the accuracy. This happens since the unnecessarily wide window causes the set of consecutive addresses $C(a)$ to include an increasing number of addresses that are not a match for a anymore, thus increasing the possibility of a wrong match.

outcome. In all test scenarios, DEVIL consistently achieves higher accuracy, with an improvement of 58% in the case of water and energy metering devices and a 26% in the synthetic dataset. It is worth noting that we did not evaluate the algorithms on the LoED dataset since it lacks the necessary information on the DevEUI of the packets, which serves as the ground truth for accuracy computation.

Furthermore, we examined the execution times of both DEVIL and the algorithm proposed in [4]. The total execution times are detailed in Table 6.4, with each entry corresponding to a specific dataset and algorithm pairing. These algorithms were executed on a laptop equipped with an AMD Ryzen 7 4800U CPU, utilizing 14 parallel execution threads. The reported time reflects the effective CPU time. From the results, it is evident that DEVIL demands more execution time compared to the comparative algorithm. This disparity can be attributed to DEVIL's consideration of the inter-arrival times, resulting in a higher computational complexity. The computational complexity of the comparative algorithm can be approximated as

Table 6.4. Accuracy of DEVIL in the water metering, energy metering and synthetic datasets. We reported the accuracy of both the first and the second step of DEVIL. The accuracy of the first step is always higher than the accuracy of the second step, due to the fact that the second step uses as input the output of the first step. In this way the accuracy of the first step serves as upper bound of the accuracy of DEVIL. In the rightmost column of the table we reported the accuracy of the de-anonymization algorithm presented in [4]. Also, we have reported the CPU time of the procedures.

Dataset	# packets	Time DEVIL	Time [4]	Step	acc. DEVIL	acc. [4]
Water metering	0.9×10^6	43.97s	16.01s	First	0.959	0.592
				Second	0.936	
Energy metering	1.1×10^6	266.62s	5.66s	First	0.946	0.576
				Second	0.913	
Synthetic (N=1500)	1.1×10^6	3137.75s	64.34s	First	0.965	0.596
				Second	0.824	

$O(|A| \cdot |C|)$, which is lower than that of DEVIL, as shown in Section 6.4.1.

It's worth noting that when compared to the algorithm presented in [4], which achieves a DevEUI matching rate of 94%, DEVIL achieves a similar level of performance. However, DEVIL distinguishes itself by not relying on the presence of Join requests sent by the EDs, making it more adaptable for flexible network monitoring. In other words, while the approach in [4] fails if some Join request messages are not observed, DEVIL continues to work successfully and derives the correct matching. The first step of DEVIL can operate with different, non-overlapping time windows, which do not necessarily need to include Join request packets. Ultimately, in the second step, it is sufficient to capture a Join request once within any of the time windows to correctly map the DevEUI to the ED.

The choice of the parameter T_w should be aligned with the characteristics of the data on which the algorithm is applied. A shorter duration for the T_w window tends to result in lower accuracy due to the reduced number of devices considered within that time frame. With a small T_w, there is a possibility that the correct device might be excluded from consideration by the algorithm, as its first packet after changing the DevAddr may occur more than T_w seconds after its last packet with the old DevAddr.

In theory, selecting T_w to be as large as the maximum estimated inter-arrival time for an application should suffice to ensure the correct DevAddr is considered by the algorithm. However, in practice, packet losses can occur, causing the observed inter-arrival time of a device to be greater than the estimated maximum. Therefore, the value of T_w should be increased accordingly. On the flip side, a smaller T_w will result in faster algorithm execution. This is because T_w directly affects the number of devices considered when selecting the next DevAddr in the first step of the algorithm. With fewer candidates to check, the algorithm can run in less time.

6.6 Countermeasures

As hinted to in Section 6.3, DEVIL relies on the presence of some periodic behavior in the sequence of inter-arrival times. Therefore, one potential countermeasure

could involve introducing a random component to disrupt this periodicity. In our experiments, we introduced a jitter, which is uniformly distributed in the range $[0, \alpha\hat{t}]$ as outlined in Eq. (6.1). This randomization introduces some level of unpredictability into the scheduling of uplink packets.

However, it's important to note that the added jitter can potentially disrupt the operation of the higher-level application. For example, if our application consists of smart sensors logging temperature data every hour, the introduction of jitter may require the ED to buffer and/or delay packets, even though this impact should be minimal for most packets. We express the random jitter in terms of α rather than absolute values to emphasize that the amount of jitter introduced should be proportionate to the current value of the non-jittered inter-arrival time. This provides a measure of how much the jitter affects the behavior of the ED and, by extension, the application.

In Figure 6.10, we have presented the accuracy of DEVIL using various values of α and different numbers of devices. We utilized the synthetic dataset to stress-test the proposed approach and assess its robustness concerning the jitter parameter. As anticipated, the accuracy of DEVIL decreases as we increase the value of α since higher values of this parameter result in greater disruption of DEVIL's time analysis. We have also plotted the accuracy of the algorithm in [4] on the same dataset. Since this algorithm does not rely on the temporal analysis of inter-arrival times, its accuracy remains constant across different values of α.

In Figure 6.11, we have reported the accuracy of DEVIL under various values of the maximum periodic pattern length S in the synthetic dataset. Notably, we observe that the performance of DEVIL remains consistent when altering the length of the patterns in the known inter-arrival times. Furthermore, we find that the accuracy of DEVIL remains constant even when increasing the number of devices N, consistently achieving an accuracy of over 90% across all values of N (i.e., $N = 150, N = 600, N = 1500$).

To summarize, the introduction of a specific jitter value into the system can be considered as a countermeasure technique against device de-anonymization performed by DEVIL. Suitable jitter values can be assessed by DEVIL and subsequently recommended to the NS to enhance the privacy robustness of certain devices.

6.7 Findings

We have presented the DEVIL algorithm designed to reconstruct the mapping between temporally changing LoRaWAN DevAddr addresses and the unique DevEUI identifier. Our approach relies on temporal traffic analysis, which involves gathering information about the timing patterns associated with the generation of uplink traffic by IoT devices.

While DEVIL is in essence an attack that could be performed by a malicious third party to discover the hidden identity of a device, a variation of the algorithm could be used for security purposes by network administrators. Indeed, one could imagine a that alerts the network operator whenever it detects that such an attack is feasible, working as a sort of intrusion detection system. Since the algorithm works only with GWs level information, it is a great candidate to be executed at the

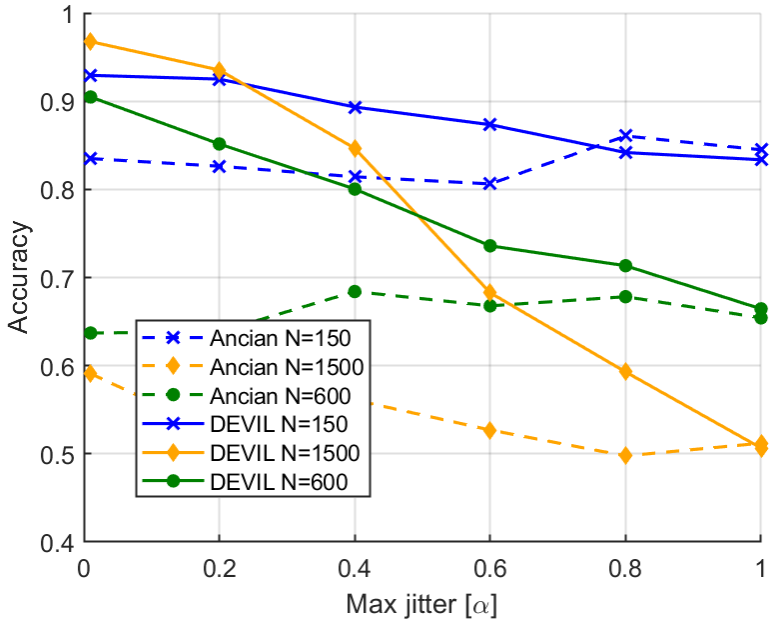


Figure 6.10. Accuracy of DEVIL and [4] with different values of the jitter α in the synthetic dataset and with different number of devices related to the evaluation of the countermeasure.

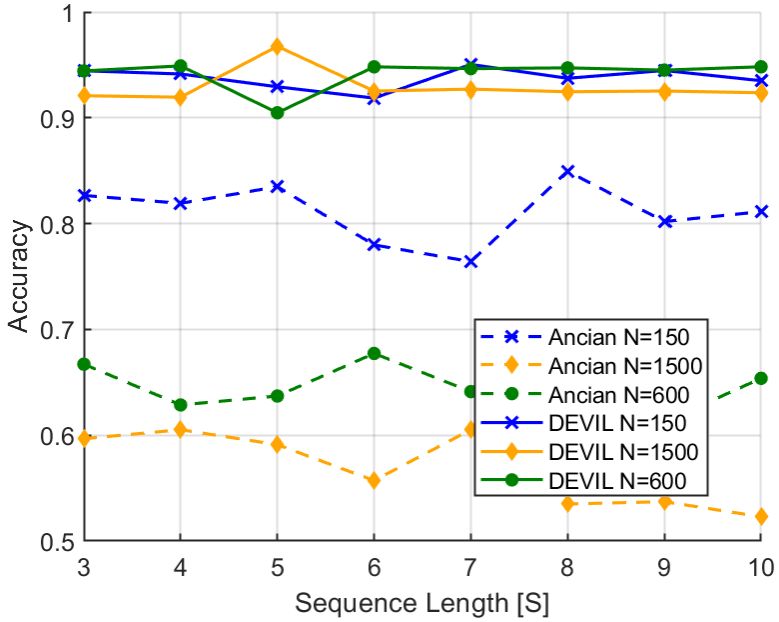


Figure 6.11. Accuracy of DEVIL with different values of pattern length S in the synthetic dataset and with different number of devices.

edge of the network, represented by the GWs in the LoRaWAN architecture. The relationship between LoRaWAN and edge computing and the variation of DEVIL as a security measure will be illustrated in the next Chapter.

Chapter 7

Mapping operations on Edge computing

In this chapter we will show how the security and resource allocation in a LoRaWAN network can be optimized through Edge computing via the use of the sEcure and seamLess EdGe-to-cloud ANalyTics (ELEGANT) framework. ELEGANT is a project founded by the European Union which has the goal to unify the software paradigm for Big Data and IoT, enabling seamless interoperability. To this mean, ELEGANT introduces a new unified API based on map-reduce operations [23]. The organization of the Chapter is as follows. We will introduce the ELEGANT framework and its principal components. Then we will propose our DEVIL de-anonymization algorithm, introduced in Chapter 6, which will be distributed leveraging ELEGANT and map-reduce. Finally we will propose a method to dynamically allocate SF in LoRaWAN, which will be engineered ex-novo fully working on ELEGANT. This will lay the foundations for the future work both on ELEGANT itself and on the distributed management of a LoRaWAN network.

7.1 ELEGANT

Nowadays, IoT-based applications process large amount of data arriving as streams from IoT devices. Such compute infrastructures are paving the way towards the future Sharing Economy Systems and they typically follow a producer/consumer model in which the analytics platform processes the IoT generated data. Since the processing capabilities of IoT devices are continually increasing, edge-processing has started being utilized as a means to:

- Increase both responsiveness and real-time characteristics of every deployed application;
- Achieve energy efficient processing by reducing the computational cycles of the heavyweight cluster or cloud-deployed Big Data frameworks;
- Increase the security of the deployment in an end-to-end manner by enabling in-situ computations thus minimizing data exchanges through the network.

Ideally, users should consider and utilize such software and hardware diverse deployments as a single system with heterogeneous hardware capabilities that they can operate on demand depending on their specific use case requirements. Hence, development on such unified systems should not differ from standard programming practises and the internal decisions on where specific computational tasks should be performed must be completely transparent to developers, as well as enhance attributes such as data integrity along with the overall platform resilience. Moreover, the code execution and data generation and transferring should be conducted in accordance with contemporary security standards which ensure protection against malicious activities and cybersecurity threats.

ELEGANT's vision is to create a new software paradigm for Big Data and IoT by unifying their programming environments and enabling the automatic and easy deployment of existing code from Big Data platforms to IoT devices and backwards in a self-adaptable way. Specifically, ELEGANT will introduce a new unified Java API that will allow the seamless execution of any source code already available on the data analytics (Big Data) side on the IoT devices via a lightweight and thin application virtualization layer. This way, the Big Data framework will deploy on demand and dynamically any existing operators at the edge, thus transforming all the interconnected IoT devices to execution units. Further, by combining and extending state-of-the art Big Data/IoT capable runtimes, such as MaxineVM [95] and TornadoVM [35], ELEGANT will produce a single elastic runtime capable of scaling from IoT devices to Big Data cloud deployments.

To enable parallelization and scalability, the unified ELEGANT API will be based on map-reduce [23]. Table 7.1 shows the complete list of the operators made available by the ELEGANT framework, accompanied by a complete description for each operator.

In the context of ELEGANT and edge computing, surely LoRaWAN promises interesting research landscapes. In a classical LoRaWAN architecture, as described in Chapter 3, the data and control layer are managed end-to-end by the devices and the Network Server. The gateways in this scenario don't perform any network management operation, acting as simple packet forwarders between the devices and the core Network Server. However, the gateways are, after all, hardware already deployed on the edge of the network, thus interesting research questions can be posed when one considers what they can be used for. Indeed, one of the main problematics posed by edge computing paradigm, is the deployment of new hardware in strategic positions of the network, which can be costly and sometimes unfeasible. But since LoRaWAN already foresees the presence of such hardware, it is worth investigating what applications and optimization can be achieved exploiting edge computing in a LoRaWAN environment.

In the next sections, we describe two optimization that can be performed on a LoRaWAN network exploiting the edge computing paradigm and the ELEGANT framework. The first optimization enhances the security of the network, bringing the DEVIL algorithm, introduced in Chapter 6, to the network edge, in order to alert the network operator if a de-anonymization attack could be possible, acting as a sort of IDS. The second optimization deals with the allocation of Spreading Factor to the devices, by means of a distributed algorithm that runs on the Gateways closer

Operator	Description
`union`	The union operator merges two streams into a single stream. Both streams need to have the same schema. It is defined as the set of all rows belonging to the first stream or the second stream or both with any resulting duplicate rows deleted.
`join`	The join operator joins tuples coming from two event streams. To define a join operator, we need two queries; thus, the operator is used to join the result of the two queries on a specified key and a common window.
`projection`	The operator is used to choose one set of columns from the stream and potentially even rename them. For some subset S of the attributes of the stream, produce from each tuple only a subset of the components.
`window.bykey`	The window operator selects a burst of data in a time window. The framework supports two types of windows, tumbling windows and sliding windows. For each type, we can set the time measures and aggregation function to use. The operator transforms a Query into a WindowedQuery. Tumbling Window and Sliding Window are supported options.
`aggregation`	The aggregation operator works in conjunction with the window operator. The framework supports the following aggregation functions: `sum`, `count`, `max`, `min`, `average`, `median`;
`filter`	The operator transforms a stream by filtering out tuples that don't meet the filter criteria. The framework supports the following predicate expressions: equal, `greaterThan`, `greaterThanOrEqual`, `lessThan`, `lessThanOrEqual`;
`map`	The operator transforms a stream by applying a transformation on tuples. The map operator accepts two parameters, i.e., a field name and a map expression;
`source`	The source management API contains a collection of methods in the Java client to retrieve, add, update and remove logical sources in an instance. The operator expects the schema and a source name as parameters. Thus, the operator creates a new query, from the specified stream name;
`sink`	The sink operator defines where the output of the query will be placed, currently on .csv files.

Table 7.1. List of the map-reduce operators exposed by ELEGANT.

to the devices.

7.2 Edge Computing in LoRaWAN

Edge computing in LoRaWAN is still a relatively unexplored field. This is due particularly to the client-server paradigm adopted by LoRaWAN. Specifically, the protocol is executed end-to-end from the ED to the NS and vice versa. In the LoRaWAN architecture, the GWs don't perform any useful operation to network management, being it fully maintained by the NS. On the one hand, this architecture improves privacy, since the content of the packets are encrypted from a GW perspective, and reduces protocol complexity, since only one key exchange between the ED and the NS is required. On the other hand, this centralized architecture makes challenging the integration of Edge computing in a LoRaWAN environment. From a hardware perspective, the LoRaWAN architecture is a great candidate to integrate Edge computing. Indeed, one of the main complexities that hinders the development of Edge computing, is the additional cost and maintenance sustained by operators in order to deploy new hardware on the Edge. As a matter of fact, LoRaWAN already foresees hardware deployed at the Edge, which are the GWs themselves.

While proposing a new distributed architecture with heavy modifications of the communication standard is for sure the easiest way to leverage the Edge computing paradigm, we are interested in maintaining the current LoRaWAN architecture unchanged, not to hinder the compatibility with existing devices and network servers, and to find out which potentialities and opportunities can be offered by the Edge computing in a LoRaWAN environment. In this Chapter we present two integrations of Edge computing with LoRaWAN by means of ELEGANT. In the first we re-engineer the DEVIL de-anonymization algorithm as a distributed edge-enabled IDS, alerting the operator whenever it detects that a de-anonymization attack is possible. The second one is a distributed edge-enabled algorithm for SF allocation. With these integrations, we show that we can indeed perform network maintenance and optimization at Edge level, which is the GW level, without making any changes to the communication protocol standard.

7.3 Security Optimization

As shown in Chapter 3 and 6, LoRaWAN specification foresees to different addresses destined to EDs: the DevEUI, which is 64 bit long and it is unique during an ED's life and stored on nonvolatile memory, and the DevAddr, which is 32 bit long temporary address assigned by the NS. During the Join-procedure, security keys are negotiated with an ED, which receives the dynamic DevAddr. NS concludes the Join procedure by sending the Join accept message to the ED and notify the procedure to the AS. This optimization focuses on security issues of the LoRaWAN, specifically it focuses on the procedures to support device anonymization. Indeed, no identification between device packets and associated user should be made available to third parties that monitor or eavesdrop the packet flow. In LoRaWAN, security protection is

provided through symmetric encryption at network and application level. Different security issues, arose in the first standard version, have been faced in the new LoRaWAN v1.1 release specification, like authentication, integrity protection, replay protection and encryption. However, there are still some weaknesses in the device anonymization and consequently have an impact on the privacy of the devices. The proposed optimization presents a procedure to detect user identity by monitoring traffic flow in order to acquire ED information, including user activities of the entities/people that are related to the ED. One of these aspects is the identification, by simply eavesdropping on the traffic, of the DevEUI.

Generally, this procedure is performed by an IDS that monitors the network traffic to detect malicious activity or policy violations. Specifically, such IDS is edge-enabled and solves some of the challenges for Edge computing IDSs described in Chapter 2, Section 2.5, as follows:

- **Traffic Encryption.** DEVIL algorithm does not require access to non-clear-text headers contained in packets. Therefore the algorithm could run at the Edge level, not requiring packet decryption, which can be performed only by the NS.
- **High Resource Variability.** ELEGANT framework solves this challenge by automatically offloading the computation to the node that best suits the current situation.
- **Distributed IDS architecture on Edge/IoT.** The execution of the DEVIL could be easily distributed. Indeed, DEVIL considers as independent the flows of packets generated by different devices. Therefore one can easily parallelize and distribute the computation.
- **Aggregated Traffic.** In the case of Edge computing in a LoRaWAN environment, the gateways are still able to distinguish packet flows due to clear-text headers.

7.3.1 Design by ELEGANT operators

While the concept of the algorithm that we will introduce is equivalent to DEVIL, shown in Chapter 6, there will be some differences at the implementation level as well as the conceptual level. Due to the use of ELEGANT, the algorithm has to be expressed in a map-reduce fashion. This means that the algorithm must be re-engineered to be run in a distributed way.

The algorithm is based on periodic sequence matching, where, in a probabilistic way, it is possible to associate the DevAddr of the on-air packets to the relevant ED. The relevant input LoRaWAN fields used in this optimization are the same of DEVIL, reported in Table 6.1. The data fields used by the algorithm are: DevAddr, DevEUI, FCnt, TMST, and TYPE. In this scenario, an ED may sometimes send a join or rejoin message.

The goal of the attacker is to associate a LoRaWAN DevAddr to the real DevEUI of an ED, by passively monitoring the traffic. In this subsection, we show how the proposed approaches can be implemented in LoRaWAN networks

across the end-to-end data management system. The basic idea is to decompose the algorithm in multiple operators that can be executed at different layers of the presented architecture. For sake of readability, we briefly summarize the algorithm as introduced in Chapter 6. We isolate the packets sent by a ED and constructed a sequence, named inter-arrival sequence, where the element i is the time elapsed between two consecutive packets i and $i+1$. Two packets are consecutive if the $\text{FCnt}_{i+1} = 1 + \text{FCnt}_i$, where FCnt_i denotes the value of the FCnt field of packet i. This ensures that packet $i+1$ is the next packet sent after i by the same device. In case of missed packets, the frame counter fields of two packets received in a row are not consecutive and the time interval between the two packets is not stored in the inter-arrival sequence. In Chapter 4 we found that many EDs send their packets following periodic temporal patterns i.e. the inter-arrival sequence has some periodic behaviors. One example of such a temporal pattern could be sending one packet every hour during the day. The temporal analysis of packets is therefore an important feature to be considered when characterizing the traffic.

Figure 7.1 shows the detected chain operators that have been considered to implement the security optimization application, together with an example of the deployment on edge and cloud layers. In the figure, the coordinator orchestrates the processing. The figure shows both the control plane, for configuring the execution plans at the works (dashed lines), and the data plane, for the routing of the data from the sources to the cloud layer (bold lines).

Taking into account the proposed approaches, the structure of the algorithm has limitations in the decomposition because it presents some data dependencies. These limits mainly concern the phases during which address associations are performed, which must be executed sequentially. We are limited to moving at the edge the amount of process related to the packet aggregation that presents the same DevAddr. In the edge part, we first connect the data steam through the source operator, and then we select a sub-group of the packet fields through the project operator (worker 3, worker 4 and worker 5 in Figure 7.1).

At this point two streams are generated; the first stream is used to `filter` only join packets (right side). Regarding the second stream (left side), to consider a time window on the stream we apply the `windowbykey` operator on the timestamp field. Thus, packets of the same DevAddr are grouped, and we compute inter-arrival time, first packet timestamp, and last packet timestamp for each group by custom function in the `aggregation` operator.

The first stream is a stream of join message packets, the extracted fields and the relative descriptions are presented in Table 7.2. As for the extracted data of the second stream and their relative descriptions, they are presented in Table 7.3.

Table 7.2. Fields extracted in the first stream of the security optimization scheme.

Field	Description	Data type
DevEUI	Unique identifier of the physical device	String
TMST	Internal clock timestamp from GW. Timestamp of the received packet at the GW	Integer
TYPE	The type of packet specifies if the packet is application or Join request	Boolean

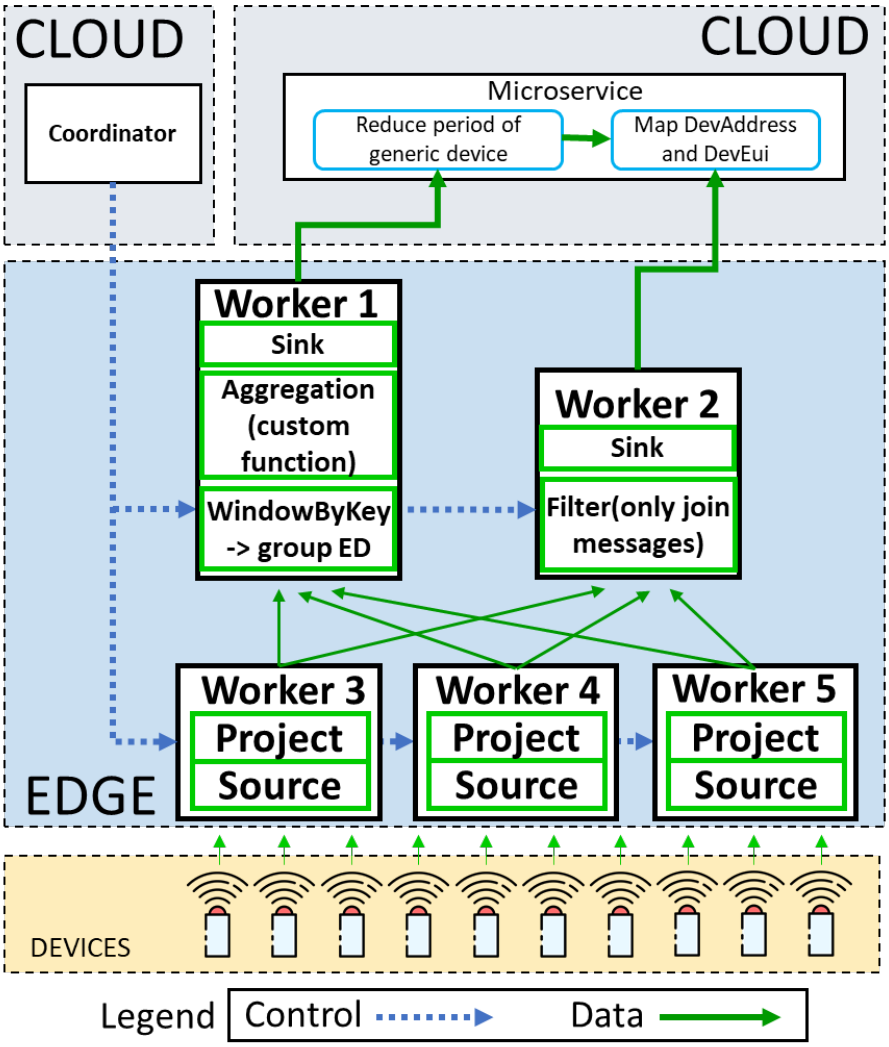


Figure 7.1. Security optimization operators edge/cloud mapping.

Both the streams are computed at edge, and they are used to feed the application microservice. As shown in Figure 7.1, it is composed from two different module, covered in figure from worker 1 and worker 2. A first reduce function extracts the period of each burst with the same DevAddr; the subroutine extracts these temporal patterns from the inter-arrival sequence, estimating the period of the sequence. It takes into account the presence of lost packets, i.e. packets that were transmitted by a device but were not received by the NS. To identify the lost packets, it observes the FCnt information. The second subroutine is a map function that finds the binding between the DevAddr with the same period (output of the first subroutine) and the DevEUI filtered by the stream presented above, the connection between the stream and the two sub routines is visible in Figure 7.1.

Table 7.3. Fields extracted in the second stream of the security optimization scheme

Field	Description	Data type
DevAddr	Unique identifier of the ED in the network.	String
inter-arrival time	List of inter-arrival time between the sequence of the ED packets in the considered `tumblingwindow` operation	List
Number of packets	The number of packets present in the `tumblingwindow`.	Integer
TMST first packet	Timestamp of the first packet in the selected `tumblingwindow`	Integer
FCnt first packet	Frame counter of the first packet in the selected `tumblingwindow`	Integer
TMST last packet	Timestamp of the last packet in the selected `tumblingwindow`	Integer
FCnt last packet	Frame counter of the last packet in the selected `tumblingwindow`	Integer

7.3.2 Query Implementation

The core of the security optimization query code, compatible with the ELEGANT java client, is reported in the following snippet of code. Obviously, the query code does not consider the part related to the microservice of the application itself because it will be customized by application development.

```
Query lora_stream_extract_period = new Query();
//select only important fields by project operator
lora_stream_extract_period.from(gatewaystream)
    .project("dev_addr","gateway","tmst","FCnt");

//window per day and aggregate according to a custom
   function, we extract
lora_stream_extract_period.window(
    TumblingWindow.of(new EventTime("tmst"),
        TimeMeasure.hours(24)
    )
).byKey("dev_addr")
    .apply(Aggregation.custom(extractInterarrival());
    .sink(new FileSink("/period.csv", "CSV_FORMAT"));

Query lora_stream_extract_join = new Query();

//select only important fields by project operator
lora_stream_extract_join.from(gatewaystream)
    .project("dev_addr", "gateway", "tmst",
        "dev_eui", "type"
    );
```

```
//filter join messages
lora_stream_extract_join.filter(
    Predicate.Attribute("type").equal("join")
).sink(new FileSink("/output.csv", "CSV_FORMAT"));
```

7.4 Resource allocation Optimization

LoRaWAN is built upon LoRa (Long Range) modulation, which employs Chirp Spread Spectrum modulation, utilizing frequency chirps with linear variation over time to encode information. Communication between EDs and GWs takes place across various frequency channels and data rates.

LoRaWAN achieves multi-data rate support by using six different Spreading Factors (SFs) (from 7 to 12), as explained in Chapter 3. The choice of SF strikes a balance between packet Time-on-Air (ToA) and delivery probability or communication range. Ideally, nodes communicate using the minimum SF that ensures correct reception by at least one nearby GW. The NS extracts and sets ED radio parameters, including SF, to optimize network performance. Various schemes can be implemented for Adaptive Data Rate (ADR) logic. The proposed algorithm, EXPLoRa-C [37], is an ADR algorithm that leverages channel capture effect and GW diversity to increase LoRaWAN cell capacity. It aims to maximize user and spatial diversity while balancing load and diversity using a "sequential water-filling" approach.

The present optimization is focus on the re-design of the data rate allocation scheme under a Big Data paradigm. Our aim is to move the processing and the optimization logic from the NS, which operates on the Cloud level, to GWs or in general to an edge level located close to GWs leveraging the ELEGANT framework. Query distribution on edge layer will bring multiple performance optimizations, including a reduction of traffic volume on the internet backhaul. For instance, data from EDs belonging to different operators is not needed to be forwarded to the NS but can be processed at the edge.

7.4.1 Design by ELEGANT operators

To implement the proposed optimization, we consider the ELEGANT framework and the available operators. The processing can be decomposed into a series of data transformation operators. To this purpose, we consider that the algorithm is executed on sensor data streams at different levels. Each ED is connected to at least one low-end node in the edge layer, which is responsible for this ED.

Figure 7.2 depicts the data splitting and parallel process allocation, starting from the knowledge of the complete data set, which is represented in the top part of the figure as a list of EDs (numbered from 1 to M) and GWs (numbered from 1 to N). Indeed, we can envision a matrix data structure, in which for each couple (ED i , GW j) we consider the average RSSI (in a given time window) value measured at the j th GW from the reception of a packet sent by the i th ED. In case the j th GW is not able to receive packets from the i th ED, the value is set to -1. The splitting

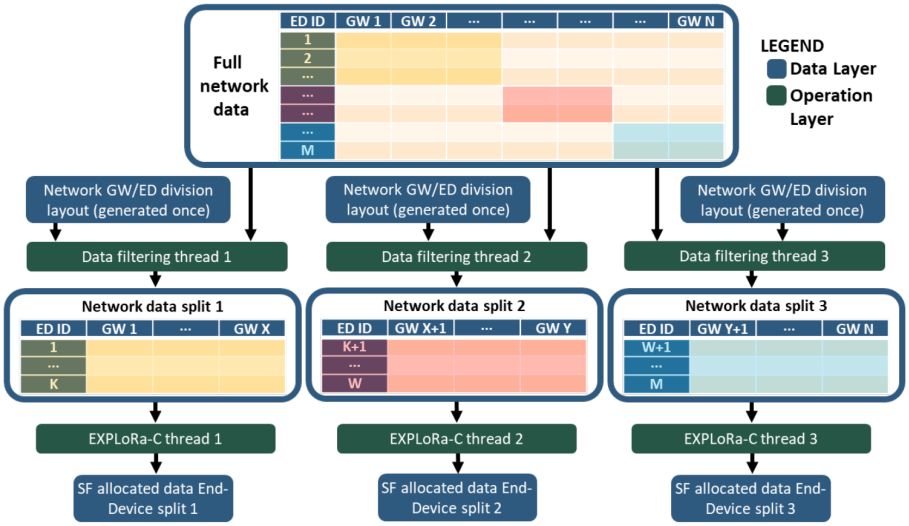


Figure 7.2. Data splitting and parallel process allocation of the EXPLoRa-C algorithm.

can be simply implemented by grouping the rows of the entire matrix M x N into G groups. In the case of Figure 12, in which G=3, we then achieve three matrices, whose maximum dimensions are M/G x N. A different process is then activated for working on each sub-matrix. The output of the parallel process or thread will be a set of SF allocations relevant for the group of EDs included in the data split.

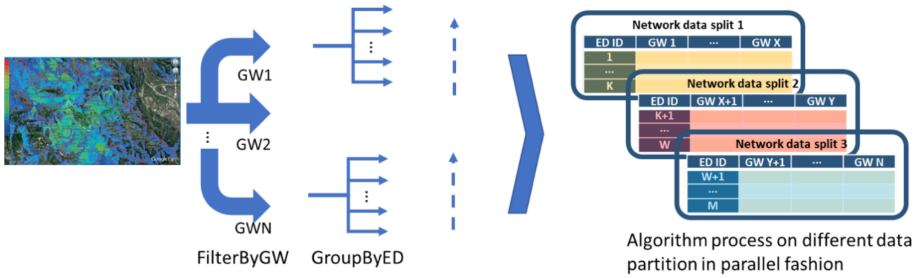


Figure 7.3. Flow diagram of the implemented optimization query.

We envision to allocate the execution of these process at the network edge. We assume that a set of GWs in physical proximity is managed by an edge server that executes one thread. Figure 7.3 shows the steps envisioned from the proposed approach.

7.4.2 Query Implementation

The following snippet of code reports the core of the optimization query using the ELEGANT API.

```
Aggregation aggregation1 =
    Aggregation.average("rssi").as("rssiaverage");
```

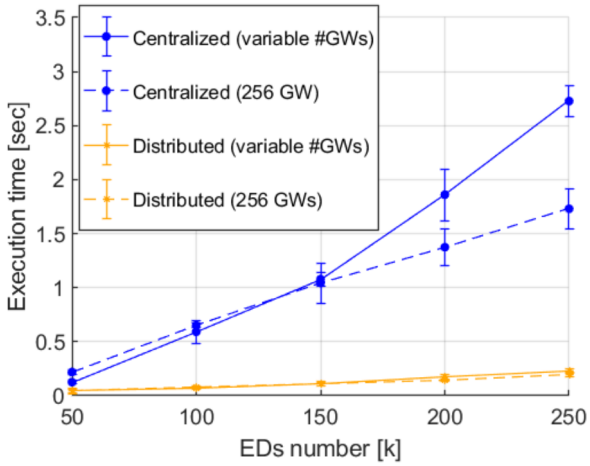


Figure 7.4. Comparison of the distributed approach to the centralized version with an increasing number of EDs, from 50×10^3 to 250×10^3.

```
// Create a streaming query
Sink queryGroup = nebulaStreamRuntime
    .readFromSource("lora_stream")
    .project(attribute("devaddr"),
       attribute("agent_time"),
        attribute("gwmac"), attribute("rssi")
    ).filter(attribute("gwmac").equalTo(id1))
    .window(TumblingWindow. of(eventTime("agent_time"),
        TimeMeasure.hours(windowTime))
    ).byKey("devaddr")
    .apply(aggregation1).sort()
    .sink(new MQTTSink(address, topicGroup, user,
        maxBufferedMSGs, timeUnit, messageDelay,
        qualityOfService, asynchronousClient)
    );
int response = nebulaStreamRuntime
    .executeQuery(queryGroup, "BottomUp");
```

We first connect the data steam through the source operator, and then we select a subgroup of the packet fields through the project operator. According to the presented scheme, we filter a stream of packets received from a specific GW. To consider a time window on the stream, we apply the `window.bykey` operator twice. In the first window, packets of the same ED and same GW are grouped for computing as additional data fields the number of packets and the average RSSI per ED. Finally, the `window.bykey` also sorts the results by using the RSSI values as ordering criterion.

The proposed approaches aim to improving performance in terms of network scalability. For this reason, we consider improvement on the execution time under different assumptions about the node density and number of served GWs. The idea is to move the processing and optimization logic from the cloud-based NS to an

intermediate edge layer (which is responsible for controlling multiple Gateways in a cluster) or even to a single Gateway. In particular, when the threads are executed by the Gateways or by the edge nodes, the volume of backhaul traffic can be significantly reduced. We are considering parallelizing the algorithm's execution across multiple edge nodes on the network layer.

Figure 7.4 shows the execution times of the two implementations (centralized and distributed with 4 workers). Different colors and markers are employed for distinguishing the performance achieved under the two scenarios. In particular, we assume that EDs vary from 50×10^3 to 250×10^3, while the number of GWs is kept constant to 256 (dashed lines) or gradually increased from 100 to 500 with a step of 100 GWs (continuous lines). Indeed, to cope with an increasing traffic demand in LoRa networks, it is common to deploy more GWs. Error bars refer to the variability of execution times achieved in multiple execution runs and in particular to a confidence interval equal to 95% of the results. From the figure, we note that the distributed approach significantly improves the execution time from 2.727s to 0.233s for the case with a variable number of GWs and from 1.730s to 0.194s for the case with a fixed number of GWs.

Chapter 8

Conclusions

In this thesis we considered the security of IoT-oriented protocols, focusing on LoRaWAN. First we reviewed the state of the art for Intrusion Detection Systems (IDSs) targeting IoT environments, which include detection techniques which may or may not be based on Edge computing paradigm. We introduced the challenges that an IDS has to face when deployed in an Edge environment. We then moved on LoRaWAN technology, analyzing its security and showing which threats and vulnerabilities have been already brought up by the existing literature. We started profiling the traffic generated by hundreds of devices, extracting key performance metrics which can be useful to a network operator in order to optimize the network operation. Analyzing the results, we found out the existence of two potential security issues in LoRaWAN. The first issue concerns a Denial-Of-Service (DoS) attack where one or more target devices are made unable to receive any downlink packet from the Network Server. This DoS has potential impacts not only on the operation of applications, such as a downlink-based application that foresees actuators, but also on the execution of the LoRaWAN protocol itself, impacting the Join procedure and the Adaptive Data Rate algorithms. The second vulnerability deals with the re-identification of DevAddr-DevEUI of a device after a Join procedure. We identified which privacy threats are the network operator and the users exposed to. For example, data can be analyzed using load monitoring techniques to infer activities of the consumers, which is considered to be sensitive data that must be protected for preserving the users' privacy. Finally, we showed the relation between Edge computing and LoRaWAN. In particular, we showed how one can leverage Edge computing to enhance the security of a LoRaWAN network by re-engineering our re-identification algorithm into a distributed Edge-enabled IDS, through the use of the ELEGANT framework. Future work can be envisioned on LoRaWAN security and the application of Edge computing in LoRaWAN environments. In particular, the integration of Edge computing in LoRaWAN has always been challenging due to the client-server architecture of LoRaWAN that foresees the gateways on the Edge as transparent to the protocol execution. We have also shown that the Edge can be useful to network management operations in LoRaWAN, by introducing an Edge computing-based radio resource allocation algorithm. We hope that following works will investigate the applications and the integration of Edge computing in LoRaWAN, which, as of today, still remain a relatively unexplored field.

Bibliography

[1] Alazab, A., Hobbs, M., Abawajy, J., and Khraisat, A. *Developing an Intelligent Intrusion Detection and Prevention System against Web Application Malware*, p. 177–184. Springer Berlin Heidelberg (2013). doi:10.1007/978-3-642-40597-6_15.

[2] Aldweesh, A., Derhab, A., and Emam, A. Z. Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems*, **189** (2020), 105124. doi:10.1016/j.knosys.2019.105124.

[3] Amit, I., Matherly, J., Hewlett, W., Xu, Z., Meshi, Y., and Weinberger, Y. Machine learning in cyber-security -problems, challenges and data sets. *ArXiv e-prints*, (2019).

[4] Ancian, L. T. and Cunche, M. Re-identifying addresses in LoRaWAN networks. Research Report RR-9361, Inria Rhône-Alpes ; INSA de Lyon (2020). Available from: https://inria.hal.science/hal-02926894.

[5] Aras, E., Ramachandran, G. S., Lawrence, P., and Hughes, D. Exploring the security vulnerabilities of lora. *2017 3rd IEEE International Conference on Cybernetics (CYBCONF)*, (2017). Available from: http://dx.doi.org/10.1109/CYBConf.2017.7985777, doi:10.1109/cybconf.2017.7985777.

[6] Bai, L., Yao, L., Kanhere, S. S., Wang, X., and Yang, Z. Automatic device classification from network traffic streams of internet of things. *ArXiv e-prints*, (2018). arXiv:1812.09882.

[7] Baiocchi, A. *Network Traffic Engineering*. Wiley (2020).

[8] Bajaj, K. and Arora, A. Dimension reduction in intrusion detection features using discriminative machine learning approach. In *IJCSI International Journal of Computer Science*, vol. 10 (2013).

[9] Barriga, J. J., Sulca, J., León, J., Ulloa, A., Portero, D., García, J., and Yoo, S. G. A smart parking solution architecture based on lorawan and kubernetes. *Applied Sciences*, **10** (2020). Available from: https://www.mdpi.com/2076-3417/10/13/4674, doi:10.3390/app10134674.

[10] Bernardinetti, G., Mancini, F., and Bianchi, G. Disconnection attacks against lorawan 1.0.x abp devices (2020). Available

from: http://dx.doi.org/10.1109/MedComNet49392.2020.9191495, doi:10.1109/medcomnet49392.2020.9191495.

[11] Bhatia, L., Breza, M., Marfievici, R., and McCann, J. A. Loed: The lorawan at the edge dataset. *Proceedings of the Third Workshop on Data: Acquisition To Analysis*, (2020). Available from: http://dx.doi.org/10.1145/3419016.3431491, doi:10.1145/3419016.3431491.

[12] Blázquez-García, A., Conde, A., Mori, U., and Lozano, J. A. A review on outlier/anomaly detection in time series data. *ArXiv e-prints*, (2020). arXiv:2002.04236.

[13] Brauckhoff, D., Salamatian, K., and May, M. Applying pca for traffic anomaly detection: Problems and solutions. In *IEEE INFOCOM 2009 - The 28th Conference on Computer Communications*. IEEE (2009). doi:10.1109/infcom.2009.5062248.

[14] Brik, V., Banerjee, S., Gruteser, M., and Oh, S. Wireless device identification with radiometric signatures. In *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, MobiCom '08, p. 116–127. Association for Computing Machinery, New York, NY, USA (2008). ISBN 9781605580968. Available from: https://doi.org/10.1145/1409944.1409959, doi:10.1145/1409944.1409959.

[15] Brocaar, O. Chirpstack open-source lorawan network server stack. https://www.chirpstack.io. Accessed: 15-08-2023.

[16] Bräuer, S., Zubow, A., Zehl, S., Roshandel, M., and Mashhadi-Sohi, S. On practical selective jamming of bluetooth low energy advertising. In *2016 IEEE Conference on Standards for Communications and Networking (CSCN)*, pp. 1–6 (2016).

[17] Butun, I., Pereira, N., and Gidlund, M. Security risk analysis of lorawan and future directions. *Future Internet*, **11** (2019). Available from: https://www.mdpi.com/1999-5903/11/1/3, doi:10.3390/fi11010003.

[18] Canbalaban, E. and Sen, S. A cross-layer intrusion detection system for rpl-based internet of things. In *Ad-Hoc, Mobile, and Wireless Networks* (2020).

[19] Chatzigiannakis, V., Papavassiliou, S., and Androulidakis, G. Improving network anomaly detection effectiveness via an integrated multi-metric-multi-link (m3l) pca-based approach. *Security and Communication Networks*, **2** (2009), 289–304. doi:10.1002/sec.69.

[20] Choudhary, S. and Kesswani, N. A survey: Intrusion detection techniques for internet of things. *International Journal of Information Security and Privacy*, **13** (2019), 86–105. Available from: http://dx.doi.org/10.4018/ijisp.2019010107, doi:10.4018/ijisp.2019010107.

[21] Danish, S. M., Nasir, A., Qureshi, H. K., Ashfaq, A. B., Mumtaz, S., and Rodriguez, J. Network intrusion detection system for jamming

attack in lorawan join procedure. In *2018 IEEE International Conference on Communications (ICC)*, pp. 1–6 (2018).

[22] De la Hoz, E., De La Hoz, E., Ortiz, A., Ortega, J., and Prieto, B. Pca filtering and probabilistic som for network intrusion detection. *Neurocomputing*, **164** (2015), 71–81. doi:10.1016/j.neucom.2014.09.083.

[23] Dean, J. and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, **51** (2008), 107–113. Available from: https://doi.org/10.1145/1327452.1327492, doi:10.1145/1327452.1327492.

[24] Desai, B. A., Divakaran, D. M., Nevat, I., Peter, G. W., and Gurusamy, M. A feature-ranking framework for iot device classification. *2019 11th International Conference on Communication Systems and Networks (COMSNETS)*, (2019). Available from: http://dx.doi.org/10.1109/COMSNETS.2019.8711210, doi:10.1109/comsnets.2019.8711210.

[25] Di Vincenzo, V., Heusse, M., and Tourancheau, B. Improving downlink scalability in lorawan. *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, (2019). Available from: http://dx.doi.org/10.1109/ICC.2019.8761157, doi:10.1109/icc.2019.8761157.

[26] Dua, S. and Du, X. *Data Mining and Machine Learning in Cybersecurity*. Auerbach Publications (2016). doi:10.1201/b10867.

[27] Dwork, C., Lynch, N., and Stockmeyer, L. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, **35** (1988), 288–323. doi:10.1145/42282.42283.

[28] Dönmez, T. C. and Nigussie, E. Security of lorawan v1.1 in backward compatibility scenarios. *Procedia Computer Science*, **134** (2018), 51–58. Available from: http://dx.doi.org/10.1016/j.procs.2018.07.143, doi:10.1016/j.procs.2018.07.143.

[29] Elhag, S., Fernández, A., Bawakid, A., Alshomrani, S., and Herrera, F. On the combination of genetic fuzzy systems and pairwise learning for improving detection rates on intrusion detection systems. *Expert Systems with Applications*, **42** (2015), 193–202. doi:10.1016/j.eswa.2014.08.002.

[30] Eskandari, M., Janjua, Z. H., Vecchio, M., and Antonelli, F. Passban ids: An intelligent anomaly-based intrusion detection system for iot edge devices. *IEEE Internet of Things Journal*, **7** (2020), 6882.

[31] Ferre, G. Collision and packet loss analysis in a lorawan network. *2017 25th European Signal Processing Conference (EUSIPCO)*, (2017). Available from: http://dx.doi.org/10.23919/EUSIPCO.2017.8081678, doi:10.23919/eusipco.2017.8081678.

[32] Flanagan, K., Fallon, E., Connolly, P., and Awad, A. Network anomaly detection in time series using distance based outlier detection with cluster density analysis. In *2017 Internet Technologies and Applications (ITA)*. IEEE (2017). doi:10.1109/itecha.2017.8101921.

[33] Franceschi, A., Garlisi, D., Spadaccino, P., Cuomo, F., and Tinnirello, I. Lorawan iot solutions: the network management becomes a big data problem. *Submitted to Computer Communications*, (2023).

[34] Frankel, S., Kent, K., Lewkowski, R., Orebaugh, A. D., W.Ritchey, R., and Sharma, S. R. Guide to ipsec vpns. *National Institute of Standards and Technology*, (2005). doi:10.6028/NIST.SP.800-77r1.

[35] Fumero, J., Papadimitriou, M., Zakkak, F. S., Xekalaki, M., Clarkson, J., and Kotselidis, C. Dynamic application reconfiguration on heterogeneous hardware. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE 2019, p. 165–178. Association for Computing Machinery, New York, NY, USA (2019). ISBN 9781450360203. Available from: https://doi.org/10.1145/3313808.3313819, doi:10.1145/3313808.3313819.

[36] Garlisi, D., Martino, A., Zouwayhed, J., Pourrahim, R., and Cuomo, F. Exploratory approach for network behavior clustering in lorawan. *Journal of Ambient Intelligence and Humanized Computing*, (2021). Available from: http://dx.doi.org/10.1007/s12652-021-03121-z, doi:10.1007/s12652-021-03121-z.

[37] Garlisi, D., Tinnirello, I., Bianchi, G., and Cuomo, F. Capture aware sequential waterfilling for lorawan adaptive data rate. *IEEE Transactions on Wireless Communications*, **20** (2021), 2019–2033. Available from: http://dx.doi.org/10.1109/TWC.2020.3038638, doi:10.1109/twc.2020.3038638.

[38] Giezeman, W. The things stack. https://www.thethingsindustries.com/stack. Accessed: 15-08-2023.

[39] Gotthard, P. Compact server for private lorawan networks. https://github.com/gotthardp/lorawan-server. Accessed: 15-08-2023.

[40] Gu, X., Yang, M., Zhang, Y., Pan, P., and Ling, Z. Fingerprinting network entities based on traffic analysis in high-speed network environment. *Security and Communication Networks*, **2018** (2018), 6124160. Available from: https://doi.org/10.1155/2018/6124160, doi:10.1155/2018/6124160.

[41] Hafeez, I., Antikainen, M., Ding, A. Y., and Tarkoma, S. Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge. *IEEE Transactions on Network and Service Management*, **17** (2020), 45–59. doi:10.1109/tnsm.2020.2966951.

[42] Hafeez, I., Ding, A. Y., Antikainen, M., and Tarkoma, S. *Real-Time IoT Device Activity Detection in Edge Networks*, p. 221–236. Springer International Publishing (2018). doi:10.1007/978-3-030-02744-5_17.

[43] Han, J. and Wang, J. An enhanced key management scheme for lorawan. *Cryptography*, **2** (2018), 34. Available from: http://dx.doi.org/10.3390/cryptography2040034, doi:10.3390/cryptography2040034.

[44] Hassan, N., Gillani, S., Ahmed, E., Yaqoob, I., and Imran, M. The role of edge computing in internet of things. *IEEE Communications Magazine*, **56** (2018), 110. doi:10.1109/MCOM.2018.1700906.

[45] Hosseinpour, F., Amoli, P., Plosila, J., Hämäläinen, T., and Tenhunen, H. An intrusion detection system for fog computing and iot based logistic systems using a smart data approach. *International Journal of Digital Content Technology and its Applications*, **10** (2016).

[46] Hupperich, T., Maiorca, D., Kührer, M., Holz, T., and Giacinto, G. On the robustness of mobile device fingerprinting: Can mobile users escape modern web-tracking mechanisms? In *Proceedings of the 31st Annual Computer Security Applications Conference*, ACSAC 2015, p. 191–200. Association for Computing Machinery, New York, NY, USA (2015). ISBN 9781450336826. Available from: https://doi.org/10.1145/2818000.2818032, doi:10.1145/2818000.2818032.

[47] Jokar, P. Intrusion detection and prevention for zigbee-based home area networks in smart grids. *IEEE Transactions on Smart Grid*, **PP** (2016), 1. doi:10.1109/TSG.2016.2600585.

[48] Khraisat, A., Gondal, I., and Vamplew, P. An anomaly intrusion detection system using c5 decision tree classifier. In *Trends and Applications in Knowledge Discovery and Data Mining* (edited by M. Ganji, L. Rashidi, B. C. M. Fung, and C. Wang), pp. 149–155. Springer International Publishing, Cham (2018). ISBN 978-3-030-04503-6.

[49] Khraisat, A., Gondal, I., Vamplew, P., and Kamruzzaman, J. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, **2** (2019). doi:10.1186/s42400-019-0038-7.

[50] Kolias, C., Kambourakis, G., Stavrou, A., and Gritzalis, S. Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset. *IEEE Communications Surveys & Tutorials*, **18** (2016), 184.

[51] Lacava, A., Giacomini, E., D'Alterio, F., and Cuomo, F. Intrusion detection system for bluetooth mesh networks: Data gathering and experimental evaluations. In *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 661–666 (2021).

[52] Leu, P., Puddu, I., Ranganathan, A., and Capkun, S. I send, therefore I leak: Information leakage in low-power wide area networks. *CoRR*, **abs/1911.10637** (2019). Available from: http://arxiv.org/abs/1911.10637, arXiv:1911.10637.

[53] Liao, H.-J., Richard Lin, C.-H., Lin, Y.-C., and Tung, K.-Y. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, **36** (2013), 16–24. doi:10.1016/j.jnca.2012.09.004.

[54] Locatelli, P., Spadaccino, P., and Cuomo, F. Hijacking downlink path selection in lorawan. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. IEEE (2021).

[55] Locatelli, P., Spadaccino, P., and Cuomo, F. Ruling out iot devices in lorawan. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–2. IEEE (2022).

[56] Magrin, D., Capuzzo, M., and Zanella, A. A thorough study of lorawan performance under different parameter settings. *IEEE Internet of Things Journal*, **7** (2020), 116–127. Available from: `http://dx.doi.org/10.1109/JIOT.2019.2946487`, `doi:10.1109/jiot.2019.2946487`.

[57] Magán-Carrión, R., Camacho, J., Maciá-Fernández, G., and Ruíz-Zafra, A. Multivariate statistical network monitoring–sensor: An effective tool for real-time monitoring and anomaly detection in complex networks and systems. *International Journal of Distributed Sensor Networks*, **16** (2020), 155014772092130. `doi:10.1177/1550147720921309`.

[58] Marais, J. M., Abu-Mahfouz, A. M., and Hancke, G. P. A survey on the viability of confirmed traffic in a lorawan. *IEEE Access*, **8** (2020), 9296–9311. Available from: `http://dx.doi.org/10.1109/ACCESS.2020.2964909`, `doi:10.1109/access.2020.2964909`.

[59] Mayzaud, A., Badonnel, R., and Chrisment, I. A distributed monitoring strategy for detecting version number attacks in rpl-based networks (2017). Available from: `http://dx.doi.org/10.1109/TNSM.2017.2705290`, `doi:10.1109/tnsm.2017.2705290`.

[60] Moore, D. Lorawan v1.0.4 specification. `https://resources.lora-alliance.org/technical-specifications/ts001-1-0-4-lorawan-l2-1-0-4-specification`. Accessed: 15-08-2023.

[61] Moore, D. Lorawan v1.1 specification. `https://resources.lora-alliance.org/technical-specifications/lorawan-specification-v1-1`. Accessed: 15-08-2023.

[62] Moore, D. Technical recommendations for preventing state synchronization issues around lorawan 1.0 join procedure. `https://lora-alliance.org/wp-content/uploads/2020/11/lorawan-1.0.x-join-synch-issues-remedies-v1.0.0.pdf`. Accessed: 15-08-2023.

[63] Nguyen, T. T. and Reddi, V. J. Deep reinforcement learning for cyber security. *ArXiv e-prints*, (2019).

[64] Niedermaier, M., Striegel, M., Sauer, F., Merli, D., and Sigl, G. Efficient intrusion detection on low-performance industrial iot edge node devices. *ArXiv e-prints*, (2019). `arXiv:1908.03964`.

[65] Pandeeswari, N. and Kumar, G. Anomaly detection system in cloud environment using fuzzy clustering based ann. *Mobile Networks and Applications*, **21** (2015). `doi:10.1007/s11036-015-0644-x`.

[66] Pickle, P. H. Lora and lorawan: A technical overview. https://lora-developers.semtech.com/uploads/documents/files/LoRa_and_LoRaWAN-A_Tech_Overview-Downloadable.pdf. Accessed: 15-08-2023.

[67] Pickle, P. H. Lora edge. https://www.semtech.com/products/wireless-rf/lora-edge. Accessed on 15-08-2023.

[68] Pickle, P. H. Lora technology is connecting our smart planet, semtech corporation. 2022. https://www.semtech.com/lora/lora-applications. Accessed on 15-08-2023.

[69] Pickle, P. H. Lora technology overview. https://www.semtech.com/lora. Accessed: 15-08-2023.

[70] Pickle, P. H. Monitoring of covid-19 vaccines storage and transportation, orionm2m. https://lora-alliance.org/member-press-release/monitoring-of-covid-19-vaccines-storage-and-transportation. Accessed on 15-08-2023.

[71] Pickle, P. H. Smart cities of the future. https://info.semtech.com/lora-smart-cities-ebook. Accessed on 15-08-2023.

[72] Qiu, S., Liu, Q., Zhou, S., and Wu, C. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, **9** (2019), 909. doi:10.3390/app9050909.

[73] Reddy, A., et al. Using gaussian mixture models to detect outliers in seasonal univariate network traffic. In *2017 IEEE Security and Privacy Workshops (SPW)*. IEEE (2017). doi:10.1109/spw.2017.9.

[74] Ringberg, H., Soule, A., Rexford, J., and Diot, C. Sensitivity of pca for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems - SIGMETRICS '07*. ACM Press (2007). doi:10.1145/1254882.1254895.

[75] Robyns, P., Marin, E., Lamotte, W., Quax, P., Singelée, D., and Preneel, B. Physical-layer fingerprinting of lora devices using supervised and zero-shot learning. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17, p. 58–63. Association for Computing Machinery, New York, NY, USA (2017). ISBN 9781450350846. Available from: https://doi.org/10.1145/3098243.3098267, doi:10.1145/3098243.3098267.

[76] Salman, O., Elhajj, I., Kayssi, A., and Chehab, A. Edge computing enabling the internet of things. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pp. 603–608 (2015). doi:10.1109/WF-IoT.2015.7389122.

[77] Salo, F., Nassif, A. B., and Essex, A. Dimensionality reduction with ig-pca and ensemble classifier for network intrusion detection. *Comput. Networks*, **148** (2019), 164.

[78] Sandhu, R., Sohal, A., and Sood, S. Identification of malicious edge devices in fog computing environments. *Information Security Journal: A Global Perspective*, **26** (2017), 1. doi:10.1080/19393555.2017.1334843.

[79] Schneible, J. and Lu, A. Anomaly detection on the edge. *MILCOM 2017 - 2017 IEEE Military Communications Conference (MILCOM)*, (2017), 678.

[80] Sedjelmaci, H., Senouci, S. M., and Al-Bahri, M. A lightweight anomaly detection technique for low-resource iot devices: A game-theoretic methodology. In *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6 (2016).

[81] Spadaccino, P., Crinó, F. G., and Cuomo, F. Lorawan behaviour analysis through dataset traffic investigation. *Sensors*, **22** (2022), 2470.

[82] Spadaccino, P. and Cuomo, F. Intrusion detection systems for iot: Opportunities and challenges offered by edge computing (2022). Available from: http://dx.doi.org/10.52953/WNVI5792, doi:10.52953/wnvi5792.

[83] Spadaccino, P., Garlisi, D., Cuomo, F., Pillon, G., and Pisani, P. Discovery privacy threats via device de-anonymization in lorawan. *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*, (2021). Available from: http://dx.doi.org/10.1109/MedComNet52149.2021.9501247, doi:10.1109/medcomnet52149.2021.9501247.

[84] Spadaccino, P., Garlisi, D., Cuomo, F., Pillon, G., and Pisani, P. Discovery privacy threats via device de-anonymization in lorawan. *Computer Communications*, **189** (2022), 1–10. Available from: http://dx.doi.org/10.1016/j.comcom.2022.02.017, doi:10.1016/j.comcom.2022.02.017.

[85] Sudqi Khater, B., Abdul Wahab, A. W. B., Idris, M. Y. I. B., Abdulla Hussain, M., and Ahmed Ibrahim, A. A lightweight perceptron-based intrusion detection system for fog computing. *Applied Sciences*, **9** (2019), 178.

[86] Sung, W.-J., Ahn, H.-G., Kim, J.-B., and Choi, S.-G. Protecting end-device from replay attack on lorawan. *2018 20th International Conference on Advanced Communication Technology (ICACT)*, (2018). Available from: http://dx.doi.org/10.23919/ICACT.2018.8323683, doi:10.23919/icact.2018.8323683.

[87] Symantec. Internet security threat report 2017.

[88] Terenzi, F., Spadaccino, P., and Cuomo, F. Privacy monitoring of lorawan devices through traffic stream analysis. In *23rd IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)* (2022).

[89] Thangavelu, V., Divakaran, D. M., Sairam, R., Bhunia, S. S., and Gurusamy, M. Deft: A distributed iot fingerprinting technique. *IEEE Internet of Things Journal*, **6** (2019), 940.

[90] Tonyali, S., Akkaya, K., Saputro, N., Uluagac, A. S., and Nojoumian, M. Privacy-preserving protocols for secure and reliable data aggregation in iot-enabled smart metering systems. *Future Generation Computer Systems*, **78** (2018), 547.

[91] Toro-Betancur, V., Premsankar, G., Slabicki, M., and Di Francesco, M. Modeling communication reliability in lora networks with device-level accuracy. *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, (2021). Available from: `http://dx.doi.org/10.1109/INFOCOM42981.2021.9488783`, `doi:10.1109/infocom42981.2021.9488783`.

[92] Utomo, D. and Hsiung, P. Anomaly detection at the iot edge using deep learning. In *2019 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW)*, pp. 1–2 (2019).

[93] Vergara, J. R. and Estévez, P. A. A review of feature selection methods based on mutual information. *Neural Computing and Applications*, **24** (2013), 175–186. `doi:10.1007/s00521-013-1368-0`.

[94] Wang, S., Li, B., Yang, M., and Yan, Z. Intrusion detection for wifi network: A deep learning approach (2019). Available from: `http://dx.doi.org/10.1007/978-3-030-06158-6_10`, `doi:10.1007/978-3-030-06158-6_10`.

[95] Wimmer, C., Haupt, M., Van De Vanter, M. L., Jordan, M., Daynès, L., and Simon, D. Maxine: An approachable virtual machine for, and in, java. *ACM Trans. Archit. Code Optim.*, **9** (2013). Available from: `https://doi.org/10.1145/2400682.2400689`, `doi:10.1145/2400682.2400689`.

[96] Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., and Wang, C. Machine learning and deep learning methods for cybersecurity. *IEEE Access*, **6** (2018), 35365–35381. `doi:10.1109/access.2018.2836950`.

[97] Yang, X., Karampatzakis, E., Doerr, C., and Kuipers, F. Security vulnerabilities in lorawan. *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, (2018). Available from: `http://dx.doi.org/10.1109/IoTDI.2018.00022`, `doi:10.1109/iotdi.2018.00022`.

[98] Ye, N., Emran, S. M., Chen, Q., and Vilbert, S. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on Computers*, **51** (2002), 810.

[99] Ząbkowski, T. and Gajowniczek, K. Smart metering and data privacy issues. *Information systems in Management*, **2** (2013), 239.

[100] Zhao, Y., Chen, J., Wu, D., Teng, J., Sharma, N., Sajjanhar, A., and Blumenstein, M. Network anomaly detection by using a time-decay closed frequent pattern. *Information*, **10** (2019), 262. `doi:10.3390/info10080262`.