



PDF Download
3773274.3774261.pdf
16 January 2026
Total Citations: 0
Total Downloads: 68

Latest updates: <https://dl.acm.org/doi/10.1145/3773274.3774261>

RESEARCH-ARTICLE

Targeted and Fully Automated Updates Over LoRaWAN for Energy-Efficient Edge AI

CLÉMENT COURAGEUX-SUDAN, Umeå University, Umea, Västerbotten, Sweden

PAUL TOWNEND, Umeå University, Umea, Västerbotten, Sweden

ATAKAN ARAL, University of Vienna, Vienna, Vienna, Austria

Open Access Support provided by:

Umeå University

University of Vienna

Published: 01 December 2025

[Citation in BibTeX format](#)

UCC '25: 2025 IEEE/ACM 18th
International Conference on Utility and
Cloud Computing
December 1 - 4, 2025
France, France

Conference Sponsors:
SIGARCH

Targeted and Fully Automated Updates Over LoRaWAN for Energy-Efficient Edge AI

Clément Courageux-Sudan
Umeå University
Umeå, Sweden
clement.courageux-sudan@umu.se

Paul Townend
Umeå University
Umeå, Sweden
paul.townend@umu.se

Atakan Aral
Faculty of Computer Science
University of Vienna
Vienna, Austria
atakan.aral@univie.ac.at

Abstract

LoRaWAN Low Power Wide Area Networks have become a standard for remote sensing over large geographical areas with limited access to energy; however, this low-power capability inherently limits communication throughput, posing scalability challenges as network density increases. Edge intelligence (edge AI) can reduce network traffic by deploying lightweight models locally that classify and filter local data before transmission to remote servers. However, models trained with limited local data that run on devices with restricted memory and execution capability necessitate continuous updates to remain accurate within dynamic environments.

Edge intelligence methods, such as federated learning, transfer global model updates to end devices for incremental training and local adaptation. While model transmission is possible using Firmware Update Over The Air (FUOTA) in LoRaWAN, current implementations necessitate manual selection of transmission parameters that highly impact update duration and energy efficiency. This paper proposes a novel approach for updating models over LoRaWAN, addressing distinct and significant challenges compared to typical scenarios over Wi-Fi or cellular networks. First, we fully automate the FUOTA process to propagate model updates to selected machines with heterogeneous communication capabilities. Then, we propose three parameter selection policies to balance the energy consumed by devices and the time taken to update entire networks. An evaluation on a testbed demonstrates the effectiveness of our approach in propagating updates automatically. Large-scale simulations of up to 300 end devices show that our energy-oriented update policy reduces the energy consumption of end devices between 2.2x and 2.7x compared to state-of-the-art baselines.

CCS Concepts

• **Networks** → **Network management; Network performance evaluation**; • **Hardware** → **Power and energy**.

Keywords

LoRaWAN, FUOTA, Energy Efficiency, Edge intelligence

ACM Reference Format:

Clément Courageux-Sudan, Paul Townend, and Atakan Aral. 2025. Targeted and Fully Automated Updates Over LoRaWAN for Energy-Efficient Edge AI. In *2025 IEEE/ACM 18th International Conference on Utility and Cloud*



This work is licensed under a Creative Commons Attribution 4.0 International License. *UCC '25, Nantes, France*

© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2285-1/25/12
<https://doi.org/10.1145/3773274.3774261>

Computing (UCC '25), December 01–04, 2025, Nantes, France. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3773274.3774261>

1 Introduction

As a MAC layer protocol on top of physical LoRa transmissions, LoRaWAN can transmit data over long distances under a limited energy budget. Compared to other Low Power Wide Area Network (LPWAN) technologies such as SigFox and NB-IoT, LoRaWAN offers greater configurability and versatility, making it suitable for a wide range of applications [36]. Despite its flexibility, the densification of IoT infrastructures combined with increasing data volume remains a challenge for a throughput-limited technology [6].

Edge intelligence is emerging as a solution to reduce the amount of data sent over the network, allowing processing tasks to be performed on sensors to classify and detect information worth sending to remote application servers [34]. Existing works propose design and performance evaluations of tiny classifiers running on embedded devices [50]; however, models are tightly tailored to the limited memory and processing constraints. This specialization necessitates continuous training and global updates to stay accurate in environments that are dynamic in time and space [40]. Moreover, bandwidth limitations prevent centralizing large-scale sensor data and initial offline training, resulting in decentralized training with numerous communication rounds [42]. Distributed learning methods like Federated Learning or Split Learning are available to build and disseminate model updates in LPWAN [19]. However, existing works focus more on application performance, hindering the impact of update transmissions over the network.

The LoRa Alliance Firmware Updates Over The Air (FUOTA) [14] permits the distribution of data objects using LoRaWAN. This method can propagate tiny edge AI model updates to a few machines but lacks validation for more dense networks [40]. While FUOTA specifications define update fragmentation and propagation using multicast, they leave many parts of the process in the hands of application developers. The emission of an update requires the manual definition of the update time and the network parameters. Since these parameters impact the range and duration of transmissions, careful selection is necessary for timely and energy-efficient updates [2, 10, 12]. As the network size increases, automating the update process is essential to ensure efficient parameter selection.

Contribution. This work presents a fully automated LoRaWAN FUOTA framework. Updates are organized into rounds, each using different network parameters to target a specific subset of devices. We introduce three device selection policies to optimize update efficiency across rounds, with various trade-offs between energy

consumption, number of update rounds, and channel usage. Compared to previous approaches, our framework does not require any manual intervention and adapts to dynamic changes in the network. Our approach is fully compliant with LoRaWAN and FUOTA specifications. We implemented and evaluated our work with the open-source ChirpStack Network Server. Validation shows substantially more energy- and time-efficient updates compared to state-of-the-art methods. We also perform extensive simulations at scale using ns-3 and assess the accuracy of simulation predictions with testbed validation. The main contributions of this paper are:

- investigating the trade-off between the emission of data from end devices towards remote applications and the execution of local applications to reduce network usage;
- automating the FUOTA process for efficient update propagation using LoRaWAN;
- proposing different policies to select the network parameters to transmit updates, and comparing their impact on energy consumption and transmission duration.

Novelty. To the best of our knowledge, this work demonstrates the first fully automated update framework for large-scale LoRaWAN through multi-round, policy-driven transmissions. Promising empirical results with up to $2.7\times$ energy savings permit preserving device battery life and network resources compared to more basic approaches, while ensuring the update of all machines. Our contribution focuses on Edge AI, but FUOTA automation can benefit all applications necessitating updates, such as anomaly detection systems, embedded operating systems, or sensor networks.

Organization. The remainder of this paper is organized as follows. Section 2 motivates this work and details background material. In Section 3, we present our automation framework and parameter selection policies. We perform experimental validation through real experiments and simulations in Section 4, and discuss future opportunities in Section 5. Section 6 compares our approach to related work for FUOTA and its use for edge AI applications.

2 Background and Context

2.1 Motivation

To highlight the importance of efficient update mechanisms in LoRaWAN, we assess the potential of low-power microcontroller applications to reduce both device energy consumption and network load. Prior studies suggest that local data classification before transmission can significantly enhance the scalability and longevity of IoT infrastructures [41, 42, 61]. However, realizing this potential requires regular and reliable updates of the embedded applications to ensure they remain accurate. To quantify the benefits of this approach, we compare the energy consumption of LoRaWAN packet transmissions with that of tiny model inferences on sensor devices.

The device under test is a *NUCLEO-WL55JC1* [57], with a LoRaWAN radio and an *ARM Cortex M4* MCU, representative of sensor devices with microcontrollers [24]. We use the *X-NUCLEO-LPM01A* power meter [56] for fine-grained measurements of the device’s power usage in different operational states. We average measurements from four devices to limit hardware variability, for an idle device (P_{idle}), the execution of inference on the MCU (P_{active}) and the emission/reception of LoRaWAN packets (P_{tx}/P_{rx}), as listed in

Table 1. We measure the duration D_{inf} and energy consumption E_{inf} of the device to run inference on neural networks of different sizes and report the results in Table 2, where the E_{inf} is directly proportional to D_{inf} . For comparison, Table 3 gives the time-on-air (*ToA*) and the energy E_{tx} to send a packet of 15 bytes on the LoRa channel with varying spreading factors.

In Table 1, local computations necessitate $3.4\times$ less power than transmissions. Additionally, the duration of inference is lower in Table 2 than LoRaWAN time on air in Table 3, particularly under higher spreading factors (i.e., lower data rate DR). Since energy consumption varies with the duration of tasks running on the device, the emission of a LoRaWAN packet consumes significantly more energy than the classification of data, from $39\times$ at the highest data rate (SF7) to $975\times$ with SF12 for a model with 64 parameters.

From these initial experiments, it is evident that filtering locally the sensors’ data can lower the number of network messages, increasing the scalability of the infrastructure, but also improving the battery life of devices by reducing energy consumption. Nevertheless, sensor data is inherently subject to spatiotemporal variations [37, 46]. These variations cannot be handled with models of limited size due to memory constraints, necessitating regular application updates to maintain accuracy [45]. The remainder of this paper concentrates on solutions to automate and efficiently disseminate these updates over LoRaWAN.

Table 1: Power measurements of an end device.

Voltage	P_{idle}	P_{active}	P_{tx}	P_{rx}
3V	13.3mW	21.4mW	73.5mW	22.8mW

Table 2: Average energy consumption for model inference (10^3 samples).

# Params	8	16	32	64
Size (B)	98	322	1,153	4,353
D_{inf} (μ s)	207	548	1,690	5,801
E_{inf} (μ J)	4.4	11.7	36	124

Table 3: Duration and energy consumed for the transmission of a 15-byte packet over LoRaWAN.¹

SF	7	8	9	10	11	12
Bitrate	5.47	3.12	1.76	980	440	250
ToA (ms)	66	123	226	411	905	1,646
E_{tx} (mJ)	4.9	9.1	16.6	30.3	66.5	121.0

¹Source: <https://www.thingsnetwork.org/airtime-calculator>, 10/2025

2.2 LoRaWAN

Based on the LoRa physical layer, LoRaWAN uses Chirp Spread Spectrum (CSS) for data transmission with low power and high resistance to noise and phenomena such as multipath fading [53, 60]. The duration of transmissions varies with the bandwidth (BW) and spreading factor (SF) settings. SF values vary between SF7 and SF12, controlling the chirp rate, where the SF is inversely proportional to the transmission's data rate (DR). Increasing the SF by one provides better transmission range and resistance to interferences, but doubles the time on air, resulting in higher energy consumption [3].

LoRaWAN infrastructures consist of *End Devices (ED)* connecting to remote *Network Servers (NS)* through *Gateways (GW)*. Gateways are packet forwarders for uplink messages from end devices and for downlink packets from the network server. The LoRaWAN MAC layer defines the rules for sending and receiving data [7]. Devices send uplink messages using ALOHA (i.e., sending data without carrier sensing). However, LoRaWAN defines three device classes impacting the capability of devices to receive downlink data [52].

Class-A is the most energy-efficient device class in LoRaWAN. Downlink data reception is only possible during two reception windows following every uplink message, *RX1* and *RX2*. Class-A devices switch off their radio the rest of the time to save energy, but cannot receive data. Class-B makes use of beacons emitted by the gateway for higher downlink capability. In addition to *RX1* and *RX2*, a device properly synchronized with the gateway can schedule downlink receptions during *Ping-Slots (PS)* [52]. Compared to class-A, beacons add an overhead for devices without regular downlink messages. In practice, class-B remains limited due to synchronization issues, clock drifts, and hardware implementation variability [43, 55]. Class-C devices never stop listening to the LoRa channel to ensure low latency and continuous data reception without synchronization, at the cost of higher energy consumption than other classes [2]. Since battery-powered devices need to limit their energy consumption, class-A is the de facto solution for low-power applications. When needed, such as for firmware updates, end devices temporarily switch to classes B or C [38, 43].

In addition to the reduced downlink capability of some device classes and the low data rate of LoRa chirps, LoRaWAN devices must comply with regulatory authorities. For this reason, *Duty Cycle* constraints define the maximum percentage of time a radio can use a LoRa channel to send data [27]. These rules apply to gateways and end devices and depend on geographical regions. For example, the EU868 region for the unlicensed 863-870 MHz band defines a maximum of 1% duty cycle except for one channel with 10% reserved for downlink messages [27]. Duty cycle constraints can delay transmissions and severely limit downlink capability when a single gateway forwards messages to many devices [13, 18, 35]. At scale, duty cycle and bandwidth constraints make the network a bottleneck, in particular for downlink messages to acknowledge requests and transmit updates [18].

2.3 LoRa alliance FUOTA specification

Additionally, the LoRa Alliance proposed Firmware Updates Over The Air (FUOTA) for efficient update transmissions [14]. The FUOTA specification describes how to spread and apply updates securely with LoRaWAN. Despite the name, FUOTA is not limited to firmware

update propagation but applies to the transmission of any data to many devices, including Edge AI model updates. In the following, we focus on the network aspect of FUOTA.

FUOTA relies on three mechanisms to transmit an object. First, the size of the update being higher than the maximum size of LoRaWAN payloads requires creating a fragmentation session and defining the size of fragments [29]. Devices need clock synchronization with the gateway to align their internal clock with the network server [28]. Third, multicast groups are defined so that many devices can receive the same data simultaneously [30].

Thus, a FUOTA round is divided into two steps, illustrated in Figure 1. The first step is the initialization of the FUOTA. It starts when the network administrator decides to transmit an update to a set of machines. When receiving uplink messages from one of the devices to update, the network server acknowledges the message, asking for clock synchronization and the preparation of a fragmentation session. When the device uses class-A, multicast session setup messages are necessary to schedule the opening of a continuous reception window during which the end device switches to class-C to receive fragments. Although switching to class-B to receive the update during *Ping-Slots* is possible, practical adoption remains limited due to the scarcity of software implementations and the technical challenges outlined in Section 2.2. The second step is the reception of the update. During this step, end devices receive update fragments one by one, with redundant data to recover from network loss [29]. Once devices receive all fragments necessary to build the original data, they apply the update and can acknowledge a successful reception to the network server.

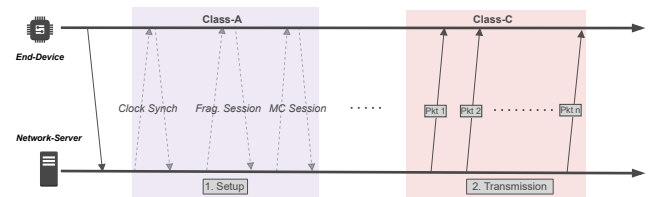


Figure 1: Overview of a FUOTA session [14].

The FUOTA specification defines several mechanisms to prepare, send, and apply end device updates. However, freedom is given to decide the ideal moment to schedule an update and to choose the network transmission parameters [14]. An intelligent and automatic selection of network parameters would significantly reduce the difficulty of manual selection at scale, where the availability and communication range of devices vary dynamically.

3 Automated LoRaWAN FUOTA

3.1 System model

A LoRaWAN infrastructure is a star of stars topology where a gateway *GW* is connected to a set of end devices $ED = \{ED_1, \dots, ED_n\}$. The duration of packet transmissions over the LoRa channel (*Time-On-Air*) and device energy consumption depend on the network parameters of each packet: the bandwidth (*BW*), spreading factor (*SF*), and payload size (*PL*) [54]. LoRa specifications [53, 54] define

the duration to transmit a packet's symbol T_{sym} as:

$$T_{sym} = \frac{2^{SF}}{BW}, \quad (1)$$

Where the number of symbols in a packet is:

$$nbSym = nbPr + 8 + \text{Max} \left(\text{Ceil} \left[\frac{8PL - 4SF + 28 + 16}{4SF} \right] (CR + 4), 0 \right), \quad (2)$$

where $nbPr$ is the size of the preamble of the packet and CR is the coding rate for error correction [54]. Assuming $nbPr$ and CR are predefined values, the time-on-air of a packet is equal to $D_{pkt} = T_{sym} * nbSym$. This duration varies with the spreading factor and the bandwidth selected for the transmission.

Equations 1 and 2 also serve to estimate the energy consumption of an end device. The LoRa interface switches between operating states to run different operations, each of them with a different power value: P_{idle} when the device is idle, $P_{standby}$ when the device listens to the LoRa channel, P_{Tx} and P_{Rx} when actively sending or receiving data. As a consequence, the energy consumption of an end device is the sum over all states of the power times the duration spent in that state,

$$E(ED_i) = \sum_{s \in \text{states}} P_s(ED_i) \times Dur_s(ED_i) \quad (3)$$

During the reception of FUOTA fragments using class-C as defined in Section 2.3, an end device alternates between receiving data P_{rx} and listening to the channel between fragments $P_{standby}$. P_{rx} and $P_{standby}$ are closely related values, as measured in [11].

$$E_{FUOTA}(ED_i) = P_{rx} \times Dur_{rx}(ED_i) + P_{standby} \times Dur_{standby}(ED_i) \quad (4)$$

A smaller update size reduces the number of fragments, decreasing Dur_{rx} and thus the energy consumption of devices [39]. Similarly, a higher duty cycle reduces the waiting time between fragments $Dur_{standby}$. As the spreading factor increases, so does the area covered by transmissions, at the cost of higher *time-on-air* and energy consumption, as shown in Table 3. Since end devices are highly heterogeneous in LoRaWAN infrastructures, a fixed configuration cannot be used to transmit the update to all devices with sufficient range while minimizing energy and *time-on-air*. In addition, battery depletion and mechanisms such as *adaptive data rate* (ADR) lead to dynamic changes in the network topology and the configuration of devices. For this reason, our approach consists of a FUOTA transmission with multiple propagation rounds using parameters that reduce the energy and duration of transmissions for selected nodes during each round.

3.2 Update agents

The automated FUOTA process relies on the cooperation between a network server application collecting global information and a client application running on end devices for local decisions. In addition to our applications, we rely on existing FUOTA software implementations to handle data fragmentation, create multicast groups, and apply the update on the device [14], as summarized in Figure 2. This design choice facilitates seamless integration with the wide range of available hardware platforms. The rest of this section

describes the client and server applications in detail, followed by the device selection policies to optimize different metrics.

The *Automation client* maintains information about the applications running on the local device, for example, an Edge AI classifier. We define a tuple $\langle appid, version, ts_{update} \rangle$, where $appid$ is a unique identifier for the application running on the device, $version$ the version number of the current application, and ts_{update} is the timestamp of the last update received from the network server. This tuple is used in the client update-loop procedure in Algorithm 1. To prepare for updates, devices send the application tuple to the network server with a period $period$ between consecutive messages. Uplink frames are sent until the server application is ready to schedule an update with the device and acknowledges the request on L.4. Following Figure 1, the next step is to synchronize the device clock, initialize the multicast, and the fragmentation session with the server application. Using the network parameters given by the server, end devices estimate on L.6 the energy consumption to receive the update of nbF fragments with parameters $rxParams$, using Equation 4. The device decides whether or not to prepare for the update reception based on this energy estimation, between L.7-11. If receiving the update consumes more than η multiplied by the battery charge, the client postpones the update to a later time, when more energy is available.

Algorithm 1: Client update loop

```

1 Input: period, AppData
2 while true do
3   ack=Uplink(appid, version, tsupdate);
4   if ack! =  $\emptyset$  then
5     nbF,rxParams=ReceiveMulticastInfo();
6     cost=estimateUpdateCost(nbF,rxParams);
7     if cost <  $\eta * GetEnergy()$  then
8       | ScheduleUpdate(nbFrag, rxParams);
9     else
10      | Sleep(period);
11    end
12  else
13    | Sleep(period);
14  end
15 end

```

The *Automation server* processes end devices' data received through the gateway and schedules updates. The *Pooling-DB*, as shown in Figure 2, stores information about updates and device requests. Each application *Objid* is associated with the *path* to the object, the latest *version* stored on the server, the *minInterval* between consecutive updates, and a map of device requests. For each ED_i requesting an update, the database stores the information included in the application tuple along with the network parameters used by ED_i to reach the gateway. Even when using ADR, storing the latest network parameters used by end devices ensures the server is able to reach a node unless its DR has decreased since the last received message. The information stored in the database is essential to select the devices and parameters to use when scheduling an update round in Algorithm 2. After receiving several requests

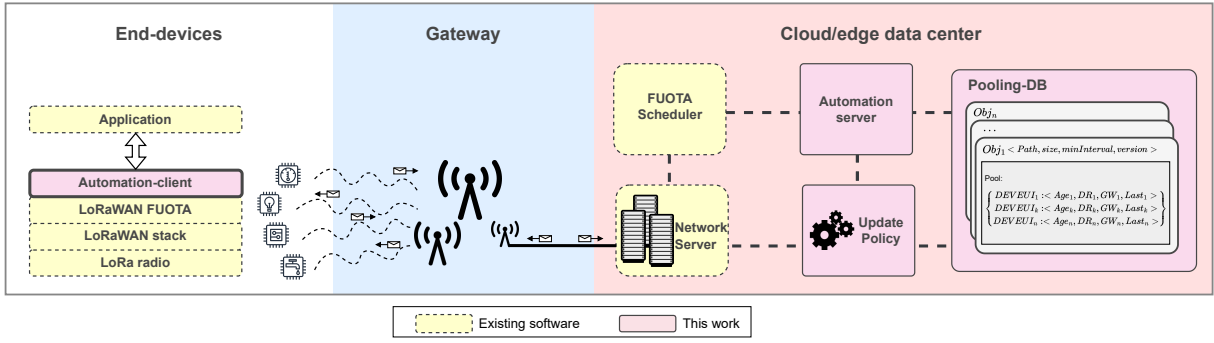


Figure 2: Integration of our update client and server within the existing LoRaWAN network middleware.

and ensuring a minimum duration $minInterval$ between consecutive transmissions on L.5, the server schedules a new round. The server selects the network configuration used for sending update fragments on L.7. As seen in Section 2.3, this choice is critical since frequencies with low duty cycle increase inter-fragment duration, while high data rate packets can't be received with a high range. Thus, a specific selection of network parameters can only reach a subset of the device pool during each round.

Algorithm 2: Server update loop

```

1 Input: upReq, lastUpdate
2 if upReq.version < db.version then
3   | pool = pool ∪ upReq;
4 end
5 if Time() - lastUpdate > minInterval then
6   | // schedule a new round
7   | TxParams = SelectNetworkParams(pool);
8   | McInfo = SetupFUOTA(TxParams);
9   | Acknowledge(TxParams, McInfo);
10 end

```

Different strategies for parameter selection can optimize different metrics: the duration to transmit the update to all devices, the energy consumed for receptions, or using as little time-on-air as possible to avoid overlapping with other applications. No configuration can optimize all metrics simultaneously, resulting in a balance depending on application requirements. To keep our approach modular, we propose three different policies. Additional multi-objective policies can easily be integrated into the framework.

3.3 Network parameters selection policies

All the policies use the data from the *Pooling-DB* to select the frequency, data rate, and nodes to include in a FUOTA round. By default, we always choose the highest bandwidth and the frequency with the highest available duty cycle to reduce inter-fragment durations. In the EU868 band used during the evaluation of this work, fragments can be sent with a 10% duty cycle at 869.525 MHz.

a) All-nodes: This policy minimizes the number of update rounds by sending the update with the spreading factor providing the longest range (i.e., lowest data rate) among machines in the pool.

Unlike the conventional FUOTA approach, which always uses the highest spreading factor SF12, we select the highest SF among the nodes to update. A similar approach was adopted in previous work [33], with more efficient updates than always using SF12. This policy decreases the number of rounds at the cost of higher reception duration for devices with high data rates, increasing their energy consumption. The *all-nodes* policy is advised for updates within limited deadlines where devices have reliable energy sources.

b) Energy-first: This policy optimizes the energy consumption of update receptions. We choose the lowest spreading factor (i.e., the highest data rate) from the device pool to reduce the *time-on-air* and the energy consumption of end devices. For each round, we only update the machines from the pool with the highest DR, since other devices might not be able to receive fragments successfully. While all devices receive fragments at maximum speed, this approach requires several rounds for heterogeneous networks. Using multiple rounds with optimal transmission settings for each device lowers average battery utilization during reception, but requires more uplink messages and overall update duration.

c) Threshold-based: As a trade-off between the high energy consumption of *all-nodes* and the high number of rounds from *energy-first*, we propose an alternative approach. Depending on the requirement of an application update, the administrator of the update server defines a threshold corresponding to the maximum number of rounds or the maximum duration to propagate an update. Before reaching the threshold, the server schedules rounds with the highest data rates available to ensure low energy. When exceeding the threshold, the update server sends the update to as many nodes as possible, even at the cost of higher energy consumption.

These policies define a single data rate and frequency for all network gateways used during multicast. Additional policies varying parameters between gateways can also be considered.

3.4 Implementation

We implement our work for evaluation on a testbed. The *automation client* implementation is made for the same Nucleo-WL55JC end devices as used in Section 2.1. Timer interrupts trigger the client

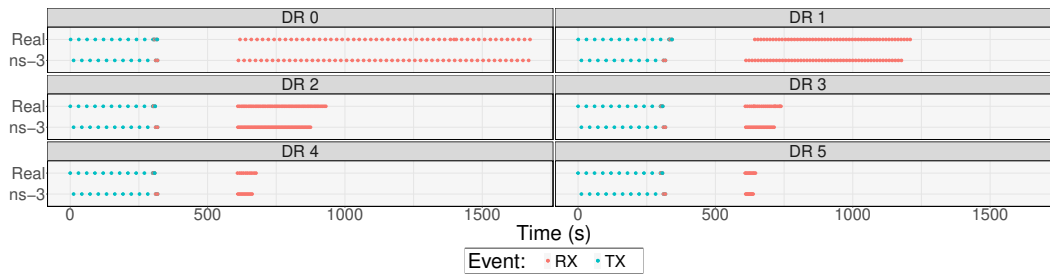


Figure 3: Network events for a FUOTA transmission to a single device using data rates DR0 to DR5. Each point is the timestamp of one network event on the real device (top) and in simulation (bottom).

loop to send periodic uplink messages. The LoRaWAN and FUOTA implementation from STMicroelectronics is not modified, meaning the application can be easily transposed to other devices.

To run the network server in a controlled environment, we implement the *automation server* application with ChirpStack [17]. Compared with alternative network servers, ChirpStack is open-source and can be used to create private LoRaWAN testbeds. The application receives requests through ChirpStack’s MQTT interface. The creation of multicast groups, fragmentation sessions, and clock synchronization is done by the ChirpStack FUOTA server [16].

We also implement our approach in the ns-3 simulator for evaluation at scale. Ns-3 is a popular framework for network simulation with packet-level accuracy. Our implementation uses the ns-3 module for LoRaWAN proposed in [31]. This module simulates class-A devices based on official specifications and time-on-air formulas [52, 54]. To model physical layer transmissions, ns-3 provides propagation models such as Okumura-Hata [22] that we use for our simulations. This model is realistic for LoRaWAN studies in urban and rural environments [5, 20]. We extend the module to implement the parts of the FUOTA specification necessary for our tool: to create multicast groups, fragmentation sessions, and to switch devices to class-C during the FUOTA emissions. We extend the MAC layer of class-A devices to add continuous reception windows used when receiving the update fragments. In addition to predictions of communication duration, this module simulates the energy consumption of end devices so that we can compare the energy efficiency of different policies. The code is available online².

4 Evaluation

4.1 Experimental setup

We evaluate our method with experiments on a real hardware testbed of four LoRaWAN end devices. At scale, we run simulations to update up to 300 machines. We do not consider multiple gateways in this study. For testbed experiments, the end devices are connected to a RAK7246G gateway³. The gateway runs a packet forwarder between end devices and our private local ChirpStack network server. Some experiments use the LPM01A power meter for energy measurements. In this study, $\eta = 0$ in Algorithm 1 so that update receptions are never postponed due to energy limitations since devices are not battery-powered. Update size varies between 1 and

6 kilobytes based on related work [19, 39, 49] and on the size of the model in Table 2.

For both testbed and simulation experiments, the *automation client* runs on end devices, while the *automation server* runs on the same machine as the network server. The evaluation is done in three steps. First, we assess the validity of FUOTA simulations compared to testbed measurements. Second, we analyze the behavior of each policy on a small scale. Third, we evaluate the scalability of our contribution compared to *naive-FUOTA*, an automated parameter selection policy that always uses the lowest data rate *DR0* as proposed in previous work [33]. Our measurements include:

Elapsed time: The duration between the beginning of the experiment and the time at which all devices fully receive the update.

Total number of update rounds: Lower values mean the gateway retransmits the same object fewer times, reducing network usage.

Average number of update rounds: This value indicates, on average, how long end devices execute an out-of-date application.

Dynamic energy consumption: Value obtained when removing idle power from energy computation. Lower values indicate a more energy-efficient *update-client* process and data reception.

Update reception duration: Higher values indicate end devices are less available to transmit messages due to update transmissions.

Reproducibility: The code used to generate the results of this paper is available at <https://github.com/klementc/fuotaa-reprod>.

4.2 Accuracy of simulations

Figure 3 shows the timestamps of uplink (TX) and downlink (RX) network events for one device receiving a 2 kilobyte update with different data rates. Results are shown for executions on real devices and in simulations. Each configuration is executed 5 times. Larger objects give the same observations, with more important update reception durations due to the higher number of fragments.

We observe the two steps described in Section 2.3. The device first sends its application tuple with *period=30s* until receiving an acknowledgement after *minInterval=300s* when the update transmission is scheduled. Second, the device sleeps until fragment reception at 600 seconds with a continuous class-C reception window.

Simulation and testbed results in Figure 3 show a variation in reception duration depending on the data rate. With DR5, the object is divided into 9 fragments, necessitating ≈ 40 seconds to be received. With DR0, the same object has 52 fragments transmitted in ≈ 16 minutes. In addition to increasing the number of fragments caused

²<https://github.com/klementc/lorawan>, last access 10/2025

³<https://docs.rakwireless.com/product-categories/wisgate/rak7246g/> 10/2025

by the payload size reduction of low data rates, the increased *time-on-air* requires longer inter-packet intervals to respect duty cycle restrictions. While a fragment is sent every 5 seconds with DR5, fragments using DR0 are spaced by 20 seconds.

Simulation results are well aligned with real-world measurements, with a maximum relative error of 10% for inter-packet arrival times at high data rates. This error is attributed to differences in handling packet queuing in ChirpStack. In ns-3, fragment transmissions occur immediately once duty cycle restrictions permit, whereas ChirpStack relies on a scheduler that checks for transmission opportunities every 500 ms, introducing additional delays.

We measure the energy consumption of the device in Figure 4. The ns-3 power model is calibrated with the empirical energy measurements from Table 1. In Figure 4, the total energy consumption increases by more than 4× between DR5 and DR0, from 7.6J (7.3J in ns-3) at DR5 to 33.3J (30.6J in ns-3) with DR0. This gap shows the importance of cautious DR selection to reduce energy usage. We observe up to 2.7J between real and simulated energy values. Clock drifts cause this error in real experiments with low data rates, but do not impact our conclusions.

Overall, simulations accurately estimate event timing, update duration, and device energy consumption during updates. As illustrated in Figures 3 and 4, updates using low data rates significantly increase transmission duration and energy consumption.

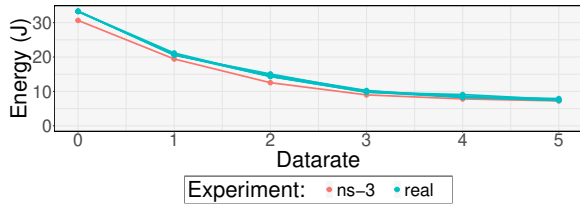


Figure 4: Energy consumption of the end device receiving the update with different data rates, 5 samples.

4.3 Comparison of automation policies

We compare the network parameter selection policies with four end devices assigned different data rates receiving a 2kB update. The baseline *naive-FUOTA* uses the lowest data rate DR0. We compare it to the *all-nodes*, *energy-first*, and *threshold* policies with thresholds of 2 and 3 rounds as defined in Section 3.3. Since the results are very similar between the simulation and testbed, we only report results from real experiments. Simulation results are available online⁴.

The duration of the polling and reception phases varies across policies, as summarized in Table 4. In both the *naive-FUOTA* and *all-nodes* approaches, all devices are updated in a single round since network parameters are selected to maximize the number of updated devices. Compared to *naive-FUOTA*, which uses DR0 for all transmissions, the *all-nodes* policy employs the lowest data rate among the devices (DR2 in this experiment), reducing the overall update duration. The other policies schedule multiple rounds with different data rates to consider device heterogeneity. *Energy-first* has the longest elapsed time, with four rounds to update all nodes.

⁴https://github.com/klementc/fuotaa-reprod/blob/main/notebook_4ed.ipynb

At the same time, it leads to the shortest individual updates. As expected, *Threshold* policies give intermediate results.

Despite higher elapsed time, *energy-first* has the lowest dynamic energy consumption E_{dyn} . In all cases, the devices' energy consumption is mostly due to fragment reception. The overhead of the higher number of uplink requests for *energy-first* remains lower than the gains of higher data rate reception.

To summarize, all policies overcome *naive-FUOTA*. *Energy-first* is suitable for environments with high energy limitations without deadline requirements, while *all-nodes* reaches all machines in one round. *Threshold* balances between elapsed time and energy.

Table 4: Comparison of the energy consumption and update duration between transmission policies.

Policy	# Rounds	Rx dur (s)	E_{dyn} (J)	Elapsed dur (s)
Naive-FUOTA	1	1057s	24.4	1,669
All-nodes	1	264s	6.2	517
Threshold_2	2	205s	5.1	1,785
Threshold_3	3	151s	4.05	2,734
Energy-first	4	111s	3.25	3,645

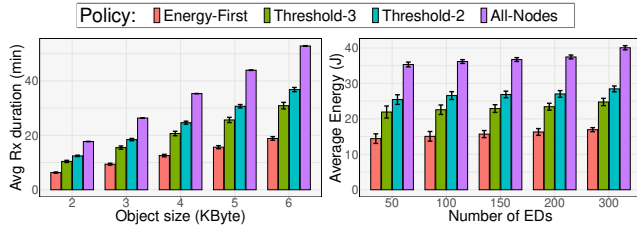
4.4 Scalability evaluation

We assess the scalability of our approach through simulations of updates in dense networks. Given the strong correspondence between the prior testbed and simulation results, we consider the simulation output highly reliable. We vary the number of end devices from 50 to 300, randomly distributed within a 19 km radius of the gateway. This setup ensures that devices utilize the full range of data rates even without ADR. We do not report results for *naive-FUOTA* because some devices consistently use DR0 across all configurations. Thus, the performance of the *naive-FUOTA* and *all-nodes* policies is similar. Scenarios are executed 10 times with different random seeds to introduce variation in packet loss.

Figure 5a shows the impact of update size on the average reception duration over all 300 devices. As expected, larger objects result in longer average reception durations across all policies, due to the increased number of fragments. In contrast, the *energy-first* policy achieves significantly lower average reception durations (from 6 to 19 minutes) by updating devices with their highest supported data rate. *Threshold* policies have a moderate increase since they use high data rates during the first rounds before emitting with lower data rates to speed up the update later.

Figure 5b presents the average dynamic energy consumption of devices receiving a 3kB update. Across all network sizes (50 to 300 end devices), the *energy-first* policy demonstrates superior energy efficiency, consuming 2.2× to 2.7× less energy than the *all-nodes* approach. Furthermore, the relative energy savings achieved by *energy-first* increase with the size of the transmitted object. The better values of *energy-first* and *threshold* come from nodes with high data rate updated in their own round, while devices with low data rates have the same update duration as using *all-nodes*.

While energy consumption is correlated with the duration of the continuous reception window, we observe an increase in energy as the number of end devices grows. This is primarily due to a higher number of devices missing initial update opportunities, requiring



(a) Mean transmission time vary-ing update sizes (300 devices). (b) Mean energy consumption of end devices (3 kB update).

Figure 5: Transmission time and energy consumption comparison of policies. The error bar shows the 95% confidence interval from ten executions.

participation in subsequent rounds. Increasing the advertisement window beyond the 30 minutes used for this experiment allows more devices to complete their session setup on the first attempt. Also, a higher number of devices increases the packet loss rate due to network congestion. This necessitates retransmissions, increasing energy consumption. Low Duty cycles exacerbate the problem, as the gateway may be unable to send acknowledgments.

When an update reception is missed, devices keep sending messages to the network server during the initialization phase. Figure 6 shows the ratio between the number of messages sent and received with 300 devices (application messages and acknowledgements). This ratio depends on two parameters. First, more rounds imply more uplink requests. For this reason, *energy-first* has the highest ratio, while *all-nodes* ensures low overhead. Increasing the duration between uplink messages reduces this value. Second, larger updates necessitate more fragments, decreasing the overhead of the initialization phase. Thus, single-packet updates for a few devices can be sent efficiently without our approach, removing the initialization time and energy overhead.

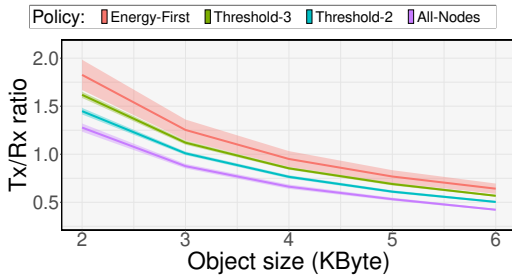


Figure 6: Ratio between the number of messages sent and received with each policy for 300 EDs. The ribbon shows the 95% confidence interval from 10 executions.

4.5 Continuous updates over time

Previous experiments evaluate a single update in a network. In this experiment, we simulate updates over an entire week, where a global edge AI model is updated based on samples received from devices. The updated model is sent daily to devices for improved accuracy when processing local data, as used in [19, 40].

Table 5 shows the average and maximum number of daily rounds to update a model on all devices. The higher values observed on average and maximum for *energy-first* show that devices can spend a long time running an outdated application, reducing the accuracy of predictions. In comparison, *all-nodes* significantly reduces the average and maximum number of rounds. If application updates are frequent, less energy-efficient policies might be necessary.

Table 5: Number of rounds to receive daily application updates (one week of simulated time).

# Nodes	100		150		200	
	Avg.	Max.	Avg.	Max.	Avg.	Max.
All-nodes	1.61	3	2.21	4	2.90	6
Threshold_2	2.69	5	2.98	6	3.25	6
Threshold_3	3.12	5	3.30	6	3.48	7
Energy-first	3.55	7	3.64	8	3.80	9

5 Improvements

We evaluated the time and energy efficiency of our update framework. Additional work could be done, such as:

Multi-gateways consideration: We do not consider multi-gateway setups. Integrating existing work to reduce duty cycle limitations with multiple gateways to transmit updates would be possible [2]. While we do not do it in this work, our framework can be extended with multi-gateway policies.

FUOTA using class-B: Switching from class-C to class-B can highly reduce the energy consumption of devices receiving fragments [2]. Using class-B multicast when mature software is available could help further reduce the energy consumption of updates.

Mobility: In our evaluation, devices are stationary and do not use adaptive data rate. Nevertheless, our continuous monitoring of the devices' data rate is adapted to dynamic configurations.

Training process: We study update transmission and compare inference to network communication, but not the training process. Different learning patterns exist, like federated learning and sensor data sampling for centralized learning. Future work should compare the network and processing overheads of different approaches.

6 Related Work

6.1 FUOTA performance

Recent surveys explored various strategies to improve the efficiency of FUOTA over LoRaWAN [9, 44]. The authors of [2, 15] show that transmitting even small updates may take several hours, especially with low data rates. These findings motivated extensive efforts to reduce the size of updates. Delta update frameworks such as FLoRa [58, 59] compress firmware by transmitting differential update files instead of full images to achieve 3× size reductions. However, delta-updates require additional on-device processing and are not adapted to update multiple devices running different versions of the same application, as can happen in dense deployments. Modular firmware designs [39] enable individual software module updates to reduce update size and device downtime.

Regardless of FUOTA, update efficiency is heavily influenced by LoRaWAN parameters. Studies such as [23, 48] demonstrate that

optimizing data rates and duty cycle constraints yields significant energy savings. Despite this, FUOTA implementations generally rely on fixed configurations or manual tuning [8, 40, 59], with little support for dynamic adaptation based on network conditions.

To overcome duty cycle restrictions and reach 100% channel utilization, sending updates with multiple gateways in parallel is proposed in [2]. Alternatively, the authors of [4, 15] propose clustering techniques to optimize the placement of gateways for transmissions at high data rates. In this way, updates can be sent at maximum speed to all nodes of a dense LoRaWAN infrastructure. While multiple gateways help mitigate duty cycle limitations and increase data rates, this cannot always apply to real environments, particularly when multiple technologies share the same frequency bands [1]. Finally, multi-gateway placement optimization is efficient for static infrastructures but lacks adaptability.

The *Divide and Upgrade (DAU)* method proposed in [33] for more energy and time-efficient updates introduces policies similar to this work. DAU emits the same update multiple times with different network parameters, but lacks a full automation of the update process. As a consequence, DAU cannot be compared to our work. While our work does not necessitate any modification of FUOTA mechanisms, DAU's negative acknowledgement to recover lost fragments necessitates heavy refactoring of existing FUOTA implementations for various hardware platforms. Additionally, our framework can easily be extended to add new transmission policies.

6.2 Update and spread of edge AI models

Recent work shows the limitations of LPWAN to transmit monitored data in remote applications without considerable loss, even starting from tens to hundreds of devices [26, 32].

The limited network capability of LoRaWAN infrastructures is a major motivation for edge AI applications. A performance evaluation of FUOTA upgrades for a tinyML agricultural monitoring application is available in [40], showing the need for energy-efficient update transmissions through single device measurements and simulations. In addition to update propagation, decentralized learning methods are necessary to update tiny models. The authors of [19] use federated learning to train and spread model updates in a LoRa mesh network. In this way, a model continuously improves through local training on sensors. However, evaluation is limited to a network with three machines and might not scale in dense use cases. TinyOL is an approach to update applications through local observations [47]. However, this approach, and other alternatives like [25], consider the local adaptation of the application and not the distribution of updates between nodes.

While local edge AI executions are promising to reduce network load, the trade-off between network usage and local processing overhead is not well known. Some studies assess the performance, accuracy, and energy consumption of lightweight models executed on microcontrollers. For example, [50, 51] give an overview of existing frameworks and their performance for inference. Recent contributions highlight poor consideration of state-of-the-art works for the training and propagation of models [21, 23, 40].

To summarize, edge processing can help reduce network usage. Existing update mechanisms using FUOTA rely on the network administrator to select the machines to update and the network

parameters. Furthermore, the energy overhead of local processing compared to network transmissions is difficult to estimate. In this work, we automated the FUOTA process and proposed parameter selection policies balancing update duration and energy consumption. We evaluated our approach at scale and estimated the trade-offs between local executions and data transmission.

7 Conclusion

Efficient distribution of updates in dense and heterogeneous LoRaWAN infrastructures is necessary for many applications, such as recent edge AI models. This paper brings a novel approach to solve this problem - rather than previous approaches based on the manual selection of update time and network transmission parameters, we automate the FUOTA process to spread updates in rounds. Our approach relies on the cooperation between lightweight update clients running on end devices and an update server application connected to the LoRaWAN network server. The proposed automation policies can balance the energy consumption of nodes, the number of update rounds, and the overall update propagation time. Experimental results on a testbed show time and energy-efficient updates using our approach compared to the traditional FUOTA using low data rates. At scale, we compared propagation policies using ns-3 simulations. While some policies can spread critical updates in a limited time, our energy-oriented approach can propagate updates more slowly but preserve the battery life of end devices.

This work highlights the capability of model updates at scale using LoRaWAN, where hundreds of model inferences are more energy efficient than a single network transmission. More generally, automated update propagation can benefit all applications where the network's size and heterogeneity complicate manual processes.

Acknowledgments

Financial support has been provided by the Kempe Foundation under contract number 3161, CHIST-ERA-22-SPiDDS-07 (TROCI Project) and Austrian Science Fund (FWF) [10.55776/I6647].

References

- [1] Kerima Saleh Abakar, Ismail Bennis, Abdelhafid Abouaissa, and Pascal Lorenz. 2022. A multi-gateway behaviour study for traffic-oriented lorawan deployment. *Future Internet* 14, 11 (2022), 312.
- [2] Khaled Abdelfadeel, Tom Farrell, David McDonald, and Dirk Pesch. 2020. How to make firmware updates over lorawan possible. In *2020 Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*. IEEE, 16–25.
- [3] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. 2017. Understanding the limits of LoRaWAN. *IEEE Communications magazine* 55, 9 (2017), 34–40.
- [4] Sabtain Ahmad, Halit Uyanik, Tolga Ovatman, Mehmet Tahir Sandikkaya, Vincenzo De Maio, Ivona Brandić, and Atakan Aral. 2023. Sustainable environmental monitoring via energy and information efficient multi-node placement. *IEEE Internet of Things Journal* (2023).
- [5] Lameya Aldhaheeri, Noor Alshehhi, Irfana Ilyas Jameela Manzil, Ruhul Amin Khalil, Shumaila Javaid, Nasir Saeed, and Mohamed-Slim Alouini. 2024. LoRa Communication for Agriculture 4.0: Opportunities, Challenges, and Future Directions. *IEEE Internet of Things Journal* (2024).
- [6] Mohammed Alenezi, Kok Keong Chai, Yue Chen, and Shihab Jimaa. 2020. Ultra-dense LoRaWAN: Reviews and challenges. *IET Communications* 14, 9 (2020), 1361–1371.
- [7] LoRa Alliance. 2022. TS001-1.0. 4 LoRaWAN® L2 1.0. 4 Specification. 10 (2022).
- [8] Anika Anwar, Nilesch Chakraborty, and Mohammad Zulkernine. 2024. Secure OTA Software Updates for Connected Vehicles Using LoRaWAN and Blockchain. In *2024 IEEE 24th International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, 1067–1076.

- [9] Jan Bauwens, Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, and Eli De Poorter. 2020. Over-the-air software updates in the internet of things: An overview of key principles. *IEEE Communications Magazine* 58, 2 (2020), 35–41.
- [10] Norhane Benkahla, Hajer Tounsi, Ye-Qiong Song, and Mounir Frikha. 2021. Review and experimental evaluation of ADR enhancements for LoRaWAN networks. *Telecommunication Systems* 77, 1 (2021), 1–22.
- [11] Martina Capuzzo, Carmen Delgado, Jeroen Famaey, and Andrea Zanella. 2021. An ns-3 implementation of a battery-less node for energy-harvesting internet of things. In *Proceedings of the 2021 Workshop on ns-3*. 57–64.
- [12] Martina Capuzzo, Carmen Delgado, Ashish Kumar Sultania, Jeroen Famaey, and Andrea Zanella. 2021. Enabling green IoT: Energy-aware communication protocols for battery-less LoRaWAN devices. In *Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. 95–98.
- [13] Martina Capuzzo, Davide Magrin, and Andrea Zanella. 2018. Confirmed traffic in LoRaWAN: Pitfalls and countermeasures. In *2018 17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*. IEEE, 1–7.
- [14] Julien Catalano. 2021. LoRaWAN firmware update over-the-air (FUOTA). *Journal of ICT Standardization* 9, 1 (2021), 21–34.
- [15] Christia Charilaou, Spyros Lavdas, Ala Khalifeh, Vasos Vassiliou, and Zinon Zinonos. 2021. Firmware update using multiple gateways in LoRaWAN networks. *Sensors* 21, 19 (2021), 6488.
- [16] ChirpStack. [n. d.]. ChirpStack FUOTA Server. <https://github.com/chirpstack/chirpstack-fuota-server>. Last accessed: 10/2025.
- [17] ChirpStack ChirpStack. [n. d.]. Open-source LoRaWAN Network Server stack. <https://www.chirpstack.io/>. Last accessed: 10/2025.
- [18] Valentina Di Vincenzo, Martin Heusse, and Bernard Tourancheau. 2019. Improving downlink scalability in LoRaWAN. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.
- [19] Nil Llisterra Giménez, Joan Miquel Solé, and Felix Freitag. 2023. Embedded federated learning over a LoRa mesh network. *Pervasive and Mobile Computing* 93 (2023), 101819.
- [20] Eugen Harinda, Salaheddin Hosseinzadeh, Hadi Larijani, and Ryan M Gibson. 2019. Comparative performance analysis of empirical propagation models for lorawan 868mhz in an urban scenario. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, 154–159.
- [21] Yohei Hasegawa and Kazuya Suzuki. 2019. A multi-user ack-aggregation method for large-scale reliable lorawan service. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.
- [22] Masaharu Hata. 1980. Empirical formula for propagation loss in land mobile radio services. *IEEE transactions on Vehicular Technology* 29, 3 (1980), 317–325.
- [23] Mohammed Jouhari, Nasir Saeed, Mohamed-Slim Alouini, and El Mehdi Amhoud. 2023. A survey on scalable LoRaWAN for massive IoT: Recent advances, potentials, and challenges. *IEEE Communications Surveys & Tutorials* 25, 3 (2023), 1841–1876.
- [24] Harry Koutsourelakis, George Kornaros, and Marcello Coppola. 2023. Energy Efficient Deep-Edge Computing through Hardware Machine Learning. In *2023 IEEE 9th World Forum on Internet of Things (WF-IoT)*. IEEE, 1–7.
- [25] Jisu Kwon and Daejin Park. 2021. Hardware/software co-design for tinyml voice-recognition application on resource frugal Edge Devices. *Applied Sciences* 11, 22 (2021), 11073.
- [26] Isadora Rezende Lopes, Paulo Rodolfo da Silva Leite Coelho, Rafael Pasquini, and Rodrigo Sanches Miani. 2025. Evaluating the Performance of LoRa Networks: A Study on Disaster Monitoring Scenarios. *IoT* 6, 1 (2025), 14.
- [27] LoRa Alliance. 2020. *LoRaWAN Regional Parameters v1.0.3 rev a0*. LoRa Alliance.
- [28] LoRa Alliance. 2022. *TS003-2.0.0 Application Layer Clock Synchronization*. LoRa Alliance.
- [29] LoRa Alliance. 2022. *TS004-2.0.0 Fragmented Data Block Transport*. LoRa Alliance.
- [30] LoRa Alliance. 2022. *TS005-2.0.0 Remote Multicast Setup*. LoRa Alliance.
- [31] Davide Magrin, Martina Capuzzo, and Andrea Zanella. 2019. A thorough study of LoRaWAN performance under different parameter settings. *IEEE Internet of Things Journal* 7, 1 (2019), 116–127.
- [32] Mandeeep Malik, Ashwin Kothari, and Rashmi Pandhare. 2024. Scalability Analysis of LoRa and Sigfox in Congested Environment and Calculation of Optimum Number of Nodes. *Sensors* 24, 20 (2024), 6673.
- [33] Wenliang Mao, Zhiwei Zhao, Mengyu Kang, Rong Cong, Geyong Min, Zheng Chang, and Xiong Wang. 2023. Reliable and Energy-Efficient Reprogramming for Smart LoRaWAN. In *2023 IEEE Smart World Congress (SWC)*. IEEE, 1–8.
- [34] Yuyi Mao, Xianghao Yu, Kaibin Huang, Ying-Jun Angela Zhang, and Jun Zhang. 2024. Green edge AI: A contemporary survey. *Proc. IEEE* (2024).
- [35] Jaco Morné Marais, Adnan M Abu-Mahfouz, and Gerhard P Hancke. 2022. Improving the sustainability of confirmed traffic in LoRaWANs through an adaptive congestion scheme. *IEEE Sensors Journal* 23, 2 (2022), 1660–1670.
- [36] Jaco M Marais, Reza Malekian, and Adnan M Abu-Mahfouz. 2017. LoRa and LoRaWAN testbeds: A review. *2017 Ieee Africon* (2017), 1496–1501.
- [37] Asif Iqbal Middy, Sarbani Roy, and Rituparna Das. 2023. Spatiotemporal variability analysis of air pollution data from IoT based participatory sensing. *Journal of Ambient Intelligence and Humanized Computing* 14, 6 (2023), 6719–6734.
- [38] Md Najmul Mowla, Neazmul Mowla, AFM Shahen Shah, Khaled Rabie, and Thokozani Shongwe. 2023. Internet of things and wireless sensor networks for smart agriculture applications-a survey. *IEEE Access* (2023).
- [39] Huy Dat Nguyen, Nicolas Le Sommer, and Yves Mahéo. 2024. Over-the-Air Firmware Update in LoRaWAN Networks: A New Module-based Approach. *Procedia Computer Science* 241 (2024), 154–161.
- [40] Chollet Nicolas, Bouchemal Naila, and Ramdane-Cherif Amar. 2022. Energy efficient firmware over the air update for tinyml models in lorawan agricultural networks. In *2022 32nd International Telecommunication Networks and Applications Conference (ITNAC)*. IEEE, 21–27.
- [41] Soohyun Park, Soyi Jung, Haemin Lee, Joongheon Kim, and Jae-Hyun Kim. 2021. Large-scale water quality prediction using federated sensing and learning: A case study with real-world sensing big-data. *Sensors* 21, 4 (2021), 1462.
- [42] Ella Peltonen, Ijaz Ahmad, Atakan Aral, Michele Capobianco, Aaron Yi Ding, Felipe Gil-Castineira, Ekaterina Gilman, Erkki Harjula, Marko Jurmu, Teemu Karvonen, et al. 2022. The many faces of edge intelligence. *IEEE Access* 10 (2022), 104769–104782.
- [43] Ales Povalac, Jan Kral, Holger Arthaber, Ondrej Kolar, and Marek Novak. 2023. Exploring LoRaWAN Traffic: In-Depth Analysis of IoT Network Communications. *Sensors* 23, 17 (2023), 7333.
- [44] Mompoloki Pule and Adnan M Abu-Mahfouz. 2019. Firmware updates over the air mechanisms for low power wide area networks: A review. In *2019 International Multidisciplinary Information Technology and Engineering Conference (IMITEC)*. IEEE, 1–7.
- [45] Visal Rajapakse, Ishan Karunanayake, and Nadeem Ahmed. 2023. Intelligence at the extreme edge: A survey on reformable TinyML. *Comput. Surveys* 55, 13s (2023), 1–30.
- [46] C Rajashekar Reddy, Tanmai Mukku, Ayush Dwivedi, Ashrit Rout, Sachin Chaudhari, Kavita Vemuri, Krishnan S Rajan, and Aftab M Hussain. 2020. Improving spatio-temporal understanding of particulate matter using low-cost IoT sensors. In *2020 IEEE 31st Annual International Symposium on Personal, Indoor and Mobile Radio Communications*. IEEE, 1–7.
- [47] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. Tinyol: Tinyml with online-learning on microcontrollers. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.
- [48] Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, Jeroen Hoebeke, and Eli De Poorter. 2018. Modelling the energy consumption for over-the-air software updates in LPWAN networks: SigFox, LoRa and IEEE 802.15. 4g. *Internet of Things* 3 (2018), 104–119.
- [49] Oscar Torres Sanchez, Guilherme Borges, Duarte Raposo, André Rodrigues, Fernando Boavida, and Jorge Sá Silva. 2025. Federated Learning framework for LoRaWAN-enabled IIoT communication: A case study. *IEEE Internet of Things Journal* (2025).
- [50] Ramon Sanchez-Iborra and Antonio F Skarmeta. 2020. Tinyml-enabled frugal smart objects: Challenges and opportunities. *IEEE Circuits and Systems Magazine* 20, 3 (2020), 4–18.
- [51] Nikolaos Schizas, Aristeidis Karras, Christos Karras, and Spyros Sioutas. 2022. TinyML for ultra-low power AI and large scale IoT deployments: A systematic review. *Future Internet* 14, 12 (2022), 363.
- [52] SemTech 2024. *LoRa® and LoRaWAN®*. SemTech. <https://www.semtech.com/uploads/technology/LoRa/lorawan-device-classes.pdf>
- [53] SemTech SemTech. 2024. *LoRa and LoRaWAN*. <https://www.semtech.com/uploads/technology/LoRa/lora-and-lorawan.pdf>. Last accessed: 10/2025.
- [54] SemTech SemTech. n.d. *SX1276 Long Range Low Power Transceiver Datasheet*. <https://www.semtech.fr/products/wireless-rf/lora-connect/sx1276>. Last accessed: 10/2025.
- [55] Yonatan Shiferaw, Apoorva Arora, and Fernando Kuipers. 2020. LoRaWAN class B multicast scalability. In *2020 IFIP Networking Conference*. IEEE, 609–613.
- [56] STMicroelectronics 2018. *STM32 Nucleo expansion board for power consumption measurement*. STMicroelectronics. https://www.st.com/resource/en/data_brief/x-nucleo-lpm01a.pdf
- [57] STMicroelectronics 2020. *STM32WL Nucleo-64 board*. STMicroelectronics. https://www.st.com/resource/en/data_brief/nucleo-wl55jc.pdf
- [58] Zehua Sun, Tao Ni, Huanqi Yang, Kai Liu, Yu Zhang, Tao Gu, and Weitao Xu. 2023. FLoRa: Energy-efficient, reliable, and beamforming-assisted over-the-air firmware update in LoRa networks. In *Proceedings of the 22nd International Conference on Information Processing in Sensor Networks*. 14–26.
- [59] Zehua Sun, Tao Ni, Huanqi Yang, Kai Liu, Yu Zhang, Tao Gu, and Weitao Xu. 2024. Flora+: Energy-efficient, reliable, beamforming-assisted, and secure over-the-air firmware update in lora networks. *ACM Transactions on Sensor Networks* 20, 3 (2024), 1–28.
- [60] Aiju Thomas and NV Eldhose. 2019. Performance evaluation of chirp spread spectrum as used in LoRa physical layer. In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*. IEEE, 1–6.
- [61] Dinarte Vasconcelos, Myat Su Yin, Fabian Wetjen, Alexander Herbst, Tim Ziemer, Anna Förster, Thomas Barkowsky, Nuno Nunes, and Peter Haddawy. 2021. Counting mosquitoes in the wild: An internet of things approach. In *Proceedings of the conference on information technology for social good*. 43–48.