POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

Politecnico di Torino

Master's Degree Thesis

Over-the-air update of ML models on IoT devices for precision agriculture

Supervisors

Prof. Umberto GARLANDO

Prof. Andrea MAGNANO

Candidate

Pedro PEREIRA MORAES

July 2025

Summary

This thesis presents the design and implementation of a system for remotely updating embedded Machine Learning (ML) models in Internet of Things (IoT) devices used in precision agriculture. The motivation stems from the need for scalable, low-power monitoring solutions capable of adapting over time through firmware updates, especially in remote agricultural deployments where manual maintenance is impractical.

The work begins with a comprehensive literature review covering communication technologies for smart farming, including WPAN, WLAN, and especially LPWAN protocols such as Sigfox, Ingenu, and LoRaWAN. LoRaWAN is identified as the most suitable option due to its support for Firmware Update Over-The-Air (FUOTA), energy efficiency, and scalability. In parallel, the review explores applications of ML in agriculture, particularly in crop health monitoring, and emphasizes the importance of running lightweight models locally on end-devices due to the limited bandwidth of LPWANs.

The implementation chapter details the full architecture of the proposed system. The solution uses the B-WL5M-SUBG1 board from STMicroelectronics, integrated with X-CUBE-AI for ML inference and STM32CubeWL for LoRaWAN FUOTA support. The backend leverages The Things Stack (TTN) as a network server and Node-RED for the application server, with MQTT used for device-server communication. The FUOTA process is orchestrated via a custom Node-RED dashboard capable of fragmenting firmware images, generating session keys, encrypting downlinks, and managing multicast transmissions.

Three experiments were conducted to validate the system: unicast FUOTA with standard firmware, unicast FUOTA with an ML model embedded, and full multicast FUOTA following LoRa Alliance protocols. The results demonstrate the feasibility of updating embedded ML models wirelessly, and highlight trade-offs in transmission duration, scalability, and regulatory constraints, such as duty cycle limits.

Overall, this work provides a complete and reproducible framework for deploying and maintaining intelligent IoT devices in precision agriculture, combining state-of-the-art ML tools with robust wireless communication protocols.

Acknowledgements

"I would like to offer my thanks to my father, Renato, my mother, Priscila, and my brother, Rafael, for their continuous and enduring support throughout my pursuit of educational prowess."

"To my supervisors, Prof. Umberto Garlando and Prof. Fabrício Junqueira, for their guidance and care during the development of this work."

"To the professors and staff of the Escola Politécnica and the Politecnico di Torino, for the important role they played in my academic education."

"And to the eLIONS Group, for giving me the opportunity to develop this research with them."

Table of Contents

List of Tables

List of Figures

Acronyms

ABP	Activation by Personalization
CSS	Chirp Spread Spectrum
DevEUI	Device Extended Unique Identifier
EUI	Extended Unique Identifier
FUOTA	Firmware Update Over the Air
IoT	Internet of Things
LoRa	Long Range protocol - physical layer
LoRaWAN	Long Range Wide Area Network
LPWAN	Low-Power Wide-Area Network
ML	Machine Learning
OTAA	Over the Air Activation
TTN	The Things Network - a LoRa cloud platform
TTS	The Things Stack

Chapter 1

Introduction

Guaranteeing food security for the entire human population is of the utmost importance. However, due to the expected population growth, by 2050 there could be an increase in food demand of more than 70% compared to the year 2008 [1]. Furthermore, to reach this goal sustainably, the increase in crop production needs to be unaccompanied of an increase in land area used for agriculture [2]. Thus, developing new technologies for Precision Agriculture is key for a sustainable agricultural system in the future [3].

Within this context, the use of Internet of Things (IoT) technology is prominent for real-time monitoring of data in a crop field [4, 5]. On addition to this, when using IoT devices for Precision Agriculture, the Long Range Wide Area Network (LoRaWAN) technology is a frequently used option for communication due to its low-power capabilities in wide area monitoring applications that require multiple end-nodes [6, 7, 8].

One of these monitoring applications is the assessment of plant status. To approach this, the use of Artificial Intelligence (AI) paired with IoT devices for estimating crop characteristics has presented promising results [5].

To allow for minimal energy cost, the type of network most commonly used in such applications has reduced data throughput, which increases the difficulty for transmitting heavy files such as images and videos, thus posing a problem when the AI models are running on the cloud [9]. Because of this, a possible solution is to have the machine learning model running on the end-node device. One of the works of eLIONS Group [10] has showed promising results on estimating plat status with ML models embedded in the monitoring device.

On the other hand, the ML model embedded on the device should be updated once a new model is developed and trained. With the devices scattered across the field, a solution for updating remotely is needed. For this, it is possible to use the Firmware Update Over-The-Air (FUOTA) process for LoRaWAN.

1.1 Objective

It is in this context that the firmware update over-the-air campaign becomes needed, which is the reason this thesis explores the integration of LoRaWAN-based FUOTA with embedded ML models to enable dynamic updates in smart farming devices.

Thus, this thesis work presents the use the B-WL5M-SUBG1 board of ST-electronics and their FUOTA support package in tandem with the X-CUBE-AI package to allow for the over-the-air update of ML models of devices that use this board. To be more specific, the choice of microcontroller board and network server used in this work stemmed from the development environment of the laboratory in which this project was born, eLIONS group. Furthermore, the platform used to manage the FUOTA campaign was based on the work of Sylvain *et al* [11], which presented a clear technical guide for a FUOTA with the STelectronics software. Finally, this thesis contains an introduction, present in this chapter, a literature review that describes communication protocols used for smart farming, details about LoRaWAN and LoRa-based FUOTA implementations.

1.2 Thesis Structure

This section will go over the structure of the entire text, starting with the chapter 2, Literature Review. After the literature review contextualizes this project, the implementation of the solution produced by this thesis is explained in the Implementation chapter, and the results obtained for this solution are presented and discussed in the Experiments chapter. The last chapter ends this thesis by furthering the conclusions achieved with this project and discussing perspectives for future works. Finally, to better explain the protocols defined by LoRa Alliance regarding FUOTA, an Appendix provides brief explanation about each protocol and its important technical specifications.

Chapter 2

Literature Review

To attain a theoretical basis for this work, an extensive research was performed to better understand the technology used in this project, and how it was used in similar projects to realize an over the air firmware update. Therefore, the first section will go over communication protocols used in smart farming, the second will go deeper into the one used for the project in this thesis and encapsulate the LoRa technology, the LoRaWAN protocol, and elucidate their characteristics. Afterwards, the third section explores different works that made use of the LoRa technology to realize an over the air firmware update, such an review is made to have a benchmark to analyse the results of this project.

2.1 Communication technologies used in precision agriculture

There is a variety of communication technologies that can be used in IoT deployments. One way to classify them is by the type of network they employ, such as WLAN (Wireless Local Area Network), WWAN (Wireless Wide Area Network), WPAN (Wireless Personal Area Network), and LPWAN (Low Power Wide Area Network). Thus, this section will explore different radio technologies used for communication in studies of IoT in the context agriculture in order to provide a reflection on the choice of network for a determined application.

In the work of Liu Kaiyi *et al.* [12] a system for monitoring temperature and humidity is designed using ZigBee, which is a wireless communication protocol intended for low-power, low-data-rate applications. It finds extensive applications in smart homes, industrial automation, healthcare and agriculture, especially in mesh networking, where devices can communicate indirectly through some intermediary nodes. ZigBee, a WPAN, has low transmission power, which is instrumental in minimizing energy consumption and allowing devices to operate for extended

periods on small batteries. Its range varies depending on the environment: It reaches up to 100 meters in open space, while indoors or in obstacles-laden areas, the distance usually drops to about 10-20 meters. Furthermore, it allows for a data transfer rate of up to 250 kbps, adequate in transferring the small amount of information that sensors and other IoT devices deal with. Therefore, the system could fulfill its function proficiently for its intended uses. However, due to the use of ZigBee, the system would find its limited use in smart farming due to its range constraint, even though temperature and humidity are important weather information for agriculture.

Sengupta *et al.* developed an IoT box for precision agriculture, named FarmFox, to acquire information on soil health by measuring its moisture, temperature, turbidity, and pH [13]. This device used Wi-Fi for its communication, which is is a widely deployed technology of wireless communication based on the IEEE 802.11 family of standards for internet connectivity at high speeds over LANs. It has found its applications in those segments of IoT that either require very high data rates or need integration with any existing WiFi infrastructure, such as smart home systems and industrial automation.

WiFi, a WLAN, works at relatively high transmission power compared to other IoT-specific technologies, but in the FarmFox, this is offset by using solar panels to power the battery of the device. Its range is stated to be 100 meters in this work. Furthermore, it supports high data rates, which is one of the strengths of this device.

Although WiFi provides high data rates, its energy consumption is higher compared to low-power alternatives, such as ZigBee or LPWAN. Since this thesis work does not tackle using an energy source like solar panels, this represents a problem. On top of that, it is limited by its range when considering a deployment across a large field.

Even though these works present a variety of solutions for particular problems in precision agriculture, for this thesis work it seems that the use of LPWANs would tackle a more broad solution. This is because it would, by definition, address the issues with range and energy cost that are present in many applications of smart farming.

2.1.1 LPWANs

One leading LPWAN in smart farming is Sigfox, which is credited to have been used in projects to monitor silo and tank levels, track grain stock temperatures, safeguard remote farmhouses and structures, secure gates to prevent livestock theft, ensure colony health with remote beehive monitoring, and oversee food temperatures throughout the cold chain [14]. An example of use of Sigfox can be found in WaterS, a prototype for water monitoring developed by Di Gennaro *et al.* It has

sensors for measuring pH, turbidity and temperature of water, and is stated achieve self-sustainability in terms of energy by using solar panels [15]. Sigfox, used in this work, is designed for IoT applications requiring low energy consumption, extended coverage, and minimal data throughput. It operates in unlicensed frequency bands, around 868 MHz in Europe and 915 MHz in the Americas.

Sigfox devices transmit at low power levels, which enables them to conserve energy effectively. This efficiency allows devices to operate on battery power for several years, significantly reducing maintenance costs. In terms of range, Sigfox excels with coverage reaching up to 10 kilometers in dense urban areas and up to 50 kilometers in rural or open landscapes, making it efficient for connecting devices in remote or sparsely populated regions.

The technology supports very low data rates. This limitation means it is suitable only for applications requiring small, infrequent data transmissions, which is not a problem for the monitoring applications in precision agriculture, such as the ones being developed by the eLIONS Group. However, it should be noted that Sigfox does not support over the air firmware updates as of 2024 [8].

There are works that identify LoRaWAN as the most suitable IoT protocol for smart farming applications due to its ability to meet three main criteria: energy efficiency, coverage, and scalability [7]. It was used in a monitory system for a peat swamp forest based on IoT [16]. In this work, the sensor nodes measure the soil and water temperature, soil moisture and water level. The collected data is sent to the network through a LoRa gateway, then inputted into their application server so that it can be displayed in a dashboard.

Another work for monitoring in smart farming where there was use of LoRaWAN was conducted in Kenya [17]. In this study, the sensors that measure soil moisture, temperature, and humidity transmit data as far as 1.2 kilometers to a central gateway. Data storage and analyses are done via cloud-based platforms for actionable insights toward optimization of resources. To be more specific, it uses the services of The Things Network for a Network Server, and it uses Node-Red to communicate the data to their Application Server. Both of these services were used in this thesis work, therefore they will be discussed further in another chapter.

This case study in Kenya claims that the use of the developed system can lead to annual savings of approximately 25% -30% on energy and water. Additionally, LoRa technology facilitated this work by providing long range communication without high energy consumption, which reinforces the argument that it is an apt choice for precision agriculture.

Ingenu's RPMA (Random Phase Multiple Access) technology is pointed out by some reviews as being another LPWAN that would be an appropriate fit for smart farming applications [8, 18]. It operates in the 2.4 GHz ISM band, which is why it can avoid several limits imposed by regulation on the uses of the sub-GHz bands. It is reported to achieves up to 168 dB link budget, and -142 dBm of receiver

sensitivity, which lets it maintain proper connectivity even over a long distance in adverse conditions. [8] According to Ingenu's website, this technology has already been deployed on agricultural applications, in partnerships with companies such as with US Sugar [19]. However, academic articles about deployments of this technology could not be found, which is a problem when considering to use it in a thesis work.

To summarize, it was concluded that LPWANs are well-suited for applications that involve monitoring points scattered in a wide area in the context of precision agriculture. On top of that, a comparison between the technlogies reviewed in this section can be better illustrated by the Table 2.1, created with information extracted from the study of Raza *et al* [8].

Table 2.1: Comparison of IoT Network Characteristics

	SigFox	LoRaWAN	Ingenu
Band	Sub-GHz ISM: EU(868 MHz)	Sub-GHz ISM: EU(433 MHz, 868 MHz)	ISM 2.4 GHz
Data Rate	100 bps(UL), 600 bps (DL)	0.3 - 37.5 kbps	78 kbps (UL), 19.5 kbps(DL)
Range	50 km (Rural)	15 km (Rural)	15 km (Urban)
Forward Error Correction	No	Yes	Yes
Medium Access Control	unslotted ALOHA	unslotted ALOHA	CDMA-like
Topology	star	star of stars	star, tree
Adaptive Data Rate	No	Yes	Yes
Over the air updates	No	Yes	Yes

Considering that Sigfox does not present support for over the air firmware updates as of the start of this project, it can not be used for this thesis work. Furthermore, since the use of LoRaWAN in academic papers is much more prolific than Ingenu, it appears to be the best choice for this thesis work. Therefore, the following chapter will delve into its modulation, and other important aspects such as those mentioned in Table 2.1.

2.2 Machine Learning for Agriculture

Neural networks have been used in precision agriculture for a variety of tasks, such as yield prediction [20], disease detection [21], weed detection [22], livestock production [23], water management [24] and soil management [25]. Furthermore, a review in the use of machine learning for agriculture [26] indicated that within

the task of crop disease detection, Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs) are the two techniques most frequently used. In this context, they fulfill the role of a classifier model. An example for this can be found in [27] where spectral reflectance features are used for detection of yellow rust on infected and healthy winter wheat canopies, which was done through a Multi-Layer Perceptron (MLP) ANN, with reported accuracy of 98.9% for healthy wheat and 99.4% for infected wheat.

Additionally, it is fair to assume that a technique that works for disease detection could work for plant health assessment, due to the similarity between both these classification tasks. For this endeavor, there is a variety of features that can be used as input for the machine learning model, such as imaging data, temperature and air humidity.

The works from the authors in [28, 29, 30] prove that plant impedance contains information on the plant status. Therefore, it can be used as a feature for estimating plant health. Hence, in the eLIONS Group, this has been investigated with ANNs, seeing as it has the capacity to deal with non-linear spaces and a great variety of feature inputs. In their work, the neural network outputs 1 for healthy plant and 0 otherwise [10, 31, 32].

However, when considering the monitoring of a large field, the use of a LPWAN is required. Therefore, the network has reduced data throughput, which increases the difficulty for transmitting heavy files. This, in turn, presents an obstacle to have a machine learning model running on the application server [9]. Due to this, it becomes necessary to have use a model that is simple enough to run directly on the IoT device. Furthermore, new and better models should be developed through the progress of researches, which creates a need for the model embedded in the device to be updated.

2.3 LoRa and LoRaWAN

When using IoT devices for precision agriculture, it is common to use Low Power Wide Area Networks (LPWAN). The one among them that is discussed in this work is the LoRaWAN (Long Range Wide Area Network) protocol. This chapter will delve into its structure, technical features and characteristics.

LoRaWAN protocol is designed to coordinate LoRa communication, which is a modulation technique derived from Chirp Spread Spectrum. That is to say that LoRa operates on the physical layer of an IoT device, handling signal modulation, while LoRaWAN operates on the Medium Access Control layer or higher, and aims to address network congestion issues and manage medium access [33].

2.3.1 Architecture

The deployment of a LoRaWAN network is organized in a star-of-stars topology, in which the main elements are called "end-devices" (or, "end-nodes"), gateways, and the "network server". In this structure, data is transmitted from end-devices to gateways over a single wireless hop, and is relayed by the gateway to the central network server through a non-LoRaWAN backhaul, with common examples beign WiFi, Ethernet, and mobile (3G/4G/5G). When a message travels in the previously described direction, it is called an uplink. On the other hand, when a message, be it a command or an acknowledgement, is sent by the network server to the gateway and relayed to the target end-device, it is called a downlink.

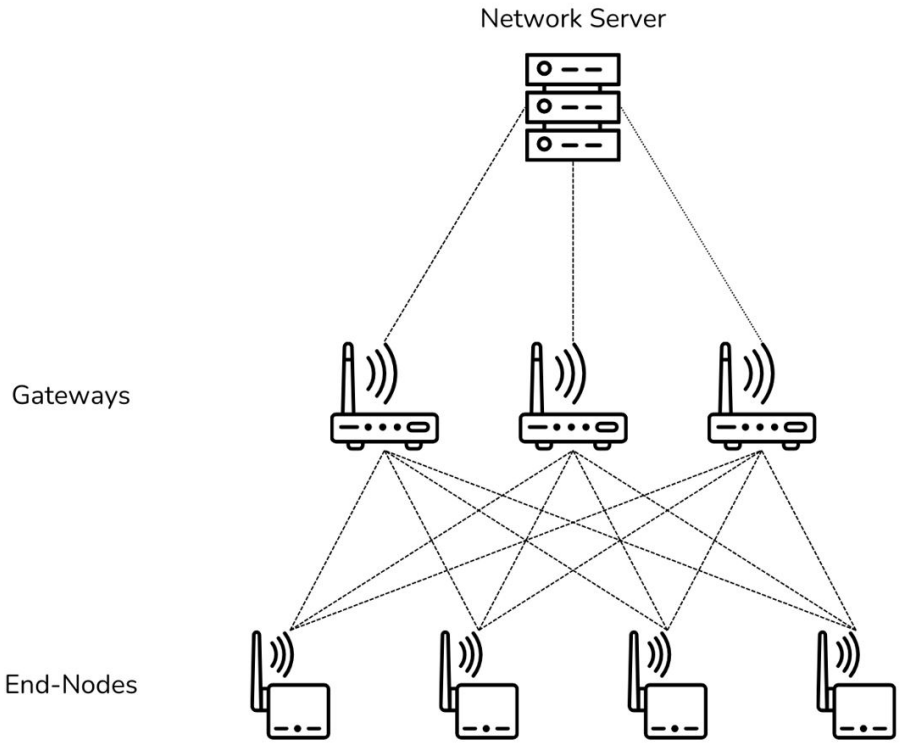

Figure 2.1: LoRaWAN Topology

In this work, the hardware used for the end-node was a B-WL5M-SUBG1 board from STelectronics, which, on top of LoRa capabilities, possesses an external flash to help with memory issues. The gateways were set up by the eLIONS Group, as they were already being used in other works. Lastly, the network server used was hosted by The Things Stack SANDBOX service, powered by The Things Industries, which will be referred to as TTN henceforth in this text, which stands for The

Things Network.

2.3.2 PHY - Physical Layer

In this section, the LoRa physical layer will be explored in order to understand how the mechanisms employed by this technology result in a robust method of long range communication with low-power usage, which justify its frequent use in IoT applications. To be more specific, the LoRa physical layer employs Chirp Spread Modulation (CSS), Forward Error Correction (FEC) and Adaptive Data Rate (ADR). These techniques will be discussed further in the following sections.

Chirp Spread Spectrum

The LoRa communication is derived from Chirp Spread Spectrum (CSS) modulation. This patented technology revolves around sinusoidal signals whose frequency varies with time linearly, which are called chirps [34]. Furthermore, CSS allows for low-energy long range communication, which are the most important traits of LoRaWAN. On top of that, it presents robustness to noise and interference. Lastly, this section will describe the mechanisms employed by CSS modulation (such as transmission and receiving) in order to understand why this technology is commonly used in IoT applications that require low-power consumption and long-range communication.

Modulation and Transmission

In order to perform the transmission, CSS modulation encodes each data symbol (a decimal value based on the binary sequence of length equal to the spreading factor SF) into a chirp signal. In this chirp, the frequency either increases (up-chirp) or decreases (down-chirp) over the entire bandwidth (BW). When the frequency in an up-chirp reaches its maximum, it wraps around to the minimum frequency and continues sweeping. For down-chirps, the frequency wraps from minimum back to maximum, covering the entire BW in one sweep [34]. Thus, the bit rate R_b and the symbol rate R_S can be written as [35]:

$$R_b = SF \times \frac{BW}{2^{SF}} \text{ bits/sec}$$

$$R_S = \frac{BW}{2^{SF}} \text{ symbols/sec}$$

Which helps illustrate that the spreading factor is a key parameter that influences data rate and power consumption. Higher spreading factors lead to longer chirps that are more resistant to noise but have a lower data rate. Conversely, lower spreading factors use shorter chirps, allowing faster data rates with shorter on-air

times. On the other hand, increasing the bandwidth has the opposite effect on data rate compared to spreading factor.

According to LoRa protocol, SF can assume integer values from 7 to 12, and BW can be 125, 250 or 500 Hz [36]. These values can be dynamically adjusted due to the Adaptive Data Rate (ADR) scheme often employed by LoRaWAN networks to optimize performance, battery life, and communication range by adjusting data rates according to signal quality and network conditions. [34, 37]. This mechanism will be further discussed in the section about the Medium Access Control (MAC) layer.

Demodulation and Receiving

The first chirps that a CSS receiver has to detect when a LoRa frame is sent to it are a series of basic up-chirps, which compose the preamble of the data frame and are used by the receiver to perform synchronization with the incoming data stream. Then, the information arriving in the data stream has to be demodulated, and for this the CSS receiver implements an operation called dechirping.

Dechirping is a process where the incoming chirp signal is multiplied by the complex conjugate of a reference chirp. This converts the received frequency-swept chirp into a single frequency signal. After this process, the receiver performs a fast Fourier transform (FFT) on the dechirped signal. As a result of the FFT there will be two peaks in the frequency domain (influenced by the SF and BW), and the transmitted symbol is identified by them, allowing the receiver to decode the transmitted data [34]. It should be noted that the use of dechirping in signal processing is an advantage for IoT application because it plays an important role in reducing noise, accurate data retrieval, and boosting the resilience of CSS modulation. [34]

Another important characteristic of the CSS receiver is the sampling frequency. Although the Nyquist theorem would instruct to use a sampling frequency $f_S \geq 2BW$, in CSS modulation the receiver selects the sampling frequency as $f_S = BW$. This enhances symbol detection through constructive aliasing, where the frequency components of the dechirped signal fold over in a way that reinforces the main peak corresponding to the transmitted symbol's frequency bin [34]. Furthermore, a lower sampling frequency than what is proposed by the theorem means less processing needed, which in turns reduces battery usage.

FEC - Forward Error Correction

FEC is a type of error correction in which the transmitter encodes additional redundant data within the frame to be transmitted, which can be used by the receiver to recover missing data. Thus, the recovery is only possible if enough data is successfully received [38].

Therefore, to address the eventual corruption of data during transmission, the LoRa physical layer employs Hamming codes [39], which is a FEC algorithm that adds redundant bits to the data following a defined Code Rate (CR). The CR is a ratio that represents the number of bits in a data stream that carry non-redundant information [38, 40].

In LoRaWAN, the CR that may be adopted are [37]:

- 4/5
- 4/6
- 5/7
- 4/8

To further exemplify this FEC algorithm, the illustration below represents a section of the data stream sent by the transmission when the code rate is 4/5. This means that the transmitter generates 5 bits of information for every 4 bits that it wants to send.

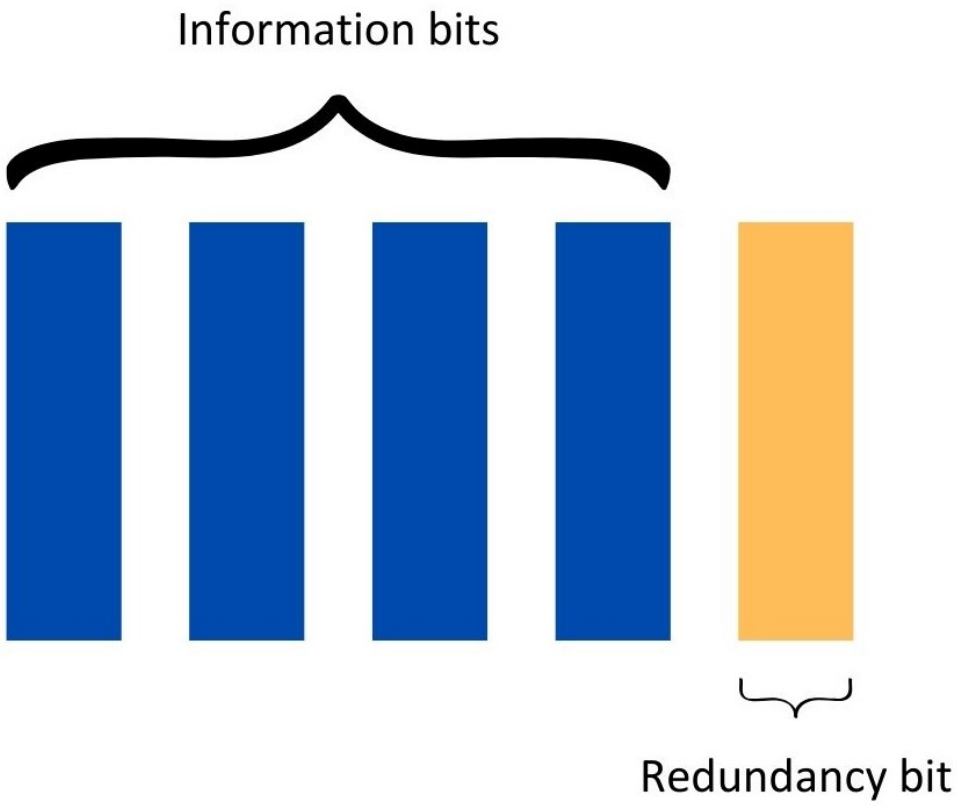

Figure 2.2: Data Transmission with CR = 4/5 for FEC

Therefore, it should come as no surprise that this mechanism has an impact on the bit rate R_b. Thus, considering this FEC mechanism, the LoRa modulation bit rate can be written as [37]:

$$R_b = SF \times \frac{BW}{2^{SF}} \times CR \text{ bits/sec}$$

Which means that a smaller CR can make the transmission more resistant to corruption, however, it decreases the bit rate and increases air time. Furthermore, this mechanism employed by LoRa helps in understanding its robustness, and is an important factor in Fragmented Data Block Transport, which is one of the main steps of FUOTA that will be discussed in the next chapter.

2.3.3 MAC - Medium Access Control

LoRaWAN is an open MAC layer specification for LoRa technology. To understand how LoRaWAN achieves features such as energy efficiency, and security in communications, it is important to understand the mechanisms that are explained in this section. This MAC layer permits important operations such as a join procedure that can keep the connections secure and dynamically add devices to the network with OTAA. It handles retransmissions, device addressing, and duty cycle restrictions so that regulatory compliance is kept and shared spectrum used in the most efficient way [34]. The MAC layer supports Adaptive Data Rate, which will dynamically change the data rate and transmission power to optimize battery life and network capacity. Other features involve device classes, where class A, B, and C serve different needs in communication-as indicated by latency sensitivity or downlink priority. These features enable LoRaWAN, in summary, to support various IoT applications that come with a respective set of demands on long-range, reliable, and lowpower communication. Thus, this section will go over these mechanisms to further understanding the use of LoRaWAN for this work.

For example, one important definition for battery-saving imposed by the MAC layer, based on regional parameters, is that in Europe the maximum duty cycle for transmission in end-devices is 1%.

ALOHA

In LoRaWAN, bi-directional end-devices communicate following an ALOHA-type protocol, based on pure ALOHA with an additional acknowledgment mechanism for the case of a confirmed message [41]. The ALOHA network protocol dates back to the 70s and is still finds prolific use in the present due to its simplicity. In pure ALOHA, the device transmit data regardless of network state, and if two devices transmit data packets at the same time there is a collision and both packets are

lost. After detecting a collision, each device waits a random time before attempting to transmit again.

2.3.4 LoRaWAN Classes

The LoRaWAN protocol define three classes for the end-devices, A, B and C. All end-nodes are Class A-capable, which is the most energy-efficient class, where an end-device has only two windows to receive a downlink from the gateway after sending it an uplink [33, 41]. In Class B, devices open receive windows on scheduled times on top of the ones that are opened after an uplink. In order to schedule the time for these slots, they receive a time synchronized beacon from the gateway. As for Class C, the end-device has its receive window active almost continuously. For this reason, it is the least energy efficient class. However, this allows for lowest latency in communication. On top of that, this class must be used to perform the over-the-air firmware update. That is to say, the end-node of interest for this project would be on Class A and switch to Class C in order to receive a firmware update, and once it is completed, it would switch back to Class A.

Join Procedure

The MAC layer sets the rules for how devices will be able to join a network. Furthermore, according to LoRaWAN protocol, an end-node may join the network through either Activation By Personalization (ABP) or over-the-air activation (OTAA) [34]. In this work, only OTAA was used. For this choice, a few considerations had to be made.

Firstly, OTAA negotiates the device and network server security by dynamically interchanging the keys during the join procedure. This automatically means that session keys are unique for each device and session, something that secures it very much. As opposed to this, ABP uses static pre-programmed keys. In case these keys are compromised-which could be because they are intercepted or replayed-security is jeopardized.

OTAA allows for better scalability of the network. One of the features of configuration is that devices can be configured to automatically rejoin the network in response to changes such as key rotations or updating networks without requiring a manual re-configuration. This flexibility makes OTAA really suitable for big deployments where it would be impracticable to handle handsets one by one.

Put differently, OTAA provides a better security and adaptability approach for most use cases in IoT, especially when long-term reliability and security may matter the most. For an end-device to use this activation method, it must be personalized with an End-device identifier (DevEUI), a Join-Server identifier (JoinEUI) and an Application key (AppKey).

The Join Procedure defined by the LoRaWAN protocol for this activation requires that end-devices initiate the procedure by sending a Join-Request frame which should contain as payload a DevEUI, a JoinEUI and a DevNonce. The transmission of this frame can follow region-specific parameters, and in Europe it follows the definitions of the EU863-870 MHz Band, that states that the broadcast of this frame shall be on the channels 868.10, 868.30 and 868.50 MHz [36].

If the end-device is accepted to join the network, the network responds to the Join Request with a Join Accept frame, through the gateway. This frame informs the end-device of downlink configuration settings and the delay between transmit window TX and receive window RX, among other identifiers and a non-repeating value for key generation.

ADR - Adaptive Data Rate

Frequently, LoRa networks will utilize ADR to dynamically specify the transmission parameters, depending on the distance of the device from the gateway. Devices that are close to a gateway can use the lower spread factors, transmitting quickly and with low power. Distant devices use higher spread factors in order to reach the gateway but still maintain low power use because CSS minimizes retransmissions even at low signal strengths.

The network server used in this work was hosted by The Things Stack. Their service can employ an ADR scheme based on Semtech's recommended algorithm [42]. It has as a first priority keeping the signal-to-noise (SNR) ratio higher than the minimum SNR necessary for the modulation. The second priority is to increase the data rate, as long as the first priority is still satisfied. As a third priority, it decreases transmission power. Lastly in case packet loss is high, it increases the number of retransmissions [42, 43].

It achieves this by calculating a SNR safety margin, given by the difference between the maximum SNR over recent transmissions determined through the method described in[44] and the minimum SNR to demodulate an uplink given the current parameters thar are defined through the method in [45]). Then, the data rate is increased as long as there is enough margin, following the definition in [46]. In case the data rate reaches its maximum while there still is margin, the transmission power is reduced. The packet loss rate is addressed through [47].

2.4 Related Work

One of the earliest work to approach the FUOTA specifications for LoRaWAN was published in 2020 [48]. In this study, the authors developed a simulation tool, called FUOTASim, to evaluate possible parameters for this technology. This tool was written with the SimPy package for Python, and is available on github [49].

Among the parameters that could be entered on the simulation were the number of end-devices, the number of gateways, the data rate of the transmission, the size of the fragments and the size of the firmware to be transmitted and the class type for the multicast session (Class B or C).

The work presented in [48] defined important metrics for evaluating OTA updates, such as total update time, energy consumption and update efficiency, which is the average ratio of end-devices that successfully receive the firmware image. One could consider that if there is a greater update time would decrease battery lifetime, seeing as the device would consume more energy the longer it stays on Class C. Furthermore, the results of this work have shown that the time metric was proportional to firmware image size within the interval of 5 Kbytes and 100 Kbytes.

In the work of [50], the firmware update process did not use a LoRa protocol based on the star-of-stars architecture such as LoRaWAN. It instead adopted the LoRaP2P+ protocol, which employs a peer-to-peer architecture, as the name suggests. In this work, experiments were performed with 1, 2, and 12 nodes, evaluating the system's performance using different spreading factors (SF8 and SF9) at distances of 1 meter and 1 kilometer. The results showed that a 335 KB firmware image could be reliably transmitted and applied even in the 12-node setup. Notably, the full FUOTA process completed within 24 hours in the most demanding scenario (12 nodes at 1 km with SF9), demonstrating the scalability and robustness of the proposed decentralized approach for rural LPWAN deployments. It should be noted that the maximum payload size in this protocol is 40 bytes per packet—significantly smaller than LoRaWAN's maximum of 115 bytes when using SF9—which implies a higher number of fragments and potentially increased overhead during the transmission process.

To address the limitations of FUOTA over LoRaWAN using a single gateway, Charilaou *et al.* [51] extended the FUOTASim simulation framework to support multi-gateway deployments. Their study demonstrated that gateway diversity can significantly enhance update performance by increasing fragment delivery success and reducing transmission distance. Simulations with up to 30 gateways and 10,000 devices showed that, particularly for larger firmware images (e.g., 100 kB), multiple gateways reduced update time from over 11 days (single gateway) to less than 19 hours. Moreover, update efficiency reached 100% when 23 or more gateways were deployed, even with low spreading factors such as SF7. The authors evaluated the impact of redundant fragments and found that, in scenarios with fewer gateways, redundancy helped improve update success, albeit at the cost of higher energy consumption. These findings underscore the importance of gateway placement and spatial diversity in enabling scalable and energy-efficient FUOTA in large-scale LoRaWAN networks.

Security has been a major concern in FUOTA research. Anastasiou *et al.* [52]

proposed a blockchain-based framework to enhance the integrity and authenticity of firmware updates over LoRa. Their architecture leverages smart contracts to verify device eligibility and firmware metadata before installation. Although their system supports multicast transmission using FUOTA specifications, the evaluation was conducted using unicast downlinks, which poses significant scalability issues. For instance, updating 200 nodes with a 50 KB image using SF12 was estimated to require up to 62 days. A complementary approach is presented by Mtetwa *et al.* [53], who employed blockchain to secure firmware delivery, focusing on devices operating in remote and unpredictable environments. While these solutions strengthen the security layer of FUOTA, the reliance on unicast transmission and the lack of gateway diversity suggest limitations in terms of update speed and network scalability.

Aiming to improve update time efficiency, Malumbres *et al.* [54] investigated the combination of unicast and broadcast transmissions in the firmware update process over LoRa networks. Their study focused on industrial IoT devices in environments where energy is not a limiting factor and minimizing update time is the main priority. Three methods were analyzed: only unicast, broadcast followed by unicast for missing chunks, and only broadcast. Through analytical modeling, simulation, and partial real-world testing, the authors showed that hybrid methods can substantially reduce update time compared to pure unicast. In some scenarios, especially with a large number of nodes and favorable link conditions, the total update time was reduced by up to two orders of magnitude. The results demonstrated that the optimal number of initial broadcast rounds depends on the packet loss probability, with higher loss environments benefiting from multiple rounds to minimize the required unicast retransmissions.

With the goal of building a perspective on the range of time that the over-the-air update can take, the Table 2.1 below was made based results found in each of the works cited in this section. Together with each update time presented, this table shows different factors that affect the update speed.

The comparison presented above highlights the wide variability in FUOTA performance across different implementations, protocols, and experimental contexts. One of the most evident trends is the direct relationship between firmware image size and update duration, which confirms that transmission time scales linearly with the volume of data, especially in scenarios using unicast. However, this is not the only determining factor. The number of gateways, type of transmission (unicast, multicast, broadcast), payload size, and data rate (which is itself determined by spreading factor and bandwidth) all play significant roles in the overall performance of the update process.

Among the studies surveyed, the shortest update time per kilobyte of data was generally observed in setups that used multiple gateways and multicast delivery. For example, Charilaou *et al.* demonstrated that increasing gateway count dramatically

Table 2.2: Comparison of image transmission parameters across different studies

	[48]	[50]	[51]	[52]	[53]	[54]
Image Size (kB)	100	335	10	50	5	100
Datarate	3	3	5	5	0	5
Time (Minutes)	220	932.5	16667	2160	31.1	6000
Number of Gateways	1	1	5	1	1	1
Number of Nodes	1	12	10000	200	1	70
Casting	Multicast	Broadcast	Multicast	Unicast	Unicast	Unicast and Broadcast
Payload Size	115	40	222	222	51	215

improved update efficiency, reducing transmission time from over 11 days to under 19 hours for a 100 kB image when 23 gateways were used. This underscores the importance of gateway diversity in reducing transmission range and improving delivery reliability, especially in large-scale deployments.

In contrast, systems that relied solely on unicast delivery—particularly those using higher spreading factors (e.g., SF12)—suffered from extended update durations. Anastasiou *et al.*'s blockchain-secured approach required up to 62 days to update 200 nodes with a 50 kB image, highlighting the practical limitations of unicast in large-scale deployments, regardless of the additional security it may offer. Even with fewer nodes, such as in the work by Ho Teck Khieng *et al.*, which used only 12 devices, the 24-hour update time for a 335 kB image shows how quickly time requirements can grow with firmware size and network constraints.

The protocol architecture played a notable role. For instance, LoRaP2P+—a peer-to-peer protocol used in one of the studies—achieved a functional performance to update 12 end-nodes in under 16 hours, but at the cost of smaller payload sizes and a decentralized model that may be less practical in managed network environments. On the other hand, hybrid approaches that combined unicast and broadcast, as studied by Malumbres *et al.*, offered significant performance gains in environments where energy constraints were less critical and rapid update cycles were prioritized.

Although the table aggregates key parameters such as payload size and data rate, it is important to note that many relevant factors—such as packet loss rates, duty cycle enforcement, gateway scheduling delays, and retransmission strategies—were

not consistently reported across studies. This makes direct comparison imperfect. Nevertheless, the overview provides a valuable reference for what can be expected in terms of performance boundaries for FUOTA over LoRa-based systems. It supports the rationale for the implementation presented in the next chapter, which was developed with the understanding that not all optimization factors could be controlled or replicated, particularly under real-world constraints such as limited gateway access, regulatory limitations on downlink scheduling, and incomplete vendor documentation.

Ultimately, the comparison serves not only to situate this thesis within the broader landscape of FUOTA research but to demonstrate that there is room for improvement, especially in balancing scalability, reliability, and update speed in constrained environments.

Chapter 3

Implementation

This chapter will detail the steps taken to bring this project into reality, with the goal of providing the reader with the necessary knowledge to reproduce the project. Therefore, this chapter is divided between a section that describes the materials and methods used for the project, and a section that explains a user guide to allow any reader to reproduce the work in this project.

To briefly explain, in practical terms, how an user would perform a FUOTA process for their project after deploying their end-device(s) that are embedded with FUOTA-able software: The user would develop their new firmware on their local machine, and generate a file containing the fragmented firmware image. Afterwards, through their Application Server (dashboard hosted on Node-Red) the user would send commands to the Network Server, hosted by The Things Network (TTN), that would then pass them on to the gateway and send them to the End-Device. These commands would be unicast downlinks to have the end-device(s) enter a multicast group, prepare for a fragmented data block transport session, or switch to Class C. Once this preparation has been done for all target end-devices, the user would upload their fragmented image onto the Application Server, and with the click of a button the server would periodically send each fragment to the Network Server, which, through a defined multicast group, would use one of the registered gateways to transmit each fragment to the end-device(s) as a multicast downlink.

In regards to the choices made for the implementation of this project, some considerations are made based on the theoretical insights discussed in the Literature Review. Firstly, the choice of LoRaWAN as the communication protocol was motivated by its balance between long-range transmission, low power consumption, and support for Firmware Update Over-The-Air (FUOTA), as indicated in Table 2.1. These characteristics are particularly relevant in the context of precision agriculture, where end-devices are deployed across wide areas with limited access to power sources. Furthermore, the academic prevalence of LoRaWAN in FUOTA-related work, as opposed to alternatives like Sigfox or Ingenu, provided a strong

basis for its selection in this project.

Additionally, the reduced throughput of LPWAN networks, highlighted in Section 2.1.1, reinforced the need for running machine learning models directly on the end-device. This design avoids the necessity of transmitting large volumes of data, such as images, through constrained channels. The discussion in Section 2.2 emphasized the feasibility of embedding lightweight neural network models in microcontrollers, which was incorporated into the firmware design using the X-CUBE-AI middleware.

Lastly, the implementation relied on an understanding of the LoRaWAN device classes, as discussed in Section 2.3.4. The project leverages the Class A mode for regular operation to preserve battery life and temporarily switches to Class C during the FUOTA session, as required by the LoRa Alliance specification. This transition was carefully implemented and automated through the use of downlink commands, as detailed in the following sections. Thus, the concepts explored in the Literature Review provided the theoretical foundation for the design choices that guided the development of the solution presented in this work.

3.1 Materials and Methods

This section aims to provide a detailed description of the software and hardware utilized in this project. It serves to establish the foundational understanding of the tools and resources required for a user to replicate the work, thereby facilitating a clearer comprehension of the procedures outlined in the subsequent sections.

3.1.1 Hardware

For the implementation of the project in this thesis, the board used as an End-Device was the B-WL5M-SUBG1 board of STMicroelectronics. This board was chosen by the eLIONS Group because on top of its LoRaWAN capabilities, it possesses an external flash memory, which should help with allowing for space to run the machine learning model on the device. Additionally, since this board is part of the WL series, it can be used with the STMCubeWL software package. This package provides helpful tools and a basic example for a FUOTA-able LoRaWAN end-device. Furthermore, the FUOTA campaings were performed remotely with the gateways at the eLIONS group office.

3.1.2 Software

The software used as a terminal to monitor the board on a local machine is Tera Term. With this open source tool, it was possible to see the application logs of the end-device through a simple configuration. The network server for this

project was hosted by The Things Stack SANDBOX, a free service provided by The Things Network, where most of eLION's LoRaWAN projects are hosted. Their feature of backing up application logs was helpful for keeping track of the update time each FUOTA session took. Node-Red was used to deploy the Application Server. This development tool, that allows for flow-based visual programming, was used to create a dashboard that would be easy to use for future researchers. This dashboard was based around the work of [11], but with modifications and additions to make it compatible with TTN and allow for the Multicast Downlink of firmware fragments. This work included writing function blocks in Javascript, and the communication between Application Server and Network Server is brokered by MQTT. To work on the firmware of the End-Device, it was necessary to use tools from STMicroelectronics, such as the STM32CubeProgrammer and STM32CubeIDE version 1.12.1. On top of that, the STMCubeWL software package was helpful as a basis and a validation set for the firmware.

3.2 Deployment and Replication Instructions

This section will go over the steps to be taken in order to reproduce the results of this work, and how to use the tools developed for it. To be more specific, this guide is for a project that performs FUOTA with the hardware and software required by the eLions Group, that is, the B-WL5M-SUBG1 board and the TTN service for Network Server. Furthermore, the code for the Application Server used in this project is specific to work with TTN and follows the stipulations of LoRa Alliance for FUOTA.

First, it will go over how to utilize the FUOTA software found in STM32CubeWL for the End-Device, without applying significant changes to it. Then, it will describe how to use the TTN services for registering an End-Device, a Gateway, and a Multicast Group as this is a necessary step for configuring the network server of this project. Afterwards, it will address the code used for the Application Server, the changes and additions made to it, and how it can be used for future researchers. Lastly, this section will go deeper into modifying the firmware for the End-Device, to allow it to run an embedded ML model, and how to modify the memory mapping to allow for more space if the need were to arise.

3.2.1 Building and programming the firmware

This section will briefly explain how to build and program the firmware provided in STM32CubeWL for FUOTA in the B-WL5M-SUBG1 board. These instructions are still valid when using the firmware developed to support ML models. First, the user should navigate to the directory with the scripts in the folder structure, open the setenv.bat file with any text editor and make sure the path CUBEPROG_EXE and

CUBEIDE_EXE is properly set. Then, the user ought to enter the STM32CubeIDE directory and run the script build.bat. Afterwards, the user should connect the board to their PC with the STLink-V3, and power the board, which can be done by connecting a USB-C cable to the board extension. Lastly run the script program.bat and power cycle the board.

To monitor the board on a local machine, the user may configure Tera Term by selecting Baud rate of 9600, and configuring New Line Receive to AUTO on the Terminal Setup tab. After power cycling the board, if Tera Term is connected through the virtual COM port of the ST-Link, the user should see the log shown in Figure 3.1.

3.2.2 The Things Stack SANDBOX - Network Server

The Things Stack SANDBOX is the name of the service used as network server for this project, referred to in this text as TTN. This subsection will explain the steps required to take in this platform. After logging on an account on TTN, it is necessary to create an application or use an existing one to register the end-device and the multicast group. In case no gateways are registered, it would be necessary to register one.

Registering the End-Device

After clicking on "Register end-device" on the Application Overview, the user must select the option "Enter end-device specifics manually" under the Input method.

Then, the Frequency plan, LoRaWAN version and Regional Parameters version used by the user's device must be selected. An example of the registration is shown in Figures 3.2 and 3.3.

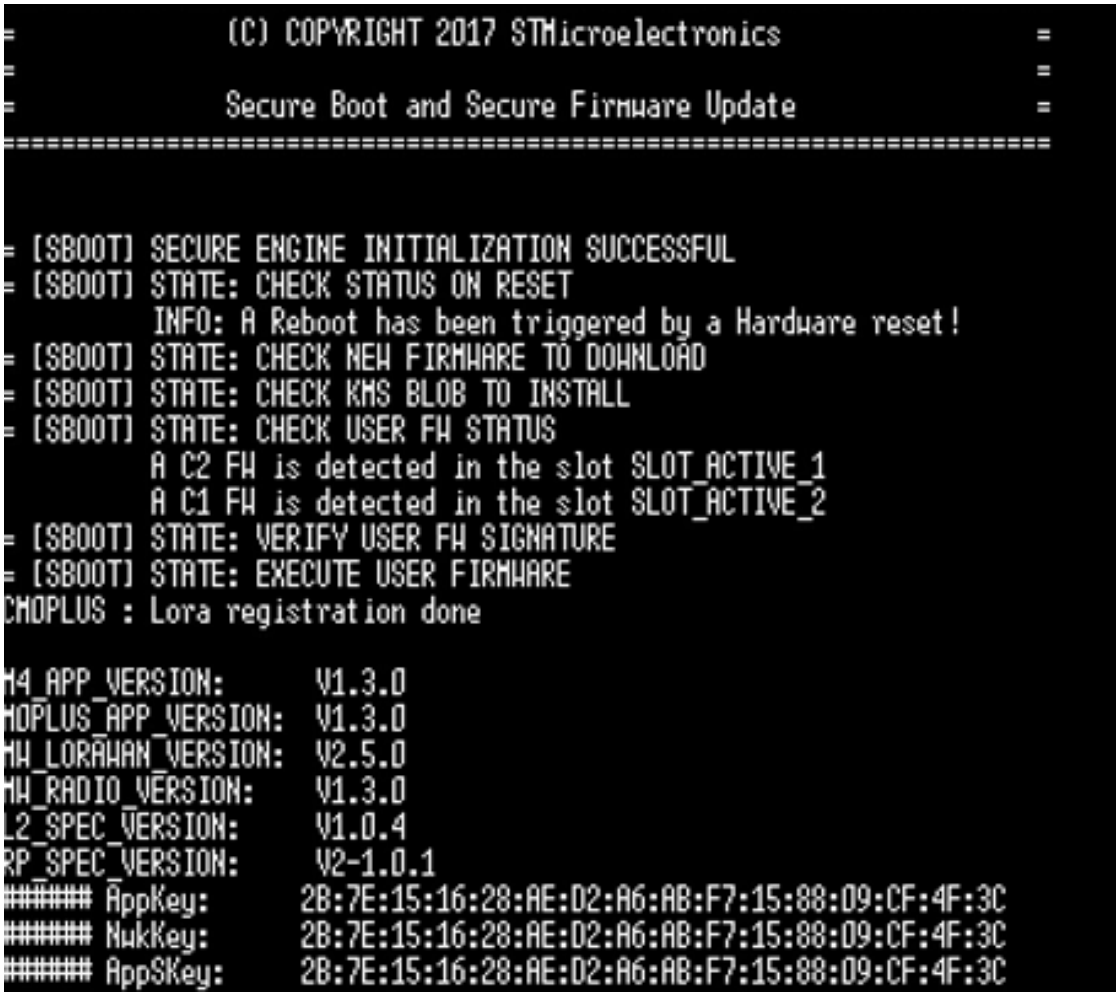

Figure 3.1: System Boot on TeraTerm

Lastly, in "Provisional information", the user must input some identification information about the end-device. This information can be found at the se-identity.h file of the user's LoRaWAN End-Node project. Furthermore, the DevEUI of their device can be found written physically on their hardware. On the se-identity.h file, the DevEUI set to all zeros means that it will use the unique DevEUI written on the hardware. With regards to the end-device ID, the user has more liberty of choice. It is simply a name that must be unique within the application. After clicking on Register end-device, the registration will be completed successfully.

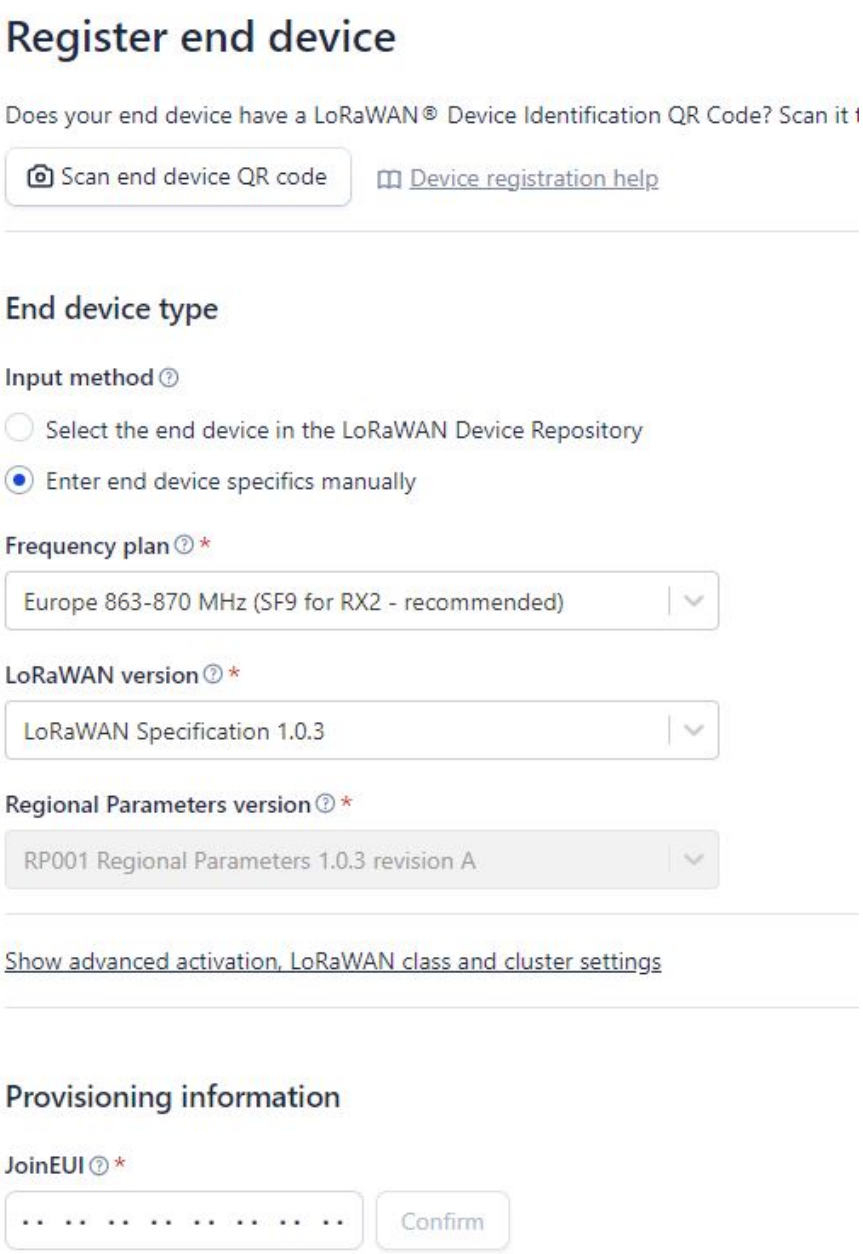

Figure 3.2: End-device Registration Part I

Gateway

After clicking on "Add new gateway", the user ought to provide the website with their gateway EUI, an arbitrary gateway ID and a gateway name. Then, the user must select the frequency bandplan used by the gateway. It should be noted that

for the gateway to work properly, the user should choose a bandplan compatible to the region of the TTN cluster they are using, for example, EU869 for the eu1 cluster.

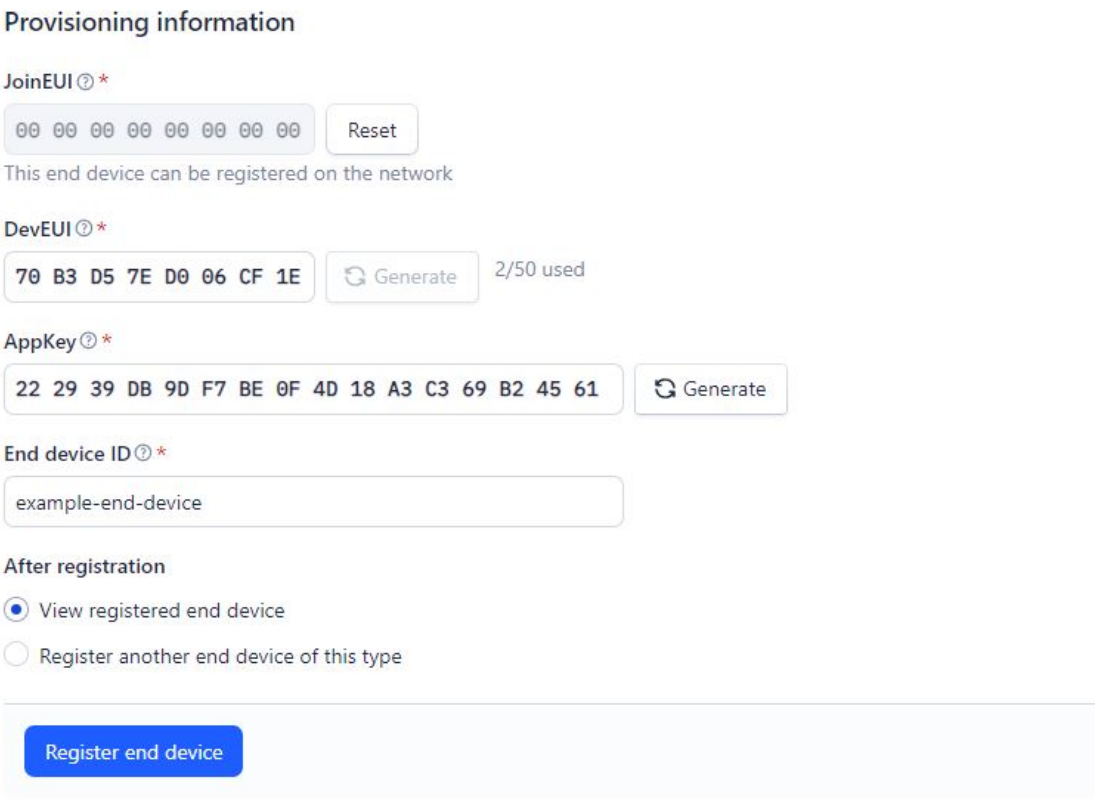

Figure 3.3: End-device Registration Part II

Multicast Group

Creating a Multicast Group on TTN is similar to an End-Device, because it is treated as a virtual end-device. The main difference lies in selecting a different activation mode: ABP & Multicast instead of OTAA. After this, the user must write, in the Provisioning information section, the Multicast Address (McAddress) as the Device Address, and the McAppSKey and McNetSKey as AppSKey, and NetSKey respectively. These multicast session keys and the address can be generated in the Application Server, as will be explained later.

Sending fragments to this virtual device is different from sending to an end-device operating in Class A. This is because in Class A, the uplinks from the end-device provide the network server with the necessary path information for sending a downlink. However, this path information is not sent by the end-device

if it is operating on Class C, therefore, the downlinks that are to be sent to the multicast group have a different format, as explained on this TTN page [55]. To summarize, the message sent by the Application Server must carry the information of the absolute time to send the downlink, and the IDs of the gateways to use for transmitting it.

3.2.3 Application Server

For any complex use of LoRaWAN, an Application Server is necessary. A FUOTA campaign is no exception to this. In this work, the application server is hosted with Node-RED and is based on the work of Sylvain *et al.* [11]. It takes the form of a dashboard and uses MQTT to communicate with the TTN server. The three main functions of this dashboard can be summarized as uplink decryption, downlink encryption and downlink automatization. The following subsections describe how an user could setup this Application Server, in order to reproduce the results of this work, and afterwards will aim at a deeper explanation of its main functions. The Application Server communicates with the network server through an Publish/Subscribe (Pub/Sub) messaging pattern. In this project, the messages are brokered via MQTT, which allows the dashboard hosted on Node-RED to receive and transmit data from and to The Things Network.

On one direction of information flow, related to the uplinks, TTN acts as the publisher, sending device messages to a specific MQTT topic, while Node-RED subscribe to that topics to consume the data. On the other direction, related to the downlinks, the dashboard acts as the publisher and the network server as the subscriber. This architecture enables real-time, event-driven communication between them.

3.2.4 Configuring the dashboard and Firmware Fragmentation

After installing the necessary software, the user should open command terminal and run node-red, then access the address localhost:1880 in any browser. Then, the user should import the .json files for the dashboard into the platform. However, before e deploying, it is necessary to configure the MQTT broker and the topic to subscribe to receive uplinks.

To do such, the user ought to double click on either the Publisher or Subscriber node (purple nodes) and then select to edit the broker. Then, in Connection, they write the address of the TTN cluster they are using (in this case, eu1.cloud.thethings.network), and port 1883. Then, in security the username is the application ID, and the password is the API key that was generated.

Then, to configure the Subscriber node, the user should click on the "Topic to subscribe" node and write "v3/**APPID**@ttn/devices/**DEVID**/up", replacing the **APPID** and **DEVID** by their application id and device's id on TTN. Finally, they can click on deploy and open their dashboard by going to the address localhost:1880/ui.

Before using the dashboard to perform FUOTA, the user must fragment the firmware image to be sent to the device. A possible logic for the generation of fragments, with redundancy, is found in [56], and this project made use of the python script written by Sylvain *et al.* [11] to fulfill this task. As defined by the LoRa Alliance protocol, each fragment of the image has 3 bytes to serve as header for its information. Therefore, a fragment of 115 bytes carries 112 bytes of actual fragmented data.

3.2.5 Multicast Session Setup

To setup a Class C Multicast session, it is necessary to generate multicast session keys that are used by the end-devices and by the network server to communicate. To realize this on the dashboard for Multicast, the user would make use of the AppKey defined in the firmware, which can be found in the file se-identity.h, as the GenAppKey for the random generation of session keys, as shown on Figure 3.4.

With the keys, it is possible to create the McGroupSetupReq command, and send it as an Unicast Downlink. If the setup is succesful, the board will send an uplink with hexadecimal payload of 0200 to the port 200, which can be seen in the dashboard. To have the board switch to Class C, the user must generate the McClassCSessionReq command. This command takes into account the absolute time to start the Class C session, and the answer that the board sends to this command informs the user of the time, in seconds, until the session start. This is leveraged in the Uplink Decryption of the dashboard, to facilitate the process for the user. Both these commands are to be sent to the port 200 of the device as Unicast Downlinks.

Uplink Decryption

The end-node will periodically send uplinks to the network server, as it is LoRaWAN standard. To receive them from the network server into the dashboard, it is necessary to use a Subscriber node to listen to data on the appropriate topic. The correct topic is defined by the identification of both the "end-device" and the "Application" on the TTN console.

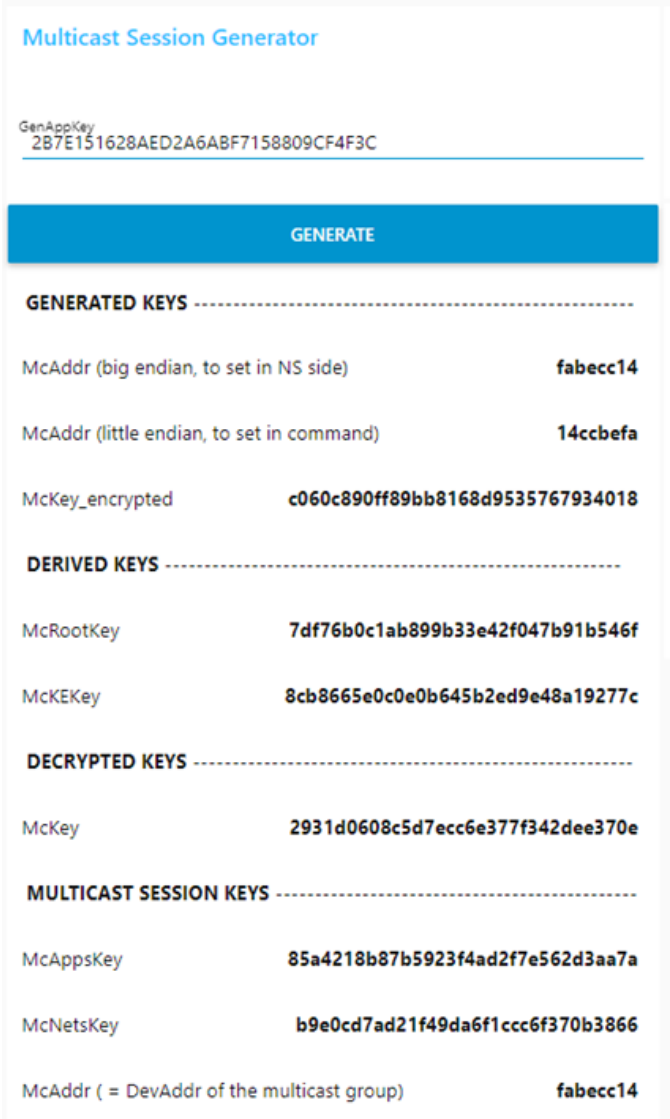

Figure 3.4: Multicast Session Keys Generator

On top of the periodic messages, the device can send messages specific to a particular protocol of the FUOTA implementation. Thus, once the message is received by the Subscriber, the first step is to evaluate which is the port of the message. This tells the user the general topic of the message, for example, an uplink related to a Data Fragmentation Session would be sent to the port 201. Then, it is necessary pass the payload from Base64 to Hex, which will help in the Uplink Study. The study based on Hex had already been implemented by Sylvain *et al.* [11] and translates the payloads defined by the LoRa Alliance protocols into plain english. An example of this is shown on Figure 3.5, where the end-device

sent an answer to a request to set up a fragmentation session.

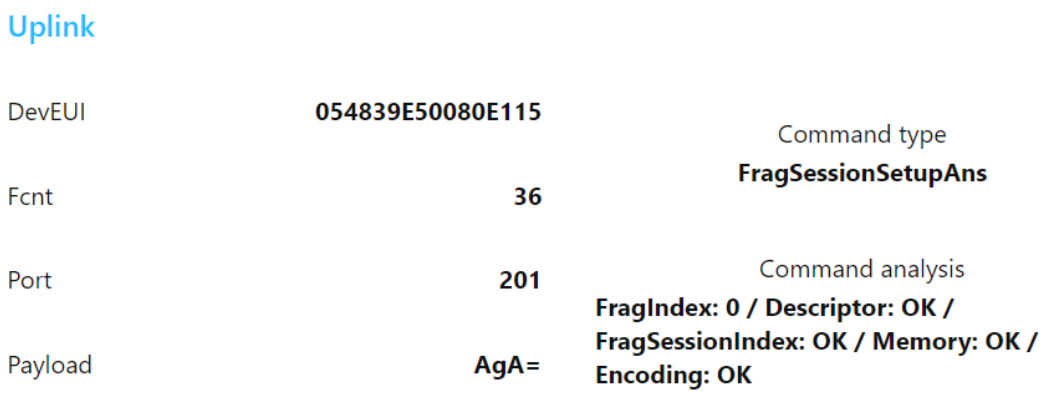

Figure 3.5: Uplink Decryption Example

An addition was made to this capability of the dashboard, to allow for reading the "Time to start the session" information, contained in the uplink that answers the request for a Class C session. This was helpful in making sure that the clock on the End-Device was synchronized with the server.

Downlink Encryption

Another helpful feature of this dashboard that was implemented in Sylvain *et al.* [11] are the windows that display commands that can be sent to the end-device in hex code. These commands are defined by the LoRa Alliance protocols summarized in the FUOTA chapter. This functionality is particular useful for generating and displaying the command to set up a fragmentation session, as the particular command depends on parameters like the number of fragments that are intended to be sent.

In the console of TTN, it is possible to send the command in hex code to the specified port of a chosen end-device. However, to send a downlink through the API of TTN server, it is necessary to encode the payload as Base64. It is necessary to configure a Publisher on Node-RED, considering the intended application and end-device identification. The general setup for a downlink in the dashboard was made following the definitions on TTN github page. Sending downlinks through the API becomes particularly important when many downlinks have to be sent in a row, which is better explained in the next subsection. Lastly, Figure 3.6 shows an example of a command generated to setup a fragmented data block transport session. This command would be sent to the port 201 of the end-device.

Figure 3.6: Command Generation Example

Downlink Automatization

As defined by the LoRa Alliance protocol, the encrypted firmware images to be updated are to be sent through fragments. That is because of the limitation on the maximum size of a LoRa packet. Therefore, to perform FUOTA, hundreds of

downlinks in sequence are necessary. In TTN console, the user has to input downlinks manually and there is no way to automatize or schedule multiple downlinks. Thus, the most important task for the Application Server is to perform multiple downlinks through the API of TTN from a single input from the user, as long as the user uploads the fragmented image file into the dashboard and succesfully sets up the fragmentation session with the device. This automatization had already been implemented by [11], and was changed to work with the API of TTN by encoding the downlinks in Base64 and changing the configuration of the Publisher. Furthermore, it was necessary to adapt the downlinks to work with as class C multicast downlinks with TTN, so further modifications were made. In this case, the ID of the gateway can be inputted by the user on the dashboard, and scheduled time for each downlink can be obtained through a chronometer block on Node-Red that sets the time as a global variable. All the fragments sent for FUOTA should be sent to the port 201.

3.2.6 Updating a ML model with FUOTA

The work to have an firmware able to be updated through FUOTA and at the same time support an ML model was based around the example in the package STM32CubeWL. In this folder structure, the projects unrelated to the FUOTA on the B-WL5M-SUBG1 were excluded.

The FUOTA project can be interpreted as 4 differents projects: SBSFU, KMS-Blob, SECore and the LoRaWAN Endnode. On addition to this, a 5th "separate" project was added to the structure, that makes use of X-CUBE-AI middleware and is used to generate the files for the ML model to be updated. To reproduce this, the user would create a New Project for the board B-WL5M-SUBG1 in STM32CubeIDE. Then, in the graphical interface, go to Software Packs > Select Components (Alt + O). Then, they should find the STMicroelectronics XCUBE-AI middleware and enable the Core in the latest version, as shown in Figure 3.7.

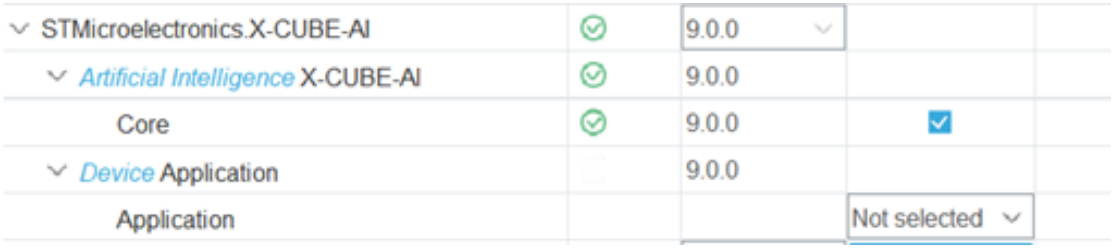

Figure 3.7: Enabling X-CUBE-AI Middleware

In the Middleware and Software Packs, the user ought to select X-CUBE-AI and load the desired model onto the Project, as shown in Figure 3.8.

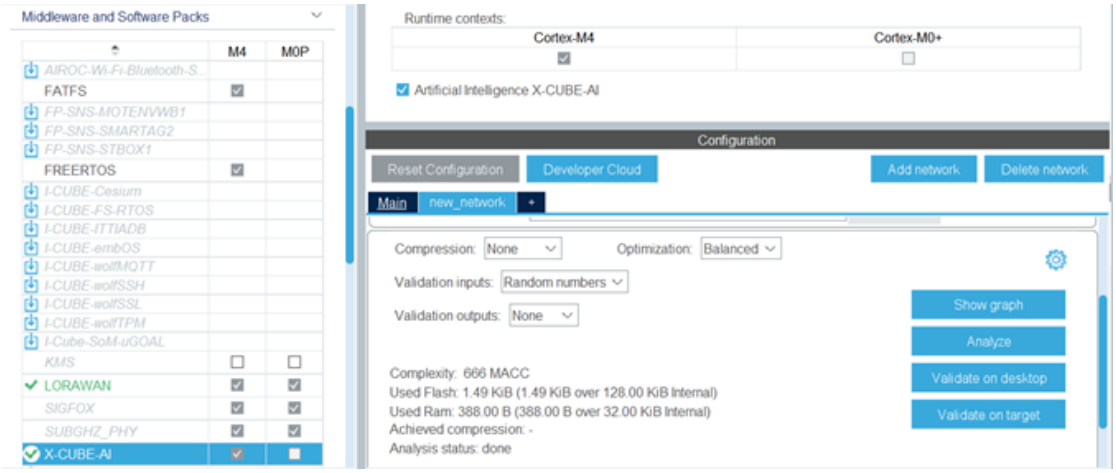

Figure 3.8: Uploading ML model to the IDE

Afterwards, the user should save the .ioc file and generate code. Then, they should proceed with the following steps:

1. Copy the `CM4/X-CUBE-AI` folder into the `LoRaWAN_End_Node_DualCore/CM4` folder of the FUOTA project.
2. Copy the `Middlewares/ST/AI` folder into the `Middlewares/ST` folder of the STM32CubeWL package.
3. Open the End Node project (`LoRaWAN_End_Node_DualCore/CM4`) on STM32CubeIDE and create virtual links to those folders.

   (a) Left click `> New > Folder > Advanced > Link to alternate location (Linked folder)`.

4. Change the `main.c` file of the core CM4 to include AI files and initialize the model.

The user then should change "include paths" and libraries on the Properties of the CM4 Project:

1. Copy the `CM4/X-CUBE-AI` folder into the `LoRaWAN_End_Node_DualCore/CM4` folder of the FUOTA project.
2. Copy the `Middlewares/ST/AI` folder into the `Middlewares/ST` folder of the STM32CubeWL package.
3. Open the End Node project (`LoRaWAN_End_Node_DualCore/CM4`) on STM32CubeIDE and create virtual links to those folders.

   (a) Left click `> New > Folder > Advanced > Link to alternate location (Linked folder)`.

4. Change the `main.c` file of the core CM4 to include AI files and initialize the model.
5. Left click on the CM4 Project and select `Properties`. Go to `C/C++ Build > Settings > MCU GCC Compiler > Include paths` and add the following paths:

   - `../../../CM4/X-CUBE-AI/App`
   - `../../../CM4/X-CUBE-AI`
   - `../../../../../../../../Middlewares/ST/AI/Inc`

6. Still on the Settings tab, go to `MCU GCC Linker > Libraries`, and add the library:

   - `NetworkRuntime900_CM4_GCC.a`

   And the library search path:

   - `../../../../../../../../Middlewares/ST/AI/Lib`

These settings are based on what was generated by the X -CUBE-AI project. With this, the necessary libraries for AI should be built into the project, as shown in Figures 3.9 and 3.10.

It should be noted that even though the board has two cores, M4 and M0+, all the software needed for running the ML model is built for the core M4. Therefore, updating only the firmware for M4 is sufficient for updating the ML model on the end-device.

Throughout the sections in this chapter, the steps taken to realize a FUOTA campaign in this project have been explained. In the following chapter, there is a presentation of the tests made aiming to validate this implementation and evaluate this solution within the context of smart farming.

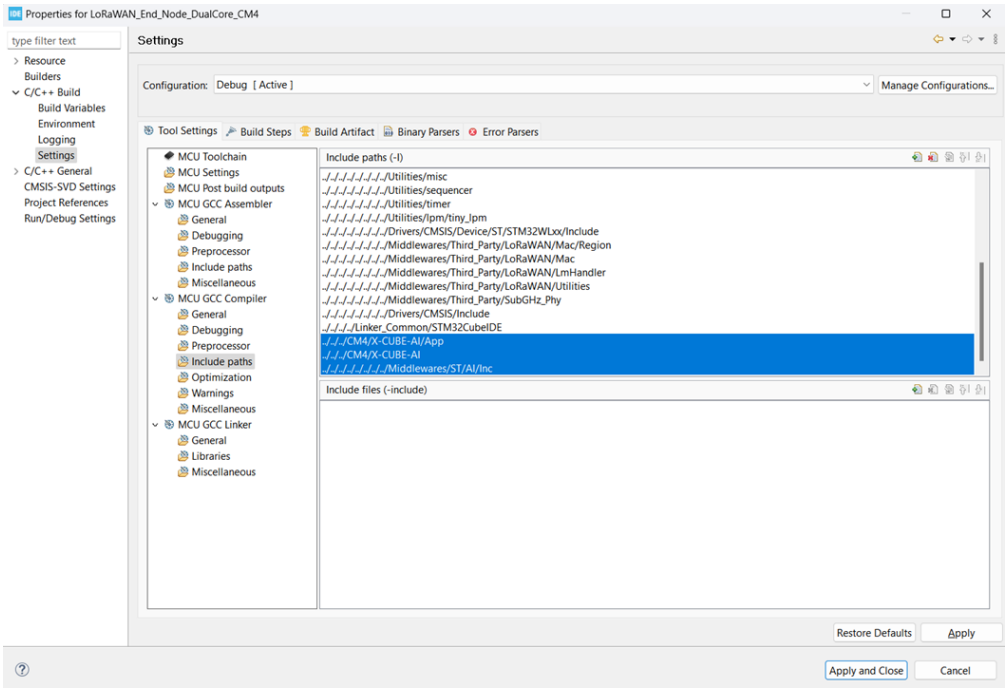

Figure 3.9: Configuring the include paths for the End-Node

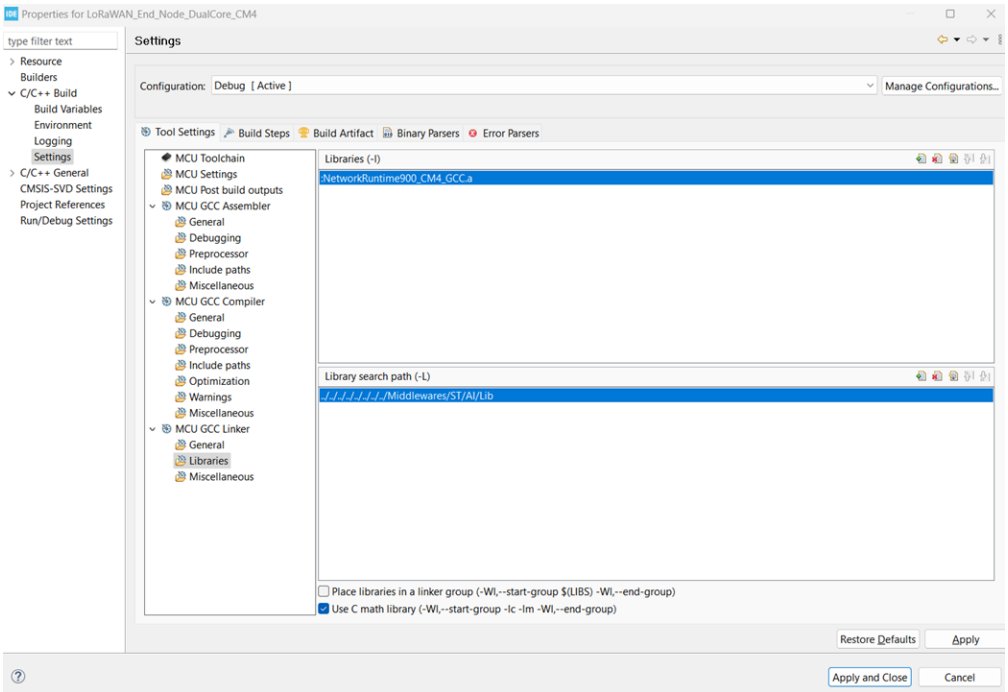

Figure 3.10: Configuring the library path for the End-Node

Chapter 4

Experiments

There are three experiments reported in this work. One was made with the FUOTA-able firmware made available by STMicroelectronics in an almost unchanged form, with fragments sent by Unicast, as to validate the expected behavior for the other two experiments, but with a firmware that can run a ML model along with the LoRaWAN software necessary for remote updates.

To keep track of time in all experiments, it was possible to use the messaging logs kept by the network server for the end-device. The end-device, while in Class A, sends periodic uplinks to a frame port defined in its firmware. By changing which frame port is used for these periodic uplinks on the firmware to be transmitted, it is possible to check that the update was completed by observing a different frame port being used for these periodical uplinks. Therefore, the first uplink to the new frame port logged on TTN would have the timestamp considered as the end of the update process. This time would account for the end-device rebooting with new firmware and reconnecting with the Network Server.

Unicast without ML

The first experiment made use of the FUOTA-able firmware made available by STMicroelectronics in an almost unchanged form. For this test, all that was needed to develop was the Application Server, which is described here, and it was performed in the first month of the thesis work as to have an initial validation of all the parts involved in the work. Since the fragments were to be sent with Unicast Downlinks, the only command necessary to setup the process was the request for a fragmentation session. Once the end-device sent an uplink signaling that the setup was succesful, the transmission can be started. Initially, the end-device would send uplinks to the frame port 2, and after the update, to the port 5. As further confirmation of a succesful update, it was possible to check the logs on TeraTerm.

Unicast with ML

In this second experiment, the possibility to remotely update the ML model running on the end-device was validated. The steps taken to complete the process are the same. However, since the firmware image to be transmitted is larger, it is expected that the process takes more time, as there are more fragments to send. Although, since it uses the board on Class A, the update time would multiply with the number of end-devices in a real applicationFurthermore, it is possible to verify the correct behavior of the board by checking the logs on TeraTerm.

Multicast with ML

This experiment followed the protocols for FUOTA defined by LoRa Alliance more rigorously, in the sense that all fragments were transmitted as multicast downlinks. This would allow for a more scalable update process in a real application with multiple devices, as the need to update each device individually is eliminated. To setup the session, it is necessary to send three unicast commands to each end-device (in this case, still one): The Multicast Session request command, that tells the end-device which multicast group to join and listen to, the Fragmentation Session Request command, to setup te data block transportation session, and the Class C Session Request command, to tell the end-device the time to switch to Class C and start receiving the multicast downlinks. The answers the board sends to each of those commands can be seen on the dashboard. When using multicast downlinks, there is a limitation on the size of the message that cannot exceed 51 bytes. Therefore, smaller fragments then the ones used before had to be generated, which means more fragments to represent the firmware image. This incurs in a increased update whe compared to the same experiment with unicast for a single end-device. Furthermore, this experiment was performed with gateways that were concomitantly used for others project. This created a restraint to its utilization, seeing as sending the downlinks with a interval too short would raise the gateway duty cycle to above 1%, which is illegal in Europe. Consequently, the update time increased due to the longer intervals between fragment downlinks.

4.0.1 Results

The results from the three experiments offer a comprehensive view of the performance characteristics and limitations of firmware update over-the-air (FUOTA) deployments using LoRaWAN in realistic settings. Each test highlights distinct aspects of the system, revealing how firmware size, transmission mode, and network configuration affect overall update efficiency.

In the first experiment, the use of unicast downlinks for a relatively small firmware image—devoid of machine learning components—confirmed the correct

operation of the FUOTA pipeline under minimal load. With 321 expected fragments and 16 lost during transmission, the process achieved a packet delivery ratio of approximately 95%, and completed in 56 minutes and 31 seconds. This result demonstrated that the system's components, including fragmentation, scheduling, and post-update reboot, functioned as intended.

The second experiment introduced a larger firmware image containing a machine learning model. As expected, the number of fragments increased to 450, with 24 lost during transmission, resulting in a similar delivery ratio of 94.67%. The update was completed in 1 hour, 19 minutes, and 22 seconds. The consistency of the reboot duration, as inferred from the delay between the last downlink and the first uplink with the new frame port, suggests that the additional firmware complexity did not affect the system's post-update behavior. These results confirm that the FUOTA procedure remains stable even when the firmware incorporates embedded ML functionality. However, since all fragments were delivered via unicast, the time required to update a larger group of devices would grow linearly with the number of nodes.

The third experiment implemented multicast transmission for a firmware image containing the ML model. This setup required a more rigorous adherence to the FUOTA protocol defined by the LoRa Alliance. Due to the use of data rate 0, the maximum payload size was constrained to 51 bytes, of which only 48 bytes were available for fragment data. As a consequence, the firmware was split into 1,049 fragments. Regulatory constraints on the gateway's duty cycle imposed additional delays, as downlink intervals had to be extended to remain compliant. The total update time reached 33150 seconds, or 9 hours 12 minutes and 30 seconds, a significant increase when compared to unicast experiments. Despite the extended duration, the update process remained reliable, with no indication of fragment loss or transmission failure.

This experiment underscores the trade-offs inherent in multicast FUOTA. Although multicast introduces longer delays under certain configurations—especially with low data rates and strict regulatory constraints—it enables scalability by allowing simultaneous delivery of update fragments to all nodes in a multicast group. In contrast to unicast, where update time scales linearly with the number of devices, multicast ensures that the total transmission duration remains constant regardless of group size. This property makes multicast essential for large-scale deployments where individual device updates would otherwise become impractically slow.

These experimental results are consistent with findings from related work and validate the system within the practical bounds of current LoRaWAN infrastructure. The longer duration observed in the multicast test was not due to transmission failures but rather to conservative payload sizing and the necessity of long inter-packet intervals. Improvements in gateway availability, optimized data rates, and

adaptive fragment sizing would help reduce update times and make multicast deployment more efficient.

In conclusion, the results demonstrate a functioning and robust FUOTA system that accommodates both basic and complex firmware images. While unicast remains suitable for smaller networks or isolated updates, multicast emerges as the only practical option for scenarios involving many devices. Future efforts may focus on optimizing transmission parameters and infrastructure usage to further reduce update times without compromising regulatory compliance or network stability.

Chapter 5

Conclusion and Future Perspective

This thesis documented the implementation of a firmware update over-the-air (FUOTA) system using multicast communication in a LoRaWAN network. This work started because of the intention of the eLIONS Group to have a practical way to update the ML models running on their smart farming devices during future deployments. Additionally, in the context of IoT for agriculture, the LoRaWAN technlogy is a viable fit that was already used by the eLIONS group, which led to this work being focused on using it for FUOTA, instead of other possible communication protocols. Furthermore, the group was habituated to the STM32 boards and development environments, which facilitated the choice of hardware for this project, followed by the choice of network server, The Things Network, where other projects of the group are hosted. Lastly, the platform used to allow for scheduling of hundreds of sequential downlinks, Node-Red, came as a natural option that had been used on similar projects and that was compatible with TTN. After working on the result of these choices, this thesis work produced a platform that can be used to realize the FUOTA procedure on other applications hosted on TTN, and a "demo" firmware for the B-SUBG1-WL5M board that can be used as a benchmark for future projects of the eLIONS Group. The system design was shaped by real-world trade-offs. Using Class C operation simplifies the delivery logic, as devices continuously listen for downlinks and do not require tight synchronization. However, this approach imposes practical constraints: devices needed to remain powered throughout the update process, and gateway timing needs to be accurate to ensure that multicast sessions were correctly scheduled. These constraints were met in the testing environment, but they highlight deployment considerations that must be taken into account in production settings.

The implementation focused on producing a working system rather than exhaustive performance benchmarking. Nonetheless, the successful update of real devices using multicast downlinks confirms the viability of the approach. The steps taken —from generating firmware packages and fragmentation sessions to provisioning the network server and managing device configurations— were documented to facilitate reuse by other developers working with similar hardware and network environments.

However, there is significant potential for extending this work. As an example, one could propose a research with a complete deployment of IoT devices on the field, in which an ML model is embedded to its B-WL5M-SUBG1 microcontroller, and evaluate the FUOTA procedure in a real smart farming scenario, outside of laboratory conditions. Further work could explore enhanced security mechanisms for firmware validation and delivery integrity. While the system as implemented does not include explicit update confirmation from end devices, this could be added in the future to improve robustness and monitoring. Additional testing under varied network conditions and with larger numbers of nodes would provide valuable insights into the system's behavior in more complex environments.

Another path to consider is to use another logic for the FUOTA process in place of what is defined by LoRa Alliance. As an example, there is the client-server logic usually implemented on IoT devices operating with WiFi, in which the server notifies the clients that an update is available. Afterwards, each client attempts to download and install the updates. With this, the server can open different sessions, and all connections are asynchronous. If a client detects a slowdown on the server, it can randomly select a time to try again. On top of that, if the server tries to send data and the network is busy, the network card randomly selects a time and then tries to send again, until it succeeds. This method reduces the strain on the server, and results in fast updates [57]. However, the speed of transmission of WiFi is greater than of LoRaWAN, the communication protocol most frequently used in smart farming applications, which could result on the client-server logic not possessing a high compatibility with precision agriculture.

Additionally, reducing the total size of the data sent to the End-Node is a path to make the update more efficient. To achieve this, there are works that point out that through a differential update, that is, only transmitting the delta (changes between firmwares) could reduce the update size by up to 80% [58]. Furthermore, partial updates that transmit only specific components, such as individual applications or parts of the network stack, uses 6 to 38 times less energy compared to a full firmware update [59]. Since the use of FUOTA in this thesis is oriented to the update of the embedded ML model, an algorithm that approaches update size reduction appears as a viable path for improvement.

Ultimately, this thesis contributes a detailed example of how FUOTA can be implemented in a LoRaWAN network, grounded in practical experimentation. The

experience gained through resolving real integration issues adds value beyond the formal specification of the protocols involved. As LoRaWAN continues to gain traction in industrial and infrastructure monitoring applications, efficient and scalable firmware update solutions will become increasingly important. The system developed here represents one step toward that goal, offering a practical foundation on which future improvements and adaptations can be built.

Appendix A

FUOTA

Within LoRaWAN, the FUOTA process is made possible to understand as being divided in parts. To define each of these parts, the LoRa Alliance published five specification documents that need to be followed by devices that aim to implement FUOTA capabilities within the LoRa standard. Therefore, each section of this appendix will summarize a protocol defined by LoRa Alliance to implement a part of the FUOTA process, in a effort to better understand the mechanisms behind this technology.

A.1 Multi-Package Access

The Multi-Package Access specification [60] plays a vital role in enabling Firmware Updates Over-The-Air (FUOTA) within LoRaWAN systems. It establishes a framework for smooth interoperability among various application layer packages. By assigning distinct Package Identifiers (PIDs) to each package, Multi-Package Access ensures devices can handle commands from different packages without conflicts. For instance, the Firmware Management package uses PID 4, while the Fragmented Data Block Transport package uses PID 3. This design supports the integration of functionalities like clock synchronization, multicast setup, and firmware updates within a single device or across a network.

Key commands such as PackageVersionReq and PackageVersionAns are central to Multi-Package Access. The PackageVersionReq command, sent by the network server, queries the version of a specific package on a device. The corresponding PackageVersionAns provides the package version details, ensuring compatibility and proper execution of updates and operations through robust version control.

Each package operates on a unique application layer port—for example, port 203 for the Firmware Management Protocol and port 201 for the Fragmented Data Block Transport. This separation ensures commands for one package do

not interfere with others. During FUOTA processes, Multi-Package Access serves as the backbone, enabling seamless management of multiple functionalities. For example, a device can handle firmware updates while simultaneously managing clock synchronization and fragmentation tasks without conflicts.

This specification underscores the importance of modularity and scalability, making it essential for efficient device and network operations in LoRaWAN systems.

A.2 Clock Synchronization

Clock synchronization is essential in the LoRaWAN ecosystem, especially for enabling efficient Firmware Updates Over-The-Air (FUOTA). The Application Layer Clock Synchronization protocol [61] ensures all devices share a unified time base, which is critical for coordinating multicast data distribution, fragmented delivery, and other scheduled tasks.

Using the AppTimeReq and AppTimeAns commands over application port 202, devices communicate their local time and drift to the server, which replies with corrected timestamps based on the GPS epoch. This ongoing exchange allows devices to stay synchronized despite clock drift caused by hardware or environmental factors.

The protocol is particularly vital for Class B and Class C devices. Class B relies on precise timing to open receive windows aligned with network beacons, while Class C multicast groups require synchronized windows to optimize data delivery. In FUOTA, synchronization ensures devices in a multicast group receive firmware simultaneously and that fragmented data is transmitted in order, reducing retries and avoiding conflicts.

While less critical for single-device updates, synchronization still offers benefits like scheduling updates during off-peak hours and supporting future network scalability. A synchronized device can be easily integrated into a larger coordinated system if needed later.

It should be noted, however, that in the firmware used in this project, the device obtains the timestamp from the gateway once it joins the network, which is why the commands for this protocol are not explicitly mentioned during the implementation description.

In short, the Application Layer Clock Synchronization protocol underpins coordinated device operations in LoRaWAN networks, providing the precise timing necessary for reliable, scalable, and efficient FUOTA processes.

A.3 Remote Multicast Setup

The Remote Multicast Setup is the required specification to implement a highly efficient mechanism for pushing data to multiple devices over a single connection in the LoRaWAN FUOTA ecosystem. This protocol [62] supports, creates, manages, or deletes multicast groups in cases that may be required for broadcasting firmware updates or any other large data payload to a fleet of devices. By doing this with multicast communication, the protocol can reduce network congestion and the time it takes to update a large number of devices.

The commands involved, such as McGroupSetupReq and McGroupSetupAns, are central to this specification, allowing the network server to configure multicast groups on end-devices. The command McGroupSetupReq carries all relevant information, including a unique Multicast Group ID (McGroupID), the multicast address (McAddr), and encrypted multicast keys (McAppSKey and McNwkSKey). The McGroupID will be used as a short form to refer to the group, reducing protocol overhead with appropriate message delivery. This enables the device, once this command is successfully processed, to send an McGroupSetupAns as an acknowledgment.

These include parameters such as the frame counters of the group, minMcFCnt and maxMcFCnt, and keys derived from the McKey, used for confidentiality and integrity of the multicast messages. For instance, an Encryption Key, McKEKey, may be used to encrypt the McKey, itself derived from a root key provisioned on the device, providing a layered mechanism of security against threats.

This protocol uses application layer port 200, separating the multicast management commands from all other functionalities, like clock synchronization or fragmented data transport. This ensures that multicast operations will never interfere with other processes, maintaining overall network efficiency and reliability.

One of the important features of the Remote Multicast Setup specification is its support for Class B and Class C multicast sessions.

Class C sessions have their devices in continuous listening mode, which is why they are ideal for real-time updates. Class B sessions make use of periodic time slots synchronized with the network beacons and are best used for devices with very restricted power resources. The commands to initiate these include McClassCSessionReq and McClassBSessionReq, while the response commands are McClassCSessionAns and McClassBSessionAns, respectively.

It is hard to imagine how effective distribution of firmware updates can be performed in FUOTA without multicast communication. Instead of sending an update to each individual device, the network server sends the firmware image to all devices in a multicast group. This approach drastically reduces the number of transmissions and hence saves bandwidth and energy on both server and device sides.

At the same time, multicast communications can cooperate very well with the other key protocols comprising the FUOTA suite: from synchronized timing—allowing multiple devices to open their RX windows all at once for simultaneous or broadcast message delivery—to assurance of delivery in cases involving fragmentation or large images of firmware for devices receiving.

In single-device scenarios, however, the role of the Remote Multicast Setup diminishes because there will be only one device to send updates to. It will not go through the trouble of handling multicast groups; instead, the server will unicast all information to the device. Safety features, like encrypted keys and validation of frame counter values, still hold. Their function is to ensure resilience against tampering even in small, single-device communications.

From a technical viewpoint, the protocol has mechanisms that prevent collisions and ensure high reliability even for big installations. In the instance where multiple devices respond, for example, to a multicast command, the response is implemented with randomized delays that eliminate the possibility of collisions in responses. This property is useful, especially in very dense networks where devices may share the same physical channel.

The Remote Setup protocol supports multicast, known as dynamic group management—a procedure allowing a network server to add or remove devices from multicast groups. Commands such as McGroupDeleteReq and McGroupDeleteAns allow removing these groups, which take resources on the device because either they are outdated or never used. This guarantees that such flexibility will keep the multicast framework updated to the dynamically evolving demands of the network while keeping it efficient and relevant.

In the end, Remote Multicast Setup is a building block of the FUOTA framework, which enables scalable and efficient data distribution across LoRaWAN networks. The fact that it manages multicast groups, has robust security features, and supports several classes of operation makes it a very versatile tool for large-scale deployments. In the case of single-device use cases, it's less important, but its basic design provides security and reliability for any kind of use case. Thus, remote setup for multicast transmissions becomes indispensable for scalability and efficiency in today's IoT ecosystem.

A.4 Fragmented Data Block Transportation

The Fragmented Data Block Transport specification [63] closes one of the most challenging gaps in the LoRaWAN FUOTA process: efficient and reliable transport of a large-sized data payload across constrained-bandwidth networks. Firmware updates, large in size regarding LoRaWAN limitations on maximum payload size, will be fragmentized into pieces that shall be transmitted in a segmented way

to end-devices. By this protocol, it has been ensured that those fragments, once received by the end-device, reassemble precisely into a complete block of information while keeping the integrity and reducing retransmissions accordingly.

The basic building blocks of this specification would therefore be commands like the FragSessionSetupReq for the initialization and handling, respectively, of fragmentation sessions. FragSessionSetupReq embeds in itself a number of critical parameters related to the session, such as the number of uncoded fragments (NbFrag), the fragment size (FragSize), and a session counter (SessionCnt). This command is transmitted by an application layer port dedicated to fragmentation operations—201—to clearly segregate all fragmentation functionalities from others, potentially coexistent, like multicast setup or clock synchronization. After internal processing of the request, the end-device responds with a FragSessionSetupAns with the acceptance of the session parameters and, if any, the problems encountered, like not enough memory.

The protocol supports both unicast and multicast transmission modes for enhanced flexibility. In multicast scenarios, fragments are broadcast to multiple devices at the same time, reducing network overhead considerably. For unicast, the protocol ensures reliable delivery to individual devices, making it well-suited for single-device FUOTA scenarios.

The Fragmented Data Block Transport protocol makes use of one of the most important innovations: Forward Error Correction (FEC), that was explained in the chapter about LoRaWAN.

This makes it possible for the receiving device to recreate the whole data block even when it suffers from the loss of pieces during transmission. The underlying reason is of crucial importance in areas such as downtown urban or sparsely populated areas, where received signals are normally weak. By minimizing the impact of retransmissions, FEC further boosts the reliability of updates at minimal bandwidth and power usages.

Each fragment sent in a session is preceded by a Message Integrity Code (MIC), which ensures the authenticity and integrity of each fragment. Once all fragments are received, the end-device will reassemble the data block and check its validity against the MIC computed in the session setup. This gives assurance that the reassembled data is corruption-free, and some confidence in the integrity of the update prior to applying it.

Mechanisms for session management are part of the fragmentation protocol. Such commands as FragSessionDeleteReq and FragSessionDeleteAns allow the network server to delete no longer needed fragmentation sessions, freeing device resources. In this respect, it's really important in keeping efficiency in either memory-restricted devices or networks that require frequent updates.

In the context of FUOTA, Fragmented Data Block Transport works in harmony with complementary protocols, such as clock synchronization and multicast setup.

Accurate timing ensures, at the device level, readiness in receiving fragments during designated windows—a process that prevents collisions, hence increasing reliability. Multicast setup allows for simultaneous delivery of fragments to multiple devices, thus leveraging the network's broadcast capabilities regarding scalability.

Because a single-device FUOTA has much less complexity of multicast coordination, the role of such features is lower in that case. However, it is worth noticing that the presence of fragmentation cannot be replaced; it is important to mention that firmware updates in many cases have a size bigger than the LoRaWAN frame payload, and hence fragment delivery management may directly be done from the server by leveraging inherent benefits of FEC along with MIC validation for delivering data over the air in a reliable and secure fashion.

The protocol considers a set of different constraints that are inherent in LoRaWAN. As an example, it takes into consideration the limitation on duty cycles by spreading fragment transmissions over time to avoid network congestion. Besides, it includes padding mechanisms that ensure the last fragment of a data block aligns with the fragment size specified. Devices are supposed to remove this padding during reassembly in order not to distort the original data.

From the scalability perspective, Fragmented Data Block Transport provides for up to four concurrent fragmentation sessions per device. This capability lets a device receive multiple data blocks parallel to each other, while network throughput and flexibility are really improved. Moreover, this protocol supports fine reporting on the session status level, for example, FragSessionStatusReq and FragSessionStatusAns commands, enabling a server to track the progress of it dynamically and act in cases of issues arising.

The conclusion is that the Fragmented Data Block Transport specification is the basis upon which the FUOTA framework provides the necessary mechanisms for the reliable and efficient delivery of large firmware updates. FEC, validation of MIC, and robust session management will be used to make sure the update is complete and secure under difficult network conditions. While less critical for single devices, the use of multicast and synchronized timing makes the very core requirement of handling large payloads an integral part—therefore, this must be the protocol present in order to update LoRaWAN FUOTA modems. This is one kind of protocol that manifests modern requirements for IoT ecosystems to be adaptive and strong in operation.

A.5 Firmware Management Protocol

The Firmware Management Protocol [64] is central to LoRaWAN FUOTA, overseeing the entire firmware update process—from querying device versions to activating new firmware. It ensures updates are secure, verifiable, and efficiently managed,

making it a cornerstone of device reliability in LoRaWAN networks.

Through a set of commands, the protocol can handle tasks like verifying firmware, managing memory, and scheduling reboots to install updates at optimal times. These functions are crucial for resource-constrained devices and allow precise control over firmware operations.

The protocol integrates tightly with FUOTA components like Fragmented Data Block Transport and Remote Multicast Setup, enabling efficient, scalable updates across multiple devices. After delivery and reassembly, the firmware's integrity is checked before activation.

Security is a key focus: the protocol enforces cryptographic checks to ensure firmware authenticity and supports secure deletion, preventing recovery of obsolete or corrupted images.

Even in single-device contexts, the protocol ensures structured, reliable updates. It operates on application layer port 203, isolating firmware management from other LoRaWAN functions for clearer, error-resistant communication.

On some implementations of the FUOTA process for LoRaWAN, it is necessary to send a message pertaining this protocol to the end-device once the fragmented data block transport has been completed, as to instruct it to update its firmware. On this project, however, the device attempts to update its firmware immediately after all the expected fragments are received, realizing a security check to guarantee its the authenticity of the new firmware image.

Bibliography

[1] Mette Wik, Prabhu Pingali, and Sumiter Broca. «Global Agricultural Performance: Past Trends and Future Prospects». In: (Jan. 2008) (cit. on p. 1).

[2] Janet Ranganathan, Richard Waite, Tim Searchinger, and Craig Hanson. «How to Sustainably Feed 10 Billion People by 2050, in 21 Charts». In: (Dec. 2018) (cit. on p. 1).

[3] Jorge Delgado, Nicholas Short, Daniel Roberts, and Bruce Vandenberg. «Big Data Analysis for Sustainable Agriculture on a Geospatial Cloud Framework». In: *Frontiers in Sustainable Food Systems* 3 (July 2019). DOI: 10.3389/fsufs.2019.00054 (cit. on p. 1).

[4] Rajendra Sishodia, Ram Ray, and Sudhir Singh. «Applications of Remote Sensing in Precision Agriculture: A Review». In: *Remote Sensing* 12 (Sept. 2020), p. 3136. DOI: 10.3390/rs12193136 (cit. on p. 1).

[5] Kirtan Jha, Aalap Doshi, Poojan Patel, and Manan Shah. «A comprehensive review on automation in agriculture using artificial intelligence». In: *Artificial Intelligence in Agriculture* 2 (June 2019). DOI: 10.1016/j.aiia.2019.05.004 (cit. on p. 1).

[6] Mohammed Jouhari, Nasir Saeed, Mohamed-Slim Alouini, and El Mehdi Amhoud. «A Survey on Scalable LoRaWAN for Massive IoT: Recent Advances, Potentials, and Challenges». In: *IEEE Communications Surveys Tutorials* PP (Jan. 2023), pp. 1–1. DOI: 10.1109/COMST.2023.3274934 (cit. on p. 1).

[7] Ercan Avşar and Md. Najmul Mowla. «Wireless Communication Protocols in Smart Agriculture: A Review on Applications, Challenges and Future Trends». In: *Ad Hoc Networks* 136 (Nov. 2022). DOI: 10.1016/j.adhoc.2022.102982 (cit. on pp. 1, 5).

[8] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. «Low Power Wide Area Networks: An Overview». In: *IEEE Communications Surveys Tutorials* PP (Jan. 2017). DOI: 10.1109/COMST.2017.2652320 (cit. on pp. 1, 5, 6).

[9] Chollet Nicolas, Naila Bouchemal, and Ramdane-Cherif Amar. «Energy efficient Firmware Over The Air Update for TinyML models in LoRaWAN agricultural networks». In: Nov. 2022, pp. 21–27. DOI: 10.1109/ITNAC55475.2022.9998338 (cit. on pp. 1, 7).

[10] Federico Cum. «A Neural network application for impedance-based plant monitoring: from a development framework towards edge computing». Master's Thesis. Politecnico di Torino, 2022. URL: https://webthesis.biblio.polito.it/24471/ (cit. on pp. 1, 7).

[11] Sylvain Montagny. *Sylvain Montagny's fuota-server*. Available here. Accessed: 2024-12-04. Savoie Mont Blanc University (cit. on pp. 2, 21, 26–29, 31).

[12] Liu Kaiyi, Kang Hengyuan, Meng Huansheng, and Zhang Fan. «Design of a New Generation of Weather Radar Intelligent Temperature and Humidity Monitoring System Based on ZigBee». In: Dec. 2019, pp. 1–3. DOI: 10.1109/ICMO49322.2019.9025847 (cit. on p. 3).

[13] Anirbit Sengupta, Biswajit Debnath, Abhijit Das, and Debashis De. «FarmFox: A Quad-Sensor based IoT box for Precision Agriculture». In: *IEEE Consumer Electronics Magazine* PP (Mar. 2021), pp. 1–1. DOI: 10.1109/MCE.2021.3064818 (cit. on p. 4).

[14] SigFox. *Use Cases: Agriculture*. Accessed: 2024-12-04. 2024. URL: Available%20%5Chref%7Bhttps://sigfox.com/use-cases/agriculture/%7D%7Bhere%7D (cit. on p. 4).

[15] Pietro Di Gennaro, Domenico Lofu, Vitanio Daniele, Pietro Tedeschi, and Pietro Boccadoro. «WaterS: A Sigfox-compliant prototype for water monitoring». In: *Internet Technology Letters* 2 (Sept. 2018), p. 6. DOI: 10.1002/itl2.74 (cit. on p. 5).

[16] Syazwan Essa, Rafidah Petra, Mohammad Uddin, Wida Suhaili, and Nur Ilmi. «IoT-Based Environmental Monitoring System for Brunei Peat Swamp Forest». In: Sept. 2020, pp. 1–5. DOI: 10.1109/ICOSICA49951.2020.9243279 (cit. on p. 5).

[17] Jared Makario, Kimutai I, and Ciira Maina. «Long Range Low Power Sensor Networks for Agricultural Monitoring - A Case Study in Kenya». In: May 2019, pp. 1–8. DOI: 10.23919/ISTAFRICA.2019.8764882 (cit. on p. 5).

[18] Neeraj Kaushik and Teena Bagga. «Smart Cities Using IoT». In: Sept. 2021, pp. 1–6. DOI: 10.1109/ICRITO51393.2021.9596386 (cit. on p. 5).

[19] Ingenu. *Ingenu's Precision Agriculture Portfolio*. Available here. Accessed: 2024-12-04 (cit. on p. 6).

[20] Xanthoula Eirini Pantazi, Dimitrios Moshou, Thomas Alexandridis, Rebecca Whetton, and Abdul Mouazen. «Wheat yield prediction using machine learning and advanced sensing techniques». In: *Computers and Electronics in Agriculture* 121 (Feb. 2016), pp. 57–65. DOI: 10.1016/j.compag.2015.11.018 (cit. on p. 6).

[21] Xanthoula Eirini Pantazi, Afroditi Alexandra Tamouridou, Thomas Alexandridis, Anastasia Lagopodi, G. Kontouris, and Dimitrios Moshou. «Detection of Silybum marianum infection with Microbotryum silybum using VNIR field spectroscopy». In: *Computers and Electronics in Agriculture* 137 (May 2017), pp. 130–137. DOI: 10.1016/j.compag.2017.03.017 (cit. on p. 6).

[22] Xanthoula Eirini Pantazi, Afroditi Alexandra Tamouridou, Thomas Alexandridis, Anastasia Lagopodi, Javid Kashefi, and Dimitrios Moshou. «Evaluation of hierarchical self-organising maps for weed mapping using UAS multispectral imagery». In: *Computers and Electronics in Agriculture* 139 (June 2017), pp. 224–230. DOI: 10.1016/j.compag.2017.05.026 (cit. on p. 6).

[23] Michel Craninx, Veerle Fievez, Bruno Vlaeminck, and Bernard De Baets. «Artificial neural network models of the rumen fermentation pattern in dairy cattle». In: *Computers and Electronics in Agriculture - COMPUT ELECTRON AGRIC* 60 (Mar. 2008), pp. 226–238. DOI: 10.1016/j.compag.2007.08.005 (cit. on p. 6).

[24] Ningbo Cui. «Modeling reference evapotranspiration using extreme learning machine and generalized regression neural network only with temperature data». In: *Computers and Electr onics in Agricu lture* 136 (Mar. 2017), pp. 71–78. DOI: 10.1016/j.compag.2017.01.027 (cit. on p. 6).

[25] Evan Coopersmith, Barbara Minsker, Craig Wenzel, and Brian Gilmore. «Machine learning assessments of soil drying for agricultural planning». In: *Computers and Electronics in Agriculture* 104 (June 2014), pp. 93–104. DOI: 10.1016/j.compag.2014.04.004 (cit. on p. 6).

[26] Azeem Mirani, Engr Dr Muhammad Suleman Memon, Rozina Chohan, Asif Wagan, and Mumtaz Qabulio. «Machine Learning In Agriculture: A Review». In: 10 (May 2021), pp. 229–234 (cit. on p. 6).

[27] Xanthoula Eirini Pantazi, Dimitrios Moshou, Roberto Oberti, Jon West, Abdul Mouazen, and Dionysios Bochtis. «Detection of biotic and abiotic stresses in crops by using hierarchical self organizing classifiers». In: *Precision Agriculture* 18 (June 2017), pp. 1–11. DOI: 10.1007/s11119-017-9507-8 (cit. on p. 7).

[28] Umberto Garlando, Lee Bar-On, Paolo Motto Ros, Alessandro Sanginario, Stefano Calvo, Maurizio Martina, Adi Avni, Yosi Shacham-Diamand, and Danilo Demarchi. «Analysis of In Vivo Plant Stem Impedance Variations in Relation with External Conditions Daily Cycle». In: May 2021, pp. 1–5. DOI: 10.1109/ISCAS51556.2021.9401242 (cit. on p. 7).

[29] Umberto Garlando, Lee Bar-On, Paolo Motto Ros, Alessandro Sanginario, Sebastian Peradotto, Yosi Shacham-Diamand, Adi Avni, Maurizio Martina, and Danilo Demarchi. «Towards Optimal Green Plant Irrigation: Watering and Body Electrical Impedance». In: Oct. 2020, pp. 1–5. DOI: 10.1109/ISCAS45731.2020.9181290 (cit. on p. 7).

[30] Lee Bar-On, Sebastian Peradotto, Alessandro Sanginario, Paolo Motto Ros, Yosi Shacham-Diamand, and Danilo Demarchi. «In-Vivo Monitoring for Electrical Expression of Plant Living Parameters by an Impedance Lab System». In: Nov. 2019, pp. 178–180. DOI: 10.1109/ICECS46596.2019.8964804 (cit. on p. 7).

[31] Huicai Liu. «Understanding plant health status from electrical impedance using Neural Networks». Master's Thesis. Politecnico di Torino, 2021. URL: https://webthesis.biblio.polito.it/21029/ (cit. on p. 7).

[32] Alessandro Lovesio. «Design of a Neural Network development framework for plant monitoring applications». Master's Thesis. Politecnico di Torino, 2021. URL: https://webthesis.biblio.polito.it/21030/ (cit. on p. 7).

[33] Mehmet Ali Ertürk, M.Ali Aydin, Talha Büyükakkaşlar, and Hayrettin Evirgen. «A Survey on LoRaWAN Architecture, Protocol and Technologies». In: *Future Internet* 11 (Oct. 2019), p. 216. DOI: 10.3390/fi11100216 (cit. on pp. 7, 13).

[34] Alireza Maleki, Ha Nguyen, Ebrahim Bedeer Mohamed, and Robert Barton. «A Tutorial on Chirp Spread Spectrum Modulation for LoRaWAN: Basics and Key Advances». In: *IEEE Open Journal of the Communications Society* PP (Jan. 2024), pp. 1–1. DOI: 10.1109/OJCOMS.2024.3433502 (cit. on pp. 9, 10, 12, 13).

[35] *AN1200.22 - LoRa Modulation Basics*. Available here. U.S. - Camarillo, California: Semtech Corporation (cit. on p. 9).

[36] LoRa Alliance. *LoRaWAN® Regional Parameters, RP002*. RP002-1.0.4. LoRa Alliance. 2022. URL: https://resources.lora-alliance.org/document/rp002-1-0-4-regional-parameters (cit. on pp. 10, 14).

[37] Rachel Kufakunesu, Gerhard Hancke, and Adnan Abu-Mahfouz. «A Survey on Adaptive Data Rate Optimization in LoRaWAN: Recent Solutions and Major Challenges». In: *Sensors (Basel, Switzerland)* 20 (Sept. 2020). DOI: 10.3390/s20185044 (cit. on pp. 10–12).

[38] Peng Zhang. *Industrial Control Technology: A Handbook for Engineers and Researchers*. William Andrew Publishing, Available here, 2008, pp. 755–756 (cit. on pp. 10, 11).

[39] Olivier Seller and Nicolas Sornin. «Low power long range transmitter». EP 2 763 321 A1. https://patents.google.com/patent/EP2763321A1/en. Aug. 2014 (cit. on p. 11).

[40] *The Things Network on FEC and Code Rate for LoRaWAN*. Available here. Accessed: 2024-12-04. The Things Network (cit. on p. 11).

[41] LoRa Alliance. *TS001-1.0.4 LoRaWAN® L2 1.0.4 Specification*. Version 1.0.4. LoRa Alliance. 2020. URL: https://lora-alliance.org/resource_hub/lorawan-l2-1-0-4-specification/ (cit. on pp. 12, 13).

[42] Semtech Corporation. *LoRaWAN – Simple Rate Adaptation Recommended Algorithm*. Class A/B Specification, Revision 1.0, Preliminary. Semtech Corporation. Oct. 2016. URL: https://www.semtech.com (cit. on p. 14).

[43] *The Things Stack on ADR*. Available here. Accessed: 2024-12-04. The Things Industries (cit. on p. 14).

[44] The Things Network. *ADR Logic - Line 218 in adr.go*. Accessed: 2024-11-21. 2024. URL: https://github.com/TheThingsNetwork/lorawan-stack/blob/5a816e8171f993db9659566286d45725698f032e/pkg/networkserver/mac/adr.go#L218 (cit. on p. 14).

[45] The Things Network. *ADR Logic - Line 232 in adr.go*. Accessed: 2024-11-21. 2024. URL: https://github.com/TheThingsNetwork/lorawan-stack/blob/5a816e8171f993db9659566286d45725698f032e/pkg/networkserver/mac/adr.go#L232 (cit. on p. 14).

[46] The Things Network. *ADR Logic - Lines 251-262 in adr.go*. Accessed: 2024-11-21. 2024. URL: https://github.com/TheThingsNetwork/lorawan-stack/blob/5a816e8171f993db9659566286d45725698f032e/pkg/networkserver/mac/adr.go#L251-L262 (cit. on p. 14).

[47] The Things Network. *ADR Logic - Lines 288-296 in adr.go*. Accessed: 2024-11-21. 2024. URL: https://github.com/TheThingsNetwork/lorawan-stack/blob/5a816e8171f993db9659566286d45725698f032e/pkg/networkserver/mac/adr.go#L288-L296 (cit. on p. 14).

[48] Khaled Hassan, Tom Farrell, David McDonald, and Dirk Pesch. «How to Make Firmware Updates over LoRaWAN Possible». In: Aug. 2020, pp. 16–25. DOI: 10.1109/WoWMoM49955.2020.00018 (cit. on pp. 14, 15, 17).

[49] Khaled Abdelfadeel. *FUOTASim package*. Available here. Accessed: 2024-12-04. University College Cork (cit. on p. 14).

[50] Daniel Khieng, Yu-Zhe Xie, Jia-Cheng Zhang, and Nen-Fu Huang. «A Long Distance Low Bandwidth Firmware Update process for LPWAN - Taking LoRaP2P+ as example». In: Jan. 2023, pp. 646–651. DOI: 10.1109/ICOIN56518.2023.10048943 (cit. on pp. 15, 17).

[51] Christia Charilaou, Spyros Lavdas, Ala Khalifeh, Vasos Vassiliou, and Zinon Zinonos. «Firmware Update Using Multiple Gateways in LoRaWAN Networks». In: *Sensors* 21 (Sept. 2021), p. 6488. DOI: 10.3390/s21196488 (cit. on pp. 15, 17).

[52] Anthi Anastasiou, Panayiotis Christodoulou, Klitos Christodoulou, Vasos Vassiliou, and Zinon Zinonos. «IoT Device Firmware Update over LoRa: The Blockchain Solution». In: May 2020, pp. 404–411. DOI: 10.1109/DCOSS49796.2020.00070 (cit. on pp. 15, 17).

[53] Njabulo Mthethwa, Nombuso Sibeko, Paul Tarwireyi, and Adnan Abu-Mahfouz. «OTA Firmware Updates for LoRaWAN Using Blockchain». In: Nov. 2020, pp. 1–8. DOI: 10.1109/IMITEC50163.2020.9334108 (cit. on pp. 16, 17).

[54] Victor Malumbres, Jose Saldana, Gonzalo Berné, and Julio Modrego. «Firmware Updates over the Air via LoRa: Unicast and Broadcast Combination for Boosting Update Speed». In: *Sensors* 24 (Mar. 2024), p. 2104. DOI: 10.3390/s24072104 (cit. on pp. 16, 17).

[55] *The Things Stack Class C Settings*. Available here. Accessed: 2025-04-02. The Things Industries (cit. on p. 26).

[56] *AN5554 - LoRaWAN® firmware update over the air with STM32CubeWL*. Available here. Geneva, Switzerland: STMicroelectronics (cit. on p. 27).

[57] Michal Kubaščík, Ing. Andrej Tupý, Ján Šumský, and Tomáš Bača. «OTA firmware updates on ESP32 based microcontrolers». In: *2024 IEEE 17th International Scientific Conference on Informatics (Informatics)*. 2024, pp. 185–189. DOI: 10.1109/Informatics62280.2024.10900824 (cit. on p. 40).

[58] Ondrej Kachman, Marcel Balaz, and Peter Malik. «Universal framework for remote firmware updates of low-power devices». In: *Computer Communications* 139 (2019), pp. 91–102. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2019.03.014. URL: https://www.sciencedirect.com/science/article/pii/S0140366418307722 (cit. on p. 40).

[59] Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, Jeroen Hoebeke, and Eli De Poorter. «Modelling the energy consumption for over-the-air software updates in LPWAN networks: SigFox, LoRa and IEEE 802.15.4g». In: *Internet of Things* 3-4 (2018), pp. 104–119. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2018.09.010. URL: https://www.sciencedirect.com/science/article/pii/S2542660518300362 (cit. on p. 40).

[60] LoRa Alliance. *TS007-1.0.0 Multi-Package Access Specification*. Tech. rep. LoRa Alliance, 2021. URL: https://resources.lora-alliance.org/technical-specifications/ts007-1-0-0-multi-package-access (cit. on p. 43).

[61] LoRa Alliance. *TS003-2.0.0 Application Layer Clock Synchronization Specification*. Tech. rep. LoRa Alliance, 2022. URL: https://resources.lora-alliance.org/technical-specifications/ts003-2-0-0-application-layer-clock-synchronization (cit. on p. 44).

[62] LoRa Alliance. *TS005-2.0.0 Remote Multicast Setup Specification*. Tech. rep. LoRa Alliance, 2022. URL: https://resources.lora-alliance.org/technical-specifications/ts005-2-0-0-remote-multicast-setup (cit. on p. 45).

[63] LoRa Alliance. *TS004-2.0.0 Fragmented Data Block Transport Specification*. Tech. rep. LoRa Alliance, 2022. URL: https://resources.lora-alliance.org/technical-specifications/ts004-2-0-0-fragmented-data-block-transport (cit. on p. 46).

[64] LoRa Alliance. *TS006-1.0.0 Firmware Management Protocol Specification*. Tech. rep. LoRa Alliance, 2021. URL: https://resources.lora-alliance.org/technical-specifications/ts006-1-0-0-firmware-management-protocol (cit. on p. 48).