Addressing Cybersecurity and Nodes Localization on LoRaWAN Networks

Amirhossein Noshadi[1,2,†], Taregh Khanjari[1,†], Zhansaya Amangeldiyeva[2], Enrico Cambiaso[1,*], Matteo Repetto[1], Sandro Zappatore[2] and Fabio Patrone[2]

[1]*Consiglio Nazionale delle Ricerche (CNR), Genoa, Italy*
[2]*Department of Electrical, Electronics and Telecommunications Engineering and Naval Architecture (DITEN), University of Genoa, Genoa, Italy*

Abstract

The growing adoption of LoRaWAN for large-scale IoT deployments makes these networks an attractive target for cyber-attacks. While anomalous behaviours might not be difficult to detect, spotting the physical location of attackers remains a major challenge, especially when attacks originate from mobile, low-cost, or self-built devices.

This work introduces an integrated cyber-physical security framework that couples anomaly detection with fine-grained wireless localization to identify and track malicious nodes. First, a supervised anomaly-based algorithm uses features extracted from network telemetry data to detect anomalies. Second, a localization engine leverages TDoA and nonlinear least squares estimation to scan the affected area and locate the attacker, even in case of the attacker movement. We validate the proposed architecture, attack model, detection pipeline, and localization strategy through experimental evaluation. For this purpose, we implemented a novel DoS attack, the DevNonce saturation attack, which exploits the join procedure in OTAA-enabled devices to exhaust nonce space and prevent legitimate nodes from connecting. Experimental results demonstrate the accuracy, feasibility, and applicability of the proposed framework to real LoRaWAN environments, and have also led to the discovery of a novel vulnerability affecting ChirpStack.

Keywords

Internet of Things, wireless sensor networks, cyber-attack, denial of service

1. Introduction

Low Power Wide Area Network (LPWAN) technologies, and in particular the Long Range (LoRa) Wide Area Network (LoRaWAN) protocol, have rapidly boosted large-scale Internet of Things (IoT) deployments. Long communication range, low energy consumption, and cost-effective infrastructure make them suitable for numerous applications, ranging from smart agriculture and asset tracking to industrial monitoring and environmental sensing [1, 2]. However, as the density and heterogeneity of IoT and LoRaWAN-connected devices continue to increase [3], so does the attack surface exposed to malicious actors. Since LoRaWAN networks are only designed to upload data to remote repositories, they have a smaller attack surface than other networks, where Denial of Service (DoS) represents anyway the most relevant threat for IoT applications. To this aim, several studies demonstrate that LoRaWAN is vulnerable to a wide range of cyberattacks, including eavesdropping, replay, spoofing, wormhole, Man-in-the-Middle (MitM), and jamming attempts, which may severely disrupt network availability, compromise data integrity, or undermine users' privacy [4, 5].

Although the LoRaWAN protocol already implements robust encryption and integrity mechanisms, and further cryptographic protections have been proposed in the meanwhile [6], the majority of IoT

Joint National Conference on Cybersecurity (ITASEC & SERICS 2026), February 09-13, 2026, Cagliari, IT
*Corresponding author.
†These authors contributed equally.
amirhosseinnoshadi@cnr.it (A. Noshadi); tareghkhanjari@cnr.it (T. Khanjari); zhansaya.amangeldiyeva@edu.unige.it (Z. Amangeldiyeva); enrico.cambiaso@cnr.it (E. Cambiaso); matteo.repetto@cnr.it (M. Repetto); sandro.zappatore@unige.it (S. Zappatore); fabio.patrone01@unige.it (F. Patrone)
0000-0002-6932-1975 (E. Cambiaso); 0000-0001-8478-2633 (M. Repetto); 0000-0002-0020-9838 (S. Zappatore); 0000-0002-0983-9131 (F. Patrone)

nodes have typically weak security mechanisms, are vulnerable to tampering, and do not run security tools due to resource constraints. Consequently, the implementation of Intrusion Detection Systems (IDS) in the network becomes crucial for identifying ongoing attacks from malicious nodes. The potential of Machine Learning (ML) to implement security solutions for LoRaWAN environments is not new [7], but the design of effective algorithms remains an open research problem, due to the heterogeneity and intermittency of LoRaWAN threats and the absence of universal indicators of compromise.

Detecting an attack, however, is only the first step. If the root cause is not eradicated at its origin, the attack may persist and evolve to elude detection and/or mitigation measures. Locating malicious nodes becomes therefore necessary, so law enforcement authorities can remove them and prosecute their owners. To this aim, localization techniques based on radio measurements, such as Time Difference of Arrival (TDoA) and Angle of Arrival (AoA), have already been investigated in LoRaWAN networks [8], with promising results but also with limitations imposed by environmental noise, multipath propagation, and node mobility. Furthermore, the long-range nature of LoRaWAN transmissions complicates precise localization, especially in dynamic or hard-to-reach environments.

To address these challenges, this work proposes an integrated framework for detecting and physically locating malicious LoRaWAN devices. In detail, the proposed system combines: (i) a dedicated IDS that inspects network traffic at the Network Server (i.e., via the Message Queuing Telemetry Transport (MQTT) broker in ChirpStack); and (ii) a fine-grained localization component based on TDoA and Non-linear Least Squares (NLS) optimization.

The main contributions of this paper are the following: (i) we design an end-to-end incident response workflow for attack detection and localization in LoRaWAN networks, integrating monitoring, Artificial Intelligence (AI)-driven IDS components, and mobile localization platforms; (ii) we describe a new vulnerability in ChirpStack v3.12.0 for LoRaWAN v1.0.3, found while evaluating the performance of the *DevNonce saturation* attack; (iii) we develop a detection module for this novel LoRaWAN DoS attack; (iv) we propose and evaluate a TDoA-NLS localization method, assessing its robustness across multiple attacker mobility models.

The remainder of the paper is structured as follows. Section 2 reviews existing work on LoRaWAN security, IDS mechanisms, and localization techniques. Section 3 defines the threat model, including the considered DevNonce saturation attack scenario. Section 4 presents the system architecture, while Section 5 details the detection and localization components and their implementation. Section 6 discusses the implementation, tests, and numerical results. Finally, Section 7 concludes the paper.

2. Related work

The LoRaWAN protocol has several exploitable vulnerabilities that enable a variety of attacks [9]. IDS solutions can detect deviations from normal patterns that indicate ongoing attacks. Such systems may rely on various approaches, including ML-based IDS [10], or strategies like anomaly detection [11]. Since the application of ML to detect attacks targeting LoRaWAN remains an active research issue [12], exploring new ML methodologies, such as eXplainable AI (XAI) [13], could yield significant insights for the research community.

Once an intrusion or attack is identified, locating the attacker becomes another crucial task [14]. Many of the aforementioned attacks involve the transmission of malicious LoRaWAN packets or interference signals by a device positioned within the network coverage area. Additional data gathered by a LoRaWAN gateway or even a basic passive receiver can assist in pinpointing the source. Techniques such as Time of Arrival (ToA), AoA, and multilateration have been explored for device localization in LoRaWAN networks [8]. TDoA-based approaches determine a device's position by measuring the time differences at which identical packets reach multiple gateways, offering high accuracy and energy efficiency. Moreover, unlike ToA, TDoA requires timing synchronization only between gateways, simplifying deployment.

Despite these advantages, various challenges hinder accurate localization of LoRaWAN devices. These include architectural complexity and environmental conditions that may degrade position estimates [15].

Node mobility further complicates localization by altering transmission behaviours and affecting packet arrival times. Ongoing research seeks to analyse mobility's impact on localization precision and develop algorithms that mitigate motion-induced errors [16]. Hybrid methods, such as integrating TDoA with AoA, have demonstrated improved accuracy, particularly in mobile or urban contexts [17]. Advanced optimization and estimation techniques, including Kalman filters, maximum likelihood estimation, NLS, and ML-based solutions, have also been employed to enhance passive TDoA localization accuracy [18], substantially improving computational performance and reducing localization errors.

3. Problem statement and threat model

Our main purpose is to detect ongoing attacks against a LoRaWAN network, and to locate the malicious nodes that originate them. We assume that the attack can be detected by inspecting the traffic at the Network Server (NS) or Application Server (AS), hence without the need to sniff the radio channel. This assumption has negligible impact on the threat model because the gateway acts as a proxy for the whole LoRaWAN data packet. Our approach splits the problem in two main components: (i) an IDS system able to detect ongoing attacks against the LoRaWAN network, and (ii) a LoRaWAN localization engine, aimed at physically locate the attacker, even in dynamic scenarios.

We consider DoS attacks that exploit protocol vulnerabilities. In particular, we focus on the preliminary join procedure that each node must carry out before sending packets to the LoRaWAN network. The node initiates such process by sending a join request message to the NS. According to [3], such frame contains the device identifier (DevEUI), application identifier (AppEUI), a device-chosen nonce (DevNonce), and an integrity check field (MIC) computed by using a shared Application Key (AppKey). In LoRaWAN v1.1 or greater, the AppEUI field is known as JoinEUI [19]. While both the DevEUI and AppEUI fields are 8 bytes long and the MIC is 4 bytes long, the DevNonce is 2 bytes long and it represents a random value generated by the device each time it attempts to join the network [20]. In LoRaWAN, join requests with the same DevEUI and duplicate DevNonce values are discarded [21]. Such action prevents connection renewals in case the same frame is received (at different times) by multiple gateways. In case the message is correctly received, a join accept message is sent to the node. In Over-The-Air Activation (OTAA) [19] environments, this behavior represents a potential threat, which we denote as *DevNonce saturation*, under the hypothesis an attacker was able to discover the AppKey (which is not unlikely in these scenarios). According to the LoRaWAN v1.0.3 standard, *"the network server keeps track of a certain number of DevNonce values used by the end-device in the past, and ignores join requests with any of these DevNonce values from that end-device"* [22]. Therefore, an attacker knowing both the DevEUI and the AppKey could craft a set of LoRaWAN frames with incremental DevNonce values, to saturate all possible slots the NS can accept. During the attack execution, the legitimate node join process would be invalidated by malicious frames. After the entire queue is saturated, the legitimate node would still be able to communicate on the network, but its (duplicate) messages would be discarded by the AS, hence making the device unable to join the network and, consequently, experiencing a DoS.

Although sending all possible DevNonce values would surely lead a DoS, as the choice of the DevNonce is random, the probability to experience a DoS increases with a larger number of malicious frames. If we define $N = 2^{16} = 65,536$ the maximum number of possible DevNonce values, N_l the number of DevNonce stored in the NS, and N_s the number of malicious frames sent (with different DevNonce values), we compute in Eq. 1 the probability P the legitimate node experiences a DoS when sending a legitimate join request.

$$P = \frac{min(N_s, N_l)}{N} \tag{1}$$

Let $N_s(t)$ and $N_l(t)$ represent, respectively, the number of malicious frames sent and the slots allocated during an attack period t. While the definition of $N_l(t)$ depends on the actual implementation of the NS, being T_s the time between one malicious frame and the next one and assuming the first one is sent at time $t = 0$, we can define $N_s(t)$ as reported in Eq. 2.

$$N_s(t) = \left\lceil \frac{t}{T_s} \right\rceil \tag{2}$$

Hence, we define in Eq. 3 the probability $P(t)$ a licit join request is discarded at time t (and, consequently, a DoS is experienced).

$$P(t) = \frac{min(N_s(t), N_l(t))}{N} \tag{3}$$

We can also define T_{N_l} the time needed to saturate all possible DevNonce slots, as reported in Eq. 4.

$$T_{N_l} = T_s \cdot N_l \tag{4}$$

Although our threat can leverage a DoS through flooding, duty-cycle regulations may influence the attack feasibility [23]. For instance, in the EU868 band, the maximum duty-cycle is 1 percent, meaning that a device can only occupy the channel for 36s per hour in each sub-band. By regulation, after each transmission, the node must wait a mandatory off-period before transmitting again on the same band. The minimum off-period interval between two join requests depends on the spreading factor and the packet time-on-air. For instance, a join request transmitted with Spreading Factor (SF) 7 and 125 kHz bandwidth occupies the air for approximately 70ms [23], resulting in a minimum off-period interval of about 7s. As a result, a real node could not be able to send a large number of join requests in short time. Given our example, for $T_s = 7$s and $N_l = N$, saturating all possible DevNonce slots requires more than 5 days ($T_{N_l} = 458,752$s). Nevertheless, the actual observance of the duty-cycle is fully left to end nodes and not enforced by the network. Therefore, the duty-cycle regulations would not have impact on the attack effectiveness. Considering nodes ignoring or bypassing the duty-cycle, T_{N_l} may assume a lower value ($1/100$ of the value reported above, in our example): having $T_s = 0.07$s, we can have a $T_{N_l} = 4,588$s, equal to around 1.5 hours.

4. System architecture

In order to design a system able to fit our scenario, it is important to consider the following: First of all, (i) it is needed to monitor the network components to collect information useful for detection purposes. Hence, (ii) detection of ongoing attacks on the network is accomplished by analyzing monitored data to identify threats against the LoRaWAN network. Finally, (iii) once an attack is detected, response activities are triggered, by activating localization engines and autonomous response actions. To implement such scenario, our architecture, depicted in Fig. 1, must define connections between components and provide details on possible interaction methods.

Our system is mainly composed of two groups of components: LoRaWAN network and security modules. While the first one includes the typical components of a LoRaWAN network, such as the NS, AS, a set of gateways and sensor nodes, the security modules component concerns detection and response activities. In detail, detection is accomplished by one or more dedicated modules monitoring the NS to get network frames. Once a potentially malicious situation is detected, a new incident is created by the incident response component, by collecting information of the LoRaWAN frame and the gateways receiving it. In case new malicious frames are sent during the same attack, the incident instance is enriched with such data. Such information is retrieved by the localization component, which estimates malicious node position based on packets metadata. Once the attacker location is estimated, such data is shared in the incident response platform, to trigger automated responses. For instance, Unmanned Aerial Vehicles (UAVs) may be involved, by leaving the ground to empower localization by acting as additional LoRaWAN gateways. It is worth mentioning that additional minor components may be integrated into the architecture: for instance, a web-based interface to display data to the user. Nevertheless, as such component and its interaction are not core activities of our incident response pipeline, they have not been depicted in the figure, to improve readability.

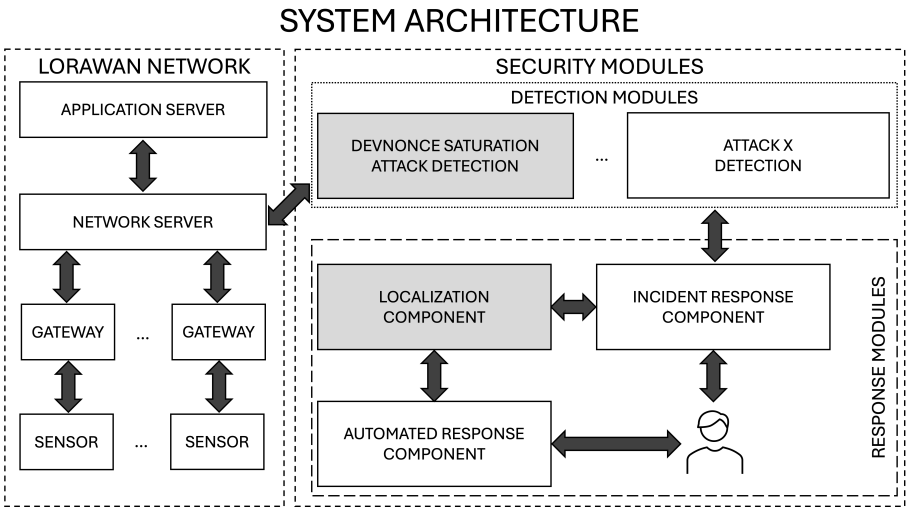


Figure 1: Architecture of the considered system

5. Security modules

In the following, we focus on describing attack detection and node localization activities.

5.1. Attack detection

By focusing on the attack introduced in this paper, we aim to develop a ML model for detecting ongoing DevNonce saturation attacks against a LoRaWAN network. Although several approaches can be used to detect the considered threat, such as statistical methods [24] or gradient boosting algorithms like XGBoost [24], we adopted the Random Forest approach for its robustness [25]. In detail, our model is designed to identify anomalies from normal conditions, characterized during training activities. Particularly, metrics referred to each DevEUI are computed, including join requests and uplink data frequency.

While in Section 3 we have focused on the definition of metrics used to characterize the considered attack and its functioning, we now define useful variables able to support detection capabilities. Particularly, let T_w be the duration of our time window in seconds and let k be the index of the considered time window. We can define a time window W^k as follows.

$$W^k = [T_w \cdot (k-1), T_w \cdot (k)) \tag{5}$$

For each DevEUI value δ, we define $N_j^k(d)$ the number of join request messages in the time window W^k. Similarly, $N_u^k(\delta)$ identifies the number of uplink data packets sent by δ.

To identify a DevNonce saturation attack, we define a binary classification problem. We define a label $\mathbf{y}^k \in \{0, 1\}$ for each time window W_k, where $\mathbf{y}^k = 1$ indicates an anomaly, while $\mathbf{y}^k = 0$ indicates a normal scenario. Using supervised learning, we train and apply a random forest algorithm that aggregates predictions from multiple decision trees [26].

Hence, for each end device δ, we observe data to extract rolling statistics over a set of N_w consecutive sliding windows, ending at the current time index k. Specifically, statistics are computed by using the current window W^k and the preceding $N_w - 1$ windows (i.e., for $N_w = 5$, the set $\{W^{k-4}, W^{k-3}, W^{k-2}, W^{k-1}, W^k\}$ is considered). By considering the number of join request messages across these windows, extrapolated features include rolling (i) sum $S_{N_j}^k(\delta)$, (ii) mean $\mu_{N_j}^k(\delta)$, (iii) standard deviation $\sigma_{N_j}^k(\delta)$. In addition, we also consider the join-to-uplink ratio $R^k(\delta)$, defined as:

$$R^k(\delta) = \frac{N_j^k(\delta)}{N_u^k(k) + 1} \tag{6}$$

where we increase the denominator by one to prevent division by zero.

From the collected data and features, for each time window W_k considered, we construct a vector $\mathbf{x}^k(\delta)$:

$$\mathbf{x}^k(\delta) = [N_j^k(\delta), N_u^k(\delta), S_{N_j}^k(\delta), \mu_{N_j}^k(\delta), \sigma_{N_j}^k(\delta), R^k(\delta)] \in \mathbb{R}^6 \tag{7}$$

While training our algorithm, vectors $\mathbf{x}^k(\delta)$ are extracted by analyzing data sequentially, considering N_w time windows. All vectors data are processed through a random forest classifier trained on labeled normal/anomalous samples, to build a trained model.

For real-time detection, at run time, the $\mathbf{x}^k(\delta)$ vector is computed using the same N_w time windows and the same T_w window duration adopted during training, to ensure feature consistency. Hence, a random forest classifier using N_t decision trees analyzes the extracted run-time vector to collect votes on the predicted class (normal vs anomaly) and determine whether an anomaly is present.

5.2. Attacker localization

To successfully perform the attacker localization, we decided to develop a TDoA localization solution based on the NLS principle and analyse the obtained accuracy considering that the attacker to localize moves following different mobility patterns.

We assume that the packets transmitted by the attacker are received by $L \geq 4$ gateways G_i, located at coordinates (x_i, y_i, z_i), with $i = 1, 2, \ldots, L$.

Denoted as t_i and t_j the packet reception time at gateways G_i and G_j, respectively, and d_i and d_j the distances between the attacker A, located at coordinates (x_a, y_a, z_a) and the gateways G_i and G_j, respectively, the difference of these distances can be obtained as:

$$d_{i,j} = d_i - d_j = (t_i - t_j) \cdot c \tag{8}$$

where c is the speed of light in air.

Considering the Euclidean distance, the left-hand side of 8 can be rewritten as:

$$d_{i,j} = \sqrt{(x_a - x_i)^2 + (y_a - y_i)^2 + (z_a - z_i)^2} - \sqrt{(x_a - x_j)^2 + (y_a - y_j)^2 + (z_a - z_j)^2} \tag{9}$$

Replacing 9 in 8, we obtain a hyperbola $h_{i,j}$ with center in G_i and focus in G_j. Considering one of the gateways as the reference gateway, we can compute the $(L-1)$ distances between the reference gateway and each of the remaining $(L-1)$ gateways and obtain $(L-1)$ hyperbolas that intersect in D. Figure 2 shows a 2D scenario representation with $L = 4$ gateways contributing in the attacker localization and choosing G_1 as the reference gateway.

However, in real scenarios, packet reception times at the gateways could depend not only on propagation times but also on additional factors that can affect each reception time differently. This could cause the $(L-1)$ hyperbolas to not intersect in the right point, or not even in a single point, with a consequent uncertainty in estimating the attacker position.

The proposed TDoA-NLS method reduces localization errors by minimizing the sum of the squares of the differences between the calculated and measured distances [27].

Let consider the measured distance difference $r_{1,i}$ between the reference gateway G_1 and the other $L-1$ gateways G_i, $i = 2, 3, \ldots, L$, as:

$$r_{1,j} = d_{1,j} + n_{1,j} \tag{10}$$

where $d_{1,j}$ is the true distance difference and $n_{1,j}$ the distance error.

The estimated attacker position $\hat{p}_a = (\hat{x}_a, \hat{y}_a, \hat{z}_a)$ corresponds to the smallest value of the cost function $J(\hat{p}_a)$ defined as:

$$J(\hat{p}_a) = [\mathbf{r} - \mathbf{d}(\hat{p}_a)]^T [\mathbf{r} - \mathbf{d}(\hat{p}_a)] \tag{11}$$

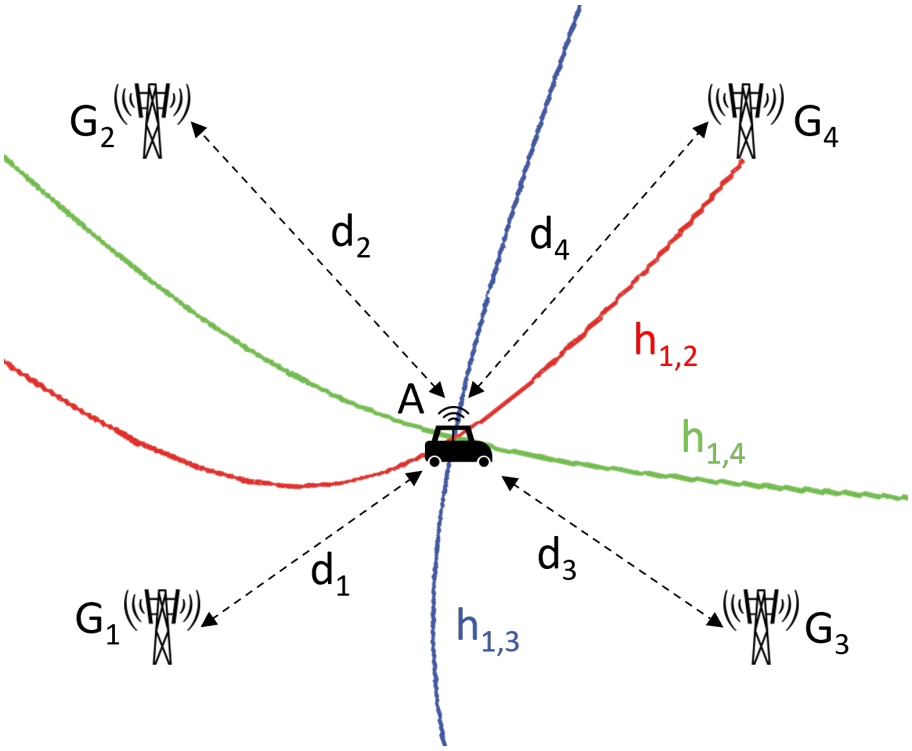


Figure 2: TDoA-based localization with four gateways

where $\mathbf{r}$ is the vector of the measured range differences:

$$\mathbf{r} = \begin{bmatrix} r_{1,2} & r_{1,3} & \cdots & r_{1,L} \end{bmatrix}^T \tag{12}$$

and $\mathbf{d}$ is the vector of the computed distance differences:

$$\mathbf{d}(\hat{p}_a) = \begin{bmatrix} \sqrt{(\hat{x}_a - x_1)^2 + (\hat{y}_a - y_1)^2 + (\hat{z}_a - z_1)^2} - \sqrt{(\hat{x}_a - x_2)^2 + (\hat{y}_a - y_2)^2 + (\hat{z}_a - z_2)^2} \\ \sqrt{(\hat{x}_a - x_1)^2 + (\hat{y}_a - y_1)^2 + (\hat{z}_a - z_1)^2} - \sqrt{(\hat{x}_a - x_3)^2 + (\hat{y}_a - y_3)^2 + (\hat{z}_a - z_3)^2} \\ \vdots \\ \sqrt{(\hat{x}_a - x_1)^2 + (\hat{y}_a - y_1)^2 + (\hat{z}_a - z_1)^2} - \sqrt{(\hat{x}_a - x_L)^2 + (\hat{y}_a - y_L)^2 + (\hat{z}_a - z_L)^2} \end{bmatrix} \tag{13}$$

6. Experimental evaluation results

Our tests focus on evaluating the efficacy of both the detection approach used to identify an ongoing DevNonce saturation attack, as well as localization capabilities.

6.1. DevNonce saturation attack execution

To evaluate the impact of DevNonce saturation attack, we built a test bed composed of an Arduino Uno R3 microcontroller equipped with a LoRaWAN shield, acting as the end device. Such node is programmed to join and repeatedly send uplink data to the network every $T_l = 30$s. Our network also includes a LoRaWAN gateway, represented by a RAK7248C WisGate Developer D4H device, integrating a Raspberry Pi as core processing unit. ChirpStack v3.12.0 has been used as network and application server, supporting LoRaWAN v1.0.3 and embedding an internal MQTT broker service.

Such network and, in particular, the legitimate end node, has been targeted by the attacker. To bypass duty-cycle limitations (see Section 3), we implemented the malicious node as a virtual node directly communicating with the NS through the same LAN. Hence, in our scenario, the malicious node impersonates both the (legitimate) gateway and sensor. In detail, our attacker repeatedly sends

malevolent join request messages with increasing DevNonce values, to saturate all N_l values the NS can store.

According to the definitions in Section 3, the aim of our tests is (i) to estimate the size of the DevNonce queue N_l, (ii) to evaluate the possibility to effectively lead a DoS on the targeted node (related to the retrieval of $P(t)$), and (iii) to analyze the effective time T_{N_l} required to saturate the entire DevNonce queue of the server.

In our scenario, we found that the time needed to send join requests for all 2^{16} possible DevNonce values is 9589.089833s, equal to around 2 hours and 40 minutes. Hence, we found in this case $T_s = 0.1463179$s. Considering such result, our focus was given on the analysis of the targeted network after all the malicious frames are sent, with the aim to identify N_l. To this aim, we analyzed the DevNonce queue on the server. In ChirpStack v3.12.0, the one we adopted during our tests, such queue is available as the `device_activation` table of the PostgreSQL database embedded in ChirpStack. By retrieving the list of records stored in such table, we found that $2^{16} - 78$ records are found. After deeply analysing the situation, as our (greedy) attacker does not verify the reception of a join accept packet, we assumed such number is due to some packet loss. Indeed, by sending new malicious frames "filling in" all 78 positions, we found that 2^{16} records are found on the `device_activation` table. We can therefore assume $N_l = 2^{16}$, hence making the system vulnerable to a DevNonce saturation attack. To ensure such statement, we powered on the legitimate node, by analysing the possibility to effectively join the network. By doing that, we observed that the node is indeed unable to join the network. As a consequence, we can state that $N_l = 2^{16}$ is implemented in our system and, being during our test $N_l = N_s = N$, referring to Eq. 1, we found a probability of DoS $P = 1$. In addition, we can state that $T_{N_l} = 9589.089833$s is our attack execution time.

Such results led us to conclude that the ChirpStack software in version v3.12.0 is affected by a vulnerability, where a malicious user could saturate the entire DevNonce queue with malevolent LoRaWAN frames. In this case, considering each single end device, adequate protection could limit the number of DevNonce values stored in the database, to limit the effects of a saturation attack: considering Eq. 1, by reducing N_l, P decreases as well. On the other side, another protection system may extend the DevNonce size: in case of a length of 4 bytes, hence $N = 2^{32}$, our attack would have taken almost 20 years to saturate all possible values.

6.2. DevNonce saturation attack detection

In order to validate the proposed detection algorithm, we set up an experimental testbed implementing a LoRaWAN scenario. In detail, our testbed is the same of the physical scenario described in Section 6.1, with the exception of the attack node. In this case, the malicious node is represented by the same (minimal) hardware of the legitimate sensor: an Arduino Uno R3 microcontroller equipped with a LoRaWAN shield.

For our evaluation tests, data retrieval is accomplished by extracting information from the MQTT broker of ChirpStack, by accessing in particular to the `gateway/+/event/up` topic. Supervised training has been accomplished by building two separate datasets. Each dataset refers to 60 minutes of network traffic. The first dataset includes legitimate traffic only, hence, data are labeled as $\mathbf{y}^k = 0$. Subsequently, we built a second dataset by conducting a DevNonce saturation attack, where malicious join requests are transmitted at random intervals $T_s \in [8, 60]$. Data included in such dataset are labeled as $\mathbf{y}^k = 1$. Hence, both the datasets have been combined to create a common training dataset.

For our tests, we consider windows of duration $T_w = T_l = 30$s. The training dataset consists of $N_m = \frac{3{,}600+3{,}600}{30} = 240$ labeled observations $\mathcal{D} = \{(\mathbf{x}^k(\delta), \mathbf{y}^k)\}_{k=1}^{N_m}$, split into training set (75%) and testing set (25%). Stratified sampling [28] is employed to preserve the original class distribution across both subsets. A random forest classifier is then trained on the resulting training set. In our configuration, the model constructs $N_t = 100$ decision trees, whose aggregated predictions form the final classifier. During our tests, we consider the analysis of a variable count of consecutive windows $N_w \in \{2, 3, 4, 5\}$, hence, four different models. For each model, obtained accuracy is, respectively, 96% ($N_w = 2$), 100% ($N_w = 3$), 98% ($N_w = 4$), and 98% ($N_w = 5$).

Testing activities have been accomplished by using the built models. In detail, we have observed a period of $T_m = 1,200$s: while during first half of the capture only the legitimate node is present in the system, in the second half the attack also takes place. Particularly, the attacker adopts $T_s = 8$s, a value not breaking the duty-cycle limits. Our detection algorithm has been validated for the entire duration of the test, by considering the four observed windows N_w. For each of the N_w windows analyzed (every $T_w = 30$s), the $\mathbf{x}^k(\delta)$ vector has been computed. Hence, the random forest algorithm has been applied by using $N_t = 100$ decision trees, by considering the model built by evaluating the same number of windows N_w. Considering such application, Fig. 3 reports the percentage of trees voting for anomalies, for the different windows considered.

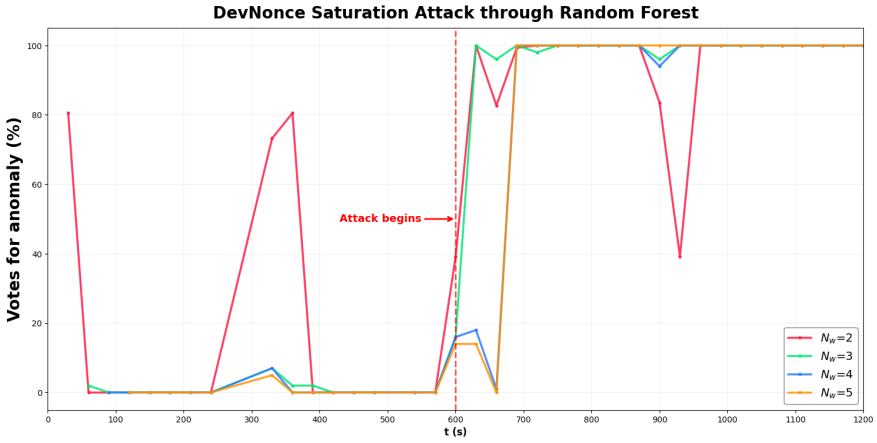


Figure 3: Results of the application of the proposed detection algorithm by adopting different observed windows N_w for a scenario with an attacker beginning at half of the observed period, until its end

By analyzing the results, it is important to consider that curves tend to rise before the attack is executed due to data discretization. In addition, it may be evident that using $N_w = 2$ is not be the right choice, as, considering for instance a detection threshold of 80%, we have anomalies detected two times in the first half of our observation, while in the remaining part detection is less accurate than for greater window sizes. Conversely, the best results are obtained for $N_w = 5$, although in this case, the detection of the attack occurs after 90s from its beginning. The same detection time is found for $N_w = 4$. Differently, for $N_w = 3$, we found results similar to greater N_w but the anomaly is detected after 30s from its beginning. In addition, in such a case, the anomaly is "voted" by 100% of the decision trees, except for $x = 900$, where such number decreases to 96%. Therefore, considering the obtained results, we can state that the use of $N_w = 3$ is the best choice for detection.

6.3. Malicious node localization

The proposed TDoA-NLS position estimation solution has been assessed by implementing the reference scenario within an OMNeT++ and Framework for LoRa (FLoRa) platform. This choice has been driven by the need to generate realistic, even if simulated, data traces about the packet reception times of the gateways to feed as input to the Python script implementing the proposed solution and also linked to the Incident response platform.

The scenario configuration settings are reported in Table 1.

Different tests have been performed considering the position of the attacker fixed or moving following different movement models, as shown in Figure 4:

1. *Scenario 1 - Fixed attacker*: the attacker position is fixed and do not change for the entire test;
2. *Scenario 2 - Linear movement attacker*: the attacker position changes following a linear pattern from West to Est at constant speed;

Table 1
Scenario configuration settings

Number of gateways L	4
Number of packets transmitted by the attacker N	20
Packet inter-transmission time	30s
Simulation duration	600s
Search area size	4200m × 2800m

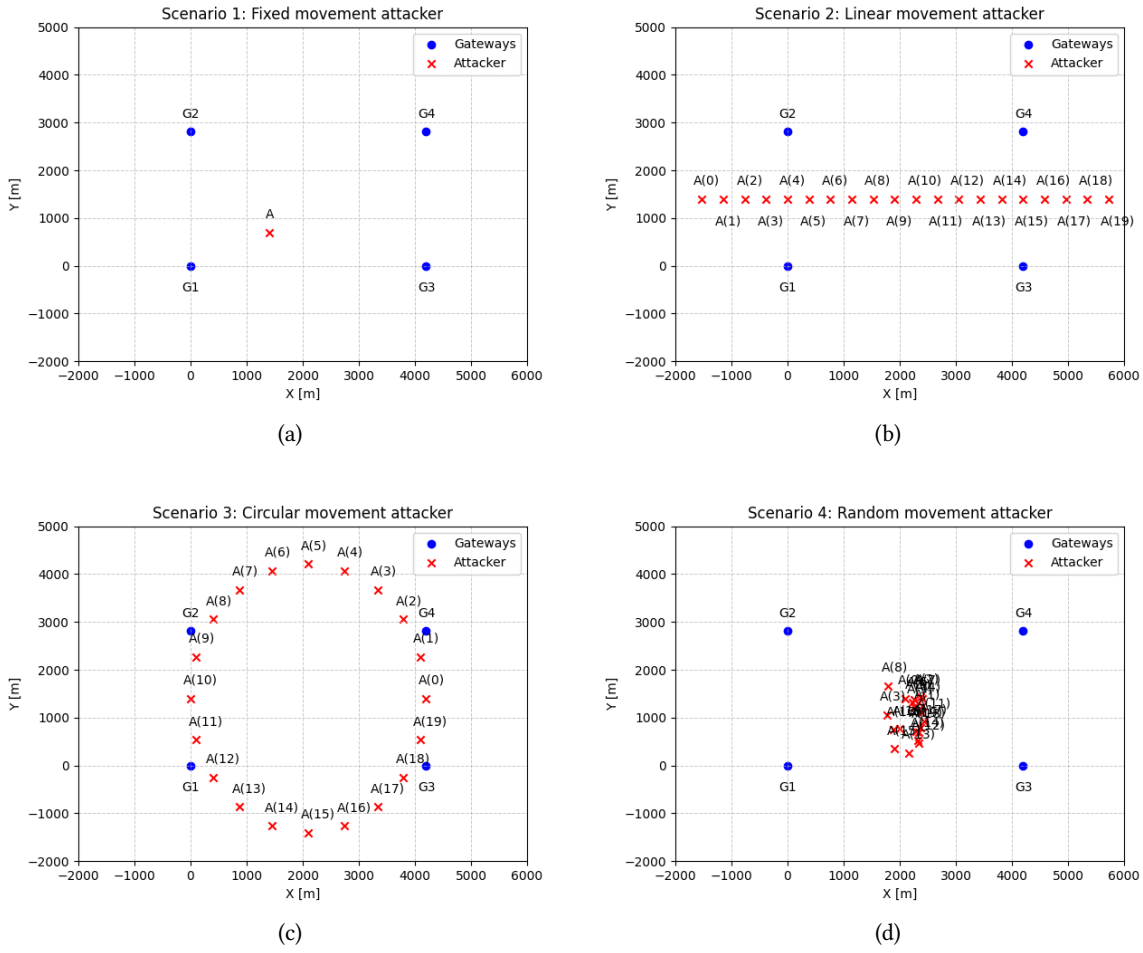


Figure 4: Considered attacker movement models: (a) Fixed, (b) Linear, (c) Circular, and (d) Random

3. *Scenario 3 - Circular movement attacker*: the attacker position changes following a circular pattern within the monitored area;

4. *Scenario 4 - Random movement attacker*: the attacker position changes following a random pattern with constant speed and direction changes after each packet transmission;

The accuracy of the estimated positions has been measured in terms of the following statistical error indicators [29]:

- *Localization Error (LE)*: Euclidean distance between each actual and estimated attacker position. It is defined as:

$$\text{LE} = \sqrt{(x_a - \hat{x}_a)^2 + (y_a - \hat{y}_a)^2 + (z_a - \hat{z}_a)^2} \tag{14}$$

- *Average Localization Error (ALE)*: average of the differences between the actual and estimated device positions. It is defined as:

$$\text{ALE} = \frac{1}{N} \sum_{k=1}^{N} \sqrt{[(x_a - \hat{x}_a)^2 + (y_a - \hat{y}_a)^2 + (z_a - \hat{z}_a)^2]} \tag{15}$$

- *Root Mean Squared Error (RMSE)*: quadratic mean of the differences between the actual and estimated device positions. It is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{k=1}^{N} [(x_a - \hat{x}_a)^2 + (y_a - \hat{y}_a)^2 + (z_a - \hat{z}_a)^2]} \tag{16}$$

Figures 5 and 6 show the obtained results.

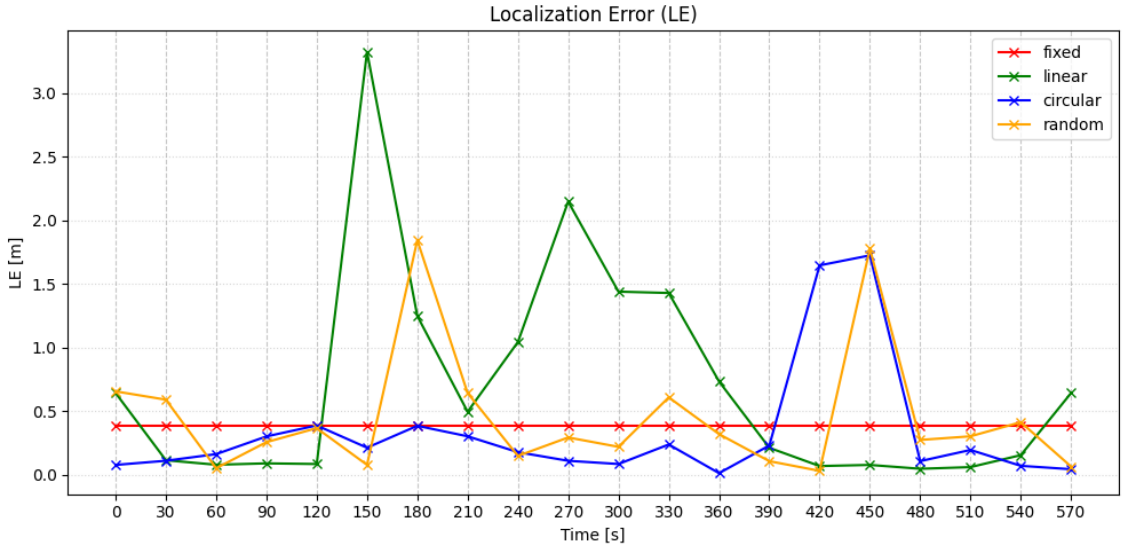
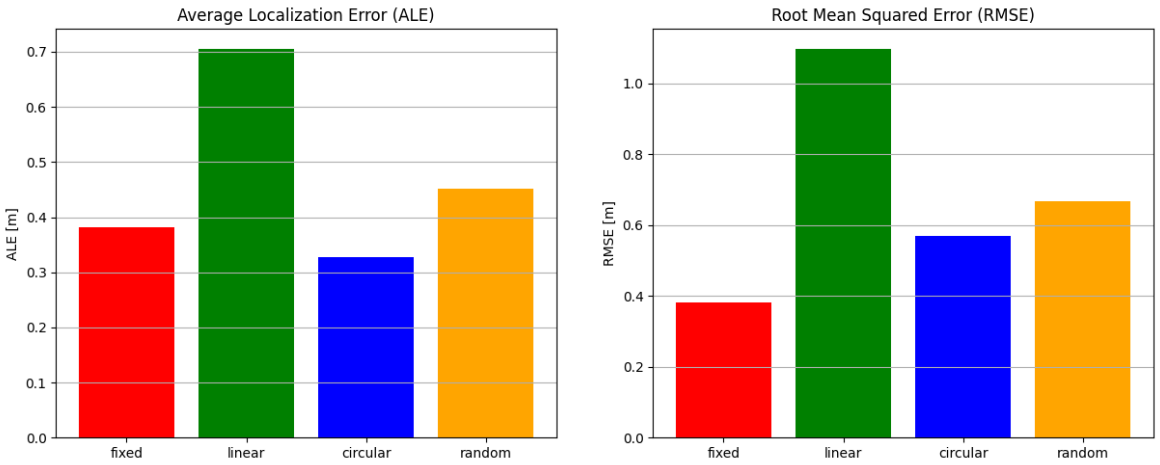


Figure 5: LE obtained in the four considered scenarios

The results demonstrate that positioning errors remain consistently low across all four evaluated scenarios. The maximum LE reaches approximately 3.5 m for the linear movement model, with most measurements falling below 1.5 m. The ALE exhibits a comparable trend, with values around 0.7 m for the linear model, and even lower values ranges between 0.3 m and 0.5 m for the other three models. Despite these variations, the differences in performance across the different attacker movement models are minimal and do not significantly impact the overall system accuracy. Furthermore, the relative position of the attacker with respect to the gateways does not influence the results, under the condition that all packets transmitted by the attacker are received by the four gateways. This observation is further supported by the RMSE values, which show a similar pattern: roughly 1.1 m for the linear models, and about 0.7 m, 0.6 m, and 0.4 m for the random, circular, and fixed models, respectively. Overall, these results are encouraging, indicating that the proposed solution achieves satisfactory localization accuracy given the considered scenario and aim, the used input information, and the fact that the system and communication technology have not been originally designed for localization purposes.



(a) (b)

Figure 6: ALE (a) and RMSE (b) obtained in the four considered scenarios

7. Conclusions

This work presented a cyber-physical security framework for detecting and localizing malicious nodes in LoRaWAN networks. We introduced the DevNonce saturation attack, a novel threat exploiting LoRaWAN DevNonce field validation during join processes. A single attacker transmitting malicious join requests could invalidate the join request of a legitimate node. While evaluating the feasibility of the attack, we discovered a vulnerability of ChirpStack v3.12.0.

To detect the proposed threat, we developed a supervised anomaly-based IDS based on random forest, trained on real data. To this aim, we found that by analyzing $N_w = 3$ consecutive temporal windows, we are able to detect the attack after 30s from its beginning. Hence, the approach proved to be able to identify a running attack in limited time: considering the scenario may expand the network to a metropolitan area, such result makes it possible to promptly advance adequate countermeasures.

While this study is mostly focused on the specific metrics in DevNonce saturation, the proposed framework architecture is potentially scalable: the anomaly detection engine modules can be adapted to other protocol-level threats, such as replay attacks or malicious gateway behavior.

To address the problem of attacker localization, we introduced a TDoA-NLS approach that estimates the attacker's position whenever its transmitted packets are captured by at least four gateways, combining TDoA measurements with an NLS refinement step to enhance accuracy. Since attackers may move, the method was evaluated under multiple movement models. Across all four scenarios, the proposed solution demonstrated robust performance, achieving a maximum localization error of 3.5m and yielding 92.5% of the estimates with an error below 1.5m.

Although TDoA-based localization is often associated with dense urban or industrial deployments, this requirement does not necessarily limit the applicability of the proposed framework. In LoRaWAN networks, the condition of having at least four receiving gateways can also be met in rural scenarios, where gateways are typically deployed with inter-site distances on the order of 10–12 km. As a result, reliable TDoA localization does not strictly depend on dense infrastructure. Nevertheless, localization accuracy remains influenced by deployment characteristics such as gateway geometry, synchronization accuracy, and propagation conditions, and may degrade in scenarios with fewer available gateways or unfavorable topologies.

Overall, the results show that DevNonce saturation is a practical and disruptive attack, while the proposed IDS and localization pipeline offer effective and deployable countermeasures to protect a LoRaWAN network. Future work will focus on refining the threat model to capture more sophisticated adversarial behaviors, including adaptive and detection-aware attackers that intentionally choose strategies to maximize detection error. Addressing such scenarios would require a different experimental

and methodological setup, potentially involving game-theoretic optimization formulations, which is beyond the scope of the present study. In addition, future research will explore online and adaptive learning approaches for detection. Although these methods are not compatible with the current experimental setup, which primarily targets known vulnerabilities, they may help overcome limitations related to the reliance on labeled data. In this context, unsupervised techniques such as Isolation Forest may be more suitable than Random Forest, as they do not depend on labeled data and are better aligned with adaptive adversarial settings. Finally, an extension to the work will consider expanding the localization framework to multi-attacker scenarios.

Acknowledgments

The work has been financed by the European Union - Next Generation EU, Missione 4 Componente 1, CUP B53D23023810001.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] D. Magrin, M. Capuzzo, A. Zanella, L. Vangelista, M. Zorzi, Performance analysis of lorawan in industrial scenarios, IEEE Transactions on Industrial Informatics 17 (2020) 6241–6250.

[2] P. J. Basford, F. M. J. Bulot, M. Apetroaie-Cristea, S. J. Cox, S. J. Ossont, LoRaWAN for smart city IoT deployments: A long term evaluation, Sensors 20 (2020) 648. doi:10.3390/s20030648.

[3] M. Jouhari, N. Saeed, M.-S. Alouini, E. M. Amhoud, A survey on scalable lorawan for massive iot: Recent advances, potentials, and challenges, IEEE Communications Surveys & Tutorials 25 (2023) 1841–1876.

[4] F. Hessel, L. Almon, M. Hollick, Lorawan security: an evolvable survey on vulnerabilities, attacks and their systematic mitigation, ACM Transactions on Sensor Networks 18 (2023) 1–55.

[5] H. Noura, T. Hatoum, O. Salman, J.-P. Yaacoub, A. Chehab, Lorawan security survey: Issues, threats and possible mitigation techniques, Internet of Things 12 (2020) 100303.

[6] S. Abboud, N. Abdoun, Enhancing lorawan security: An advanced aes-based cryptographic approach, IEEE Access 12 (2023) 2589–2606.

[7] J. S. Nixon, J. J. A. Celin, C. Ganesh, U. P. Kumar, D. Kumar, et al., Lorawan security framework to mitigate dos attacks at the gateway level using machine learning, in: 2024 IEEE Pune Section International Conference (PuneCon), IEEE, 2024, pp. 1–3.

[8] H. Kwasme, S. Ekin, RSSI-based localization using LoRaWAN technology, IEEE Access 7 (2019) 99856–99866. doi:10.1109/ACCESS.2019.2929212.

[9] H. Ruotsalainen, G. Shen, J. Zhang, R. Fujdiak, LoRaWAN physical layer-based attacks and countermeasures, a review, Sensors 22 (2022) 3127. doi:10.3390/s22093127.

[10] G. Kocher, G. Kumar, Machine learning and deep learning methods for intrusion detection systems: recent developments and challenges, Soft Computing 25 (2021) 9731–9763. doi:10.1007/s00500-021-05893-0.

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

[12] H. Noura, T. Hatoum, O. Salman, J.-P. Yaacoub, A. Chehab, LoRaWAN security survey: Issues, threats and possible mitigation techniques, Internet of Things 12 (2020). doi:10.1016/j.iot.2020.100303.

[13] I. Vaccari, A. Carlevaro, S. Narteni, E. Cambiaso, M. Mongelli, explainable and reliable against adversarial machine learning in data analytics, IEEE Access (2022). doi:10.1109/ACCESS.2022.3197299.
[14] M. Repetto, E. Cambiaso, F. Patrone, S. Zappatore, Flying drones to locate cyber-attackers in lorawan metropolitan networks, arXiv preprint arXiv:2509.15725 (2025).
[15] L. E. Marquez, M. Calle, Understanding LoRa-based localization: Foundations and challenges, IEEE Internet of Things Journal 10 (2023) 11185–11198.
[16] N. Podevijn, J. Trogh, A. Karaagac, J. Haxhibeqiri, J. Hoebeke, L. Martens, P. Suanet, K. Hendrikse, D. Plets, W. Joseph, TDoA-based outdoor positioning in a public LoRa network, in: European Conference on Antennas and Propagation (EuCAP), IET, 2018, pp. 1–4.
[17] M. Aernouts, N. BniLam, N. Podevijn, D. Plets, W. Joseph, R. Berkvens, M. Weyn, Combining TDoA and AoA with a particle filter in an outdoor LoRaWAN network, in: Position, Location and Navigation Symposium (PLANS), IEEE, 2020, pp. 1060–1069.
[18] I. El Mourabit, A. Badri, A. Sahel, A. Baghdad, Hyperbolic equation solving algorithms for LTE mobile positioning using TDOA measurements, in: International Conference on Information Technologies and Integrated Production Systems, 2016.
[19] S. Loukil, L. C. Fourati, A. Nayyar, K.-W.-A. Chee, Analysis of lorawan 1.0 and 1.1 protocols security mechanisms, Sensors 22 (2022) 3717.
[20] O. Pospisil, R. Fujdiak, K. Mikhaylov, H. Ruotsalainen, J. Misurec, Testbed for LoRaWAN security: Design and validation through man-in-the-middle attacks study, Applied Sciences 11 (2021) 7642. doi:10.3390/app11167642.
[21] W.-J. Sung, H.-G. Ahn, J.-B. Kim, S.-G. Choi, Protecting end-device from replay attack on lorawan, in: 2018 20th International conference on advanced communication technology (ICACT), IEEE, 2018, pp. 167–171.
[22] LoRa Alliance, Lorawan specification version 1.0.3, https://resources.lora-alliance.org/document/lorawan-specification-v1-0-3, 2018. Accessed: 2025-11-26.
[23] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, T. Watteyne, Understanding the limits of lorawan, IEEE Communications magazine 55 (2017) 34–40.
[24] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 785–794. doi:10.1145/2939672.2939785.
[25] P. A. A. Resende, A. C. Drummond, A survey of random forest based methods for intrusion detection systems, ACM Computing Surveys (CSUR) 51 (2018) 1–36.
[26] A. Parmar, R. Katariya, V. Patel, A review on random forest: An ensemble classifier, in: International conference on intelligent data communication technologies and internet of things, Springer, 2018, pp. 758–763.
[27] S. A. Zekavat, R. M. Hatcher, Handbook of Position Location-Theory, Practice, and Advances, John Wiley & Sons, Incorporated, 2019.
[28] X. Meng, Scalable simple random sampling and stratified sampling, in: International conference on machine learning, PMLR, 2013, pp. 531–539.
[29] D. Jiawey, A. Espinal, V. S. Padilla, LoRaWAN-Based RSSI-Trilateration Model for Node Location: A Simulation Integrating Flora and Omnet++, Transport and Telecommunication Journal 25 (2024) 218–229.